UNIVERSITATEA "POLITEHNICA" DIN TIMIŞOARA
FACULTATEA DE ELECTRONICĂ ŞI TELECOMUNICAȚII
DEPARTAMENTUL DE ELECTRONICĂ APLICATĂ

Teză de doctorat

SISTEM SENZORIAL PENTRU RECUNOAŞTEREA GESTURILOR UNEI MÂINI UTILIZÂND REȚELE NEURONALE IMPLEMENTATE ÎN FPGA

Doctorand: Ing. Ştefan Oniga

Conducător ştiințific: Prof.Dr.Ing. Virgil Tiponuț

2005

CUPRINS

Capitolul I

Introducere

Roboţii umanoizi reprezintă o temă de interes pentru multe grupuri de cercetare din întreaga lume. S-au făcut eforturi considerabile pentru dezvoltarea roboţilor umanoizi şi au fost obţinute rezultate impresionante din punct de vedere tehnologic, în special în problema roboţilor bipezi păşitori. În particular dezvoltarea umanoizilor necesită implementarea capacităţii de manipulare care este încă o problemă complexă în robotică.

În ultimele două decenii s-a acordat o atenţie sporită construirii mâinilor de roboţi. Manipularea cu dexteritate este una din temele de interes major în asamblarea industrială, în aspectele de proiectare a protezelor şi de asemenea în studiul mişcării umane. Ca urmare a acestor eforturi în SUA şi Japonia au fost construite mâini cu dexteritate ridicată. Shimoga [121] prezintă câteva aspecte legate de controlul acestor mâini. În timp ce s-au obţinut progrese semnificative în proiectarea, construcţia şi controlul mâinilor de roboţi, unele aspecte importante rămân de explorat, incluzând aici aspectele legate de sistemul senzorial.

În proiectarea şi construcţia sistemelor senzoriale pentru o mână artificială se pun multe probleme plecând de la alegerea tipului de senzori folosiţi, a numărului acestora, tehnologia de realizare, prelucrarea şi interpretarea semnalelor obţinute de la senzori. La acest din urmă aspect s-au concentrat cercetările din cadrul prezentei teze.

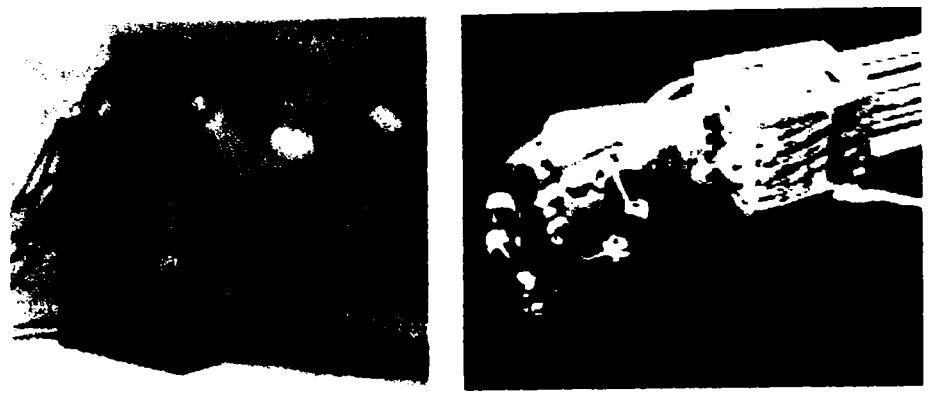
1.1 Mâna artificială

Studiile efectuate de cercetători din întreaga lume subliniază complexitatea extraordinară a sistemului senzorial al mâinii umane. O mână artificială care poate fi o mână de robot, proteză de mână sau o mână umană căreia i se protezează anumite funcţii pierdute, poate (trebuie) să fie capabilă să reproducă doar parţial această complexitate. Prezenta lucrare doreşte să aducă contribuţii în domeniul recunoaşterii gesturilor unei astfel de mâini. Soluţia propusă a identificat o serie de probleme care se pot rezolva precum şi unele care necesită cercetări ulterioare.

Datorită conţinutului lui ştiinţific şi utilităţii în majoritatea aplicaţiilor roboţilor, problema manipulării a fost investigată în amănunt şi sunt disponibile multe rezultate, atât în partea de sistem senzorial al mâinii cât şi în schemele de comandă. Popularitatea acestei teme este demonstrată de numărul mare de universităţi şi centre de cercetare care au dezvoltat mâini de robot cărora le-au dat propriul nume.

1.1.1 Mâna de robot umanoid

În trecut au fost dezvoltate mâini cu scopul de a realiza cercetări în domeniul prinderii şi a manipulării. Se vor prezenta în continuare cele mai evoluate patru mâini de robot realizate de universităţile Stanford-JPL, Utah/MIT, Belgrade/USC şi respectiv de către DLR (Deutches Zentrum fur Luft-und Raumfahrt).



(a) Mâna Standford/JPL (b) Mâna Utah/MIT

(c) Mâna Belgrade/USC (d) Mâna Hirzinger

Figura 1.1 Mâini de roboți umanoizi

1.1.1.1 Mâna Stanford / JPL

J. Salisbury de la Salisbury Robotics a proiectat mâna Standford/JPL [126]. Aceasta este o mână cu trei degete fiecare din ele are trei grade de libertate şi 4 cabluri de comandă; mâna este comandată de un set de actuatori cu 12 servo-motoare de curent continuu cu reductoare de turaţie 25:1 (figura 1.1.a). În principal are următoarele tipuri de senzori încorporaţi:

1.1.1.1.1 Senzori de tensiune din tendoane

Tensiunea din tendoane este detectată la baza fiecărui deget prin detecţia solicitării unui element încastrat peste care tendonul trece (printr-un scripete). În partea de sus şi de jos a consolei sunt timbre tensometrice conectate ca o jumătate de punte. Semnalul de ieşire zgomotos este îmbunătăţit cu un filtru activ trece jos de ordinul 2. Efectul neliniar al frecării în elementul de comandă este redus pentru că tensiunea este detectată la baza fiecărui deget.

1.1.1.1.2 Senzorii tactili

Un senzor forţă - moment cu 6 axe plasat în vârful degetului este folosit pentru a urmării forţele de contact şi orientările lor. În principal are 8 timbre tensometrice în jumătăţi de punte conectate în cruce Malteză, care conectează învelişul exterior la bază.

1.1.1.2 Mâna Utah/MIT

Această mână cu 4 degete a fost construită în colaborare de către Center of Engineering Design a Universităţii Utah şi Laboratorul de inteligenţă artificială de la Massachusetts Institute of Technology (figura 1.1.b) [180]. Mâna Utah/MIT are patru grade de libertate pentru fiecare deget şi un deget mare tot cu 4 grade de libertate. Aspectul mâinii

este aproximativ antropomorf. Mâna cu 16 grade de libertate este acţionată folosind 32 tendoane şi actuatoare pneumatice. Actuatoarele pneumatice sunt rapide, au frecări mici şi pot genera forţe mari. Comanda de la nivelul cel mai de jos al mâinii Utah/MIT include un controler analogic pentru fiecare din cele 16 segmente. Comanda la nivel superior este mapată într-o arhitectură VME constând dintr-un procesor din familia 68000.

Senzorii folosiţi în această mâna sunt prezentaţi în paragraful următor.

1.1.1.2.1 Senzorii de unghi a articulaţiei

Cercetătorii de la Utah/MIT au luat în considerare două alternative cu privire la locul de amplasare a senzorilor care să furnizeze informaţii precise cu privire la unghiul articulaţiei, informaţie necesară în scopul controlului mâinii. Cele două alternative au fost:

1. Senzori plasaţi în articulaţiile din degete
2. Senzori plasaţi în actuator pentru a măsura deflecţia tendoanelor care acţionează mecanismul degetelor. Aceasta necesită determinarea unghiului articulaţiei prin calcul.

A doua abordare nu a fost favorabilă din mai multe motive, cel mai important dintre acestea fiind faptul că este supusă erorilor excesive.

Printre posibilităţile investigate în timpul deciziei asupra metodei de determinare a unghiului au fost:

- Potenţiometrică: Aceasta metodă este simplă dar introduce probleme legate de împachetare, fiabilitate şi posibilitatea pătrunderii impurităţilor.
- Capacitivă şi optică: Această abordare bazată pe tehnica măsurării discrete, dovedită a fi complexă, are rezoluţie limitată, este fragilă şi poate fi puţin fiabilă în anumite condiţii.
- Magnetică: Abordarea magnetică pentru determinarea unghiului articulaţiei, folosind senzori cu efect Hall, oferă un sistem foarte sigur, proporţional şi compact. Elementele sistemului sunt încapsulate pentru a reduce posibilitatea pătrunderii murdăriei şi altor impurităţi. Nivelul de zgomot produs de sistem este redus şi semnalele sunt suficient de netede, pentru a putea fi direct diferenţiate pentru a furniza informaţii legate de viteză. Fiecare senzor furnizează o ieşire corespunzând unei devieri unghiulare de la 0 la 95 de grade. Este linear într-un domeniu de 5%. Alte avantaje ale senzorilor cu efect Hall sunt:
    1. Semnal de ieşire continuu,
    2. Bandă de frecvenţă de lucru mare,
    3. Forţa de frecare redusă,
    4. Fără contact mecanic,
    5. Durată de funcţionare ridicată,
    6. Insensibilitate al impurităţile înconjurătoare.

Sistemul prezintă totuşi un dezavantaj, şi anume lucrul în prezenţa unui câmp magnetic puternic poate produce erori. S-a propus un sistem cu senzori cu efect Hall duali, în care tranzistoarele sunt plasate într-o configuraţie în punte, pentru a desenzibiliza sistemul la câmpurile magnetice exterioare.

1.1.1.2.2 Senzori de tensiune din tendoane

Cercetătorii de la Utah/MIT au considerat că tensiunea din tendoane trebuie monitorizată pentru a putea furniza informaţii cu privire la cuplul din articulaţie precum şi pentru compensarea sistemului de execuţie. Acest lucru a putut fi realizat prin instalarea senzorului de tensiune la distantă în punctul de inserţie al tendoanelor la fiecare articulaţie şi aproape de ieşirea fiecărui actuator. Acest sistem dual poate furniza informaţii pentru compensarea caracteristicilor elastice şi de frecare a tendoanelor. Totuşi datorită problemelor legate de încapsulare şi de complexitate, 32 de senzori de tensiune a tendoanelor au fost

poziţionate în încheietura mâinii. Fiecare senzor foloseşte un timbru tensiometric semiconductor a cărei ieşire este proporţională cu tensiunea din tendoane. Ele furnizează o ieşire liniară pentru tensiunea din tendoane între 0 şi 30 de pounds (13.59 kg). Circuitele electronice atât pentru senzorii de unghi cât şi cei pentru tensiunea din tendoane sunt localizate în sistemele de control de la nivelul inferior.

1.1.1.2.3 Senzorii tactili

Ansamblul de senzori tactili folosiţi acoperă toate segmentele degetului şi palma mâinii Utah/MIT [50]. Senzorii pe baza de cauciuc folosesc detecţia capacitivă şi electrozi flotanţi în stratul superior. Aceşti senzori folosesc circuite electronice locale pentru excitare, filtrare, conversie analog - digitală, şi comunicaţie serială. Elementele tactile individuale (taxel) sunt dispuşi la distanţa între centrele lor de 2,77 mm. Această spaţiere a fost dictată de cerinţe de gabarit pentru a obţine un semnal de ieşire destul de mare. Porţiunea plană din palmă conţine 64 de senzori, primul segment din deget conţine 76 de senzori, segmentul 2 conţine 36 de senzori iar vârful degetului conţine 58 de senzori. Sensibilitatea sistemului senzorial testat a fost de aproximativ 2,5 x 10^4 Nm^{-1} în domeniul lui de operare (<5 N). Rezoluţia locală a ariei de senzori din palmă este de aproximativ 1 mm. Datorită domeniului lui dinamic pentru forţă, senzorul tactil poate fi folosit ca un dispozitiv de reacţie în controlul forţei tranzitorii. S-a dovedit experimental că forţa de contact este mai bine controlată cu acest sistem senzorial decât prin folosirea unui senzor de forţă (mai puţin local) cum ar fi un senzor de forţă - moment plasat în încheietura mâinii. Totuşi combinaţia senzor tactil plus senzor de forţă conduce la cel mai bun răspuns, în timp ce domeniul dinamic poate fi împărţit între ei. Dezvoltată iniţial pentru mâna realizată de universităţile Utah/MIT, sistemul senzorial poate fi uşor re-modelat pentru alte forme de segmente de deget.

1.1.1.2.4 Senzori externi

Mâna are cinci segmente demontabile ocupând suprafaţa disponibilă în structura degetelor. În timpul experimentelor secţiunile selecţionate ale elementelor structurale au fost îndepărtate prin prelucrare pentru a face loc senzorilor tactili. De exemplu senzorii pentru detecţia contactului direct, presiunii normale, temperaturii, etc., pot fi încorporaţi. Această abordare adaugă flexibilitate pentru experimente prin posibilitatea adăugării de diverşi senzori astfel încât diverse metodologii pot fi încercate fără a se impune o anumită geometrie. Comunicaţia cu senzorii externi este posibilă prin conductoare care sunt ataşate prin sloturi laterale în deget [180].

1.1.1.3 Mâna Belgrade/USC

Universitatea Belgrade şi Universitatea de Sud California au cooperat în realizarea unei mâini antropomorfe cu cinci degete (figura 1.1.c). În această mână deciziile pregătire pentru prindere sunt luate folosind un sistem vizual extern [59]. Mâna este echipată cu două seturi de senzori. Acestea sunt:

1. Senzori de poziţie/unghi al articulaţiei,
2. Senzori de forţă.

1.1.1.3.1 Senzorii de poziţie/unghi al articulaţiei

Senzori de unghi al articulaţiei sunt plasaţi în toate degetele inclusiv degetul mare. Deoarece această mâna este proiectată pentru "prindere reflexă", principala buclă de reacţie este prevăzută să se închidă la acele semnale specifice care determină relaţia mână - obiect: apariţia contactului şi forţa de contact. Prin urmare nu mai este nevoie de comandă pentru

servo sisteme care să includă controlul stabil, de mare precizie şi rapide. Aceste aspecte conduc la folosirea unor potenţiometre mici de plastic conductor (Bourns, Inc. [237]), câte unul pentru fiecare deget, care au o foarte mare rezoluţie (de ordinul 320 ohmi/mm) şi o durată de viaţă foarte mare (de ordinul un milion de cicluri). Ele sunt extrem de compacte (35x10x3 mm) şi pot fi montate direct în interiorul structurii degetului. Ele oferă două avantaje suplimentare:

1. Poziţia absolută a degetului este întotdeauna disponibilă,
2. Nu este nevoie de întrerupătoare pentru a marca poziţia iniţială, cum ar fi în cazul folosirii unui codificator de poziţie a arborelui.

Aceste potenţiometre sunt montate lateral pe mecanismul care conduce braţul mobil pentru fiecare pereche de degete, permiţând senzorului de poziţie să furnizeze informaţii despre mişcarea compozită a perechilor de degete în loc de mişcarea individuală a fiecărui deget. Datorită acestei adaptări a formei nu este necesară reacţia poziţiei individuale.

1.1.1.3.2 Senzorii de forţă

În acest proiect, 23 senzori de forţă de contact sunt amplasaţi pe vârful degetului şi în palmă pentru a determina contactul cu un obiect şi forţa care se exercită. Figura 1.2 prezintă localizarea acestor senzori de contact. Traductoarele folosite sunt de tipul Force Sensing Resistor (FSR) capabile să simtă forţe cuprinse în domeniul 20 g la 5 kg (produse de Interlink Electronics [78]).

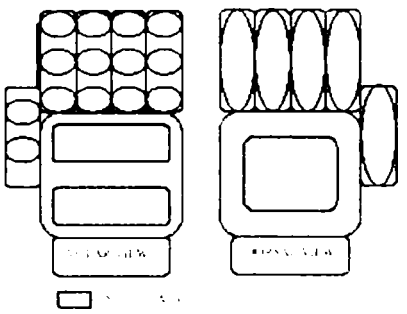


Figura 1.2 Localizarea FSR pe mâna Belgrade/USC.

Foliile subţiri flexibile de FSR sunt potrivite pentru montarea pe suprafeţe diverse şi pot fi fabricare în forme speciale. Diferite tipuri de conductoare metalice şi semiconductoare sunt adăugate pentru a reduce dependenţa de temperatură. Filmul rezultat îşi schimbă rezistenţa într-un mod relativ uniform ca o funcţie de comprimare. Repetabilitatea semnalului de ieşire este în interiorul domeniului de 1 %, menţinând histerezisul la minimum.

1.1.1.3.3 Senzorii de alunecare

Încorporarea în sistemul de control a mâinii Belgrade/USC a informaţiei legate de alunecare este în dezvoltare [59]. Strategia de prindere adoptată de cercetătorii de la Universitatea Belgrade şi Universitatea de Sud California necesită o adaptare a forţei reflexe în timpul fazei iniţiale de prindere ca şi în cursul oricărei schimbări ulterioare în echilibrul forţei. Această caracteristică reduce posibilitatea unei prinderi instabile. Senzorii de alunecare vor îmbunătăţii oportunitatea de a detecta o prindere instabilă. Două concepte au fost luate în considerare pentru detecţia alunecării:

1. Senzorii de alunecare pe baza de termistoare au avantajul unui preţ scăzut şi a dimensiunii mici. Totuşi răspunsul lor poate fi zgomotos.

2. Un mic senzor cilindru rotativ (11x4x3 mm) a fost construit de echipa de la Belgrade/USC, care este suficient de mic pentru a fi inclus în vârful degetului. Au fost folosite traductoare de rotaţie standard pentru a obţine un semnal cu zgomot redus. Rezoluţia estimată a acestui tip de senzor este de aproximativ 1 mm.

1.1.1.4 Mâna Hirzinger

Institutul pentru robotică şi dinamica sistemelor, Institutul de cercetări spaţiale din Germania DLR (Deutches Zentrum fur Luft-und Raumfahrt) a dezvoltat o mână multisenzorială cu 4 degete de mare precizie cu un total de 12 grade de libertate (figura 1.1.d), cu toate actuatoarele integrate în palma mâinii sau direct în degete [64], [65], [66], [67].

Complexitatea sistemului de comandă al mâinii de robot este bine evidenţiat de mâna Hirzinger, care are următoarele caracteristici:

- Sistem multiprocesor
- Arhitectura sistemului de comandă împărţită în două niveluri,
  - nivelul de comandă global al mâinii implementat pe un PC
  - nivelul local de comandă a degetelor
    - Un microcontroler/deget pentru managementul informaţiei
    - Un procesor de semnal în virgulă mobilă de 60 MFLOPS

De asemenea sistemul senzorial al acestei mâini este foarte complex, fiecare deget cu cele trei grade de libertate încorporează câte 28 de senzori.

Sistemul senzorial întrebuinţat de la aceasta mână este prezentată în continuare.

1.1.1.4.1 Senzorii de poziţie unghiulară

Utilizează un senzor optic de poziţie absolută. Acest senzor se compune dintr-un dispozitiv de detecţie a poziţiei unidimensional (PSD Position Sensitive Devices) iluminat de un LED infraroşu printr-un slot de măsură gravat în formă de spirală. Fiecare deget foloseşte 4 asemenea senzori. Senzorul de poziţie optic este construit compact (diametru 17 mm şi grosime 4,8 mm) folosind un circuit imprimat optimizat echipat exclusiv cu dispozitive SMD. De asemenea este integrat regulatorul de tensiune şi circuitul de condiţionare a semnalului analogic. Rezoluţia unghiulară este de 9 biţi cu o eroare de liniaritate mai mică de 1% cu unghiul maxim măsurabil de 110^0.

1.1.1.4.2 Senzorii de moment

Senzorii de moment transformă forţele din vârful degetelor în cupluri raportate la axele articulaţiilor. Acestea sunt construite cu timbre tensometrice montate pe grinzi flexibile minuscule. Circuite electronice miniaturizate plasate în deget realizează preprocesarea semnalelor asigurând calitatea optimă a semnalelor. Aceşti senzori folosiţi în 5 puncte diferite din deget sunt capabili să măsoare cupluri mai mici de 1,8 N-m cu o rezoluţie de 9 biţi.

1.1.1.4.3 Senzori de forţă de contact

Folii tactile care determină existenţa şi mărimea forţelor externe acoperă toate segmentele degetelor. Acestea sunt bazate pe rezistoare detectoare de forţă (Force Sensing Resistor - FSR) din materiale semiconductoare şi dispuse într-un configuraţie XYZ. Fiecare deget are patru senzori câte unul pentru fiecare segment de legătură şi unul pentru vârful degetului. Fiecare senzor de pe segmentele de legătură aduce 3 trei informaţii folositoare:

- Mărimea forţei normale,
- Doi parametrii, indicând, localizarea forţei aplicate.

Senzorul plasat pe vârful degetului poate măsura numai poziţia unghiulară a forţei aplicate pe suprafaţa curbată. Aceşti senzori sunt capabili să măsoare forţe cuprinse între 0.5 N şi 10 N, cu o rezoluţie de 35 mN. Circuitele analogice ale acestui sistem au fost proiectate astfel încât numărul firelor de ieşire din senzor să fie minim. În prezent există 10 fire pentru un deget şi 30 de fire pentru întreaga mână. Tabelul 1.1 compară dispunerea tridimensională XYZ cu cel mai uzual aranjament pentru senzorii tactili: multi - taxel (taxel = element al matricii de senzori tactili).

Tabelul 1.1. Comparaţie dintre senzorii multi-taxel şi de tip pad XYZ

Parametru	Multi-taxel	Pad XYZ
Numărul elementelor	16x16	1
Spaţierea	2 mm	0,28mm
Distribuţia forţei	bună	nu
Cross-talk	rea	nu
Mărimea forţei	rea	bună
Determinarea alunecării	Calc. Δxc, Δyc	uşoară
Sensibilitate	20g/mm2	50 g/mm2
Estimarea localizării	Calc. xc, yc	directă
Numărul firelor	32	4
Nr. fire pentru 3 degete	288	30
Protecţie la suprasolicitare	slabă	bună
Montarea degetului	dificilă	uşoară
Circuitele electronice	complexe	simple
Rată de scanare	80 Hz (12,5ms)	500 Hz (2ms)
Preţ	scump	ieftin

1.1.1.4.4 Senzori de temperatură

Fiecare senzor foloseşte 5 senzori de temperatură, plasaţi în diferite locaţii, pentru a culege informaţii pentru compensarea de temperatură. Ei sunt capabili să măsoare în domeniul $0\text{-}100^0$C, cu o rezoluţie de 0.1^0C.

1.1.1.4.5 Senzori vizuali

Mâna Hirzinger utilizează o cameră minusculă, care este localizată în palmă. LED-urile cu proiecţie de lumină care sunt plasate în vârful degetelor sunt folosite pentru a simplifica procesarea imaginii.

1.1.2 Proteze de mână

În paralel cu dezvoltarea mâinilor de roboţi umanoizi, problema dezvoltării de proteze de mână a fost cercetată pe larg în domeniul tehnologiilor de reabilitare: principalul scop fiind realizarea unei mâini asemănătoare mâinii umane a cărei cerinţe principale sunt mărimea şi greutatea reduse, lipsa de zgomot în funcţionare, etc. În prezent există cinci moduri de reabilitare a funcţionării unei mâini amputate. Printre acestea se pot enumera aşa numitele *proteze cosmetice*, în general realizate prin duplicarea braţului colateral. Aceste proteze sunt uşoare şi necesită puţină întreţinere dar au funcţionalitate redusă sau zero. *Protezele acţionate cu corpul* sunt comandate de mişcările corpului, de obicei ale umărului. Protezele controlate mioelectric sunt în momentul de fată cel mai bun mijloc de reabilitare parţială a funcţionalităţii unui braţ amputat, dar până acuma există mâini ce au capacitatea de prindere doar un grad de libertate, controlate de semnale electromiografice (EMG) pe un canal sau două. Cea mai avansată mână mioelectrică disponibilă comercial este mâna OttoBock SUVA (Figura 1.3).

Figura 1.3 Mâna OttoBock SUVA

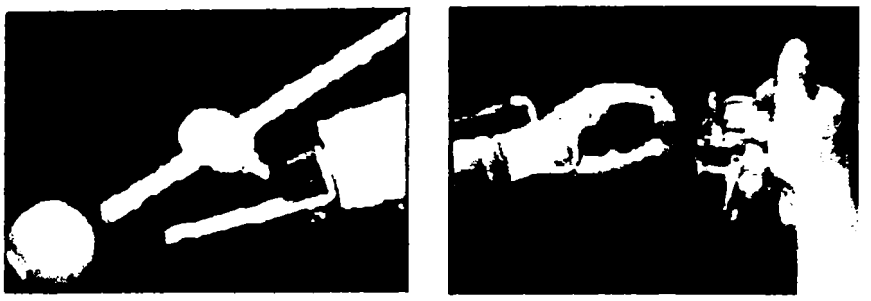


Figura 1.4 Proteze (a) pentru bowling (b) pentru pescuit.

Protezele hibride combină ultimele două tipuri. O altă abordare reprezintă protezele proiectate special pentru anumite activităţi cum ar fi pescuitul sau bowlingul (Figura 1.4).

Cu toate îmbunătăţirile realizate de noile componente şi materiale utilizate majoritatea protezelor rămân simple dispozitive de apucare cu una sau două grade de libertate. Aceasta se datorează faptului că mai mult de două grade de libertate nu pot fi uşor controlate de muşchii din porţiunea de braţ funcţională.

Doar recent mai multe grupuri au proiectat proteze de mână cu patru sau mai multe grade de libertate, prin combinarea intrărilor de le unul sau două canale EMG cu informaţiile furnizate de senzori, pentru a permite circuitelor electronice să comande mai multe segmente. Figura 1.5 prezintă mâna Losh (a) şi mâna NTU (b).

În continuare se prezintă câteva succese mai importante din domeniul cercetărilor pentru realizarea unei mâini artificiale.

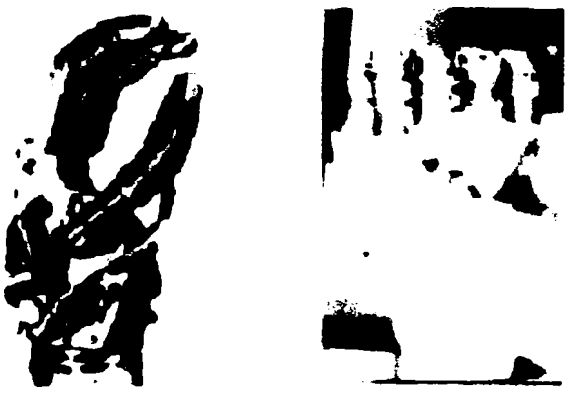


Figura 1.5 (a) mâna Losh şi (b) mâna NTU

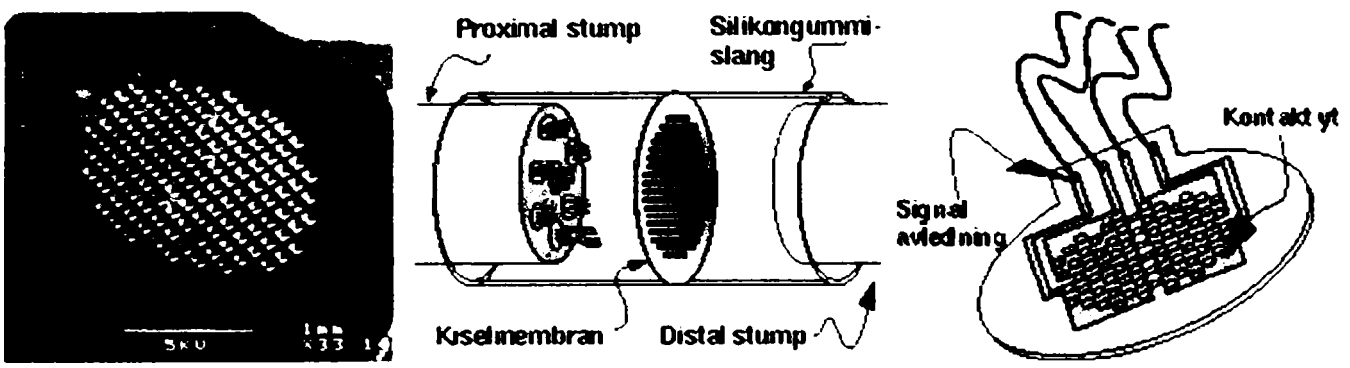
1.1.2.1 Mâna artificială dezvoltată la Universitatea LUND

Obiectivul realizat de cercetătorii de la Universitatea Lund este dezvoltarea unei noi strategii pentru comanda motorului unei proteze de mână cu ajutorul semnalelor electrice generate de electrozi din muşchi sau microchipuri implantate în sistemul nervos periferic, sau central. Utilizarea reţelelor neuronale artificiale (**RNA**) este esenţială în îndeplinirea acestui scop. Scopul este de asemenea dezvoltarea simţurilor artificiale pentru o astfel de proteză şi pentru pacienţii care şi-au pierdut funcţiile nervilor senzoriali.

S-a demonstrat că axonul sciatic de şobolan este capabil de regenerare printr-o gaură într-un electrod implantat din siliciu sub formă de sită. Mai mult s-au înregistrat semnale cu ajutorul chipului după stimularea electrică a rădăcinii nervului. De asemenea s-a demonstrat că axonul sistemului nervos central este capabil de creştere într-un chip dacă este atras de nervi periferici. Reţelele neuronale artificiale au fost folosite pentru recunoaşterea semnalelor complexe de pe muşchi culese cu electrozi multipli de suprafaţă pentru a asocia diverse semnale specifice cu mişcări specifice a unei mâini virtuale.

Membrana este implantată în calea nervului a cărui parte distantă este afectată (figura 1.6.a). După câteva luni mănuchiul de nervi a crescut prin găurile membranei de siliciu şi s-a regenerat (reconectat) cu mănunchiul inferior (figura 1.6.b). Depunând o reţea de conductoare pe membrana de siliciu semnalele culese de la nervi vor fi disponibile pentru controlul protezei (figura 1.6.c).

În ceea ce priveşte bio-compatibilitatea siliciului studiile prezintă că aceasta este comparabilă cu cea a titanului care este cunoscut ca un material bio-compatibil.



(a) Membrană de siliciu cu matrice de găuri **(b) Reconectarea nervilor prin membrana de siliciu** **(c) Membrană de siliciu cu reţea de conductoare.**

Figura 1.6 Regenerarea nervului printr-o membrană de siliciu

Nervul proximal creşte prin membrana de siliciu perforată şi reacţionează cu nervul inferior. Semnalele provenite de la nerv prin fiecare gaură prin membrana de siliciu sunt monitorizate şi procesate de o reţea neuronală pentru a le face să comande mişcările de bază ale protezei.

În două experimente s-a testat principiul sensibilităţii artificiale prin utilizarea unui senzor piezorezistiv sau detecţia stimulilor vibrotactili bazată pe senzori folosind simţul auzului. Aceste experimente indică că este posibil să se creeze simţuri artificiale într-o proteză sau o mână cu disfuncţii senzoriale.

1.1.2.2 Proiectul GRIP pentru dezvoltarea unei mâini artificiale

În trecut, soluţiile la pierderea totală sau parţială a funcţiilor mâinii a fost investigată în două moduri separate pentru două patologii diferite: prin dezvoltarea unor proteze de mână mai îndemânatice şi mai sensibile în cazul amputărilor; şi prin dezvoltarea instrumentelor pentru stimulare electrică funcţională (Functional Electrical Stimulation - FES) a resturilor de muşchi sau nervi prin intermediul unor electrozi externi sau implantaţi, pentru persoanele cu anatomia mâinii intactă dar cu pierderea parţială sau totală a funcţiei nervilor. Proiectul GRIP [248] reprezintă o abordare unificată ale celor două clase de patologii. Motivaţia unei abordări unificate a restaurării funcţionalităţii în cazul amputărilor şi a peroanelor paralizate constă în observaţia: factorul cheie în ambele patologii este interfaţa de comandă, care este o interfaţă (implantată sau externă) care permite persoanelor cu handicap să controleze proteza sau unele funcţii de bază ale mâinii paralizate într-un mod simplu şi natural, cu posibilitatea de a detecta unele informaţii senzoriale.

Obiectivul proiectului GRIP este de a dezvolta o versiune simplă a unei interfeţe "naturale", şi de a evalua eficacitatea ei într-un sistem cu stimulare electrică a funcţionării pentru a restaura funcţiile membrului superior în cazul persoanelor cu membru paralizat, folosind reacţia de la un senzor artificial.

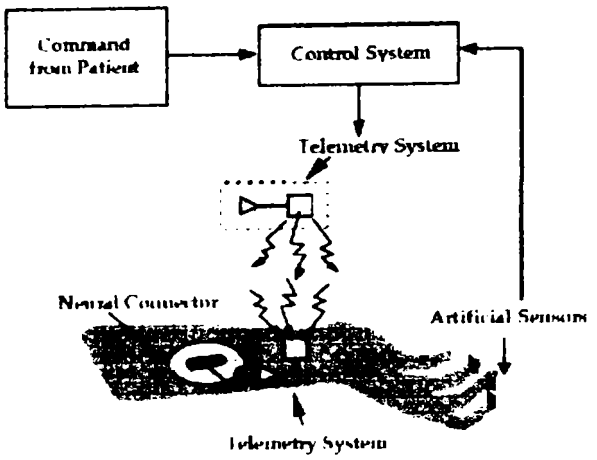


Figura 1.7 Schema sistemului stimulării electrice funcţionale

Rezultatele proiectului GRIP reprezintă o soluţie practică pentru nevoile unei părţi importante a persoanelor cu handicap. De asemenea vor reprezenta un pas intermediar important în obiectivul mai dificil de realizarea unei proteze de mână "cibernetică" şi un sistem cu stimulare electrică funcţională folosind semnalele nervoase aferente de la receptori naturali cutanaţi sau muşchi.

Controlul în buclă închisă a stimulatorului va putea fi implementat folosind semnale de la senzorii artificiali tactili şi de la senzorii de mişcare a degetului încorporate într-o mănuşă purtat de persoana cu handicap. Controlerul pentru sistemul cu stimulare electrică funcţională (FES) se va baza pe reţele neuronale artificiale şi tehnica fuzzy şi va încorpora strategii bazate pe rezultate obţinute pe experienţe reacţii cognitive. Contactul între sistemul de control şi sistemul "natural" va fi realizată printr-o legătură radio. Interfaţa om/maşină va fi furnizată sau de senzori de tip regenerativ precum cele dezvoltate în proiectul INTER sau de tipul manşetă.

1.1.2.3 Proiectul INTER pentru dezvoltarea unei mâini artificiale

Proiectului INTER (**I**ntelligent **N**eural In**TER**face) susţinut de Uniunea Europeană, la care participă mai multe universităţi şi centre de cercetare din Italia, Spania, Elveţia şi Germania, investighează aspectele fundamentale legate de proiectarea şi fabricarea unei noi generaţii de microsisteme aplicabile ca proteze neurale [141], [142], [143] , [144], [145] .

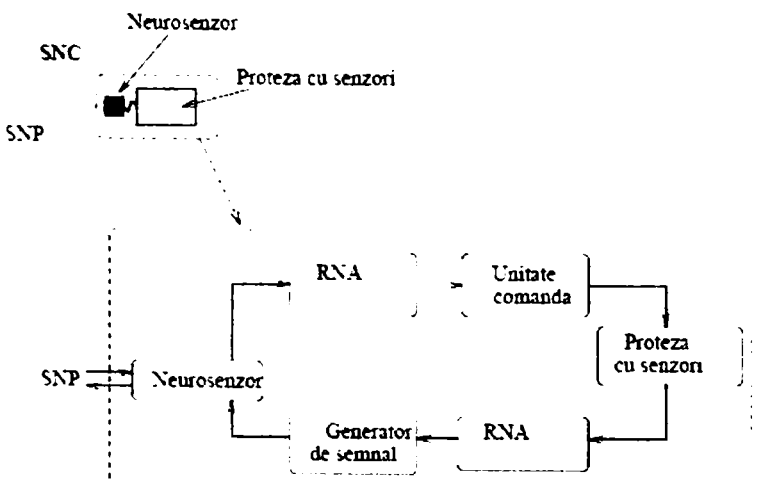


Figura 1.8 Configuraţia unei proteze controlate bio-neural

Semnalele provenite de la sistemul nervos periferic sunt culese şi amplificate de un **neurosenzor,** de tip regenerativ. Semnalul este procesat de o reţea neuronală artificială care elimină interferenţa între canale. **Unitatea de comandă** utilizează aceste semnale pentru comanda protezei. Proteza este echipată cu **senzori**. Semnalele provenite de la senzor sunt procesate de o a doua **RNA** şi transmise **generatorului de semnal.** Acesta furnizează sistemului nervos periferic un semnal prin intermediul neurosenzorului. În consecinţă proteza este complet controlată prin intermediul sistemului nervos periferic ca un braţ natural.

1.1.2.3.1 Neurosenzorii

Primele cercetări privind neurosenzorii de tip regenerativ au fost efectuate între anii 1965-1967 de Frishkoff, Goldstein şi alţii [69]. În 1969 Marks a demonstrat regenerarea nervului sciatic la broască printr-un implant poros [145]. Edell a informat despre prima înregistrare obţinută cu o arie de tip regenerativ pe substrat de siliciu. Înregistrările cu amplitudinea de ordinul a 150 μV au fost obţinute de la un astfel de dispozitiv implantat în nervul sciatic al iepurilor [45]. În 1991 Najafi a prezentat rezultatele stimulării nervilor utilizând un neurosenzor de tip regenerativ [205]. Aceşti neorosenzori permit stimularea cu impulsuri de curent până la 40μA.

Kovacs a implantat o arie de microelectrozi, fabricaţi pe un substrat de siliciu, în nervul pereonal al şobolanilor [69]. El a folosit aceste arii pentru a înregistra şi a stimula nervii la 13 luni după operaţie. Au fost înregistrate potenţiale de vârf de ordinul a 100μV. El a stimulat nervul cu stimuli de curent relativ mare dar de scurtă durată de aproximativ 500μA/10μs.

Principiul neurosenzorului utilizat în proiectul INTER este prezentat în figura 1.9. Nervii periferici al vertebratelor se regenerează dacă au fost secţionaţi. Din acest motiv, axonii secţionaţi chirurgical se vor regenera prin găurile de trecere prin chipul perforat. Semnalele nervilor pot fi culese de electrozi care înconjoară unele dintre găurile de trecere. Semnalele amplificate sunt transmise blocurilor care comandă proteza.

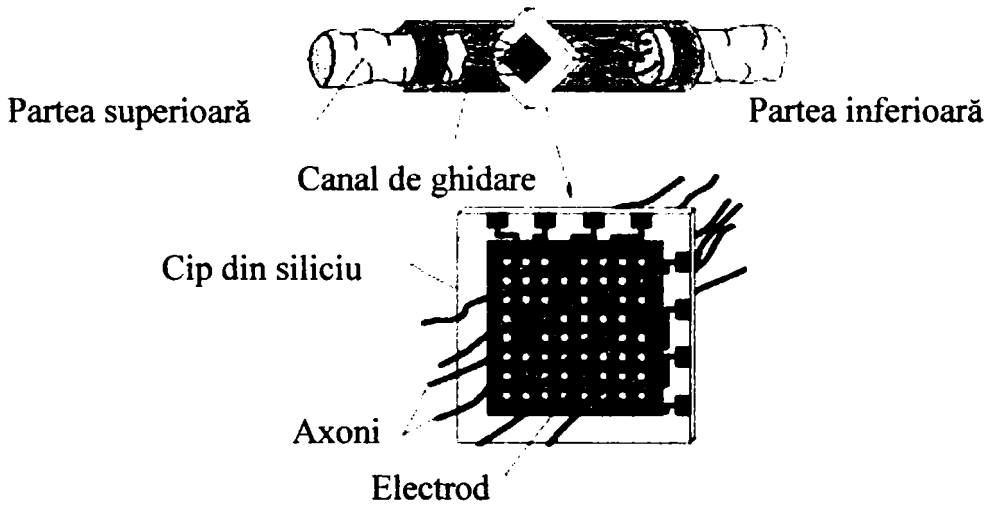


Figura 1.9 Modul de implementare a neuro-senzorului de tip regenerativ

1.1.2.3.2 Modul de operare

O vedere globală asupra soluţiei adoptate pentru procesarea semnalelor este prezentată în figura 1.10. Pentru a evita efectele datorate înregistrării, cum ar fi interferenţa, datele preprocesate (adică amplificate şi filtrate) vor fi separate în funcţie de sursa lor folosind metoda cunoscută sub numele de Independent Components Analysis (INCA), prezentată de Jutte şi Herault [28].

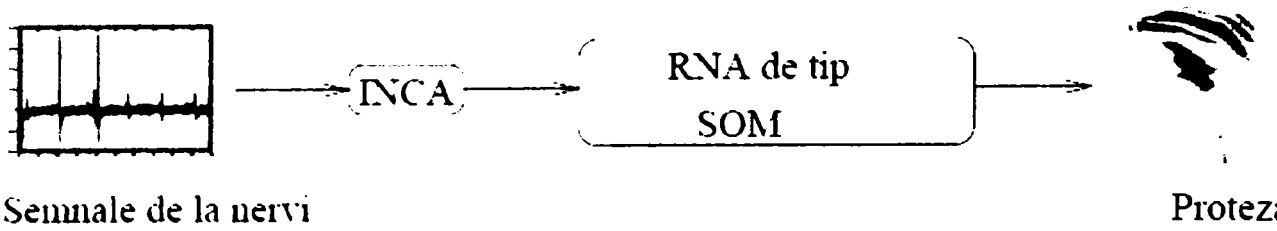


Figura 1.10 Soluția propusă pentru procesarea semnalelor

Clasificarea semnalelor nervilor trebuie realizată cu un algoritm de învățare nesupravegheat deoarece trebuie să se verifice modul de abordare adoptat cu rezultatele din testele În Vivo. Din acest motiv se utilizează RNA cu autoorganizare de tip Kohonen (SOM).

1.1.2.3.3 Setul de date

În cadrul experimentelor realizate [141] s-a folosit un set de date pus la dispoziție de IBMT, St. Ingbert. Aceste date au fost culese cu un singur electrod din nervul gastric al unui crab. Au fost identificați 21 de neuroni motori. Durata înregistrării este de 40 de secunde. Setul de date a fost înregistrat cu frecvență de eşantionare de 5 kHz.

1.1.2.3.4 Clasificarea semnalelor folosind RNA cu autoorganizare tip Kohonen

Pentru clasificarea setului de date s-a folosit o rețea neuronală bidimensională de tip SOM cu 10 neuroni pe fiecare dimensiune. Setul de instruire a constat din 2667 vectori cu câte 6 componente.

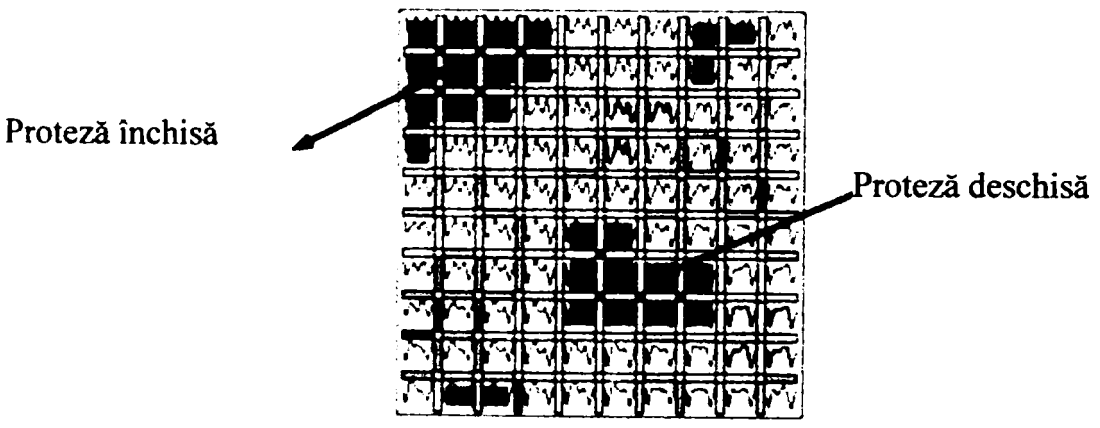


Figura 1.11 RNA instruită şi grupările obținute.

Două dintre grupări au fost alese pentru a fi alocate unei acțiuni a protezei.

Fiecare pătrat al rețelei instruite prezentat în figura 1.11 reprezintă un neuron. Dreptunghiurile gri corespund distanței euclidiene între neuroni. Curbele din interiorul pătratelor reprezintă vectorii codificați.

RNA SOM conține de asemenea grupările care s-au obținut după instruire. Fiecare grupare reprezintă semnalele de la un axon respectiv de la un grup de axoni. Aceasta înseamnă că la fiecare apariție a semnalului cules de pe nerv el va fi clasificat în grupul lui. Din acest motiv este posibil să fie recunoscute semnale de la anumiți axoni pentru a putea controla mişcarea unei protezei. Această proteză are două grade de libertate: 1) închis/deschis şi 2) viteză/putere (dacă mâna întâlneşte un obstacol el trece automat din modul viteză în modul putere). Primul grad de libertate este codat direct în RNA aşa cum este prezentat în figura 1.11. Al doilea grad de libertate este codat în frecvența apariției semnalului.

1.2 Sistem senzorial pentru o mână artificială

Mâna unui robot autonom trebuie să posede dexteritate, echilibru, stabilitate şi o bună comportare dinamică. Ea trebuie să fie controlată pentru a atinge aceste cerinţe. Sistemul senzorial optim, împreună cu sistemul de control, permit robotului să îndeplinească sarcini complexe şi dificile autorizând robotul să reacţioneze inteligent la schimbarea condiţiilor şi a mediului.

Capabilităţile protezelor de mână actuale sunt limitate la închiderea şi deschiderea mâinii. Aceasta limitează considerabil utilitatea protezei în comparaţie cu posibilităţile de mişcare ale mâinii umane sănătoase. Pentru a dezvolta proteze de mână mai avansate trebuiesc rezolvate două probleme. Prima este dezvoltarea unor soluţii mecanice avansate care să permită mai multe grade de libertate. Cea de a doua problemă, care este abordată în această lucrare, este de a găsi soluţii pentru sistemul senzorial capabil să furnizeze informaţiile necesare controlului acestor abilităţi suplimentare ale unor astfel de proteze.

Mâna umană are 22 de grade de libertate conţinute în degetul mare, patru degete şi palma mâinii. Aceasta înseamnă un set de 5 degete cu câte 4 grade de libertate montate într-o palmă cu 2 grade de libertate. Multe mâini de robot au fost dezvoltate de-a lungul anilor devenind tot mai antropomorfe în ceea ce priveşte aspectul lor şi funcţionalitatea lor. Totuşi multe provocări de ordin mecanic continuă să existe în ceea ce priveşte rigiditatea şi fiabilitatea, dar dezvoltarea a ajuns la punctul în care construcţia mecanică poate răspunde cerinţelor unui sistem de control adecvat.

Acest paragraf prezintă un sistem senzorial posibil pentru segmentele degetelor şi palma unei mâini artificiale.

O mâna de robot poate incorpora câteva, sau toate din următoarele tipuri de senzori:

1. Senzori de poziţie a articulaţiei
2. Senzori de poziţie a degetelor
3. Senzori de cuplu din articulaţie
4. Senzori de forţă de contact
5. Senzori de presiune
6. Senzori de alunecare
7. Senzori de proximitate
8. Senzori tactili
9. Senzori vizuali
10. Senzori de temperatură

Senzorii esenţiali pentru o mână artificială sunt senzorii de poziţie a articulaţiei şi a degetelor (flexare) şi senzorii de forţă de contact.

1.2.1 Senzorii de poziţie a articulaţiilor

Poziţia articulaţiei este principala informaţie cerută pentru servo sistemele care controlează mişcarea mâinii. Aceasta informează circuitul de control asupra deplasării diferitelor articulaţii şi astfel despre configuraţia cinematică a mâinii. Există multe tipuri disponibile pentru senzorii de poziţie dintre care se pot aminti:

- Inductiv diferenţial,
- Encoder,
- Resolver,

- Linear variable Displacement Transformer (LVDT),
- Rotary variable Displacement Transformer (RVDT),
- Potenţiometrice,
- Dispozitive capacitive,
- Dispozitive bazate pe efectul curenţilor turbionari,
- Dispozitive optice.

Pentru determinarea poziţiei unghiulare a degetelor se poate utiliza, ca senzor, fibra optică. Fibra optică poate fi folosită atât ca mediu de ghidare a unui fascicol de lumină de la sursă la receptor, dar şi ca detector pentru măsurarea gradului de curbare a fibrei optice la aplicarea unei forţe exterioare. Forţa aplicată fibrei optice va modifica poziţia suprafeţei interne de reflexie deci şi unghiul de incidenţă a fascicolului de lumină. În cazul în care unghiul de incidenţă va fi mai mare decât unghiul limită de reflexie internă, lumina va fi absorbită. Cu cât fibra optică este deformată mai tare cu atât se pierde mai multă lumină şi fibra optică transmite un fascicol de lumină de intensitate mai scăzută. Poziţia unghiulară a degetelor (gradul de flexare, îndoire) se măsoară funcţie de atenuarea fascicolului de lumină prin fibra optică.

Figura 1.12 Atenuarea luminii funcţie de gradul de îndoire a fibrei optice

1.2.2 Senzori tactili

Senzorii tactili detectează forţa care acţionează între mâna robotului şi obiect. Ei cuprind senzori de atingere care detectează dacă robotul atinge sau nu obiectul (ca de exemplu un întrerupător), senzori de forţă de prindere care detectează forţa cu care obiectul este strâns, senzori de presiune care determină presiunea aplicată asupra obiectului şi senzori de alunecare care detectează dacă obiectul apucat alunecă sau nu. Senzorii de distribuţie a presiunii aparţin de asemenea acestei categorii. Ei determină distribuţia planară a presiunii pe suprafaţa obiectului care este prins de mână. O subtilitate este reprezentată de senzorii de tipul matrice în care senzorii sunt aranjaţi orizontal şi vertical într-o arie regulată. Senzorii forţei de prindere sunt utilizaţi pentru a comanda forţa de strângere iar senzorii de presiune de tip matrice sunt utilizaţi pentru a obţine date despre obiectul care este prins. În cazul în care se utilizează senzori de forţă, dacă obiectul a fost ridicat fără a fi spart sau scăpat, pentru operaţii ulterioare este necesară măsurarea presiunii aplicate şi a locului aplicării pe suprafaţa degetelor. Aceasta duce la necesitatea utilizării matricelor de senzori de presiune. În forma cea mai simplă o matrice de senzori de presiune nu este altceva decât o arie regulată de senzori de tensiune sau încărcare folosind elemente piezoelectrice sau senzori de presiune pe bază de semiconductoare. Elementele semiconductoare prezintă avantajul că sunt de dimensiuni mici, pot fi încapsulate la densitate mare şi sunt foarte sensibile. De asemenea se pot integra împreună cu circuite care realizează o anumită procesare a semnalului obţinându-se astfel un senzor inteligent.

În stadiul actual, există multe restricţii în construirea unei mâini de robot cu atâţia mecanoreceptori câţi există pe mâna umană (de exemplu structura mecanică, materialele pentru senzori, procesarea semnalelor, costul, etc.) Este rezonabil să se realizeze unele din cele mai importante caracteristici care se folosesc pentru manipulare cum ar fi de exemplu:

1. Semnal de contact pentru a determina dacă degetul este sau nu în contact cu un obiect,
2. Mărimea forţei de contact care este foarte importantă pentru a menţine o strângere stabilă a obiectului,
3. Localizarea suprafeţei de contact care indică poziţia şi orientarea regiunii de contact. Este de obicei foarte dificil de măsurat şi de asemenea foarte important pentru un deget să-şi ajusteze poziţia de apucare pentru a realiza o apucare stabilă,
4. Discriminarea între două puncte, care este abilitatea de a distinge doi stimuli. Este de dorit să existe posibilitatea de a măsura distribuţia forţei pe o suprafaţă de contact,
5. Forţa de forfecare care este mărimea forţei exercitate tangenţial la suprafaţa senzorului şi este foarte folositoare pentru determinarea alunecării unui obiect apucat de mână.

Din cele prezentate mai sus se poate concluziona care sunt cerinţele generale pentru un senzor tactil de pe un deget de robot, după cum urmează:

- Să fie capabile să măsoare mărimea, direcţia şi punctul de aplicare a forţei externe aplicate pe corpul degetului;
- Greutate mică astfel încât să nu cauzeze erori într-un control fin al articulaţiilor degetelor;
- Sensibilitate mare şi robusteţe ridicată pentru a suporta forţele externe (normale şi de forfecare), 0,5 - 10g;
- Domeniu dinamic bun şi rezoluţie spaţială rezonabilă, 1-2 mm, răspuns rapid, cel puţin 100 Hz;
- Histerezis şi derivă cu temperatura scăzute;
- Liniaritatea este de dorit dar poate fi tolerată şi neliniaritatea între anumite limite;
- Consum redus de putere;
- Uşor de fabricat şi să permită protecţie la supraîncărcare;
- Durabile şi ieftine.

1.2.2.1 Senzori de forţă

Senzorii de forţă determină forţa care acţionează în mâna robotului şi sunt indispensabili în operaţiile de asamblare şi ajustare când operaţia se execută în timpul ajustărilor fine ale forţei din braţului robotului. Aceşti senzori detectează solicitarea (tensiunea) în încheietura mâinii pentru a determina forţe şi momente în diverse direcţii.

În cea mai simplă formă un senzor de contact furnizează informaţii binare de tipul DA sau NU, indicând dacă există sau nu un contact. Mai mult decât atât tipurile mai avansate pot să dea unele sau toate din următoarele informaţii despre forţa sau forţele de contact:

- Mărimea forţei de contact,
- Localizarea punctului de aplicare a forţei în sistemul de referinţă a mâinii,
- Orientarea forţei aplicate în sistemul de referinţă a mâinii.

Senzorii de forţă de contact pot fi implementaţi cu:

- Timbre tensometrice (piezorezistive sau piezoelectrice),
- Rezistoare detectoare de forţă (FSR = Force Sensing Resistor)
- Senzori tactili funcţionând pe principii optice,
- Dispozitive bazate pe fluide electroreologice,
- Sondă de forţă implementabilă artificial (AIFP = Artificially Implantable Force Probe).

În plus faţă de acestea, unele aranjamente speciale a senzorilor de contact sunt utilizate pentru a determina fenomenul de alunecare, care de fapt determină forţele în direcţia tangenţială.

Senzori de forţă de forma segmentelor degetelor pot fi imaginaţi uşor, dar senzori de forţă de dimensiunile segmentelor degetelor umane pun probleme de miniaturizare. Consultând

literatura pentru diferite materiale pentru senzori au ieşit în evidenţă două tipuri de folii senzitive la presiune, una piezo-rezistivă (FSR) [78], şi una piezo-electrică (PVDF) [239].

Aceste tipuri de senzori sunt populare în rândul cercetătorilor care se ocupă cu senzorii tactili. Alte grupuri s-au specializat în miniaturizarea timbrelor tensometrice, dar majoritatea utilizează senzorii FSR pentru determinarea formei [74], şi foliile PVDF pentru comportarea dinamică. O abordare deosebită o reprezintă simţul tactil bazat pe transmisia ultrasunetelor într-un material elastic, dezvoltată de Shinoda [183]. Din păcate această tehnologie ingenioasă este greu de implementat.

Iniţial literatura a sugerat că foliile FSR sunt folositoare doar pentru detectarea profilurilor de presiune care se schimbă încet, în timp ce PVDF va genera un răspuns mult mai bun la schimbări rapide ale presiunii. Cum ambele măsurători sunt de dorit ar fi optimă adoptarea unei tehnici de stratificare, prin care senzorul PVDF ar fi plasat la suprafaţă şi FSR la bază aşa cum a fost realizat în prima fază şi de către Grupul de Neuroinformatică, din cadrul Facultăţii Tehnice a Universităţii Bielefeld (figura 6.13), [97].

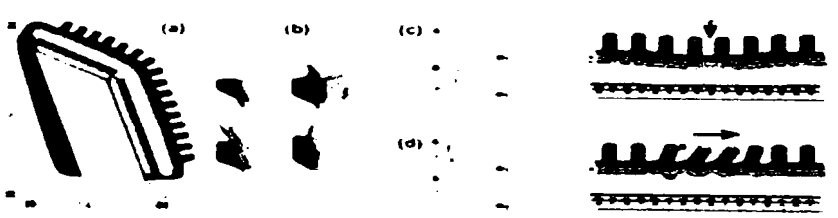


Figura 1.13 Senzorul tactil dezvoltat la Facultatea Tehnică a Universităţii Bielefeld

Construcţia senzorului tactil constă din patru paduri FSR plasate pe un corp de aluminiu, acoperite cu un elastomer şi o membrană de cauciuc cu excrescenţe pentru determinarea alunecării. Secţiunea (a) prezintă structura stratificată iar figura (b) prezintă fazele constructive. Cele două tipuri de senzori utilizate sunt capabile să detecteze atât presiunea statică (cu rezistorul FSR, (c)) cât şi mişcarea de alunecare (cu foliile PVDF ataşate membranei (d)). Construcţia a fost destul de reuşită dar insuficient de fiabilă.

O abordare mai nouă şi mai durabilă se poate baza pe folosirea numai a senzorilor FSR. Măsurătorile efectuate indică că domeniul de frecvenţă a senzorilor FSR este mai mare decât cel atribuit lor. Chiar este posibil să fie de ajuns pentru detectarea cu succes a alunecării.

1.2.2.2 Rezistoare detectoare de forţă - FSR (Force sensing resistor)

Această tehnologie unică a fost dezvoltată şi patentată de Interlink Electronics, CA [78]. Senzorul FSR este un dispozitiv cu un film subţire de polietilenă (PTF) care prezintă o scădere a rezistenţei la o creştere a forţei aplicate pe suprafaţa activă. Caracteristica rezistenţa funcţie de forţa aplicată este prezentată în figura 1.14.

În forma cea mai simplă acest senzor poate măsura numai forţele normale. Totuşi, se poate concepe o dispunere pe cele trei axe XYZ cum a fost sugerat de Liu, H. şi Hirzinger, G. în [74], pentru a le adăuga abilitatea de a detecta localizarea forţei şi de asemenea, alunecarea. Aceste dispozitive sunt disponibile în variate forme şi mărimi, cea mai mică fiind de 0,5 x 0,5 cm, cu domeniul pentru grosimi cuprins între 0,2 şi 1,25 mm. Aceşti senzori au o rezoluţie pentru forţă mai bună de 0.5% din domeniu maxim, cu o repetabilitate cuprinsă între +/-2% şi +/- 5%. Domeniul temperaturilor de lucru este cuprins între 30^0C şi 170^0C şi nu este afectat semnificativ de zgomot/vibraţie. Durata de viaţa este mai mare de 10 milioane de cicluri.

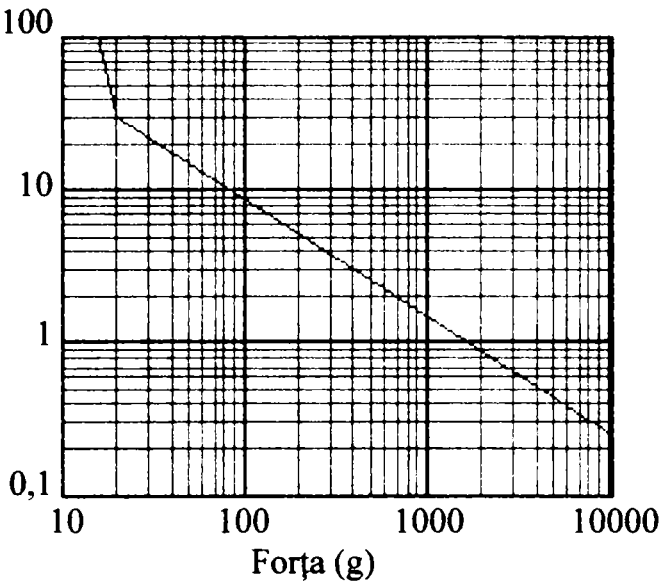


Figura 1.14 Caracteristica de răspuns a FSR (Rezistenţă funcţie de forţă)

Un senzor de forţă rezistiv este compus din două părţi. Prima este un material rezistiv aplicat pe un film. A doua este un set de contacte digitizate aplicate pe un alt film. Figura 1.15 prezintă această configuraţie. Materialul rezistiv face o legătură electrică între două seturi de conductoare de pe celălalt film. Când se aplică o forţă acestui senzor se realizează o conexiune mai bună între contacte, deci creşte conductivitatea. Pentru un domeniu larg de forţe, conductivitatea este o funcţie liniară cu forţa aplicată (F ~ G, F ~1/R). Figura 1.14 prezintă rezistenţa senzorului ca o funcţie de forţa aplicată. Este important de notat existenţa a trei regiuni de operarea senzorului. Prima este regiunea de tranziţie abruptă care are loc undeva în aproprierea unei forţe de 10 grame. În această regiune rezistenţa se schimba foarte repede. Această comportare este foarte folositoare pentru realizarea comutatoarelor folosind FSR. Deasupra acestei regiuni forţa este aproximativ proporţională cu 1/R până la atingerea regiunii de saturaţie. Când forţa atinge această valoare, forţe suplimentare nu conduc la descreşterea substanţială a rezistenţei.

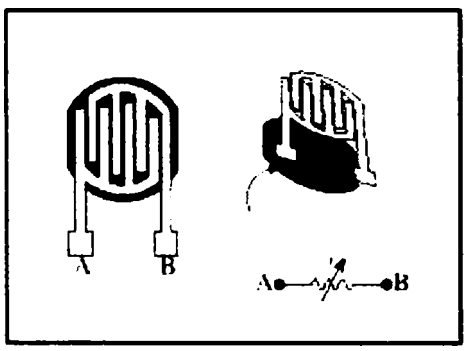


Figura 1.15 Rezistor detector de forţă tipic

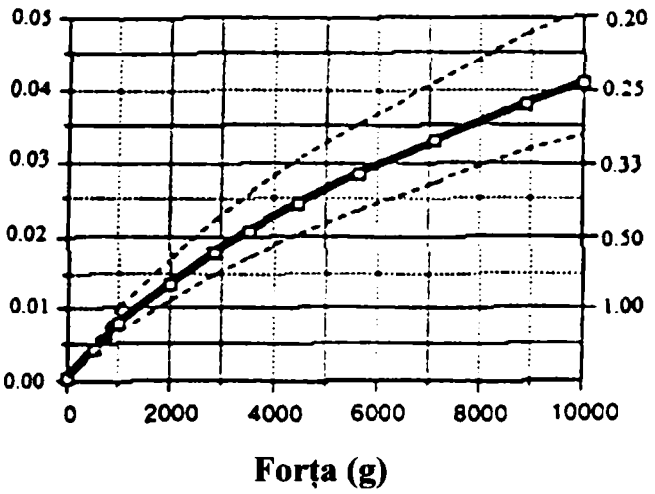


Figura 1.16 Conductanţă funcţie de forţa aplicată, pentru un FSR tipic

Figura 1.16 prezintă graficul conductanţei funcţie de forţa aplicată. Pentru un senzor FSR tipic. De remarcat că axa x este liniară şi deasupra punctului de frângere modificarea conductanţei este aproximativ liniară cu forţa.

1.3 Posibilităţi de utilizare a unui sistem senzorial pentru o mână artificială

Un sistem senzorial pentru o mână artificială poate avea multiple posibilităţi de utilizare. Mâna artificială poate fi mâna unui robot care trebuie comandată şi care necesită un sistem senzorial pentru controlul acesteia. De asemenea mâna artificială poate fi proteza implantată unui pacient care şi pierdut braţul şi care de asemenea are nevoie de un sistem senzorial pentru controlul în buclă închisă a acesteia. Pe de altă parte însă putem să folosim aceeaşi denumire de mână artificială şi pentru mâna unui pacient cu anatomia mâinii intactă dar cu pierderea parţială sau totală a funcţiei nervilor, deci şi a simţurilor care trebuiesc protezate prin intermediul unui sistem senzorial. Toate cele trei cazuri pot fi tratate unificat prin dezvoltarea unui sistem senzorial prevăzut cu simţurile minimale pentru o mână care să fie implementate într-o mănuşă senzorială care poate fi uşor purtată de pacient, sau poate îmbrăca o mână de robot antropomorfă.

1.3.1 Sistem senzorial pentru o mână de robot

În cazul utilizării unei mănuşi senzoriale ataşate unui mâini de robot pentru a furniza informaţii senzorial pentru sistemul de comandă, o reţea neuronală feed-forward poate învăţa relaţia neliniară dintre poziţia vârfului degetului şi măsurătorile mănuşii senzoriale, aşa cum se prezintă în figura 1.17.

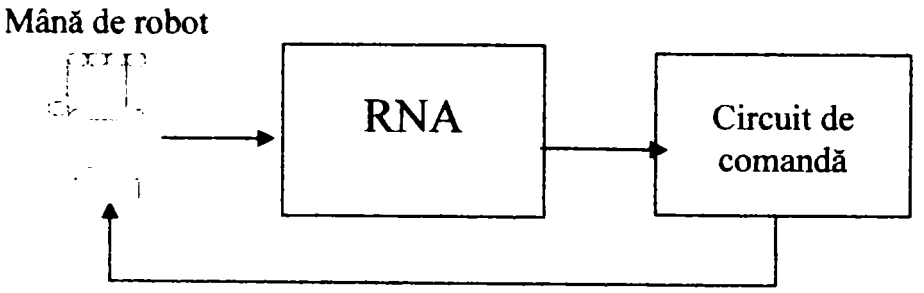


Figura 1.17 Utilizarea unei mănuşi senzoriale într-un sistem de comandă în buclă închisă pentru o mână de robot

1.3.2 Sistem senzorial pentru o proteză de mână

În acest caz proteza echipată cu senzori (mănuşă senzorială) furnizează semnale ce sunt procesate de o RNA care face asocierea semnalelor provenite de la senzori cu semnalele de comandă provenite de la sistemul nervos periferic.

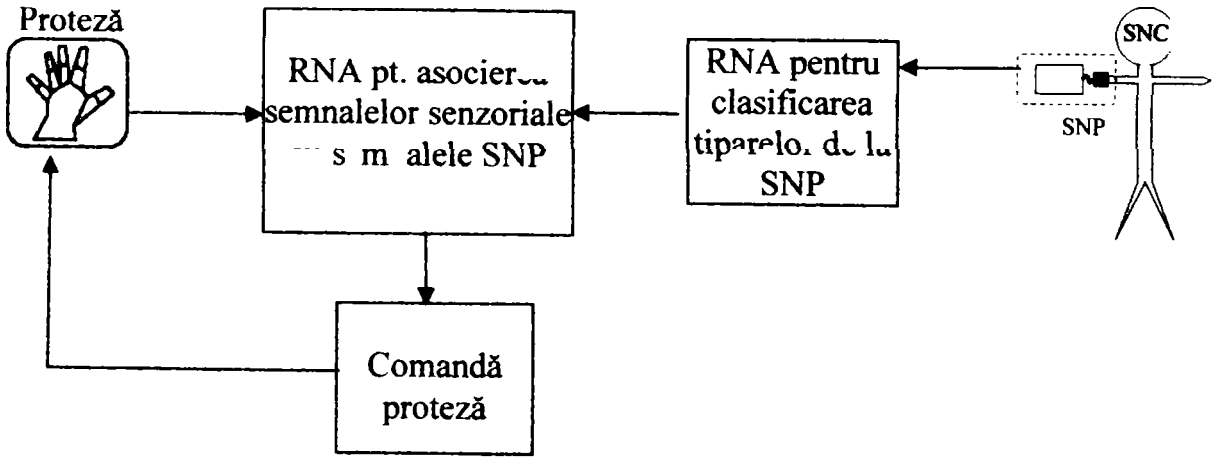


Figura 1.18 Mănuşa senzorială într-un sistem de comandă pentru o proteză

1.3.3 Sistem senzorial pentru o mână umană

În cazul utilizării sistemului senzorial pentru un pacient cu anatomia mâinii intactă dar cu pierderea parţială sau totală a simţurilor, reţeaua neuronală artificială clasifică semnalele senzoriale provenite de la mănuşa senzorială şi le transmite unui generator de semnal care stimulează prin intermediul unui neurosenzor sistemul nervos periferic al pacientului (figura 1.19).

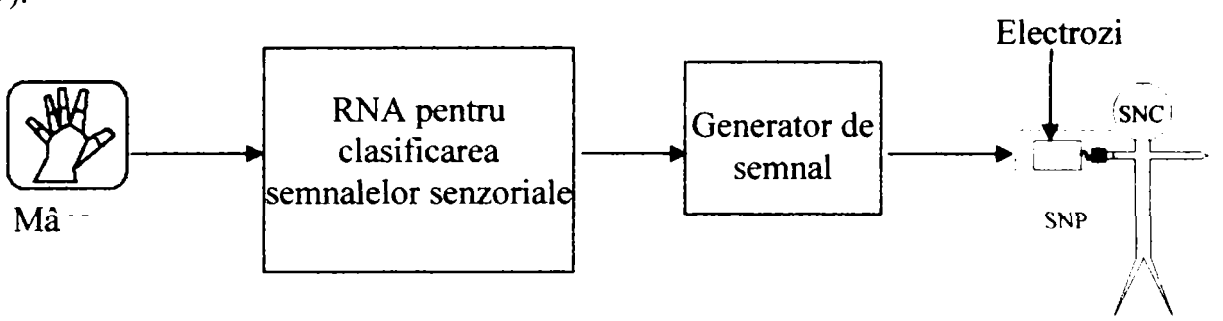


Figura 1.19 Utilizarea unei mănuşi senzoriale pentru furnizarea de informaţii senzoriale sistemului nervos periferic

1.3.4 Manipularea comandată cu mănuşi senzoriale

Această aplicaţie se referă la comanda de la distantă a unei mâini de robot. Sistemul include o mână de robot, şi o mână comandată de un operator ce poartă o mănuşa echipată cu senzori (figura 1.20). Controlul la distanţă poate însemna controlul unei mâini aflate la câţiva metrii (de exemplu în încăperea alăturată) sau la distanţe foarte mari sub apă sau în spaţiu.

Figura 1.20 Telemanipularea folosind mănuşi senzoriale

1.3.5 Operarea de la distanţă cu mănuşi senzoriale

Utilizarea unei mănuşi senzoriale şi a unui sistem de recunoaştere a gesturilor care permite controlul unor unelte chirurgicale aflate la distanţă sau unelte pentru chirurgia endoscopică mai complexe având mai multe grade de libertate.

1.3.6 Interfaţă pentru recunoaşterea limbajului semnelor

Mănuşa senzorială poate servi ca dispozitiv de intrare pentru un sistem de recunoaştere a semnelor utilizate pentru comunicare de către persoanele cu dizabilităţi de vorbire. Utilizarea unui sintetizator de voce comandat de sistemul de recunoaştere permite acestor persoane să comunice.

1.3.7 Interfeţe om-calculator

Utilizarea unor mănuşi senzoriale ca dispozitive de intrare pentru calculator care să permită manipularea unor obiecte în realitatea virtuală, este o temă de mare actualitate. De asemenea ele pot fi de un real folos persoanelor cu anumite dizabilităţi să folosească calculatorul.

1.4 Obiectivele şi soluţiile propuse de prezenta teză

În ultimul timp există un interes crescând pentru realizarea de interfeţe senzoriale inteligente, cât mai naturale, care să nu necesite cunoştinţe de specialitate pentru setarea parametrilor. Este de dorit ca aceste interfeţe să posede capacitate de învăţare şi comportament adaptiv, ceea ce se poate obţine prin utilizarea reţelelor neuronale.

Domeniul aplicaţiilor posibile pentru interfeţe senzoriale inteligente este foarte larg, de la interfeţele om-calculator inteligente care să permită persoanelor cu orice tip de handicap să folosească calculatorul şi să comunice până la orice tip de aparat industrial sau casnic cu capacitate de învăţare şi adaptare.

Lucrarea de faţă şi-a propus realizarea unui mediu integrat hardware-software care permite dezvoltarea rapidă a unei interfeţe inteligente folosind:

- senzori specifici aplicaţiei
- module hardware care pot fi uşor interconectate
- module descrise cu ajutorul limbajul de descriere hardware VHDL (**V**ery High Speed Integrated Circuit, **H**ardware **D**escription **L**anguage) care gestionează senzorii („drivere VHDL")
- reţele neuronale artificiale (utilizate pentru extragerea trăsăturilor, recunoaşterea tiparelor)

Având acest cadru, dezvoltarea unui senzor inteligent nou, constă în proiectarea şi sinteza unor drivere noi (descrise în VHDL) pentru senzorii folosiţi şi a unor reţele neuronale specifice aplicaţiei.

Analizând aplicaţiile din paragraful 1.3, s-a ajuns la concluzia că implementarea unui sistem de recunoaştere a gesturilor unei mâini folosind reţele neuronale implementate în circuite logice programabile de tip FPGA (Field Programmable Gate Arrays), ar prezenta un pas important în crearea unei mâini artificiale complet echipată cu senzori. Din păcate realizările de până acum prezintă doar rezolvări parţiale a acestei teme, şi în consecinţă abordarea ei prezintă interes.

Soluţia propusă pentru implementarea unui sistem senzorial pentru o mână artificială se compune din următoarele blocuri funcţionale:

1.4.1 Senzorii pentru mâna artificială

Senzorii esenţiali pentru o mână artificială sunt senzorii de poziţie a articulaţiei mâinii şi degetelor şi senzorii de forţă de contact. Sistemul realizat constă dintr-o mănuşă senzorială având 5 senzori de poziţie a articulaţiei degetelor (câte unul pentru fiecare deget) un senzor de înclinare şi un senzor de răsucire a mâinii. Acestei mănuşi i se pot ataşa senzori de forţă de tip FSR care se pretează pentru lipire pe orice suprafaţă şi furnizează informaţii cu privire la mărimea forţei cât şi la locul de exercitare a contactului. Se pot folosii 6 senzori de forţă, câte unul pentru vârful fiecărui deget şi unul pentru palmă. Mănuşa senzorială poate fi folosită în oricare din următoarele cazuri: mână de robot antropomorfă, proteză sau mână umană.

1.4.2 Circuitele de preprocesare a semnalelor

Senzorii au fost astfel aleşi încât ei să se poată interfaţa uşor cu circuite digitale. În consecinţă circuitele folosite pentru achiziţia şi procesarea semnalelor senzoriale constau dintr-un sistem de achiziţie cu un microcontroler şi un sistem de dezvoltare pentru circuite FPGA. Pentru experimentări s-au folosit diverse sisteme de dezvoltare pentru circuite FPGA, care permit implementarea unor circuite specializate în preprocesarea semnalelor pentru sistemul senzorial propus şi reţele neuronale pentru clasificarea semnalelor senzoriale.

Configuraţia soft utilizată conferă sistemului o mare flexibilitate prin aceea că permite partajarea specificaţiilor proiectului între FPGA şi microcontroler. Astfel partea soft a aplicaţiei se poate implementa în microcontroler în timp ce partea hard se implementează în circuitul logic programabil.

Calitatea esenţială a sistemului experimental realizat, constă în faptul că este uşor reconfigurabil prin utilizarea unor unelte software, ceea ce permite experimentarea unui număr mare de circuite de procesare respectiv de reţele neuronale, pentru identificarea circuitului optim aplicaţiei. Partea hardware poate fi descrisă utilizând limbajul VHDL iar partea soft este descrisă utilizând limbajul de asamblare a microcontrolerului.

1.4.3 Reţele neuronale artificiale

Utilizarea reţelelor neuronale artificiale în clasificarea semnalelor senzoriale este esenţială. Dată fiind cantitatea foarte mare a informaţiilor senzoriale rezultă un număr foarte mare de combinaţii (gesturi) posibile. Fiecare senzor utilizat are ieşirea pe 8 biţi deci cei 13 senzori pot furniza $2^{8x13} = 2^{104}$ combinaţii posibile. Nu se impune identificarea tuturor combinaţiilor posibile ci doar a gesturilor esenţiale pentru o mână. Pentru aceasta se pot folosii reţele neuronale cu autoorganizare. Metodele de învăţare posibile sunt cele cu algoritm de tip hebbian şi competitiv.

Reţelele neuronale pot oferii soluţii la următoarele probleme identificate pe parcursul documentării:

- Interfeţele echipate cu RNA pot încorpora strategii bazate pe rezultate obţinute din experienţe cognitive.
- Reţelele neuronale artificiale feed-forward pot învăţa relaţia neliniară dintre poziţia vârfului degetului şi semnalele furnizate de mănuşa de date.
- Procesarea semnalelor provenite de la senzori pentru eliminarea interferenţelor şi zgomotelor pentru obţinerea unor semnale curate.

- O reţea neuronală artificială secundară poate clasifica semnalele pentru a asocia anumite clase de semnale cu gesturi ale mâinii.
- Posibilitatea includerii în acelaşi chip a circuitelor de procesare şi a RNA duce la realizarea unor sisteme dedicate cu reţele neuronale specifice aplicaţiei .

În consecinţă utilizarea reţelelor neuronale pentru adăugarea proprietăţilor de învăţare şi de adaptabilitate pentru un senzor inteligent este esenţială iar utilizarea circuitelor logice programabile reprezintă o facilitate importantă în ceea ce priveşte implementare hardware a acestora.

1.5 Structura tezei

Prezenta teză abordează problema sistemului senzorial pentru o mână artificială. În cadrul acestui capitol au fost trecute în revistă realizările de mâini artificiale existente şi sistemul senzorial al acestora. Din analiza făcută a reieşit că un sistem senzorial minimal trebuie să conţină senzori de poziţie şi de forţă. A rezultat de asemenea necesitatea utilizării reţelelor neuronale pentru a realiza un sistem senzorial capabil să înveţe gesturi noi şi să se adapteze utilizatorului. De aceea o mare parte din teză este dedicată implementării RNA în circuite logice programabile de tipul FPGA.

Capitolul II prezintă posibilităţile existente în ceea ce priveşte implementarea reţelelor neuronale. Se face clasificarea implementărilor după diverse criterii şi apoi sunt trecute în revistă implementările software şi hardware cu avantajele si dezavantajele lor. Se prezintă o comparaţie între implementările în circuite integrate specifice aplicaţiei (ASIC) şi cele în circuite FPGA. O altă comparaţie se referă la procesoarele digitale de semnal DSP (Digital Signal Procesor) şi FPGA. Implementările hibride hardware-software sunt prezentate în continuare şi se evidenţiază avantajele conferite de această abordare. Concluzia capitolului este că implementările hibride prezintă premisele atingerii scopului propus şi în consecinţă este adoptată pentru implementarea RNA.

Întrucât teza se concentrează asupra implementării RNA în FPGA în capitolul III se face o descriere a arhitecturii circuitelor FPGA şi a problemelor care se pun la implementarea în acest tip de circuit. Pentru a putea face o comparaţie între implementările existente se face o clasificare a acestora şi o prezentare a performanţelor obţinute de alte lucrări.

Capitolul IV prezintă METODA, prima contribuţie importantă din cadrul prezentei teze. Este vorba de elaborarea unui mediu integrat pentru implementarea hibridă hardware-software a reţelelor neuronale. Sunt comparate pentru început etapele proiectării tradiţionale a RNA şi etapele metodei propuse. Se prezintă în continuare structura hardware şi cea software implicată în acest proces. Sunt detailate etapele implementării şi se prezintă blocurile specifice RNA create folosind această metodă. Capitolul se încheie cu prezentarea modelului unui neuron realizat cu blocurile din biblioteca RNA.

În capitolul V se face uz de metoda dezvoltată pentru implementarea reţelelor neuronale feed-forward. După prezentarea structurii şi a metodei de antrenare a RNA FF se prezintă posibilităţile existente în ceea ce priveşte gradul de paralelism al implementării. Se prezintă implementări de RNA FF cu unul şi două straturi cu paralelism de neuron sau de strat antrenate prin metoda Hebbiană şi cu propagarea înapoi a erorii. Apar aici primele contribuţii cu privire la optimizarea implementărilor din punct de vedere a ariei şi a frecvenţei. Rezultatele obţinute în urma implementării sunt identice sau superioare modelelor software, ceea ce demonstrează eficienţa metodei dezvoltate.

Implementarea reţelelor competitive simple reprezintă obiectivul capitolului VI. De aceea se prezintă structura unei astfel de reţele şi modul de implementare software a fazei de antrenare. Pentru implementarea hardware a fazei de propagare se prezintă noile blocuri

specifice RNA competitive, create de autor care se adăuga bibliotecii RNA dezvoltate. Sunt implementate reţele competitive cu paralelism de neuron şi paralelism de strat. Se prezintă o simplificare a algoritmului de calcul a distanţei euclidiene în urma căreia implementarea hardware se face mai uşor.

Capitolul VII introduce o altă contribuţie importantă a prezentei teze şi anume elaborarea unui sistem de recunoaştere a gesturilor statice ale unei mâini senzoriale folosind două reţele neuronale implementate hardware. Prima reţea are rol de preprocesare a datelor senzoriale în timp ce cea doua reţea face clasificarea datelor de intrare într-una din gesturile învăţate. Tipul reţelei clasificatoare este diferit de cele folosite de alţi autori. Capitolul debutează cu o prezentare a soluţiilor alese pentru sistemul senzorial şi a metodei de recunoaştere adoptate. În continuare se prezintă sistemul de recunoaştere compus din platforma hardware şi pachetul software, etapele implementării şi simularea sistemului de recunoaştere. Urmează modelarea şi implementarea sistemului folosind reţelele dezvoltate în cadrul capitolelor V şi VI. În finalul capitolului se prezintă compararea rezultatelor obţinute cu alte metode şi posibilităţile de aplicare ale acestui sistem.

Teza se încheie cu prezentarea concluziilor şi a contribuţiilor aduse pe parcursul elaborării acesteia.

Capitolul II

Implementări hardware de reţele neuronale artificiale

Acest capitol prezintă o clasificare în funcţie de diverse criterii şi apoi o trecere în revistă a implementărilor software şi hardware a reţelelor neuronale artificiale. Sunt prezentate avantajele şi dezavantajele implementărilor software şi respectiv a implementărilor hardware analogice, digitale şi hibride.

2.1 Introducere

Unul din cele mai active domenii de cercetare din zilelor noastre este cel al implementării software sau hardware a reţelelor neuronale. În ultimul deceniu au fost dezvoltate o mare varietate de circuite care să exploateze paralelismul propriu modelului reţelelor neuronale artificiale. Acestea includ implementări analogice, digitale şi hibride.

În anii precedenţi mulţi cercetători au încercat să exploateze caracteristicile proprii reţelelor neuronale (cum ar fi structura regulată, adaptabilitatea, toleranţa la defecte) pentru a realiza implementări hardware eficiente, care erau considerate mai performante decât implementările realizate folosind tehnica de calcul.

Datorită creşterii vitezei procesoarelor utilizate în calculatoarele personale (PC) şi în staţiile grafice precum şi a procesoarelor digitale de semnal, performanţele oferite de implementările hardware (cost, viteza, etc.) depăşesc rareori pe cele ale implementărilor software pe un procesor de uz general. Din acest motiv implementările hardware nu sunt atât de atractive cum păreau la început şi aceasta explică utilizarea lor redusă în aplicaţiile practice precum şi insuccesul circuitelor neuronale comerciale.

Se pune deci întrebarea daca mai are rost să se realizeze implementări hardware de reţele neuronale artificiale ? Dacă răspunsul este pozitiv care sunt aplicaţiile pentru care o implementare hardware este mai eficientă. Există o serie de aplicaţii în care implementările hardware oferă beneficii importante la un preţ mult mai scăzut decât alte tehnici de implementare disponibile [132], şi anume:

- ca blocuri componente ale sistemelor dedicate (embedded systems), în majoritatea echipamentelor electronice de larg consum şi electronica auto;
- în sistemele pe un chip (system-on-a-chip), unde toate componentele sistemului, inclusiv circuitele neuronale şi fuzzy, sunt implementate pe acelaşi chip;
- în aplicaţiile unde este necesară o viteză foarte mare, de exemplu în detectoarele de coliziune utilizate în experienţele din domeniul fizicii energiilor înalte;
- în aplicaţiile unde este necesar un consum de putere foarte scăzut, de exemplu în echipamentele alimentate de la baterie cum ar fi stimulatoarele cardiace;
- în aplicaţiile unde este necesară o toleranţă crescută la defecte, de exemplu în aplicaţiile spaţiale sau pentru a creşte randamentul în procesul de fabricaţie a circuitelor integrate.

2.2 Clasificarea implementărilor reţelelor neuronale artificiale

Există o gamă foarte largă de implementări software şi hardware a reţelelor neuronale artificiale şi există de asemenea în literatură câteva studii privind aceste implementări. Având în vedere diversitatea implementărilor clasificarea acestora este destul de dificilă şi diferă de la un autor la altul [32], [106], [210], [229]. În această lucrare s-a ales clasificarea propusă de

Schoenauer în [210]. Clasificarea se poate face după arhitectura sistemului, gradul de paralelism, tipul de paralelism implementat, comunicația între elementele de procesare, reprezentarea numerică, etc.

2.2.1 Clasificarea după arhitectura sistemului

În funcție de arhitectura sistemului există implementări de la un singur neurochip până la neurocomputere complete. Neurochip-urile pot servi ca un periferic pentru microprocesor sau pot fi utilizate independent pe o placă. Plăcile acceleratoare sunt conectate la un PC şi de obicei oferă mai multă flexibilitate decât un neurochip. Ele sunt echipate cu un procesor de uz general sau cu procesor dedicat. Exemple de neurocomputere care utilizează neurochip-uri dedicate sunt CNAPS [39] realizat de firma Adaptive Solutions şi SINAPSE al firmei Siemens . Există de asemenea plăci acceleratoare care utilizează aceleaşi neurochip-uri.

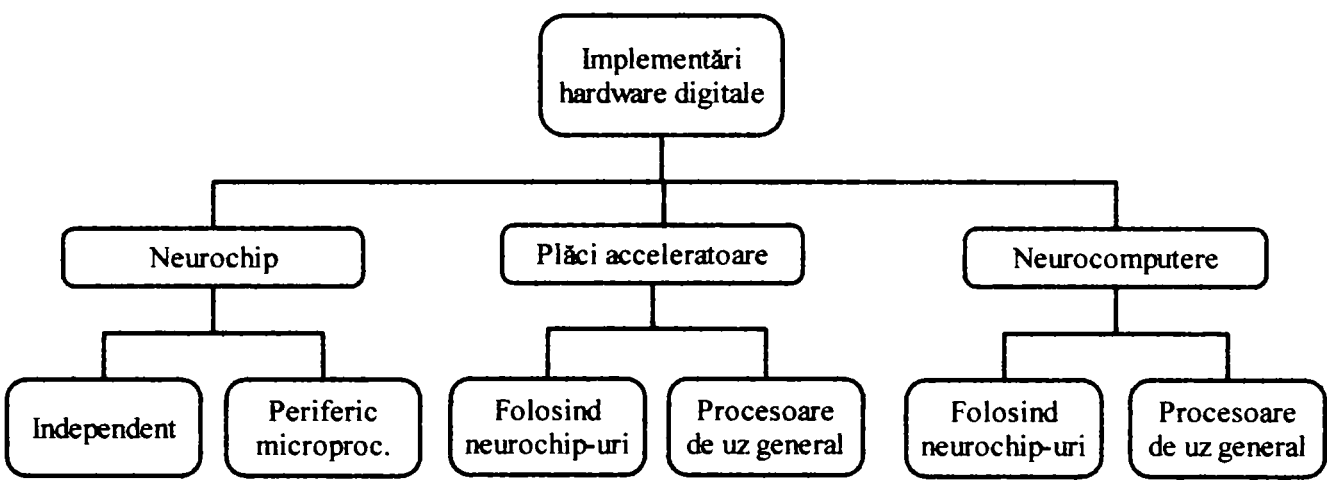


Figura 2.1 Clasificarea după arhitectura sistemului

2.2.2 Clasificarea după gradul de paralelism

Numărul de elemente de procesare determină gradul de paralelism al sistemului. Cu cât există mai multe unități de procesare paralele cu atât sunt prelucrate mai rapid datele. Dar gradul de paralelism este costisitor în raport cu aria de siliciu ocupată. De aceea sistemele cu un grad mare de paralelism întrebuințează elemente de procesare mai simple. Astfel gradul de paralelism se întinde de la implementări paralele cu doar câteva elemente de procesare, cunoscute sub denumirea de sisteme cu granulație mare (coarse-grained), până la implementări unu-la-unu a nodurilor de procesare neuronală, numite masiv paralele. Intre aceste clase nu există limite precise, ceea ce se poate observa în figura 2.2 din suprapunerea granițelor pentru clase învecinate.

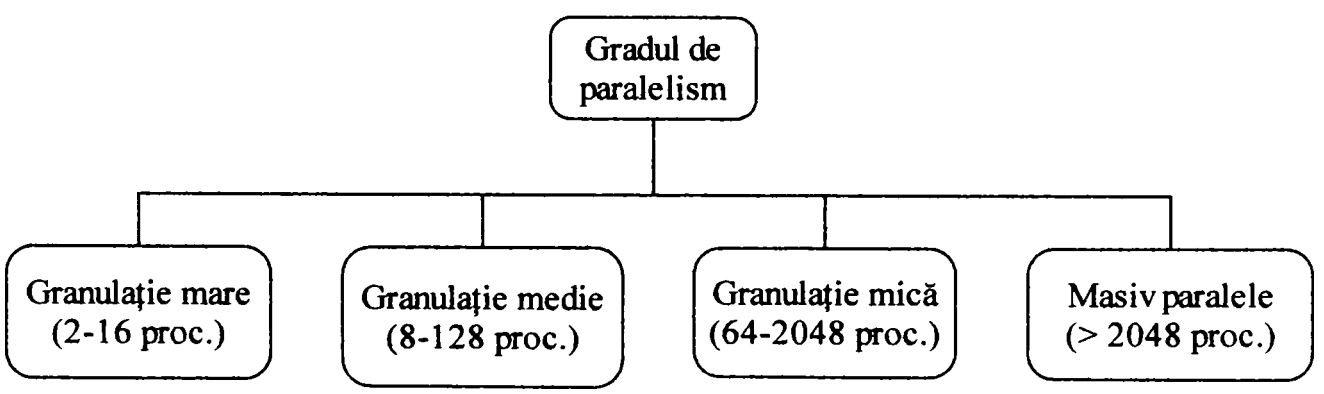


Figura 2.2 Clasificarea după gradul de paralelism

2.2.3 Clasificarea după reprezentarea numerică

În general rețelele neuronale artificiale au cerințe mici față de precizia de reprezentare numerică, precizia necesară fiind dependentă de aplicație și algoritm. Profitând de această proprietate, implementările digitale folosesc reprezentarea în virgulă fixă. Reprezentarea în virgulă fixă este mai simplă decât cea în virgulă mobilă și ocupă mai puțină arie de siliciu, ceea ce se traduce într-un cost mai scăzut. Alte reprezentări numerice cum ar fi codificarea în tren de impulsuri stocastic.

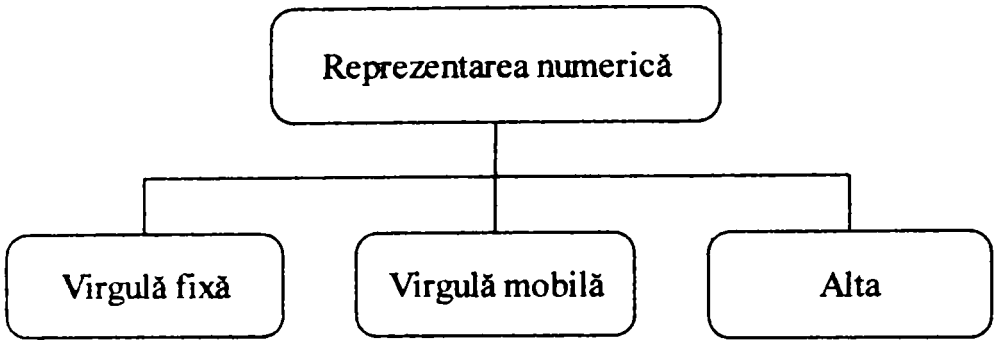


Figura 2.3 Clasificarea după reprezentarea numerică

2.2.4 Clasificarea după tipul de paralelism

În funcție de corespondența realizată în procesul de mapare a elementelor rețelei neuronale la elementele de procesare paralelă putem avea un paralelism al sinapselor, al neuronilor, al straturilor sau chiar o mapare a întregii rețele neuronale la un singur element de procesare.

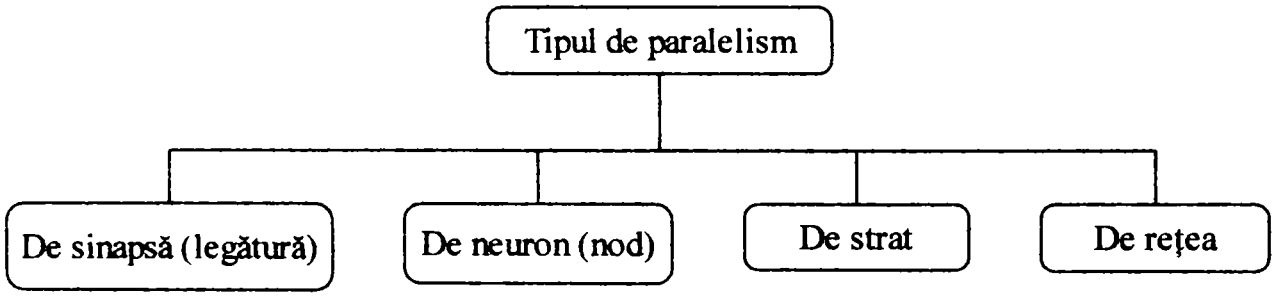


Figura 2.4 Clasificarea după tipul de paralelism

Paralelismul sinapselor este o mapare a fiecărei sinapse a unui neuron la un element de procesare diferit. Astfel o rețea de N neuroni, fiecare cu n sinapse se implementează folosind Nxn elemente de procesare care lucrează în paralel. Este cel mai mare grad de paralelism care poate fi atins într-o rețea neuronală, toate conexiunile unui neuron cu alți neuroni sunt calculate în același timp. Aceasta implementare este rapidă dar inflexibilă, deoarece în cazul unei rețele de dimensiuni diferite este necesară modificarea arhitecturii. Transferul de date între straturi poate avea loc paralel, deoarece un neuron are nevoie de toate semnalele sinaptice în același timp. Acest paralelism conduce la performanțe ridicate ale rețelei dar o arhitectură poate fi folosită doar pentru o anumită implementare.

În cazul paralelismului nodurilor (neuronilor), toate sinapsele, intrarea netă și funcția de activare a unui neuron sunt mapate la același element de procesare. O rețea compusă din N neuroni este implementată folosind N elemente de procesare care lucrează în paralel. Elementul de memorare a ponderilor trebuie să fie individual pentru fiecare neuron, deoarece fiecare neuron trebuie să acceseze memoria cu ponderile proprii în același timp. Transferul de date între straturi trebuie să fie serial deoarece un neuron poate calcula doar o conexiune la un

moment dat. Neuronul mai are nevoie de element de stocare temporară pentru datele de ieşire deoarece transferul serial al datelor între straturi permite trimiterea unui singur cuvânt de date la un moment dat de timp.

Paralelismul straturilor este o mapare a unui strat la cel puțin un element de procesare, astfel pentru o rețea cu L straturi sunt necesare cel puțin L elemente de procesare care lucrează în paralel. Este necesar un element de stocare central pentru ponderi şi un buffer pentru memorarea temporară a valorilor de intrare şi ieşire. Transferul de date între straturi poate fi serial sau paralel, în funcție de numărul de elemente de procesare pe strat. Cerințele hardware sunt minime deci implementare poate fi realizată folosind un calculator personal.

2.2.5 Clasificare după comunicația dintre elementele de procesare

Elementele de procesare paralelă cresc viteza de calcul doar când nu sunt în stare inactivă. Pentru performanțele rețelei este esențială comunicația dintre elementele de procesare, astfel încât acestea să fie „alimentate" cu date suficiente. Cele mai utilizate metode de comunicație din rețele neuronale artificiale sunt prezentate în figura 2.5

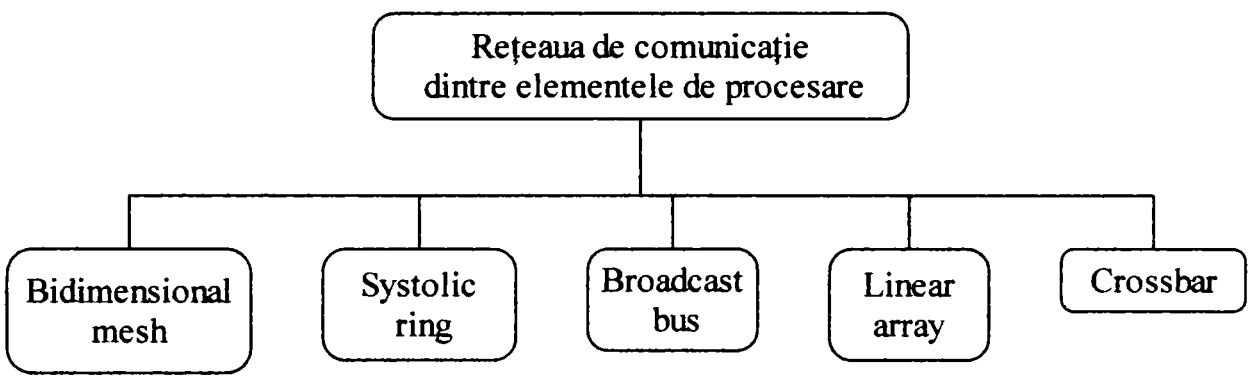


Figura 2.5 Clasificare după comunicația dintre elementele de procesare

2.2.6 Clasificarea RNA după modul de implementare

Numărul de cercetări legate de implementarea rețelelor neuronale desfăşurate în lumea întreagă este mult prea mare pentru o trecere în revistă şi o clasificare atotcuprinzătoare. În continuare se prezintă clasificarea propusă de Ruckert (1993) care a fost preluată de mai mulți autori [106], [229].

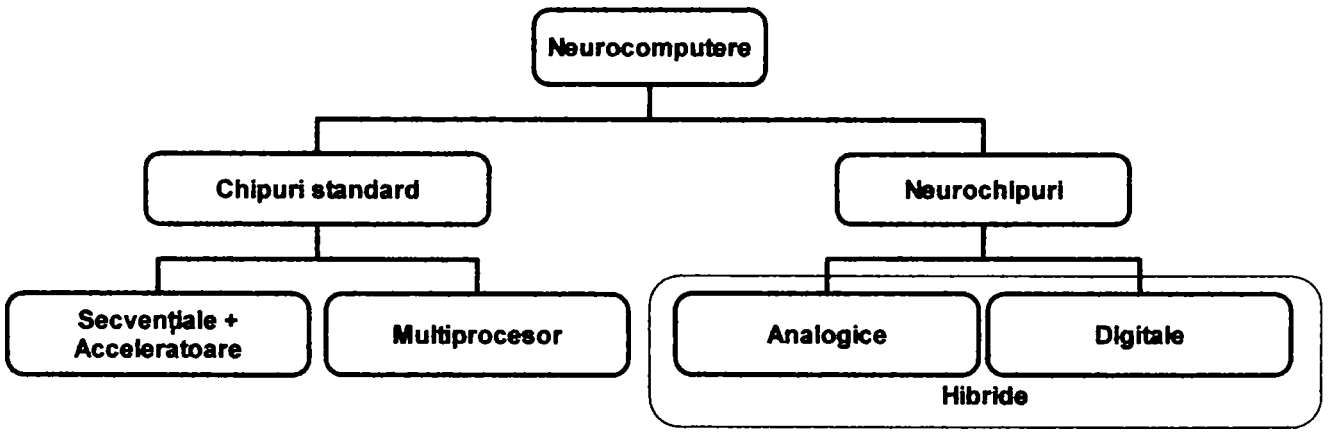


Figura 2.6

Primele două categorii reprezintă implementări bazate pe circuite integrate standard şi ele sunt denumite de anumiți autori ca implementări software a rețelelor neuronale. Ele constau din calculatoare personale sau stații grafice (echipate cu procesoarele secvențiale)

eventual cu o placă de accelerare şi sisteme paralele multiprocesor care de obicei rulează independent şi sunt monitorizate de un calculator gazdă.

Următoarele categorii constau în implementări realizate cu circuite neuronale dedicate. Acestea pot fi analogice, digitale sau hibride.

În continuare sunt prezentate diversele tehnici de implementare pe baza clasificării de mai sus.

2.3 *Implementarea software a reţelelor neuronale artificiale*

Prin implementări software se înţeleg în general acele implementări care se bazează pe un soft ce rulează pe un procesor de uz general (pe un PC) eventual cu o placă de accelerare. Există mai multe posibilităţi de implementare software a reţelelor neuronale, care vor fi prezentate în continuare.

- Programe software dedicate simulării reţelelor neuronale (cum ar fi Aspyrine, Explorenet, Neural Ware, Neurosolutions, etc.). Acestea sunt în general foarte complexe şi flexibile dar destul de lente, acesta fiind preţul plătit pentru flexibilitate.
  Plăcile de accelerare neuro-fuzzy sunt plăci ce pot fi inserate într-un calculator de uz general (PC sau staţie grafică). Ele sunt asociate programelor de simulare neuro-fuzzy oferind deci aceeaşi flexibilitate la o viteză de execuţie mult mai mare dar şi la un preţ de cost mult mai mare.
- Pachete de funcţii dedicate specifice domeniului RNA (cum este de exemplu Neural Network Toolbox) pentru programe de simulare de uz general cum sunt Matlab sau Maple. Acestea au performanţe comparabile sau chiar mai bune decât programele de simulare dedicate, fiind mult mai deschise în ceea ce priveşte conectarea blocurilor neuronale cu alte blocuri matematice, grafice şi funcţii de control. De asemenea ele sunt şi mai ieftine dacă se presupune că programul principal este deja achiziţionat. Aceasta a dus la o mai mare răspândire a acestei metode decât a programelor dedicate de simulare.
  Recent, firma Xilinx a dezvoltat un program care face conversia unei descrieri a sistemului din Simulink într-o descriere hardware care poate fi sintetizată şi apoi implementată în circuite programabile de tipul FPGA sau în ASIC
- Implementări prin programe scrise într-un limbaj de programare cum ar fi de exemplu C/C++. Acestea au în general o viteză de execuţie superioară (până la de zece ori mai mare), dar necesită o fază de elaborare mai îndelungată iar codul rezultat este mai lung. De aceea de obicei ele sunt elaborate după o fază prealabilă de dezvoltare realizată folosind programele sau pachetele dedicate.
- Implementări software ca parte componentă a unui sistem hibrid hardware-software. În prezent asemenea sisteme reprezintă una din cele mai atractive şi promiţătoare domenii de cercetare din aria implementărilor hardware.

2.3.1 Implementări bazate pe calculatoare + plăci de accelerare

Plăcile de accelerare sunt cele mai uzuale sisteme neuronale hardware comerciale, deoarece sunt ieftine, disponibile, simple de conectat la un calculator, furnizate de obicei cu software prietenos. Uneori utilizatorul este deja familiarizat cu unealta software care rulează pe sistemele standard, ne-accelerate, astfel încât prin adăugarea unei plăci acceleratoare, acesta va observa doar o reducere a timpului de antrenare sau propagare a reţelei neuronale.

Creşterea de viteză este de aproximativ un ordin de mărime faţă de implementarea secvenţială.

Dintre plăcile de accelerare disponibile comercial se amintesc [106]:

- **ANZA plus**, (Hecht-Nielsen Cooperation), conţine un procesor MotorolaNC68020 şi un coprocesor în virgulă mobilă MC 68881. Performanţele lui sunt: 1M elemente de procesare (PE), 1,5 M conexiuni, 1500 actualizări pe secundă a ponderilor (CUPS), 6 M conexiuni pe secundă (MCPS);
- **Ni1000 Recognition Accelerator Hardware**, (Intel Corp.), este o placă PC dezvoltată pentru recunoaşterea optică a caracterelor (OCR). Reţelele implementate sunt RNA bazate pe funcţii radiale, şi backpropagation. Creşterea de viteză este 100-1000 fată de implementarea în DSP sau procesor uzual;
- **SAIC SIGMA-1,** (Science Applications International Corporation), este un sistem construit cu un procesor în virgulă mobilă Delta. Performanţe: 3,1 M PE, 11 MCUPS.

Alte plăci comerciale sunt:

- **NT6000** (Neural Technologies Limited, HNC, and California Scientific Software)
- **Balboa 860 coprocessor board** (HNC, CA, USA)
- **IBM NEP** (IBM Scientific Center, Palo Alto)
- **NBS** (Micro Devices, FL, USA)
- **Neuro Turbo I, Neuro Turbo II** (Nagoya Institute of Technology, Mitec Corp.).

2.3.2 Implementări cu procesoare de uz general

Implementarea RNA este posibilă folosind unul sau mai multe procesoare de uz general. Cu toate că aceste implementări nu sunt foarte eficiente, datorită avantajelor legate de larga lor disponibilitate şi a preţului relativ scăzut, există foarte multe neurocomputere realizate cu procesoare de uz general [106], [229]:

- **WISARD** (Imperial College London, UK)
- **Mark III and IV** (TRW-HNC, CA, USA). Marc III a fost primul neurocomputer disponibil comercial. Performanţele lui sunt: Mark III: compus din 15 procesoare, MC68020 + MC68881. 65.000 PE, peste 1.000.000 conexiuni, 0,45 MCPS; Mark IV: 236.000 PE. 5,5 M conexiuni, 5 MCUPS.
- **Sandy/8** (Fujitsu Laboratories, Japan) realizat cu 256 DSP-uri TMS320C30. Performanţa estimată pentru o reţea backpropagation: 135 MCUPS.
- **GCN** (Sony Corp., Japan)
- **Topsi** (Texas Instruments, USA), este o arhitectură bazată pe DSP-uri, cu 100-1000 module conţinând un DSP şi un procesor de uz general. Performanţe: 150 MCUPS – 1,4 GCUPS.
- **DREAM Machine** (Hughes Research Laboratories, CA, USA)
- **RAP** (ICSI, Berkeley, CA), dezvoltat de ICSI (International Computer Science Institute) constă în 4 până la 40 procesoare digitale de semnal în virgulă mobilă Texas Instruments de tipul TI-TMS320C30, 256 Kocteţi memorie RAM statică şi 4 Mocteţi memorie RAM dinamică. Aceste componente sunt conectate prin intermediul unor circuite FPGA Xilinx. O singură placă poate atinge 57 MCPS la calculul unei reţele MLP (multi-layer perceptron) la propagarea înainte, respectiv 13,2 MCPS cu antrenare backpropagation.

- **COKOS** (University of Tübingen, Germany), CO-processor for Kohonen's Self-organizing feature map. Performanţe: 16 MCUPS (paşi de antrenare Kohonen).
- **REMAP** (Luleå University of Technology and Halmstad University, Sweden), REMAP (Real-time Embedded Modular Adaptive Parallel processor project) are o performanţă estimată de 100 MCPS - 1000 MCPS.
- **General Purpose Parallel Neurocomputer** (Tampere University, Finland), acest neurocomputere de uz general constă într-un număr de procesoare DSP identice (TMS320C25), care poate implementa reţele neuronale diverse, cum ar fi MLP, backpropagation, Kohonen SOFM, etc.
- **TI NETSIM** (Texas Instruments and Cambridge University, UK), constă într-o serie de plăci emulatoare RNA conectate într-o arie 3-D. O placă este construită cu procesoare 80188. Pot fi adresate până la 15 plăci NETSIM în fiecare din cele 3 dimensiuni. Fiecare placă implementează 256 noduri cu 256 sinapse pe neuron. Neurocomputer-ul NETSIM suportă majoritatea arhitecturilor RNA. Performanţa globală este de 450 MCS şi 90 MCUPS.
- **GeNet** - Generic Network, (IBM Research Division, Munich, Germany)

2.4 Implementarea hardware a reţelelor neuronale artificiale

În ultimii zece ani mulţi producători au realizat reţele neuronale de uz general comerciale dar acestea au avut un succes de piaţă limitat datorită flexibilităţii lor limitate, a suportului tehnic insuficient, precum şi a performanţelor slabe.

Implementările folosind circuite dedicate cresc viteza de calcul a reţelelor neuronale cu două ordine de mărime faţă de implementările folosind procesoare de uz general.

Principalele metode de implementare hardware electronice sunt implementările analogice, digitale şi mixte.

2.4.1 Implementarea analogică a reţelelor neuronale artificiale

Implementarea analogică este caracterizată de viteza mare de procesare dar şi de precizia scăzută. Una din problemele majore ale implementărilor analogice este stocarea nevolatilă (sau pe termen lung) a ponderilor sinaptice.

Circuitele electronice analogice au câteva caracteristici care pot fi utilizate direct pentru implementarea reţelelor neuronale. Astfel amplificatoarele operaţionale pot implementa uşor funcţii neuronale precum integrarea şi funcţia sigmoid. Aceste calcule, altfel foarte complexe, sunt efectuate prin procese fizice cum ar fi însumarea curenţilor sau a sarcinilor. Reţele neuronale simple cum ar fi de exemplu memoriile asociative (fără antrenare) pot fi implementate într-un chip care poate conţine 1000 neuroni cu câte 1000 intrări fiecare, având o viteză de calcul de 100 GCPS.

Dezavantajele implementărilor analogice sunt sensibilitatea la zgomote şi variaţiile parametrilor ceea ce duce la o precizie limitată a calculelor. Circuite construite după acelaşi proiect nu vor funcţiona la fel, dar acelaşi lucru se poate spune şi despre creierul biologic. Pe lângă dificultăţile legate de proiectarea circuitelor analogice problema reprezentării ponderilor adaptabile limitează aplicabilitatea circuitelor analogice. Ponderile pot fi reprezentate prin rezistoare, dar acestea nu sunt adaptabile după producerea circuitului. Circuitele cu ponderi fixe pot fi utilizate numai în faza de propagare. Tehnicile care permit implementarea ponderilor variabile utilizează capacităţile, tranzistoarele cu poartă flotantă, dispozitivele cu cuplare prin sarcină (CCD), etc. Principalele probleme legate de aceste tehnici de

implementare apar datorită variațiilor parametrilor procesului, volatilitatea şi slaba compatibilitate cu procesul tehnologic VLSI standard. Setul de ponderi pentru aceste circuite antrenabile se obțin prin antrenare pe un calculator şi încărcare în chip.

Pentru a beneficia de viteza implementărilor analogice şi de proprietatea de adaptabilitate a rețelelor neuronale ar fi necesară implementarea procesului de antrenare pe chip, dar implementarea majorității regulilor de învățare în circuitele VLSI analogice este foarte dificilă.

Cu toate că circuitele analogice nu vor atinge niciodată flexibilitatea ce poate fi obținută cu circuitele digitale, viteza lor şi spațiul mai mic ocupat pe chip le fac atractive pentru implementarea rețelelor neuronale. Un alt avantaj îl reprezintă posibilitatea mai directă de interfațare cu lumea reala, preponderent analogică, în timp ce implementările digitale vor necesita întotdeauna convertoare analog digitale şi digital analogice rapide pentru a prelua informații din lumea reală, respectiv pentru a trimite date înapoi.

În continuare sunt prezentate câteva neurochip-uri pur analogice adaptive [106].

- **ETANN,** Electrically Trainable Analog Neural Network (Intel Inc., CA, USA), a fost primul neurochip disponibil comercial care permite încărcarea ponderilor în urma antrenării dar nu permite învățarea on-board. Ponderile analogice sunt stocate ca diferența de tensiune între două porți flotante. Chip-ul conține 64 de neuroni şi 1024 sinapse şi are o viteză de 2GCPS.
- **The Mod2 Neurocomputer** (Naval Air Warfare Center Weapons Division, CA) Arhitectura acestui circuit este inspirată de structura sistemului olfactiv, auditiv şi vizual uman. Mod2 este realizat cu circuite ETANN într-o arhitectură extensibilă care permite implementarea a diverse tipuri de rețele neuronale. O implementare inițială constă în 12 chip-uri ETTAN cu o viteză de calcul de 1,2 GCPS fiecare.
- **Fully analog chip** (Kansai University, Japan). Circuitul prototip constă într-un unitate neuronală, o unitate de antrenare şi o unitate de control.

2.4.2 Implementarea digitală a rețelelor neuronale artificiale

Printre avantajele implementărilor digitale se pot enumera:

- sensibilitatea scăzută la perturbații electrice
- precizia ridicată
- uşurința proiectării
- repetabilitatea
- stocarea facilă a ponderilor
- raport performanță cost mai bun față de implementările analogice

Dezavantajele acestei metode de implementare sunt reprezentate de:

- aria mare de siliciu ocupata
- consumul ridicat de putere

În continuare se prezintă câteva implementări care folosesc circuite neuronale digitale [39], [40 - 43], [106], [210], [229].

- **SYNAPSE-1, SY**nthesis of **N**eural **A**lgorithms on a **P**arallel **S**ystolic **E**ngine este un neurocomputer construit de Siemens AG, care constă dintr-o arie sistolică de procesoare, aranjate pe două rânduri şi 4 coloane. Aceste procesoare sunt circuite neuronale speciale MA16 care implementează direct funcții comune rețelelor neuronale, cum ar fi multiplicarea matricială sau determinarea maximului. Toate

celelalte calcule care nu sunt critice se realizează folosind circuite comerciale. Ponderile sunt stocate în afara chip-ului în memorii locale. Fiecare chip conţine o arie sistolică de 4 module de procesare, conţinând multiplicatoare de 16x16 biţi. Configuraţia standard pentru SYNAPSE-1 constă în 8 neurochip-uri MA16, două procesoare CISC MC68040, şi o memorie DRAM de 128 Mocteţi. Prototipul atinge viteza de 5,1GCPS (şi 33 MCUPS), la o frecvenţă de ceas de 40 MHz.

- **CNAPS,** Connected Network of Adaptive Processors, (Adaptive Solutions, Inc., USA) este o arie de procesoare SIMD (Single Instruction, Multiple Data) cu 64 elemente de procesare pe chip, cum este prezentat în figura 2.7 [39].

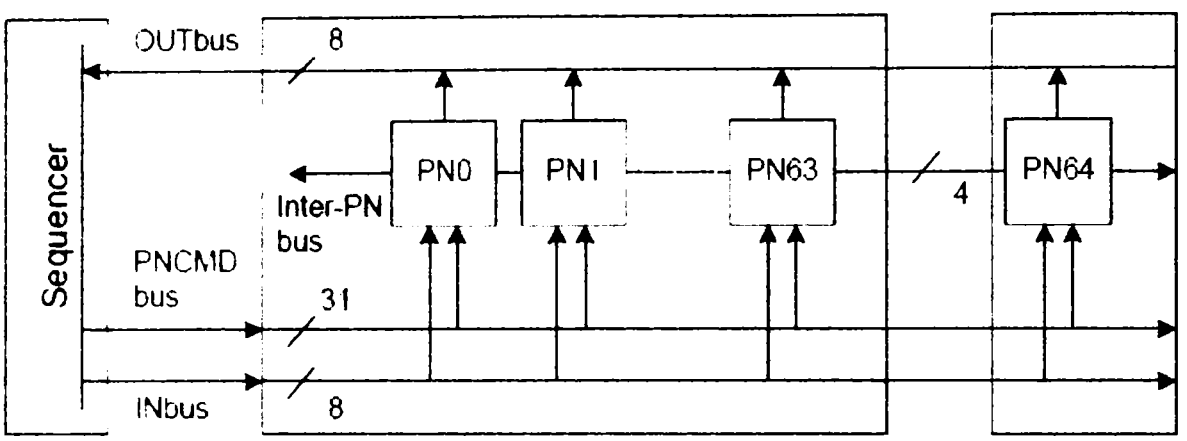


Figura 2.7 Arhitectura CNAPS

Cele 64 elemente de procesare sunt circuite N6400 (figura 2.8), special proiectate, care se aseamănă cu circuitele DSP cu multiplicatoare de precizie limitată. Circuitele N6400 sunt conectate în cascadă şi comunică prin magistrale comune.

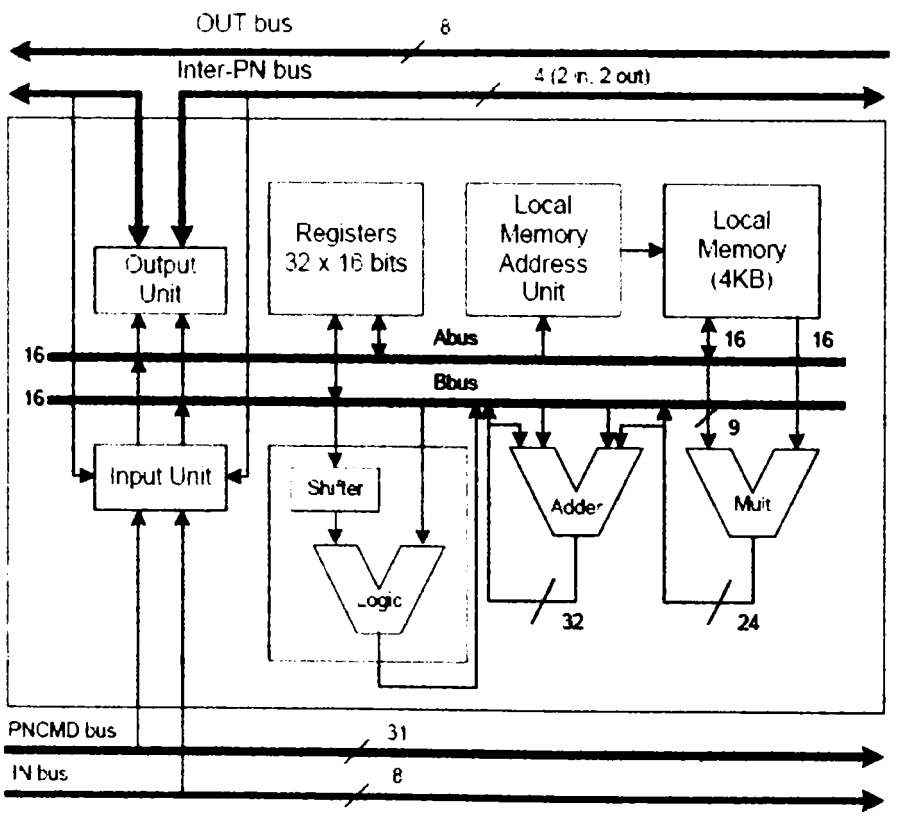


Figura 2.8 Arhitectura elementului de procesare din CNAPS

Chip-urile dispun de o memorie SRAM suficientă pentru a stoca 128K ponderi pe 16 biţi. Este prevăzut cu posibilitatea învăţării pe chip. Fiecare N6400 conţine 64 elemente de procesare şi 256 Kocteţi memorie de ponderi cu o rezoluţie de (1-16 biţi). Un singur chip poate atinge 1,6 GCPS (256 MCUPS, backpropagation) pentru ponderi pe 8 sau 16 biţi şi 12,8 GCPS pentru ponderi pe 1 bit.
Sistemul standard constă în patru chip-uri pe o placă controlate de un chip comun.

- **SNAP,** SIMD Neurocomputer Array Processor, (HNC, CA, USA), se bazează pe chip-urile HNC 100 NAP (Neural Array Processor), fiecare constând într-o arie uni-dimensională de 4 celule aritmetice. Organizarea elementelor de procesare este similară cu cea a sistemului CNAPS. O placă SNAP conţine patru chip-uri HNC 100 NAP. O configuraţie standard de 2 plăci SNAP boards (32 procesoare) atinge 500 MCPS şi 128 MCUPS (backpropagation).

- **CNS Connectionist Supercomputer** (ICSI, Berkeley, CA, USA), compus din 1024 noduri de procesare, care dau capacitate de calcul de 2 TeraOps (integer operations per second) şi 32 Goctet de RAM. Un capăt al reţelei este rezervat pentru cuplarea dispozitivelor de Intrare/Iesire, permiţând o lăţime de bandă de până la 8 Gocteţi/s. Performanţa este de: 1011 CPS şi 310 CUPS, care este de aproximativ 10.000 mai mare decât a unui staţii RISC.
- **Hitachi WSI** (Hitachi Central Research Laboratory, Kokubinji Tokyo, Japan). Prototipul a fost lansat în 1990 fiind cea mai rapidă implementare a algoritmului backpropagation. Performanţe: 1152 noduri, 2,3 GCUPS.
- **LNeuro 1.0,** Learning Neurochip, (Neuromimetic Chip, Philips, Paris, France), este un circuit neuronal digital de uz general care implenetează o reţea cu 32 neuroni de intrare şi 16 neuroni de ieşire. Performanţe: 16 circuite LNeuro pe 4 plăci dedicate ajung la 19 MCPS, 4,2 MCUPS.
- **Mantra I** (Swiss Federal Institute of Technology) Performanţe: 400 MCPS, 133 MCUPS (backpropagation).
- **ZISC036** (IBM Micro electronics, France). Neurochip-ul Zero Instruction Set Computer (ZISC036) este primul dintr-o serie de implementări integrate a reţelelor neuronale bazate pe funcţii radiale. Circuitul a fost proiectat pentru recunoaştere şi clasificare în timp real. Are în componenţă 36 de neuroni cu memorie şi regiştrii pentru stocarea datelor. Circuitele se pot lega în cascadă permiţând implementarea de reţele de dimensiuni mari. Comunicaţia între circuite se face prin magistrale de comunicaţie. Circuitele ZISC pot fi uşor interfaţate cu alte resurse hardware cum ar fi PC-urile.

Alte implementări, amintite în literatură, care folosesc circuite neuronale digitale sunt:

- **MY-NEUPOWER** (Hitachi Microcomputer System, Ltd., Tokyo, Japan)
- **Biologically-Inspired Emulator** (Duisberg, Germany)
- **INPG Architecture** (Institute Nationale Polytechnique de Grenoble, France)
- **BACHUS** (Darmstadt University of Technology, Univ. of Düsseldorf, Germany)
- **UTAK1,** Unit for Trained Adaptive Knowledge, (Catalunya University, Spain)
- **UCL** (Esprit Pygmalion/Galatea Project at University College London, UK)

2.4.3 Implementarea hibridă a rețelelor neuronale artificiale

Atât implementarea analogică cât și cea digitală oferă avantaje unice, așa cum s-au prezentat în paragrafele anterioare, dar de asemenea au și unele dezavantaje. Implementarea hibridă combină avantajele implementărilor analogice și a celor digitale.

În această categorie intră implementările bazate pe procesarea trenurilor de impulsuri (pulse stream) inspirate de comportamentul neuronilor reali. Ele reprezintă o abordare promițătoare a implementărilor hibride. În tehnica procesării impulsurilor starea analogică a neuronului este reprezentată ca o secvență de impulsuri. Aceasta oferă o serie de avantaje cu privire la consumul de putere, viteza de calcul și de propagare. Există mai multe tehnici de modulare a impulsurilor:

- Modularea în amplitudine a impulsului, în care amplitudinea unui impuls este modulată reflectând starea neuronului.
- Modularea în frecvență a impulsului. Valoarea analogică este reprezentată de frecvența unui impuls. Suma a două semnale se poate face simplu folosind un SAU logic, iar produsul se poate face cu un ȘI logic. Astfel operațiile elementare pot fi implementate cu porți logice simple.
- Modularea în durată a impulsului.
- Modularea în fază a impulsului.

Pe scurt există două avantaje ale procesării trenurilor de impulsuri:

a. Circuitele analogice (de exemplu multiplicatoarele) ocupă mult mai puțin spațiu pe chip decât echivalentele lor digitale.
b. Valorile analogice codate în format digital sunt robuste și ușor de transmis.

Printre implementările hibride cele mai cunoscute sunt [106]:

- **ANNA,** Analog Neural Network Arithmetic and logic unit, (AT&T Bell Labs, NJ, USA), este un chip care poate fi folosit pentru implementarea unei game largi de arhitecturi dar este optimizat pentru rețele neuronale conectate local și cu ponderi partajate. Ponderile sunt învățate în afara chip-ului, cuantificate la rezoluția chip-ului, și apoi încărcate în memoria ponderilor. Acestea sunt reprezentate de tensiuni. Interfața cu chip-ul este pur digitală cu două convertoare D/A care fac conversia valorii ponderilor reprezentate pe 6 biți în tensiunea corespunzătoare. Placa conține un procesor DSP în virgulă mobilă DSP-32C pentru faza de învățare și calculul stratului de ieșire a rețelei backpropagation. Chip-ul ANNA are 4096 sinapse și 8 neuroni. Performanțe: 5000 MCPS (valoare de vârf), 1000 la 2000 MCPS (în medie).
- **Epsilon,** Edinburgh Pulse Stream Implementation of a Learning Oriented Network, (Dept. of Electrical Engineering, Edinburgh University, UK), este un bloc constructiv generic care constă în 30 de noduri și 3600 ponderi sinaptice. Chip-ul are un singur strat dar poate fi legat în cascadă pentru a forma o rețea mai mare. Performanțe: 360 MCPS

2.4.4 Comparație între implementările ASIC și FPGA

2.4.4.1 Circuite integrate specifice aplicației (ASIC)

Circuitele ASIC sunt dispozitive dedicate, proiectate special pentru o aplicație și pot fi de tip analogic, digital sau pulse stream. Ele includ o varietate de soluții de la rețele neuronale de mici dimensiuni până la dimensiuni medii (de obicei mai puțin de 10.000 de sinapse, uneori până la 100.000). Avantajele lor sunt vitezele mari de procesare, putere consumată

scăzută şi un raport scăzut preţ/performanţă în cazul producţiei în serie mare. Cu toate că în ASIC se pot implementa orice funcţii, odată fabricate ele nu mai pot fi modificate, lipsa de flexibilitate fiind unul din dezavantajelor lor majore.

Faza de dezvoltare a unui circuit nou este cea mai costisitoare parte a unui proiect, cu un necesar de 1 până la 20 oameni-lună şi un cost pentru măşti, de la mai multe sute de mii de dolari (în tehnologia de 180 nm) până la 1,5 milioane dolari (în tehnologia de 90 nm). Deci circuitele ASIC se pretează doar la producţii în serie mare sau când sunt necesare performanţe ridicate. Aceasta este un dezavantaj care limitează utilizarea lor pentru implementarea RNA.

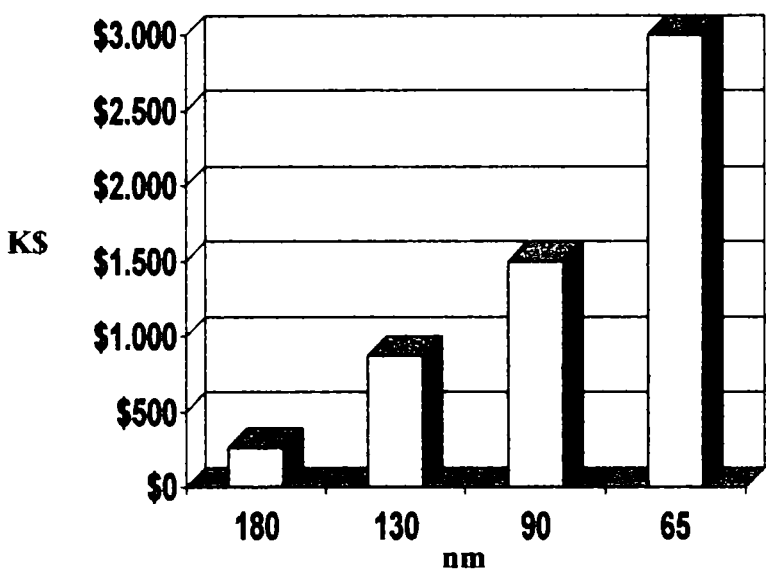


Figura 2.9 Costul măştilor pentru diverse tehnologii de fabricaţie a circuitelor ASIC

2.4.4.2 Circuite logice programabile

Ariile de porţi programabile de către utilizator (FPGA) sunt dispozitive ce pot fi configurate pentru implementarea aproape oricărui circuit digital deci şi a reţelelor neuronale. Ele prezintă avantajul unei foarte mari flexibilităţi deoarece pot fi reprogramate uşor. FPGA-urile sunt ideale pentru aplicaţiile prototip, în care trebuie să se producă versiuni numeroase ale proiectului, până la obţinerea variantei finale, optime. Acestea sunt, de asemenea, ideale pentru lansarea rapidă a produselor şi pentru aplicaţiile de volum redus. Tabelul 2.1 prezintă o comparaţie din punct de vedere a costurilor între implementările FPGA şi ASIC.

Tabelul 2.1 Comparaţie costuri FPGA – ASIC

	FPGA	ASIC.
Eşantioane prototip	50-1000 $	>100.000 $
Software PC	100-5000 $	>25.000 $
Software staţii de lucru	10.000 $	>50.000 $

Circuitele FPGA permit o reconfigurare software rapidă a circuitului hardware pentru a adapta hardware-ul la o gamă largă de aplicaţii. Pe baza legii lui Moore, FPGA-urile cresc în densitate şi viteză în aceeaşi măsură ca şi procesoarele. Cu toate că FPGA-urile plătesc un preţ în ceea ce priveşte viteza şi densitatea (aria) datorită flexibilităţii lor, pentru anumite aplicaţii cu un număr suficient de operaţii paralele ele pot oferii un raport preţ/performanţă mai bun şi putere disipată mai mică decât procesoarele actuale sau DSP-urile.

Datorită arhitecturii lor circuitele FPGA sunt potrivite pentru calculul paralel, ele fiind intens folosite în aplicaţiile de procesare a imaginii şi încep să fie folosite în implementarea reţelelor neuronale.

Circuitele cu FPGA pot fi proiectate cu aceleaşi unelte şi limbaje de programare ca şi circuitele ASIC dar reprogramabilitatea lor reduce în mod semnificativ timpul de lansare pe piaţă şi costurile nerecurente (NRE, non-recurrent engineering), care includ: costurile proiectării, simulării, amortizarea programelor de proiectare, fabricarea măştilor şi a tiparelor de testare, documentaţia, marketingul, etc. În acelaşi timp costurile recurente (RE, recurrent engineering) care includ fabricarea, testarea, încapsularea, asamblarea chip-ului, suportul tehnic aferent, etc., sunt mai mari (figura 2.10). De aceea ele sunt preferate pentru producţia în serie mică şi medie.

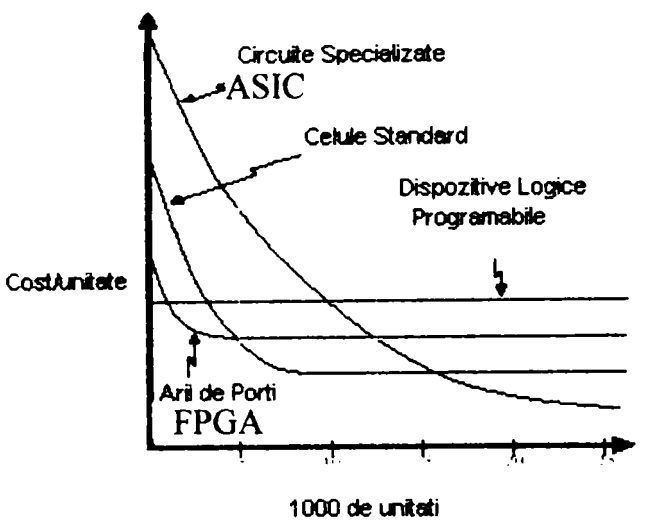


Figura 2.10 Evoluţia costurilor de fabricaţie

Complexitatea circuitelor FPGA este exprimată în numărul de porţi logice echivalente (unde o poartă = 4 tranzistoare) sau în numărul de celule logice care reprezintă un bloc component al unui FPGA. Există circuite de la câteva sute de porţi până la câteva milioane de porţi, astfel ele permit implementarea de reţele neuronale cu un singur neuron până la câteva sute de neuroni cu o capacitate de memorare de câteva sute de Kb pentru memorarea ponderilor. Reţelele neuronale trebuie să fie de dimensiuni mari deci pot uşor depăşi capacitatea unui singur FPGA din ziua de azi. O soluţie posibilă este stocarea parametrilor reţelei într-o memorie externă de tipul SDRAM sau DDRAM, conectată direct la FPGA.

2.4.5 Comparaţie între implementările folosind circuite DSP şi FPGA

Principalele cerinţe pentru implementările hardware ale reţelelor neuronale sunt necesitatea operării în timp real (sau la o viteză cât mai ridicată) şi posibilitatea adaptării la schimbarea datelor de intrare, respectiv a condiţiilor de calcul. Figura 2.11 prezintă o comparaţie între principalele tipuri de implementări din perspectiva celor două cerinţe [178].

Prima cerinţă este cel mai bine satisfăcută de circuitele ASIC proiectate şi realizate special pentru a răspunde aplicaţiei sau domeniului de aplicaţii. Dezvoltarea unor circuite specifice duce însă de obicei la costuri prea mari Cea de a doua cerinţă este de obicei adresată de circuitele programabile de tipul DSP. Cu toată flexibilitatea procesoarelor digitale de semnal, la implementarea unei reţele neuronale arhitectura reţelei trebuie adaptată la resursele procesorului. Avantajul flexibilităţii se plăteşte printr-o reducere a performanţelor şi un consum de putere mai ridicat.

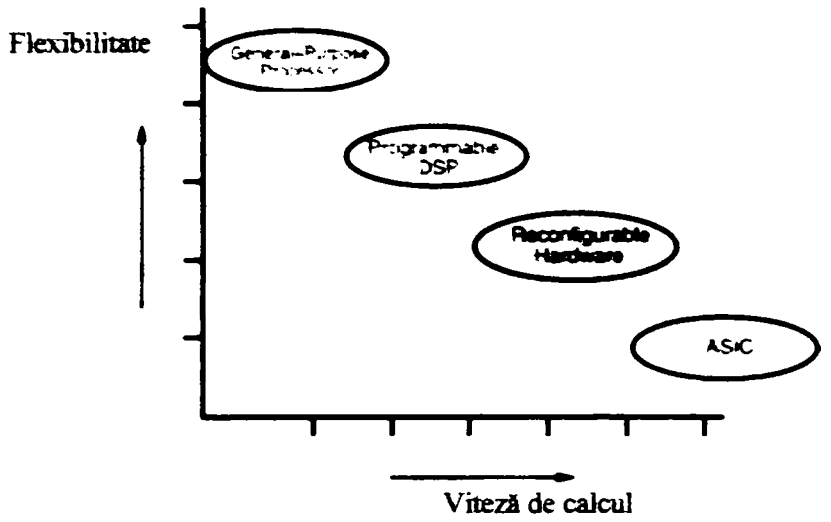


Figura 2.11 Spectrul implementărilor hardware a RNA

După cum se poate vedea din figura 2.11 circuitele reconfigurabile (FPGA) oferă un compromis între performanțele unui circuit specific şi flexibilitatea unui circuit programabil prin soft. Ca şi circuitele ASIC, acestea se remarcă prin abilitatea de a implementa direct circuite specializate în hardware. În plus, la fel ca şi circuitele DSP pot fi uşor modificate în funcție de modificarea condițiilor de operare şi a setului de date.

Procesoarele digitale de semnal convenționale utilizează o arhitectură care prin natura ei este serială. Blocurile de multiplicare – acumulare (MAC) sunt de obicei resurse partajate. Rețelele neuronale folosesc intensiv blocurile MAC deoarece fiecare intrare a neuronului trebuie multiplicată cu un coeficient şi apoi rezultatele însumate. Cu cât rețeaua neuronală conține mai mulți neuroni şi cu cât fiecare neuron are mai multe intrări, creşte şi numărul de blocuri MAC necesare pentru calculul rezultatului. Astfel în cazul unei rețele simple având un singur strat cu 15 neuroni cu câte 15 intrări sunt necesare un număr de 225 operații de multiplicare acumulare pentru calculul ieşirii rețelei pentru un singur vector de date aplicat la intrare. Folosirea unui procesor cu frecvență de lucru ridicată creează dificultăți suplimentare.

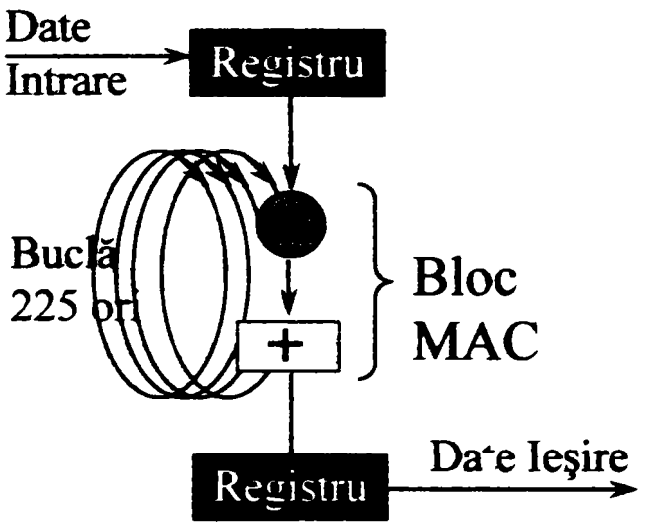


Figura 2.12 Implementarea unei RNA folosind un singur bloc MAC

Implementarea RNA folosind DSP-uri este foarte eficientă doar în cazul unor semnale cu frecvență mică şi medie.

În cazul utilizării unui procesor cu un singur bloc MAC, relaţia dintre rata de eşantionare şi complexitatea reţelei (număr de operaţii MAC necesare) este dată de formula:

$$\frac{Frecventa\ procesor}{Nr.\ operatii\ pe\ esantion} = Rata\ \max.\ de\ esantionare \qquad (2.1)$$

Odată cu creşterea complexităţii reţelei neuronale creşte numărului de operaţii MAC, şi în consecinţă timpul de calcul a răspunsului reţelei pentru fiecare eşantion de date de intrare este mai lung . Din relaţia 2.1 rezultă necesitatea scăderii ratei de eşantionare (Figura 2.13).

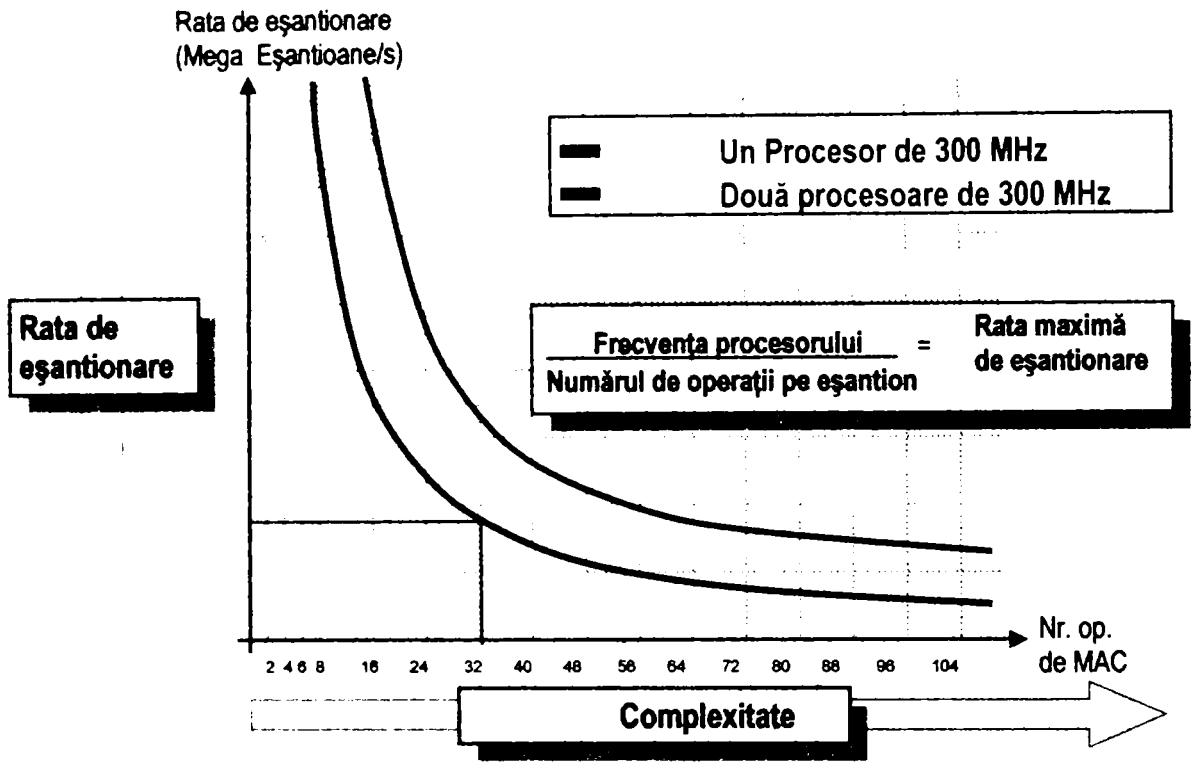


Figura 2.13 Procesarea secvenţială limitează performanţele sistemului

La funcţionarea în timp real, pentru semnale de frecvenţă foarte ridicată este necesară utilizarea tehnicii de procesare în paralel utilizând mai multe blocuri MAC.

La implementarea în circuite FPGA a procesării paralele, un neuron foloseşte un bloc MAC dacă se implementează un paralelism de nod sau câte un bloc de multiplicare pentru fiecare intrare (sinapsă) a neuronului şi apoi un bloc de acumulare, în cazul paralelismului de sinapsă.

În primul caz, RNA din exemplul precedent (prezentat în figura 2.12) furnizează un răspuns corect după 15 unităţi de timp (o unitate de timp = timpul necesar efectuării unei operaţii de multiplicare – acumulare), deoarece cei 15 neuroni lucrează în paralel.

În cel de-al doilea caz răspunsul RNA se obţine după o singură unitate de timp, deoarece toate înmulţirile dintre datele de intrare şi ponderile sinaptice sunt efectuate în paralel.

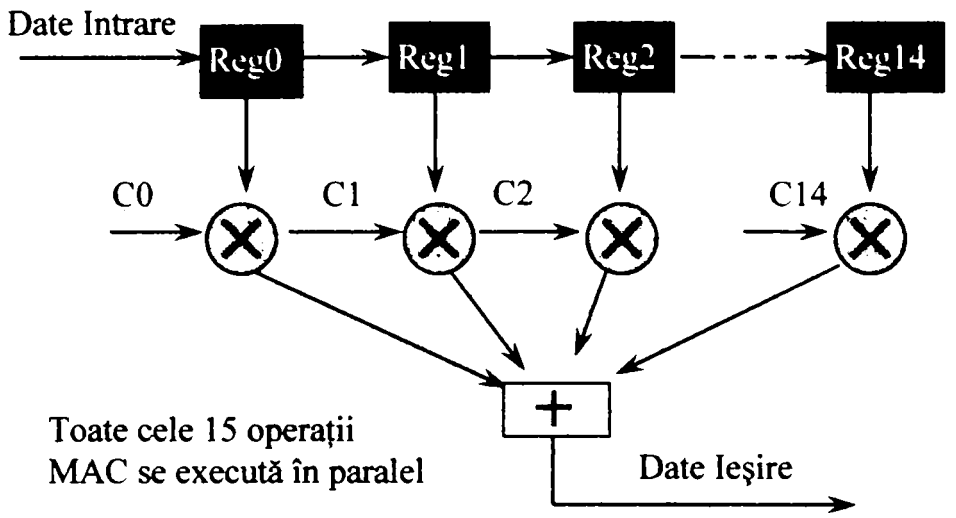


Figura 2.14 Implementarea unui paralelism de sinapsă pentru un neuron

Performanța circuitelor FPGA poate atinge 500 miliarde de MAC pe secundă într-un Virtex II XC2V8000 care este cu mult mai mult decât ceea ce poate fi realizat cu un procesor DSP convențional. O comparație între performanțele celui mai rapid DSP şi a câtorva circuite FPGA este prezentată în tabelul 2.2.

Tabelul 2.2 Comparație între performanțele DSP şi FPGA

Funcția implementată	**Cel mai rapid procesor DSP**	**Virtex-II**	**Virtex-II Pro**	**Spartan-3**
8 x 8 MAC	5,7 miliarde MAC/s	0,5 Tera MAC/s	1 Tera MAC/s	0,27 Tera MAC/s
Filtru FIR - 256 date/coeficienți - 16-biți	11.16 MSPS 720 MHz	180 MSPS 180 MHz	300 MSPS 300 MHz	140 MSPS 140 MHz
FFT Complex - 1024 puncte, date 16-biți	8,5 μs 720 MHz	0,914 μs* 140 MHz	0,853 μs** 150 MHz	0,914 μs*** 140 MHz

* Folosind 96 de multiplicatoare dedicate şi 24 blocuri RAM din Vitex-II (XC2V3000)
** Folosind 96 de multiplicatoare dedicate şi 24 blocuri RAM din Vitex-II Pro (XC2VP30)
*** Folosind 96 de multiplicatoare dedicate şi 24 blocuri RAM din Spartan-3 (XC3S4000)

Procesorul de semnal cu care s-a făcut comparația este TMS320C64x. Acesta are patru blocuri MAC de 16x16 biți care pot lucra în paralel la o frecvență maximă de 720 MHz. Fiecare bloc MAC poate fi folosit ca două blocuri separate de 8x8 astfel încât rezultă:

$$8 \times 720 \text{ MHz} = 5{,}76 \text{ miliarde MAC/s}$$

Circuitul FPGA XC2V8000 are 46.592 celule logice elementare (slice-uri), care permit efectuarea a aproximativ 461 miliarde MAC/s şi de asemenea 168 multiplicatoare dedicate care lucrează la frecvența de ~ 180 Hz (pentru 8x8) astfel încât:

$$168 \times 180 \text{ MHz} = 30 \text{ miliarde MAC/s},$$

deci un total general de 461 + 30 = 491 miliarde MAC/s.

2.5 Implementări hibride hardware software

Deoarece atât implementările hardware cât şi cele software au avantaje şi dezavantaje, o combinaţie potrivită a celor două metode poate duce la crearea de sisteme cu performanţe superioare şi consum redus de putere sau sisteme dedicate care sunt tot mai răspândite.

Avantajele implementărilor hibride rezidă în combinarea dintre flexibilitatea părţii software şi performanţele de viteză şi consum de putere ale părţii hardware.

Dezavantajul implementărilor hibride este reprezentat de complexitatea mai ridicată de proiectare decât în cazul implementărilor software. De aceea majoritatea implementărilor sunt software, în cazul aplicaţiilor în care ele oferă performanţe cum ar fi viteză, consum şi fiabilitate corespunzătoare, chiar dacă alte caracteristici cum sunt costul şi dimensiunile sunt mai puţin performante. Implementările HW/SW devin atractive în cazul în care sunt necesare performanţe ridicate (viteză mai mare, dimensiuni mai mici, cost mai mic, fiabilitate mai mare, consum mai redus). Acesta este şi cazul implementărilor reţelelor neuronale, care necesită un efort de calcul mare.

În prezent există puţine platforme comerciale de uz general pentru sisteme hibride HW/SW, cum ar fi:

- Carmen produs de SISDA [184]
- SMT355 produs de Sundance [254]
- E5 şi A7 produs de Triscend [255]
- Excalibur produs de Altera [256]

de aceea multe aplicaţii necesită dezvoltarea de unui chip sau a unei plăci specifice.

Tehnica de proiectare HW/SW codesign este cea mai potrivită pentru implementarea RNA. Ea permite proiectarea întregului sistem folosind acelaşi limbaj pentru descrierea, simularea, ajustarea şi depanarea părţii HW cât şi acelei SW în mod unitar. Sistemul este descris cu ajutorul unui limbaj potrivit (cum ar fi Simulink, Java, etc.), cu care se poate genera un cod software (pentru un emulator sau DSP) sau descrierea unui sistem digital (un ASIC sau FPGA) sau combinaţia cea mai eficientă a celor două din punct de vedere a costului, dimensiunii, puterii consumare, etc.

În cazul reţelelor neuronale faza de învăţare se pretează a fi implementată software (procesoare de uz general, Matlab), putându-se utiliza reprezentarea în virgulă mobilă ceea ce duce la convergenţa şi acurateţea reţelei. Faza de propagare trebuie implementată hardware datorită cerinţelor de viteză. Utilizând circuite dedicate (FPGA) operaţiile de multiplicare-acumulare pot fi executate în paralel. În software (şi în DSP) acest paralelism este imposibil deoarece execuţia este secvenţială.

Sistemele pe un chip (SoC's, sistems-on-a-chip) reprezintă un caz particular al sistemelor hibride HW/SW, în care ambele părţi, cea HW şi cea SW sunt integrate pe aceeaşi arie de siliciu. Acestea pot conţine şi circuite analogice sau dispozitive de putere.

2.6 Alte aspecte pentru selectarea modului de implementare

În continuare sunt prezentate câteva caracteristici care chiar dacă nu afectează direct performanţele reţelelor implementate, trebuiesc luate în considerare când se optează pentru una sau alta dintre posibilităţile de implementare.

2.6.1 Flexibilitatea

Flexibilitatea este foarte importantă ori de câte ori se doreşte utilizarea unei reţele neuronale în mai multe aplicaţii, unde arhitectura reţelei, numărul de straturi, numărul de neuroni ai stratului ascuns, numărul de intrări sau de ieşiri, sau valoarea ponderilor se poate modifica. Cu cât sunt mai flexibile cu atât acestea se pot adapta mai uşor aplicaţiei.

Implementările software sunt cele mai flexibile dar au rareori performanțe bune. Implementările cu ponderi fixe (cele analogice, mixte şi în circuite ASIC) sunt cele mai puțin flexibile, ele se pot utiliza doar pentru producție de serie de mare, caz în care costurile nerecurente devin neglijabile. Implementările complet paralele cu ponderi programabile (sau antrenabile) sunt mult mai flexibile dar ridică problema memorării ponderilor în cazul implementărilor hardware analogice. În prezent circuitele logice programabile de tipul FPGA (în combinație cu tehnica HW/SW codesign) oferă cel mai bun compromis între flexibilitate, cost şi timp de lansare pe piață, cu excepția producției în serii mari sau cerințe foarte particulare. Tehnica HW/SW codesign îmbunătățeşte şi flexibilitatea implementărilor digitale şi hibride prin reducerea semnificativă a timpului de (re)proiectare.

2.6.2 Integrarea cu alte sisteme

În majoritatea aplicațiilor rețeaua neuronală este doar o parte din sistem, deci ea trebuie să fie integrată împreună cu alte subsisteme foarte diferite (filtrare, preprocesare, interfețe utilizator, circuite de evaluare a performanțelor, etc). Integrarea este dificilă în cazul implementărilor analogice şi hibride (pulse stream), mai simplă în digital şi foarte simplă în software. Tehnica HW/SW codesign simplifică foarte mult integrarea în cazul implementărilor hibride.

2.6.3 Timp de lansare pe piață

Acest aspect este o preocupare importantă pentru aplicațiile industriale, dar este mai puțin importantă pentru proiectele academice. Acest timp este alcătuit din două componente:

1. Faza de proiectare, care depinde de efortul economic depus;
2. Faza de fabricație, care depinde în cea mai mare măsură de tehnologia de fabricație aleasă.

Circuitele ASIC analogice şi pulse stream au cel mai lung timp de lansare pe piață, deoarece trebuiesc proiectate şi simulate la nivel de tranzistor şi apoi testate. Faza de proiectare necesită de obicei între 6 şi 24 oameni-lună, astfel timpul total nu poate fi mai mic de câteva luni, o valoare mai apropiată de realitate fiind cuprinsă între 6 până la 24 de luni.

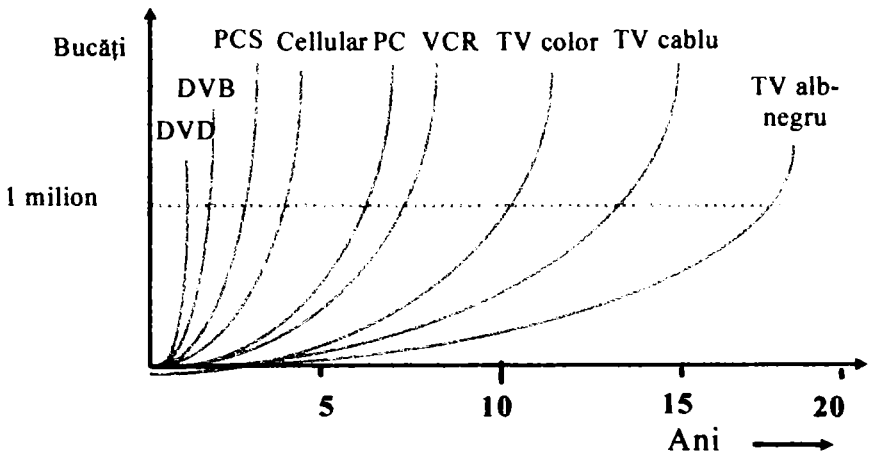


Figura 2.15 Dinamica schimbărilor în industria electronică

Circuitele digitale sunt mai simplu de proiectat şi testat deoarece pot fi proiectate la nivel de poartă logică sau nivel funcțional. Faza de proiectare durează între 0,5 până la 2 oameni-lună pentru FPGA sau software şi respectiv între 1 şi 6 luni pentru ASIC, ajungând astfel la un timp de lansare pe piață de 0,5 până la 2 luni, pentru FPGA sau SW, respectiv între 1 şi 6 luni pentru ASIC digitale.

În industria electronică a bunurilor de larg consum există o tendinţă clară de reducere a timpului până la care se ajunge la producţia în masă dar şi a timpului de menţinere pe piaţă a unui produs (figura 2.15).

Reducerea timpului de lansare pe piaţă duce la mărirea profitului aşa cum se prezintă în figura 2.16. Prima companie care lansează pe piaţă un produs nou ocupă cu aproximativ 40% mai mult decât următoarea companie.

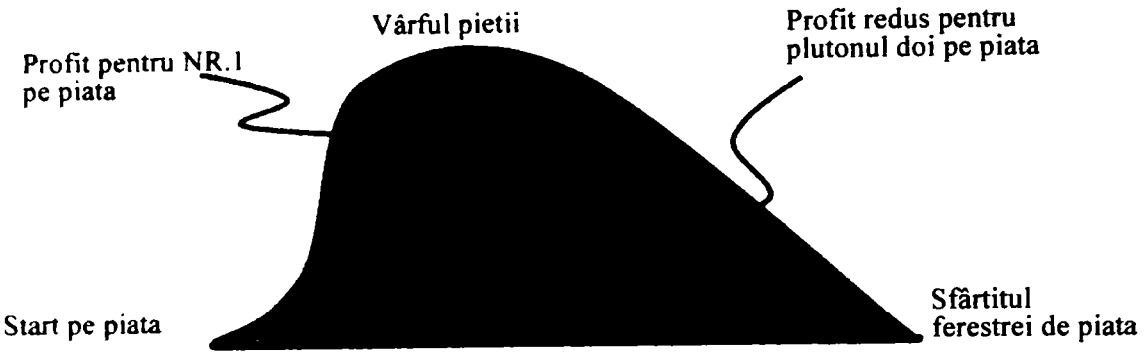


Figura 2.16 Importanţa scurtării timpului de lansare pe piaţă a unui produs

Utilizarea unei tehnici de codesign HW/SW asociată cu circuitele FPGA poate micşora şi mai mult timpul de lansare pe piaţă până la o săptămână. Prezenta teză propune o metodă de implementare folosind o tehnică de codesign HW/SW folosind un toolbox pentru Simulink creat de firma Xilinx pentru procesarea digitală a semnalelor. Aceasta permite integrarea facilă a reţelelor neuronale împreună cu alte blocuri funcţionale hardware sau din bibliotecile standard Simulink. În prezent acesta este unul din foarte puţinele medii de proiectare HW/SW integrat.

Implementările software au cel mai scurt timp de lansare pe piaţă, datorită marii flexibilităţi şi interfeţei prietenoase a limbajelor de programare şi a compilatoarelor. Acesta este motivul principal pentru care ele sunt mai prezente pe piaţă decât implementările hardware.

2.6.4 Toleranţa la erori

Orice defect într-un procesor secvenţial duce la oprirea calculului şi blocarea întregii reţele, în timp ce anumite defecte într-o reţea complet paralelă afectează calculul contribuţiei unei singure sinapse (eventual a unui neuron), care de obicei are un efect neglijabil asupra performanţelor întregii reţele. Reţelele paralele sunt în mod intrinsec mult mai tolerante la erori decât reţelele secvenţiale în timp ce reţelele parţial paralele se situează între cele două.

2.6.5 Testabilitatea

Testarea este o problemă majoră pentru orice implementare hardware. Există câteva tipuri de verificări care trebuiesc făcute, cu impact diferit asupra costurilor:

- *testarea proiectului*, se referă la verificarea corectitudinii proiectării circuitelor şi se face prin intermediul simulării şi duce la descoperirea majorităţii defectelor de proiectare. Proiectele academice sunt mai puţin testate în timp ce proiectele industriale sunt testate cu mare atenţie. Această testare este de obicei costisitoare, având un impact semnificativ asupra costurilor nerecurente, cu excepţia cazurilor când se utilizează pentru implementare procesoare comerciale sau FPGA-uri.
- *testarea chip-ului,* se referă la verificarea faptului că chip-ul produs este fără defecte sau, când pe chip există unităţi de procesare (PEs, processing elements) suplimentare, localizarea şi substituirea elementelor de procesare defecte. Testarea chip-ului se face de către producător, pentru dispozitivele industriale sau comerciale, respectiv de

proiectant în cazul dispozitivelor academice. Testarea chip-ului este foarte dificilă în cazul circuitelor ASIC analogice şi pulse stream. În cazul circuitelor FPGA sau a altor procesoare comerciale testarea este făcută deja de către fabricant. Testarea chip-ului afectează costurile recurente.

- *autotestare,* permite ca chip-ul să se autotesteze în timpul funcţionării normale. Este o facilitate întâlnită numai în sistemele foarte complexe care afectează dimensiunile chip-ului şi deci costurile recurente.
- *testarea sistemului,* este verificarea funcţionării corecte a întregului sistem compus din chip, plăci, soft, interfeţe utilizator, senzori, actuatori, etc.

2.7 Concluzii

Acest capitol a prezentat şi comparat posibilităţile existente pentru implementarea hardware a reţelelor neuronale artificiale. Au fost prezentate avantajele şi dezavantajele tehnicilor de implementare analogice, digitale şi hibride, aşa cum sunt ele sumarizate în tabelul 2.3 [132].

Tabelul 2.3 Comparaţie între performanţele diverselor implementări ale RNA

	Viteză	Putere	Dim. chip	Repet-abilitate	Precizie	Timp lansare	Cost nerec.	Cost rec.	Cant.	Integrabilitate	Flexibilitate
Analogic, pond. volatile	Foarte mare	Foarte mică	Mic-med.	Mică	Mică	Foarte mare	Foarte mari	Mici-med.	Mare	Mică-med.	Mică
Analogic, ponderi fixe	Foarte mare	Foarte mică	Mic-med.	Mică	Mică	Foarte mare	Foarte mari	Med-mari	Mare	Mică-med.	Foarte mică
Pulse stream, pond. volatile	Mare	Mică	Mică	Mică-med.	Mică-med.	Foarte mare	Mari	Mici-med.	Mare	Mică	Mică
Pulse stream, ponderi fixe	Mare	Mică	Mică	Mică-med.	Mică-med.	Foarte mare	Mari	Med-mari	Mare	Mică	Foarte mică
Digit. ASIC, off-chip	Mică-med	Med-mare	Mic-med. *	Foarte mare	Oarecare	Mare	Mari	Mici	Mare	Medie	Mică-med.
Digit. ASIC, on-chip	Mare	Med-mare	Mare	Foarte mare	Oarecare	Mare	Mari	Mici	Mare	Medie	Mică-med.
Digit. FPGA, off-chip	Mică-med	Mare	Mic-med. *	Foarte mare	Oarecare	Mic	Mici	Medii	Med-mare	Medie-mare	Mare
Digit. FPGA, on-chip	Mare	Mare	Mare	Foarte mare	Oarecare	Mic	Mici	Medii	Med-mare	Medie-mare	Mare
SW în DSP sau PC	Mică-med	Foarte mare	Mic-med. *	Foarte mare	8,16 sau 32 biţi	Foarte mic	Foarte mici	Mici-med.	Orice	Foarte Mare	Foarte mare

* Includ memorie externă pentru ponderi.

Următoarele atribute permit compararea caracteristicilor implementărilor hardware a reţelelor neuronale artificiale.

1. **Tipul dispozitivului**

RNA pot fi implementate hardware folosind un:

a. neuro-chip sau neurocomputer
b. chip standard

2. Proprietăţile neuronilor

Atributele legate de proprietăţile neuronilor implementaţi sunt:

a. numărul neuronilor
b. locul memorării stării neuronului: pe chip sau în afara chip-ului
c. modul de memorare a stării neuronului: analogic/digital
d. precizie (număr de biţi)

Numărul de neuroni se specifică direct. Starea neuronului poate fi memorată pe chip sau în afara lui. În cazul memorării pe chip aceasta poate fi făcută în formă analogică sau digitală. Sub formă analogică există posibilitatea reprezentării sub forma unei tensiuni sau a unei frecvenţe. Precizia este numărul de biţi care sunt utilizaţi pentru reprezentarea valorii de ieşire a neuronului.

3. Ponderile

Caracteristicile legate de ponderi sunt:

a. memorarea ponderilor pe chip sau în afara chip-ului
b. numărul de sinapse
c. tipul ponderilor: analogic/digital

Memorarea ponderilor se poate face pe chip sau în afara lui. În cazul memorării pe chip-aceasta poate fi în format digital sau analogic. În formă analogică ponderile sunt stocate sub formă de tensiune sau sarcini electrice.

4. Caracteristicile de activare

Caracteristicile legate de activare, care necesită calculul sumei de produse sunt:

a. calcul analogic sau digital
b. ieşirea blocului de activare: probabilistic sau deterministic

Calculul activării (ieşirii nete) se face totdeauna pe chip în formă analogică sau digitală. Dacă rezultatul sumei de produse se aplică direct funcţiei de activare atunci ieşirea este deterministică, iar dacă ieşirii nete sau semnalului de intrare i se adaugă un factor de zgomot ieşirea devine probabilistică.

5. Caracteristicile funcţiei de transfer

Atributele care determină caracteristicile funcţiei de activare sunt:

a. implementarea funcţiei de transfer: pe chip sau în afara lui
b. tipul funcţiei de transfer: analogic sau digital
c. evaluarea funcţiei de transfer: prag, tabele de memorii sau calcul

Funcţia de transfer poate fi implementată în afara chip-ului sau pe chip sub formă analogică sau digitală. Implementarea în formă digitală a funcţia de transfer presupune utilizarea unei tabele de memorii (look-up table), o comparare cu un prag sau un calcul direct. În cazul implementării analogice se pot utiliza circuite electronice cum ar fi amplificatoarele operaţionale.

6. Modul de învăţare

În funcţie de modul cum are loc procesul de învăţare aceasta poate fi:

a. învăţare pe chip sau în afara lui
b. de sine stătătoare sau pe un calculator gazdă

Ponderile pot fi actualizate în urma unui proces de învăţare. Dacă este posibilă învăţarea pe chip, actualizarea ponderilor se poate face în mod automat pe chip. În cazul învăţării off-chip calculul ponderilor se face pe un calculator gazdă şi apoi se încarcă pe chip.

7. Viteza

Există două caracteristici de viteză:

a. viteza de învăţare
b. viteza de propagare

Viteza de învăţare se referă la calculul şi actualizarea ponderilor în faza de învăţare. Se măsoară în număr de conexiuni actualizate pe secundă (CUPS).

Viteza de propagare se referă la calculul produsului dintre ponderi şi semnalele de intrare şi calculul funcţiei de transfer. Ea indică cât de potrivită este arhitectura pentru algoritmul folosit şi se măsoară în conexiuni pe secundă (CPS).

8. Numărul de intrări şi de ieşiri

Numărul de intrări şi de ieşiri poate fi fix sau în cazul circuitelor reconfigurabile variabil.

Pe baza celor prezentate mai sus tabelul 2.4 [15] prezintă o serie de dispozitive comerciale. S-au folosit următoarele notaţii:

I: intrare
O: ieşire
A: blocul funcţiei de activare
W: blocul ponderilor
S: blocul de stare a neuronului (Neuron state block)
T: funcţia de transfer
L: instruire
a: analogic
d: digital
o: pe chip
f: în afara chip-ului

Tabelul 2.4 Clasificarea implementărilor hardware a RNA

Clasa reţelei neuronale hardware	Caracteristici
Neurochip-uri digitale de uz special (Special Purpose Digital Neuro-chips)	
TiNMANN – Kohonen SOFM	Id/Ad/Wdo/Sdo/Tdo/Lo
Hughes M1718 – Mulilayer Perceptron	Id/Ad/Wdo/Sdo/Tdo/Lf
EPLI France chip – Hopfield	Id/Ad/Wdo/Sdo/Tdo/Lo
Neurochip-uri analogice de uz special (Special Purpose Analog Neuro-Chip)	
Synaptics 110XX Object Recognized chip – Silicon Retina	Ia/Aa/Wao/Sao/Tao/Lf
Jolus Hopkins University chip –Silicon Cortex	Ia/Aa/Wao/Sao/Tao/Lf
Caholic University of Louvain processor chip – LVQ, RBF	Ia/Aa/Wao/Sao/Tao/Lo
Neurochip-uri digital-analogice de uz special (Special Purpose D-A Neuro-chips)	
BELLCORE CLNN 32 – Bolzmann Machine	Ia/Aa/Wdo/Sao/Tao/Lo
Kakadu-Multilayer Perceptron	Ia/Aa/Wdo/Saf/Tao/Lf
BELL Labs NET 32K – Template Matching	Id/Aa/Wso/Sdo/Tdo/Lf
Fujitsu chip	Ia/Aa/Wdo/Sao/Tao/Lf
Jet Propulsion Laboratory chip – Hopfield	Ia/Aa/Wdo/Sao/Tao/Lf

Clasa reţelei neuronale hardware	Caracteristici
National Microelectronics Center (CNM) chip – ART1	Id/Aa/Wao/Sdo/Tao/Lo
Neurochip-uri digitale de uz general (General Purpose Digital Neuro-chips)	
Ni1000 (Nestor)	Id/Ad/Wdo/Sdo/Tdo/Lo
Hirai's Chip	Id/Ad/Wdo/Sdo/Tdo/Lo
Digital chip (HITACHI	Id/Ad/Wdo/Sdo/Tdo/Lo
MA 16 (Siements	Id/Ad/Wdf/Sdo/Tdf/Lf
GENES IV (EPFL)	Id/Ad/Wdf/Sdo/Tdf/Lo
ZISC 036 (IBM)	Id/Ad/Wdo/Sdo/Tdo/Lo
MT19003 (Micro Circuit Engineering)	Id/Ad/Wdf/Sdf/Tdf/Lf
MD – 1220 (Micro Devices)	Id/Ad/Wdo/Sdo/Tdo/Lf
NLX – 420 (Neural Logix)	Id/Ad/Wdf/Sdo/Tdo/Lf
OBL (Oxford Computer)	Id/Ad/Wdo/Sdo/Tdo/Lo
Pram – 256 (UCLi)	Id/Ad/Wdf/Sdo/Tdo/Lo
L-Neuro 1 (Philips)	Id/Ad/Wdo/Sdo/Tdf/Lf
100-NAP (HNC)	Id/Ad/Wdf/Sdo/Tdo/Lo
N64000 (Inova)	Id/Ad/Wdo/Sdo/Tdo/Lo
MM32K (Current Technology)	Id/Ad/Wdo/Sdo/Tdo/Lo
RN-200 (RICHO)	Id/Ad/Wdo/Sdo/Tdo/Lo
Neurochip-uri digital-analogice de uz general (General Purpose D–A Neuro-chips)	
EPSILON (University of Edinburgh)	Ia/Aa/Wao/Sdo/Tao/Lf
ANNA (AT & T Bell Labs)	Id/Aa/Wao/Sdo/Tao/Lf
ETANN (Intel)	Ia/Aa/Wdo/Sao/Tao/Lf
Neuro Classifier (Mesa Research)	Ia/Aa/Wdo/Sao/Tao/Lf
Neuroprocessor CCD chip (MIT)	Ia/Aa/Wdo/Sao/Tdo/Lf
Neurocalculatoare digitale de uz general (Neurocomputers General Purpose Digital)	
CNAPs (Adaptive Solutions)	Id/Ad/Wdo/Sdo/Tdo/Lo
BSP400 (Leiden University)	Id/Ad/Wdo/Sdo/Tdo/Lo
WSI (Hitachi)	Id/Ad/Wdo/Sdo/Tdf/Lo
MARK (TRW)	Id/Ad/Wdo/Sdo/Tdo/Lo
SNAP (HNC)	Id/Ad/Wdf/Sdo/Tdo/Lo
MANTRA (EPFL)	Id/Ad/Wdf/Sdo/Tdf/Lo
SYNAPSE (Siemens)	Id/Ad/Wdf/Sdo/Tdf/Lo

Implementările analogice şi pulse stream sunt preferate pentru o producţie în serie mare, pentru un consum redus de putere, rată de eşantionare mare, dimensiuni mici, de preferinţă cu ponderi fixe şi o precizie mai mică.

Implementările digitale cu memorie pe chip sunt preferate pentru o precizie mai mare, repetabilitate mare, sensibilitate mică la perturbaţii, testabilitate mai bună, flexibilitate mai mare şi compatibilitate cu alte tipuri de procesare, de obicei pentru reţele neuronale de dimensiuni mai mici.

Implementările digitale cu memorie în afara chip-ului sunt depăşite în toate domeniile de DSP-uri sau alte procesoare, care oferă performanţe cel puţin egale la preţuri mai mici şi un timp de dezvoltare mai mic. Doar consumul de putere este ceva mai redus.

Implementările digitale pot fi realizate mai uşor datorită existenţei programelor de proiectare asistată. Există şi câteva circuite comerciale de uz general care dispun de un set de instrucţiuni sau de o arhitectură specifică reţelelor neuronale.

Implementările analogice sunt mult mai greu de proiectat şi sunt potrivite numai pentru producţiile de serie mare sau aplicaţii foarte specifice, fiind singura soluţie în cazul cerinţelor legate de consumul de putere redus şi frecvenţa de eşantionare ridicată.

Toate implementările software şi hibride cu FPGA sunt mult mai ieftine în cazul seriilor mici şi medii, cu excepţia aplicaţiilor de viteză foarte mare. Implementările hibride şi de tipul „sistem pe un chip" sunt foarte potrivite pentru sistemele dedicate oferind cel mai bun raport preţ/performanţă, datorită efortului mic de proiectare în tehnica HW/SW codesign.

Proiectarea integrată hardware-software (HW/SW codesign) a reţelelor neuronale artificiale devine tot mai răspândită deoarece oferă:

- viteză de procesare foarte mare sau medie spre mare la un preţ mai scăzut, prin combinarea circuitelor HW mai rapide cu părţile software mai ieftine şi mai flexibile;
- proiectarea simplă a „sistemelor pe un chip" care integrează pe un chip părţile hardware şi software;
- toleranţă crescută la defecte deoarece părţile HW şi SW se pot testa sau îşi pot compensa defectele reciproc.

În concluzie implementările în circuite dedicate hardware pierd teren în timp ce implementările hibride HW/SW sunt tot mai folosite pentru dezvoltarea rapidă de sisteme specifice ce includ şi blocuri de reţele neuronale.

Figura 2.17 [106] prezintă performanţele pentru mai multe arhitecturi. Performanţa neurocomputerelor este exprimată în numărul de conexiuni. Viteza este măsurată în conexiuni pe secundă CPS. Poziţia în grafic este starea de fapt la un moment dat şi are doar scop orientativ. Implementările seriale sunt poziţionate în colţul din stânga-jos. Sistemele multiprocesor, masiv paralele realizate cu neurochip-uri analogice au cele mai mari dimensiuni şi cea mai mare viteză. Pentru comparare graficul prezintă şi patru implementări biologice. Performanţele creierului uman, exprimate în număr de conexiuni şi număr de conexiuni pe secundă, depăşesc cu mult implementările artificiale.

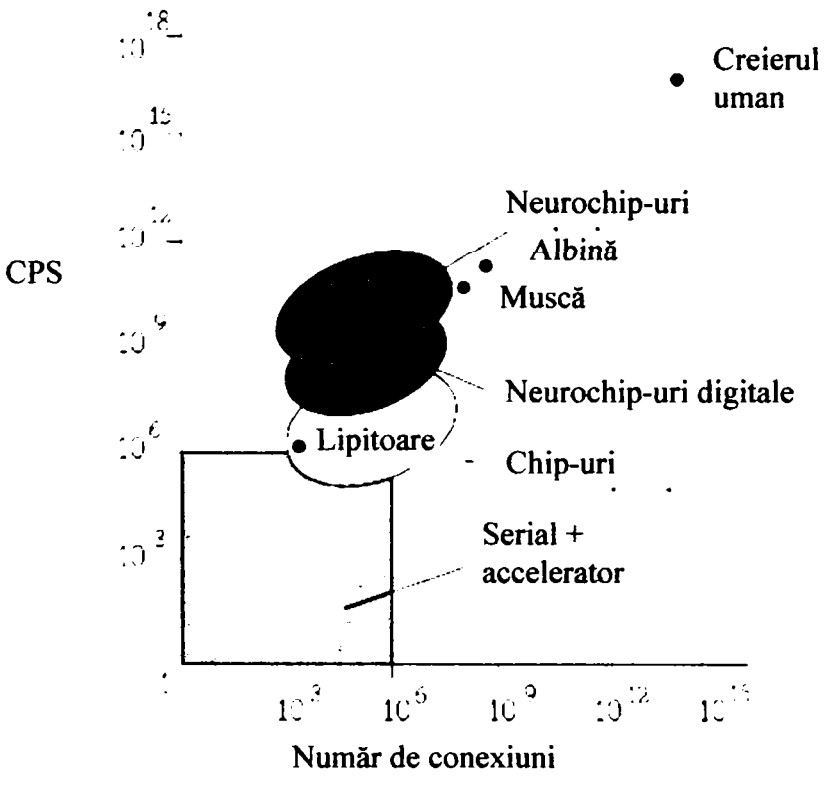


Figura 2.17 Performanţele neurocomputerelor

Capitolul III

Implementarea reţelelor neuronale artificiale în FPGA

3.1 Introducere

Reţelele neuronale artificiale sunt de inspiraţie biologică şi necesită calcul paralel. Un model real poate să atingă 1 milion de neuroni şi zeci de miliarde de conexiuni. Microprocesoarele şi procesoarele digitale de semnal nu sunt potrivite pentru implementarea calculului paralel. Proiectarea de module complet paralele este posibilă în circuitele ASIC dar sunt scumpe, dezvoltarea lor necesită un timp îndelungat, iar RNA obţinută este potrivită unui singur tip (clase) de aplicaţii.

Problemele care se pun la implementarea reţelelor neuronale (software sau hardware) se referă la:

- topologia reţelei (cu propagare înainte sau cu reacţie)
- numărul de intrări şi de ieşiri
- numărul de neuroni
- numărul de sinapse per neuron
- numărul de straturi
- algoritmul de învăţare
- operaţii (calculul) în virgulă mobilă

Pentru implementările hardware se mai pun suplimentar următoarele probleme:

- tehnologia folosită (analogic, digital sau hibrid)
- reprezentarea numerică a intrărilor, ponderilor sinaptice şi a ieşirilor funcţiei de activare (în virgulă fixă sau mobilă)
- precizia reprezentării intrărilor, ponderilor sinaptice şi a ieşirilor funcţiei de activare (număr de biţi)

Costul, performanţele şi reconfigurabilitatea sunt determinate de tehnologia aleasă pentru implementare:

- implementările analogice au o performanţă bună şi preţ scăzut, dar sunt dificil de implementat datorită cerinţelor legate de referinţele de tensiune şi există probleme legate de stocarea ponderilor [173]
- implementările digitale în ASIC au preţ mai ridicat şi performanţe mai scăzute, dar sunt mai uşor de implementat

Ambele tehnologii, atât cea analogică cât şi cea digitală în ASIC au topologie fixă deci sunt dedicate unei anumite aplicaţii.

Circuitele FPGA oferă nu numai paralelismul calculelor ci şi flexibilitatea oferită de reprogramabilitate, scurtarea ciclului de fabricaţie şi reducerea costurilor. De asemenea ele permit testarea diferitelor topologii de RNA folosind acelaşi hardware.

Datorită avantajelor oferite circuitele FPGA sunt tot mai folosite pentru implementarea RNA în hardware aşa cum se prezintă şi în [10]-[14], [18]-[19], [36], [71], [75], [96], [107], [116], [149]-[151], [185], [192], [216].

În continuare este prezentată arhitectura circuitelor FPGA, problemele care se pun la implementarea RNA în FPGA şi se trec în revistă implementările existente.

3.2 Descrierea arhitecturii FPGA

Ariile logice reconfigurabile sunt cele mai răspândite circuite programabile în momentul de faţă. Ciclul redus de proiectare şi densitatea elementelor integrate fac ca aceste circuite să fie preferate în proiectarea sistemelor digitale. Efortul de proiectare şi riscul sunt mai reduse în comparaţie cu reducerea costurilor de proiectare, dar utilizate într-un volum mic ele sunt costisitoare.

Ariile logice FPGA au fost introduse în 1985 de firma XILINX. Poziţia de lider a fost păstrată prin continua îmbunătăţire a circuitelor, şi prin oferta soluţiilor de proiectare completă. Criteriile pe baza cărora firma şi-a păstrat prestigiul se pot unii în aşa numitul "triunghi de siliciu", în care fiecare latură (siliciu, software şi servicii oferite) îşi are importanţa sa.

Procesul de proiectare cu circuite XILINX FPGA este rapid şi eficient, iar durata acestui proces este de câteva zile în comparaţie cu câteva săptămâni, termen obişnuit cu alte tipuri de circuite programabile.

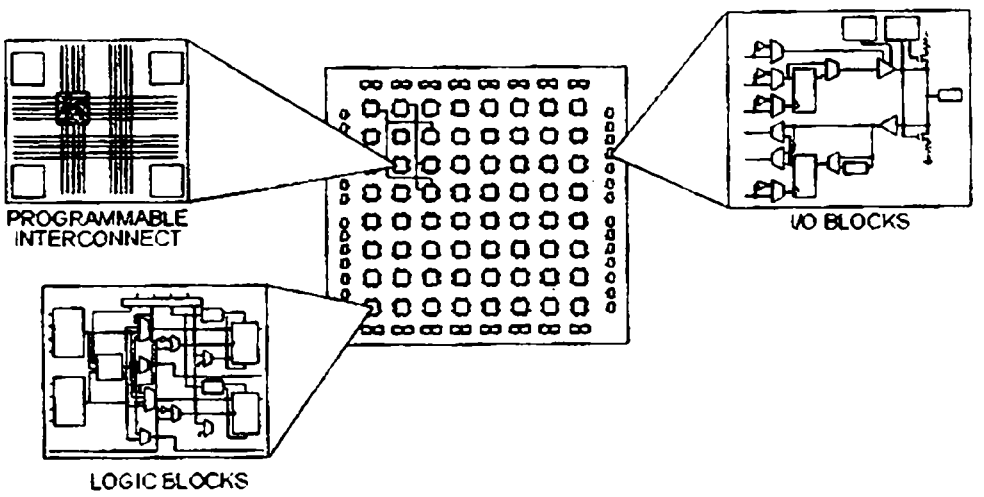


Figura 3.1 Arhitectura internă FPGA

Circuitele FPGA au structura internă asemănătoare cu cea a ariilor generice. Structura internă prezentată în figura 3.1, este organizată într-o matrice de celule înconjurate la periferie de celulele de intrare/ieşire I/O. Segmentele de interconexiuni din metal pot realiza prin intermediul punctelor de conexiune, legături între celulele logice configurabile şi celulele de intrare/ieşire.

Abundenţa de porţi logice, registre, interfeţe I/O cu o viteză de răspuns mare, sunt doar câteva caracteristici ale circuitelor logice reprogramabile FPGA. Seria circuitelor cu aria de configurare de tip SRAM include în principal următoarele familii de produse: Spartan-I, Spartan-II, Spartan 3, Virtex, Virtex-II, Virtex-II Pro şi Virtex-4 [221-225].

În continuare se prezintă pe scurt structura unui circuit din familia Virtex-II folosit pentru implementarea de reţele neuronale artificiale. Circuitele din familia Virtex-II sunt proiectate pentru dezvoltarea de aplicaţii de mare performanţă, la o densitate medie şi mare. Familia Virtex-II oferă soluţii complete pentru aplicaţii din domeniul telecomunicaţii, reţele, procesarea digitală a semnalelor, etc. De asemenea circuitele sunt prevăzute cu posibilitatea de interfaţare cu varietate foarte mare de standarde, printre care PCI, LVDS, DDR, etc.

Realizate în tehnologie CMOS de 0,15 μm arhitectura lor este optimizată pentru viteză mare la un consum mic de putere. Aşa cum se poate vedea în tabelul 3.1 familia este compusă din 11 circuite care conţin între 40.000 şi 8 milioane de porţi logice echivalente.

Tabelul 3.1 Membrii familiei Virtex-II

Circuitul	Porţi logice echivalente	CLB (1 CLB =4 slice=Max 128 biţi)			Blocuri multiplicat.	Bloc SelectRAM		DCM	Nr. max. I/E
		Arie de Linii x Coloane	Slice-uri	Max. RAM Distribuit Kbiţi		Bloc 18K bit	Max RAM (Kbiţi)		
XC2V40	40K	8 x 8	256	8	4	4	72	4	88
XC2V80	80K	16 x 8	512	16	8	8	144	4	120
XC2V250	250K	24 x 16	1.536	48	24	24	432	8	200
XC2V500	500K	32 x 24	3.072	96	32	32	576	8	264
XC2V1000	1M	40 x 32	5.120	160	40	40	720	8	432
XC2V1500	1.5M	48 x 40	7.680	240	48	48	864	8	528
XC2V2000	2M	56 x 48	10.752	336	56	56	1.008	8	624
XC2V3000	3M	64 x 56	14.336	448	96	96	1.728	12	720
XC2V4000	4M	80 x 72	23.040	720	120	120	2.160	12	912
XC2V6000	6M	96 x 88	33.792	1.056	144	144	2.592	12	1.104
XC2V8000	8M	112 x 104	46.592	1.456	168	168	3.024	12	1.108

3.2.1 Descrierea structurii interne a circuitelor FPGA din familia Virtex-II

Familia **Virtex-II [223]** este implementată într-o structură regulată, flexibilă cu o arhitectură programabilă, realizată prin blocuri logice configurabile (*Configurable Logic Blocks, CLBs)*, blocuri de intrare/ieşire (*Input/Output Blocks, IOBs),* module de memorie bloc RAM (*Bloc SelectRAM*), multiplicatoare dedicate *Multiplier Blocks),* blocuri DCM (*Digital Clock Manager*), şi resurse de conexiuni programabile *(Active Interconnect Technology Interconnects).*

Circuitele sunt configurate cu ajutorul memoriei interne (inaccesibilă utilizatorului). Configuraţia este realizată fie în mod activ de către **FPGA** prin citirea unei memorii **EPROM** (serie/paralel), sau configuraţia este înscrisă de un microprocesor, microcontroler sau alt **FPGA** prin mai multe metode, care vor fi descrise în cele ce urmează.

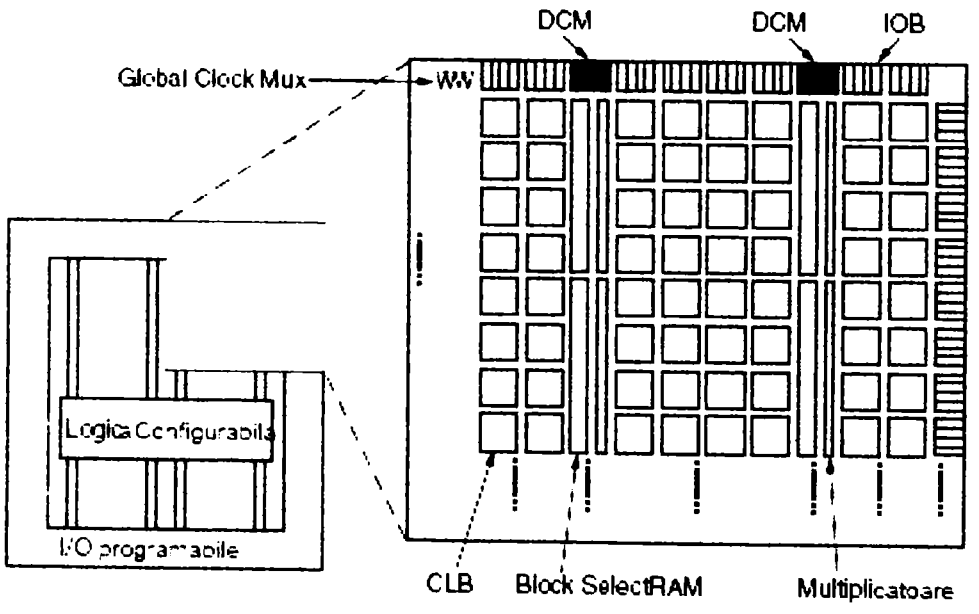


Figura 3.2 Arhitectura Virtex-II

Descrierea blocurilor componente

Structura internă programabilă de utilizator include două elemente majore configurabile:

- Blocurile logice configurabile (**CLBs);**
- Blocurile de intrare/ieşire (**IOBs).**

Logica internă configurabilă include patru elemente majore organizate într-o arie regulată:

- Blocurile CLB furnizează elementele funcţionale pentru logica combinaţională şi secvenţială. Fiecărui CLB îi este asociat un 3-state buffer (**TBUF)** a cărui ieşire este conectată la liniile lungi asociate;
- Modulele de memorie Block SelectRAM asigură elemente de stocare de 18 Kbiţi de tip dual-port RAM;
- Multiplicatoare dedicate de 18 x 18 biţi;
- Blocurile DCM (Digital Clock Manager) oferă soluţii complet digitale pentru compensarea întârzierilor pe traseele de ceas, multiplicarea şi divizarea frecvenţei semnalului de tact;

O nouă generaţie de interconexiuni programabile numite Active Interconnect Technology, furnizează resursele de conectare între elementele CLB, IOB.

Blocurile IOB furnizează interfaţa între semnalele interne şi exteriorul circuitului (legătură realizată fizic prin intermediul pinilor). Sunt suportate standardele cele mai uzuale precum şi cele mai noi.

Funcţia logică realizată de fiecare bloc configurabil este implementată prin intermediul memoriei statice de configurare. Valorile stocate în aceste memorii determină starea blocurilor şi a interconexiunilor în interiorul **FPGA.**

3.2.1.1 Blocurile logice configurabile (CLB)

CLB-urile includ patru unităţi funcţionale numite slice-uri şi două bufere cu 3 stări. Elementele principale ale unui bloc configurabil sunt prezentate în figura 3.3.

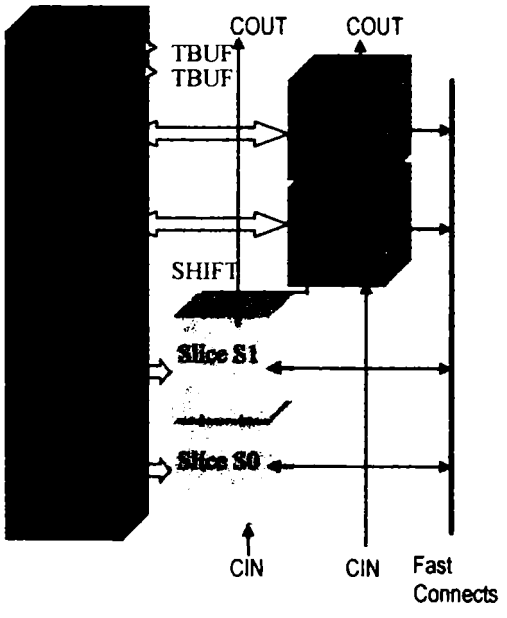


Figura 3.3 Structura internă a blocului logic configurabil (CLB)

Fiecare slice este echivalent şi conţine:

- Două generatoare de funcţii (F şi G) cu câte patru intrări;
- Două elemente de stocare
- Porţi logice
- Multiplexoare

Aşa cum se poate vedea în figura 3.4 fiecare generator de funcţii poate fi implementat ca tabele de memorii (look-up table, LUT) cu patru intrări, memorie distribuită de 16 biţi (RAM16 distributed SelectRAM), sau un registru de deplasare de 16 biţi (SRL16). Ieşirea generatorului de funcţii alimentează atât ieşirea din slice cât şi intrarea D a elementului de stocare.

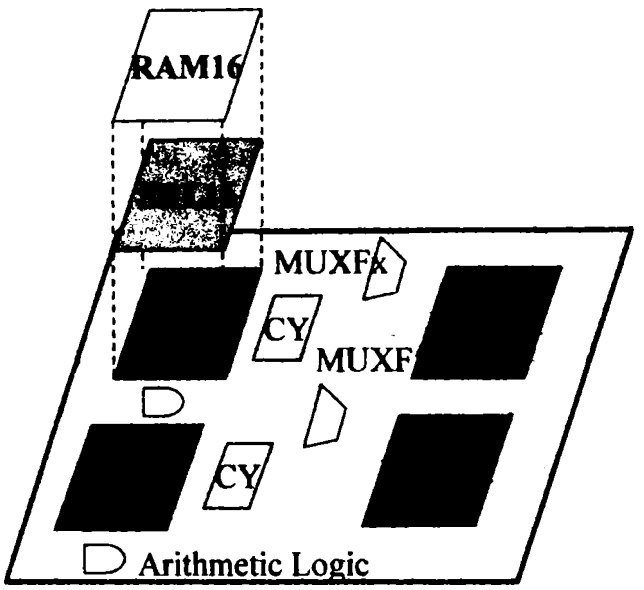


Figura 3.4 Structura internă a unui SLICE

În figura 3.5 se poate observa o reprezentare simplificată a unei celule logice elementare dintr-un circuit Virtex-II, iar o reprezentare mai detailată a jumătăţii superioare dintr-un slice se poate vedea în figura 3.6. Două astfel de celule compun un slice.

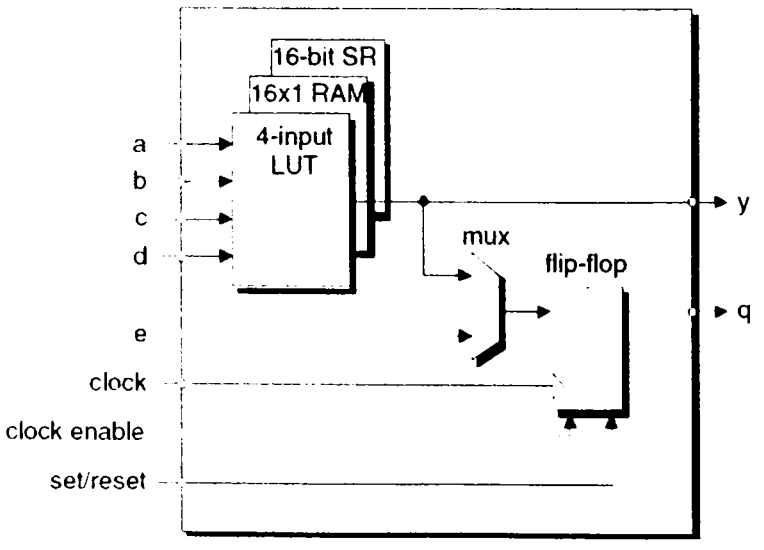


Figura 3.5 Celula logică elementară din circuitul Virtex-II

Blocul configurabil CLB conține două elemente de stocare/registru (bistabili D), care se pot utiliza pentru stocarea rezultatelor date de generatoarele de funcții. De asemenea elementele de registru sau generatoarele de funcții se pot utiliza și independent.

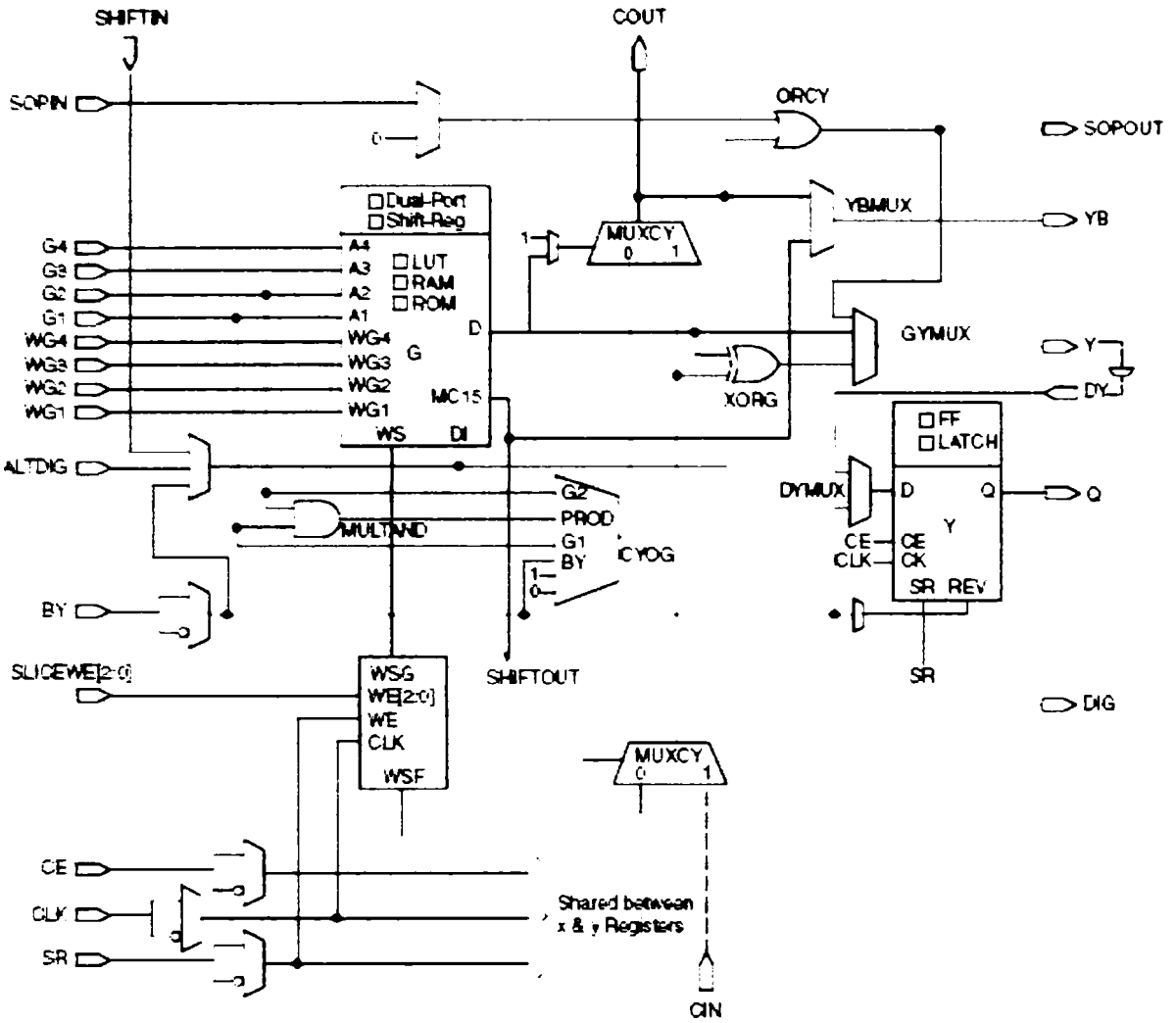


Figura 3.6 Jumătatea superioară a unui slice Virtex-II

3.2.1.2 Module de memorie de tip Block SelectRAM

Circuitele Virtex-II conțin blocuri de memorie RAM de 18 Kbiți. Acestea completează resursele de memorie distribuită (distributed SelectRAM) ce pot fi implementate cu ajutorul blocurilor CLB. Fiecare modul de memorie bloc SelectRAM este un dual port RAM de 18 Kbiți cu semnale de ceas și semnale de control sincron independente care accesează o arie de stocare comună. Ambele porturi sunt identice din punct de vedere funcțional. Modurile de configurare posibile a acestor module de memorie sunt prezentate în tabelul 3.2.

16K x 1 bit	2K x 9 biți
8K x 2 biți	1K x 18 biți
4K x 4 biți	512 x 36 biți

Tabelul 3.2 Modurile de configurare posibilă a memoriei Block SelectRAM

Aceste blocuri sunt aranjate în coloane, așa cum se poate observa în figura 3.2. Numărul de coloane și blocuri pe coloană depinde de circuit Memoria totală disponibilă pentru această familie este cuprinsă între 72 Kbiți și 3 Mbiți.

3.2.1.3 Blocurile multiplicatoare

Un bloc multiplicator din Virtex-II este un multiplicator pe 18x18 biţi a două numere cu semn reprezentate în complement faţă de 2. Aceste multiplicatoare pot fi asociate cu blocurile de memorie SelectRAM de 18 Kbiţi, sau pot fi folosite independent. Ele sunt optimizate pentru viteză de lucru ridicată şi au un consum de putere mai mic în comparaţie cu un multiplicator implementat în slice.

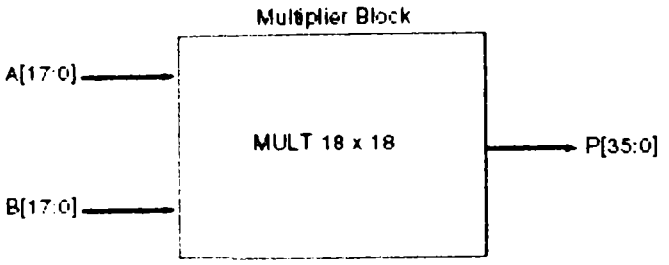


Figura 3.7 Blocul multiplicator

Fiecare bloc de memorie SelectRAM şi bloc multiplicator este legat la patru matrici de interconexiuni, ca în figura 3.8.

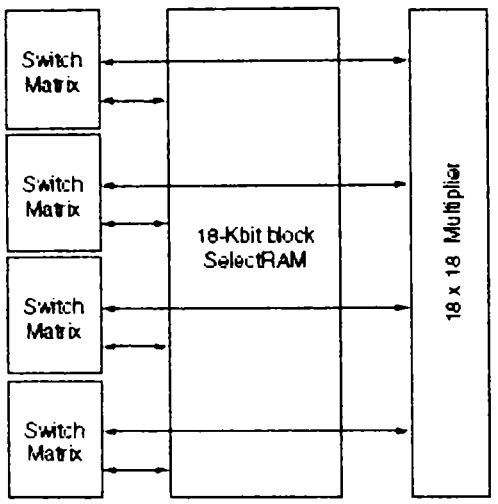


Figura 3.8 Modul de interconectare între blocurile SlectRAM şi multiplicatoare

Acest mod de legare este optimizat astfel încât blocul de memorie să alimenteze multiplicatorul cu date pe 18 biţi. Utilizarea acestora împreună cu un acumulator realizat folosind tabele de memorii (LUT) permite implementarea foarte eficientă a unui bloc de multiplicare acumulare (MAC), specific procesoarelor digitale de semnal, folosit de obicei pentru realizarea filtrelor digitale FIR şi IIR, precum şi a reţelelor neuronale artificiale.

Localizarea şi organizarea multiplicatoarelor este similară cu cea a blocurilor de memorie deoarece fiecare multiplicator este asociat cu un bloc de memorie SelectRAM de 18 Kbiţi. Numărul de multiplicatoare diferă pentru fiecare membru al familiei, fiind cuprins între 4 şi 168 pentru familia Virtex-II, respectiv 12 şi 556 pentru familia Virtex-II Pro. Multiplicatoarele ating o frecvenţa maximă de lucru de 245 MHz la Virtex-II şi 300MHz la Virtex-II Pro.

Pe lângă blocurile multiplicatoare dedicate, se mai pot implementa multiplicatoare folosind blocurile logice configurabile care dispun de resurse ce permit implementarea eficientă a multiplicatoarelor.

3.2.1.4 Blocurile de intrare/ieşire (IOBs)

Blocurile configurabile de intrare/ieşire realizează interfaţa între mediul exterior şi structura internă a circuitului **FPGA.** Fiecare IOB controlează un pin al circuitului integrat. Blocurile de intrare ieşire se pot configura ca şi

- port de intrare cu un registru opţional cu rată simplă de date sau rată dublă de date (double data rate, DDR),;
- port de ieşire cu registru opţional rată simplă sau dublă de date şi un bufer opţional cu trei stări;
- port bidirecţional (orice combinaţie a configuraţiilor de intrare sau ieşire).

Aceste trei registre pot fi bistabile de tip D active pe front sau latch-uri active pe nivel. Blocurile IOB suportă o mare varietate de standarde de intrare-ieşire asimetrice sau diferenţiale. Astfel sunt suportate următoarele standarde de intrare-ieşire asimetrice:

- LVTTL, LVCMOS (3,3V, 2,5V, 1,8V, şi 1,5V)
- PCI-X (133 MHz şi 66 MHz) la 3,3V
- PCI (66 MHz şi 33 MHz) la 3,3V
- CardBus (33 MHz) la 3,3V
- GTL şi GTLP
- HSTL (Class I, II, III, şi IV)
- SSTL (3,3V şi 2,5V, Class I şi II)
- AGP-2X

Blocurile posedă facilitatea de control digital al impedanţei (digitally controlled impedance, DCI), care asigură automat controlul digital al impedanţei pentru fiecare element de intrare ieşire.

Blocurile IOB suportă de asemenea următoarele standarde de intrare-ieşire diferenţiale:

- LVDS
- BLVDS (Bus LVDS)
- ULVDS
- LDT
- LVPECL

Pinii adiacenţi sunt folosiţi pentru fiecare pereche diferenţială. Două sau patru blocuri IOB sunt conectate la o matrice de conexiune pentru a avea acces la resursele de conexiune (figura 3.9).

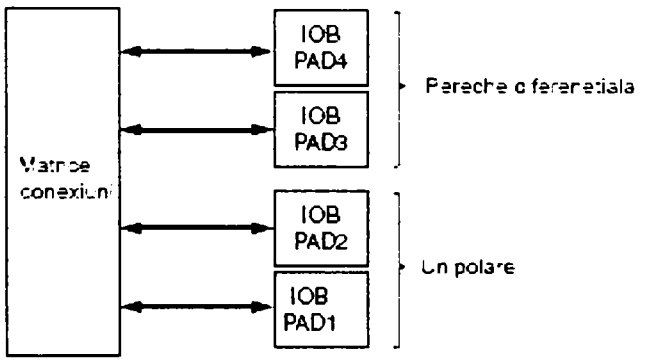


Figura 3.9 Modul de conectare a blocurilor de intrare/ieşire

În figura 3.10 este prezentată diagrama simplificată a blocului IOB.

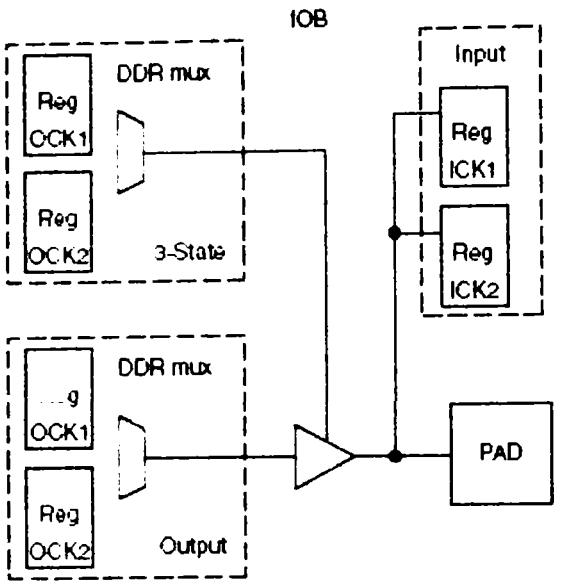


Figura 3.10 Blocul IOB

Blocurile IOB sunt împărţite în 8 bancuri care rezultă din împărţirea fiecărei laturi în două bancuri, cum se prezintă în figura 3.11. Fiecare banc are mai mulţi pini de alimentare V_{CCO} conectaţi la aceeaşi tensiune, a cărei valoare depinde de standardul de ieşire folosit. În bancuri diferite pot fi folosite mai multe standarde de intrare-ieşire diferite în acelaşi timp.

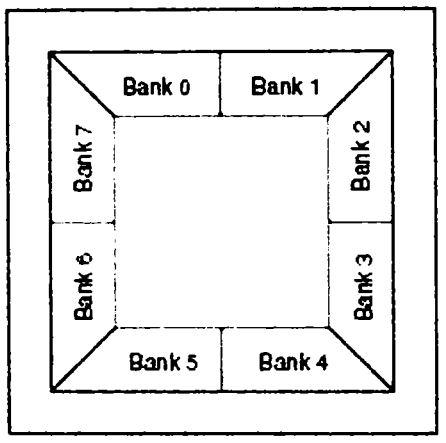


Figura 3.11 Bancurile de blocuri IOB

3.2.1.5 Interconexiunile programabile

Toate blocurile IOB, CLB, bloc SelectRAM, multiplicatoarele şi DCM folosesc aceeaşi schemă de interconectare şi acelaşi acces la matricea de conexiune globală. Există un număr de 16 *linii globale de ceas* (global clock lines), 24 *linii lungi* (long lines) verticale şi orizontale pe linie sau coloană şi de asemenea resurse secundare şi locale masive, care permit realizarea de conexiuni rapide. Interconexiunile din Virtex-II sunt afectate nesemnificativ de încărcare datorită bufferelor de ieşire, iar amplasarea legăturilor este astfel proiectată încât să minimizeze diafonia.

Resursele orizontale şi verticale pentru fiecare rând sau coloană includ:

- 24 linii lungi
- 120 linii de 6 lungimi (hex lines)
- 40 linii duble
- 16 linii de conexiune directă (totale în toate cele patru direcţii)

Blocurile logice sunt conectate la matrici de interconectare identice (figura 3.12).

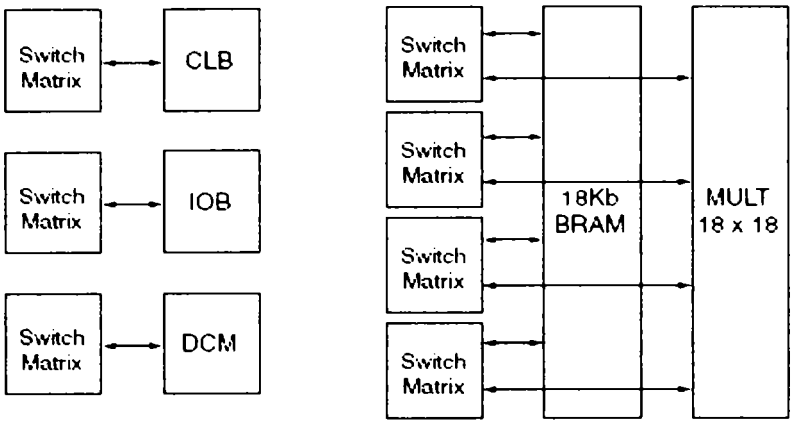


Figura 3.12 Tehnologia Active Interconnect

Toate resursele de conexiune sunt segmentate pentru a oferii avantajele unei soluții ierarhice.

Fiecare dispozitiv Virtex-II poate fi reprezentat ca o arie de matrici de interconectare care au blocuri logice atașate, cum se prezintă în figura 3.13.

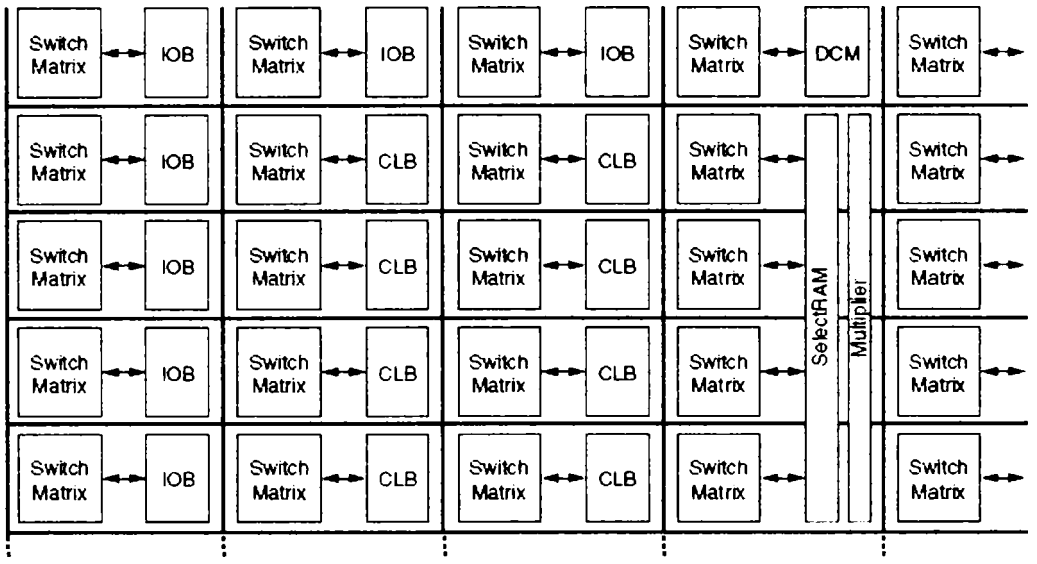
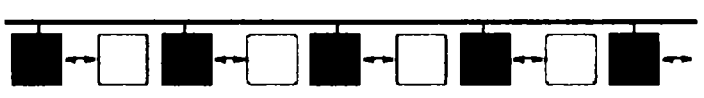
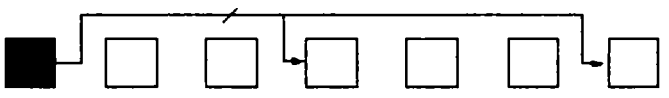
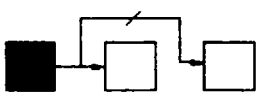


Figura 3.13 Resursele de interconectare

Programele de plasare şi interconectare (place-and-route) folosesc avantajul oferit de această structură regulată pentru a obține performanțe maxime şi timp de compilare mic.

3.2.1.5.1 Resursele de interconectare ierarhică

Majoritatea semnalelor sunt realizate folosind resursele de conexiune globale, care sunt localizate în canalele de conexiune orizontale şi verticale între matricile de conexiune.

24 linii lungi orizontale 24 linii lungi verticale	
120 linii Hex orizontale 120 linii Hex verticale	
120 linii duble orizontale 120 linii duble verticale	

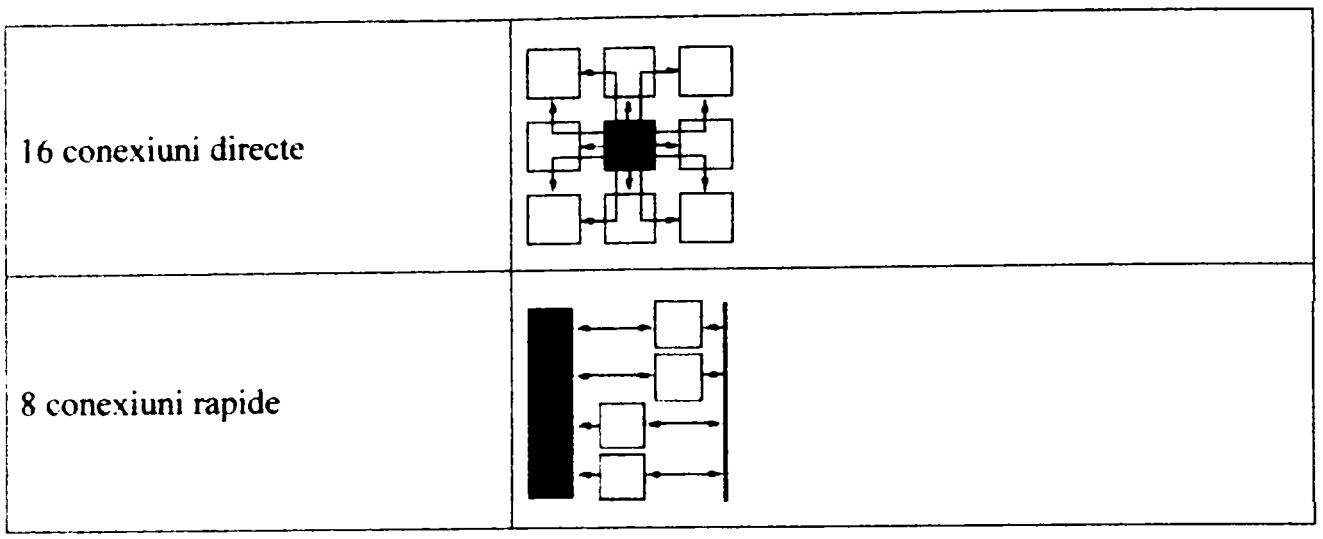


Figura 3.14 Resursele de interconectare ierarhică

- Liniile lungi sunt legături bidirecţionale care distribuie semnalul în dispozitiv. Liniile lungi verticale sau orizontale acoperă întreaga înălţime şi lăţime a dispozitivului.
- Liniile de 6 lungimi (hex lines) leagă semnalele la fiecare al treilea sau al şaselea bloc distanţa în toate cele patru direcţii. Organizat sub forma unei table de şah, ele pot fi alimentate de la un singur capăt.
- Liniile duble leagă semnalele la fiecare bloc sau al doilea bloc, în toate cele patru direcţii.
- Liniile de conexiune directă leagă semnalele între blocurile vecine pe verticală, orizontală şi diagonală.
- Liniile de conexiune rapidă sunt conexiuni locale, interne CLB, de la ieşirile din LUT la intrările LUT.

3.2.1.5.2 Conexiuni dedicate

În plus fată de resursele de conexiune globale şi locale, sunt disponibile şi conexiuni dedicate cum ar trasee globale de ceas, magistrale cu trei stări, etc.

3.2.2 Circuitele din familia Virtex-II Pro

În continuare sunt prezentate pe scurt îmbunătăţirile aduse de familia Virtex-II Pro şi diferenţele faţă de familia Virtex-II.

Dispozitivele din familia Virtex-II Pro, realizate în tehnologie de 0,13 μm, sunt adaptate pentru proiectare cu module de tip proprietate intelectuală (IP cores) şi module adaptabile [224]. Dispozitivele din această familie încorporează până la 24 tranceivere full-duplex cu o rată de transmisie cuprinsă între 622 Mb/s şi 3,125Gb/s. Tot ca o noutate ele conţin până la patru nuclee de procesoare RISC PowerPC cu o frecvenţă de 300 MHz având o arhitectură Harvard. De asemenea ele dispun de:

- mai multă memorie, până la 10 Mbiţi în bloc RAM şi 1.738 Kbiţi în memorie RAM distribuită;
- mai mulţi pini per capsulă (până la 1704);
- mai multe multiplicatoare (până la 556), cu blocuri de memorie RAM asociate;
- blocuri IOB care suportă 22 standarde de intrare-ieşire asimetrice şi 6 standarde diferenţiale.

Membrii familiei Virte-II Pro şi resursele de care dispun acestea sunt prezentate în tabelul 3.3.

Aceste dispozitive oferă soluţii complete pentru aplicaţii de telecomunicaţii, wireless, reţele, video şi procesare digitală a semnalelor.

Tabelul 3.3 Dispozitivele din familia Virtex-II Pro

Dispozitiv	Tran-ceivere RocketIO	Proce-soare PowerPC	Celule Logice	CLB		Multipli-catoare 18x18	Bloc SelectRAM		DCM	Nr. max pini utilizator
				Slice	RAM distribuit (Kb)		Blocuri 18 Kb	BlocRAM Total (Kb)		
XC2VP2	4	0	3.168	1.408	44	12	12	216	4	204
XC2VP4	4	1	6.768	3.008	94	28	28	504	4	348
XC2VP7	8	1	11.088	4.928	154	44	44	792	4	396
XC2VP20	8	2	20.880	9.280	290	88	88	1.584	8	564
XC2VP30	8	2	30.816	13.696	428	136	136	2.448	8	644
XC2VP40	12	2	43.632	19.392	606	192	192	3.456	8	804
XC2VP50	16	2	53.136	23.616	738	232	232	4.176	8	852
XC2VP70	20	2	74.448	33.088	1.034	328	328	5.904	8	996
XC2VP100	20	2	99.216	44.096	1.378	444	444	7.992	12	1.164
XC2VP125	24	4	125.136	55.616	1.738	556	556	10.008	12	1.200

3.2.3 Circuitele din familia Virtex-4

Familia Virtex-4 reprezintă cea mai nouă generaţie de circuite FPGA a firmei Xilinx realizată în tehnologie de 90 nm [225]. Arhitectura circuitului are la bază un nou bloc inovativ, numit ASML (Advanced Silicon Modular Block), unic în industria circuitelor logice programabile. Familia conţine trei platforme:

- LX, oferă soluţii pentru aplicaţiile logice de înaltă performanţă;
- FX, oferă soluţii complete pentru sistemele dedicate de înaltă performanţă;
- SX, oferă soluţii de înaltă performanţă pentru aplicaţiile de procesare digitală a semnalului

Posibilitatea combinării caracteristicilor şi numărul mare de blocuri de tip proprietate intelectuală hardware (hard-IP blocks) completează soluţiile de sistem. Dispozitivele conţin de asemenea:

- procesoare PowerPC
- controlerul Ethernet MAC (Media Access Control) cu trei moduri de operare (10/100/1000 Mb/s)
- tranceivere seriale cu viteza cuprinsă între 622 Mb/s şi 11,1 Gb/s
- slice-uri dedicate procesării digitale a semnalelor
- circuite de managementul semnalului de ceas de frecvenţă mare (Xesium Clock Tehnology)

Prin combinarea unei mari varietăţi de caracteristici flexibile, familia Virtex-4 îmbunătăţeşte foarte mult capabilităţile circuitelor logice programabile şi reprezintă o alternativă la tehnologia ASIC

Dintre îmbunătăţirile şi modificările notabile faţă de dispozitivele din familiile Virtex anterioare, se remarcă următoarele:

- o creştere de până la 40% a vitezei de lucru a celulelor logice;
- includerea de blocuri dedicate funcţiilor DSP (XtremeDSP Slice) funcţionând la 500 MHz. Acestea conţin:

- multiplicatoare pe 18x18 biţi, dedicate,
- niveluri de pipeline opţionale pentru îmbunătăţirea performanţelor,
- blocuri multiplicatoare-acumulatoare sau multiplicatoare-sumatoare,
- acumulator opţional pe 48 de biţi pentru operaţiile de multiplicare acumulare (MACC),
- sumator integrat,
- blocurile de multiplicare sau MACC sunt cascadabile,
- creşterea vitezei cu 100% fată de generaţia anterioară;

- blocuri de memorie integrate funcţionând la 500 MHz, care conţin:
  - blocuri de memorie integrată de până la 10 Mb
  - mod de operare sincron sau asincron
  - arhitectura dual-port
  - creşterea vitezei cu 100% fată de generaţia anterioară;

Tabelul 3.4 Dispozitivele din familia Virtex-4

	Blocuri logice configurabile					Bloc RAM				
Dispozitiv	Arie Rând x Col.	Celule logice	Slice-uri	RAM distrib.	Slice-uri XtremDSP	Blocuri 18 Kb	RAM max (Kb)	Blocuri PowerPC	Ethernet MAC	Tran-ceivere
XC4VLX15	64 x 24	13.824	6.144	96	32	48	864	N/A	N/A	N/A
XC4VLX25	96 x 28	24.192	10.752	168	48	72	1.296	N/A	N/A	N/A
XC4VLX40	128 x 36	41.472	18.432	288	64	96	1.728	N/A	N/A	N/A
XC4VLX60	128 x 52	59.904	26.624	416	64	160	2.880	N/A	N/A	N/A
XC4VLX80	160 x 56	80.640	35.840	560	80	200	3.600	N/A	N/A	N/A
XC4VLX100	192 x 64	110.592	49.152	768	96	240	4.320	N/A	N/A	N/A
XC4VLX160	192 x 88	152.064	67.584	1056	96	288	5.184	N/A	N/A	N/A
XC4VLX200	192 x 116	200.448	89.088	1392	96	336	6.048	N/A	N/A	N/A
XC4VSX25	64 x 40	23.040	10.240	160	128	128	2.304	N/A	N/A	N/A
XC4VSX35	96 x 40	34.560	15.360	240	192	192	3.456	N/A	N/A	N/A
XC4VSX55	128 x 48	55.296	24.576	384	512	320	5.760	N/A	N/A	N/A
XC4VFX12	64 x 24	12.312	5.472	86	32	36	648	1	2	N/A
XC4VFX20	64 x 36	19.224	8.544	134	32	68	1.224	1	2	8
XC4VFX40	96 x 44	41.904	15.552	243	48	144	2.592	2	4	12
XC4VFX60	128 x 52	56.880	25.280	395	128	232	4.176	2	4	16
XC4VFX100	160 x 68	94.896	42.176	659	160	376	6.768	2	4	20
XC4VFX140	192 x 84	142.128	63.168	987	192	552	9.936	2	4	24

3.2.3.1 Slice-ul XtremeDSP

Blocul XtremeDSP (sau DSP48) este un multiplicator de mare performanţă şi o unitate aritmetică cu o mare flexibilitate care poate constitui blocul constructiv pentru implementarea în FPGA a diverşilor algoritmi de procesare digitală a semnalelor. O schemă bloc detailată a structurii DSP48 este prezentată în figura 3.15.

Slice-ul XtremeDSP include patru părţi principale:

- Regiştrii de intrare-ieşire, care asigură o frecvenţă maximă de ceas de 500 MHz
- Multiplicatorul de 18x18 biţi
- Blocul de adunare/scădere cu trei intrări
- Multiplexoarele (op-mode multiplexers), reprezintă cheia funcţionării structurii, ele asigură deosebita flexibilitate a blocurilor DSP48.

În dispozitivele Virtex-4 blocurile XtremeDSP sunt aranjate în coloane.

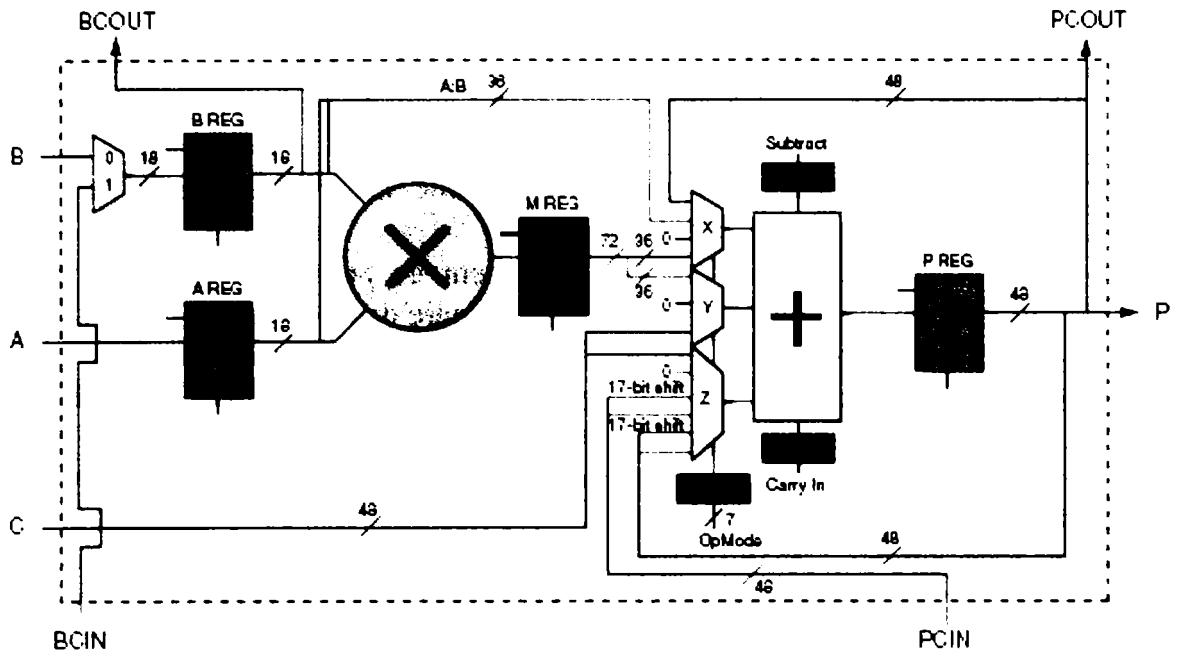


Figura 3.15 Blocul XtremeDSP din circuitele Virtex-4

3.3 Probleme care se pun la implementarea RNA

Acest paragraf descrie toți parametrii de performanță relevanți care trebuiesc urmăriți la implementarea rețelelor neuronale în circuite FPGA. În funcție de aplicație și constrângerile ei doar o parte dintre aceşti parametrii au relevanță.

3.3.1 Paralelism şi virtualizare

Limitările majorității procesoarelor de uz general (microprocesoarele din calculatoarele personale, DSP-urile, microcontrolerele, etc.) sunt date de faptul că ele dispun de o unitate de procesare secvențială, ceea ce înseamnă că doar un singur calcul poate fi executat la un moment de timp dat. Pe de altă parte, gradul mare de paralelism al implementărilor hardware dedicate oferă o creştere a performanţelor de viteză şi toleranță la defecte. Astfel o rețea neuronală, având N intrări şi M neuroni, necesită N x M elemente de procesare.

Având un număr Q > 1 de elemente de procesare (PE), se obține o creştere a vitezei şi o reducere a timpului de calcul cu un factor de până la Q. În practică factorul Q poate fi atins numai în cazul unui calcul repetitiv cum este şi cel cerut de algoritmii rețelelor neuronale, pentru N x M ≥ Q. Se poate defini un factor de virtualizare:

$$v = ceil(\frac{NxM}{Q}) \tag{3.1}$$

Unde funcția ceil întoarce numărul întreg superior argumentului
Există câteva cazuri particulare relevante:

- $v = 1$, adică pentru un element de procesare pe sinapsă. Acestea poartă numele de sisteme complet paralele şi ele calculează întreaga arie de sinapse într-un singur ciclu de timp T_{CS}, care este timpul necesar pentru efectuarea unei operații elementare. Timpul T_{CS} reprezintă un număr C_S de perioade de ceas;

- v = N, adică un element de procesare pe neuron. Fiecare PE calculează secvențial în v cicluri consecutive, v T_{CS}) contribuția tuturor sinapselor asociate unui neuron. Aceasta se numește paralelism de neuron sau virtualizare per neuron;
- v >> N, M, adică un număr relativ mic de PE calculează in v cicluri consecutive ieșirile neuronilor;
- v =NxM, adică există un singur element de procesare. De fapt este vorba despre implementările software cu procesoare de uz general.

Primul caz este tipic pentru implementările analogice și pulse-stream, mai rar pentru implementări digitale deoarece Q este limitat de dimensiunile circuitului. Odată proiectat numărul de elemente de procesare mai poate fi modificat numai prin reprogramare

Următoarele trei cazuri sunt sisteme parțial paralele, tipice implementărilor digitale. Se mai spune că, Q elemente de procesare virtualizează NxM sinapse. Această abordare este mai flexibilă și expandabilă, numărul de sinapse poate fi ușor modificat, modificând v și păstrând numărul de PE constant. Factorul de virtualizare indică de câte ori este utilizat secvențial fiecare PE pentru calculul întregii arii sinaptice.

3.3.2 Antrenarea

Antrenarea este una din marile probleme ale rețelelor neuronale, deoarece necesită un algoritm mai complex și mai puțin regulat decât faza de propagare înainte și este de multe ori executată o singură dată într-un ciclu de existența a sistemului.

Antrenarea necesită întotdeauna atât hardware cât și software adițional. O parte din hardware poate fi utilizat atât în faza de antrenare cât și în faza de propagare. Sistemele complet paralele de obicei nu reutilizează blocurile hardware.

Există trei cazuri principale:

- Antrenarea off-line (numită și off-chip) antrenează rețeaua pe un calculator extern (de exemplu, cu un program de simulare comercial), fără HW dedicat. În acest caz este nevoie de un model exact al dispozitivului hardware pentru a putea fi utilizat de simulator în procesul de antrenare a rețelei. Se pot utiliza ponderi „numai-citire" (în cazul circuitelor ASIC pentru o antrenare inflexibilă ante-fabricație), inscriptibile o singură dată (în cazul FPGA pentru o antrenare mai flexibilă, ante-programare), sau inscriptibile (pentru antrenarea post-fabricație sau post-programare, care este mult mai flexibilă dar necesită memorare nevolatilă).
- Antrenarea cu dispozitivul HW în bucla de simulare numită chip-in-the-loop sau Hardware-in-the-loop, realizează antrenarea prin intermediul unui calculator. Dispozitivul hardware efectuează faza de propagare înainte (prezentând astfel comportamentul lui exact), în timp ce calculatorul efectuează actualizarea ponderilor (faza de antrenare). În acest caz nu este necesar modelul dispozitivului HW. Pentru antrenare continuă calculatorul trebuie să fie conectat permanent la dispozitiv

  O variantă a acestei metode de antrenare este folosită la implementările hibride HW/SW, unde algoritmul de antrenare este implementat software în nucleul de procesor din chip, în timp ce faza de propagare înainte este implementată in HW digital. Astfel se obține atât o viteză mare în faza de propagare înainte cât și o flexibilitate ridicată în faza de antrenare (care adeseori necesită o viteză mai redusă de execuție).

  Ponderile trebuie să fie inscriptibile și nevolatile, cu excepția antrenării continue.

- Antrenarea on-chip, antrenează reţeaua neuronală prin intermediul unor circuite adiţionale implementate pe acelaşi chip. Cu toate că nu necesită un calculator extern aceasta soluţie este destul de rar utilizată deoarece necesită dispozitive HW mai complexe (uneori de două-trei ori mai costisitoare), oferind o facilitate care este folosită de obicei o singură dată la începutul ciclului de viată a sistemului. Antrenarea on-chip devine atractivă când este necesară antrenarea continuă, cum este cazul sistemelor neuronale adaptive.

Este important de subliniata că spre diferenţă de reţelele neuronale analogice, cele digitale pot fi descrise printr-un model precis, repetabil şi testabil, astfel încât orice metodă de antrenare a lor este potrivită.

Rezoluţia ponderilor, necesită o analiză separată, deoarece mulţi algoritmi de antrenare necesită o rezoluţie minimă, care este mai mare decât precizia necesară pentru reţeaua neuronală. Această problemă poate fi depăşită folosind mai mulţi biţi la memorarea ponderilor decât sunt folosiţi la multiplicare.

3.3.3 Memorarea ponderilor

Antrenarea necesită întotdeauna calculul şi modificarea ponderilor. Calculul ponderilor poate fi efectuat o singură dată înainte de programarea circuitului, caz în care ponderile nu trebuiesc actualizate (pot fi de tip numai-citire sau inscriptibile o singură dată), sau în mod repetat, caz în care ele trebuiesc actualizate deci trebuie să fie reinscriptibile. Antrenarea repetată este folosită în sisteme adaptabile (la condiţiile de funcţionare) sau flexibile (adaptabile la aplicaţii diferite).

3.3.4 Parametrii de performanţă

Acest paragraf descrie parametrii de performanţă care sunt cei mai relevaţi pentru implementările HW dedicate.

3.3.4.1 Zgomotul şi precizia

Precizia unui sistem neuronal este limitată de un număr de factori, numiţi în mod generic, zgomote. Zgomotele pot fi deterministice, dacă efectul lor poate fi prevăzut şi este reproductibil, sau stocastic, dacă efectul lor este total aleator, deci nereproductibil. Există două tipuri mari de zgomote cu impact diferit asupra preciziei:

- Zgomotul de cuantizare este datorat rezoluţiei limitate şi este o funcţie de numărul de biţi, b, utilizaţi. Zgomotul de cuantizare este deterministic şi este dat de partea trunchiată din semnalul digital. Rezoluţia şi zgomotul de cuantizare sunt invers proporţionale cu 2^b.
- Zgomotul numeric se datorează aproximărilor de calcul, din orice tip de implementare. Este dat de diferenţa dintre răspunsul circuitului real şi cel ideal sau a modelului folosit la antrenare. Uneori se datorează trunchierii suplimentare la ieşirea multiplicatorului digital sau a funcţiei de activare.

Ambele tipuri de erori sunt funcţie de lungimea b a cuvântului, deci precizia lor poate fi ajustată în funcţie de cerinţe prin alegerea lungimii corespunzătoare a cuvântului. Sistemele digitale sunt intrinsec repetabile.

Dispozitivele digitale dedicate pot fi realizate pentru orice lungime b >= 1 a cuvântului, în timp ce procesoarele de uz general au o lungime predefinită (de obicei b=8, 16, 32 biţi). Precizia, consumul de putere, dimensiunile şi costul sunt afectate de b, de aceea la proiectare trebuie să se accepte un compromis între aceşti parametrii. Lungimea cuvintelor

este de obicei cuprinsă între 4 şi 16 biţi. Nu există o regulă care să precizeze lungimea optimă a cuvintelor şi nici precizia minimă necesară pentru o aplicaţie dată. Ca o regulă empirică, este rareori un avantaj o precizie mai mare de calcul decât este disponibilă la intrare sau ieşire. În plus cu cât un neuron are mai multe sinapse cu atât ele trebuie să fie de precizie mai mică, deoarece erorile aleatoare deseori se anulează reciproc. Numai o simulare poate indica dacă o lungime de cuvânt sau o precizie dată, este suficientă sau trebuie crescută, şi care este precizia obţinută de întreaga reţea neuronală.

3.3.4.2 Perioada de eşantionare şi lărgimea de bandă

Două mărimi importante care definesc performanţele sunt perioada de eşantionare T_S şi lărgimea de bandă B, care sunt relevante când se procesează semnale eşantionate în timp respectiv semnale continue.

Pentru implementările digitale şi software:

$$T_S \geq \max_i\{T_i \nu_i\} = \frac{\max_i\{C_i \nu_i\}}{F_{CK}} \tag{3.2}$$

unde T_i reprezintă timpul de întârziere (delay time), C_i = ciclurile de ceas pe operaţie, ν_i = factorul de virtualizare, iar F_{CK} este frecvenţa de ceas.

Din Teorema lui Nyquist rezultă:

$$B \leq \min_i\{\frac{1}{2T_i \nu_i}\} = \frac{F_{CK}}{2\max_i\{C_i \nu_i\}} \tag{3.3}$$

3.3.4.3 Numărul de pini

Numărul de pini este adeseori o problemă a implementărilor HW, şi influenţează în mod semnificativ dimensiunile, costul sau viteza.

Sistemele complet paralele ($\nu = 1$) au de obicei ponderile memorate pe chip, în timp ce în cazul sistemelor cu $\nu > 1$, pot avea ponderile memorate pe chip sau în afara lui. În cazul ponderilor stocate pe chip, pinii sunt utilizaţi pentru transferul semnalelor de intrare şi ieşire în timp ce în cazul ponderilor stocate în memorii externe ei sunt utilizaţi şi pentru transferul ponderilor.

De exemplu o reţea neuronală cu un singur strat cu N neuroni cu câte M intrări şi un transfer de date la un ciclu de ceas, trebuie să efectueze NxM calcule, să transfere N+M intrări şi ieşiri şi să citească NxM ponderi din memoria externă de date în $C_S \bullet \nu$ cicluri de ceas (aşa cum rezultă din 3.2). Dacă $\nu > 1$ semnalele de intrare şi ieşire respectiv ponderile pot fi multiplexate ceea ce duce la reducerea numărului de pini. Numărul de pini, excluzând pinii de alimentare P_{AL} şi cei de configurare, este dat de:

$$N_P \approx \frac{b(N+M)}{C_S \nu} + \log_2(C_S \nu) \tag{3.4}$$

pentru implementările digitale cu stocarea ponderilor pe chip, şi b pini pe intrare/ieşire;

$$N_P \approx \frac{b(N+M+NM)}{C_S \nu} + \log_2(C_S \nu) \tag{3.5}$$

pentru implementările digitale cu stocarea ponderilor off-chip, şi b pini pe intrare/ieşire. N, M şi b reprezintă numărul intrărilor, ieşirilor şi lungimea cuvintelor, iar $\log_2(C_S \nu)$ sunt biţii de adresă asociaţi. Numărul de pini este de multe ori limitat din motive de cost la $N_P \leq N_{Pmax}$, ceea ce duce la limitarea lui ν :

$$\nu > \frac{b(N+M+NM)}{C_S N_{P\max}} \tag{3.6}$$

3.3.4.4 Viteza

Un alt parametru important este viteza, care se exprimă de obicei în conexiuni pe secundă (CPS) sau conexiuni actualizate pe secundă (CUPS). Viteza implementărilor hardware se poate exprima astfel:

$$CPS \approx \frac{\sum_i O_i}{T_S} \tag{3.7}$$

unde O_i reprezintă numărul de operaţii elementare ale blocului i iar T_S este din ecuaţia 2.2.

În cazul stocării ponderilor în memorii exterioare, numărul de pini limitează viteza la:

$$CPS < \frac{F_{CK} N_{P\max}}{B} \quad sau \quad CPS < \frac{F_{mem}}{b} \tag{3.8}$$

unde F_{mem}, exprimat în biţi pe secundă, reprezintă viteza cu care memoria poate furniza date.

3.4 Clasificarea implementărilor RNA

Clasificarea implementărilor RNA se poate face după mai multe criterii, două dintre ele fiind prezentate în continuare.

3.4.1 Scopul reconfigurării

Implementările în FPGA a reţelelor neuronale exploatează caracteristica de reconfigurabilitate a circuitelor FPGA. Această reconfigurabilitate poate fi exploatată în diverse moduri şi în funcţie de scopul cu care se face reconfigurarea aceasta există mai multe abordări:

Dezvoltarea şi simularea unei RNA, exploatează faptul că reţelele neuronale implementate în FPGA pot fi reconfigurate uşor şi rapid de un număr nelimitat de ori, ceea ce permite testarea diferitelor arhitecturi şi algoritmi de antrenare sau dovedirea unui concept. Un exemplu în acest sens este proiectul Ganglion [116] care permite implementarea unei reţele neuronale cu propagare înainte (FF) cu trei straturi. Implementarea este masiv paralelă şi atinge 20 milioane de conexiuni pe secundă (MCUPS) utilizând 640-784 blocuri logice configurabile pe neuron. Reţeaua utilizează 2 circuite FPGA Xilinx XC3090 şi o memorie PROM de 2k x 8 pe neuron oferind performanţe ridicate dar la o arie consumată mare.

Creşterea densităţii de elemente funcţionale pe dispozitiv, care se obţine prin reconfigurarea parţială a circuitului FPGA în timp real, în două moduri posibile. Prima modalitate permite multiplexarea în timp a resurselor circuitului FPGA pentru fiecare pas secvenţial din algoritmul unei RNA. De exemplu Eldredge în [93], prezintă o implementare a unei RNA în care algoritmul de antrenare backpropagation este împărţit în faza de propagare înainte şi cea de propagare înapoi a erorii executate pe acelaşi circuit FPGA multiplexat în timp.
A doua modalitate permite multiplexarea în timp a resurselor circuitului FPGA pentru fiecare reţea neuronală care este specializată cu un set de operanzi constanţi pentru diferite faze din timpul execuţiei. Această tehnică este cunoscută ca „dinamic constant folding". James-Roxby şi Blodget au implementat un perceptron multistrat 4-8-8-4, într-un dispozitiv Xilinx Virtex, folosind multiplicatoare cu coeficienţi constanţi reprezentând ponderile sinaptice [116]. Toate ponderile pot fi modificate prin reconfigurare dinamică în mai puţin de 69 μs. În cazul ambelor metode, rezultate bune se pot obţine dacă timpul de reconfigurare este mic în comparaţie cu timpul de calcul.

Adaptarea topologiei se referă la faptul că dispozitivele FPGA reconfigurabile dinamic permit implementarea RNA cu topologie modificabilă. Un exemplu în acest sens este prezentat de Perez-Uribe şi Sanchez în [10]. În timpul antrenării topologia şi precizia de calcul poate fi adaptată în funcţie de anumite criterii de învăţare.

3.4.2 Reprezentarea datelor

Multe cercetări atestă că este posibilă antrenarea RNA cu ponderi numere întregi. Interesul pentru utilizarea ponderilor numere întregi constă în faptul că multiplicatoarele pentru numere întregi se pot implementa mai eficient decât cele în virgulă mobilă. Există şi reprezentări ale ponderilor ca puteri ale lui doi. Acestea u avantajul operaţiile de multiplicare necesare într-o RNA pot fi efectuate printr-o serie de operaţii de şiftare. Există câteva încercări de implementare în FPGA a RNA cu ponderi reprezentate în virgulă mobilă, dar până în prezent nu există relatări despre implementări reuşite.

O reprezentare distinctă este cea numită „bit stream arithmetic" care utilizează secvenţe de numere binare în care numărul de biţi în starea unu logic împărţit la numărul total de biţi, reprezintă valoarea numărului real. Aceasta are avantajul înlocuirii operaţiilor de multiplicare şi acumulare cu operaţii logice simple [169].

3.5 Prezentarea implementărilor în FPGA a RNA

Prima implementare reuşită a unei reţele neuronale în circuite FPGA a fost publicată cu ceva mai mult de o decadă în urmă (Cox, C.E. and E. Blanz, GangLion - a fast field-programmable gate array implementation of a connectionist classifier. IEEE Journal of Solid-State Circuits, 1992).

De atunci interesul pentru implementările în FPGA a reţelelor neuronale artificiale a crescut foarte mult odată cu creşterea capacităţii circuitelor FPGA care ajung astăzi la densităţi de peste 10 milioane de porţi logice pe chip, lucrând la frecvenţe de sute de MHz.

Pentru a putea compara performanţele obţinute de diversele implementări existente în FPGA a RNA am stabilit câteva criterii comune de evaluare. Acestea ţin cont de parametrii enumeraţi în paragraful 3.3 şi de cele prezentate la începutul capitolului. Astfel pe lângă numele proiectului şi elementele de identificare a autorilor sau evidenţiat următoarele:

- Aplicaţia căreia i se dresează implementarea reţelei neuronale
- Arhitectura utilizată şi numărul de neuroni pe strat

- Precizia de reprezentare a datelor şi a ponderilor
- Tipul şi caracteristicile blocului MAC
- Modul de implementare a funcţiei de activare
- Gradul de paralelism implementat
- Modul cum se face antrenarea
- Performanţele obţinute
- Tipul dispozitivului FPGA utilizat
- Resursele utilizate de un neuron sau întreaga reţea.

Pe lângă implementările amintite în paragraful precedent se prezintă în continuare câteva din cele mai cunoscute implementări şi caracteristicile lor cele mai importante:

1. Hardware Implementation of a Feedforward Neural Network using FPGAs [18]

Autori	Aydoğan Savran, Serkan Ünsal
Instituţia	Ege University, Department of Electrical and Electronics Engineering
Aplicaţia	problema SAU exclusiv cu 3 intrări
Arhitectura	FF cu algoritm de antrenare backpropagation
Număr de neuroni / strat	3-5-1
Precizie date	8 biţi
Precizie ponderi	8 biţi
MAC	8x8, ieşire multiplicator 16 biţi, ieşire acumulator 16 biţi
Funcţia de activare	look-up table, 1024x8 ROM 10 biţi
Paralelism	neuron
Antrenare	off-chip, software în Matlab folosind NN Toolbox
Performanţe (CPS)	10 cicluri de ceas (10x20ns) pentru calculul răspunsului (20 multiplicări şi 20 de acumulări) => 10 nS/MAC ⇔ 100MCPS.
Dispozitiv FPGA utilizat	Spartan-II cu 200.000 porţi logice
Resurse utilizate	
- Neuron	12 slice-uri,
- Logica de control	12 slice-uri
- Funcţia de activare	2 BlockRAM
- RNA	91 slice-uri, 4 BlockRAM

2. A Fast FPGA Implementation of a General Purpose Neuron [216]

Autori	Valentin Salapura, Michael Gschwind, Oliver Maischberg
Instituţia	Institut fur Technische Informatik, Technische Universitat Wien
Arhitectura	FF şi recurente
Număr de neuroni / strat	4-4-2
Precizie date	8 biţi (numere întregi fără semn)
Precizie ponderi	8 biţi (numere întregi cu semn)
MAC	ieşire multiplicator 16 biţi cu semn, ieşire acumulator 20 biţi
Funcţia de activare	diverse, de la funcţia prag la sigmoid
Resurse / neuron	51 blocuri logice CLB, 1458 porţi echivalente

3. A Generic Building Block for Hopfield NNs with On-Chip Learning [150]

Autori	Michael Gschwind, Valentina Salapura, Oliver Maischberger
Instituţia	Institut fur Technische Informatik, Technische Universitat Wien
Arhitectura	RNA Hopfield
Număr de neuroni	64

Antrenare	on-chip
Performanţe (CPS)	25 MCPS.
Dispozitiv FPGA utilizat	Xilinx XC 4000
Resurse utilizate	
- Neuron	26 CLB
- RNA	1664 CLB

4. Artificial Neural Network Implementation on a Fine-Grained FPGA [169]

Autori	P. Lysaght, J Stockwood, J. Law, D. Girma
Instituţia	Communications Division, Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow
Aplicaţia	problema sau exclusiv
Arhitectura	pulse-stream
Număr de neuroni / strat	4
Precizie ponderi	4 biţi
Antrenare	off-chip
Performanţe (CPS)	0,77 MCPS.
Dispozitiv FPGA utilizat	Atmel AT6005
Resurse utilizate	
- de un strat (4 neuroni)	938 celule (din 3136)

5. Design and FPGA-Implementation of a Neural Network [75]

Autori	Henriette Ossoinig, Erwin Reisinger, Christian Steger, Reinhold Weiss
Instituţia	Institute for Technical Informatics, Graz University of Technology
Arhitectura	FF cu algoritm de antrenare backpropagation
Număr de neuroni / strat	3-3-1
Precizie date	8 biţi
Precizie ponderi	8 biţi
MAC	8x8, ieşire multiplicator 16 biţi, ieşire acumulator 18 biţi
Funcţia de activare	sigmoid implementat cu look-up table, pe 8 biţi
Paralelism	neuron
Antrenare	off-chip
Dispozitiv FPGA utilizat	4x Xilinx XC4013 (13.000 porţi echivalente) şi 1x XC 4005

6. Fast Neural Network Implementation (Placa ECX) [160]

Autor	Miroslav Skrbek
Instituţia	Department of Computer Science and Engineering, Czech Technical University Prague
Aplicaţia	XOR, recunoaşterea semnalului sonarului, şa.
Arhitectura	perceptron şi (RBF) reţele neuronale bazate pe funcţii radiale
Număr de neuroni / strat	60-140-10
Precizie date	8 sau 16 biţi
Precizie ponderi	8 sau 16 biţi
MAC	şiftare şi adunare
Funcţia de activare	aproximaţie liniară
Performanţe (CPS)	3,5 MCPS
Dispozitiv FPGA utilizat	2x Xilinx XC3090 respectiv XC4010

Resurse utilizate (pentru varianta cu XC4010)

	Max	BP8	BP8D	BP16	RBF8	RBF8D	RBF16
CLB	400	239	368	365	211	332	308
Blocuri I/O	61	57	57	57	57	57	57
Flip-flop-uri	800	130	175	160	124	165	154

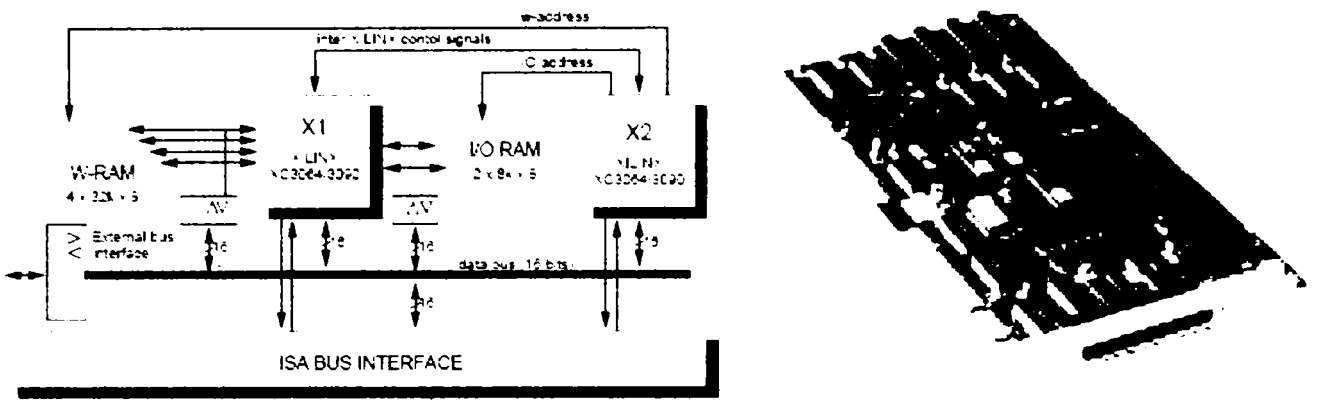


Figura 3.16 Schema bloc şi fotografia plăcii ECX

7. Run-Time Reconfiguration Artificial Neural Network [94], [95]

Autori	James G. Eldredge şi Brad L. Hutchings
Instituţia	Department of Electrical and Computer Engineering Brigham Young University
Arhitectura	FF-BP
Funcţia de activare	look-up table
Paralelism	neuron
Dispozitiv FPGA utilizat	Xilinx XC3090
Resurse utilizate	1-6 neuroni într-un FPGA

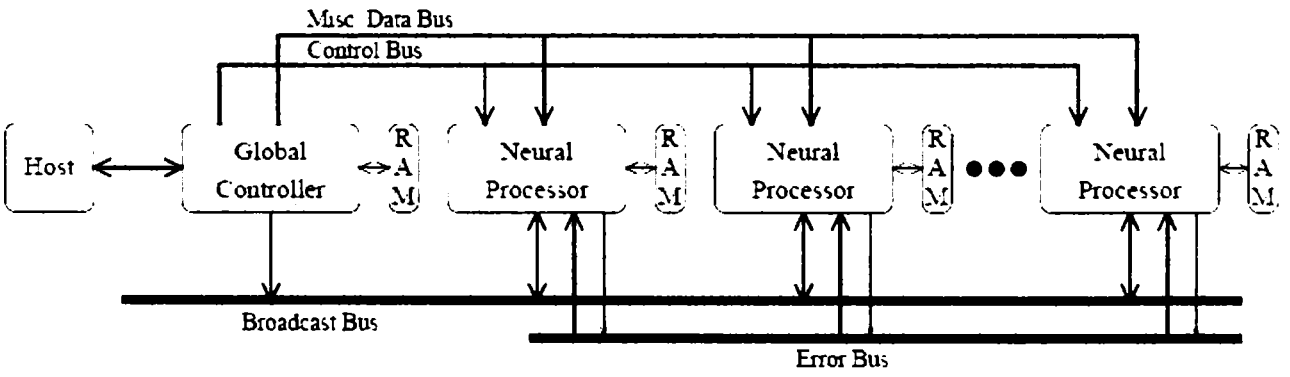


Figura 3.17 Arhitectura RRANN

Performanţele de antrenare ale circuitului sunt prezentate în figura 3.18 iar cele din faza de propagare în figura 3.19.

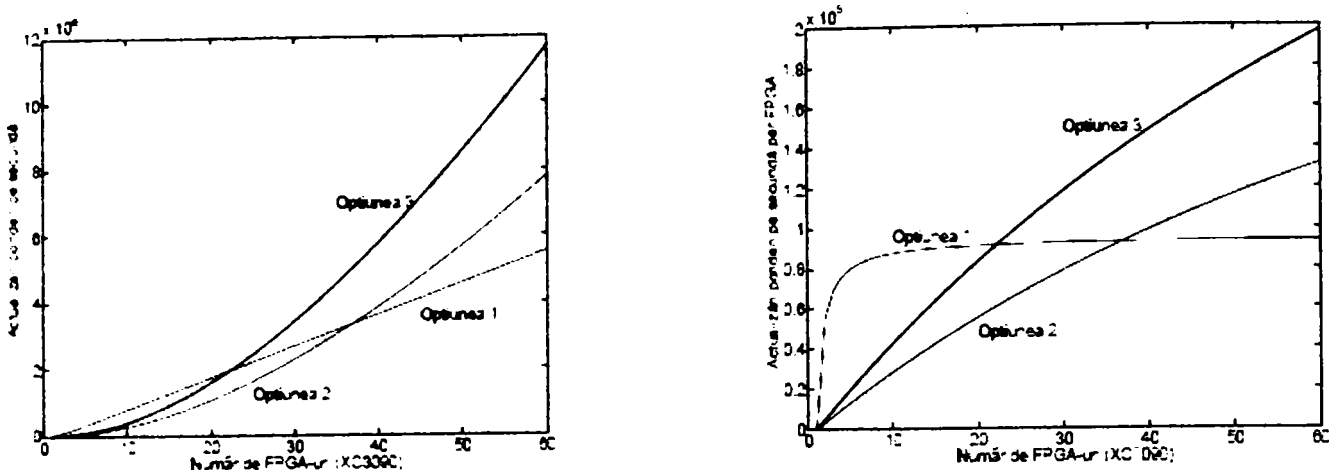


Figura 3.18 Performanțele la instruirea circuitului RRANN (CUPS)

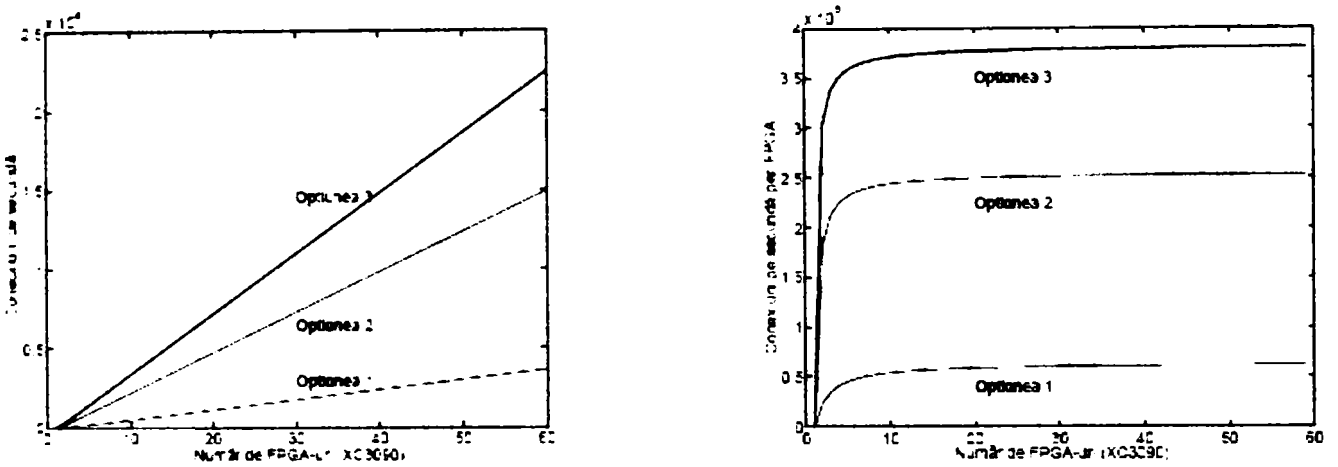


Figura 3.19 Performanțele circuitului RRANN (CPS)

8. FPGA Implementation of Very Large Associative Memories [41]

Autori Dan Hammerstrom, Changjian Gao, Shaojuan Zhu, Mike Butts

Instituția OGI School of Science and Engineering, Oregon Health and Science University

Dispozitiv FPGA utilizat Spartan-II

Resurse utilizate Placa acceleratoare: *Relogix Mini-Mirax Memory-Intensive Reconfigurable Accelerator* compusă dintr-o pereche FPGA+SDRAM

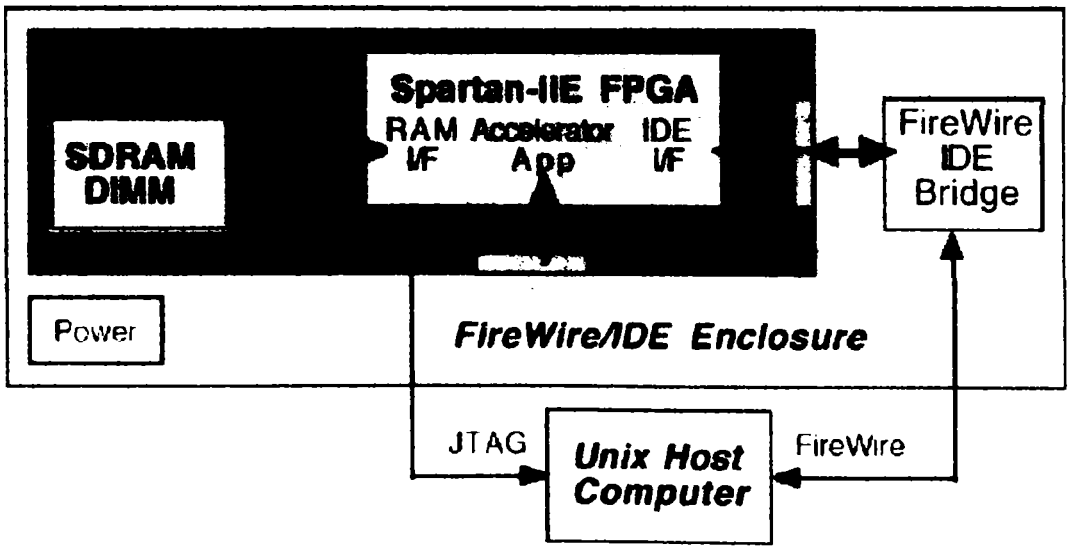


Figura 3.20 Placa Mini-Mirax

Performanțe (CPS):

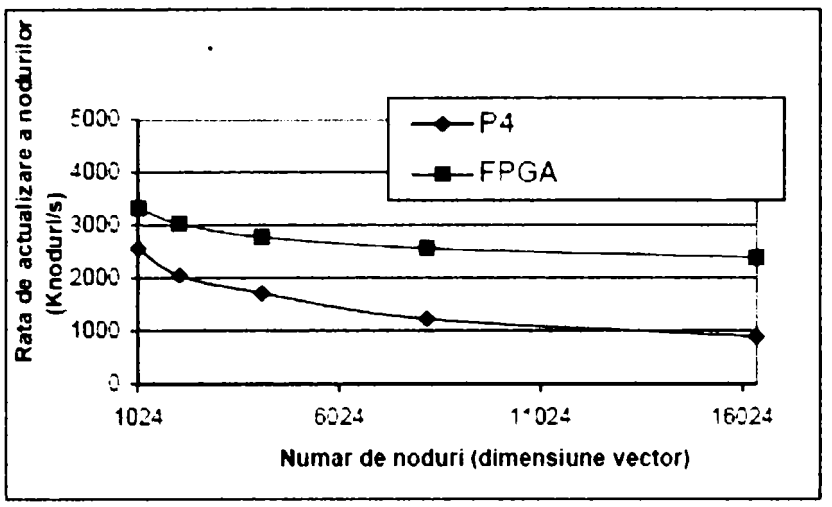


Figura 3.21 Comparație între rata de actualizare a nodurilor pentru P4 și FPGA

9. FPGA Implementation of an Adaptable-Size Neural Network [10]

Autori	Andres Perez-Uribe și Eduardo Sanchez
Instituția	Logic Systems Laboratory, Swiss Federal Institute of Technology
Arhitectura	FF cu 2 straturi
Număr de neuroni / strat	adaptabil
Precizie date	8 biți
Dispozitiv FPGA utilizat	Xilinx XC4013
Resurse utilizate	un microcontroler 68331, 4 circuite FPGA XC4013 și 4 dispozitive de interconecare programabile I-Cube 320

3.6 Concluzii

Cel mai mare avantaj al implementării RNA în FPGA rezidă în viteza mare de procesare care poate fi obținută datorită arhitecturii masiv paralele a circuitelor FPGA.

Alte avantaje ale implementării în FPGA a RNA sunt :

- flexibilitatea dată de posibilitatea reprogramării (reconfigurării) ceea ce permite experimentarea diferitelor arhitecturi
- principii de proiectare clare și unelte puternice de proiectare a circuitelor digitale (VHDL)
- stocarea ponderilor nu este o problemă
- sensibilitate mică la zgomote și față de temperatură

Implementarea aritmeticii în virgulă mobilă (utilizată mai ales în faza de învățare) în FPGA este prea costisitoare. De aceea faza de învățare se pretează a fi implementată software, folosind programe dedicate (Matlab sau Neural Network Tolbox pentru Simulink), putându-se utiliza reprezentarea în virgulă mobilă ceea ce duce la convergența și acuratețea rețelei. În cazul folosirii unei precizii ridicate în faza de propagare, scade viteza datorită numărului mare de operații de multiplicare-acumulare care trebuiesc executate. Precizia ponderilor în faza de propagare poate fi micșorată față de precizia calculată în faza de învățare, erorile rezultate fiind neglijabile [173]. Utilizând circuite dedicate (FPGA) operațiile de multiplicare-acumulare pot fi executate în paralel. În software și în DSP acest paralelism este imposibil iar execuția este secvențială.

Capitolul IV

Mediul integrat hardware – software pentru implementarea RNA

În mod tradiţional reţelele neuronale artificiale sunt implementate software, folosind limbaje de programare, pentru utilizare off-line sau on-line (în timp real) la viteze medii. În cazul în care este nevoie de viteze de lucru ridicate sau implementarea ca bloc component a unui sistem dedicat este necesară implementarea RNA direct în circuite reconfigurabile de tipul FPGA. Aceste dispozitive combină avantajul reprogramabilităţii cu creşterea vitezei, datorată calculului masiv paralel ce rezultă din arhitectura lor.

În ultimii ani au fost dezvoltate diferite tehnici şi arhitecturi pentru implementarea RNA folosind circuitele FPGA. Unele implementează direct întreaga reţea neuronală, inclusiv funcţia de activare şi folosesc procesarea paralelă în scopul obţinerii de viteze ridicate cu preţul utilizării creşterii resurselor utilizate. Altele utilizează multiplexarea în timp a resurselor (cum ar fi multiplicatoarele) pentru a elimina problemele legate de resurse cu preţul scăderii vitezei.

Dezvoltările recente în sinteza circuitelor digitale au automatizat în mare măsură procesul de implementare în ASIC sau FPGA a unor algoritmi specifici procesării digitale a semnalelor precum şi a reţelelor neuronale artificiale. În mod tradiţional aceste circuite erau dezvoltate folosind limbaje de descriere hardware cum ar fi VHDL sau Verilog, dar acestea oferă un suport insuficient pentru descrierea comportamentului concurenţial (paralel) al RNA [79]. Apariţia uneltelor de sinteză bazate pe extensii ale limbajului de programare C cum ar fi Handel-C facilitează modelarea comportamentului concurenţial al RNA. Totuşi aceste limbaje au capabilităţi de sinteză limitate şi modele slab dezvoltate de simulare în Matlab şi Simulink. Aceste probleme au fost depăşite de recenta introducere a unor unelte de sinteză care, dintr-o descriere comportamentală în Matlab Simulink a unui circuit de procesare digitală a semnalelor, realizează sinteza directă a unui circuit digital adecvat (folosind limbajul VHDL ca limbaj intermediar). Mai exact modelul Matlab Simulink este conceput folosind toolboxul Xilinx Blockset (care este comparabil în linii mari cu un subset al toolboxului Matlab DSP), în continuare modelul este simulat în Simulink şi apoi transformat într-o descriere în limbaj VHDL folosind utilitarul Xilinx System Generator. Proiectul este apoi sintetizat prin procedeul obişnuit pentru descrierile VHDL şi implementat în FPGA ca un circuit de prelucrare digitală a semnalelor specific aplicaţiei

Acest capitol prezintă o nouă metodă concepută şi testată în întregime de autor [186], [187], [192], pentru implementarea reţelelor neuronale artificiale în circuite logice programabile FPGA. Metoda permite proiectanţilor sistemului să se concentreze asupra specificaţiilor de nivel înalt, şi permite un răspuns rapid cu privire la performanţele şi costurile arhitecturii şi algoritmului de antrenare ales. Printre aplicaţiile posibile sunt dezvoltarea unor senzori inteligenţi cu capacitate de învăţare şi comportament adaptiv bazaţi pe reţele neuronale, pentru orice dispozitiv industrial sau casnic precum şi a unor periferice inteligente care să permită oamenilor cu orice tip de handicap să folosească calculatorul şi să comunice [193], [194].

4.1 Etapele proiectării RNA

Până în trecutul foarte apropiat proiectanţii de reţele neuronale artificiale care doreau să implementeze în FPGA dispuneau doar de metoda de dezvoltare în doi paşi prezentată în figura 4.1.

Proiectantul reţelei neuronale poate modela şi simula reţeaua în Matlab/Simulink dar poate să aibă cunoştinţe insuficiente despre implementarea în FPGA pentru a exploata avantajele arhitecturii FPGA şi a evita o implementare greşită.

După o modelare reuşită în Simulink trebuie proiectat sistemul în VHDL, etapă care poate fi executată de un proiectant FPGA care poate să nu fie un expert în RNA iar implementarea să nu funcţioneze exact cum a fost conceput de către proiectantul RNA. Chiar dacă transpunerea în cod VHDL este făcută de însuşi proiectantul RNA, aceasta necesită cel puţin acelaşi timp ca şi proiectarea la nivel înalt şi implică şi riscul unei transpuneri incorecte.

De asemenea nu există posibilitatea de a co-simula sistemul astfel încât eventualele probleme pot fi observate foarte târziu, abia după implementare şi testarea circuitului. Reproiectarea înseamnă timp pierdut şi costuri suplimentare.

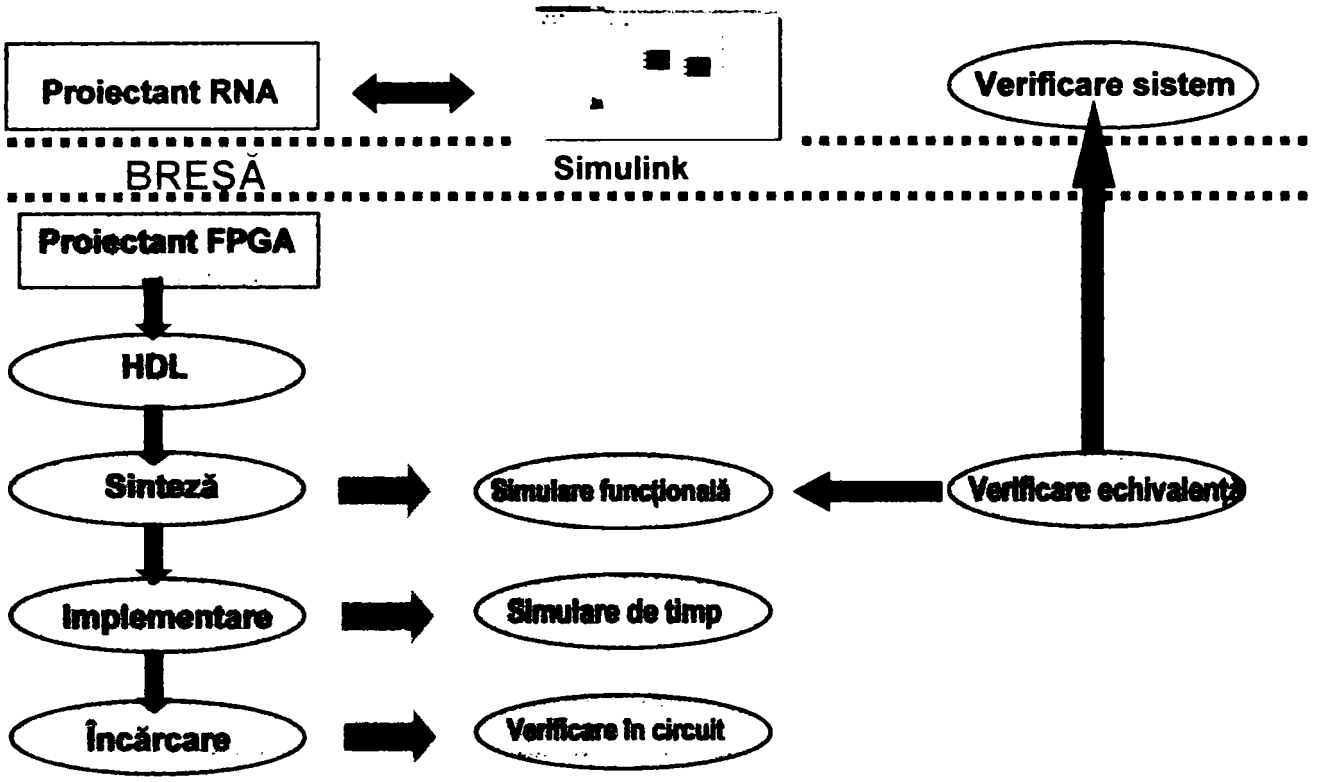


Figura 4.1 Metodologia tradiţională de proiectare a RNA

O transformare automată şi exactă a modelului de nivel înalt din Simulink într-o descriere VHDL direct sintetizabilă şi implementabilă în FPGA este esenţială şi poate fi realizată folosind utilitarul System Generator, dezvoltat recent de către firma Xilinx pentru implementarea circuitelor de procesare digitală a semnalelor. Acesta este o extensie pentru Simulink care adaugă biblioteca Xilinx Blockset, bibliotecilor din Simulink, şi integrează etapele proiectării DSP de la descrierea de nivel înalt până la fişierul de configurare a FPGA.

Metoda concepută de autorul prezentei teze foloseşte System Generator pentru a oferi proiectanţilor de RNA o metodologie şi un mediu de proiectare familiar şi uşor de folosit (Matlab/Simulink), care permite implementarea directă a reţelelor neuronale în FPGA. Principalele obiective propuse pentru acest proiect sunt:

- Dezvoltarea de blocuri specifice pentru RNA în Simulink folosind Xilinx blockset;
- Implementarea facilă în hardware a diverse RNA în faza de cercetare dezvoltare, în scopul determinării arhitecturii optime, şi al algoritmului care se pretează cel mai bine la aplicaţia dezvoltată;

- Dezvoltarea de sisteme dedicate noi bazate pe reţele neuronale implementate în circuite logice programabile FPGA.

Noua metodologie de proiectare a RNA folosind programul System Generator şi blocuri specifice RNA dezvoltate de autor şi/sau blocuri din biblioteca Xilinx Blockset este prezentată în figura 4.2. Ea va fi descrisă pe larg în paragraful 4.4.

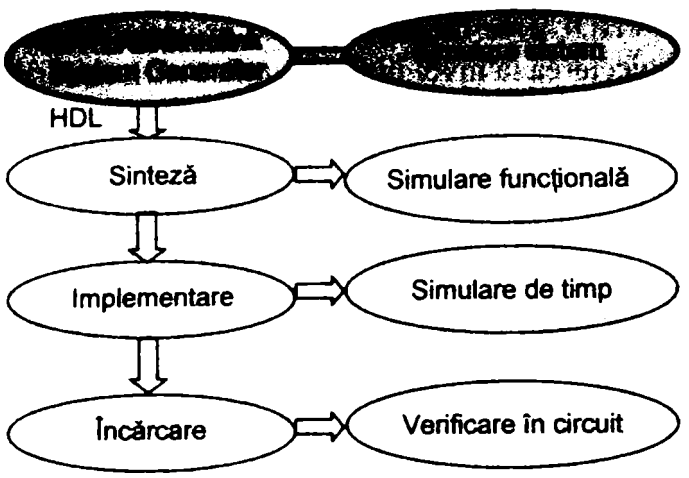


Figura 4.2 Metodologia de proiectare a RNA folosind System Generator

4.2 Structura hardware

Platforma hardware a mediului integrat dezvoltat de autor trebuie să asigure suportul necesar pentru proiectarea şi implementarea reţelelor neuronale [1], [2]. Ea trebuie să asigure datele necesare în faza de antrenare, şi respectiv fluxul de date în faza de propagare. Principalele părţi componente ale platformei sunt:

- Sursă de date analogice şi/sau digitale
- Sistem de achiziţie de date
- Calculator gazdă
- Sistem de dezvoltare cu circuite FPGA

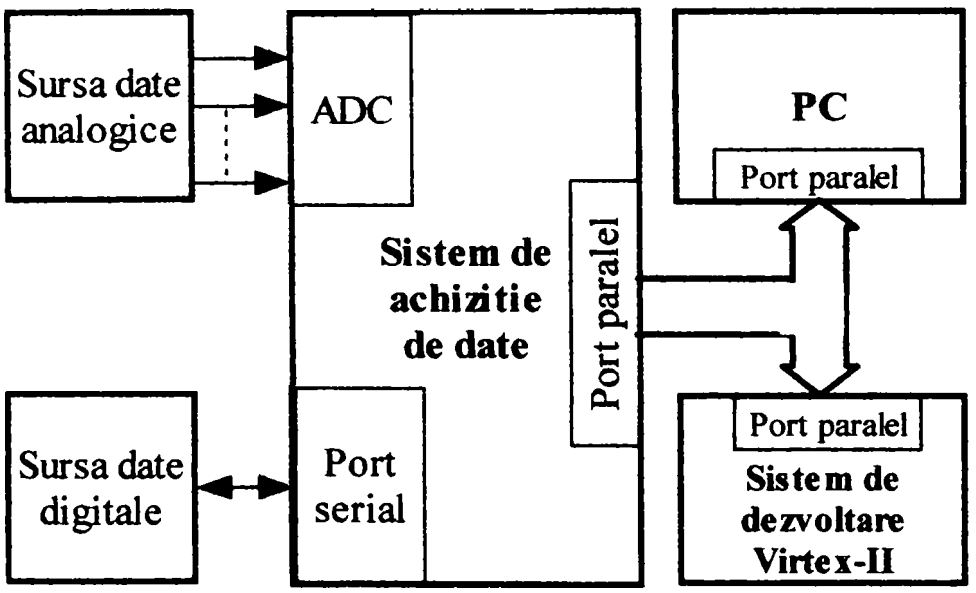


Figura 4.3 Platforma hardware pentru implementarea RNA

Platforma a fost astfel concepută încât să poată fi folosită împreună cu diverse surse de date analogice sau digitale. De aceea sistemul de achiziţie a datelor permite preluarea datelor digitale de pe un port serial precum şi a celor analogice prin intermediul unui convertor

analog digital. De asemenea sistemul de achiziţie a datelor este prevăzut cu un port paralel prin care trimite date la un calculator personal şi/sau la sistemul de dezvoltare cu circuite FPGA.

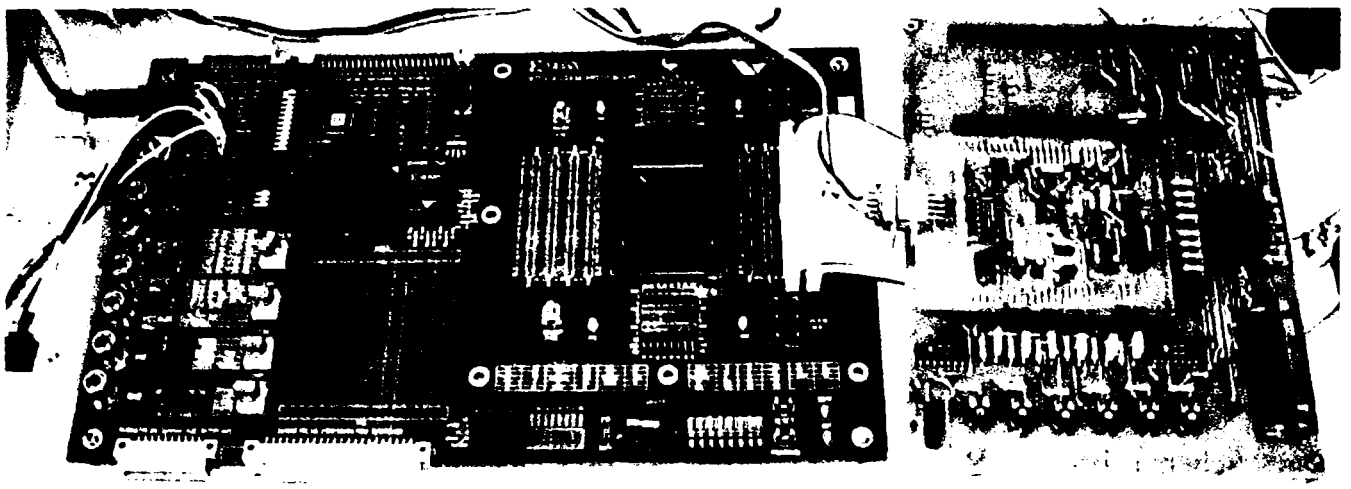


Figura 4.4 Fotografia platformei

4.2.1 Sistemul de achiziţie de date

Sistemul de achiziţie de date este construit în jurul unui microconvertor ADuC 812BS.

Microconvertorul, produs de Analog Devices, este un sistem de achiziţii de date integrat, compus dintr-un nucleu de microprocesor pe 8 biţi de tip 8051, prevăzut cu un convertor analog-digital pe 12 biţi. Sistemul are 8k Bytes de memorie program Flash/EE, programabilă în sistem, 640k Bytes de memorie Flash/EE de date, 256 Bytes de memorie de date SRAM. De asemenea poate accesa 16M Bytes de memorie de date externă şi 64k Bytes de memorie de program externă.

Caracteristicile acestui microconvertor sunt:

- 8 intrări analogice multiplexate folosind un convertor analog-digital pe 12 biţi
- 2 convertoare digital-analogice pe 12 biţi
- Senzor de temperatură integrat
- Controler DMA pentru capturi de mare viteză de la ADC-uri în RAM
- Frecvenţa nominală de 12MHz
- 3 numărătoare/temporizatoare pe 16 biţi
- 32 linii de intrare/ieşire programabile
- Capacitate de curent mare pe portul 3
- 9 surse de întreruperi cu 2 nivele de prioritate
- Alimentare la 3V sau 5V
- Funcţionare în 3 moduri: Normal, Inactiv sau cu consum redus
- Port serial UART de intrare/ieşire
- Port I^2C şi SPI
- Watchdog
- Monitor pentru sursa de alimentare

Schema bloc a microconvertorului ADuC812 este prezentată în figura 4.5

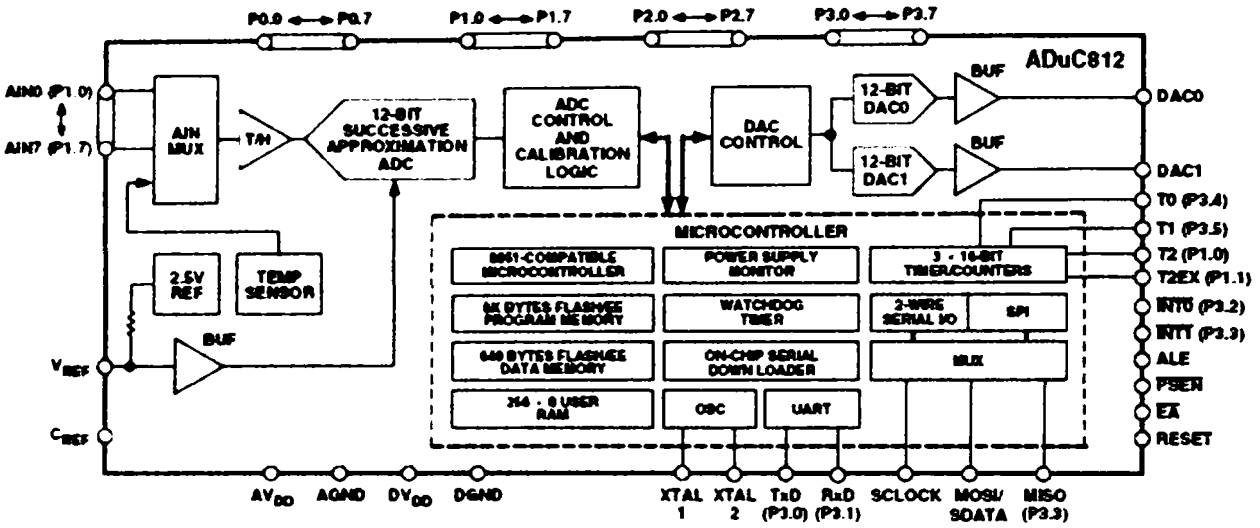


Figura 4.5 Blocurile funcționale ale microconvertorului AduC812BS

Schema sistemului de achiziție realizat este prezentată în figura 4.6.

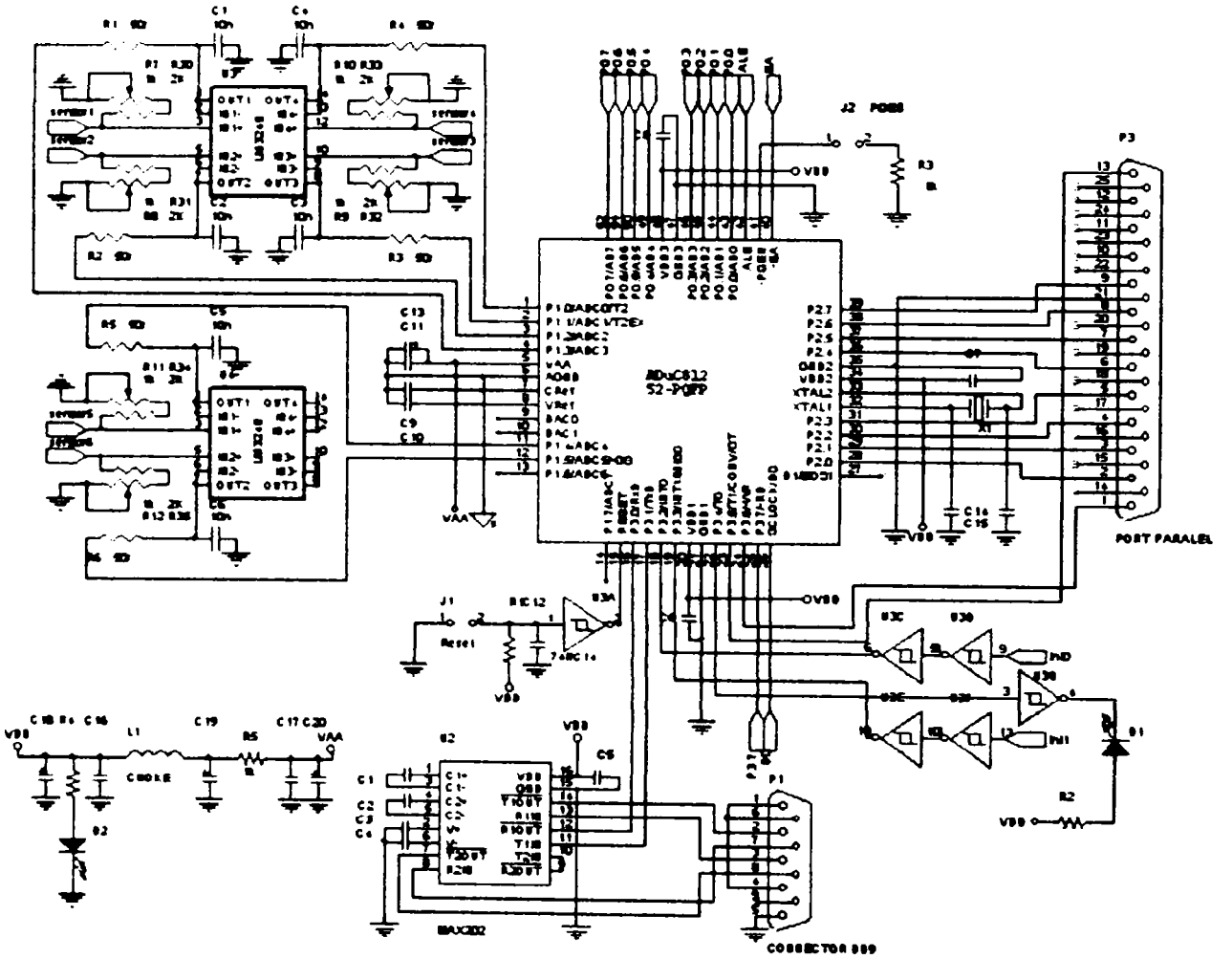


Figura 4.6 Schema electrică a sistemului de achiziție folosind modulul ADuC812

Datele digitale pot fi achiziționate prin portul serial UART standard al microconvertorului căruia i s-a atașat un circuit MAX202 pentru adaptarea de nivel TTL/RS232.

Datele analogice se pot achiziționa prin intermediul celor opt canale ale convertorului analog digital (ADC) cu aproximații succesive pe 12 biți. Senzorii utilizați în experimentele efectuate au fost conectați la microconvertor prin intermediul unor circuite de condiționare.

Portul paralel asigură transmisia datelor achiziţionate spre calculator în faza antrenare a reţelelor neuronale artificiale, respectiv spre sistemul de dezvoltare cu FPGA în faza de programare. Protocolul implementat este un protocol asincron cu confirmare şi condiţionare totală care pe lângă cei 8 biţi de date mai foloseşte şi două semnale de control, *rdy* şi *ack*. Codul care rulează în microcontroler nu face diferenţă între PC sau alt dispozitiv hardware conectat la portul lui paralel. Atâta timp cât protocolul de comunicaţie este respectat, el va furniza date indiferent cine le solicită.

Pentru transferul datelor spre calculator se utilizează un script Matlab care rulează pe calculatorul gazdă. Codul prezentat în Anexa 1 foloseşte utilitarul Data Acquisition Toolbox şi permite citirea datelor de la sistemul de achiziţie şi preluarea lor în Workspace sub forma unei matrici de vectori de date. Acestea se salvează într-un fişier care este folosit de Neural Network Toolbox în faza de antrenare.

Pentru transferul datelor la sistemul de dezvoltare cu FPGA, în circuitul FPGA se implementează pe lângă circuitele neuronale şi un port paralel cu acelaşi protocol pentru a asigura datele în faza de propagare.

Codul sursă pentru microconvertor scris în limbajul C şi compilat cu Keil uVision2 este prezentat în Anexa 2. Microconvertorul este programabil în sistem prin portul serial.

4.2.2 Sistemul de dezvoltare cu circuite programabile FPGA

Structura HW a întregului sistem a fost concepută modular ca să ofere un mediu de dezvoltare uşor şi flexibil. Astfel, ca sistem de dezvoltare cu circuite FPGA, poate fi folosită orice placă disponibilă, în funcţie de numărul de neuroni ce se doresc a fi implementaţi. Pe parcursul experimentelor făcute au fost utilizate mai multe plăci de dezvoltare comerciale precum şi plăci proiectate şi realizate de către autor [190], [191], [46]. Placa de dezvoltare care permite implementarea unui număr mai mare de neuroni este un sistem de dezvoltare comercial, care conţine circuite Virtex-II cu 1.000.000 porţi logice echivalente, realizată de firma Xilinx în scopul dezvoltării circuitelor prototip.

Platforma prototip conţine două circuite FPGA, unul notat DUT (Device Under Test), este disponibil pentru implementarea circuitelor concepute de utilizator, iar cel de al doilea, numit Service FPGA, are rolul de a face conexiunile între resursele plăcii.

Principalele facilităţi oferite de această platformă sunt:

- Posibilitatea alimentării de la trei surse independente V_{CCINT}, V_{AUX} şi V_{CCO}
- Posibilitatea selectării V_{CCO} pentru fiecare banc de I/O
- Porturi de configurare MultiLINX sau Parallel Cable III
- Patru intrări de ceas global, două prin conectori SMB 50Ω, două socluri pentru oscilatoare tip LVTTL
- Oscilator programabil pe placă
- Frecvenţă de ceas selectabilă (de la 25 MHz la 90 MHz)
- Soclu pentru PROM de configurare
- Port JTAG pentru programarea circuitului FPGA utilizator (DUT) precum şi a circuitelor PROM XC17Vxx şi XC18Vxx
- Conectoare care permit interconectarea mai multor platforme
- Zona pentru dezvoltarea de circuite prototip permite lipirea unor conectoare injectarea sau extragerea unor semnale

Schema bloc a platformei prototip Virtex-II este prezentată în figura 4.7.

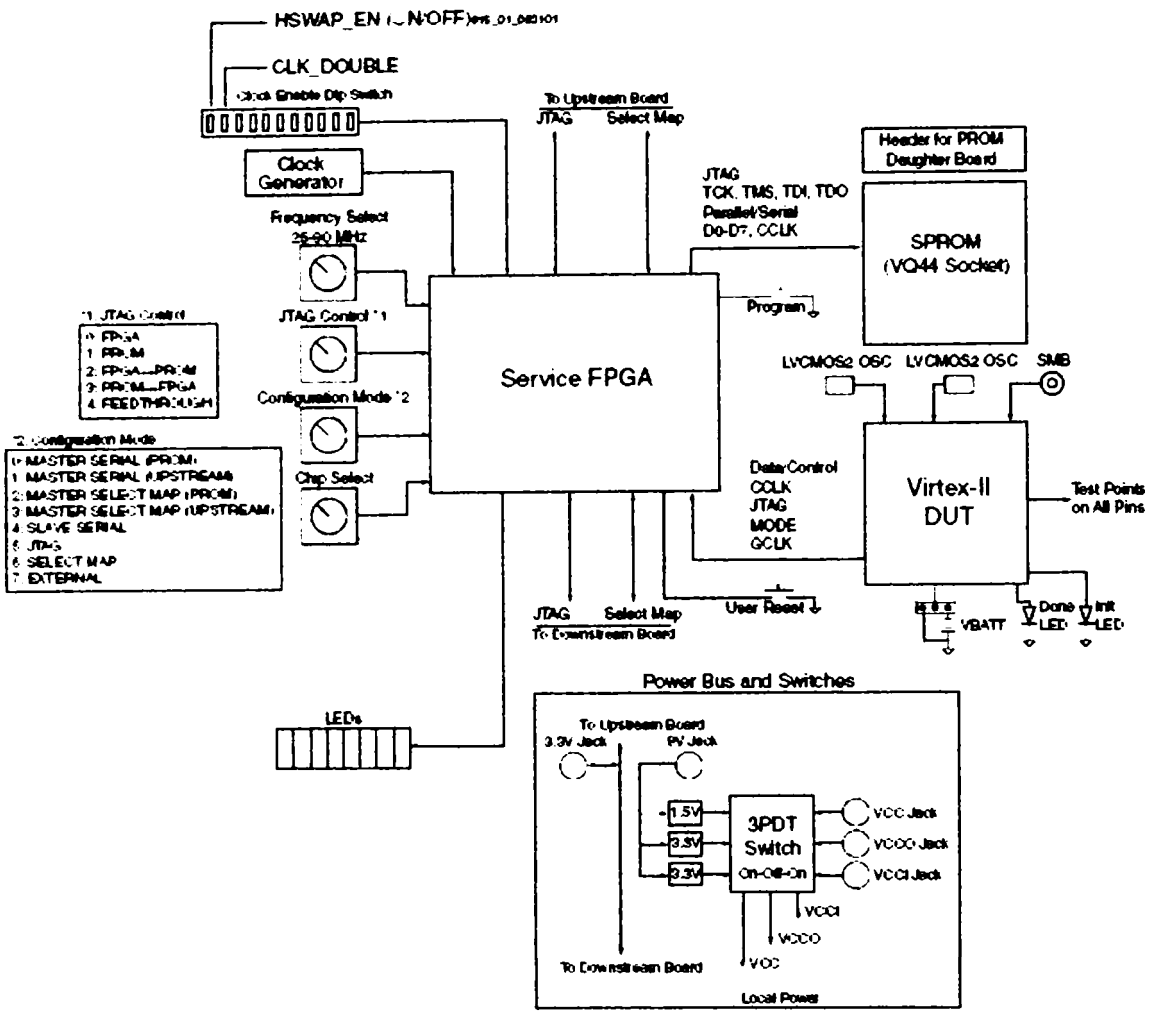


Figura 4.7 Schema bloc a platformei prototip Virtex-II

Circuitul FPGA poate fi configurat direct din mediul Xilinx ISE, sau un alt program similar, prin cablul conectat la portul paralel al calculatorului.

Circuitul XC2V1000 este componenta principală a sistemului de dezvoltare. El este conţinut într-o capsulă cu 456 pini, de tip BGA. El este alcătuit dintr-o arie de 40x32 blocuri logice configurabile conţinând 5120 slice-uri, fiind echivalent cu 1.000.000 porţi logice. Alte caracteristici notabile ale circuitului sunt:

- Memorie RAM distribuită, implementabilă în CLB: 160 Kbiţi
- 40 blocuri multiplicatoare de 18x18 biţi
- Memorie bloc SelectRAM 40 blocuri de 18 Kbiţi, adică un total de 720 Kbiţi
- Număr maxim de intrări/ieşiri disponibile utilizatorului: 432
- 8 blocuri Digital Clock Manager
- Blocurile de I/O suportă o mare varietate de standarde de intrare-ieşire asimetrice şi diferenţiale.

4.3 Structura software a mediului de implementare a RNA

Arhitectura software a mediului integrat HW/SW pentru implementarea RNA permite pe de o parte implementare părţii software a aplicaţiei, care constă în antrenarea RNA, şi pe de altă parte proiectarea părţii hardware a acesteia, adică implementarea RNA în FPGA, pentru faza de propagare. O simplă enumerare a resurselor software întrebuinţate pentru dezvoltarea mediului integrat HW/SW arată complexitatea acesteia:

- Keil uVision2 – dezvoltarea programului pentru microconvertorul ADuC 812 din componenţa SAD
- Windows Serial Downloader (WSD v6.04) – pentru încărcarea programului în microconvertor
- Matlab – mediul de programare utilizat pentru modelarea RNA
  - Data Acquisition Toolbox – extensia Matlab pentru achiziţia semnalelor
  - NTPort Library 2.5 – utilitar utilizat pentru controlul porturilor PC din Matlab
  - Neural Network Toolbox – extensia Matlab utilizată pentru antrenarea RNA
  - Simulink – pentru simularea RNA
  - System Generator – rulând în cadrul programului Simulink, pentru:
    - modelarea cu blocuri Xilinx a RNA
    - generarea fişierelor proiect Xilinx
    - generarea codului VHDL sintetizabil
    - generarea unui fişier Testbench în limbaj VHDL pentru simularea RNA
    - generarea fişierelor de constrângere
- Xilinx ISE – mediul integrat pentru sinteza şi implementarea RNA în FPGA
  - ModelSim XE – permite simularea în diverse etape a reţelei de implementat
  - Impact – folosit pentru încărcarea fişierului de configurare în FPGA
  - ChipScope – utilizat în faza de depanare a proiectului, permite vizualizarea circuitului implementat în FPGA
- VHDL – folosit ca limbaj intermediar între Simulink şi Xilinx, precum şi pentru includerea unor blocuri de tip proprietate intelectuală (IP) în modelul RNA

Dintre cele de mai sus primele două programe se utilizează numai în faza de dezvoltare a platformei HW/SW, în timp ce restul programelor sunt utilizate în faza de implementare HW/SW a RNA. În timp ce programul Xilinx ISE este folosit numai pentru implementarea hardware a fazei de propagare a RNA, mediul Matlab este folosit atât pentru achiziţia datelor de antrenare (folosind Data Acquisition Toolbox), implementarea software a fazei de antrenare a RNA (Neural Network Toolbox) cât şi pentru modelarea şi simularea RNA precum şi pentru generarea codului VHDL sintetizabil (folosind System Generator ca extensie pentru Simulink).

4.3.1 Matlab/Simulink

Primul program implicat în implementarea HW/SW a unei reţele neuronale, Matlab, este un limbaj de programare, şi un mediu de proiectare la nivel înalt. Fiind un program familiar unui număr mare de ingineri în general, şi proiectanţilor de RNA în particular, şi beneficiind de o arhitectură deschisă este o platformă ideală pentru dezvoltarea de unelte de proiectare la nivel înalt a reţelelor neuronale. Utilizatorul poate să dezvolte funcţii specifice aplicaţiei cu fişiere de tip .m (Matlab) şi funcţii-S (Simulink). Mediul Matlab oferă proiectantului RNA avantajul utilizării unui număr mare de biblioteci de funcţii matematice, reţele neuronale, procesare de semnal, etc. De exemplu Neural Network Toolbox conţine un ansamblu de funcţii Matlab care implementează majoritatea arhitecturilor şi a algoritmilor de

instruire a reţelelor neuronale. Funcţiile sunt flexibile şi permit proiectantului să specifice arhitectura, funcţia de transfer, modul de conectare între unităţi, şi algoritmul de instruire.

Simulink este un mediu de modelare şi simulare a sistemelor dinamice complet integrat în programul Matlab. El dispune de obiecte specifice diverselor domenii, (numite blocuri) cu ajutorul cărora se poate construi modelul. Editorul grafic este folosit pentru realizarea modelului şi accesul la motorul de simulare. Simulink suportă simularea comportamentului concurenţial al sistemelor ceea ce îl face potrivit pentru modelarea de circuite hardware de mare performanţă.

De asemenea este posibilă interfaţarea între Matlab şi Simulink, ceea ce permite utilizatorului să includă fişiere .m în modelul Simulink, să importe date din Workspace (mediul de lucru Matlab) şi să salveze date în Workspace.

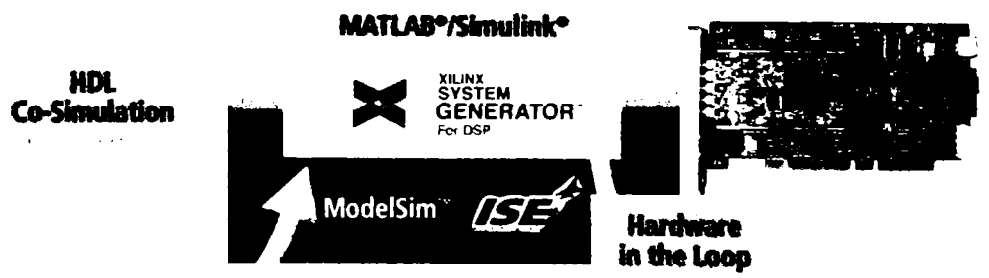
Toate facilităţile prezentate mai sus, sunt exploatate de metoda de implementare a RNA dezvoltată de autor.

4.3.2 System Generator

System Generator este o extensie a programului Simulink care îi adaugă un set de biblioteci de blocuri Xilinx implementabile in hardware. Fiecare bloc este „bit and cycle true", ceea ce înseamnă că ele se comportă identic la nivel de bit şi ciclu în Simulink şi în hardware. În Simulink aceste blocuri pot fi combinate cu blocuri din alte biblioteci neimplementabile în FPGA. Parametrii blocurilor Xilinx, ca şi a celorlalte blocuri pot fi setate de utilizator. System Generator transferă în hardware numai porţiunea din model creată cu blocuri Xilinx. Comportamentul porţiunii cu blocuri non-Xilinx poate fi surprins într-un fişier HDL testbench care foloseşte vectorii de simulare din Simulink. Aceasta permite comparaţia între modelul din Simulink şi implementarea hardware.

System Generator reprezintă un proces de proiectare integrat de la reprezentarea sistemului în Simulink la fişierul de configurare a circuitului FPGA. El asigură legătura între diferite tehnologii cum ar fi suita Matlab pentru proiectarea sistemului, mediul software integrat Xilinx ISE pentru implementarea lui şi programul ModelSim pentru simulare. Programul permite co-simularea sistemului modelat în Simulink împreună cu aşa numitele „cutii negre" (black box) care încorporează modele HDL. Procesul se numeşte „HDL Co-Simulation". De asemenea în simulare pot fi încorporate şi plăci hardware, ceea ce permite testarea hardware şi duce la accelerarea simulării. Acest tip de simulare poartă numele de „Hardware in the Loop Co-Simulation" [220].

Simulare sistem RNA



Implementare

Figura 4.8 System Generator ca legătură între diverse tehnologii

4.4 Etapele implementării hardware-software a RNA

Mediul integrat HW/SW a fost conceput de autor ca o platformă de dezvoltare pentru circuite prototip folosind circuite programabile (FPGA) şi un microcontroler. Etapele implementării reţelelor neuronale artificiale folosind mediul integrat HW-SW sunt prezentate în figura 4.9. Platforma este conectată la portul paralel al unui calculator personal pe care rulează programele Matlab/Simulink, System Generator, ModelSim, Xilinx ISE şi altele. Microcontrolerul este folosit pentru implementarea sistemului de achiziţie de date şi adaptarea semnalelor de intrare la cerinţele intrărilor reţelelor neuronale. Circuitul programabil Xilinx FPGA de tipul Virtex-II XC2V1000 este folosit pentru implementarea reţelelor neuronale. Calculatorul este folosit pentru implementarea părţii software a aplicaţiei, şi proiectarea părţii hardware. Astfel programul Matlab prin toolboxul Data Acquisition controlează achiziţia datelor de antrenare a reţelei neuronale, şi permite antrenarea reţelelor neuronale prin toolboxul Neural Network.

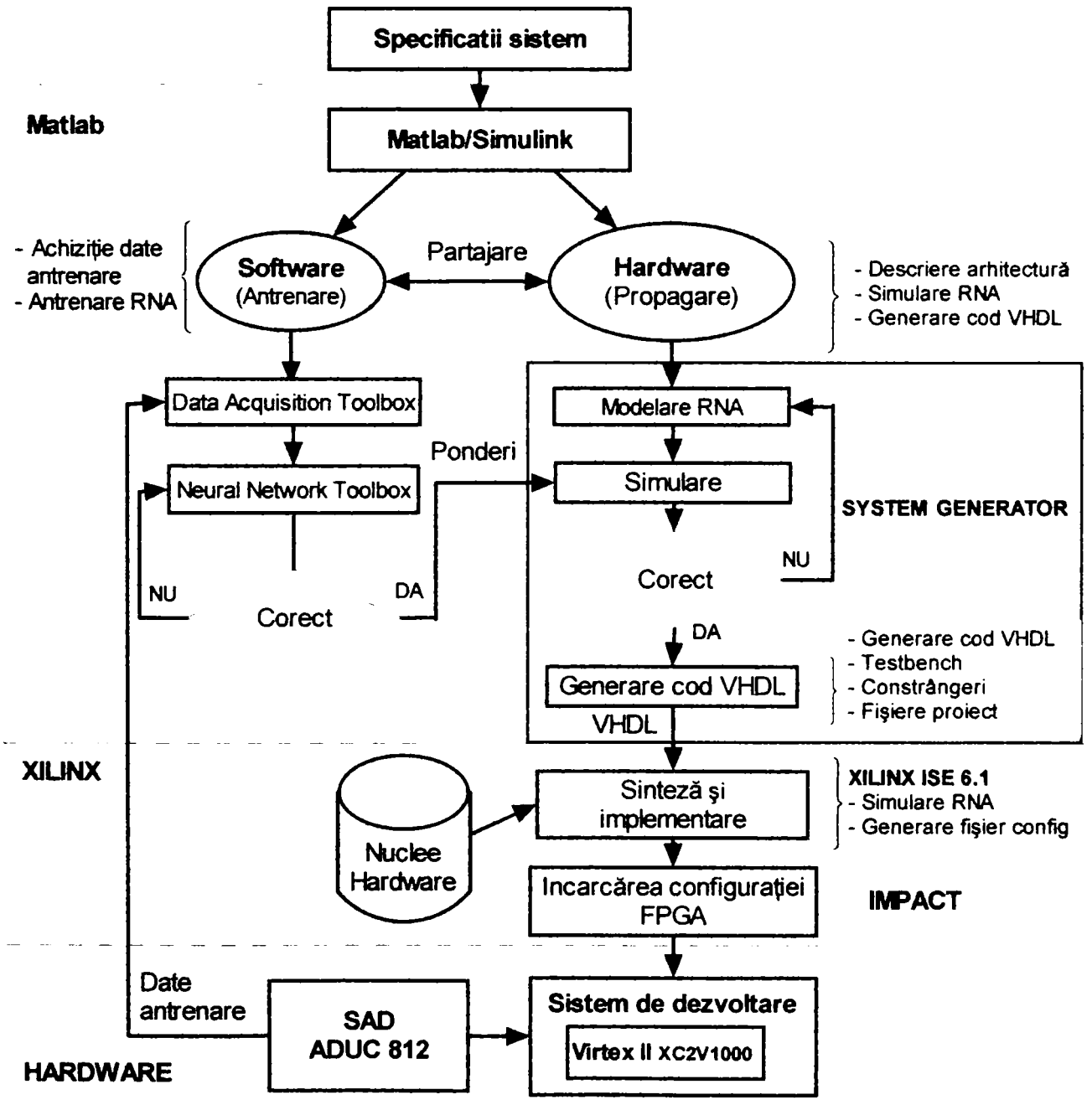


Figura 4.9 Mediul integrat HW – SW pentru implementarea RNA

Programul Simulink care rulează în mediul Matlab folosit în general pentru modelarea, simularea şi analiza sistemelor dinamice, permite de asemenea modelarea şi simularea diverselor arhitecturi de reţele neuronale cu ajutorul extensiei System Generator create de firma Xilinx şi a unor blocuri IP create de autor. După identificarea configuraţiei optime pentru aplicaţia dată, System Generator transformă reprezentarea de nivel înalt a RNA într-un cod VHDL sintetizabil. Mediul Xilinx ISE asigură sinteza şi implementarea codului VHDL în dispozitivul FPGA ales, iar utilitarul Impact permite încărcarea fişierului de configurare în FPGA.

Ramura din stânga a diagramei reprezintă implementarea software. Ramura din dreapta prezintă etapele implementării hardware folosind blocuri specifice reţelelor neuronale artificiale dezvoltate de autor şi/sau alte blocuri din biblioteca Xilinx Blockset, simularea acestui model şi generarea codului VHDL.

4.4.1 Implementarea software a fazei de antrenare

4.4.1.1 Achiziţia datelor

Sistemul de achiziţie de date este conectat prin intermediul portului paralel la calculator pe care rulează programul Matlab şi programul de achiziţie creat de autor folosind Data Acquisition Toolbox. Achiziţia unui set de date este iniţiată din mediul Matlab. După setarea portului paralel al calculatorului în regim bidirecţional, se face setarea parametrilor achiziţiei. Mai exact se specifică dimensiunile vectorului de date şi numărul de eşantioane pentru fiecare vector. Vectorii de date sunt furnizaţi de către sistemul de achiziţie în format paralel. La atingerea numărului de eşantioane specificat achiziţia se opreşte şi pe monitor apare întrebarea dacă se doreşte achiziţia unui nou vector. În caz afirmativ procesul se reia. În caz negativ procesul se termină şi datele achiziţionate sunt înregistrate într-un fişier împreună cu informaţiile cu privire la numărul de vectori, dimensiunea vectorului de date şi numărul de eşantioane pentru fiecare vector. Datele achiziţionate servesc ca vectori de instruire sau de test pentru orice tip de reţea neuronală care se doreşte a fi implementată şi care se simulează folosind mediul Matlab.

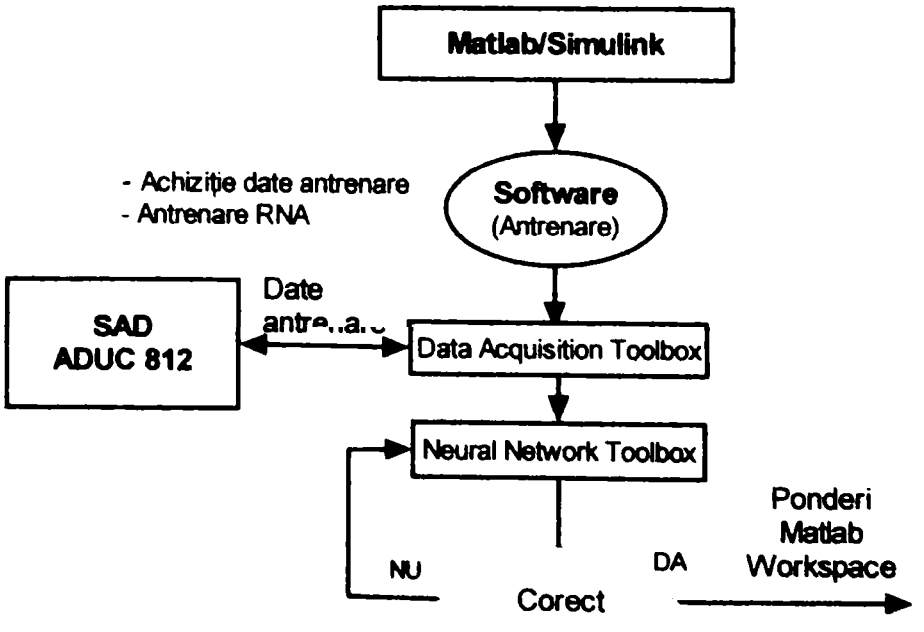


Figura 4.10 Etapele implementării software a RNA

4.4.1.2 Modelarea şi antrenarea RNA cu NN Toolbox

Avantajele implementării software a fazei de antrenare a RNA au fost prezentate în capitolele anterioare. Dintre acestea cel mai mare avantaj în faza de elaborare a unui prototip este oferit flexibilitatea implementării. Metoda propusă foloseşte Neural Network Toolbox din mediul Matlab pentru implementarea fazei de antrenare. Proiectarea, antrenarea şi simularea reţelelor neuronale se fac în mod obişnuit printr-un cod Matlab. Interfaţa grafică Network/Data Manager oferă o soluţie prietenoasă pentru importul, crearea, antrenarea, simularea, utilizarea, şi exportul reţelelor neuronale şi a datelor.

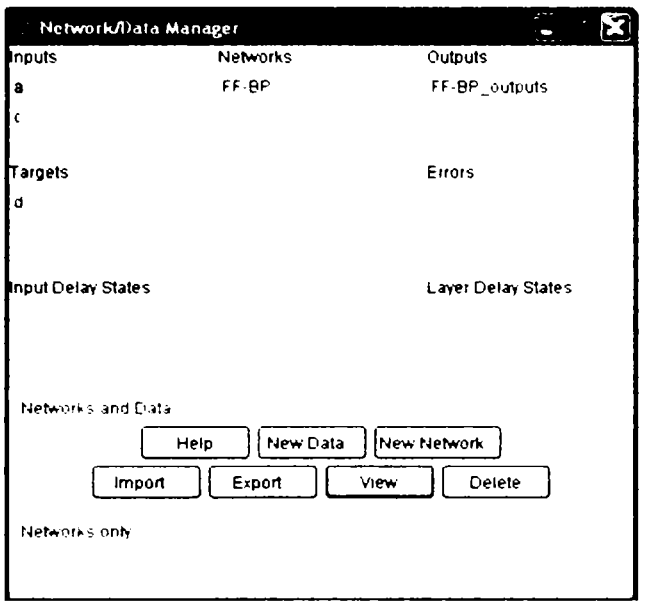


Figura 4.11 Interfaţa grafică Network/Data Manager

Etapele proiectării RNA folosind interfaţa grafică sunt prezentate în figura 4.12.

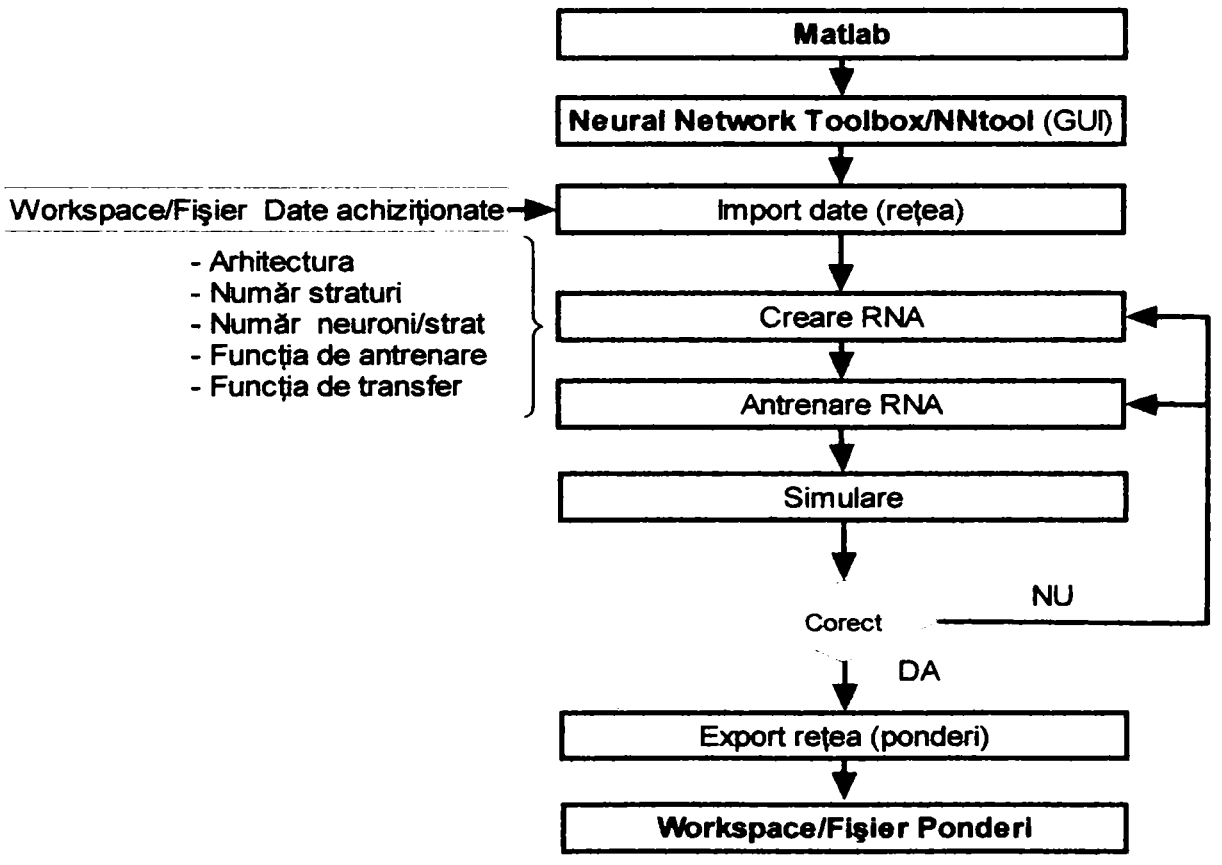


Figura 4.12 Etapele proiectării reţelei neuronale

Datele achiziţionate pot fi încărcate direct din mediul de lucru Matlab, dacă achiziţia de date şi crearea reţelei se face în aceeaşi sesiune de lucru, sau din fişierul în care a fost salvat de programul de achiziţie, în caz contrar.

A doua etapă este crearea propriu-zisă a reţelei care constă în alegerea tipului reţelei, a funcţiilor de antrenare, adaptare, performanţă, a numărului de straturi, numărului de neuroni pe strat, şi a funcţiei de activare pentru fiecare strat, aşa cum se prezintă în figura 3.13. Odată creată reţeaua poate fi vizualizată (figura 3.14).

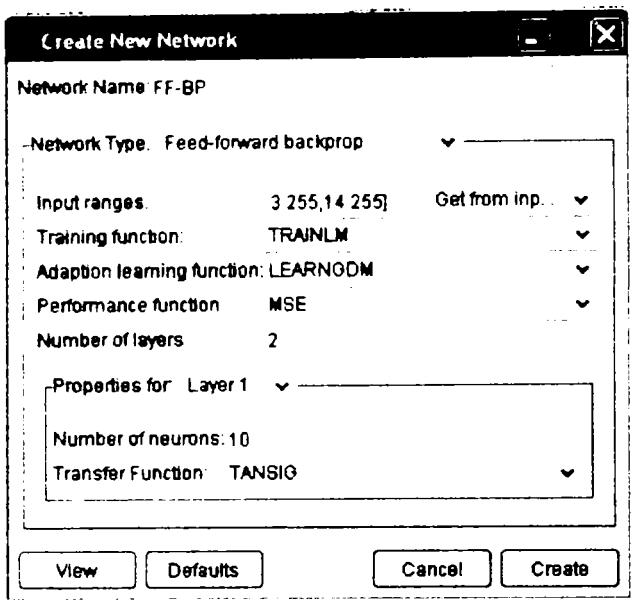


Figura 4.13 Crearea unei RNA noi

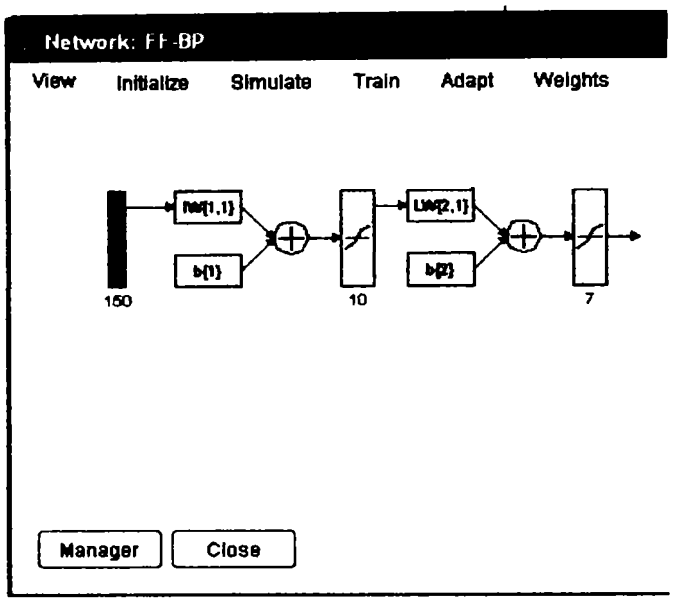
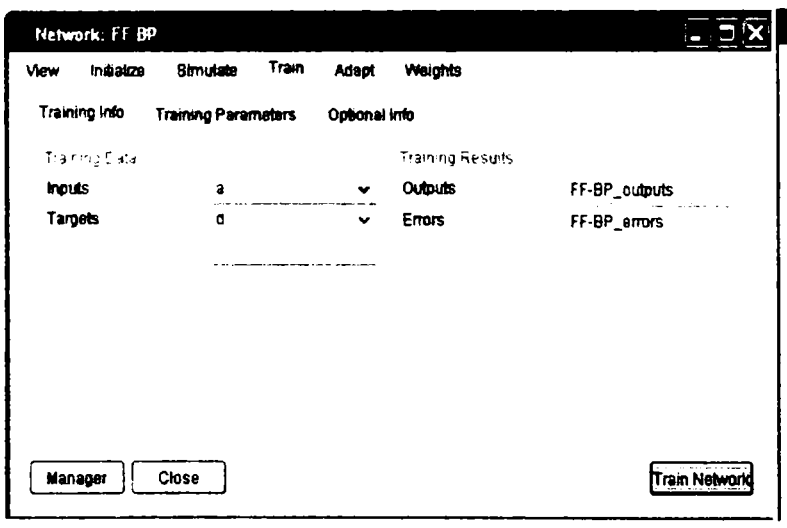
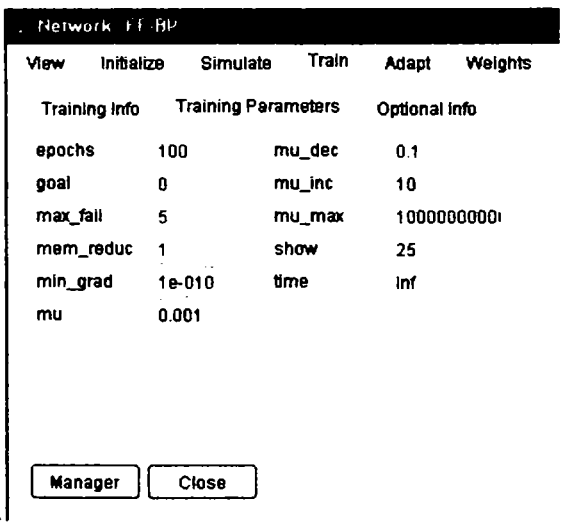


Figura 4.14 Vizualizarea reţelei create

În continuare se poate trece la iniţializarea parametrilor antrenării şi apoi la antrenarea propriu-zisă a reţelei. Se selectează datele de intrare, datele de ieşire ţintă precum şi parametrii antrenării.





a) Setarea datelor de antrenare **b) Parametrii antrenării**

Figura 4.15 Antrenarea RNA

În cazul în care în timpul antrenării anumiţi parametrii impuşi nu pot fi atinşi, se ajustează parametrii de antrenare sau se poate crea o nouă reţea cu altă arhitectură sau alţi parametrii.

Dacă antrenarea s-a efectuat cu succes, reţeaua neuronală poate fi simulată folosind în acest scop un set de date intrare-ieşire diferit faţă de cel utilizat în faza de antrenare.

Răspunsul reţelei la datele de intrare sunt prezentate în figura 4.17.

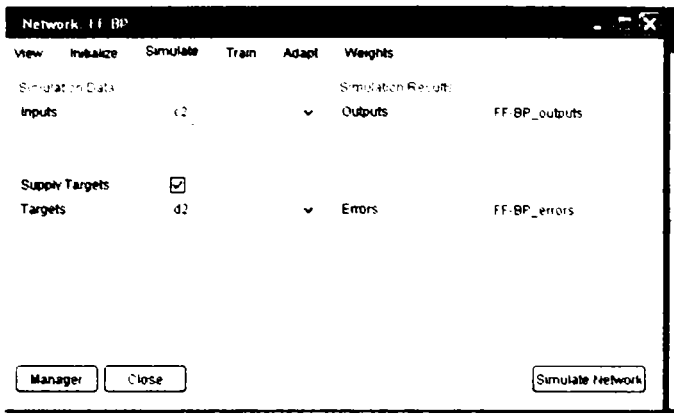


Figura 4.16 Simularea RNA

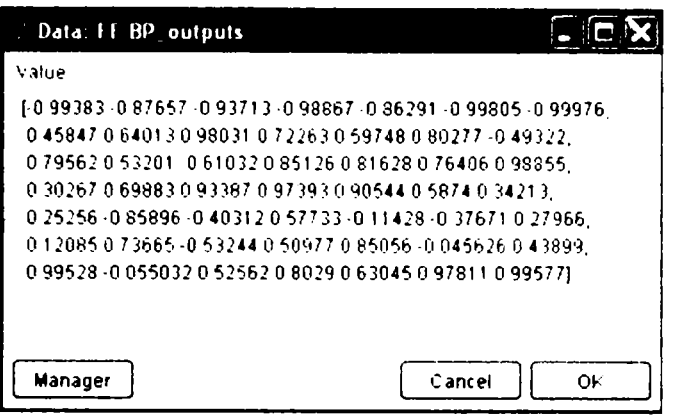


Figura 4.17 Rezultatul simulării

Figura 4.18 prezintă posibilitatea vizualizării ponderilor şi a factorilor de deplasare a scării (bias) pentru fiecare strat în parte.

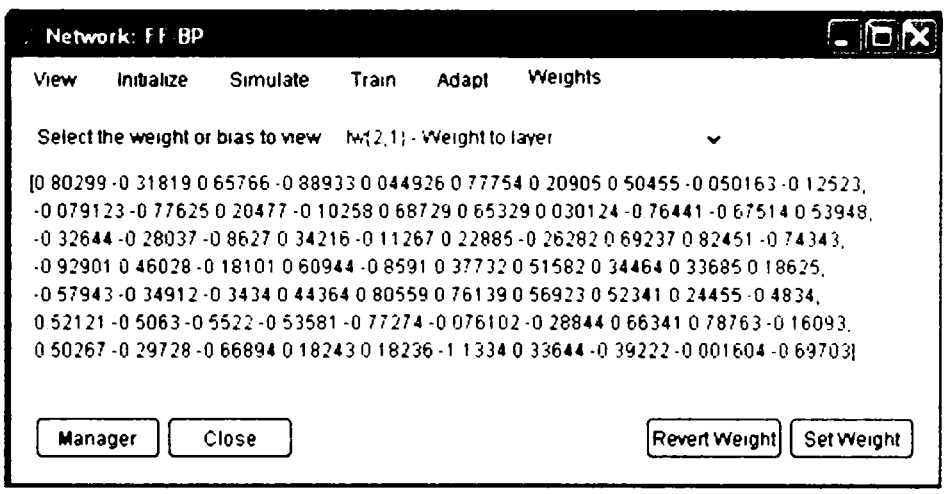


Figura 4.18 Vizualizarea ponderilor şi a bias-ului

Se pot face mai multe simulări cu diverse arhitecturi de reţele neuronale, diverse tipuri şi diverşi algoritmi de învăţare pentru a determina soluţia care satisface cerinţele impuse cu cele mai puţine resurse (număr minim de straturi respectiv neuroni/strat). În acest mod pot fi create diverse tipuri de reţele neuronale artificiale pentru a identifica cea care corespunde cel mai bine aplicaţiei date.

Reţeaua neuronală creată şi antrenată poate fi exportată în mediul de lucru Matlab sau salvată într-un fişier, în vederea implementării hardware a fazei de propagare.

4.4.2 Implementarea hardware a fazei de propagare

Dacă în faza de dezvoltare a reţelei neuronale este importantă flexibilitatea în faza de propagare una din cele mai importante caracteristici este viteza de răspuns a reţelei. De aceea pentru implementare se folosesc tot mai des circuitele FPGA caracterizate de o structură masiv paralelă ceea ce duce la implementări cu performanţe foarte bune.

Descrierea unei reţele neuronale într-un mediu familiar cum este Matlab/Simulink şi utilizarea System Generator pentru transformarea reprezentării la nivel de sistem într-un fişier de configurare a circuitelor hardware, reprezintă o soluţie flexibilă, eficientă din punct de vedere al utilizării resurselor şi rapidă, de implementare a RNA. Paragraful următor descrie metoda concepută pentru modelarea unei reţele neuronale folosind System Generator.

4.4.2.1 Modelarea şi simularea RNA cu System Generator

System Generator este un software creat de firma Xilinx care permite modelarea şi simularea sistemelor de procesare a semnalelor bazate pe circuite FPGA, prin intermediul interfeţei grafice pentru modelarea sistemelor dinamice, oferite de Simulink. System Generator constă într-o bibliotecă pentru Simulink numită Xilinx Blockset, şi un software care transformă un model Simulink într-un model implementabil hardware. System Generator mapează parametrii sistemului definiţi în Simulink la entităţi, arhitecturi, porturi, semnale şi atribute ale implementării hardware. În plus, generează automat fişiere de comandă pentru programele de simulare HDL, uneltele de sinteză şi implementare FPGA, astfel că utilizatorul poate folosi mediul grafic de la specificaţiile sistem până la realizarea hardware. Etapele implementării hardware a reţelelor neuronale artificiale folosind System Generator sunt prezentate în figura 4.19.

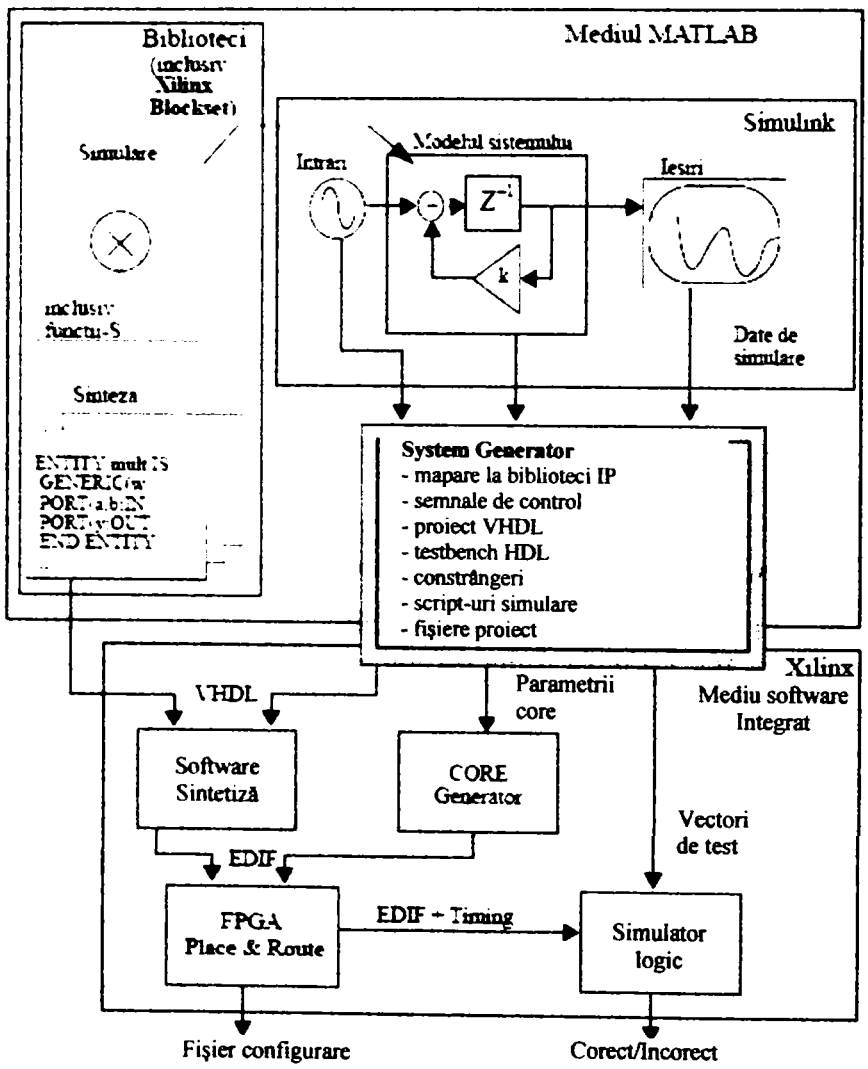


Figura 4.19 Etapele implementării hardware a RNA folosind System Generator

Crearea unui model implementabil de reţea neuronală începe cu crearea unui model nou în Simulink (fişier .mdl). Se adaugă blocurile Xilinx dorite şi se interconectează într-un mod similar ca într-un un editor schematic. Blocurile Xilinx pot fi combinate cu alte elemente Simulink, dar numai blocurile sau subsistemele realizate din blocuri din biblioteca Xilinx Blockset pot fi implementate. La fel ca majoritatea blocurilor din Simulink, blocurile Xilinx au parametrii care pot fi setaţi de utilizator.

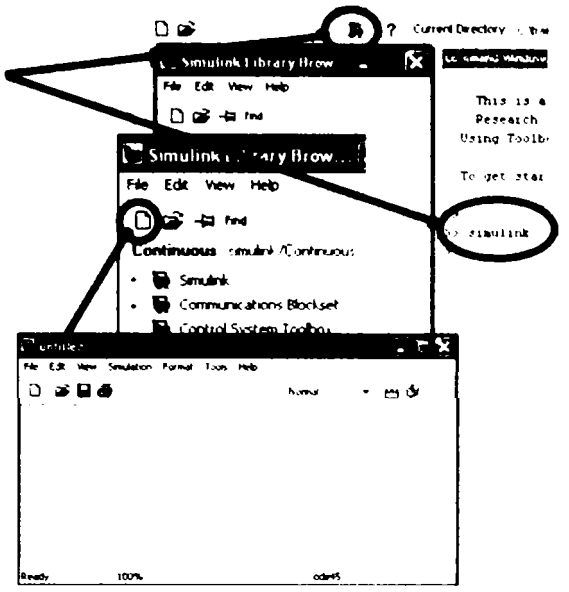


Figura 4.20 Crearea unui modelul nou de RNA

Biblioteca Xilinx blockset conţine funcţii logice, aritmetice, DSP şi pentru comunicaţii. Aceste blocuri pot fi văzute folosind utilitarul Simulink Library Browser.

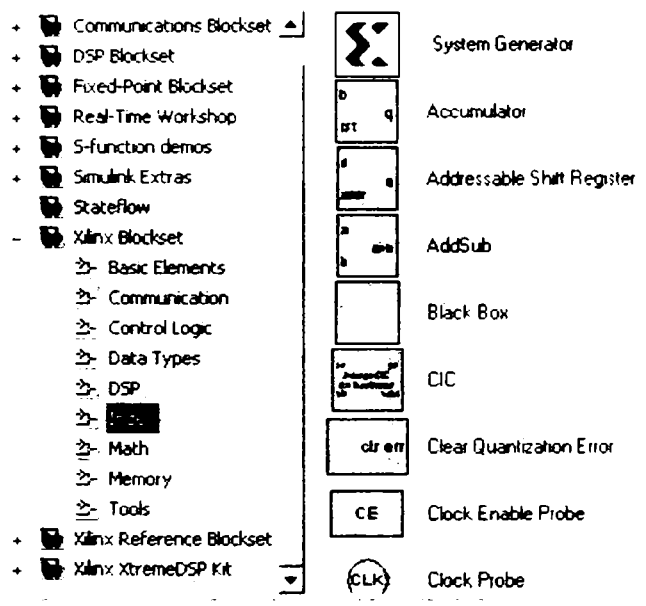


Figura 4.21 Biblioteca Xilinx Blockset Simulink Library Browser

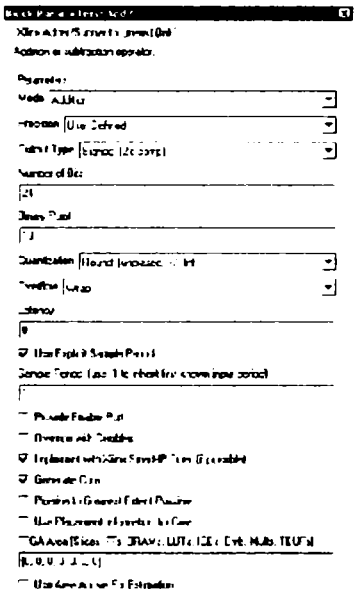


Figura 4.22 Fereastră de dialog pentru setarea în parametrilor

Un model în System Generator poate fi împărţit în mai multe secţiuni. Secţiunea prezentată cu albastru în figura 4.23 reprezintă partea realizabilă hardware şi ea va fi implementată în FPGA. Blocurile galbene reprezintă interfeţe de intrare şi ieşire între blocurile Xilinx şi alte blocuri Simulink. Blocurile de intrare-ieşire reprezintă porturile de intrare-ieşire ale entităţii VHDL top-level (pinii dispozitivului). Celelalte blocuri Simulink delimitate cu linie punctată verde reprezintă surse care vor alimenta circuitul respectiv dispozitive de ieşire ce permit testarea proiectului. Orice bloc Simulink poate fi interfaţat cu blocurile Xilinx cu ajutorul blocurilor de I/E, pentru a face conversia între reprezentarea numerelor în dublă precizie şi reprezentarea în virgula fixă.

Blocul System Generator (de culoare maro) este obligatoriu în blocul ierarhic superior, pentru a putea simula orice model care foloseşte blocurile Xilinx.

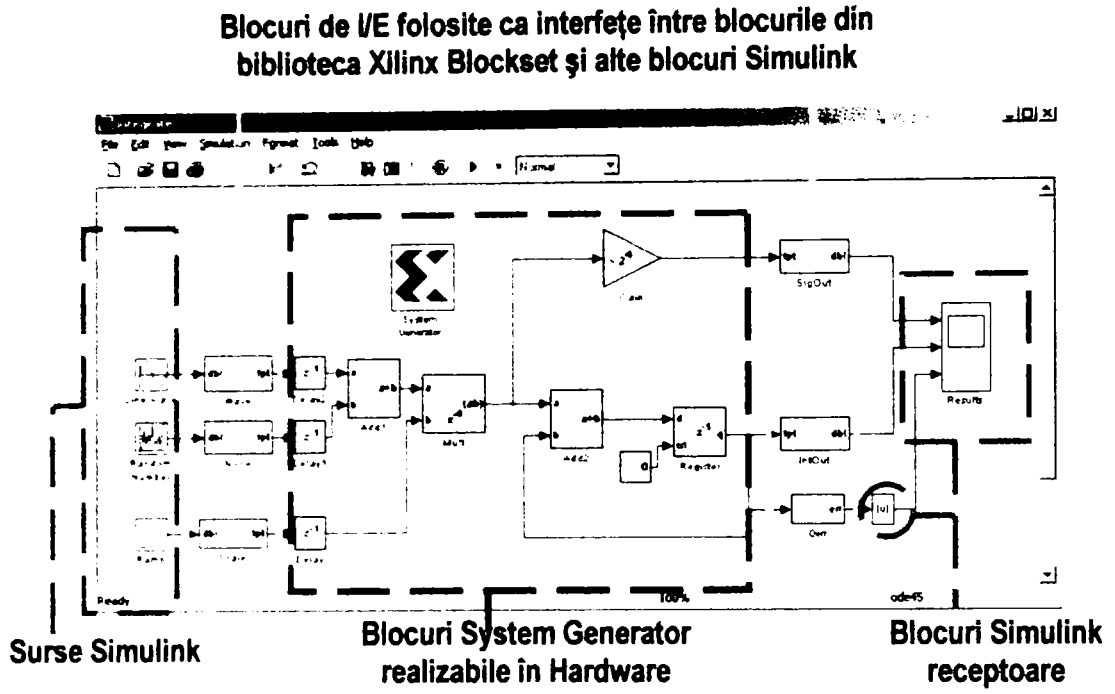


Figura 4.23 Model System Generator care prezintă diferitele secțiuni ale design-ului

Următoarea etapă este cea de simulare a modelului creat. Rezultatele pot văzute prin înserarea de blocuri de tip Scope în model.

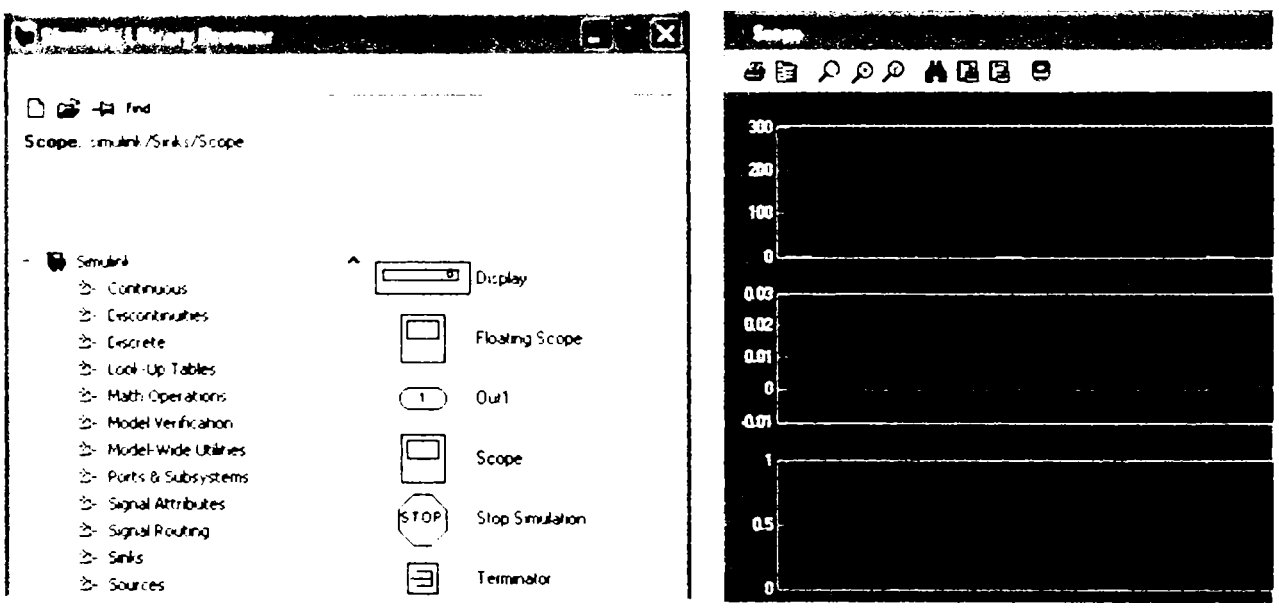


Figura 4.24 Simularea modelului RNA

Ultima etapă din procesul System Generator este reprezentată de generarea codului VHDL. După crearea modelului şi simularea acestuia se poate trece la generarea proiectului sintetizabil în format VHDL. Acest lucru este posibil prin intermediul blocului System Generator. Dacă proiectul este de tip ierarhic atunci în blocul ierarhic superior se inserează un bloc System Generator, ca în figura 4.25.

Pentru generarea codului VHDL şi a modulelor Xilinx IP Core se face dublu clic pe blocul System Generator şi se fac setările cu privire la:

- tipul compilării
  - HDL Netlist
  - NGC Netlist
  - Bitstream

- EDK Export Tool
- Hardware Co-Simulation

- tipul dispozitivul în care se doreşte implementarea
- directorul în care se vor scrie fişierele
- programul de sinteză folosit
- perioada semnalului de ceas pentru FPGA
- locaţia pinului de ceas a FPGA
- generarea unui model pentru simulare (testbench)
- perioada sistemului Simulink

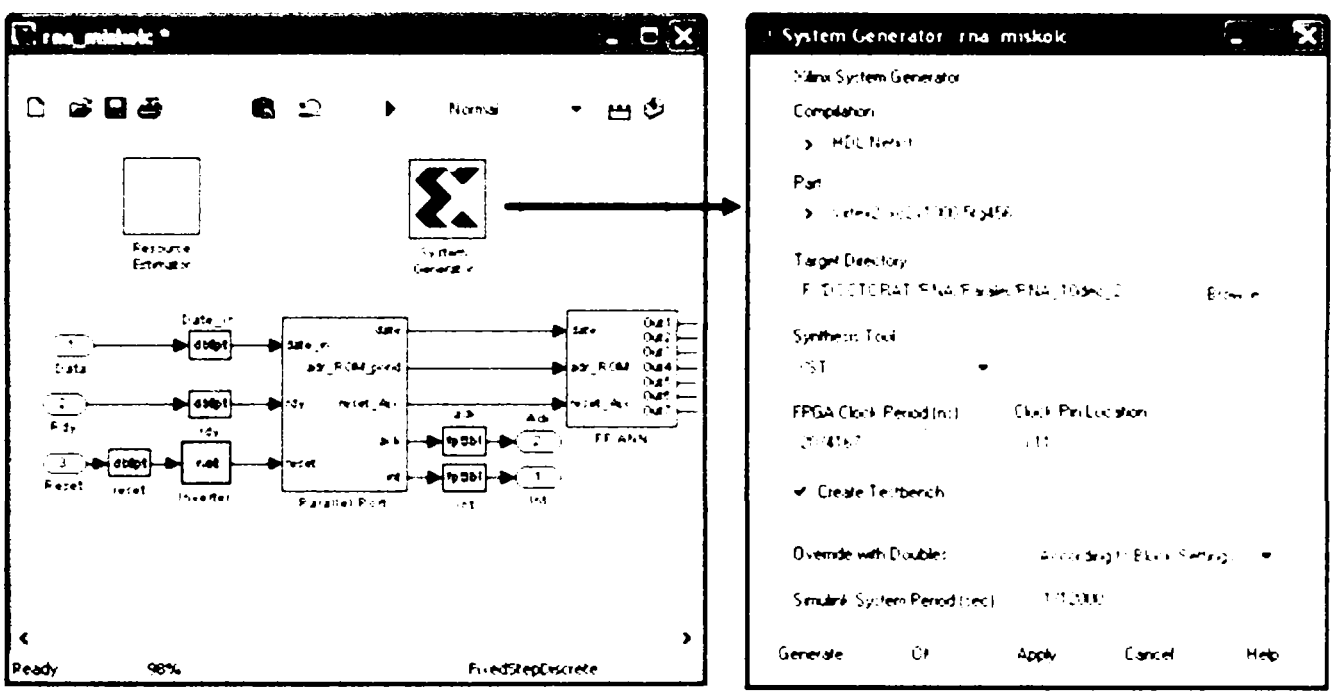


Figura 4.25 Setarea parametrilor System Generator şi generarea codului VHDL

Butonul Apply salvează setările făcute şi lasă fereastra deschisă, în timp ce butonul OK salvează setările şi închide fereastra. Generarea codului VHDL se face prin apăsarea butonului Generate.

System Generator produce diverse fişiere, majoritatea de tip VHDL:

- Fişiere de descriere a modelului

.VHD : Fişiere VHDL echivalente modelului Simulink.

.EDN : Fişiere pentru implementarea modulelor Xilinx Core.

.XCF : Fişier cu constrângerile de timp. Acest fişier este foarte important pentru atingerea performanţelor specificate. Fişierul .XCF este generat pentru programul de sinteză Xilinx Synthesis Tool iar fişierul .NCF pentru Synplify şi Leonardo Spectrum.

- Fişiere proiect

.NPL : Fişiere proiect pentru Project Navigator. Crucial pentru sintetiza proiectului printr-o simplă apăsare de buton.

.TCL : Script tcl pentru crearea proiectului Synplify şi Leonardo.

- Fişiere simulare

.DO : Fişiere de tip DO pentru programul de simulare ModelSim. Fiecare fişier DO este apelat automat de proiectul ISE şi este folosit pentru simulare funcţională, post-translatare, post-asociere şi post-plasare şi interconectare.

.DAT : Acestea sunt fişierele de date conţinând vectorii de test din System Generator.

.VHD : nume_proiect_testbench este modelul VHDL folosit pentru testare. El foloseşte fişierele .DAT pentru a verifica rezultatele simulării.

4.4.2.2 Sinteza şi implementarea folosind Xilinx ISE

După generarea codului urmează simularea comportamentală, sinteza şi implementarea folosind un program care suportă dispozitivele Xilinx. În lucrarea de faţă s-a folosit programul de simulare ModelSim Xilinx Edition şi programul de sinteză XST (Xilinx Synthesis Technology) din cadrul Xilinx ISE 6.2i.

System Generator a creat un fişier proiect ISE cu extensia .npl. Deschiderea acestui fişier încărca proiectului în Xilinx ISE Project Navigator care permite sinteza şi implementarea acestuia. Ca în orice proces Xilinx opţiunile de sinteză şi implementare pot fi modificate.

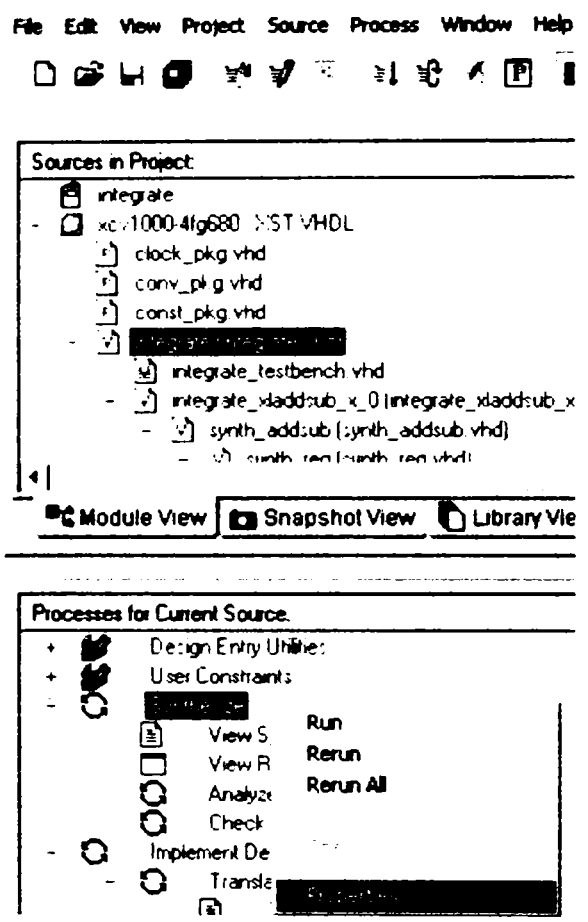


Figura 4.26 Mediul XILINX ISE 6.2i pentru sinteza şi implementarea proiectului

Diagrama procesul de sinteză şi implementare este prezentată în figura 4.27.

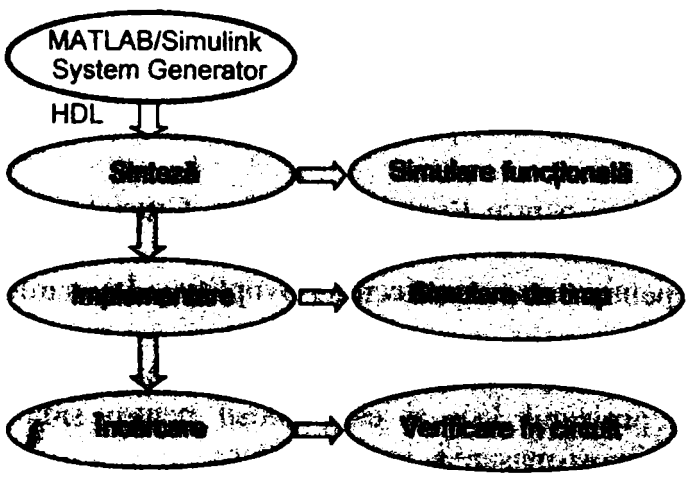


Figura 4.27 Procesul de sinteză şi implementare cu Xilinx ISE

Primul pas în această fază este sinteza codului *VHDL*, care constă în convertirea codului într-un set de primitive sau componente care pot fi asamblate în tehnologia către care este orientat proiectul. Proiectantul poate să "ajute" uneltele de sinteză prin specificarea unor constrângeri specifice tehnologiei.

Următoarea etapă este simularea funcţională cu simulatorul ModelSim. El va face uz de fişierul pn_behavioral.do creat de System Generator. Pentru rularea simulării în modulul surse din Project Navigator se selectează fişierul nume_proiect_testbench.vhd şi apoi se dă comanda Simulate Behavioral VHDL Model. Aceasta comandă deschide consola ModelSim şi rulează fişierul pn_behavioral.do.

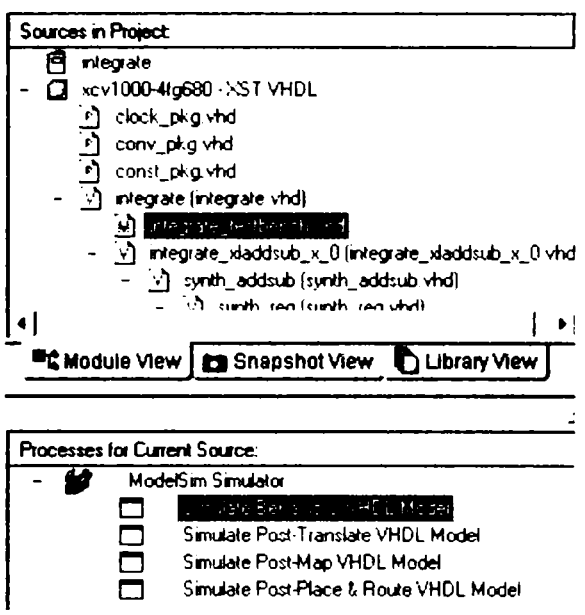


Figura 4.28 Etapa de simulare comportamentală a proiectului

Etapa de implementare constă în translatarea, maparea, plasarea şi interconectarea circuitelor, precum şi generarea fişierului de configurare .bit. În această etapă se mai pot exercita câteva constrângeri asupra proiectului cum ar fi plasarea anumitor module în chip sau atribuirea de pini intrărilor şi ieşirilor externe, etc. Procesul poate fi declanşat în urma selecţiei fişierului top-level VHDL şi lansarea comenzii Generate Programming File. Project Navigator lansează în execuţie diverse procese care sunt necesare pentru obţinerea fişierului de configurare.

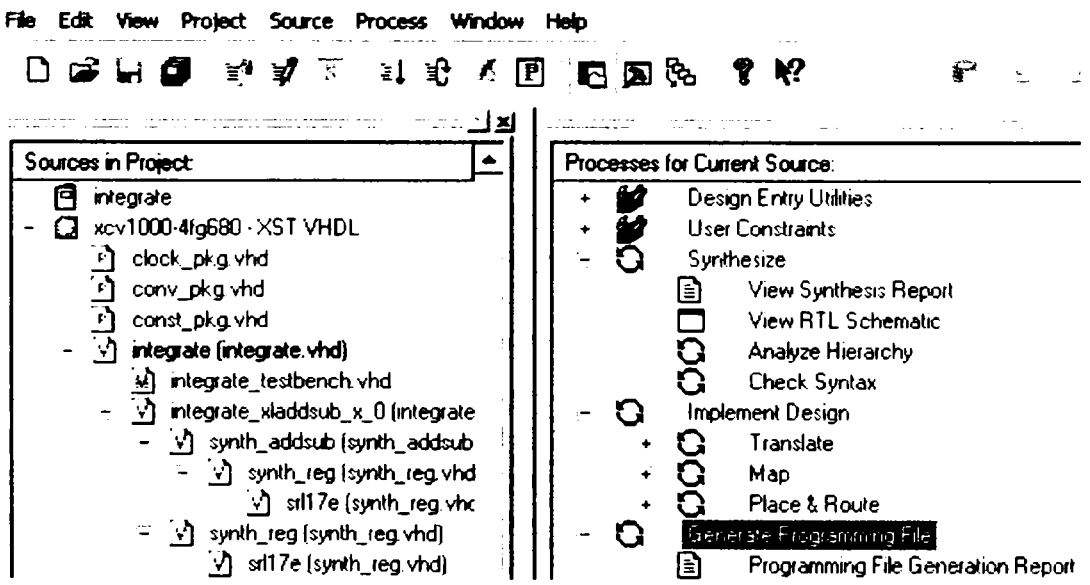


Figura 4.29 Etapa de implementare şi generare a fişierului de configurare

Fişierele obţinute în timpul creări fişierului de configurare pot fi vizualizate. Astfel de exemplu poate fi vizualizat modul în care sunt plasate componentele circuitului în FPGA folosind utilitarul Floorplanner. Un alt fişier creat este fişierul cu întârzierile reale din circuit, întârzieri datorate lungimii traseelor, încărcărilor electrice şi a altor factori.

Faza finală este cea de verificare a întârzierilor (*timing verification*). Un alt fişier .do creat de System Generator este fişierul pn_postpar.do care serveşte pentru rularea simulării post implementare. Dacă se lansează procesul de simulare Post-place & Route VHDL Model se deschide consola ModelSim şi fişierul pn_postpar.do va fi folosit pentru compilare şi rularea modelului de testare. Testbench-ul foloseşte stimulii de intrare folosiţi în Simulink şi compară rezultatele cu ieşirile corespunzătoare generate de Simulink. Dacă proiectul este lipsit de erori acest raport apare în consolă.

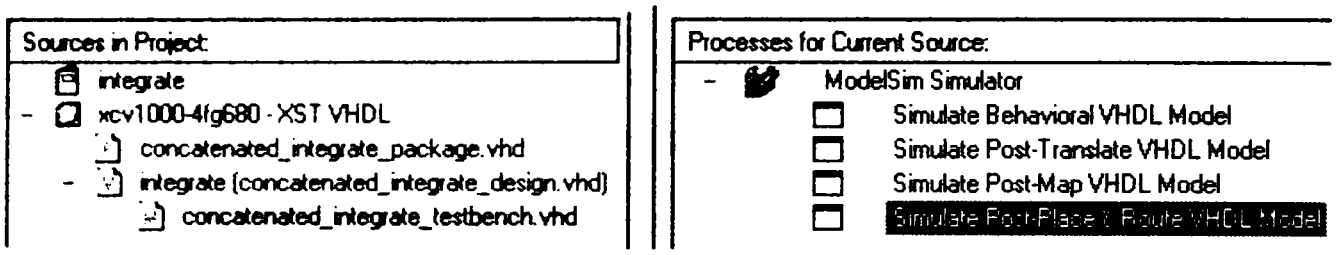


Figura 4.30 Simularea post plasare şi rutare

La fel ca în decursul oricărui proces de proiectare este posibil să avansezi doi paşi şi apoi să revii un pas sau chiar mai rău. Fiecare etapă oferă posibilitatea de a reveni asupra soluţiilor. Problema cea mai nedorită este acea de a fi obligat să revii din *faza finală*, de exemplu când un proiect sintetizat ocupă mai mult spaţiu decât cel oferit de circuitul FPGA sau CPLD folosit. În aceste cazuri trebuie revenit asupra proiectului şi trebuie regândit.

Odată ce fişierul de configurare este creat aceasta se încarcă în circuitul FPGA folosind unul din programele cunoscute cum este de exemplu utilitarul Impact.

4.4.2.3 Co-simulare HDL şi Co-simulare cu hardware în buclă

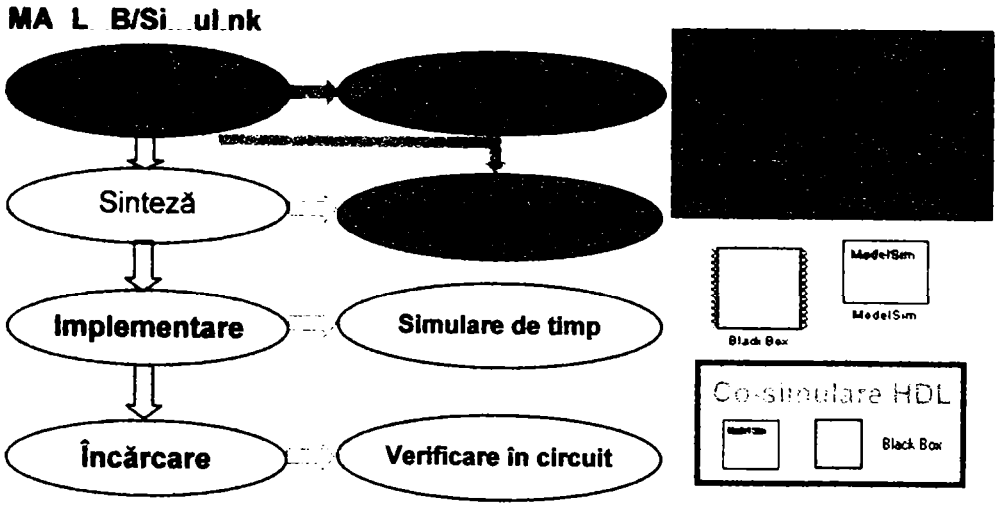


Figura 4.31 Etapele Co-simulării HDL

Diagrama din figura 4.31 prezintă a doua modalitate, din cele trei existente, de proiectare a unei reţele neuronale artificiale folosind System Generator. Această metodă poartă numele de Co-simulare HDL. Ea permite utilizarea în proiect a unor blocuri de tip Black Box care pot include coduri VHDL dezvoltate de utilizator sau module de tip proprietate intelectuală, împreună cu blocuri Xilinx System Generator şi generarea unui proiect sintetizabil care poate fi implementat folosind Xilinx ISE Project Navigator. De

asemenea foloseşte blocul ModelSim care invocă simulatorul ModelSim pentru simularea proiectului. Rezultatele simulării sunt aduse înapoi în Simulink şi rezultatele pot fi vizualizate folosind blocurile tipice Simulink.

Cea de a treia modalitate de proiectare a unei reţele neuronale artificiale folosind System Generator este prezentată în diagrama din figura 4.32. Aceasta este un proces tipic pentru verificare rapidă cu hardware în buclă cunoscută sub numele de „Hardware in the loop Co-Simulation". Un proiect poate conţine Black-Box-uri, pentru includerea de coduri VHDL dezvoltate de utilizator sau module IP, blocuri de interfaţă hardware, şi blocuri Xilinx System Generator. În acest proces se generează un proiect sintetizabil şi implementabil folosind programul Xilinx, generându-se fişierul de configurare .bit şi un bloc „hardware in the loop". Adăugând acest bloc proiectului, utilizatorul poate porni simularea care încarcă fişierul de configurare în hardware şi preia în timp real răspunsul circuitului hardware, care este reintrodus în Simulink şi rezultatele pot fi vizualizate folosind blocurile tipice Simulink.

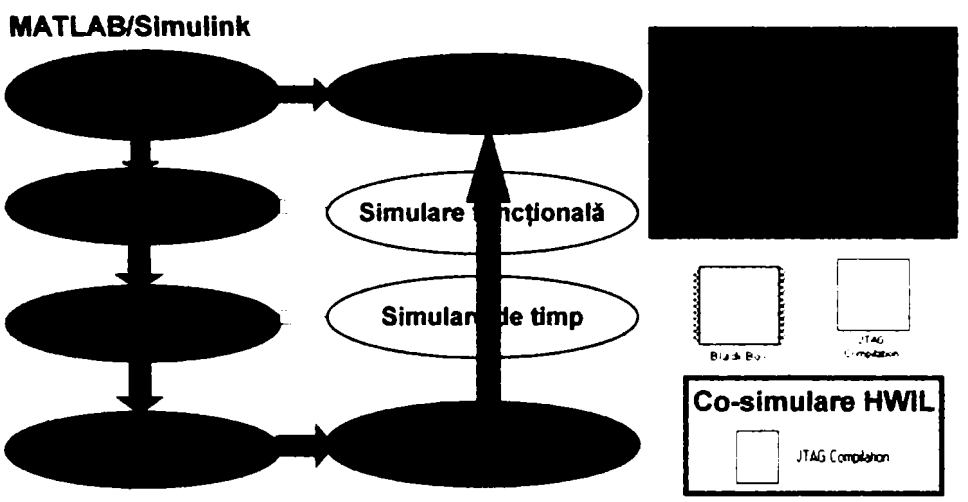


Figura 4.32 Etapele Co-simulării cu hardware în buclă

4.5 Crearea bibliotecilor cu blocuri specifice RNA

Metoda prezentată permite implementarea unei reţele neuronale artificiale prin simpla utilizare a bibliotecilor furnizate cu programul System Generator (Xilinx Toolbox), sau crearea unor biblioteci specifice RNA. Biblioteca Xilinx oferă proiectantului o mare varietate de blocuri pentru modelarea oricărui circuit, dar modelarea unei reţele neuronale este destul de dificilă de aceea crearea unei biblioteci cu blocuri specifice RNA este foarte utilă.

În cadrul prezentei lucrări s-a dezvoltat o bibliotecă cu blocuri RNA care pot fi configurate de proiectantul reţelei şi sunt direct implementabile. Astfel au fost dezvoltate de la elementele mai simple cum ar fi blocuri de multiplicare – acumulare, funcţii de activare, până la diverse reţele cu un singur strat cu un număr variabil de neuroni. Toate aceste elemente sunt parametrizabile prin interfaţa grafică sau încarcă parametrii din spaţiul de lucru al programului Matlab sau dintr-un fişier specificat.

4.5.1 Modelul neuronului artificial

Reţelele neuronale artificiale sunt inspirate şi modelate după reţelele neuronale biologice a căror complexitate este de exemplu în cazul creierului uman de 10^{11} neuroni fiecare cu o medie de 10^3-10^4 conexiuni. Transmisia semnalelor între neuronii biologici prin intermediul sinapselor este un proces chimic complex în care sunt emise substanţe purtătoare de informaţie (neurotransmiţători). Efectul acestora este creşterea sau scăderea potenţialului în interiorul celulei receptoare. Dacă potenţialul atinge un prag, neuronul este excitat şi produce impulsuri nervoase care sunt transmise prin intermediul axonilor spre alţi neuroni. Aceasta

este caracteristica pe care încearcă să o reproducă şi modelul propus de McCulloch Pitts. Modelul prezentat în figura 4.33, cu mici variaţii este modelul cel mai des folosit în reţelele neuronale artificiale.

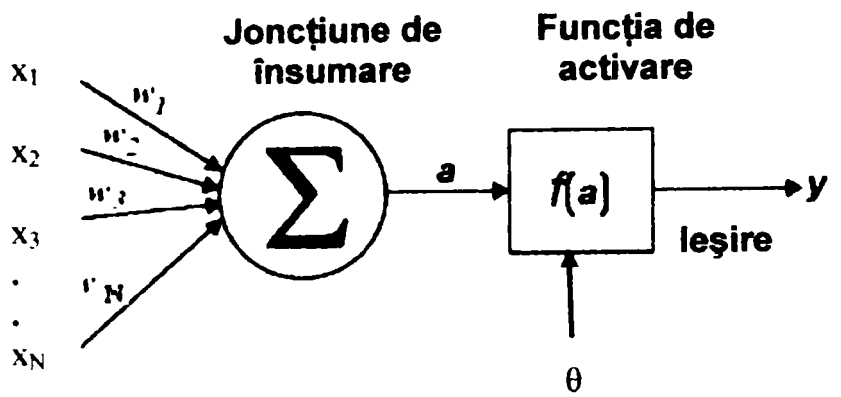


Figura 4.33 Modelul neuronului artificial

Neuronul artificial prezentat în figura de mai sus are N intrări fiecare având asignată o pondere w_i (i=1..N). Componenta principală a neuronului este sumatorul ponderat care calculează intrarea netă a neuronului conform relaţiei:

$$a = \sum_{i=1}^{N} w_i x_i \tag{4.1}$$

Semnalul de ieşire a neuronului este o funcţie a acestei valori:

$$y(x) = f(a-\theta) = f\left(\sum_{i=1}^{N} w_i x_i - \theta\right) \tag{4.2}$$

unde θ reprezintă valoarea pragului de activare al neuronului.

Funcţia f poartă numele de funcţie de activare. Iniţial funcţia propusă de modelul McCulloch Pitts era funcţia prag dar sunt larg utilizate şi alte funcţii cum ar fi funcţia liniară, saturaţie, sigmoid, etc.

O analiză a implementărilor hardware a reţelelor neuronale propuse în literatură conduce la diagrama bloc din figura 4.34 [15].

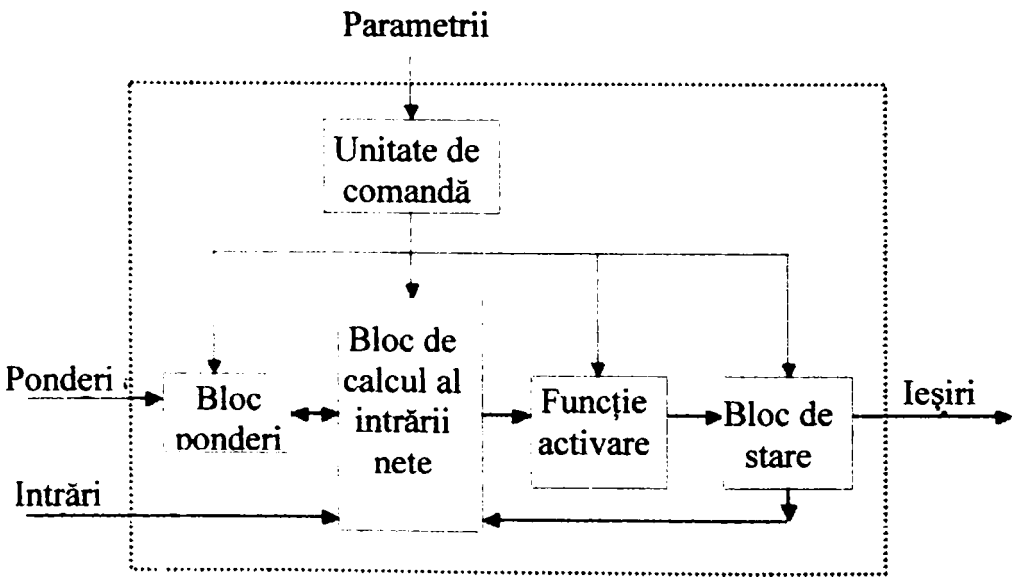


Figura 4.34 Schema bloc a neuronului artificial

Blocul de calcul al intrării nete realizează produsul $x_i w_i$ şi însumarea acestor produse. Apoi ieşirile sunt obţinute în blocul de memorarea stării neuronului în urma aplicării funcţiei de activare asupra sumei de produse.

Un element de procesare neuronal trebuie să fie compus din următoarele blocuri principale: buffer de intrare (memorie date), memorie ponderi, multiplicator – acumulator, funcţia de activare, buffer de ieşire şi blocul de comandă.

Ecuaţia 4.1 se poate implementa prin operaţia de multiplicare–acumulare (MAC), operaţie fundamentală în procesarea digitală a semnalelor şi multe alte aplicaţii, folosind:

- un singur bloc MAC (paralelism de neuron)
- N blocuri MAC (pentru o implementare complet paralelă, sau paralelism de sinapsă)
- 1 până la N blocuri MAC (în cazul unei implementări parţial paralele)

Datele de intrare şi ponderile trebuie să fie stocate în memorii şi apoi introduse sub controlul unităţii de comandă în blocul multiplicator-acumulator. Modelul unui bloc MAC împreună cu memoriile de date şi ponderi aferente este prezentat în figura 4.35.

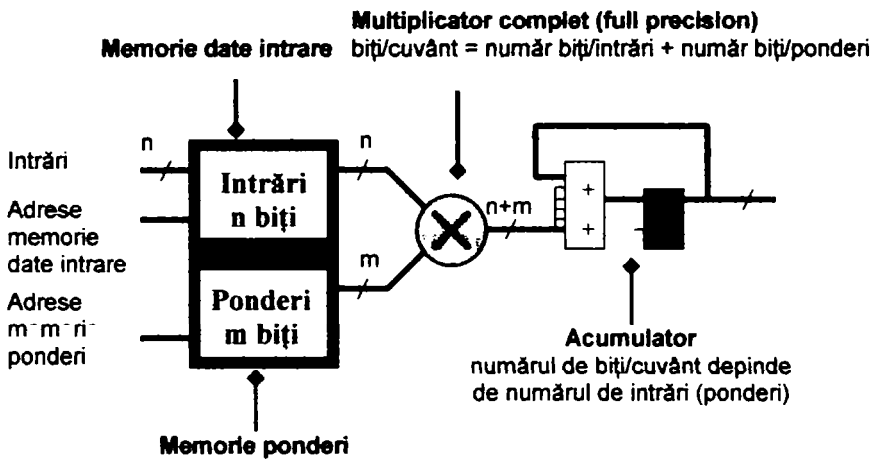


Figura 4.35 Modelul unui bloc de MAC

Blocul multiplicator - acumulator acceptă date de intrare şi ponderi cu sau fără semn pe n respectiv m biţi. Toţi aceşti parametrii pot fi modificaţi direct din interfaţa utilizator a blocul MAC. Ponderile şi datele trebuiesc stocate în memorii.

4.5.1.1 Implementarea blocului multiplicator-acumulator

Deoarece nu există un bloc MAC în biblioteca Xilinx Blockset, aceasta trebuie creată folosind blocurile Xilinx existente, o soluţie posibilă este prezentată în figura 4.36.

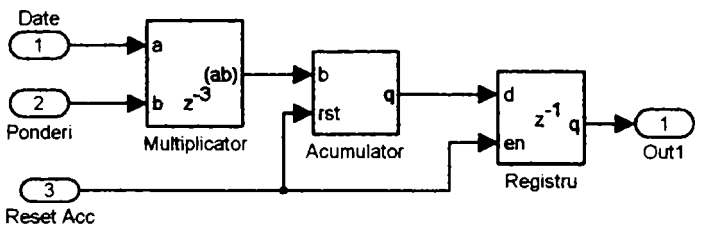


Figura 4.36 Blocul MAC realizat cu blocuri Xilinx

Multiplicatorul poate fi implementat foarte eficient folosind blocurile multiplicatoare dedicate care există în circuitele FPGA de tipul Virtex-II, Virtex-II Pro sau Spartan-3. De exemplu în circuitul Virtex-II XC2V250 (echivalent cu 250.000 porţi logice) există 24 de

multiplicatoare dedicate pe 18 biţi, în XC2V1000 (echivalent cu un milion porţi logice), 40, iar în Virtex-II Pro XC2VP125, 556 de astfel de multiplicatoare. Precizia necesară este în funcţie de numărul de biţi ai datelor de intrare respectiv a ponderilor. Aceasta poate fi setată la precizie maximă sau valoarea aleasă de utilizator, aşa cum se poate observa în figura 4.37 a respectiv b.

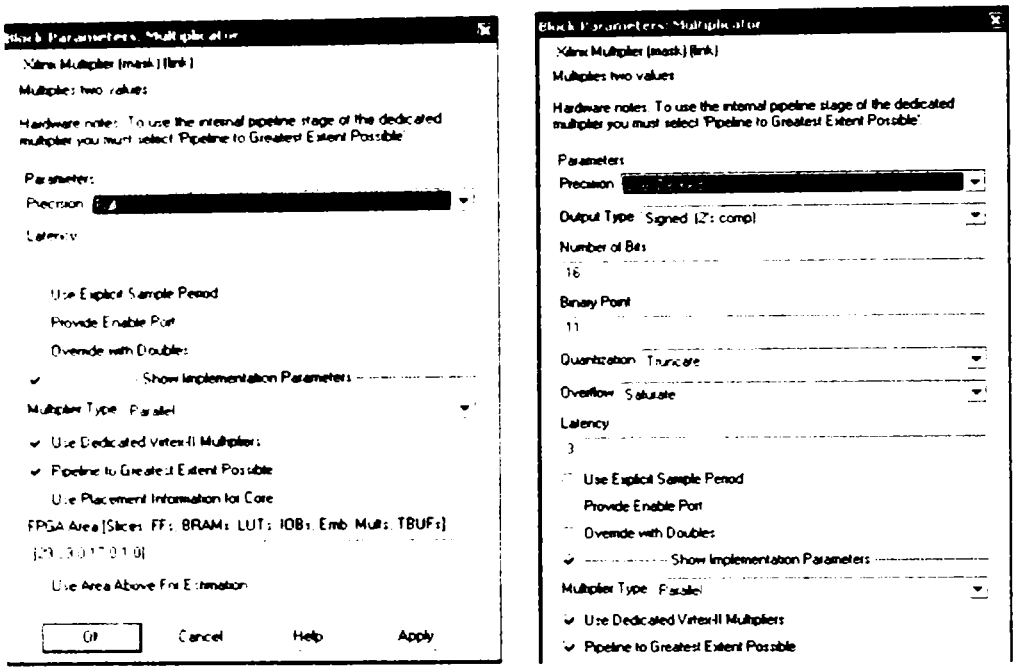


Figura 4.37 Setarea parametrilor blocului multiplicator

Multiplicatorul poate fi implementat eficient chiar şi în alte tipuri de circuite prin utilizarea unor blocuri specifice, create folosind Xilinx LogiCORE Generator.

Acumulatorul se implementează folosind blocul acumulator din biblioteca Xilinx. Precizia acumulatorului depinde de numărul de operaţii de acumulare deci de numărul de intrări ale neuronului şi trebuie să fie suficient de mare pentru a nu apărea depăşire. Astfel numărul de biţi ai acumulatorului este dat de relaţia 4.3:

$$nr.biti\_ac = ceil(\log_2(p)) + (n+m) \tag{4.3}$$

unde p reprezintă numărul de operaţii de acumulare, iar n+m reprezintă numărul de biţi ai datelor de intrare în acumulator.

După reset acumulatorul nu se reiniţializează cu zero ci cu valoarea curentă a intrării, ceea ce permite blocului MAC o funcţionare continuă. Un registru asigură memorarea temporară a datelor pe parcursul operaţiilor de acumulare.

Resursele folosite pentru implementarea uni MAC depind de un număr mare de factori, printre care numărul de biţi ai intrărilor, ai ponderilor, precizia multiplicatorului, numărul de biţi ai acumulatorului, şi nu în ultimul rând de modul de implementare a multiplicatorului (cu multiplicatoare dedicate sau folosind resurse logice CLB).

De exemplu, pentru o reprezentare a datelor de intrare pe 8 biţi fără semn (UFix_8_0), a ponderilor pe 12 biţi cu semn (Fix_12_12), un multiplicator de precizie maximă (Fix_20_12), precizia acumulatorului calculată după formula 4.3 (Fix_23_12) şi un neuron cu 8 intrări resursele utilizate sunt prezentate în tabelul 4.1. Tabelul prezintă „cazul cel mai defavorabil" pentru implementările, cu şi fără multiplicatoare dedicate, în trei familii de dispozitive FPGA (primele două fiind folosite în experimentele efectuate).

Aceste resurse depind în foarte mare măsură de parametrii de mai sus şi de aceea în funcţie de aplicaţie ei trebuie modificaţi pentru un raport optim între resurse şi performanţă.

Tabelul 4.1 Resursele consumate pentru implementarea unui bloc MAC

Resurse consumate	MAC implementat cu un multiplicator dedicat VIRTEX-II				MAC implementat cu un multiplicator Xilinx LogiCORE			
	MAC	Mult.	Acu.	Reg.	**MAC**	Mult.	Acu.	Reg.
Slice-uri	**58**	20	26	12	**93**	55	26	12
Bistabile	**87**	40	24	23	**156**	109	24	23
Memorii Block RAM	**0**	0	0	0	**0**	0	0	0
Tabele de memorii (LUT)	**50**	0	50	0	**154**	104	50	0
Multiplicatoare dedicate	**1**	1	0	0	**0**	0	0	0
% dintr-un Spartan-II, 50.000 porţi	--				12,10 %			
% dintr-un Virtex-II, 1.000.000 porţi	1,13 %				1,81 %			
% dintr-un Virtex-II Pro, XC2VP125	0,1 %				0,16 %			

De exemplu în cazul în care precizia acumulatorului poate fi redusă de la 23 la 20 de biţi, iar precizia cerută a ieşirilor este de 3 biţi blocul MAC poate fi modificat ca în figura 4.38.

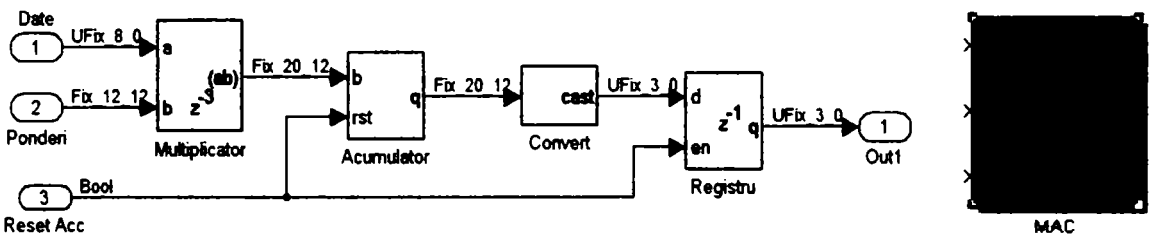


Figura 4.38 Bloc MAC modificat şi simbolul acestuia

Intre acumulator şi ieşire se poate introduce un bloc convertor care reduce numărul de biţi pe care este reprezentată cuvântul de ieşire, astfel scade şi numărul de bistabile necesare pentru implementarea registrului de ieşire. Resursele consumate se modifică astfel: 54 slice-uri, 64 bistabile, 61 LUT-uri, 1 multiplicator dedicat. Parametrii pot fi modificaţi direct din interfaţa grafică a blocului MAC prezentată în figura 4.39 pentru două setări diferite.

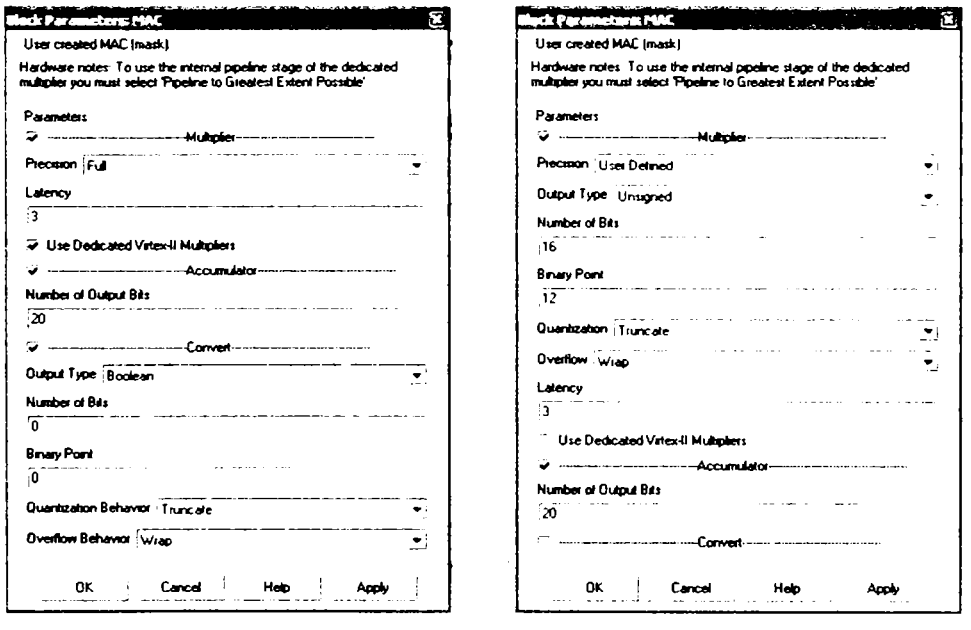


Figura 4.39 Interfaţa grafică pentru setarea parametrilor MAC

Prin diverse optimizări ce sunt prezentate în Capitolul V, resursele pot fi scăzute până la 10 slice-uri pentru un MAC implementat cu un multiplicator dedicat VIRTEX-II.

4.5.1.2 Memoriile de date şi de ponderi

Datele şi ponderile trebuiesc să fie stocate într-o memorie. Se poate opta între memoria distribuită (SelectRAM) şi memoria bloc (block SelectRAM). Utilizarea memoriei distribuite este recomandată numai în cazul în care nu sunt disponibile memorii bloc sau cantitatea de date sau ponderi de memorat este mică.

Pentru implementarea memoriei de date se poate utiliza o memorie RAM dual-port care permite înscrierea datelor cu o anumită viteză şi citirea acestora la portul B, la o viteză diferită. Numărul cuvintelor de date memorate este determinat de numărul perechilor date-ponderi care intră în procesul de multiplicare-acumulare pentru a produce un rezultat. Numărul de biţi pe cuvânt depinde de precizia de reprezentare a datelor de intrare.

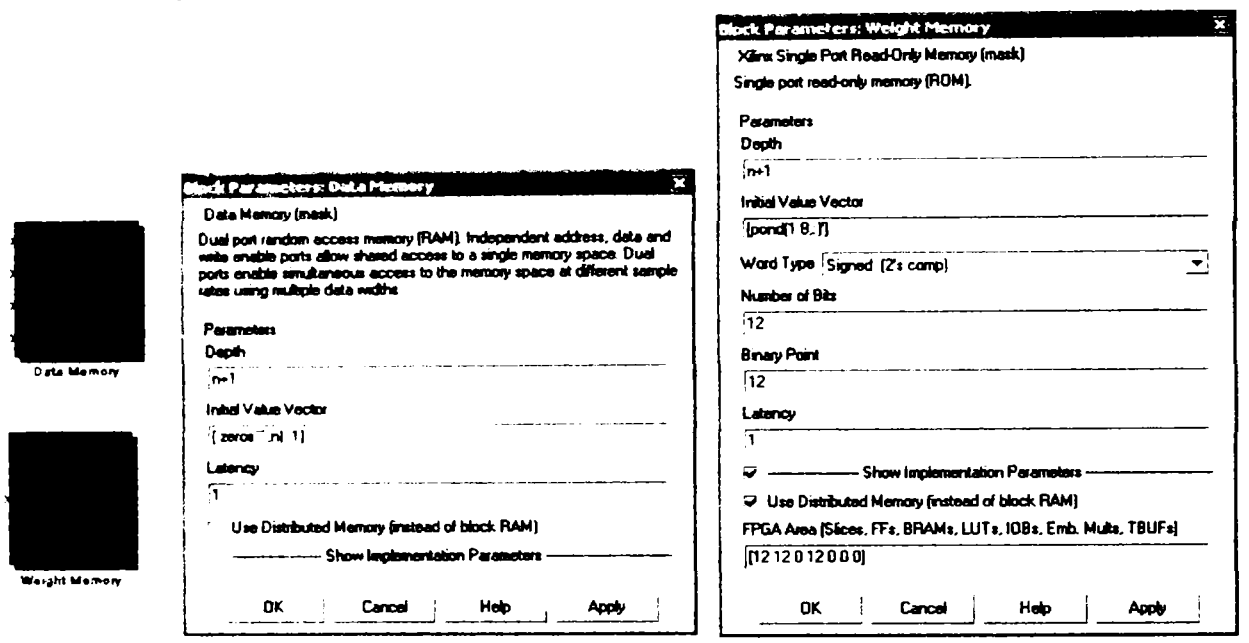


Figura 4.40 Memoriile de date şi ponderi şi meniurile de setare a parametrilor

Pentru memorarea ponderilor se poate folosi o memorie ROM distribuită dacă numărul intrărilor neuronului nu este mare, pentru a păstra memoriile de tip bloc pentru memorarea datelor.

Precizia de reprezentare a ponderilor este una din cele mai importante alegeri la implementarea RNA în FPGA. O precizie mai mare înseamnă mai puţine erori de cuantizare în implementarea finală, în timp ce o precizie mai redusă conduce la un circuit mai simplu, viteză mai mare şi reducerea resurselor necesare implementării şi a puterii consumate. O modalitate de a rezolva acest compromis este de a determina precizia minimă necesară pentru rezolvarea problemei date. În mod tradiţional precizia minimă este determinată prin încercări succesive, simulând soluţiile înainte de implementare. Mai mulţi autori au studiat problema preciziei minime concluziile fiind diferite în funcţie de tipul reţelei şi a aplicaţiei. Holt şi Baker [102] au determinat că o reprezentare pe 16 biţi a ponderilor este în general suficientă pentru o reţea de tip perceptron cu mai multe straturi şi algoritm de învăţare cu propagare înapoi a erorii.

Recent s-au făcut progrese mai importante în determinarea preciziei minime din punct de vedere teoretic. Draghici [181] a demonstrat în că în cazul cel mai defavorabil domeniul necesar pentru ponderi [-p, p] este estimat prin distanţa minimă între tipare din clase diferite:

$$d = (\sqrt{n})/(2p) \tag{4.4}$$

unde p este un număr întreg şi n este dimensiunea intrării.

4.5.1.3 Funcţia de activare

Implementarea directă a unor funcţii neliniare cum este şi funcţia sigmoid este foarte costisitoare. Există două abordări practice posibile pentru aproximarea unor astfel de funcţii pentru a putea fi implementate în FPGA [116].

Aproximarea prin liniarizare pe porţiuni este o metodă de aproximare a unei funcţii neliniare prin funcţii de gradul I de forma:

$$y = ax + b \tag{4.5}$$

Dacă coeficienţii sunt aleşi să fie puteri ale lui 2, atunci funcţia sigmoid poate fi obţinută printr-o serie de operaţii de deplasare şi adunare. Multe implementări ale funcţiei sigmoid folosesc o astfel de aproximare [37]

Implementarea folosind tabele de memorii este cea de a doua metodă uzuală folosită pentru implementarea funcţiei de activare. Pentru aceasta se pot folosi memoriile Single-Port ROM. Conţinutul memoriei este determinat în Matlab şi înscris în memorie prin intermediul unui fişier.

Cu toate că ambele implementări reprezintă doar aproximări (de ordinul 1 respectiv de ordinul 0) ale funcţiilor de transfer, este de preferat o îmbunătăţire a performanţelor de viteză, cost, mărime, în detrimentul unei uşoare scăderi a preciziei. Aceste implementări se numesc hardware prietenoase. Figura 4.41 [4] prezintă capacitatea memoriei necesare pentru implementarea prin tabele de memorii în funcţie de precizia echivalentă pentru aproximarea de ordinul 0 şi 1 a funcţiei sigmoid.

$$f(x) = \frac{1}{1 + e^{\lambda x}} \tag{4.6}$$

unde λ este o constantă.

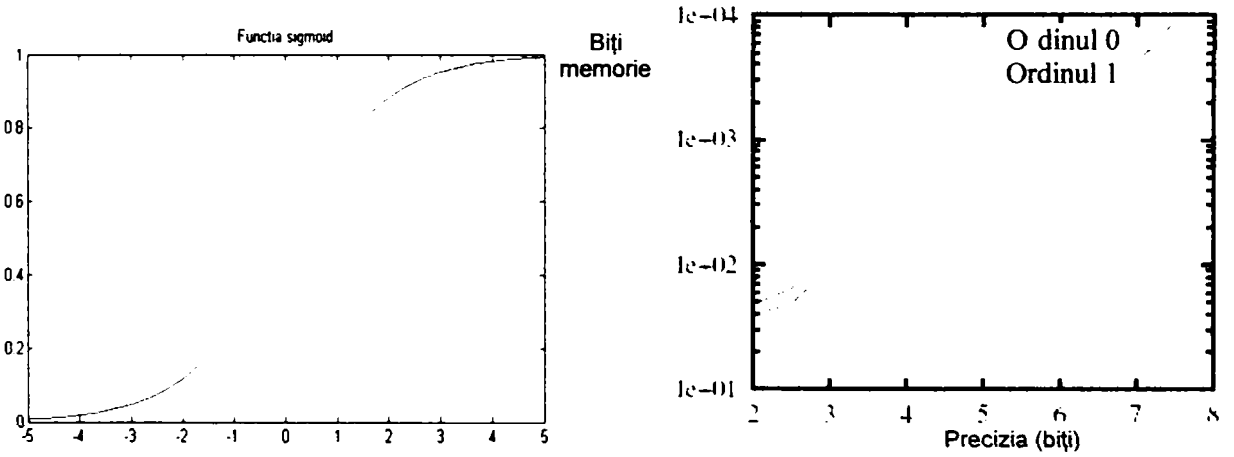


Figura 4.41 Funcţia sigmoid şi capacitatea memoriei necesare implementării

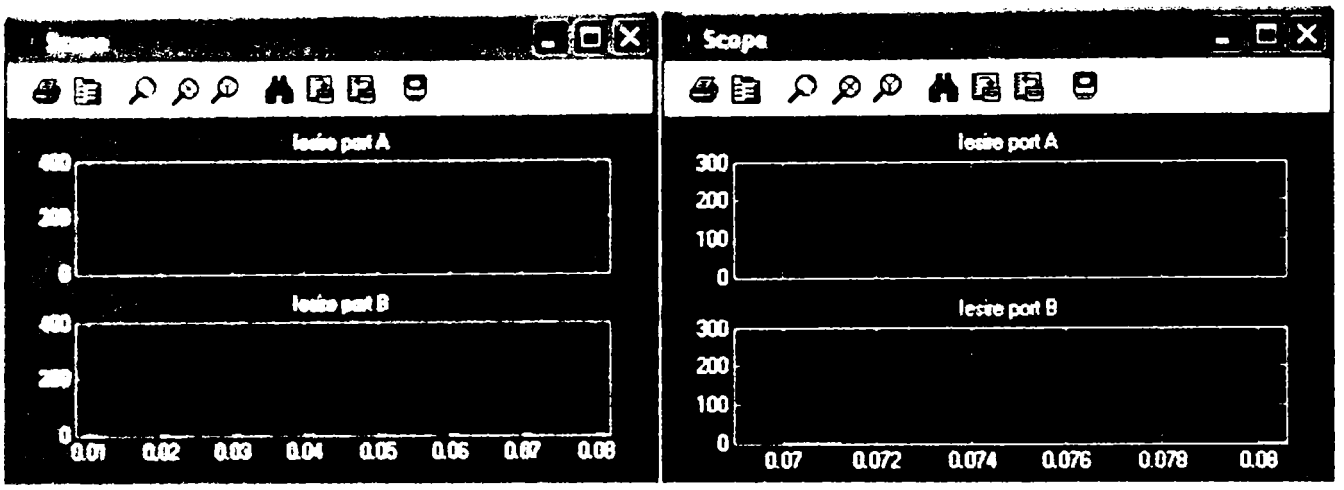
4.5.1.4 Blocul de comandă

Blocul de comandă controlează fluxul de date de intrare şi ieşire din memoria de date, citirea ponderilor şi furnizează semnalele de control pentru MAC.

Înscrierea datelor în memoria de date de tip dual port are loc la viteza cu care acestea sunt furnizate la intrare. După ce şi ultimul eşantion, reprezentând ultima intrare a neuronului, a fost memorat, extragerea datelor din memorie în vederea multiplicării cu ponderile aferente se face la o rată de n+1 ori mai mare, unde n reprezintă numărul de sinapse ale neuronului.

Astfel pe durata ultimului eşantion de intrare, la portul B a memoriei de date sunt disponibile cele n date de intrare la care se adaugă factorul de deplasare a scării (bias).

În figura 4.42.a se pot vedea cele două ieşiri ale memoriei de date în cazul unui neuron cu 7 intrări, iar figura 4.42.b prezintă cele două ieşiri pe durata ultimului eşantion de intrare.



a) b)

Figura 4.42 Ieşirile memoriei de date pe durata înscrierii celor 7 intrări (a) respectiv pe durata ultimului eşantion de intrare (b)

Concomitent cu extragerea datelor de la portul B a memoriei de date blocul de comandă furnizează adresele pentru memoria de ponderi pentru extragerea ponderilor cu care trebuiesc multiplicate intrările. Multiplicatorul introduce un timp de latentă de 3 perioade de eşantionare iar memoria 1 perioadă. Astfel după 4/(n+1) din durata unui eşantion de intrare procesul de multiplicare-acumulare pentru un set de date de intrare este complet şi blocul de comandă furnizează semnalul de enable pentru registrul de ieşire care memorează starea neuronului şi de reset pentru acumulator.

Implementarea logicii de comandă prezentată mai sus se poate face cu ajutorul circuitului din figura 4.43 folosind numărătoare pentru generarea adreselor pentru memorii.

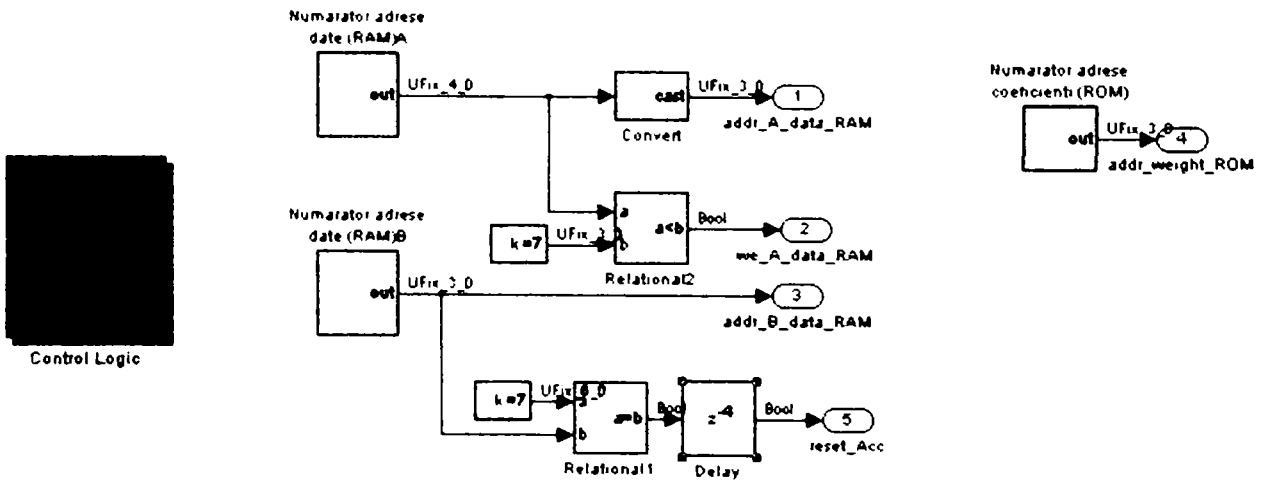


Figura 4.43 Simbolul şi structura blocului de comandă

Formele de undă generate de acest bloc sunt prezentate în figura 4.44.

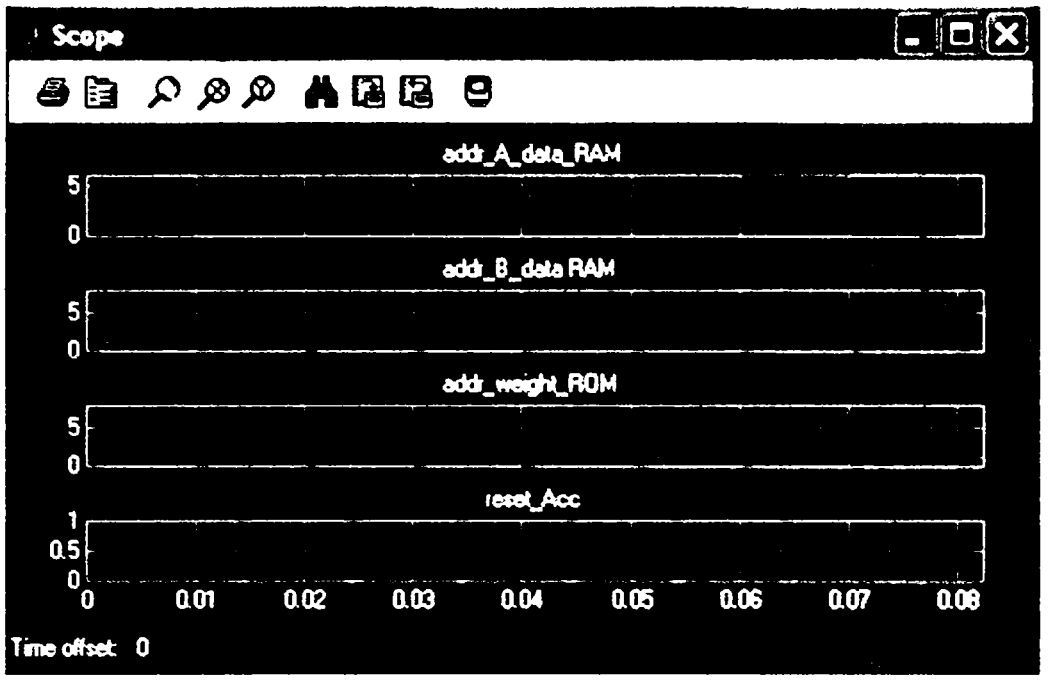


Figura 4.44 Semnalele de control generate de blocul de comandă

4.5.1.5 Neuron implementat folosind biblioteca RNA Blockset

Biblioteca RNA blockset realizată, conţine un prim set de blocuri specifice reţelelor neuronale artificiale, create de autor (figura 4.45).

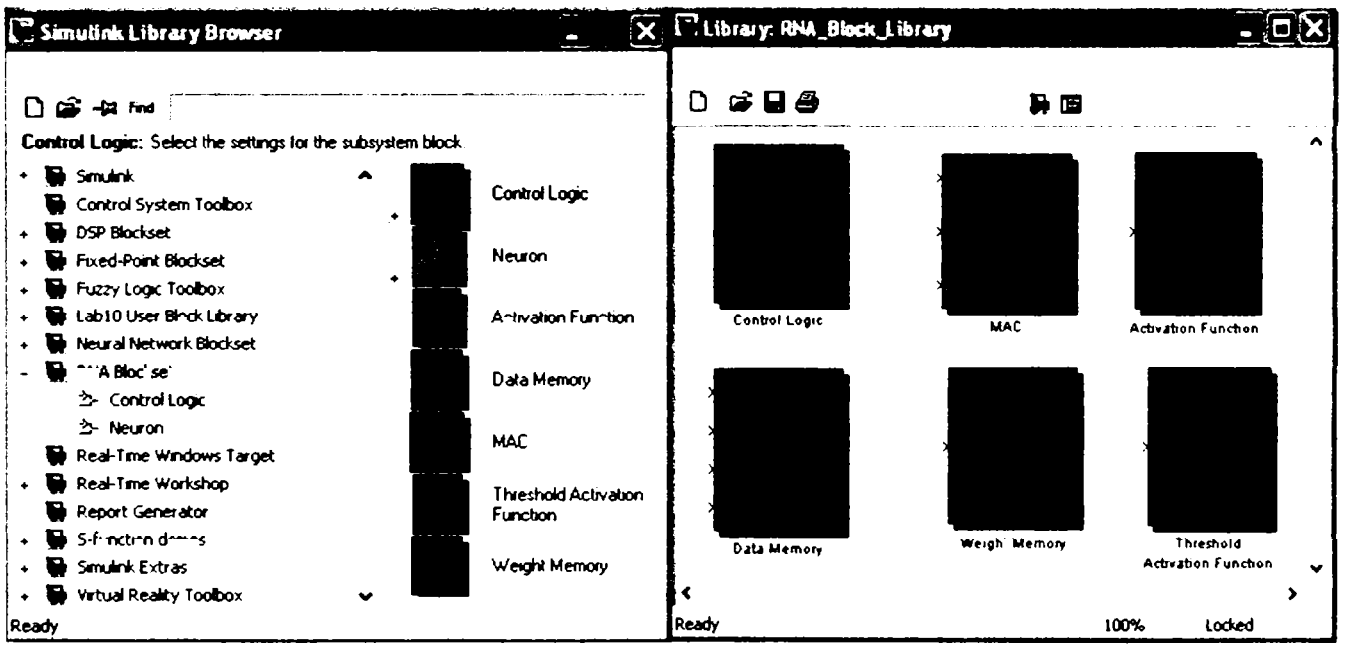


Figura 4.45 Biblioteca RNA Blockset

Cu ajutorul blocurilor de mai sus se poate trece la implementarea unui neuron după modelul din figura 4.34. Se obţine astfel neuronul artificial din figura 3.46.

Parametrii neuronului pot fi setaţi din interfaţa utilizator. Principalii parametrii sunt: numărul de intrări ale neuronului, numărul de biţi folosiţi pentru reprezentarea datelor de intrare, a ponderilor şi ieşirilor, tipul şi precizia multiplicatorului, numărul de biţi alocaţi acumulatorului, numele fişierelor ce se încarcă în memoria de ponderi şi în blocul de activare.

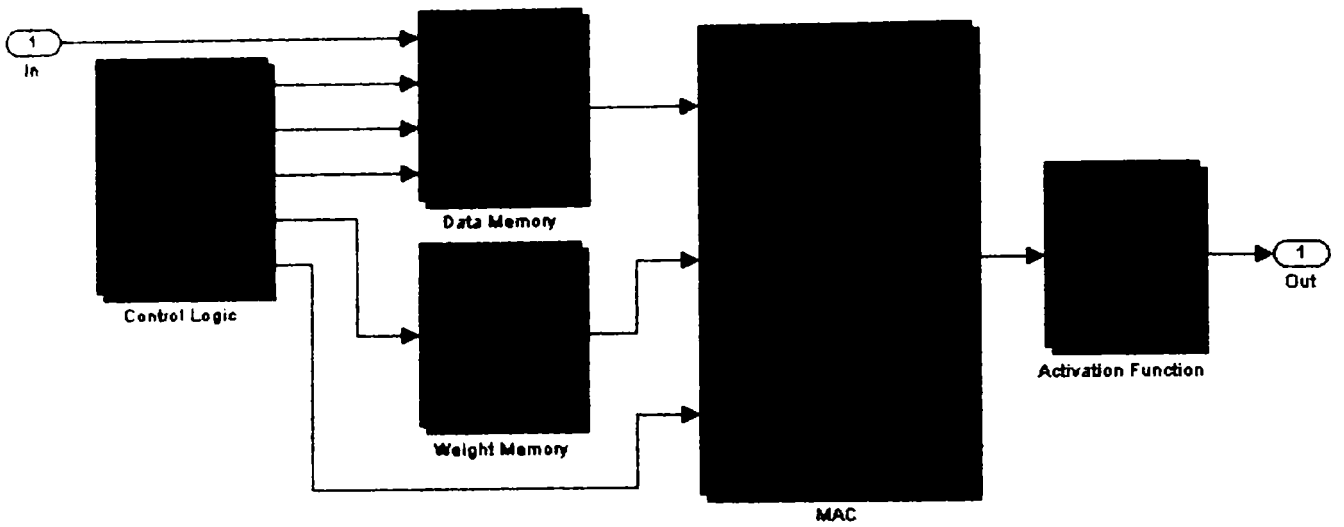


Figura 4.46 Neuron realizat folosind blocurile din biblioteca RNA Blockset

4.6 Concluzii

Din concluziile capitolelor anterioare au reieşit avantajele implementării hibride a RNA. Folosirea unui calculator cu un soft adecvat pentru implementarea etapei de antrenare a oferă avantajul flexibilităţii, în timp ce utilizarea circuitelor FPGA pentru faza de propagare prezintă avantajul vitezei de procesare.

Eforturile depuse în acest capitol s-au concentrat asupra dezvoltării unei platforme integrate hardware-software şi a unei metode pentru implementarea hibridă a reţelelor neuronale. Platforma hardware se compune dintr-un sistem de achiziţie a datelor, o placă de dezvoltare pentru circuite FPGA şi un calculator. Această platformă permite proiectanţilor antrenarea reţelelor neuronale într-un mediu familiar oferit de Matlab/Simulink. Pentru modelarea reţelelor neuronale se foloseşte acelaşi mediu şi o biblioteca dezvoltată cu blocuri specifice reţelelor neuronale: blocul de calcul al intrării nete (MAC), diverse funcţii de activare, memorii pentru ponderi, blocul de comandă, etc. Mediul permite co-simularea celor două modele pentru a verifica buna funcţionare a modelului hardware.

Simularea modelelor hardware create cu blocurile concepute de autor, arată funcţionarea identică cu modelul software şi în consecinţă viabilitatea metodei.

Dintre contribuţiile personale aduse în cadrul acestui capitol se pot aminti:

- Elaborarea unei metode pentru implementarea hibridă a RNA;
- Proiectarea şi realizarea unei platforme hardware care permite atât achiziţia datelor de antrenare şi testare, precum şi implementarea fazei de propagare;
- Conceperea structurii software a mediului de implementare a RNA;
- Crearea unei biblioteci cu blocuri specifice RNA folosind blocuri din biblioteca Xilinx Blockset a programului Simulink/System Generator. Aceste blocuri sunt parametrizabile prin intermediul interfeţei utilizator, sau, pot să preia parametrii din mediul de lucru al Matlab;
- Testarea blocurilor create prin modelarea unui neuron şi co-simularea cu un model software;
- O primă evaluare a resurselor necesare pentru implementarea unui MAC şi determinarea numărului minim de biţi ai acumulatorului.

Capitolul V

Implementări de reţele neuronale feed-forward

Reţelele neuronale de tip feed-forward (FF), sau cu propagare înainte, sunt printre cele mai răspândite tipuri de reţele neuronale artificiale utilizate în aplicaţiile de asociere între un set de modele de intrare şi un set de modele ţintă. Din acest motiv primul tip de reţea implementată folosind metoda prezentată în capitolul anterior este acest tip de reţea.

5.1 Structura

O reţea de tip FF are o structură organizată pe straturi. Fiecare strat primeşte semnale doar de la stratul imediat inferior şi trimite semnale doar spre stratul imediat superior. Nu există conexiuni între neuronii din acelaşi strat. Cele N_i intrări alimentează primul strat care are doar rol de conectare datelor de intrare la reţeaua neuronală, fără a avea rol în procesare datelor. Urmează apoi straturi intermediare numite şi straturi ascunse care pot fi în număr de nici unul, unul sau mai multe. Stratul de ieşire, este atât un strat de procesare cât şi un strat de conectare şi la ieşirea lui se obţine starea neuronului. În figura 5.1 este prezentată arhitectura RNA FF cu *k* straturi intermediare.

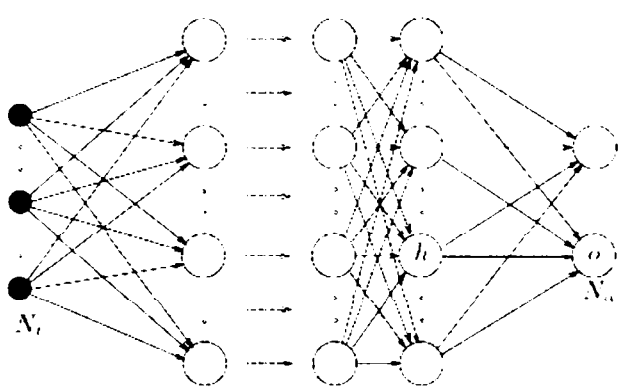


Figura 5.1 RNA de tip FF cu *k* straturi

Valoarea de ieşire a fiecărui neuron este dată de relaţia:

$$y_j = f\left(\sum_{i=k}^{N} w_{ij}\, x_i - \theta\right) \tag{5.1}$$

Nu există o regulă care să stabilească numărul de straturi ai unei reţele neuronale şi nici numărul de neuroni pe strat, aceştia se stabilesc de obicei experimental. Ca o regulă generală este bine ca reţeaua să fie cât mai simplă. În majoritatea cazurilor un strat ascuns este suficient pentru a rezolva orice problemă. Numărul de straturi ascunse nu este în general mai mare decât doi. Numărul de neuroni din straturile de intrare şi de ieşire depind de aplicaţie, în timp ce numărul de neuroni din stratul (straturile) ascunse se determină prin încercări. Un număr prea mare duce nu numai la o creştere importantă a timpului necesar pentru antrenare (care poate ajunge şi la câteva zile) ci şi la o diminuare a capacităţii de generalizare a reţelei. Astfel RNA va răspunde foarte bine la datele pe care le-a învăţat dar va da rezultate slabe la datele de test. De aceea cel mai bine este să se pornească cu un număr mic de neuroni şi acest număr să fie crescut dacă este necesar.

Cele două faze în implementarea unei RNA sunt faza de antrenare şi faza de utilizare propriu-zisă a RNA (numită şi faza de propagare).

5.2 Antrenarea RNA

Antrenarea unei reţele neuronale se face folosind un set de date de intrare şi setul de date de ieşire corespondent (numite şi valori ţintă). RNA îşi modifică ponderile pentru a învăţă asociaţiile corecte între intrări şi ieşiri. Există multe metode de antrenarea a RNA dar cele mai cunoscute sunt metoda gradientului descendent şi metoda sau algoritmul Hebbian, [218]. În primul caz ponderile sunt ajustate ţinând cont de gradientul funcţiei de cost conform relaţiei 5.2 [53], [217].

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{5.2}$$

unde η reprezintă rata de învăţare, iar δ_j gradientul local.

În cel de al doilea caz ponderile sunt ajustate ţinând cont de valoarea dorită a ieşirii (t_i) ca în relaţia 5.3 [101].

$$\Delta w_{ij}(n) = \eta\, t_i(n) x_j(n) \tag{5.3}$$

Indiferent de algoritmul de antrenare ales aceasta se poate face în timp real pe chip sau off-line. În prezenta teză s-a optat pentru cea de a doua variantă datorită avantajelor legate de simplitatea implementării şi a versatilităţii atât de necesare în faza de dezvoltare a prototipurilor.

5.3 Paralelizarea RNA FF

După determinarea ponderilor următoarea etapă în proiectarea unei RNA este maparea structurii RNA în hardware. Există mai multe posibilităţi de mapare în funcţie de gradul de paralelism, aşa cum au fost prezentate în capitolul II.

Paralelismul straturilor însemnă că există cel puţin un element de procesare pe strat. Figura 5.2 prezintă o RNA compusă din 3 straturi.

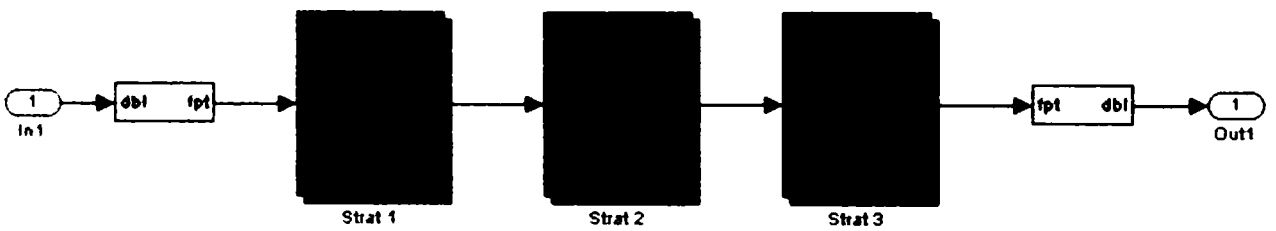


Figura 5.2 Paralelism de strat pentru o RNA cu 3 straturi

Fiecare strat are structură asemănătoare cu a unui neuron, cu următoarele caracteristici specifice:

- Memoria de date este comună tuturor neuronilor şi are NxM locaţii, unde N este numărul de neuroni din strat şi M numărul de intrări ai neuronului;
- Memoria de ponderi este comună tuturor neuronilor şi are dimensiunea Nx(M+1);
- Datele sunt transferate succesiv între straturi.

Paralelismul de neuron este cel mai obişnuit mod de implementare paralelă a RNA. Acest mod de implementare are următoarele caracteristici:

- Necesită un element de procesare pentru fiecare neuron;
- Fiecare neuron are o memorie de ponderi proprie cu o capacitate de M+1 locaţii (unde M este numărul de sinapse ale neuronului);
- Deşi ieşirile neuronilor sunt disponibile simultan, datele sunt transferate succesiv, deoarece un element de procesare din stratul următor poate calcula doar o singură conexiune la un moment dat;
- Blocul de activare poate fi comun tuturor neuronilor.

Figura 5.3 prezintă un strat neuronal compus din 3 neuroni. Datele de intrare x_0-x_2 se aplică succesiv la intrările fiecărui neuron. Neuronii lucrează în paralel şi după M+1 cicluri de multiplicare (în figură M=3) şi unul de acumulare, ieşirile y_0-y_2 sunt disponibile simultan la ieşirile neuronilor. Multiplexorul transferă succesiv datele la ieşirea stratului RNA.

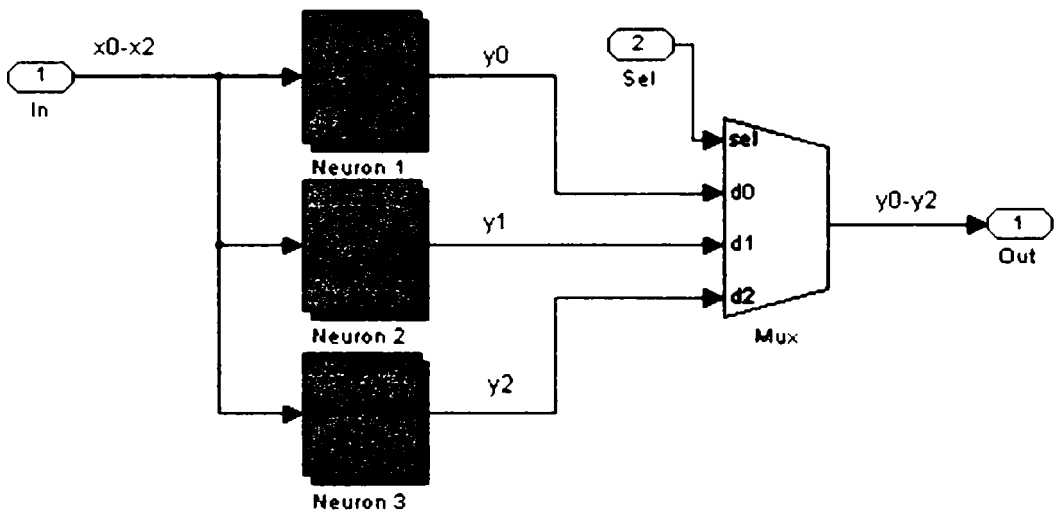


Figura 5.3 Paralelism de neuron pentru un strat RNA cu 3 neuroni cu câte 3 sinapse

Paralelismul de sinapsă reprezintă cel mai mare grad de paralelism, toate conexiunile neuronului fiind calculate în acelaşi timp. Caracteristicile acestei implementări sunt:

- Fiecare sinapsă este mapată la un element de procesare (multiplicator), o reţea de N neuroni cu M sinapse necesită NxM elemente de procesare.
- Datele de intrare şi ponderile trebuiesc furnizate simultan tuturor neuronilor dintr-un strat.
- Transferul de date între straturi are loc în mod paralel deoarece neuronul din stratul următor are nevoie de toate datele de intrare în acelaşi timp.
- Aceasta implementare este cea mai rapidă dar necesită foarte multe resurse şi are o arhitectură fixă (inflexibilă).

Figura 5.4 prezintă un strat neuronal compus din 3 neuroni cu câte 2 sinapse. datele de intrare (x_0, x_1) se aplică simultan tuturor intrărilor. Multiplicatoarele calculează simultan, într-un singur pas, produsele dintre intrări şi ponderile aferente, iar sumatoarele calculează intrarea

netă a fiecărui neuron. Ieşirile neuronilor, în format paralel, se obţin după aplicarea funcţiei de activare asupra intrării nete.

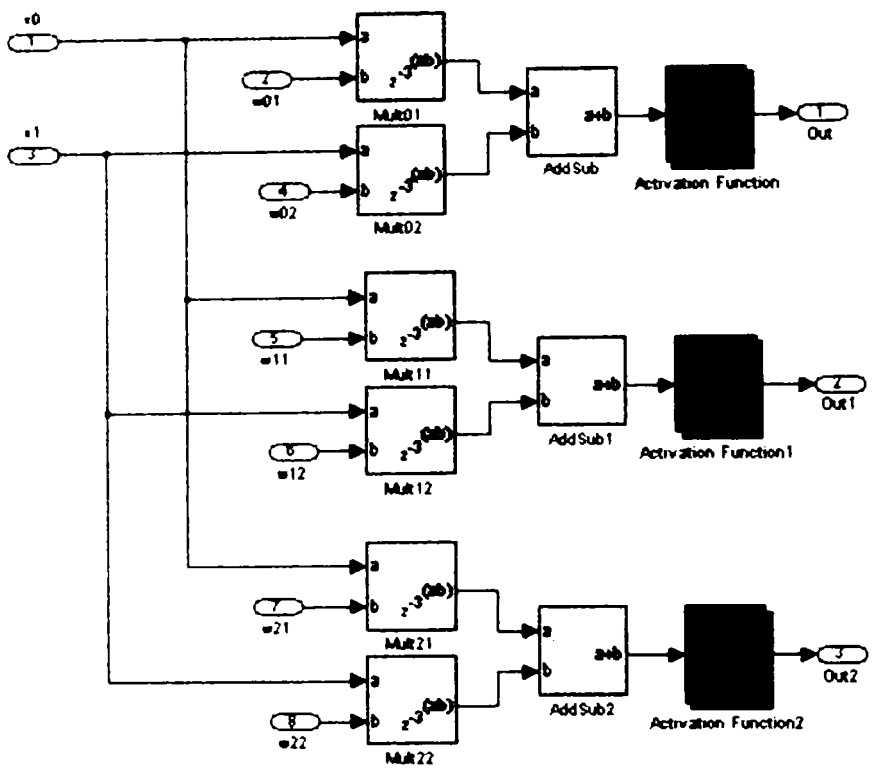


Figura 5.4 Paralelism de sinapsă (strat RNA cu 3 neuroni cu câte 2 sinapse)

Deşi paralelismul de sinapsă este caracterizat de cea mai mare viteză, datorită dezavantajelor sale legate de utilizarea unor resurse foarte mari şi a inflexibilităţii, această implementare nu face obiectul prezentei teze.

În continuare se vor prezenta celelalte două tipuri de implementări (din punct de vedere a gradului de paralelism) cu accentuarea paralelismului de neuron.

5.4 Implementarea unei RNA FF cu paralelism de neuron

Această implementare reprezintă un compromis între performanţele de viteză, pe de o parte, respectiv, resursele necesare implementării şi flexibilitate, pe de altă parte. Din acest motiv acesta este tipul de paralelizare ales pentru implementare.

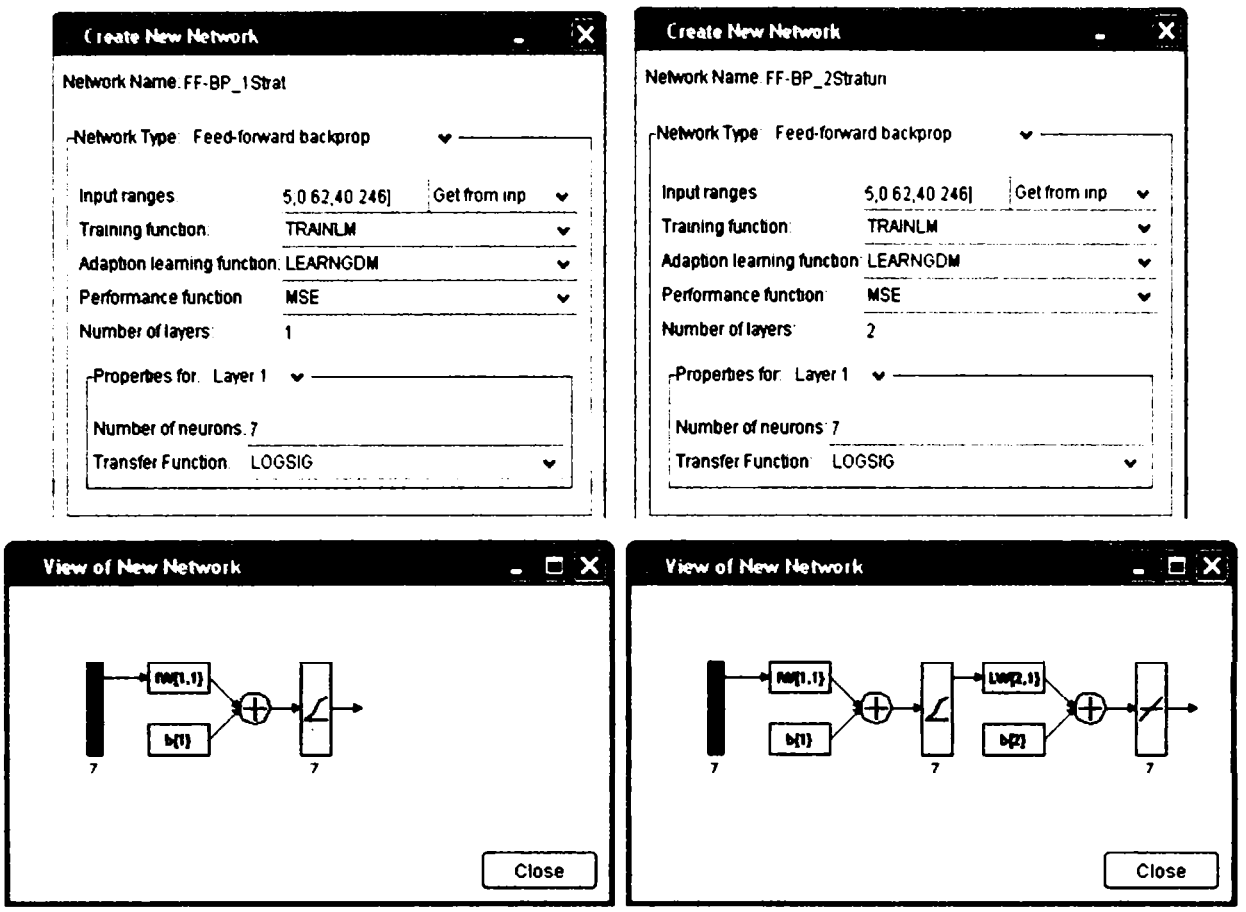
După cum s-a prezentat în capitolul IV, în prezenta teză s-a optat pentru implementarea software a fazei de antrenare a RNA, respectiv implementarea hardware a fazei de propagare. În continuare se prezintă modul de antrenare respectiv implementarea hardware a fazei de propagare.

5.4.1 Antrenarea unei RNA FF

Reţelele neuronale implementate au fost antrenate prin mai multe metode de antrenare. Una din metodele de antrenare testate este metoda Levenberg-Marquardt, care face parte din categoria algoritmilor rapizi de antrenare a RNA FF-BP.

Cu ajutorul NNToolbox pot fi create diverse reţele neuronale pentru a identifica acea reţea care răspunde cel mai bine aplicaţiei date. Reţeaua neuronală poate fi creată folosind

interfața grafică așa cum se prezintă în figura 5.5. În figură se pot vedea două rețele create, prima compusă dintr-un strat cu 7 neuroni (figura 5.5.a), iar cea de a doua are două straturi cu câte 7 neuroni (figura 5.5.b).



a) b)

Figura 5.5 Exemple de rețele neuronale create folosind Network/Data Manager

Figura 5.6 prezintă rezultatele antrenării celor două rețele pentru același set de perechi de date de intrare – ieșire. În cazul rețelei cu un singur strat se observă că nu se poate atinge criteriul de performanță stabilit, în timp ce la cel de al doilea tip de rețea această performanță se atinge după 204 epoci de antrenare.

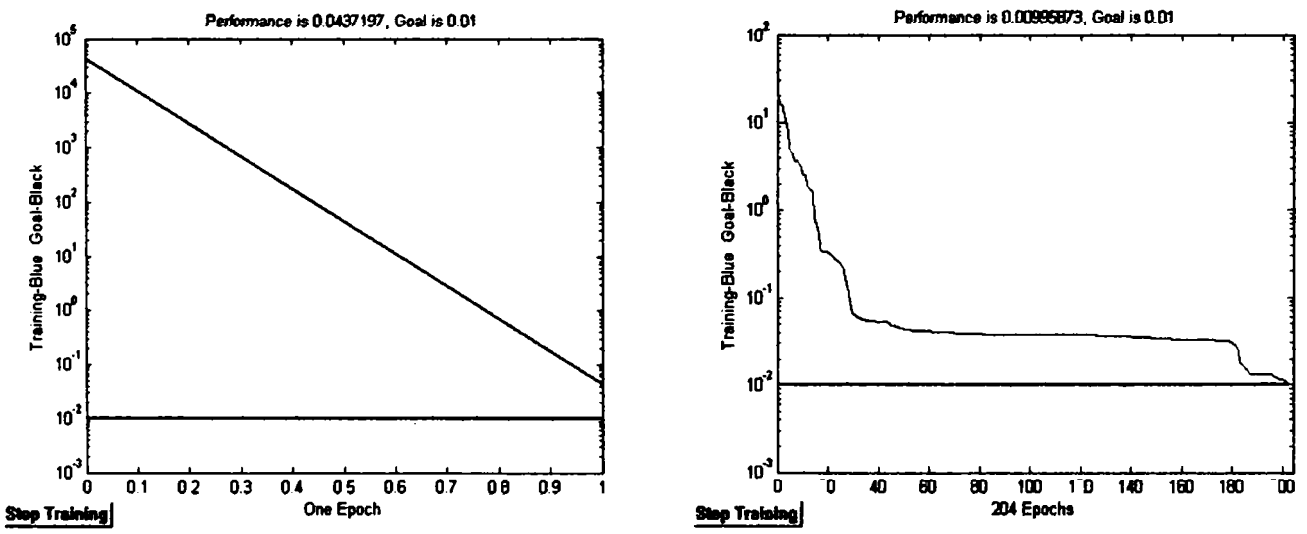


Figura 5.6 Antrenarea rețelelor create

Rețelele pot fi create și antrenate și fără a folosi interfața grafică, cu ajutorul unui cod Matlab. În continuare se prezintă codurile corespunzătoare celor două rețele de mai sus.

```
set_instruire;          % incarca setul de date
net=newff(minmax(c),[7],{'logsig'},'trainlm');
net.trainParam.epochs = 10;
net.trainParam.show = NaN;
net.trainParam.goal = 1e-2;
   Antrenarea RNA
 [net,tr]=train(net,c,d);
   Salvarea ponderilor si a bias-ului
ponderi=[net.iw{1,1} net.b{1}];
save pond ponderi   % se salveaza in pond.mat
```

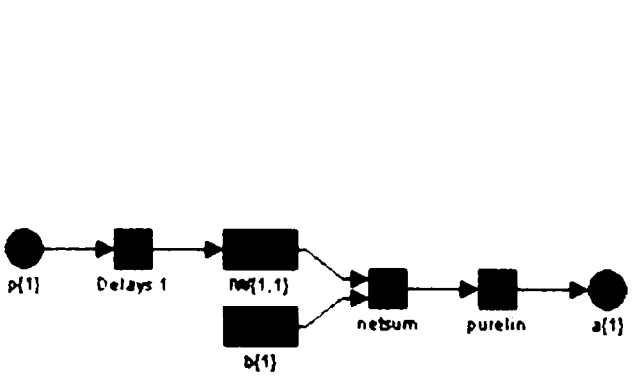
a) RNA cu un strat cu 7 neuroni

```
set_instruire; % incarca setul de date
net=newff(minmax(c),[7 7],{'tansig' , 'purelin'},'trainlm');
net.trainParam.epochs = 500;
net.trainParam.show = 25;
net.trainParam.goal = 1e-2;
%  Antrenarea RNA
 [net,tr]=train(net,c,d);
% Salvarea ponderilor si a bias-ului
ponderi=getx(net);
save pond ponderi   % se salveaza in pond.mat
```

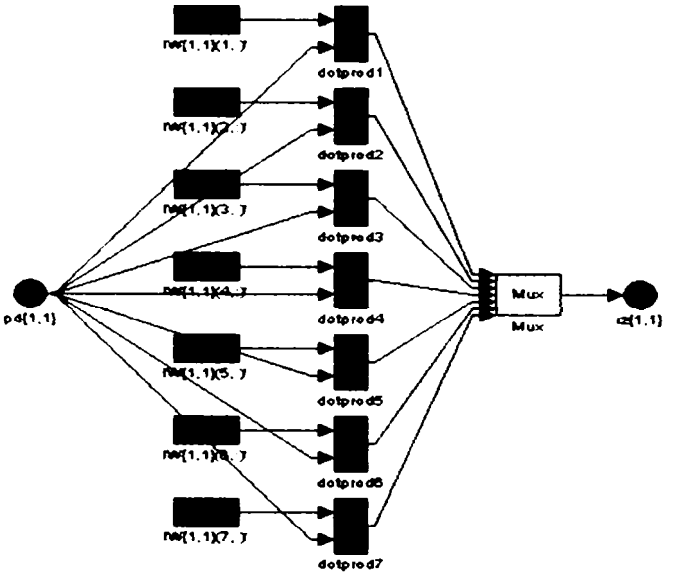
b) RNA cu 2 straturi cu câte 7 neuroni

Figura 5.7 Crearea și antrenarea rețelelor neuronale cu ajutorul unui cod Matlab

Ponderile rețelei antrenate se pot salva într-un fișier ponderi cu ajutorul ultimelor două instrucțiuni din figura 5.7. După antrenare rețeaua poate fi încărcată și în Simulink pentru simulări. Acest model poate servi ca etalon pentru modelul care urmează să fie dezvoltat folosind Sistem generator. Cele două modele pot fi simulate simultan pentru a observa eventualele diferențe.



a) Stratul 1 al RNA FF



b) Blocul IW{1,1}

Figura 5.8 Modelul rețelei în Simulink

5.4.2 Implementarea hardware a RNA FF cu paralelism de neuron

Pentru implementarea unui strat neuronal cu paralelism de neuron trebuiesc respectate toate particularităţile enumerate în paragraful 5.3. Pentru aceasta se modifică structura neuronului prezentat în capitolul anterior astfel încât aceasta va conţine numai un bloc MAC şi o memorie pentru ponderi, aşa cum se prezintă în figura 5.9.

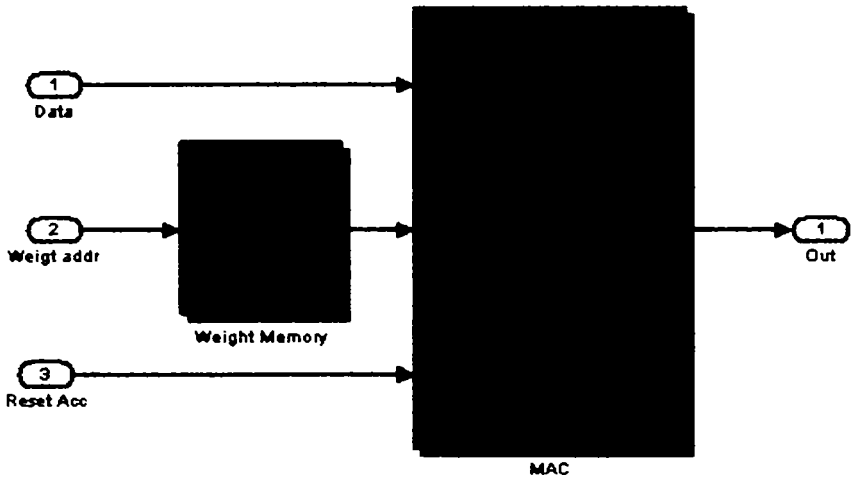


Figura 5.9 Neuron modificat pentru implementarea paralelismului de neuron

Folosind neuroni astfel modificaţi, o memorie de date şi un bloc de control logic se poate implementa un strat neuronal cu un număr oarecare de neuroni. Figura 5.10 prezintă un strat neuronal compus din 7 neuroni. Deoarece neuronii stratului următor nu au nevoie de toate datele simultan, multiplexorul transmite datele de ieşire a stratului RNA în mod succesiv. În consecinţă blocul funcţiei de activare poate fi comun tuturor neuronilor din strat.

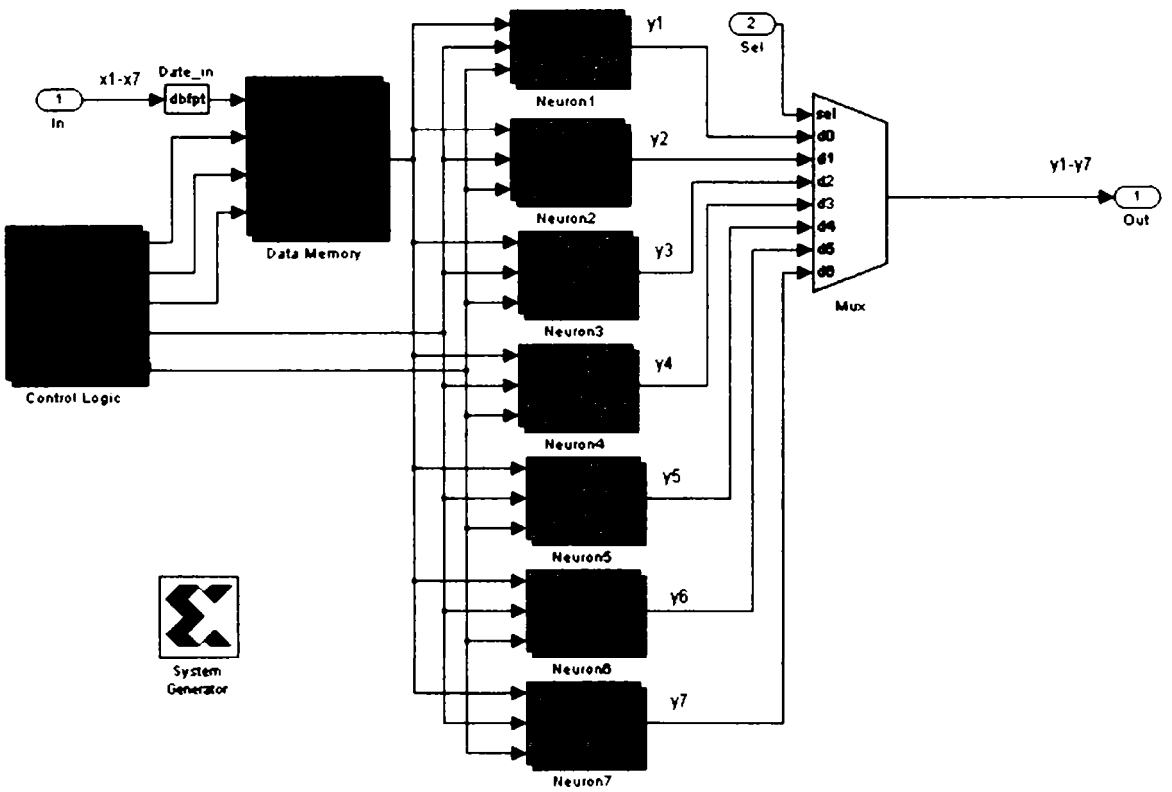


Figura 5.10 Modelul hardware al unui strat neuronal compus din 7 neuroni

Memoria de date, de tip dual port RAM, are 8 locaţii (M+1, unde M este dimensiunea vectorului de intrare, iar 1 reprezintă bias-ul). Ea este înscrisă cu datele prezentate la intrarea RNA cu viteza cu care datele sunt furnizate şi este citită la o viteză de M+1 ori mai mare, astfel încât pe perioada înscrierii ultimului element al vectorului de intrare la ieşirea B a memoriei sunt furnizate cele 7 intrări plus bias-ul (figura 4.11).

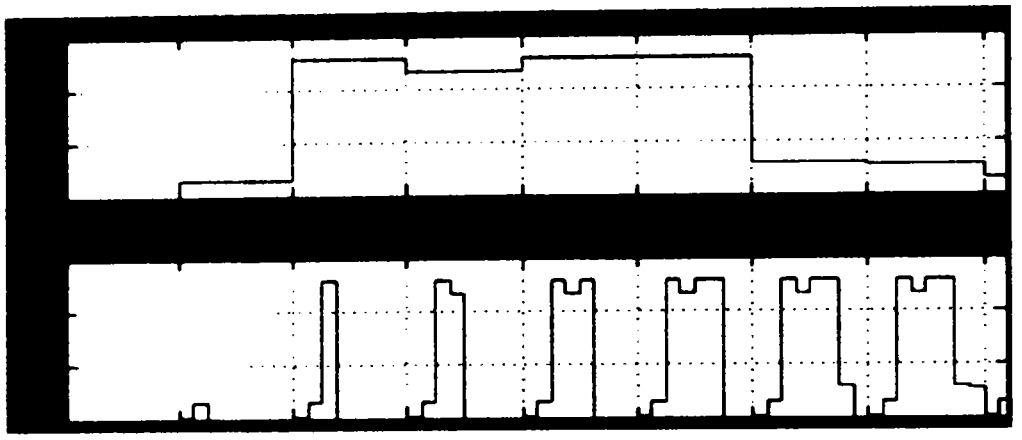


Figura 5.11 Datele de intrare la ieşirile A şi B ale memoriei de date

Blocul de comandă asigură corelarea datelor de intrare cu ponderile, şi asigură semnalele de comandă pentru MAC. Figura 5.12 prezintă formele de undă obţinute la simularea modelului din figura 5.10 pentru neuronul 1.

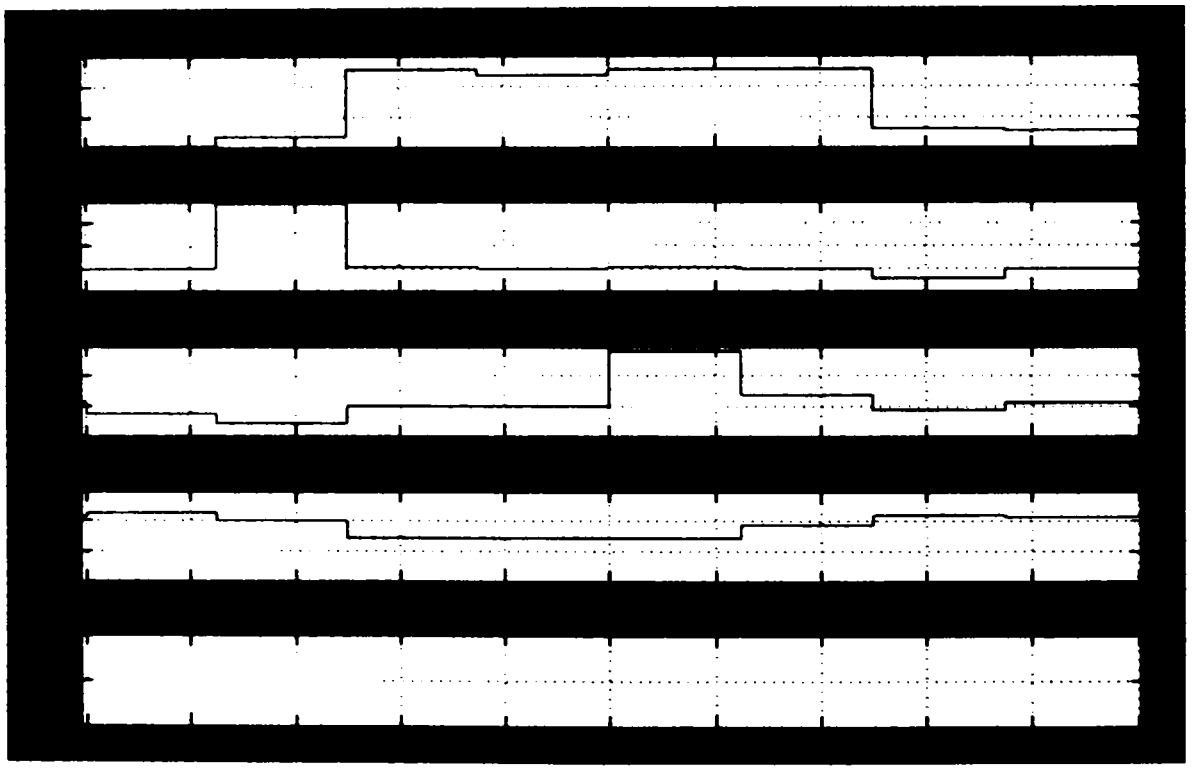


Figura 5.12 Formele de undă pentru Neuron1

Figura 5.13. prezintă formele de undă obţinute la simularea stratului neuronal din figura 5.10 pentru un set de perechi date intrare-ieşire:

$$x = [32, 255, 232, 255, 255, 61, 54],\ y = [1, 7, 7, 7, 7, 1, 1].$$

Se remarcă că după sosirea ultimului element al vectorului de intrare, datele de intrare cărora li s-a adăugat bias-ul sunt prezentate la intrarea MAC. Simultan cu sosirea la intrare a unui nou vector de date, la ieşire sunt furnizate datele asociate vectorului de intrare precedent.

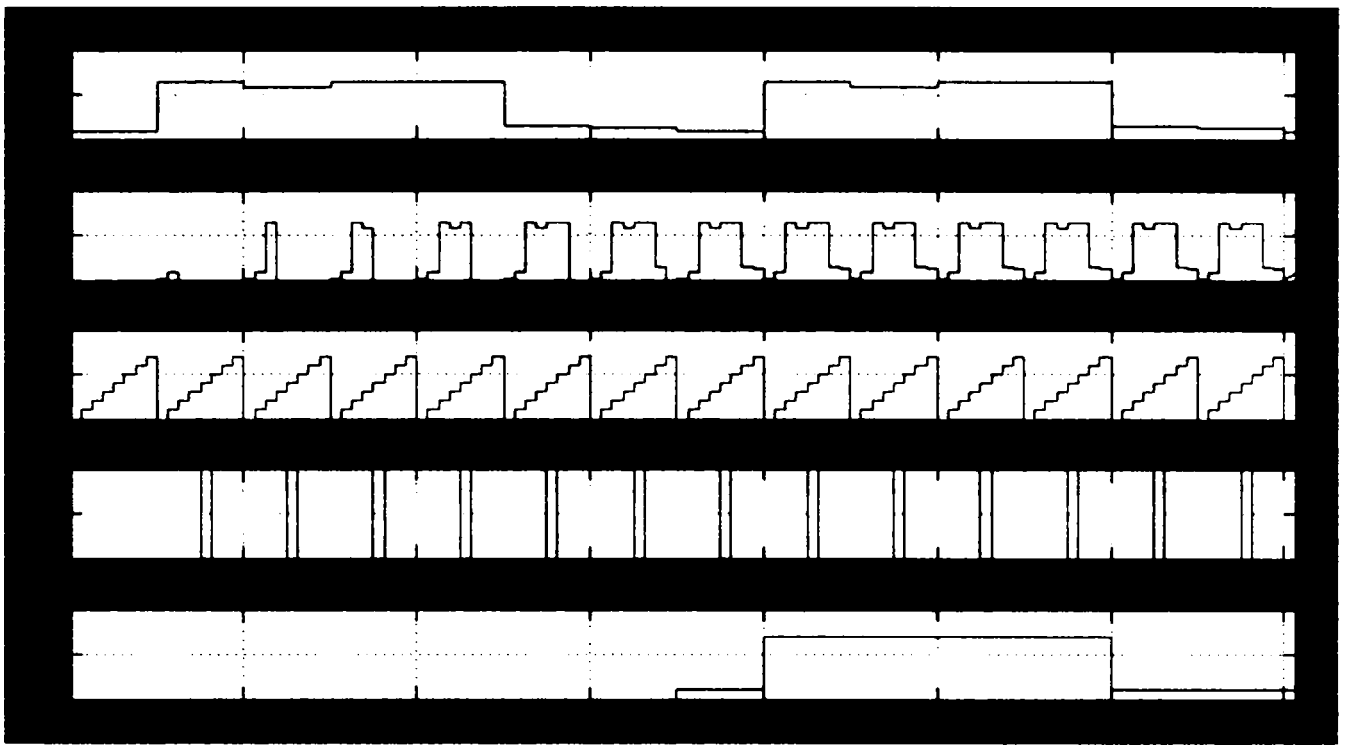


Figura 5.13 Formele de undă corespunzătoare stratului neuronal din figura 5.10

Erorile la ieşirea RNA pentru setul de antrenare sunt reprezentate în figura 5.14. Linia albastră reprezintă erorile modelului rețelei create şi simulate cu NNtools iar linia roşie erorile modelului Simulink creat cu System Generator. Numărul total al erorilor este 49 din 1050 deci procentul de asociere corectă este de 95,33%.

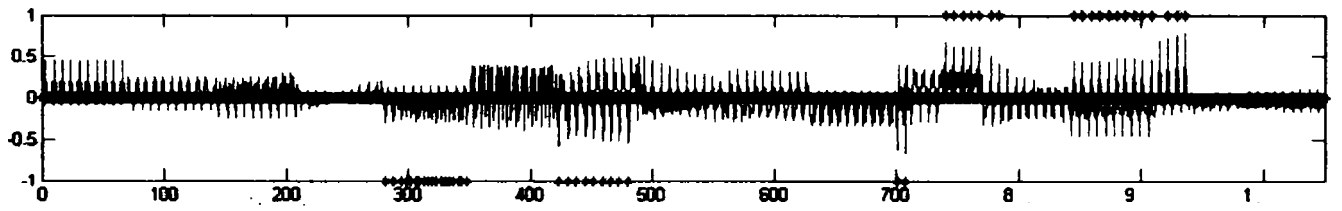


Figura 5.14 Erorile la ieşirea RNA

Răspunsul rețelei fiind cel dorit se poate trece la generarea proiectului Xilinx si implementarea acestuia.

Simularea funcțională folosind ModelSim XE (figura 5.15) arată comportarea identică a proiectului Xilinx cu modelul Simulink.

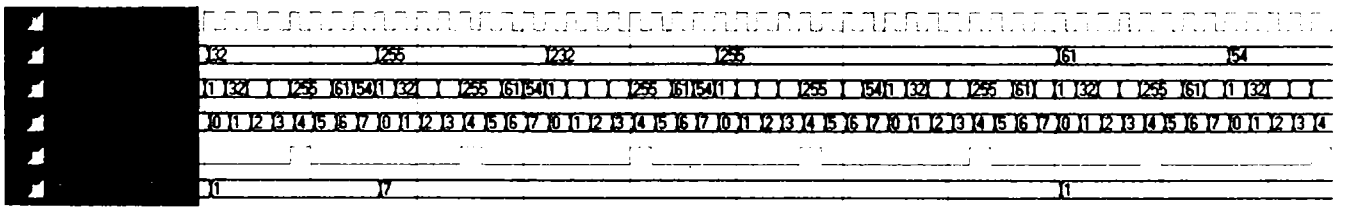


Figura 5.15 Simulare funcțională strat neuronal

5.4.2.1 Implementarea RNA cu algoritm de antrenare Hebbian

Metoda de antrenare Hebbiană supervizată este una din metodele folosite în antrenarea reţelelor asociative. În continuare se prezintă o reţea neuronală artificială compusă dintr-un strat cu 7 neuroni, antrenată folosind regula de învăţare Hebbiana.

Regula Hebbiană calculează produsul dintre intrări şi ţinte pentru a calcula ponderile. În acest exemplu se implementează o reţea asociativă care asociază vectorului de intrare x vectorul de ţintă t, modificarea ponderilor este dată de relaţia 5.3, astfel încât pentru η=1 actualizarea ponderilor este dată de relaţia:

$$w_{ij} = w_{ij} + t_i x_j \tag{5.4}$$

Pentru ponderi iniţiale nule, matricea ponderilor se determină cu relaţia:

$$W = t_1 x_1^T + t_2 x_2^T + ... + t_m x_m^T = \sum_{q=1}^{m} t_q x_q^T \tag{5.5}$$

sau sub formă matricială:

$$W = TX^T \tag{5.6}$$

Relaţia de mai sus se aplică în cazul în care vectorii de intrare sunt ortogonali. Dacă nu sunt ortogonali ei pot fi ortogonalizaţi prin diverse metode cum ar fi şi transformarea Householder. Se poate astfel determina o matricea W a ponderilor care satisface relaţiile:

$$Wx_1' = t_1,\ Wx_2' = t_2,\ .....,\ Wx_m' = t_m \tag{5.7}$$

sau sub formă matricială:

$$WX' = T \tag{5.8}$$

unde x_i' reprezintă vectorii de intrare ortogonalizaţi iar X' şi T matricile construite din vectorii coloană respectivi. Deoarece X' este o matrice ortogonală inversa ei este egală cu transpusa ei, astfel încât matricea ponderilor poate fi calculată simplu cu ajutorul relaţiei:

$$W = Y(X')^T \tag{5.9}$$

Există diverse variante ale algoritmului de antrenare Hebbiană care oferă anumite îmbunătăţiri. Pentru a analiza performanţă unei reţele se poate folosi relaţia:

$$F(W) = \sum_{q=1}^{m} \left\| t_q - Wx_q \right\|^2 \tag{5.10}$$

În formă matricială relaţia de mai sus se scrie:

$$F(W) = \left\| T - WX \right\|^2 = \left\| E \right\|^2 \tag{5.11}$$

unde :

$$\|E\|^2 = \sum_i \sum_j e_{ij}^2 \tag{5.12}$$

Pentru a îmbunătății performanțele rețelei trebuie minimizată funcția F. Dacă există o inversă pentru matricea X, F(W) va avea valoarea zero pentru:

$$W = TX^{-1} \tag{5.13}$$

Dacă nu există o inversă a matricei X, F(W) poate fi minimizată folosind pseudoinversa matricei X. Astfel matricea ponderilor se calculează cu relația:

$$W = TX^{+} \tag{5.14}$$

unde:

$$X^{+} = (X^T X)^{-1} X^T \tag{5.15}$$

Programul Matlab permite determinarea facilă a matricei ponderilor folosind funcția *pinv* care calculează pseudoinversa unei matrici:

$$W = T * pinv(X) \tag{5.16}$$

Pentru o pereche compusă din matricea X a vectorilor de antrenare de dimensiune 15 x 7 și matricea T a vectorilor țintă de dimensiune 15 x 7:

32	4	37	247	28	39	3	36	51	17	1	45	15	13	34
255	2	0	39	3	33	9	21	6	255	192	12	2	56	252
232	233	0	0	0	21	9	10	0	255	28	23	31	194	58
X= 255	255	255	12	15	35	29	227	57	255	205	209	240	255	255
255	255	255	12	24	1	55	15	238	255	223	209	228	31	255
61	61	1	58	59	23	35	45	27	58	57	8	57	13	13
54	50	51	60	54	246	239	218	220	54	41	236	59	50	52

1	0	1	7	0	1	0	1	1	0	0	1	0	0	1
7	0	0	1	0	1	0	0	0	7	6	0	0	1	7
7	7	0	0	0	0	0	0	0	7	0	0	0	6	1
T= 7	7	7	0	0	1	0	7	1	7	6	6	7	7	7
7	7	7	0	0	0	1	0	7	7	6	6	7	0	7
1	1	0	1	1	0	1	1	0	1	1	0	1	0	0
1	1	1	1	1	7	7	6	6	1	1	7	1	1	1

matricea ponderilor rezultă:

Tabelul 5.1 Matricea ponderilor

	0.028975	0.00088032	-0.00033009	0.00018113	-0.00044445	-0.0043131
	-0.0010741	0.028859	-0.0014851	-0.0012257	0.0012437	0.0029301
	0.0017167	-0.0018789	0.032755	-0.0011173	-0.0001889	-0.0035418
W=	-0.0013623	0.00024185	-0.0020491	0.030918	-0.0019598	0.00070352
	-0.00053585	-0.00079111	-0.00065214	-0.0015604	0.030691	-0.0027596
	-0.0010582	-7,61380E-05	-0.00081411	0.00056685	-0.00084007	0.021023
	-0.0023104	0.00028514	-2,50820E-06	-0.0010169	-0.00021267	-0.0052287

Ponderile astfel calculate se încarcă într-o rețea nou creată folosind Neural Network Toolbox compusă dintr-un strat cu 7 neuroni, ca cea din figura 5.8. Rezultatele simulării, comparate cu vectorul țintă permit determinarea erorilor rețelei.

Erorilor rețelei antrenate se determină cu ajutorul relațiilor:

$$Y=\text{round}(\text{sim}(\text{net},C)) \tag{5.17}$$

$$\mathbf{E=Y-T}$$

unde C reprezintă matricea vectorilor de test, Y matricea ieşirilor reţelei, T matricea vectorilor ţintă, iar E matricea erorilor.

Astfel RNA antrenată prin algoritmul hebbian şi având ponderile reprezentate în precizia maximă (double precision= 64 biţi în virgulă mobilă), generează la ieşire 3 vectori eronaţi ca răspuns la cei 15 vectorii de test aplicaţi la intrare adică un procent de recunoaştere de 80%. Numărul de ieşiri eronate este de 4 din 105 (15 vectori cu câte 7 elemente) adică un procent de asociere corectă de 96,19%.

După cum s-a prezentat în capitolele anterioare, precizia de calcul a ponderilor trebuie să fie cât mai mare pentru a obţine convergenţa RNA şi ponderi sinaptice corecte şi deci mai puţine erori de cuantizare în implementarea finală. De aceea faza de învăţare se pretează a fi implementată software, folosind reprezentarea în virgulă mobilă ceea ce duce la convergenţa şi acurateţea reţelei. Precizia ponderilor în faza de propagare poate fi micşorată faţă de precizia calculată în faza de învăţare, erorile rezultate fiind neglijabile. Pentru calculul erorilor reţelei la diferite precizii de reprezentare a ponderilor am elaborat un cod Matlab care este prezentat în Anexa 3. Astfel reducând precizia de reprezentare a ponderilor şi simulând din nou reţeaua se obţine următorul grafic al erorilor în funcţie de numărul de biţi:

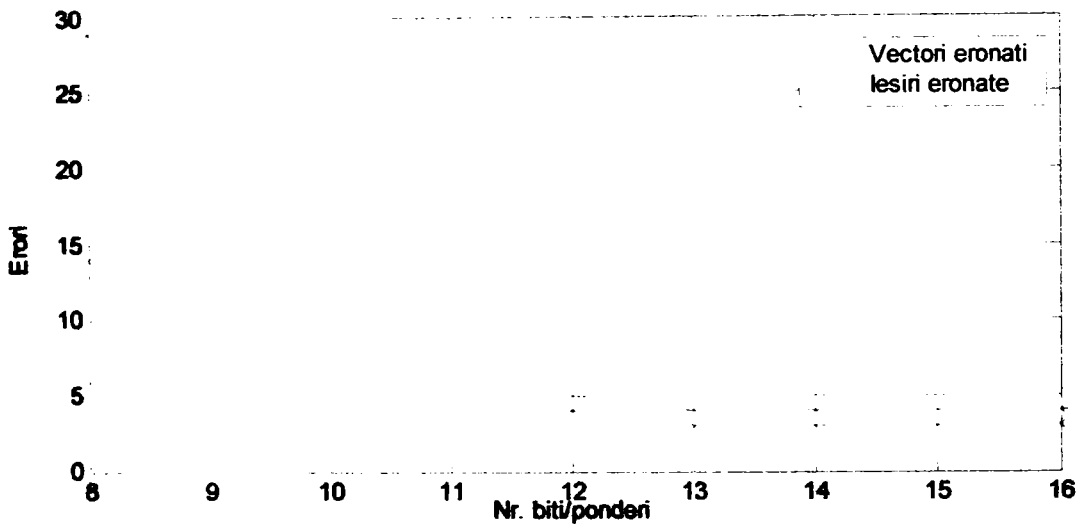


Figura 5.16 Erorile modelului software al RNA pentru setul de 15 vectori de test

Graficul cu linie roşie reprezintă numărul de elemente eronate din matricea ieşirii, Y, iar cel de culoare albastră numărul de vectori eronaţi.

Se poate observa că şi la o reprezentare pe 11 biţi se obţine numărul minim de 3 vectori de ieşire eronaţi, o creştere a numărului de biţi folosiţi pentru reprezentarea ponderilor peste acest număr nu duce la scăderea numărului de vectori eronaţi.

Aceeaşi reţea neuronală, implementată folosind modelul hardware creat în System Generator, simulat pentru diverse precizii de reprezentare a ponderilor conduce la graficul erorilor din figura 5.17. Din nou a fost necesară elaborarea unui cod Matlab care rulează în mod repetat modelul RNA pentru diverse precizii ale ponderilor. Acest cod este prezentat în anexa 4.

Se poate observa ca la reprezentarea pe 11 biţi a ponderilor se obţine un minim de 2 vectori de test eronaţi, o performanţă chiar mai bună decât a reţelei implementate software. Este de reţinut că numărul erorilor nu este mai mic la o reprezentare a ponderilor pe 64 de biţi.

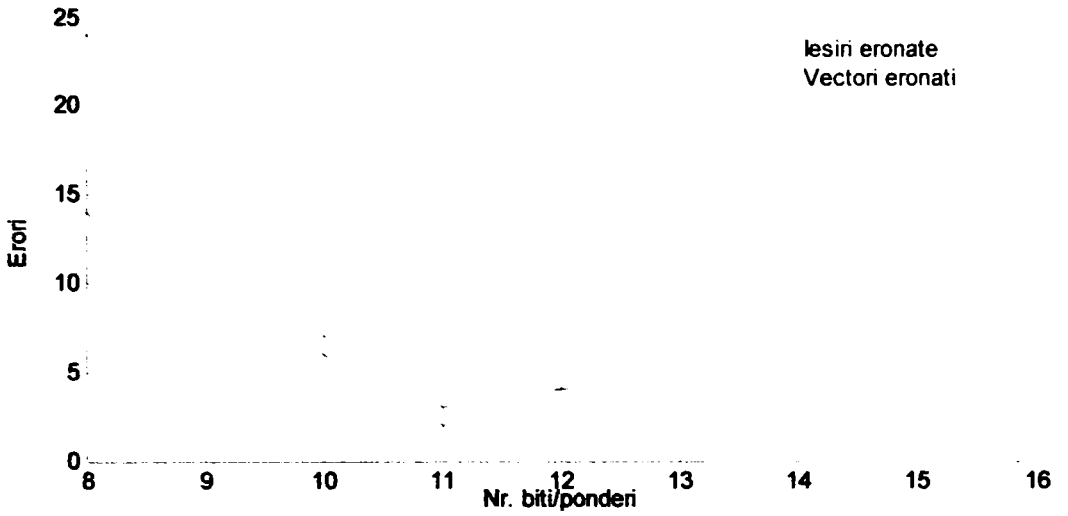


Figura 5.17 Erorile modelului hardware al RNA pentru setul de 15 vectori de test

În tabelul următor se prezintă influența numărului de biți utilizați la reprezentarea ponderilor asupra erorilor dar şi asupra resurselor utilizate pentru implementarea rețelei, precum şi a frecvenței maxime de lucru a circuitului implementat.

Tabelul 5.2 Performanțele şi resursele RNA pentru diverse precizii

Număr biți/ponderi	Poziții eronate	Vectori eronați	Slice-uri	Frecvența maximă (MHz)
8	24	14	363	110.135
9	9	7	436	106.516
10	7	6	459	105.371
11	**3**	**2**	**467**	**104.226**
12	4	4	509	103.597
13	3	3	510	103.005
16	3	3	575	99.855

Din tabel rezultă că reprezentarea pe 11 biți a ponderilor este un optim atât în ceea ce priveşte numărul de erori cât şi al resurselor utilizate, respectiv al frecvenței maxime de lucru.

Reluând simulările şi experimentele pentru un alt set de vectori de antrenare şi de test de dimensiune 150 x 7, obținem următorul grafic al erorilor în funcție de numărul de biți, pentru modelul software al RNA:

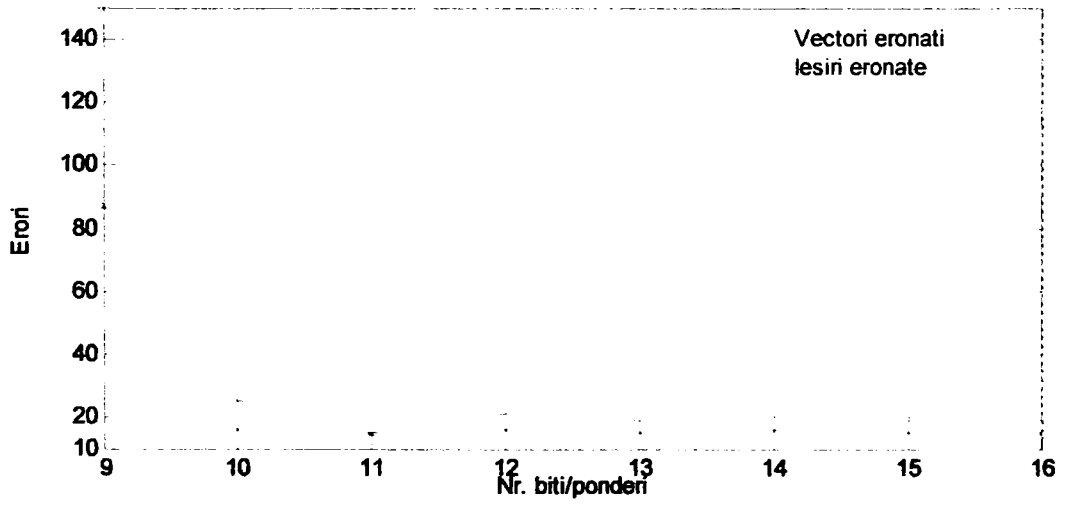


Figura 5.18 Erorile modelului software al RNA pentru setul de 150 vectori de test

Din nou se poate constata că numărul de erori este minim (14 vectori) pentru o reprezentare pe 11 biţi a ponderilor. Procentul de asociere corectă a vectorilor în acest caz este de 90,66%.

În cazul modelul hardware (System Generator), pentru diverse precizii de reprezentare a ponderilor obţinem acelaşi număr de erori ca şi în cazul simulării. Figura 5.19 prezintă un detaliu pentru precizia de reprezentare a ponderilor cuprinsă între 10 şi 16 biţi/ponderi.

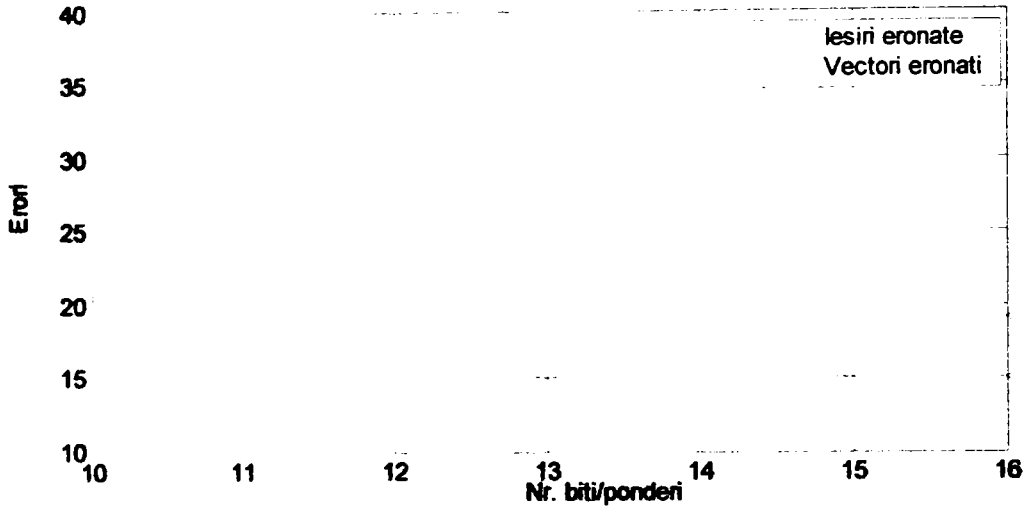


Figura 5.19 Erorile modelului hardware a RNA pentru setul de 150 vectori de test

Minimul erorilor se obţine pentru o precizie de 11 biţi a ponderilor. Tabelul următor prezintă rezultatele implementării hardware a reţelei pentru diverse precizii a ponderilor.

Tabelul 4.3 Performanţele şi resursele RNA pentru diverse precizii

Număr biţi/ponderi	Poziţii eronate	Vectori eronaţi	Slice-uri	Frecvenţa maximă (MHz)
8	446	145	336	117.976
9	150	87	364	114.151
10	25	16	378	113.362
11	**15**	**14**	**399**	**112.583**
12	21	16	420	111.815
13	19	15	441	111.092
14	20	16	455	110.377
15	19	15	483	109.705
16	19	15	497	109.074

Se poate face o comparaţie a erorilor reţelei implementate software şi a celei hardware în funcţie de vectorii de test pentru precizia de 11 biţi (figura 5.20). Se observă că erorile sunt identice şi ele sunt concentrate la ultimii 10 vectori de test (care sunt identici). Acest vector nu a putut fi „învăţat" de RNA prin metoda de instruire utilizată.

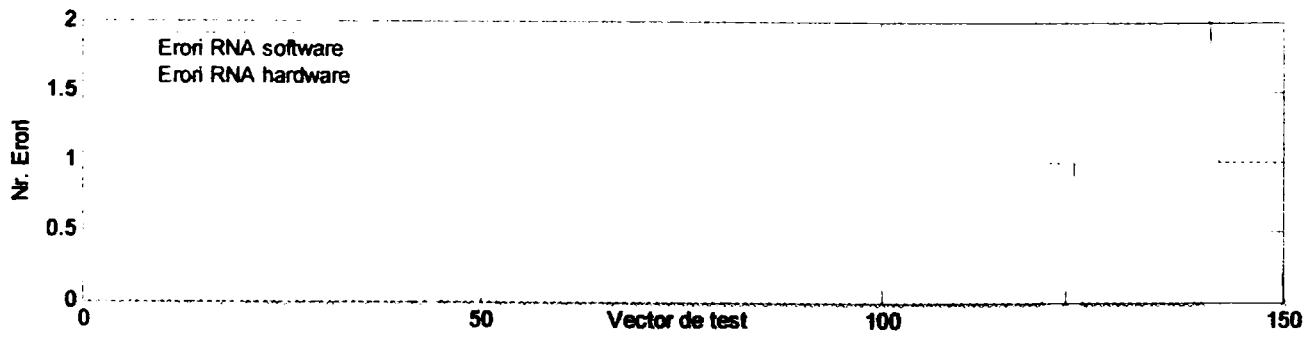


Figura 5.20 Comparaţie între erorile RNA implementate SW şi HW

O analiză a erorilor în funcție de neuronul care răspunde eronat arată că erorile sunt localizate la neuronii 5 şi 6 cu câte 6, respectiv 9, răspunsuri greşite.

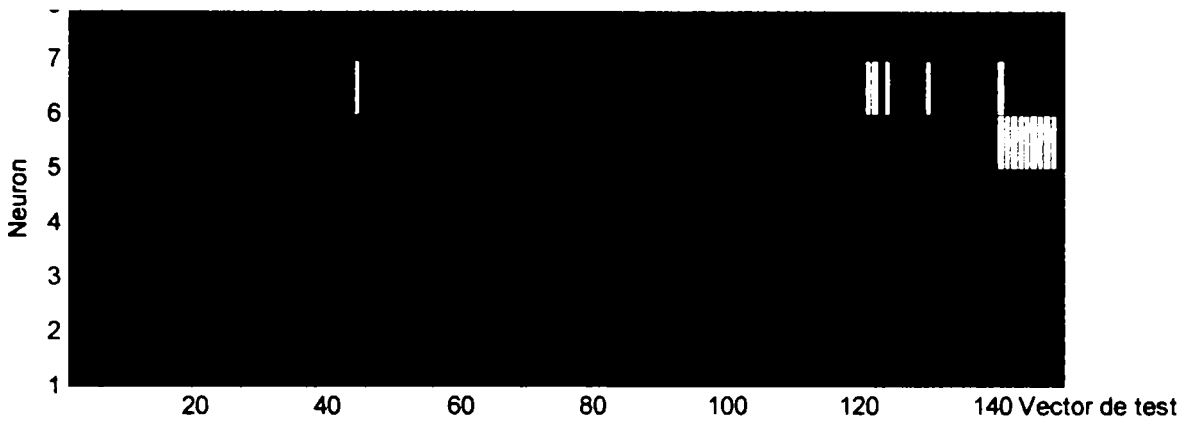


Figura 5.21 Erorile la ieşirea neuronilor pentru o precizie a ponderilor de 11 biţi

Tabelul 5.4 prezintă ponderile calculate conform algoritmului Hebbian de antrenare supervizată, iar tabelul 4.5 ponderile utilizate în implementarea hardware.

Tabelul 5.4 Ponderile RNA calculate conform algoritmului Hebbian de antrenare

	P1	P2	P3	P4	P5	P6	P7
n1	0.02894	0.00027947	0.0011209	-0.016213	0.014308	-0.0022519	0.00069446
n2	-0.00056411	0.030878	-0.0016755	-0.0054595	0.0074901	-0.0026472	7.2017e-005
n3	-0.00079991	0.0030779	0.027673	-0.0067897	0.0088756	0.00077266	-0.002036
n4	-0.0013081	0.00097814	0.003329	0.011071	0.017741	-0.0018778	-0.00059441
n5	-0.0013119	0.0014151	0.0010534	-0.016167	0.046071	-0.0021935	-0.00048481
n6	-0.00097642	0.000264	0.0014618	-0.0059054	0.0043962	0.029967	-0.00094025
n7	-0.00044253	0.00059301	-0.0015293	0.014659	-0.016329	-0.0027892	0.030669

Tabelul 5.5 Ponderile utilizate în implementarea hardware a RNA

	P1	P2	P3	P4	P5	P6	P7
n1	0.028809	0.00048828	0.00097656	-0.016113	0.01416	-0.0024414	0.00048828
n2	-0.00048828	0.030762	-0.0014648	-0.0053711	0.0073242	-0.0024414	0
n3	-0.00097656	0.0029297	0.027832	-0.0068359	0.0087891	0.00097656	-0.0019531
n4	-0.0014648	0.00097656	0.003418	0.01123	0.017578	-0.0019531	-0.00048828
n5	-0.0014648	0.0014648	0.00097656	-0.016113	0.045898	-0.0019531	-0.00048828
n6	-0.00097656	0.00048828	0.0014648	-0.0058594	0.0043945	0.029785	-0.00097656
n7	-0.00048828	0.00048828	-0.0014648	0.014648	-0.016113	-0.0029297	0.030762

O prezentare mai intuitivă a ponderilor este prezentată în figura 5.22.

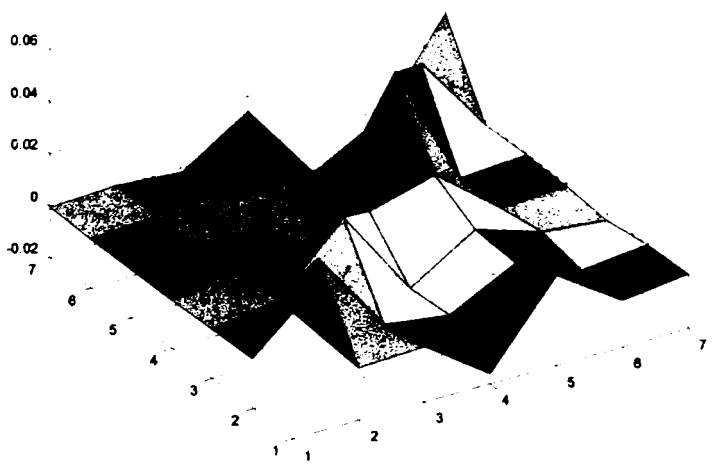


Figura 5.22 Ponderile utilizate în implementarea hardware a RNA

5.4.2.2 Optimizarea RNA cu algoritm de antrenare Hebbian

După identificarea preciziei optime de reprezentare a ponderilor din punctul de vedere al erorilor, se poate trece la optimizarea parametrilor de implementare a reţelei, şi anume minimizarea resurselor utilizate şi maximizarea frecvenţei de lucru. Pentru aceasta se poate face un studiu al variaţiei celor doi parametrii în funcţie de diverşi factori, cum ar fi numărul de biţi ai multiplicatorului, modul cum se face cuantizarea în blocul multiplicator şi convertor, şi cum se tratează depăşirea în multiplicator, acumulator şi convertor. Astfel numărul de biţi ai multiplicatorului influenţează major resursele utilizate de RNA. Numărul de biţi necesari pentru a obţine precizia maximă a multiplicatorului este dată de relaţia:

$$nbm_{\max} = n_i + n_p \tag{5.18}$$

unde n_i şi n_p reprezintă numărul de biţi utilizaţi la reprezentarea datelor, respectiv a ponderilor. Pentru exemplul considerat nbm_{max}=8+11=19. Cu toate că în aparenţă reducerea numărului de biţi ai multiplicatorului ar duce la reducerea resurselor utilizate de multiplicator, în realitate nu se întâmplă aşa în orice situaţie. În cazul utilizării unui număr mai mic de biţi decât cel specificat de relaţia 5.19 trebuiesc luate decizii legate de modul cum se face cuantizarea (prin trunchiere sau rotunjire) respectiv cum se tratează depăşirea (saturare sau wrap). Folosirea opţiunilor de saturare (în cazul depăşirii) şi rotunjire (în cazul cuantizării) necesită logică suplimentară de calcul şi reduc performanţele de viteză.

Numărul minim de biţi necesari pentru reprezentarea corectă a părţii întregi a ieşirii multiplicatorului se poate determina cu relaţia dedusă de autor:

$$nbm_{\min} = \text{ceil}(\log_2(\text{round}(\max(x_{\max} * \max(W), \text{abs}(x_{\max} * \min(W)))))) + 1 \tag{5.19}$$

unde x_{max} reprezintă valoarea maximă posibilă pentru o intrare, W este matricea ponderilor, iar 1 reprezintă bitul de semn. Pentru vectorii şi coeficienţii prezentaţi în paragraful 5.4.2.1: x_{max}=255, min(W)= -0.01632930323365 iar max(coef)= 0.04607123369787, astfel încât: nbm_{min}= 5

Numărul de biţi ai părţii fracţionare se păstrează egal cu numărul de biţi utilizaţi pentru reprezentarea ponderilor (11). Astfel precizia de reprezentare a ieşirii multiplicatorului poate fi modificată între cea maximă Fix_19.11 şi cea minimă Fix_16.11.

Pentru determinarea numărului de biţi ai acumulatorului s-a dedus relaţia:

$$nba_{\max} = (n_i + n_p) + ceil(\log_2(M + 1)) \tag{5.20}$$

unde M reprezintă numărul sinapselor neuronului. Valoarea maximă respectiv minimă la ieşirea acumulatorului este dată de relaţiile:

$$Y_{acc\_max} = \max(W * X) \tag{5.21}$$

$$Y_{acc\_min} = \min(W * X) \tag{5.22}$$

Numărul de biţi necesari reprezentării părţii întregi poate fi redus până la:

$$nba_{\min} = \text{ceil}(\log2(\text{round}(\max(\max(W * X), \text{abs}(\min(W * X)))))) + 1 \tag{5.23}$$

Numărul minim de biţi ai acumulatorului este limitat inferior de constrângerea impusă de blocul Xilinx pentru care numărul de biţi ai ieşirii nu poate fi mai mic de numărul de biţi ai intrării.

O sinteză a rezultatelor obținute în urma implementării RNA pentru diverse setări ale blocurilor multiplicatoare, acumulatoare și convertoare este prezentată în tabelul 5.6:

	Nr. biți multiplic.	Cuantizare multiplic.	Depășire multiplic.	Nr. biti acumulat.	Depășire acumulat.	Cuantizare convert.	Depășire convert.	Slice-uri	F. max. (MHz)
1	nbm_{max}=19	-	-	nba_{max}=22	saturare	rotunjire	saturare	473	112.583
2	-//-	-	-	n_i+n_p=19	saturare	rotunjire	saturare	430	115.018
3	-//-	-	-	-//-	wrap	rotunjire	saturare	353	150.421
4	-//-	-	-	-//-	wrap	trunchiere	saturare	354	149.970
5	-//-	-	-	-//-	wrap	trunchiere	wrap	304	179.122
6	-//-	-	-	-//-	wrap	rotunjire	wrap	269	179.122
7	nbm_{min}=16	trunchiere	saturare	-//-	wrap	rotunjire	wrap	350	162.364
8	-//-	rotunjire	saturare	-//-	wrap	rotunjire	wrap	350	162.364
9	-//-	rotunjire	wrap	-//-	wrap	rotunjire	wrap	269	179.122
10	-//-	trunchiere	wrap	-//-	wrap	rotunjire	wrap	269	179.122
11	-//-	trunchiere	saturare	-//-	saturare	rotunjire	saturare	521	106.216
12	-//-	rotunjire	wrap	nbm_{min}=16	wrap	rotunjire	wrap	269	184.680
13	-//-	trunchiere	wrap	nbm_{min}=16	wrap	trunchiere	wrap	283	184.680
14	nbm_{max}=19	-	-	nba_{max}=22	wrap	rotunjire	wrap	269	174.344

Tabelul 5.6 Dependența resurselor utilizate și a frecvenței de lucru de parametrii blocurilor componente

Linia 11 din tabel prezintă cazul cel mai defavorabil. Cu toate că numărul de biți utilizați pentru multiplicator este mai mic decât cel corespunzător preciziei maxime, logica suplimentară necesară pentru implementarea opțiunii de saturare este costisitoare atât din punct de vedere al resurselor cât și al frecvenței de lucru.

Cazul cel mai avantajos în valori absolute este prezentat în linia 12. Comparând performanțele acestei setări cu cele din lina 6 se constată că resursele utilizate sunt aceleași chiar dacă multiplicatorul și acumulatorul lucrează pe un număr mai mic de biți (16 față de 19), crește însă frecvența de lucru. Totuși cele mai avantajoase setări sunt cele din lina 14 deoarece resursele utilizate sunt cele minime chiar la precizia maximă pentru multiplicator și acumulator, astfel că nu sunt necesare calcule în plus pentru determinarea numărului minim de biți pentru cele două blocuri. Scăderea de performanță în viteză nu este mare.

În continuare se mai pot face optimizări specifice procesului de sinteză și implementare. Astfel se poate stabili ca sinteza să fie optimă din punct de vedere al ariei, ceea ce duce la scăderea în continuare a resurselor utilizate. De exemplu pentru setările din linia 12 implementarea ocupă 255 slice-uri. Dacă se face în continuare și o optimizare a primitivelor instanțiate frecvența de lucru poate crește la 186,916 MHz

În urma implementării rețelei astfel optimizate într-un dispozitiv Virtex-II XC2V1000, se obține următorul raport complet al resurselor utilizate:

Device utilization summary:

Number of Slices:	243	out of	5120	5,12%
Number of Slice Flip Flops:	410	out of	10240	4,35%
Number of 4 input LUTs:	189	out of	10240	2,57%
Number of bonded IOBs:	23	out of	324	10,5%
Number of BRAMs:	8	out of	40	2%
Number of MULT18X18s:	7	out of	40	17%
Number of GCLKs:	1	out of	16	6%

Din raportul de mai sus rezultă că se pot implementa 40 de neuroni, folosind cele 40 de multiplicatoarele dedicate disponibile în circuit.

Resursele utilizate de un neuron sunt de 28 slice-uri un multiplicator dedicat şi un Block RAM.

Frecvenţă maximă de lucru este de 186,916 MHz. Astfel, deoarece se efectuează o înmulţire a datelor de intrare cu ponderea aferentă la fiecare impuls de ceas, un multiplicator poate efectua un număr 186,916 MMAC/s (mega operaţii MAC/s), astfel cele 7 multiplicatoare ale stratului neuronal pot efectua 1308,412 MMAC/s. Numărul maxim de conexiuni pe secundă care poate fi executat de cele 40 de multiplicatoare disponibile în circuit este de 7,477 GCPS.

RNA poate fi implementat şi fără a folosii multiplicatoarele dedicate. Astfel se obţine următorul raport de implementare:

Number of Slices:	469 out of	5120	9,16%
Number of Slice Flip Flops:	782 out of	10240	7,64%
Number of 4 input LUTs:	823 out of	10240	8,04%
Number of bonded IOBs:	34 out of	324	10,5%
Number of BRAMs:	1 out of	40	2%
Number of GCLKs:	1 out of	16	6%

Frecvenţă maximă de lucru este de 155,892 MHz.

La o implementare combinată a blocurilor MAC folosind toate cele 40 de multiplicatoare dedicate precum şi logica distribuită într-un circuit de tip Virtex-II XC2V1000 se pot implementa un număr maxim de aproximativ 101 neuroni la o utilizare aproape completă a resurselor aşa cum se vede în extrasul din raportul de implementare:

Number of MULT18X18s	40 out of	40	100%
Number of SLICEs	5118 out of	5120	99%

Frecvenţa maximă de lucru este de 155,892 MHz deci RNA implementat atinge:

$$101 \times 155{,}892 \text{ MMAC/s} = 15{,}75 \text{ GCPS}$$

După furnizarea ultimului eşantion al vectorului de intrare, calculul intrării nete a neuronilor se face în 4 impulsuri de tact, adică jumătate din perioada vectorului de intrare. Elementele vectorului rezultat pot fi transmise succesiv stratului următor.

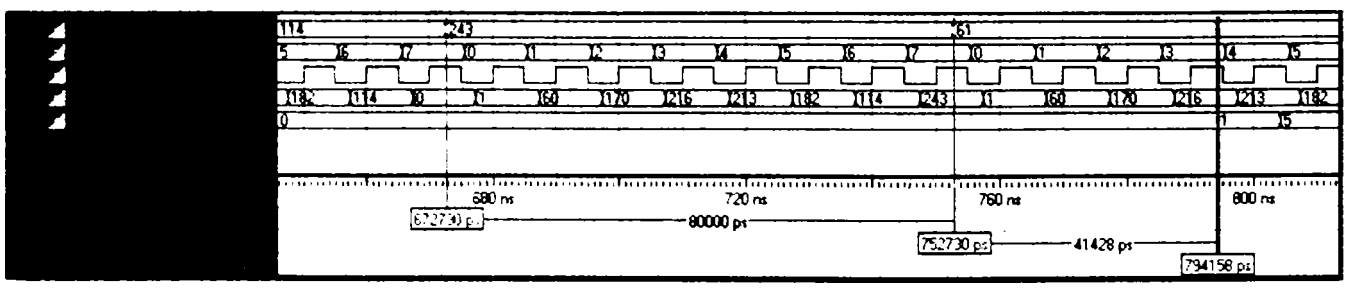


Figura 5.23 Simularea modelului hardware a RNA

Frecvenţa maximă a semnalului de intrare depinde de numărul de conexiuni ale neuronilor din strat (M) şi este dată de relaţia:

$$f_{max} = f_{clk} / (M+1) \qquad (5.24)$$

$$f_{max} = 186{,}916 / 8 = 23{,}365 \text{ MHz}$$

O reducere mult mai importantă a resurselor utilizate se obține dacă timpul de latență al multiplicatoarelor se reduce de la 3 la unu. Astfel resursele necesare pentru implementarea unui neuron scad de la 28 la 12 slice-uri, scade şi timpul necesar pentru calculul răspunsului la două impulsuri de ceas.

5.4.2.3 Implementarea RNA FF cu propagarea înapoi a erorii

Algoritmul de antrenament cu propagare înapoi a erorii (BP, backpropagation) presupune modificarea ponderilor pentru minimizarea funcției cost. Algoritmul de antrenare standard prezintă dezavantaje, cum ar fi convergența lentă, influențată de parametrii rețelei. Algoritmii rapizi de antrenare converg spre valorile optime în mai puține epoci. Dezavantajul lor constă în complexitatea mai mare a calculelor, astfel că numărul mai mic de epoci de antrenare nu însemnă neapărat şi un timp mai scurt. Unul din algoritmii rapizi de antrenare este Levenberg-Marquardt (LM), care a fost folosită în cadrul acestei teze pentru antrenarea rețelelor implementate.

Algoritmul Levenberg-Marquardt, ca şi metoda cvasi-Newton, elimină necesitatea calculării hessianului (derivata parțială de ordinul 2 a funcției cost) utilizat de metoda Newton. Metoda cvasi-Newton aproximează hessianul funcției de cost prin matricea calculată cu relația:

$$H = J^T J \tag{5.25}$$

Gradientul se calculează conform relației 5.26.

$$g = J^T e \tag{5.26}$$

unde, J este matricea Jacobiană care conține derivata de ordinul unu a erorilor în raport cu ponderile şi erorile, iar **e** este vectorul erorilor rețelei. Calculul matricii Jacobiene este mult mai simplă decât calculul matricii hessiene.

$$\mathbf{J(x)} = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w^1_{1,1}} & \frac{\partial e_{1,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{1,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,1}}{\partial b^1_1} & \cdots \\ \frac{\partial e_{2,1}}{\partial w^1_{1,1}} & \frac{\partial e_{2,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{2,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{2,1}}{\partial b^1_1} & \cdots \\ \vdots & \vdots & & \vdots & \vdots & \\ \frac{\partial e_{S^M,1}}{\partial w^1_{1,1}} & \frac{\partial e_{S^M,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{e_{S^M,1}}}{\partial w^1_{S^1,R}} & \frac{\partial e_{e_{S^M,1}}}{\partial b^1_1} & \cdots \\ \frac{\partial e_{1,2}}{\partial w^1_{1,1}} & \frac{\partial e_{1,2}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{1,2}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,2}}{\partial b^1_1} & \cdots \\ \vdots & \vdots & & \vdots & \vdots & \end{bmatrix} \tag{5.27}$$

Matricea H calculată conform relației 5.25 poate fi singulară şi în acest caz H^{-1} nu există deci nu se poate calcula modificarea ponderii conform relației:

$$\Delta w = -\eta \frac{\nabla \varepsilon(w_0)}{H^{-1}} \tag{5.28}$$

De aceea metoda Levenberg-Marquardt aproximează hessianul cu matricea:

$$G = H + \mu I = \nabla^2 \varepsilon(w) + \mu I \tag{5.29}$$

în care μ reprezintă un factor de obicei pozitiv, iar I matricea identică. matricea G este simetrică şi nesingulară chiar dacă hessianul este singular.

Astfel Δw devine:

$$\Delta w = -\eta \frac{\nabla \varepsilon(w_0)}{\left[\nabla^2 \varepsilon(w) + \mu I\right]^{-1}} \tag{5.30}$$

iar actualizarea se face conform relaţiei:

$$x_{k+1} = x_k - [J^T J + \mu I]^{-1} J^T e \tag{5.31}$$

Reţeaua poate fi creată şi antrenată folosind Neural Network Toolbox sau cu un cod Maltab, aşa cum s-a prezentat în paragraful 5.4.1.

5.4.2.3.1 Implementarea unui strat al RNA FF BP

Pentru un strat neuronal cu 7 neuroni prezentat în figura 5.5.a sau descris de codul 5.7.a şi acelaşi set de 150 vectori de antrenare, simulat în precizie dublă se obţin 8 vectori de ieşire eronaţi. Acest rezultat nu este surprinzător având în vedere că stratul neuronal nu a atins performanţele de antrenare cerute, aşa cum se poate vedea din figura 5.6.a.

Se poate trece în continuare la deducerea numărului minim (optim) de biţi necesari pentru reprezentarea ponderilor. Simulând din nou reţeaua pentru diverse preciyii de repreyentare a ponderilor se obţine următorul grafic al erorilor în funcţie de numărul de biţi/ponderi:

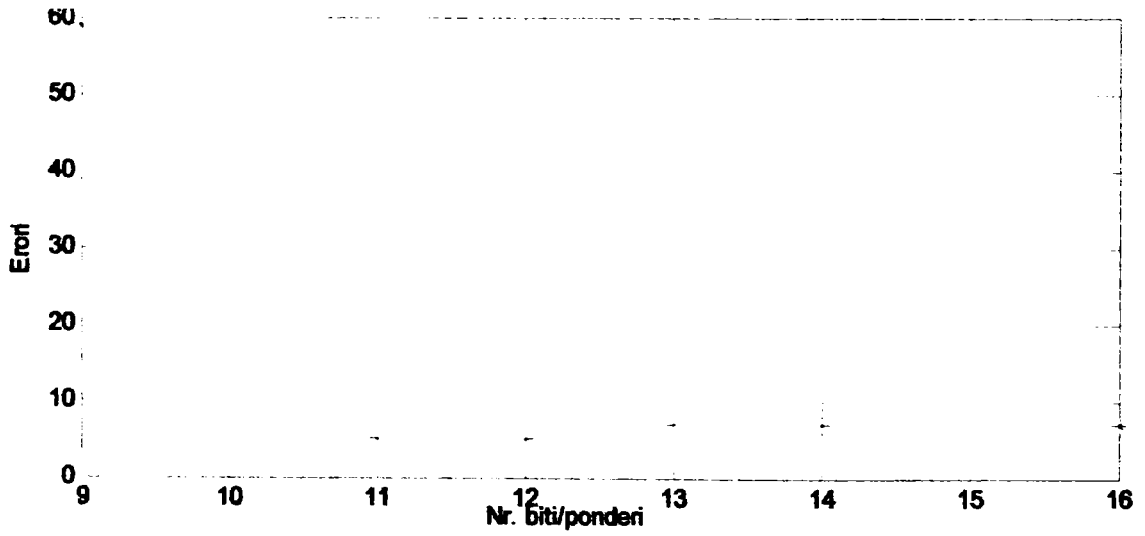


Figura 5.24 Erorile modelului software al RNA FF-BP-LM 1x7

Surprinzător este faptul că pentru 11 respectiv 12 biţi/ponderi se obţin mai puţine erori ale reţelei decât pentru o reprezentare a ponderilor în dublă precizie (5 în loc de 8).

Reţeaua neuronală, implementată folosind modelul creat în System Generator, pentru diverse precizii de reprezentare a ponderilor conduce la graficul erorilor din figura următoare:

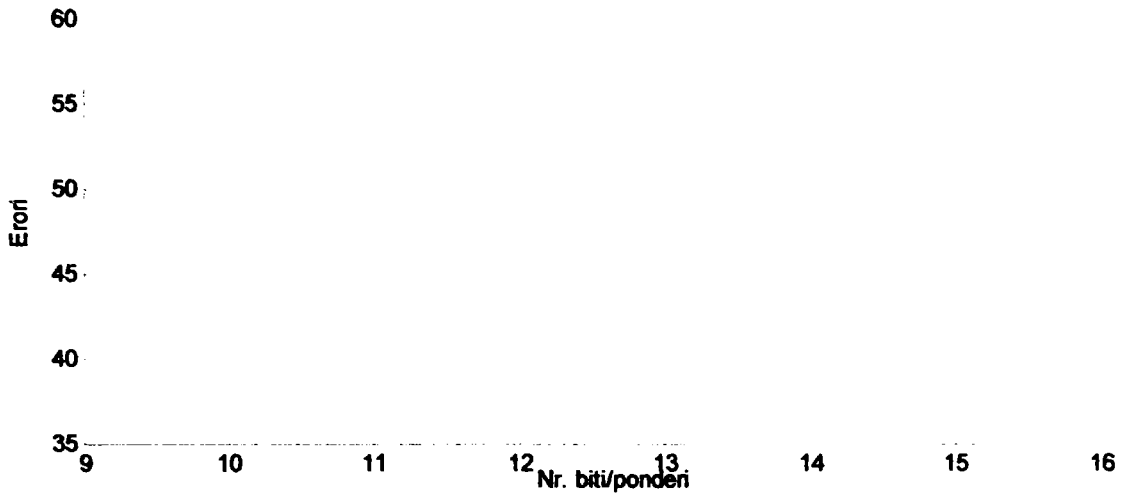


Figura 5.25 Erorile modelului hardware al RNA FF-BP-LM 1x7

Minimul erorilor (37) se obține pentru o precizie de 12 biți a ponderilor. Procentul de asociere corectă a vectorilor în acest caz este de numai 75,33%. Acest procent scăzut se datorează în parte reducerii preciziei de reprezentare a ponderilor, care au fost calculate în dublă precizie, dar şi modului cum se face această reducere. Pentru a obține un procent mai ridicat de asociere corectă se analizează care este modul cel mai potrivit pentru a face această reducere a preciziei. În mod implicit, blocul de memorie ROM folosit pentru stocarea ponderilor face o reducere la numărul de biți specificați folosind cuantizarea prin rotunjire la valoarea cea mai apropiată exprimată cu numărul de biți specificați (funcția Matlab „round"). În cazul depăşirii valorii maxime se aplică saturarea la valoarea maximă.

Pentru a modifica modul de reducere a preciziei de reprezentare a ponderilor, înainte de încărcarea acestora în memorie se poate face o reducere explicită a precizie folosind funcții Matlab. De exemplu:

```
q = quantizer( 'fixed', 'fix', 'saturate', [nb bp]);
pond=round(q, ponderi);
```
(5.32)

permite exprimarea ponderilor în virgulă fixă cu semn. „q" reprezintă parametrii de cuantizare. Cuantizarea se face prin rotunjire la cea mai apropiată valoare de zero exprimată pe numărul de biți specificați de „nb" din care „bp" biți sunt utilizați pentru exprimarea părții fracționare. Depăşirea se tratează prin saturare.
Posibilitățile de rotunjire sunt cele obişnuite din Maltab: round, fix, ceil, floor sau convergent. În cazul depăşirii se poate opta între: wrap şi saturate.

Testând diverse variante pentru reducerea preciziei se obțin rezultatele prezentate în tabelul următor:

Tabelul 5.6 Ponderile utilizate în implementarea hardware a RNA

Cuantizare	**Număr vectori eronați**								
Nr. biți / Metoda	8	9	10	11	12	13	14	15	16
round	109	50	44	60	37	39	41	41	41
fix	123	78	19	22	35	37	38	39	41
ceil	150	150	141	101	77	53	49	48	46
floor	150	147	82	10	23	24	30	38	41
convergent	109	50	44	60	37	39	41	41	41

Numărul minim de erori (10) se obţine pentru rotunjire la cea mai apropiată valoare de minus infinit (floor), pentru o reprezentare pe 11 biţi a ponderilor.

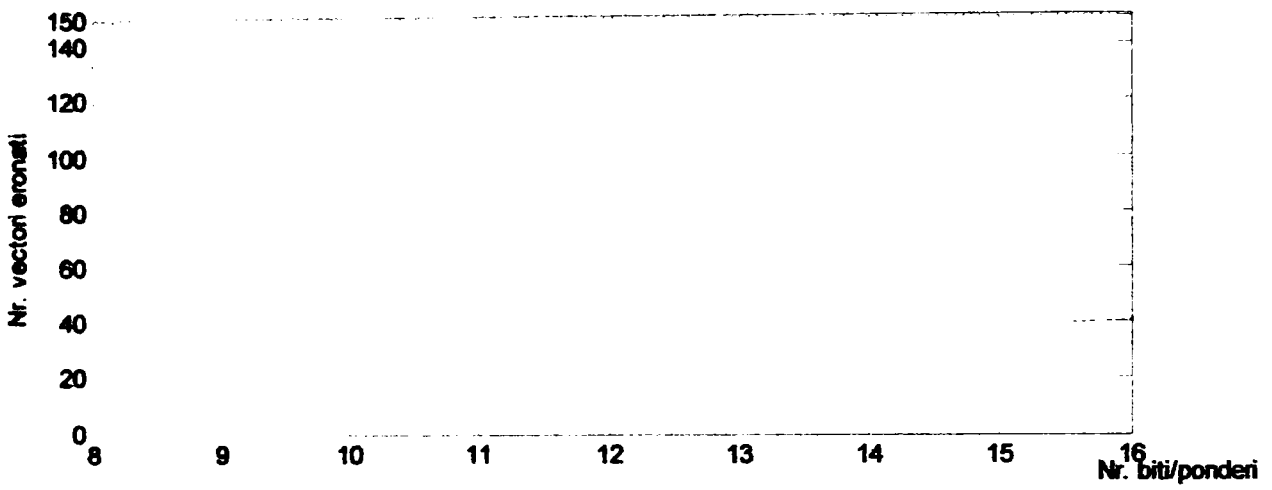
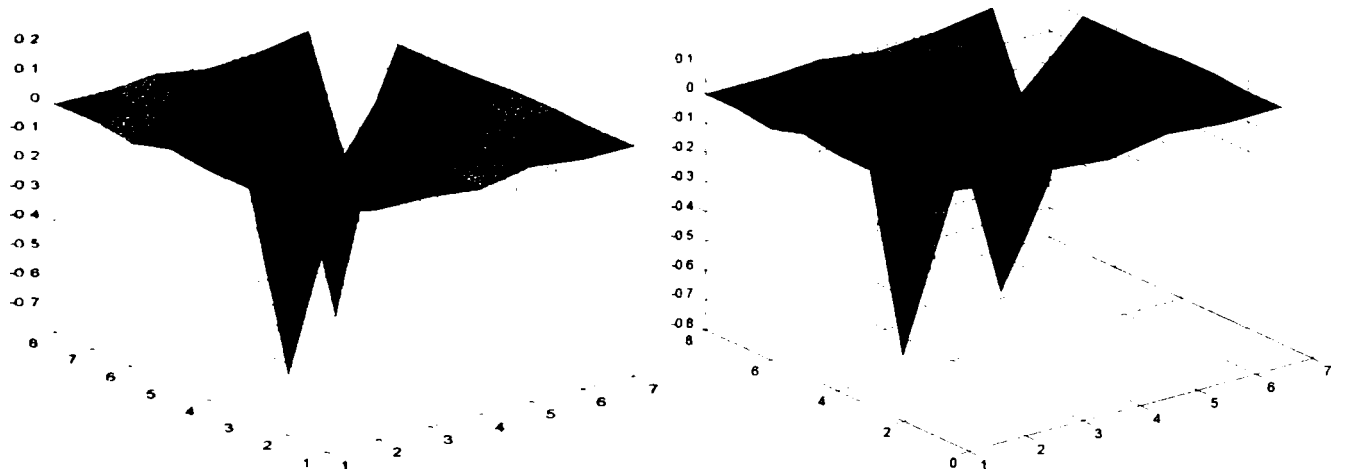


Figura 5.26 Erorile RNA FF-BP-LM 1x7 implementate cu corecţia ponderilor

Ponderile calculate conform relaţiei 5.32, pentru nb=bp=11, prin rotunjire la cea mai apropiată valoare de zero exprimată în precizie fix_11_11 (round), respectiv prin rotunjire la cea mai apropiată valoare de minus infinit exprimată în precizie fix_11_11 (floor) sunt reprezentate în figura următoare.



a) round **b) floor**

Figura 5.27 Ponderile RNA FF-BP-LM 1x7 obţinute prin reducerea preciziei

Precizia reţelei implementate hardware devine astfel 93,33%. Se poate face o analiză a erorilor, comparând erorile modelului software al reţelei pentru precizia de 11 biţi cu erorile modelului hardware al reţelei pentru aceeaşi precizie (figura 5.28).

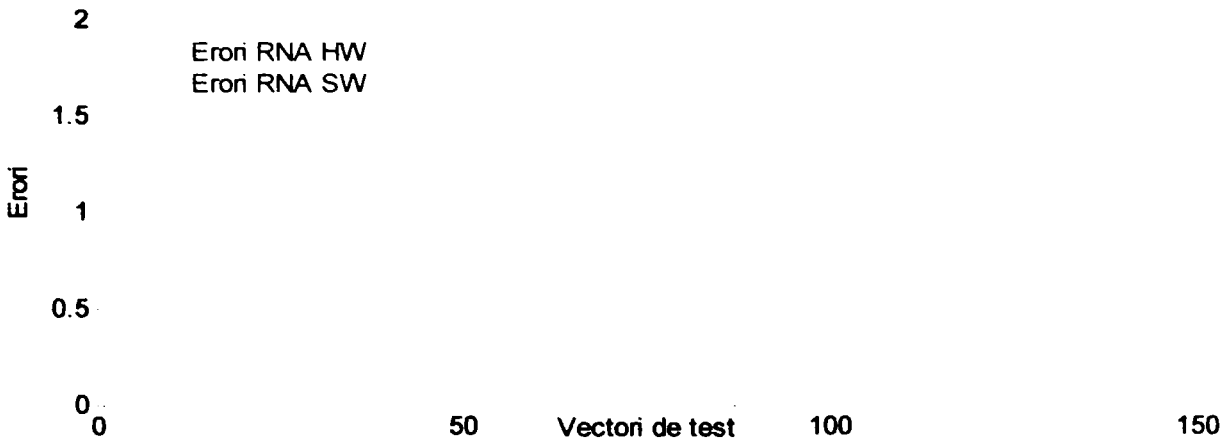


Figura 5.28 Erorile RNA FF-BP-LM 1x7 software respectiv hardware

Figura 5.29 prezintă o analiză a erorilor în funcție de neuronul care răspunde eronat. Se poate observa că neuronul 6 generează cele mai multe erori, 10 din totalul de 12. De altfel tot neuronul 6 este răspunzător şi pentru cele 8 erori ale rețelei implementate software.

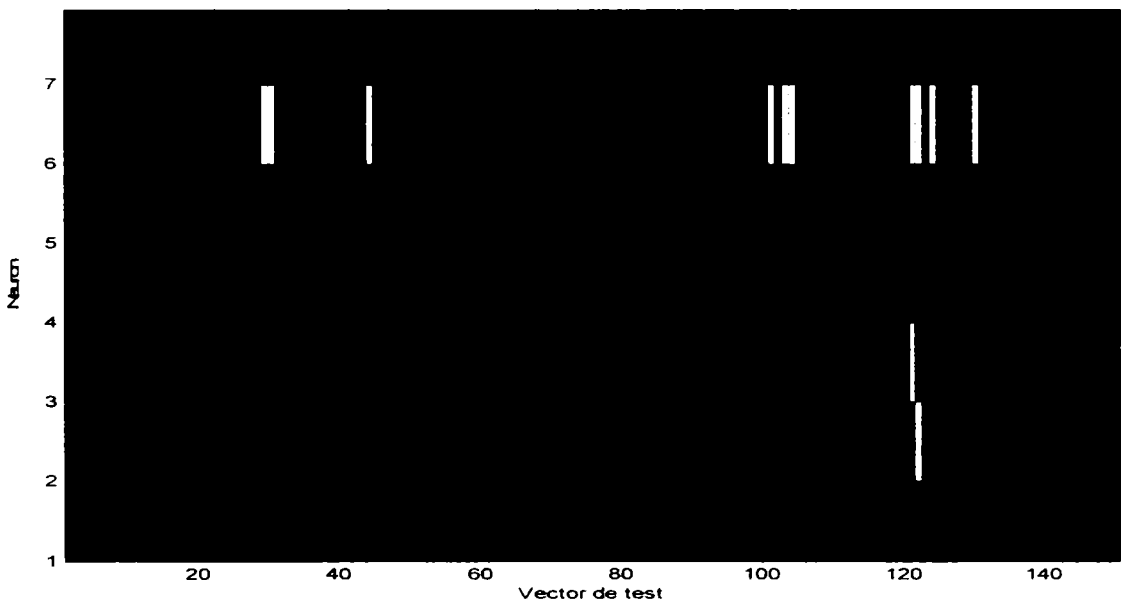


Figura 5.29 Erorile la ieşirea neuronilor pentru o precizie a ponderilor de 11 biți

Toate discuțiile de la paragraful 5.4.2.2. privind optimizarea implementării sunt valabile şi în acest caz. Performanțele rețelei şi resursele necesare pentru implementare sunt identice cu RNA antrenat prin algoritmul Hebbian. De fapt rețelele au aceeaşi structură, diferența dintre ele fiind doar modul de antrenare şi eventual numărul de biți necesari reprezentării ponderilor.

Resursele utilizate de un neuron sunt 14 slice-uri, un multiplicator dedicat şi o memorie de tip Block RAM, iar frecvența maximă de lucru a circuitului în urma optimizării sintezei este de 186, 916 MHz.

Pentru verificarea funcționării corecte a rețelei, aceasta a fost simulată post implementare cu programul ModelSim folosind testbench-ul şi vectorii de test generați de modelul Simulink/System Generator. Frecvența semnalului de tact utilizat este de 100 MHz. În figura următoare se prezintă o porțiune din această simulare.

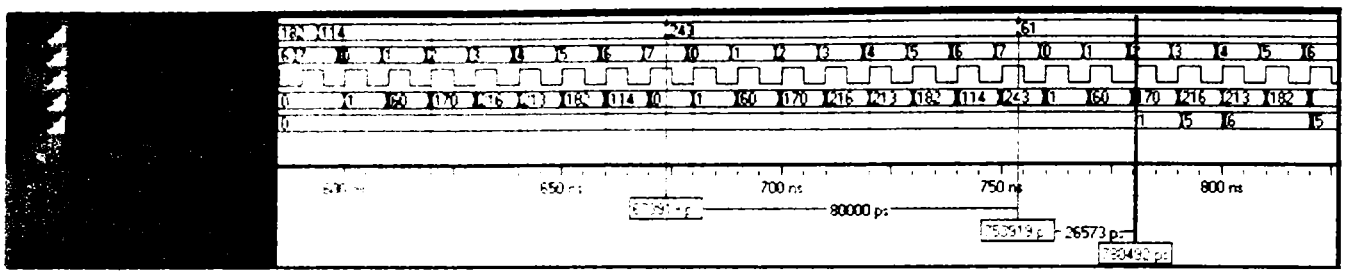


Figura 5.30 Simularea post implementare a RNA FF-BP-LM 1x7

5.4.2.3.2 Implementarea RNA FF BP cu două straturi

În continuare se prezintă implementarea unei rețele neuronale compusă din 2 straturi cu câte 7 neuroni. Rețeaua creată folosind NNToolbox este prezentată în figura 5.5.b, respectiv codul 5.7.b. Analizând ponderile rețelei antrenate se pot trage câteva concluzii importante cu privire la numărul minim de biți necesari pentru reprezentarea acestora. Astfel numărul de biți necesari pentru reprezentarea părții întregi este dat de relația 5.33, dedusă de autor.

$$nbp_{I\_min} = \text{ceil}(\log2(\max(\max(W), \text{abs}(\min(W))))) + 1 \qquad (5.33)$$

În cazul exemplului considerat:

$$\max(W) = 7{,}8996; \quad \min(W) = -10{,}0390 \quad \Rightarrow \quad nbp_{I\_min}=5 \qquad (5.34)$$

Determinarea numărului minim de biți necesari reprezentării părții fracționare se face prin simularea la diverse precizii a RNA folosind un cod Matlab scris în acest scop (Anexa 5). Rezultatele obținute sunt prezentate în figura 5.31. Numărul minim de erori se obține pentru 12 biți, iar pentru mai mult de 14 biți numărul de erorilor nu se mai modifică el fiind identic cu numărul erorilor rețelei software simulate în dublă precizie.

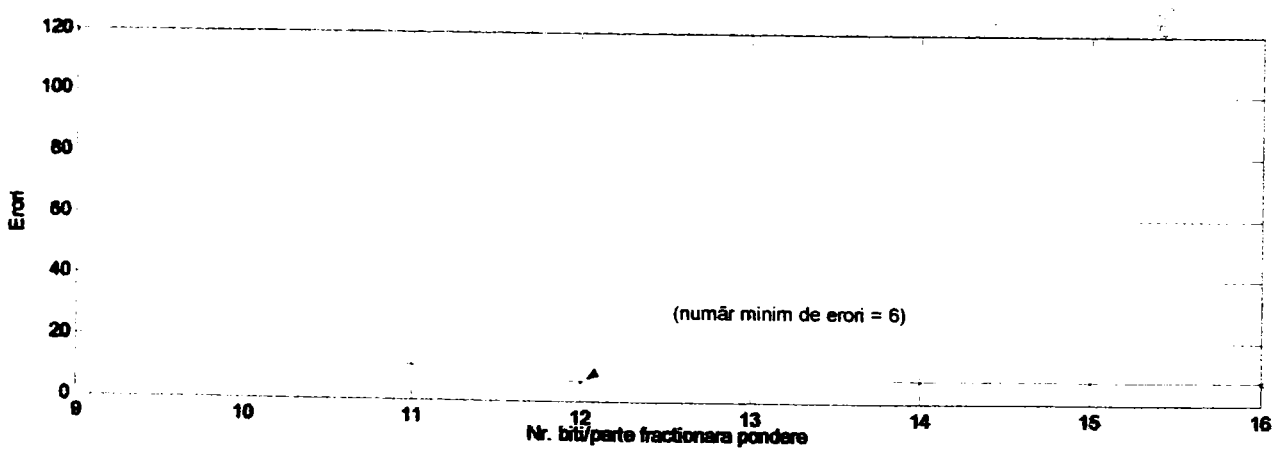
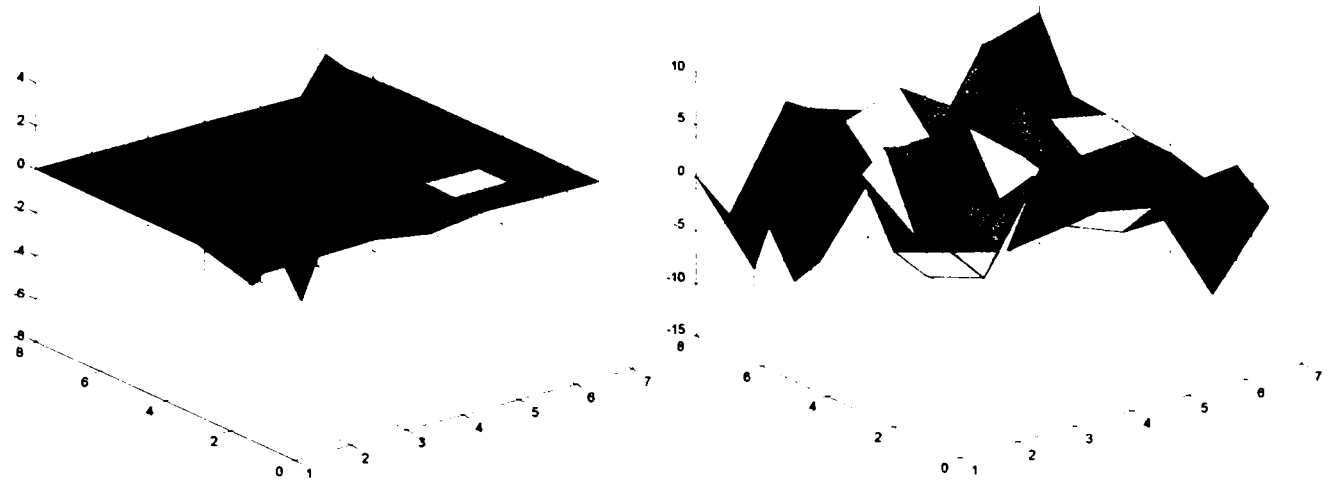


Figura 5.31 Erorile modelului software al RNA FF-BP-LM 2x7

În concluzie ponderile pot fi reprezentate în virgulă fixă folosind un bit de semn, 4 biți pentru partea întreagă şi 12 biți pentru partea fracționară, deci tipul datelor în Simulink este Fix_17_12.

Ponderile şi bias-ul în precizia specificată sunt reprezentate în figura următoare:



a) strat 1 **b) strat 2**

Figura 5.32 Ponderile şi biasul RNA FF-BP-LM 2x7

Modelul rețelei neuronale construit cu blocuri implementabile hardware, create de autor şi blocuri din biblioteca Xilinx, este prezentat în figura următoare.

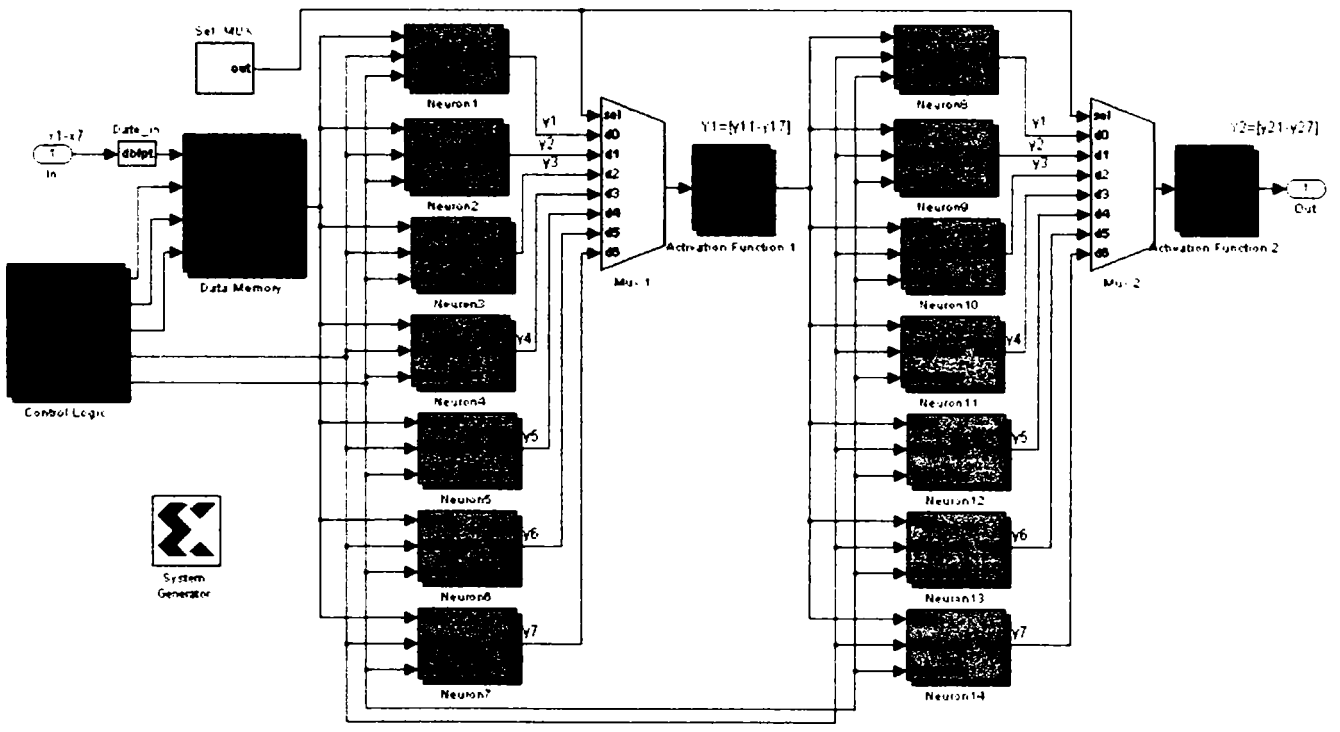


Figura 5.33 Rețea neuronală FF-BP-LM 2x7 cu paralelism de neuron

Simularea rețelei la precizia maximă a multiplicatoarelor şi acumulatoarelor conduce la rezultatele din figura 5.34. Se poate observa o „întârziere" a răspunsului rețelei de M_2+4 eşantioane de la aplicarea vectorului de intrare, unde M_2 reprezintă numărul de intrări ale neuronilor din stratul 2.

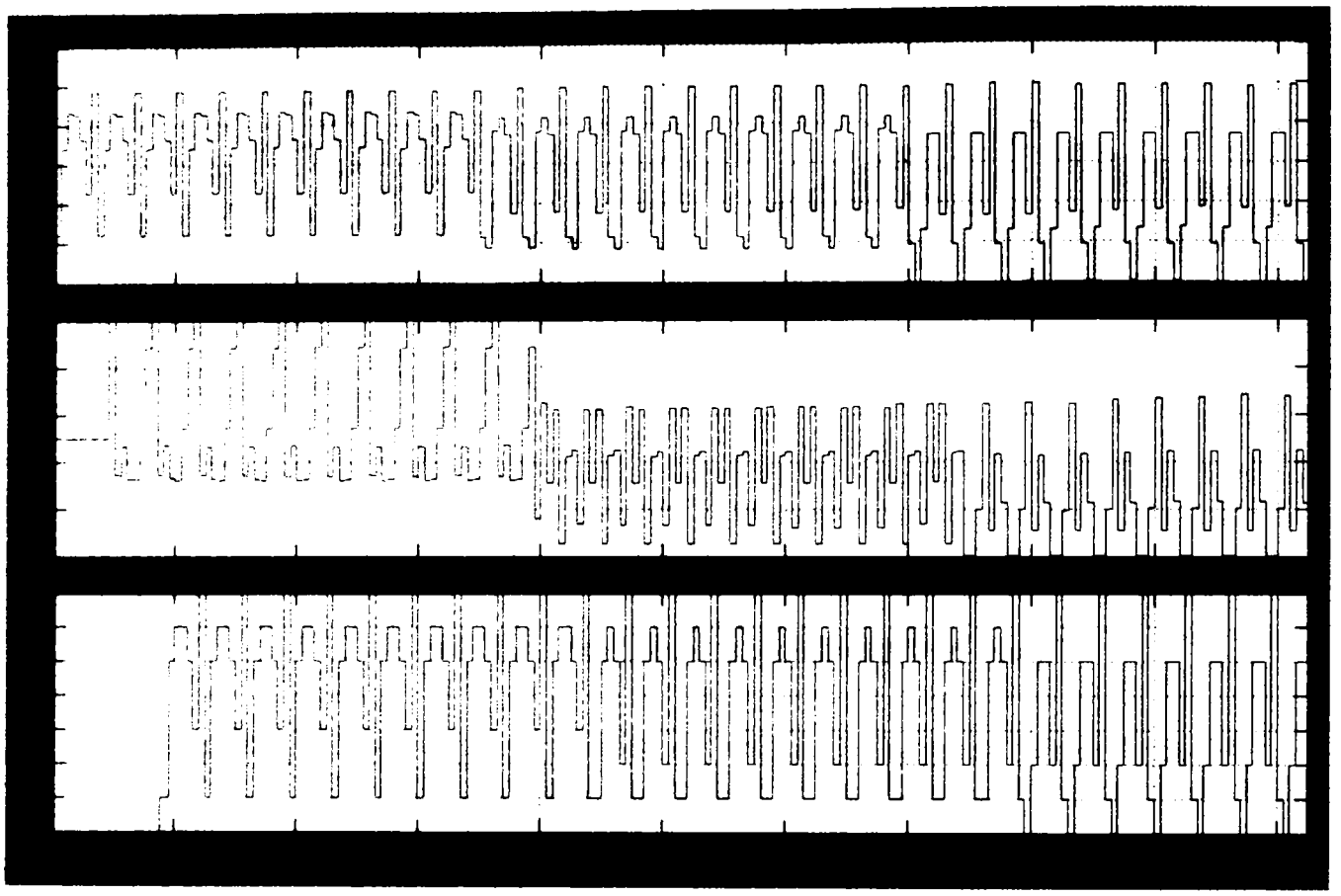


Figura 5.34 Formele de undă pentru RNA FF-BP-LM 2x7 din figura 4.32

Testând diverse variante pentru reducerea preciziei de reprezentare a ponderilor, numărul minim de erori (5) se obţine pentru rotunjire la cea mai apropiată valoare, pentru o reprezentare pe 12 biţi. Astfel procentul de asociere corectă este de 99,52%.

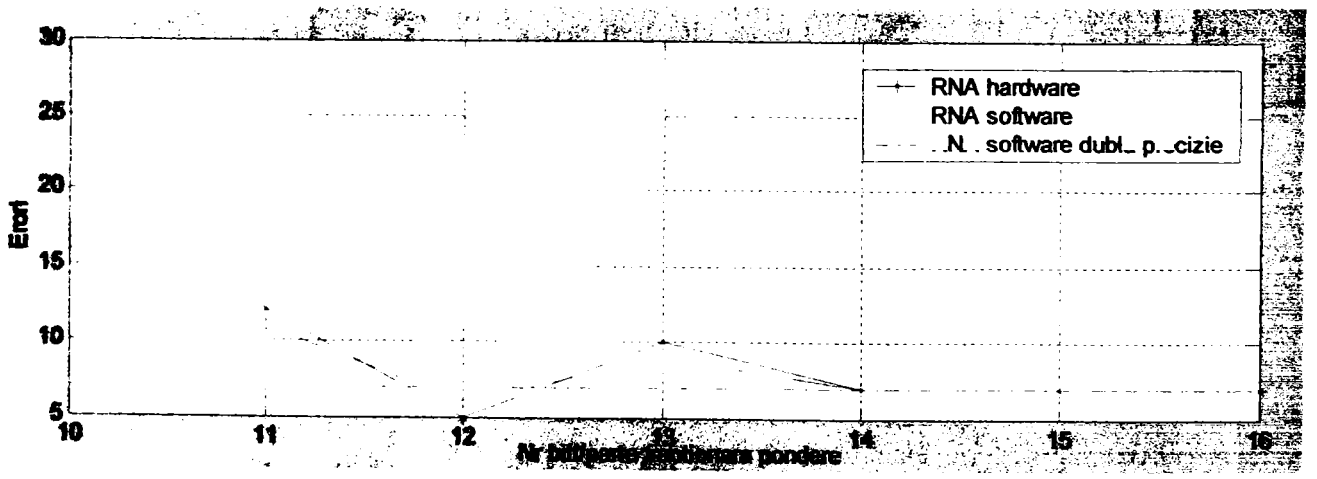


Figura 5.35 Dependenţa erorilor RNA FF-BP-LM 2x7 de numărul de biţi

Numărul erorilor reţelei implementate software şi a celei implementate hardware sunt identice pentru o reprezentare pe mai mult de 14 biţi a părţii fracţionare. Pentru 12 biţi modelul RNA hardware prezintă cele mai puţine erori (5) în timp ce modelul RNA software prezintă 6 erori, faţă de RNA software simulat în dublă precizie care are 7 erori.

Analiza acestor erori în funcţie de vectorii de test este prezentată în figura 5.36.

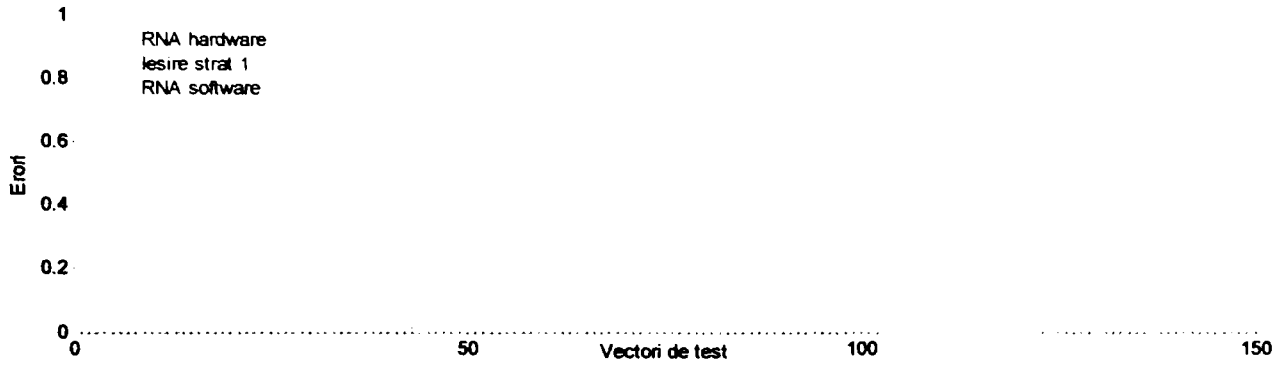


Figura 5.36 Dependența erorile RNA FF-BP-LM 2x7 de vectorii de test

Se constată că din cei 150 de vectori de test, 5 conduc la erori atât în cazul rețelei software simulate în dublă precizie cât şi a celei hardware, în plus rețeaua software mai prezintă două erori pe care modelul hardware reuşeşte să le corecteze.

Erorile celor 7 neuroni din stratul de ieşire RNA pentru cei 150 de vectori de test sunt reprezentate în figura următoare.

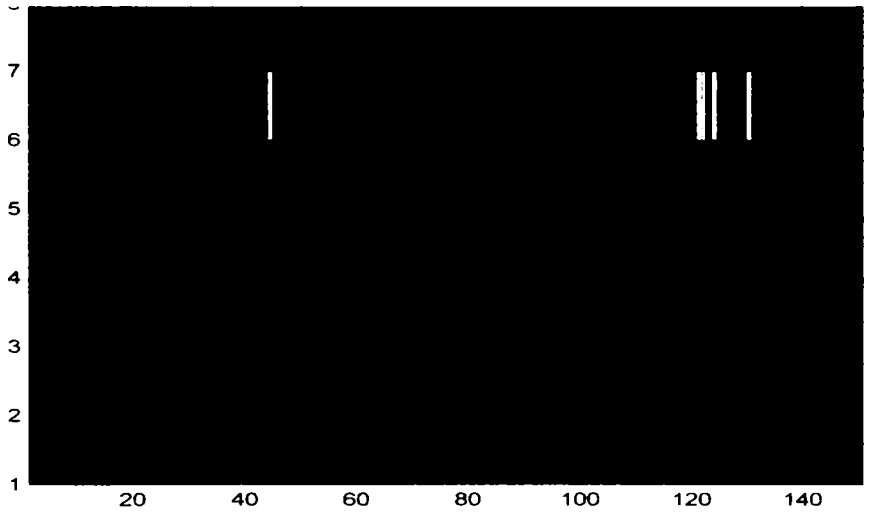


Figura 5.37 Erorile la ieşirea RNA

5.4.2.3.3 Optimizarea implementării RNA FF BP cu două straturi

Implementarea RNA FF BP cu două straturi a câte 7 neuroni, la precizia maximă pentru multiplicatoare şi acumulatoare conduce la următorul raport de implementare:

Device utilization summary:

Number of MULT18X18s	14 out of 40	35%
Number of RAMB16s	18 out of 40	45%
Number of SLICEs	653 out of 5120	12%
Number of BUFGMUXs	1 out of 16	6%

Un neuron utilizează 45 de slice-uri, un multiplicator şi un Block RAM.

Frecvența maximă de lucru a circuitului este de 143,962MHz.

Dacă se reduce numărului de biți pe care se face înmulțirea, acumularea şi respectiv memorarea datelor la ieşirea stratului neuronal, până la limita la care circuitul nu prezintă mai multe erori se pot reduce resursele consumate pentru implementare, aşa cum se poate observa din raportul de implementare următor:

Device utilization summary:

Number of MULT18X18s	14 out of 40	35%
Number of RAMB16s	18 out of 40	45%
Number of SLICEs	554 out of 5120	11%
Number of BUFGMUXs	1 out of 16	6%

Cu toate că reducerea numărului de biţi ai multiplicatorului şi acumulatorului a fost substanţială (de exemplu de la 25 la 12 pentru multiplicatoarele din primul strat), introducerea suplimentară a circuitelor pentru calculul rotunjirilor face ca resursele utilizate să nu scadă în aceeaşi măsură, în timp ce frecvenţa de lucru a circuitului scade la 135,465MHz.

Împărţirea resurselor pe blocuri este următoarea:

	Bloc c-dă strat1	Neuroni strat 1	Funcţie activare strat 1	Bloc c-dă strat 2	Neuroni strat 2
Slice-uri	65	217	9	32	231
Bloc RAM	1	7	2	1	7
Multiplicatoare	0	7	0	0	7

Un neuron din stratul unu utilizează 31 de slice-uri, un multiplicator dedicat şi un Block RAM, în timp ce unul din stratul doi utilizează cu două slice-uri mai mult.

O posibilitate de reducere mult mai substanţială a resurselor utilizate o reprezintă reducerea timpului de latenţă a multiplicatorului de la 3 perioade a semnalului de ceas la o perioadă. Numărul slice-urilor utilizate de un neuron din stratul unu se reduce la 12, iar a unui neuron din stratul doi la 14. Numărul total de slice-uri utilizate de întreaga reţea este de numai 290. Calculul ieşirii reţelei se face în M_2+2 eşantioane de la aplicarea vectorului de intrare.

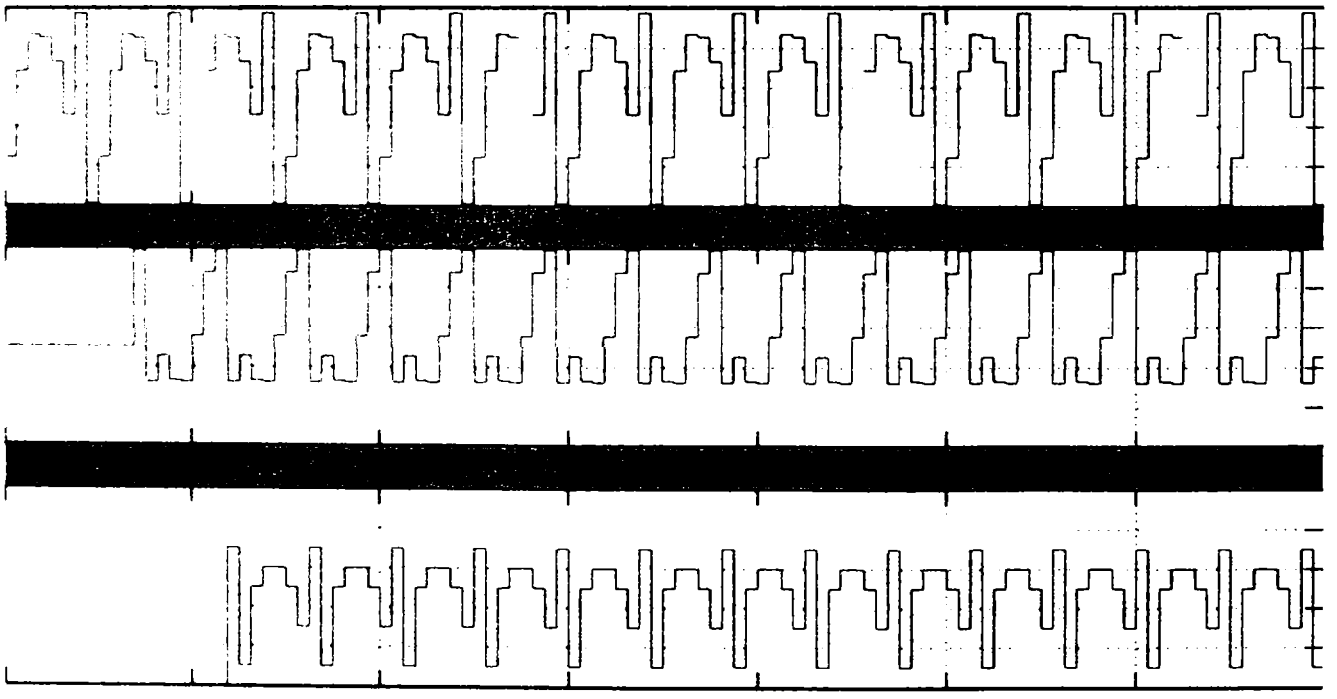


Figura 5.38 Formele de undă pentru RNA FF-BP-LM 2x7 din figura 5.32

Dezavantajul scăderii timpului de latenţă a multiplicatoarelor este reducerea frecvenţei maxime de lucru a circuitului la 109,6 MHz.

Simularea post implementare a reţelei cu un semnal de tact cu perioada de 10 ns, furnizează rezultatele prezentate în figura următoare:

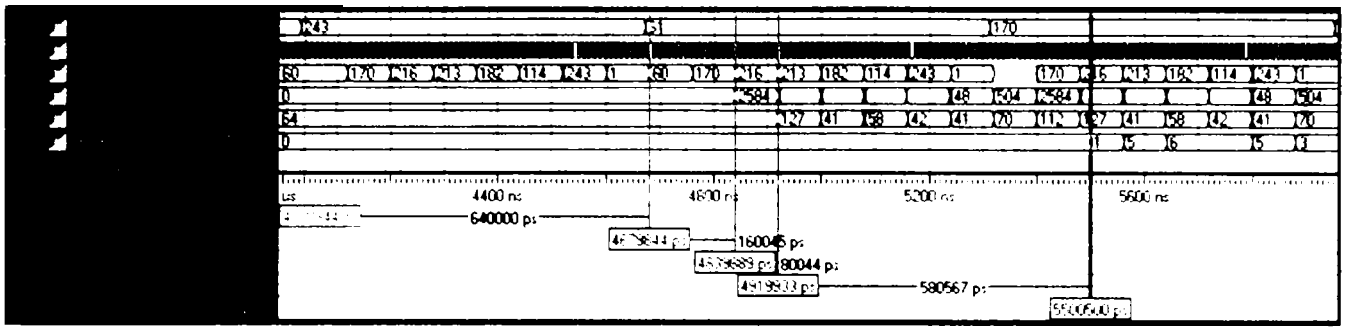


Figura 5.39 Simularea post implementare a RNA FF-BP-LM 2x7

Un element al vectorului de intrare are durata de 8 impulsuri de clock (8x10 ns) deci vectorul de intrare compus din 7 eşantioane, plus bias-ul este furnizat complet blocului de multiplicare acumulare în Ts=640 ns. Primului strat furnizează primul rezultat al ieşirii sale după 240 ns, deci cu o întârziere egală cu aproximativ 3 Ts/(n+1). Încă 560 ns sunt necesare pentru transferul de la ieşirea stratului 1 în stratul 2, după care la 20 ns RNA începe să furnizeze eşantioanele vectorului de ieşire. Întârzierea totală între aplicarea ultimului eşantion al vectorului de intrare şi furnizarea primului eşantion al vectorului de ieşire este de aproximativ 820 ns deci aproximativ 1,28 Ts.

Pornind de la relaţia dintre durata unui element al vectorului de intrare şi perioada semnalului de ceas:

$$T_s=(M_1+1)(M_2+1)T_{clk} \tag{5.35}$$

unde, M_1 şi M_2 reprezintă numărul de conexiuni ale neuronilor din stratul 1 respectiv din stratul 2, frecvenţă maximă a semnalului de intrare poate fi calculată cu relaţia 5.36.

$$f_{max}=\frac{f_{clk}}{(M_1+1)(M_2+1)} \tag{5.36}$$

$$f_{max} = 135{,}465 / 64 = 2{,}17 \text{ MHz}$$

Dacă semnalul de intrare are o frecvenţă mai mare este nevoie de modificarea blocului de multiplicare acumulare astfel încât fiecare operaţie de multiplicare acumulare să se desfăşoare pe un impuls de ceas. Pentru aceasta s-a modificat acumulatorul astfel încât bias-ul să poată fi încărcat înainte de a începe acumularea rezultatelor multiplicării vectorului de intrare cu ponderile. Noua arhitectură a blocului de multiplicare acumulare este prezentată în figura următoare:

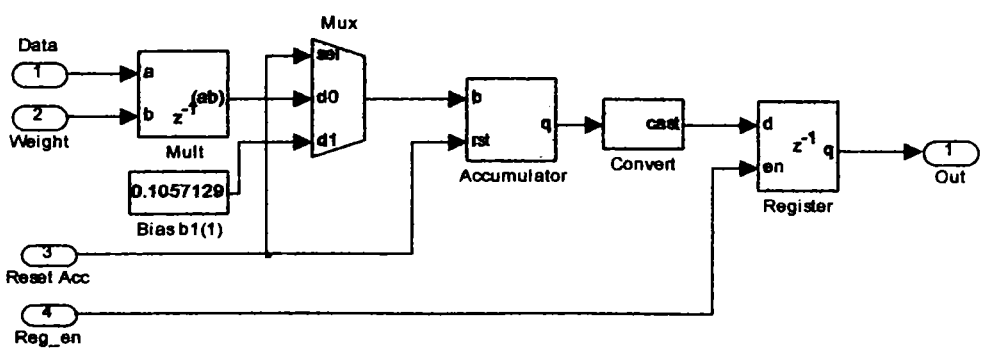


Figura 5.40 Arhitectura modificată a blocului MAC

Resursele necesare pentru un neuron cresc la 18 slice-uri pentru stratul unu respectiv 21 pentru stratul doi, un multiplicator dedicat şi un Block RAM. Resursele totale folosite de RNA sunt prezentate în tabelul următor:

	Bloc c-dă strat1	Neuroni strat 1	Funcţie activare strat 1	Bloc c-dă strat 2	Neuroni strat 2
Slice-uri	39	126	9	20	147
Bloc RAM	1	7	2	1	7
Multiplicatoare	0	7	0	0	7

Rezultatele simulării post implementare a reţelei modificate cu un semnal de tact cu perioada de 10 ns, sunt prezentate în figura următoare:

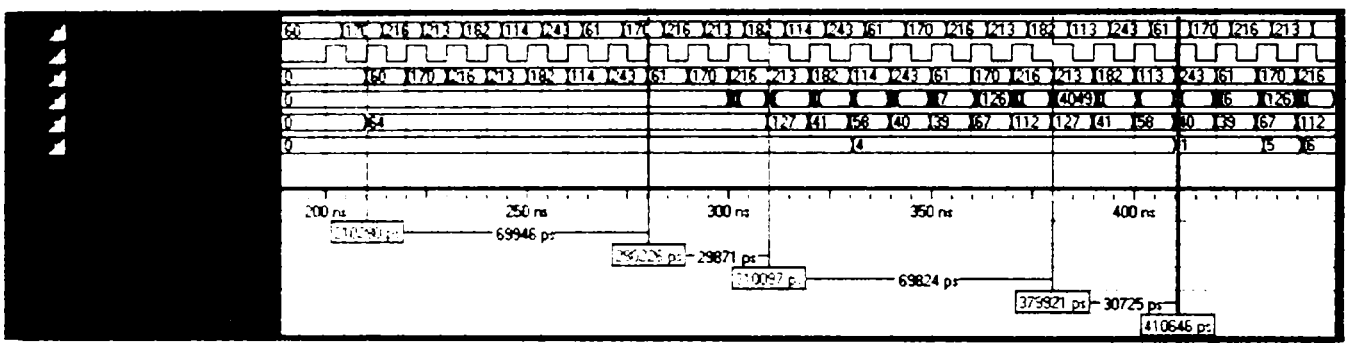


Figura 5.41 Simularea RNA cu acumulator modificat pentru încărcarea bias-ului

Un element al vectorului de intrare are durata egală cu perioada impulsului de ceas (10 ns) deci vectorul de intrare compus din 7 eşantioane, este furnizat complet blocului de multiplicare acumulare în 7xTs=70 ns. Primului strat furnizează primul rezultat al ieşirii sale după 3 impulsuri de tact (30 ns), datorate multiplicatorului, registrului de la ieşirea MAC şi funcţiei de activare. Încă 70 ns sunt necesare pentru transferul de la ieşirea stratului 1 în stratul 2, după care la 30 ns RNA începe să furnizeze eşantioanele vectorului de ieşire. Întârzierea totală între aplicarea ultimului eşantion al vectorului de intrare şi furnizarea primului eşantion al vectorului de ieşire este de aproximativ 200 ns, deci 20 Ts.

Frecvenţă maximă a semnalului de intrare este:

$$f_{max}=f_{clk} \tag{5.37}$$

Frecvenţă maximă a semnalului de ceas care rezultă din raportul de implementare este de 129,7 MHz în consecinţă şi semnalul de intrare poate avea această frecvenţă.

5.5 *Implementarea unei RNA cu paralelism de strat*

O reţea neuronală cu paralelism al straturilor presupune existenţa unui element de procesare pe strat. În paragraful 5.3 s-au prezentat particularităţile unei astfel de reţele, iar figura 5.2 prezintă schema bloc a unei reţele compusă din trei straturi deci cu 3 elemente de procesare.

Pentru verificarea performanţelor unei RNA cu paralelism de strat s-a implementat un strat neuronal compus din N=7 neuroni fiecare având M=7 intrări plus bias-ul, aşa cum se prezintă în figura următoare:

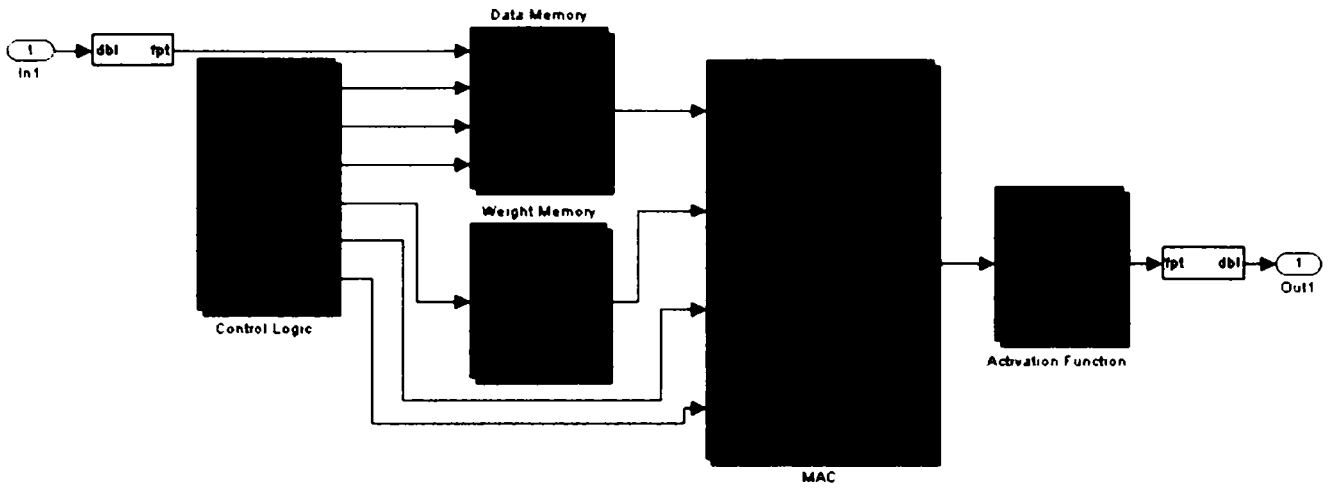


Figura 5.42 Strat neuronal compus din 7 neuroni

Folosind rezultatele obținute în paragraful 5.4.2.1 la implementarea unei rețele antrenate prin algoritmul Hebbian, se utilizează precizia de reprezentare a ponderilor care a dus la cele mai puține erori, și anume 11 biți, toți alocați părții fracționare. Rezultatele simulării post implementare a RNA cu paralelism de strat sunt prezentate în figura 5.43. S-a folosit același set de vectori de test ca la RNA cu paralelism de neuroni compus din 10 x 15 vectori. Prima simulare prezintă formele de undă în cazul în care multiplicatorul este implementat cu un timp de latență T_{l_MAC}=1, iar cea de a doua simulare pentru T_{l_MAC}=3. Un element al vectorului de intrare are durata de 8 x T_{clk}. După aproximativ 2,5 T_s, respectiv 4,5 T_s, de la aplicarea ultimului eșantion al vectorului de intrare la intrarea stratului neuronal, la ieșire este disponibil primul eșantion al vectorului de ieșire.

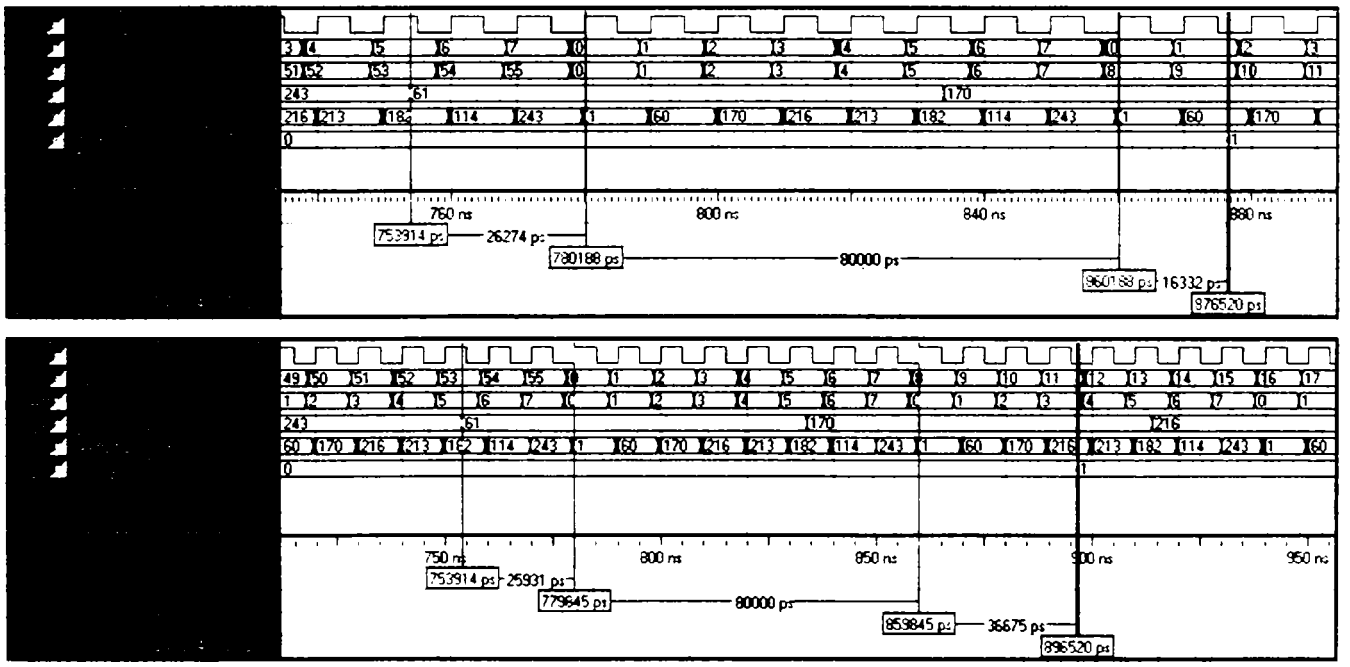


Figura 5.43 Simularea post implementare a RNA cu paralelism de strat

Numărul erorilor rețelei implementat hardware este identic cu cel implementat software, și anume de 15 elemente din totalul de 1050 (98,57%), ceea ce duce la 14 vectori eronați din 150 (93,33%). Analiza erorilor funcție de vectorul de test și neuron este prezentată în figura 5.43.

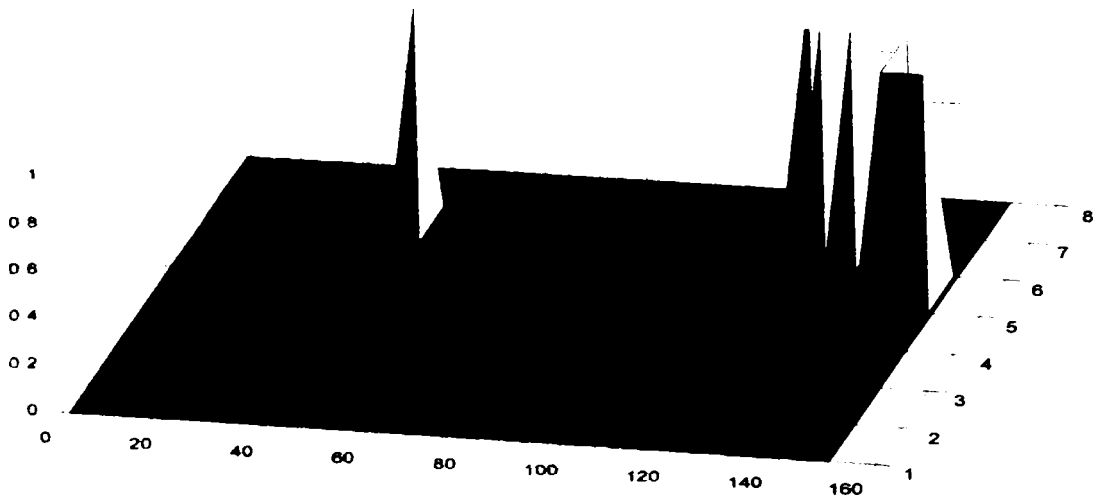


Figura 5.44 Erorile stratului neuronal funcţie de vector şi neuron

Resursele utilizate şi frecvenţă maximă de lucru depind în mare măsură de timpul de latenţă al multiplicatorului. Rezultatele obţinute pentru doi timpi de latenţă diferiţi sunt prezentate în tabelul următor:

	$T_{latenţă\ MAC}=1$	$T_{latenţă\ MAC}=3$
Slice-uri	45	64
Bloc RAM	2	2
Multiplicatoare	1	1
Frecvenţa maximă	140,49	161,89

Diferenţă de 19 slice-uri se datorează în întregime multiplicatorului. Resursele utilizate de blocul de comandă sunt acelaşi în ambele cazuri (33 slice-uri), şi ele sunt incluse în datele prezentate mai sus. Funcţia de activare fiind liniară, practic nu necesită implementarea nici unui circuit hardware. În consecinţă resursele utilizate de un strat neuronal compus din 7 neuroni este de numai 12 (21) slice-uri, 2 memorii block RAM şi un multiplicator dedicat.

Frecvenţa maximă a semnalului de intrare este de Nx(M+1) ori mai mică decât frecvenţa maximă de lucru adică de 2,5 (2,89) MHz.

5.6 Concluzii

Acest capitol a permis testarea metodei dezvoltate în capitolul precedent şi studiul efectelor pe care îl au parametri blocurilor create asupra performanţelor reţelei. Rezultatele obţinute dovedesc corectitudinea metodei, iar studiul efectuat se constituie într-un ajutor important pentru cei care doresc să implementeze reţele neuronale în FPGA. Au fost implementate reţele cu unul sau două straturi cu propagare înainte. Reţelele au fost antrenate prin doua metode diferite, şi anume prin algoritmul Hebbian şi prin metoda Levenberg-Marquardt.

Studiul efectului numărului de biţi utilizaţi pentru reprezentarea ponderilor a demonstrat că un număr de 11-12 biţi este de obicei suficient pentru a obţine aceeaşi precizie ca cea obţinută de modelul software care utilizează ponderi pe 64 biţi. O altă concluzie se

referă la importanța modului cum se face reducerea preciziei ponderilor. Printr-o reducere adecvată se poate obține o precizie mai bună a modelului hardware decât acelui software așa cum s-a arătat în figura 5.35.

Erorile au fost studiate și în funcție de vectorii de test sau de neuronii rețelei. Un spațiu important s-a acordat influenței parametrilor blocurilor componente asupra resurselor utilizate și a frecvenței maxime de lucru. Cea mai mare influență asupra resurselor utilizate o au timpul de latență al multiplicatorului și modul cum se face cuantizarea respectiv cum se tratează depășirea. S-a relevat că cele mai avantajoase opțiuni sunt timp de latență 1, cuantizare prin trunchiere și wrap în cazul depășirii. Astfel, resursele necesare pentru un singur MAC au putut fi reduse de la aproximativ 60 de slice-uri până la 12 slice-uri.

Pentru evaluarea performanțelor unei rețele neuronale sunt importante frecvența de lucru. De aceea au fost deduse și verificate relațiile care permit determinarea frecvenței maxime de lucru.

În faza de propagare performanțele rețelei se măsoară în număr de conexiuni pe secundă. S-a determinat că un circuit de tipul celui folosit (Virtex-II XC2V1000) permite implementarea a câteva sute neuroni deci circuitul poate atinge câteva zeci de GCPS. Un circuit de capacitate mai mare și cu frecvență de lucru mai ridicată cum ar fi cele din familiile Vurtex-II Pro sau Virtex-4 permit implementarea a câtorva mii de neuroni și pot realiza câteva TCPS.

Verificarea experimentală, cu ajutorul platformei hardware, a modelele concepute arată o funcționare identică cu rezultatele obținute în urma simulării modelelor.

Contribuțiile personală a autorului aduse în acest capitol sunt următoarele:

- Modelarea unei rețele cu propagare înainte cu paralelism de neuron antrenată cu algoritmul Hebbian supervizat.
- Elaborarea unui script Matlab care permite evaluarea influenței numărul de biți folosiți pentru reprezentarea ponderilor asupra erorilor RNA implementate software.
- Studiul influenței numărul de biți/ponderi asupra erorilor modelului hardware a RNA. Elaborarea unui script Matlab pentru reprezentarea acestei dependențe.
- Reprezentarea erorilor RNA în funcție de vectorii de test și de neuron.
- Optimizarea RNA implementat din punct de vedere al resurselor HW necesare implementării respectiv pentru creșterea frecvenței maxime de lucru.
- Determinarea relației analitice de calcul a numărului minim de biți pentru reprezentarea părții întregi a ieșirii multiplicatorului din componența MAC.
- Determinarea relației analitice de calcul a numărului de biți ai acumulatorului, pentru reprezentarea părții întregi
- Implementarea unor rețele cu propagarea înapoi a erorilor cu paralelism de neuron având unul sau două straturi.
- Elaborarea unui script Matlab pentru analiza influenței metodei de reducere a ponderilor asupra erorilor.
- Implementarea unei rețele neuronale artificiale BP cu paralelism al straturilor.
- Elaborarea unui script Matlab pentru reprezentarea erorilor funcție de vector și neuron.
- Folosirea unui singur bloc al funcției de activare la ieșirea unui strat neuronal cu paralelism de neuron.
- Determinarea frecvenței maxime a semnalului de intrare în funcție de frecvența maximă a circuitului și a parametrilor rețelei.
- Verificarea experimentală tuturor circuitelor modelate.

Capitolul VI

Implementări de RNA competitive simple

Reţelele cu autoorganizare şi învăţare de tip competitiv cunoscute şi sub denumirea de hărţi de trăsături cu autoorganizare (SOFM, Self-Organazing Feature Map) reprezintă unele din cele mai interesante tipuri de RNA. Ele sunt inspirate din modul de organizare a informaţiei în creierul uman. Cortexul cerebral este subdivizat în zone cum ar fi: cortexul motor, cortexul somatosenzorial, cortexul vizual şi cortexul auditiv [208].

Sarcina pe care o execută reţelele competitive este de clasificare a modelele de intrare. Modelele similare sunt clasificate în aceeaşi clasă, reprezentată de aceeaşi unitate de ieşire. Fiecare neuron va deveni specializat în recunoaşterea unei anumite trăsături a datelor de intrare

6.1 Structura

Reţelele neuronale cu autoorganizare sunt caracterizate de faptul că ele învaţă nesupervizat să descopere trăsături, regularităţi şi corelaţii ale datelor de intrare. Neuronii reţelelor competitive simple sunt aranjaţi într-un strat de ieşire unidimensional, total conectaţi la neuronii stratului de intrare prin intermediul unor ponderi excitatorii. Ei sunt în permanentă competiţie, la un moment dat doar unul singur fiind activat.

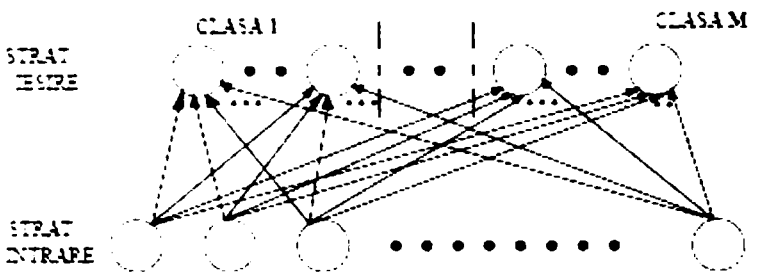


Figura 6.1 RNA cu învăţare competitivă simplă

Fiecare neuron are atâtea conexiuni de intrare câte atribute sunt utilizate în clasificare. Pornind de la un set de ponderi iniţiale, generate aleator, procedura de antrenare constă în găsirea neuronului cu ponderile cele mai apropiate de vectorul de intrare şi declararea lui ca neuron câştigător. Există două metode pentru a determina similaritatea. Prima dintre ele determină intrarea netă a fiecărui neuron conform relaţiei:

$$net_k = \sum_i^N w_{ki} x_i \tag{6.1}$$

Neuronul pentru care intrarea netă are valoarea cea mai mare este declarat câştigător.

A doua metodă constă în calculul distanţei dintre vectorul de intrare şi vectorul ponderilor, cea mai utilizată fiind distanţa Euclidiană, ca în relaţia următoare [207]:

$$net_k = \sqrt{\sum_i^N [x_i - w_{ki}]^2} \tag{6.2}$$

Va câştiga competiţia neuronul pentru care relaţia 6.2 are valoarea cea mai mică.

Procesul de învăţare constă în modificarea ponderilor conform relaţiei:

$$\Delta w_{ji} = \eta(x_j - w_{ji}) \tag{6.3}$$

pentru neuronul j care a câştigat competiţia, şi

$$\Delta w_{ki} = 0, \tag{6.4}$$

pentru k ≠ j. η reprezintă rata de învăţare.

Prin aceasta vectorul ponderilor neuronului câştigător j, se apropie şi mai mult de tiparul prezentat la intrare.

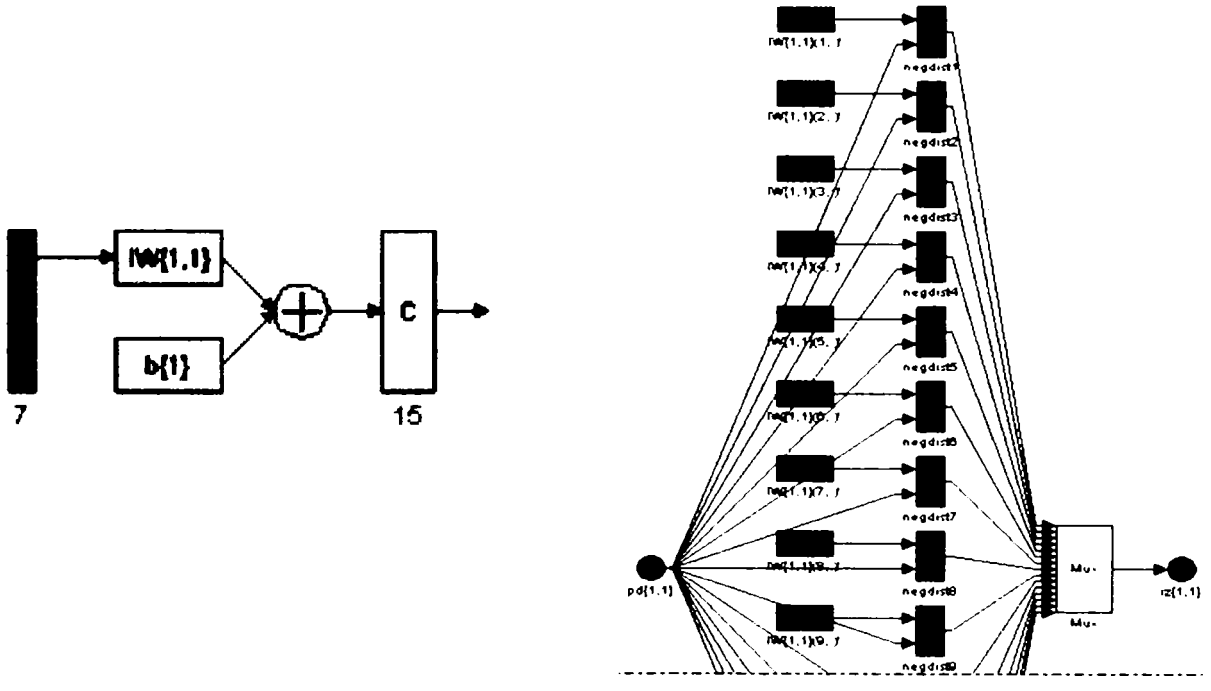
După încheierea antrenării, procesul de clasificare constă în calculul distanţei dintre vectorul de intrare şi fiecare neuron şi declararea intrării ca aparţinând clasei reprezentate de neuronul învingător.

6.2 *Implementarea unei RNA competitive simple*

După cum s-a prezentat în capitolele anterioare, s-a optat pentru implementarea software a fazei de antrenare a RNA, respectiv implementarea hardware a fazei de propagare. În continuare se prezintă modul de antrenare respectiv implementarea hardware a fazei de propagare.

6.2.1 Antrenarea RNA competitive simple

Pentru antrenarea RNA competitivă se creează o RNA simplă cu ajutorul unui cod Matlab sau folosind interfaţa grafică Network/Data Manager. În figura următoare se prezintă o reţea competitivă având 7 neuroni în stratul de intrare şi 15 neuroni în stratul de ieşire.



a) Arhitectura RNA **b) Structura blocului IW{1,1}**

Figura 6.2 RNA competitivă creată cu NNToolbox

Blocul IW{1,1} prezentat în figura 6.2.b calculează negativul distanţei euclidiene dintre vectorul de intrare aplicat şi vectorii formaţi din liniile matricei ponderilor. Dacă toate bias-urile sunt zero intrarea netă maximă a neuronului poate fi zero, când vectorul de intrare este identic cu vectorul ponderilor neuronului. Funcţia de transfer competitivă primeşte intrarea netă a neuronului şi furnizează valoarea zero pentru toţi neuronii cu excepţia celui câştigător, adică acela care are intrarea netă cea mai pozitivă, pentru care furnizează valoarea unu.

După antrenarea reţelei cu un set de 10x15 de vectori de test (prezentat în Anexa 6) şi simularea ei se obţin rezultatele din figura următoare.

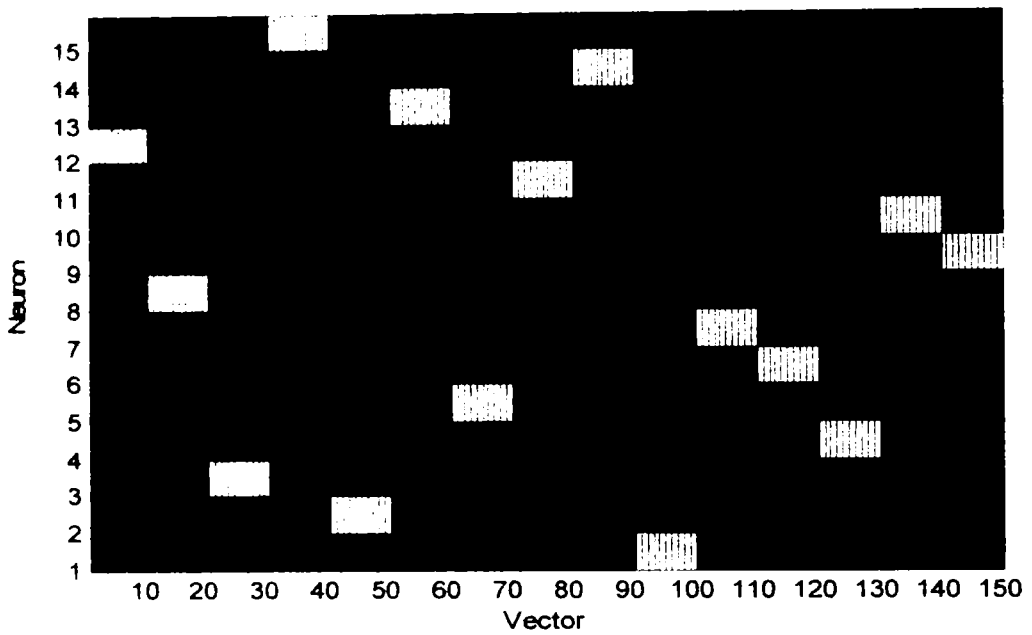


Figura 6.3 Rezultatele simulării RNA competitive

Se poate observa că la un moment dat doar un singur neuron este activ şi că cele 15 seturi de câte 10 vectori sunt împărţite în 15 clase diferite. Astfel primul set de 10 vectori face parte din clasa 12, al doilea set din clasa 8 şi aşa mai departe.

Rezultatele simulării fiind cele dorite (clasificarea în 15 clase diferite ale celor 15 seturi de vectori) ponderile reţelei sunt salvate.

6.2.2 Implementarea hardware a RNA competitive

Pentru implementarea unei RNA competitive trebuie creat un model similar celui din figura 6.2 folosind blocuri Xilinx. Calculul distanţei Euclidiene conform relaţiei 6.2 este dificilă datorită radicalului care este greu de implementat hardware. Deoarece comparaţia între neuroni poate fi făcută şi între pătratele distanţelor dintre vectorii de intrare şi ponderile neuronilor, implementarea calculului radicalului nu este necesară şi relaţia 6.2 poate fi înlocuită de:

$$y_k = \sum_{i}^{N} \left[x_i - w_{ki} \right]^2 \tag{6.5}$$

Pentru aceasta se modifică structura neuronului prezentat în capitolul anterior astfel încât aceasta va conţine o memorie pentru ponderi, un bloc de calcul al diferenţelor x_i-w_{ki}, un bloc MAC pentru calculul sumei pătratelor acestor diferenţe şi blocul funcţiei de activare competitive, aşa cum se prezintă în figura 6.4. Blocul funcţiei de activare este comun tuturor neuronilor stratului de ieşire.

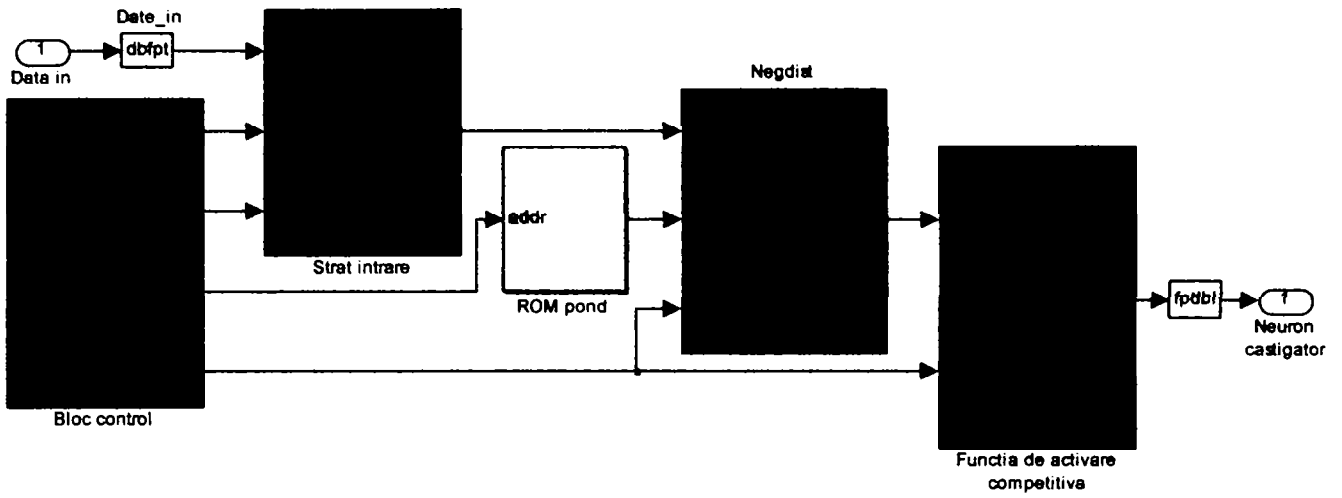


Figura 6.4 Modelul hardware al RNA competitive

Blocurile noi create şi adăugate bibliotecii RNA sunt blocul Negdist de calcul al expresiei din relaţia 6.5 şi blocul funcţiei de activare competitive.

Blocul Negdist se compune dintr-un bloc AddSub din biblioteca Xilinx Blockset şi un bloc MAC creat de autor. Blocul Addsub este folosit pentru calculul diferenţelor x_i-w_{ki}, iar blocul MAC determină suma pătratelor acestor diferenţe.

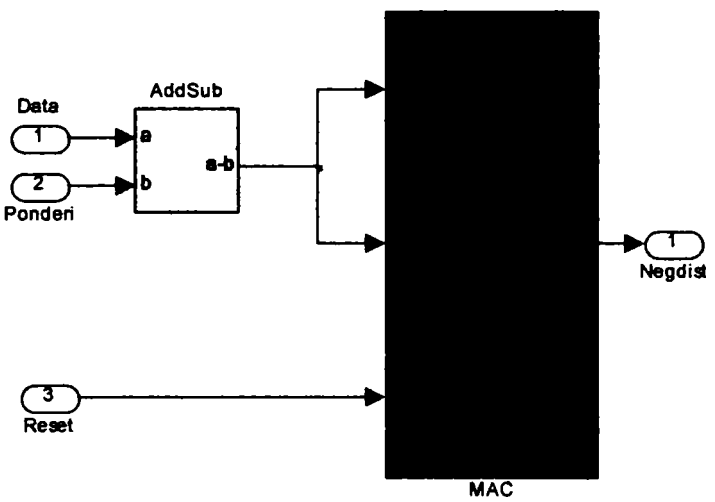


Figura 6.5 Structura blocului Negdist

Blocul funcţiei de activare, prezentat în figura următoare, se compune din două părţi. O primă parte determină minimul dintre ieşirile de tipul y_k a neuronilor ce compun stratul de ieşire, iar cea de a doua parte determină poziţia neuronului cu ieşirea minimă.

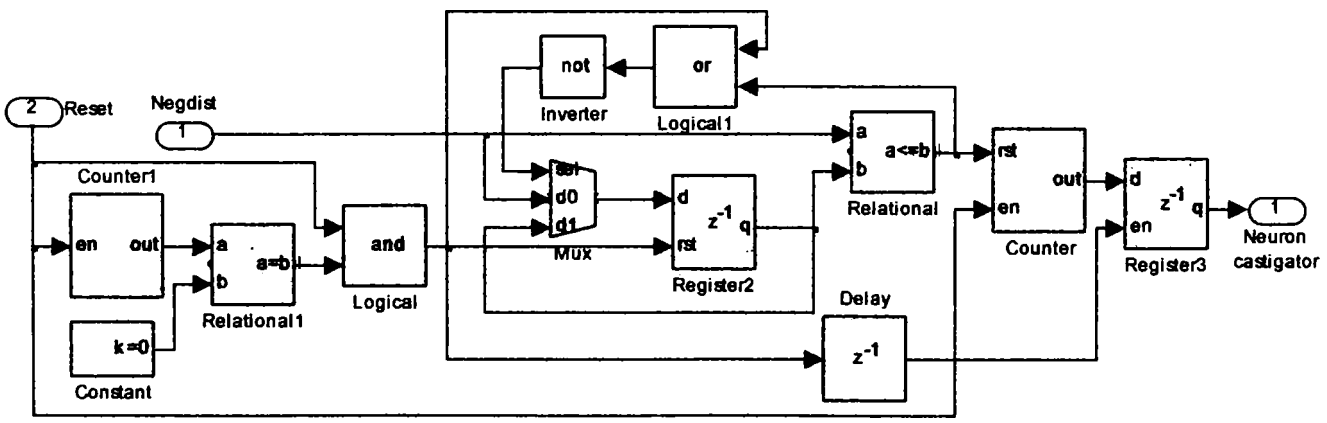
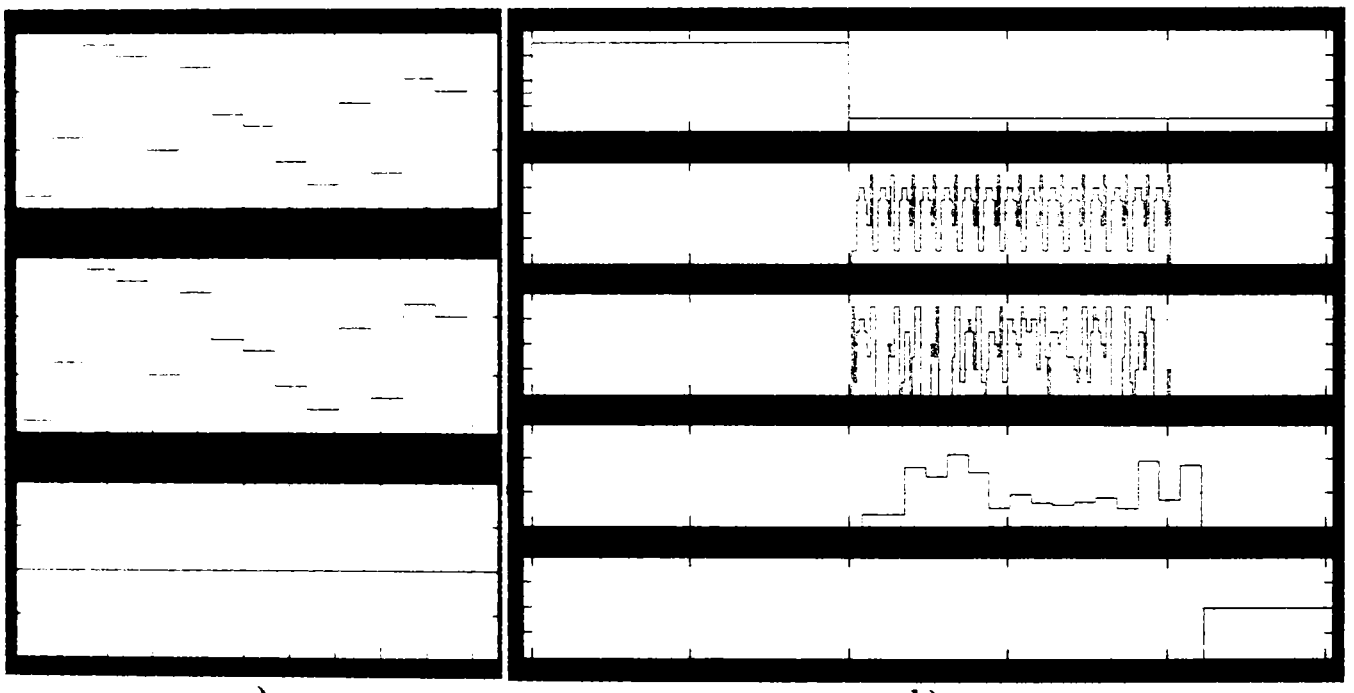


Figura 6.6 Blocul funcţiei de activare competitive

6.2.2.1 Implementarea RNA competitive cu paralelism de strat

Pentru implementarea unei RNA competitive cu paralelism de strat este nevoie de un singur element de procesare pe strat. Pentru verificarea performanţelor unei RNA competitive cu paralelism de strat s-a implementat o reţea având stratul de intrare compus din n_1=7 neuroni şi stratul de ieşire n_2=15 neuroni. Modelul hardware are aceeaşi structură ca şi modelul neuronului prezentat în figura 6.4. RNA competitivă cu paralelism al straturilor este compusă dintr-un bloc de control, un strat de intrare realizat cu o memorie de tip Block RAM, o memorie de ponderi comună tuturor neuronilor, blocul de calcul al distanţei Euclidiene simplificate, şi blocul funcţiei de activare competitive, de asemenea comun tuturor neuronilor. Arhitectura este aceeaşi indiferent de numărul neuronilor din stratul de intrare, respectiv de ieşire. Diferă doar parametrii blocurilor, capacitatea memoriilor şi viteza de calcul.

Rezultatele simulării reţelei sunt prezentate în figura următoare. Primul grafic din figura 6.7.a prezintă clasificarea în cele 15 clase a celor 15 x 10 vectori de intrare, realizată de implementarea software a reţelei. Cel de-al doilea grafic prezintă clasificarea realizată de modelul hardware, în timp ce ultimul prezintă erorile acestuia fată de modelul software. În figura 6.7.b se prezintă un detaliu al simulării modelului hardware.



a) b)

Figura 6.7 Simularea modelului software şi hardware a RNA competitive

După aplicarea unui vector la intrarea stratului de intrare acesta este aplicat stratului de ieşire la o frecvenţă de $(n_1+1)n_2$ ori mai mare deoarece blocul negdist trebuie să efectueze n_1+1 calcule pentru fiecare dintre cei n_2 neuroni.

După cum se poate observa nu există nici o diferenţă între clasificarea realizată de modelul hardware faţă de cea realizată de implementarea software. Simularea modelului hardware folosind alte două seturi de câte 15 x 10 vectori de test duce la aceleaşi rezultate, fără erori. Rezultatele acestor simulări sunt prezentate în Anexa 7.

Resursele consumate de RNA implementată într-un circuit Virtex II XC2V1000 sunt prezentate în tabelul următor:

Tabelul 6.1 Resursele utilizate de RNA competitivă

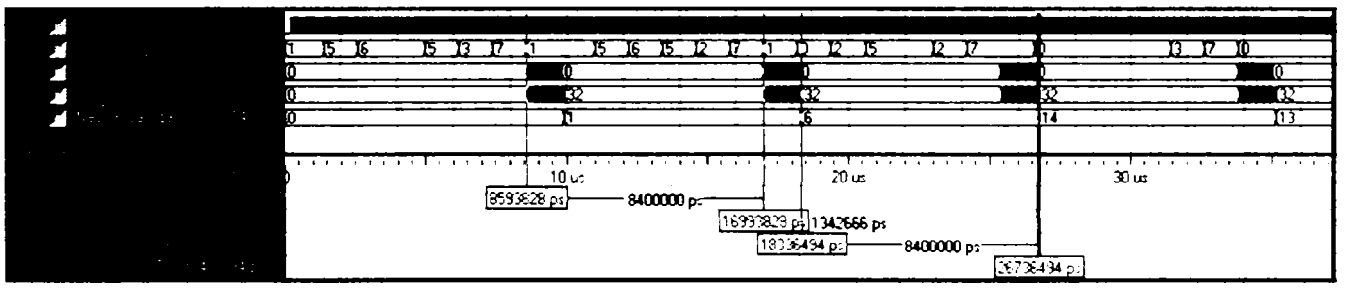
	Bloc c-dă	Strat intrare	Memorie ponderi	Bloc calcul distanţă	Funcţie activare	TOTAL
Slice-uri	16	3	0	15	34	68
Bistabile	15	4	3	25	27	74
Bloc RAM	0	1	1	0	0	2
Tabele de mem.	24	0	0	12	40	76
Multiplicatoare	0	0	0	1	0	1

Rezultatele simulării post implementare, prezentate în figura următoare, confirmă funcţionarea corectă a circuitului implementat. Elementele vectorilor de intrare sunt aplicate stratului de intrare în mod succesiv cu frecvenţa $1/T_s$. unde T_s este durata unui element al vectorului de intrare egală cu:

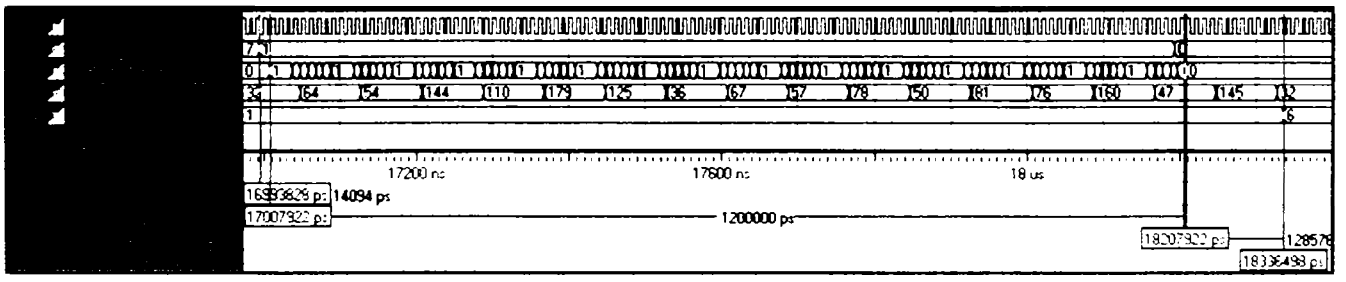
$$T_s = (n_1+1)n_2T_{clk} = 1200 \text{ ns} \tag{6.6}$$

pentru un semnal de ceas cu frecvenţa de 100 MHz . Perioada vectorului de intrare este de 8400 ns. Primul vector de test din figură ([1 5 6 6 5 3 7]) este atribuit primei clase, neuronul câştigător la aplicarea celui de al doilea vector ([1 1 5 6 5 2 7]) este neuronul 6, ş.a.m.d.

Stratul de intrare transmite succesiv aceste elemente stratului de ieşire cu frecvenţă $(n_1+1)n_2/Ts$ pentru ca acesta să efectueze cele $(n_1+1)n_2$ calcule. Neuronul care câştigă competiţia este determinat după 128 ns.



a) Detaliu asupra primilor 4 vectori de test



b) Detaliu asupra timpilor de calcul pentru al doilea vectorul de test

Figura 6.8 Simularea post implementare a RNA competitive cu paralelism de strat

Frecvenţa maximă a semnalului de ceas din raportul de sinteză este de 136 MHz. Frecvenţa maximă cu care se pot aplica elementele vectorului de intrare este de:

$$F_{s\,max} = F_{clk\,max}/(n_1+1)n_2 = 1{,}133 \text{ MHz} \tag{6.7}$$

6.2.2.2 Implementarea RNA competitive cu paralelism de neuron

RNA competitivă cu paralelism al neuronilor este compusă dintr-un bloc de control, un strat de intrare şi un strat de ieşire. Stratul de ieşire este alcătuit dintr-un număr de neuroni egal cu numărul claselor în care dorim să se facă clasificarea şi un bloc de calcul al funcţiei de activare, comun tuturor neuronilor. Neuronul este alcătuit dintr-o memorie de ponderi specifică fiecărui neuron şi un bloc de calcul al distanţei Euclidiene simplificate. Figura următoare prezintă RNA competitivă implementată cu paralelism al neuronilor, având aceeaşi parametrii ca şi RNA implementată cu paralelism al straturilor (7 neuroni în stratul de intrare şi 15 neuroni în stratul de ieşire).

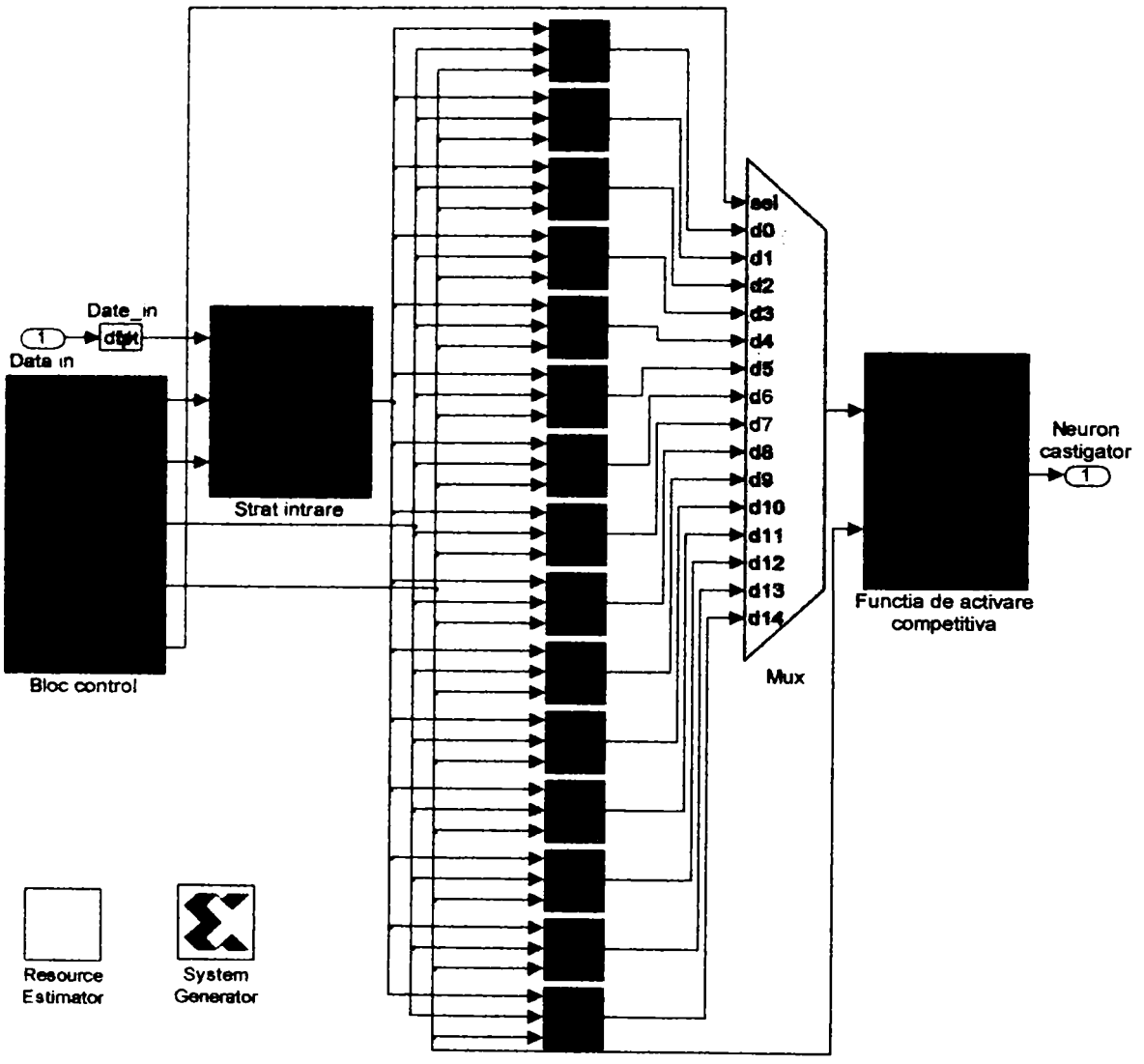


Figura 6.9 Modelul hardware al RNA competitive cu paralelism de neuron

Modificările aduse structurii RNA pentru implementarea paralelismului de neuron vor fi prezentate în continuare. Semnalele de comandă sunt generate de blocul de comandă care trebuie adaptat în funcţie de numărul de neuroni. Acest bloc generează adresa de înscriere a vectorului de intrare în stratul de intrare, realizat sub forma unei memorii dual port. Blocul de comandă mai generează adresa de la care se citesc datele din memoria stratului de intrare care sunt folosite şi de memoriile de ponderi din structura fiecărui neuron. De asemenea generează semnalele de reset şi semnalul de selecţie pentru multiplexor. Adaptarea blocului de comandă este facilă deoarece parametrii cu privire la numărul de neuroni, numărul de operaţii pe durata

unui eşantion al semnalului de intrare, întârzierile, etc., se încarcă automat din mediul de lucru Matlab.

Semnalele generate de blocul de comandă sunt prezentate în figura 6.10:

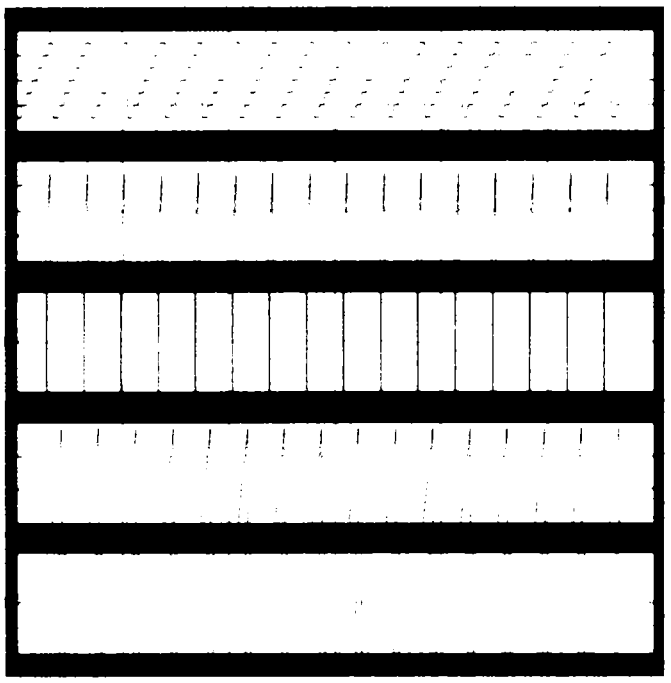


Figura 6.10 Semnalele generate de blocul de c-dă

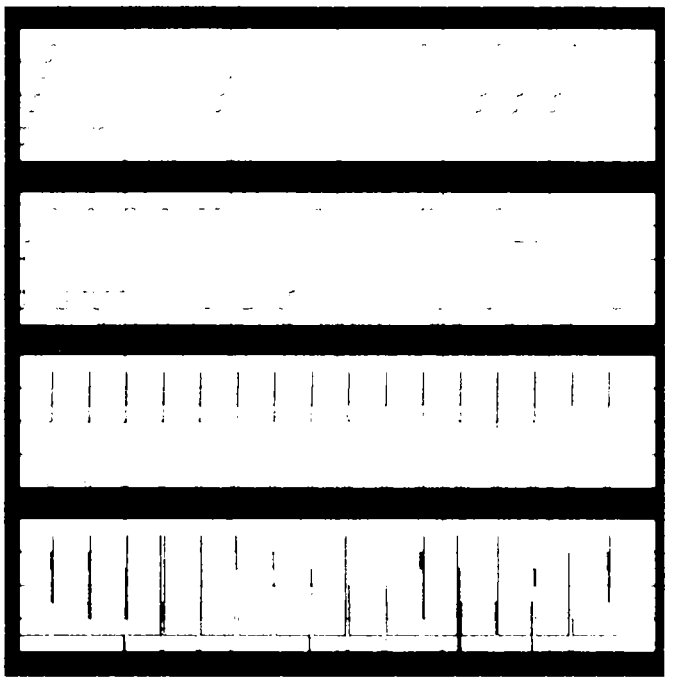
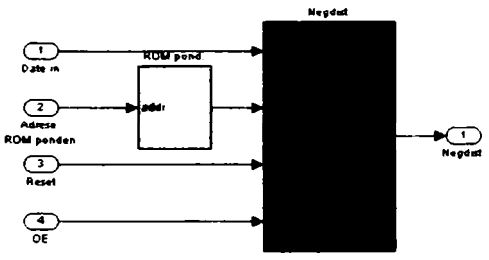


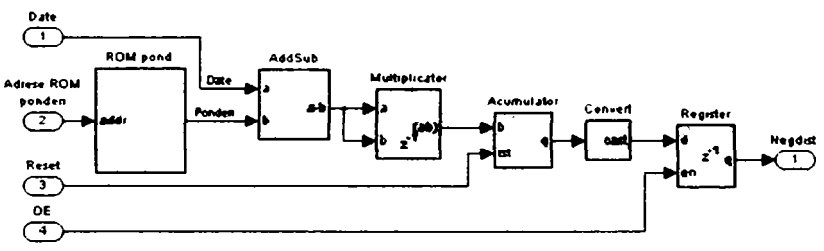
Figura 6.11 Forme de undă strat intrare

Stratul de intrare are aceeaşi structură ca şi în cazul RNA cu paralelism de strat diferă doar frecvenţa cu care se înscriu, respectiv se citesc datele. Figura 6.11 prezintă formele de undă corespunzătoare stratului de intrare.

Un neuron conţine pe lângă blocul de calcul al distanţei Euclidiene, similar cu cel prezentat în figura 6.5, şi memoria de ponderi, aşa cum se prezintă în figura următoare:



a) Schema bloc



b) Structura cu blocuri Xilinx

Figura 6.12 Neuron din structura RNA competitive cu paralelism de nod

În figura 6.12.b se prezintă structura unui neuron realizat folosind blocurile Xilinx. Blocul AddSub calculează diferenţele dintre date şi ponderi, iar blocul multiplicator ridică la putere aceste diferenţe. blocul acumulator face suma pătratelor diferenţelor, rezultatele fiind stocate temporar în registrul de ieşire. Astfel blocul de mai sus implementează cu succes relaţia 6.5 care permite determinarea ieşirii nete a neuronului.

Pentru calculul minimului şi determinarea neuronului care are ieşirea de valoare minimă s-a implementat un bloc de calcul al funcţiei de activare competitivă, similar celui din figura 6.6. Deoarece există un singur bloc al funcţiei de activare competitivă, ieşirile neuronilor sunt furnizate succesiv la intrarea lui cu ajutorul unui multiplexor cu 15 intrări. Semnalul de selecţie pentru multiplexor este generat de blocul de comandă.

Un detaliu cu formele de undă corespunzătoare primilor trei vectori de test pentru RNA competitivă cu paralelism de neuron sunt prezentate în figura 6.13.

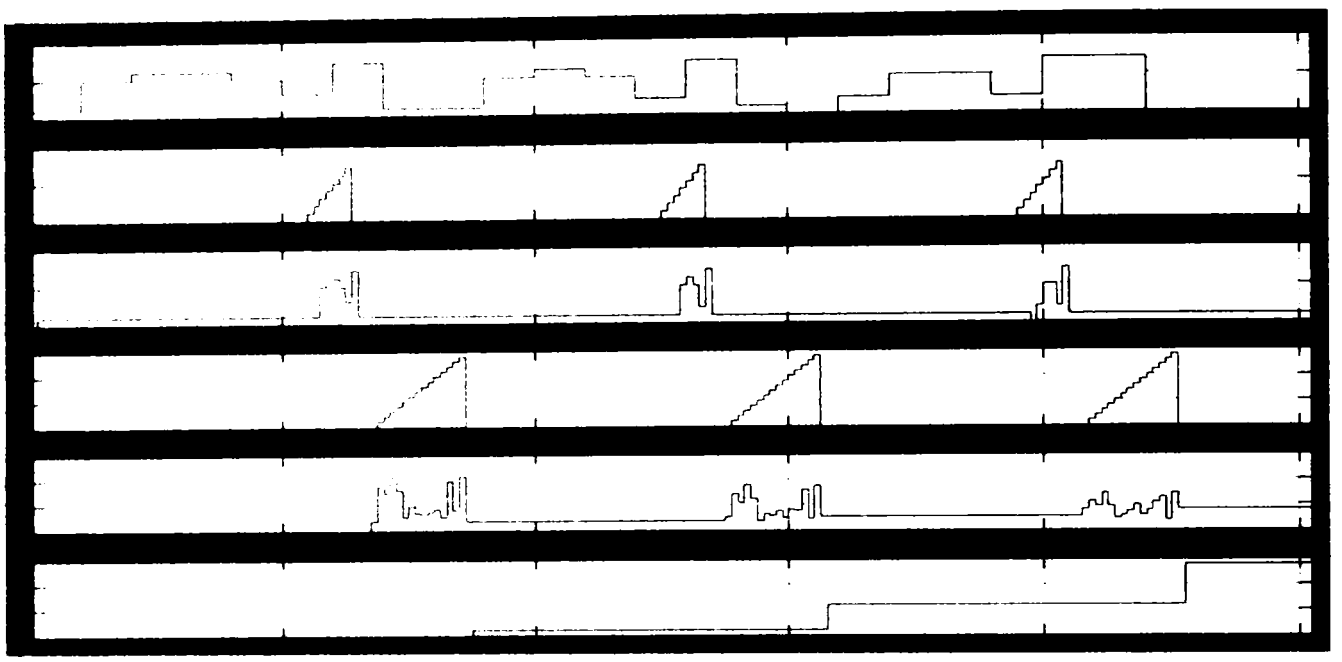


Figura 6.13 Forme de undă corespunzătoare răspunsului RNA la primii 3 vectori

Simulările modelului hardware pentru două seturi distincte de câte 15 x 10 vectori de test sunt prezentate în Anexa 8.

Rezultatele implementării RNA competitive cu paralelism al neuronilor într-un circuit FPGA XC2V1000 sunt prezentate în tabelul următor:

Tabelul 6.2 Resursele utilizate de RNA competitivă cu paralelism neuronal

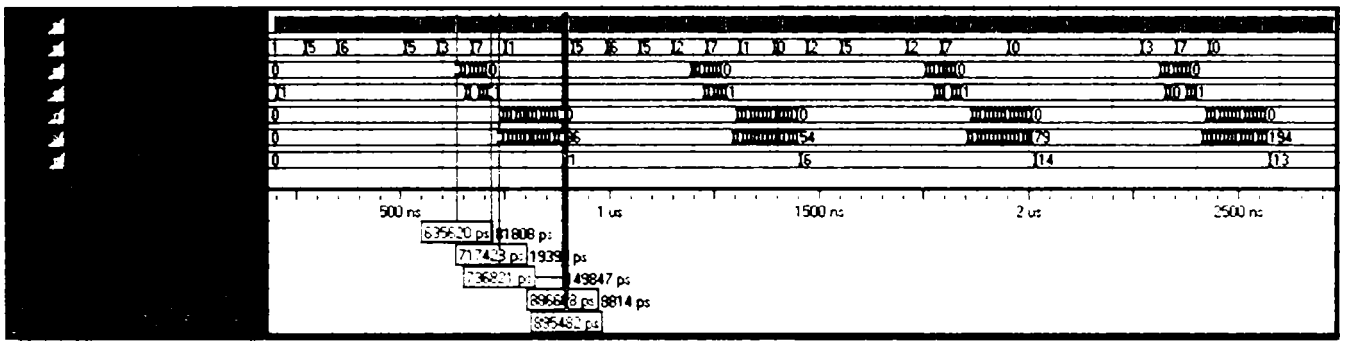
	Bloc c-dă	Strat intrare	15 Neuroni (resurse/neuron)	Multiplexor ieşiri neuroni	Funcţie activare	TOTAL	% din XC2V1000
Slice-uri	19	0	12	43	29	271	5,27
Bistabile	17	0	18	2	23	312	3,05
Bloc RAM	0	1	1	0	0	16	40
Tabele de memorii	29	0	12	81	35	325	3,18
Multiplicatoare	0	0	1	0	0	15	37,5

Resursele utilizate pentru implementarea unui neuron sunt de numai 12 slice-uri, 1 Block RAM şi un multiplicator dedicat. Totalul resurselor necesare pentru implementarea RNA competitiv cu 15 neuroni, reprezintă doar un mic procent din totalul resurselor disponibile în circuit, cu excepţia blocurilor de memorie şi a multiplicatoarelor dedicate care sunt utilizate în proporţie de 40% respectiv 37,5%. Pentru a implementa mai mult de 40 de neuroni blocul multiplicator poate fi implementat folosind logica distribuită în locul multiplicatoarelor dedicate al căror număr este limitat la 40. Resursele per neuron cresc astfel la 22 slice-uri, 30 bistabile, 1 block RAM, 31 LUT-uri, 0 multiplicatoare. De asemenea memoriile pot fi implementate folosind memoria distribuită disponibilă în circuit (160 kbiţi) în loc de memoriile de tip Block RAM. Resursele necesare mai cresc în acest caz cu 3 slice-uri, 3 bistabile şi 3 LUT-uri. Se poate face o estimare a numărului maxim de neuroni competitivi care pot fi implementaţi în acest circuit, ca în tabelul următor. În urma calculelor efectuate rezultă că numărul maxim de neuroni cu circuitele aferente ce pot fi implementaţi în circuitul ales este de aproximativ 181. Dintre aceştia 40 se implementează cu multiplicatoare dedicate şi 141 folosind logica distribuită.

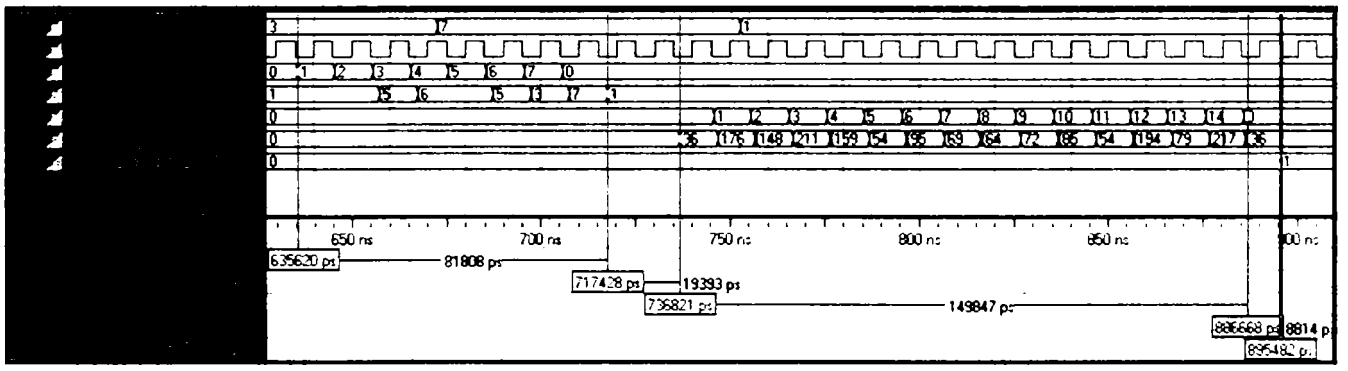
Tabelul 6.3 Resursele utilizate de numărul maxim de neuroni ce se pot implementa in XC2V1000

	RNA 15 neuroni varianta 1	RNA 40 neuroni varianta 1	Total XC2V1000	Disponibil	RNA 15 neuroni varianta 2	Nr. neuroni	TOTAL 40 neuroni var 1+ 141 neuroni var 2
Slice-uri	271	723	5120	4397	466	141	5103
Bistabile	312	832	10240	9408	640	221	6848
Bloc RAM	16	40	40	0	0	-	40
LUT-uri	325	867	10240	9373	655	215	7024
Multiplicatoare	15	40	40	0	0	-	40

Rezultatele simulării funcţionale a reţelei implementate, mai exact un detaliu asupra primilor 4 vectori de test şi asupra timpilor de calcul a răspunsului pentru primul vector de test sunt prezentate în figura 6.14.a respectiv 6.14.b.



a) Detaliu asupra primilor 4 vectori de test



b) Detaliu asupra timpilor de calcul pentru primul vector de test

Figura 6.14 Simularea funcţională a RNA competitive cu paralelism de neuron

Simularea s-a făcut cu un semnal de ceas de 100 MHz. Elementele vectorilor de intrare sunt înscrise în memoria stratului de intrare în mod succesiv la intervale egale de timp:

$$T_s = (n_1+1)T_{clk} = 80 \text{ ns} \qquad (6.8)$$

Perioada vectorului de intrare este de 560 ns.

Stratul de intrare transmite succesiv aceste elemente plus bias-ul, stratului de ieşire la intervale de timp $Ts/(n_1+1)$. Fiecare din cei n_2 neuroni ai acestui strat efectuează (n_1+1) calcule. Transferul ieşirilor către blocul de calcul al funcţiei de activare competitive se face în 15 perioade de ceas. Neuronul care câştigă competiţia este determinat după 8,8 ns.

Rezultatele simulării arată că primul vector de test din figură [1 5 6 6 5 3 7] este atribuit primei clase, cel de al doilea vector [1 1 5 6 5 2 7] clasei 6, ş.a.m.d.

Frecvenţa maximă a semnalului de ceas din raportul de sinteză este de 123,7 MHz. Frecvenţa maximă cu care se pot aplica elementele vectorului de intrare este dată de relaţia:

$$F_{s\,max} = F_{clk\,max}/(n_1+1) = 15{,}45 \text{ MHz} \tag{6.9}$$

6.3 Concluzii

În acest capitol s-a prezentat implementarea cu succes a unor reţele neuronale competitive simple utilizate în sarcini de clasificare a modelelor. Antrenarea reţelei se face software cu ajutorul programului Matlab.

Prima reţea competitivă implementată este cea cu paralelism de strat care dispune de un singur element de procesare. Pentru modelarea reţelei în Simulink/System Generator au fost create şi adăugate bibliotecii RNA Blockset două blocuri noi şi anume blocul de calcul al distanţei Euclidiene şi blocul funcţiei de activare competitive. Pentru a putea fi implementat calculul distanţei Euclidiene s-a făcut o simplificare pornind de la observaţia că pentru a compara radicalul a două numere este suficient să comparăm cele două numere. Astfel a fost evitată implementarea hardware a operaţiei de extragerea radicalului. Structura reţelei este simplă şi foarte flexibilă permiţând implementarea unui număr oarecare de neuroni, parametrii reţelei fiind preluaţi din mediul de lucru Matlab. RNA implementat clasifică corect toţi vectorii de antrenare şi două seturi diferite de câte 15 x 10 vectori de test. Resursele utilizate sunt minime (în jur de 1,3% dintr-un circuit Virtex II XC2V1000 pentru o reţea cu 15 neuroni) ceea ce permite realizarea unor reţele de dimensiuni mari. Frecvenţa maximă de furnizare a vectorilor de intrare este de 1,133 MHz.

Cea de a doua reţea implementată este cu paralelism de neuron ea dispunând de atâtea blocuri de calcul al distanţei Euclidiene câţi neuroni are reţeaua, şi un singur bloc al funcţiei de activare. Structura reţelei se modifică în funcţie de numărul de neuroni care se doresc a fi implementaţi. Şi această reţea clasifică corect toţi vectorii de antrenare şi de test furnizaţi. Resursele utilizate depind de numărul neuronilor în cazul implementării reţelei cu 15 neuroni aceasta utilizează 5,27% din numărul total de slice-uri şi 37,5 % din numărul de multiplicatoare dedicate. Numărul maxim de neuroni competitivi ce pot fi implementaţi într-un circuit ca cel specificat mai sus este estimat la aproximativ 181. Frecvenţa maximă cu care pot fi aplicaţi vectorii la intrarea reţelei reţele este de 15,45 MHz.

Primul tip de reţea este avantajoasă în cazul în care frecvenţa tiparelor care trebuiesc clasificate este mai mică de 1 MHz deoarece permite realizarea unor reţele de dimensiuni mari fără a modifica structura circuitului. În cazul unor vectori de intrare cu frecvenţa cuprinsă între 1 MHz şi 15 MHz trebuie utilizată cea de a doua reţea. Modelul trebuie creat în funcţie de numărul de neuroni, iar dimensiunea reţelei nu poate depăşii 180 de neuroni pentru circuitul specificat.

Pentru reţele de dimensiuni mai mari şi frecvenţe de lucru mai ridicate se pot folosi circuitele FPGA de dimensiuni mai mari şi frecvenţă de lucru mai mare. Astfel circuitul XC2VP125 din familia Virtex II Pro conţine 556 multiplicatoare dedicate şi peste 55.000 slice-uri ceea ce permite implementarea a peste 2000 de neuroni competitivi. Frecvenţa de lucru a acestor circuite poate atinge 400-500 MHz. Frecvenţa semnalului de intrare este $1/(n_1+1)$ din frecvenţa maximă a circuitului, unde n_1 reprezintă numărul de neuroni ai stratului de intrare (numărul de dimensiuni ai vectorului de intrare).

Dintre contribuţiile aduse de autor în acest capitol se pot amintii:

- Crearea modelului hardware pentru blocul Negdist care permite calculul sumei pătratelor diferenţelor dintre elementele a doi vectori (relaţia 6.5).
- Conceperea unui algoritm pentru determinarea neuronului pentru care distanţa dintre vectorul pondere şi vectorul de intrare este minimă.
- Crearea modelului hardware pentru blocul funcţiei de activare competitive.
- Modelarea unei RNA competitive cu paralelism de strat folosind blocurile IP create.
- Modelarea în Simulink/System Generator a unei RNA competitive cu paralelism de neuroni.
- Estimarea resurselor ocupate în circuitul FPGA şi alegerea unui circuit FPGA cu caracteristicile cele mai potrivite.
- Determinarea frecvenţei maxime a semnalului de intrare în funcţie de frecvenţa maximă a circuitului şi parametrii reţelei (număr de intrări respectiv număr de neuroni).
- Crearea unui bloc de evaluare a erorilor modelului hardware.
- Simularea, implementarea şi verificarea experimentală a tuturor modelelor concepute cu ajutorul platformei hardware.

Capitolul VII

Aplicaţii ale RNA hibride la recunoaşterea gesturilor statice ale unei mâini

7.1 Soluţia aleasă pentru sistemul senzorial al mâinii artificiale

În urma studiului efectuat în [189], precum şi a celor prezentate în capitolul I s-a dedus că un sistem senzorial minimal pentru o mână artificială trebuie să fie compus din senzori de poziţie a articulaţiei degetelor completaţi cu senzori de forţă. De aceea soluţia propusă în această lucrare constă dintr-o mănuşa senzorială care permite determinarea poziţiei articulaţiei degetelor pe care se pot fixa 6 senzori de forţă de contact amplasaţi pe vârful degetelor şi în palmă pentru a determina contactul cu un obiect şi forţa care se exercită [187], [188].

7.1.1 Senzorii de forţă de contact

Folii tactile care determină existenţa şi mărimea forţelor externe sunt bazate pe rezistoare detectoare de forţă (Force Sensing Resistor - FSR) din materiale semiconductoare. Fiecare deget are câte un senzor pentru vârful degetului. Aceşti senzori sunt capabili să măsoare forţe cuprinse între 0,5 N şi 10 N, cu o rezoluţie de 35 mN. Figura 7.1 prezintă localizarea acestor senzori de contact. Foliile subţiri flexibile de FSR sunt potrivite pentru montarea pe suprafeţe diverse şi pot fi fabricare în forme speciale.

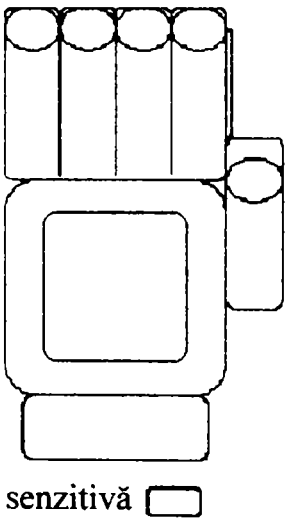


Figura 7.1 Localizarea FSR pe mănuşa senzorială

7.1.2 Senzorii de poziţie şi de înclinare

Din multitudinea de soluţii posibile pentru determinarea poziţiei articulaţiilor unei mâini s-a ales varianta unei mănuşi senzoriale din dorinţa de a crea o interfaţă om-maşină cât mai naturală, portabilă, suficient de robustă şi fiabilă şi nu în ultimul rând, din raţiuni economice.

7.1.2.1 Mănuşa senzorială

În ultimii ani odată cu creşterea în popularitate a realităţii virtuale (VR) a început să fie folosită pe o scară din în ce mai largă un nou dispozitiv de intrare: mănuşa senzorială (sau mănuşa de date). Mănuşa de date este un dispozitiv multisenzorial care poate furniza o mare

cantitate de date. Datorită caracteristicilor sale de interfaţare naturală cu mâna umană, mănuşa senzorială creează posibilitatea îmbunătăţirii performanţelor în domeniile în care azi este tot mai utilizat: teleoperarea şi comanda roboţilor [51], [146], telechirurgia şi antrenarea chirurgicală [51], [146], realitatea virtuală [16], [60], [61], [148], producţia industrială cu aplicaţii CAD/CAM [16], [146], etc.

Dintre multiplele posibilităţi existente pe piaţă cele mai cunoscute cinci sunt prezentate în continuare [27], [164], [243], [250], [252].

Figura 7.2 Manuşa VPL Dataglove

Figura 7.3 Mattel Power Glove

1. VPL DataGlove, una din cele mai populare mănuşi, utilizează fibre optice montate în buclă în jurul fiecărei articulaţii ale degetelor. Gradul de flexare este determinat prin măsurarea intensităţii luminii reflectate de fibră. Mănuşa VPL este confecţionată din Lycra, conţine 14 senzori şi are avantajul că este uşoară [250].
2. Mattel Power Glove [176] senzori de flexare realizaţi din două straturi de cerneală conductoare conţinând particule de carbon plasate pe un substrat. La flexarea substratului distanţa dintre particulele de carbon creşte ceea ce duce la creşterea rezistenţei senzorului. Această rezistenţă este transformată într-o valoare unghiulară printr-o procedură de calibrare.

Figura 7.4 Manuşa Vertex CyberGlove

Figura 7.5 Exos Dextrous Hand Master

3. Virtex CyberGlove [252] este echipată cu 16-24 timbre tensometrice situate pe un material elastic. Rezoluţia pentru unghiul articulaţiei este în jur de $0,5^0$ şi este constantă pentru toată amplitudinea mişcării.
4. Exos Dextrous Hand Master, are un schelet metalic fixat pe dosul palmei. Unghiul fiecărei articulaţii este măsurat de senzori cu efect Hall plasaţi pe articulaţiile scheletului mecanic. Are patru senzori pe fiecare deget. Avantajul faţă de alte mănuşi este reprezentat de sensibilitatea şi rezoluţia mare. [252].
5. Mănuşa 5DT – Data Glove 5 a firmei 5DT foloseşte senzori de flexare cu fibră optică [243]. Există mai multe tipuri de mănuşi cu 5 respectiv 14 senzori disponibile atât pentru mâna dreaptă cât şi pentru cea stângă. Figura 7.6 prezintă mănuşa 5DT 5.

Figura 7.6 Mănuşa 5DT- Data Glove 5

În alegerea tipului de mănuşă pe lângă caracteristicile tehnice, un rol important îl joacă şi preţul de achiziţie. Astfel preţurile la care pot fi achiziţionate mănuşile de date prezentate sunt următoarele: VPL Data Glove 8.000$, Matel PowerGlove 62$ (nu mai este disponibil), Virtex CyberGlove în funcţie de numărul de senzori intre 9.800 şi 14.500 $, Exox DHM 15.000$, şi 5DT-5 (varianta cu 5 senzori) 495$.

În urma analizei cerinţelor pe care trebuie să le îndeplinească sistemul senzorial pentru recunoaşterea gesturilor statice ale unei mâini artificiale, precum şi a raportului calitate - preţ, rezultă că soluţia optimă o reprezintă mănuşa senzorială 5DT 5. Aceasta a fost achiziţionată pentru a fi folosită în determinările experimentale din cadrul prezentei teze. Principalele ei caracteristici sunt prezentate în continuare.

Mănuşa senzorială 5DT 5 permite determinarea flexării degetelor şi orientarea mâinii (înclinare şi răsucire). Pentru aceasta mănuşa este dotată cu 5 senzori de poziţie (câte unul pentru fiecare deget) şi un senzor de înclinare cu două axe. De asemenea mănuşa este prevăzută cu o interfaţă serială (RS232). Figura 7.7 prezintă mănuşa şi modul de amplasare a senzorilor şi a circuitelor de interfaţare cu calculatorul, iar tabelul 7.1 prezintă rolul fiecărui senzor. Senzorii de poziţie a articulaţiei degetelor au o rezoluţie de 8 biţi deci permit determinarea a 256 poziţii pentru fiecare deget. Senzorul de înclinare are o rezoluţie tot de 8 biţi şi permite determinarea poziţiei mâinii cu o precizie de 0,5° în domeniul -60° la +60°. Comunicaţia cu calculatorul are loc prin interfaţa serială RS232 (3 fire: GND, TX, RX), full duplex la 19200 bps. Structura blocului de date este: 8 biţi date, 1 bit de stop, fără paritate.

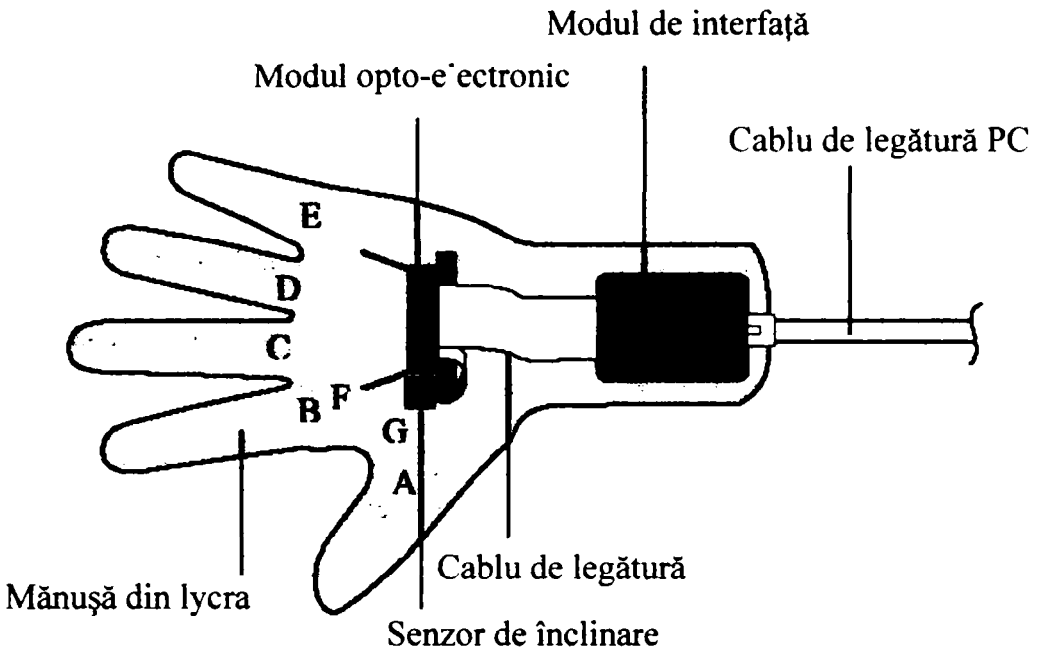


Figura 7.7 Mănuşa senzorială 5DT

Rata de eşantionare la mănuşa standard (5 degete, înclinarea şi răsucirea), este de 200 de eşantioane pe secundă.

Tabelul 7.1. Rolul senzorilor de pe mănuşa senzorială

Senzor	Descriere
A	Flexarea degetului mare
B	Flexarea degetului arătător
C	Flexarea degetului mijlociu
D	Flexarea degetului inelar
E	Flexarea degetului mic
F	Unghiul de înclinare a mâinii
G	Unghiul de răsucire a mâinii

Informaţiile sunt eşantionate şi trimise cu frecvenţa de 200 Hz. Un pachet de date constă în 9 octeţi şi are următoarea structură:

Header	D1	D2	D3	D4	D5	înclinare	răsucire	checksum

Header-ul este întotdeauna un octet cu valoarea 0x80 (128 zecimal), iar octetul corespunzător gradului de flexare a degetelor este asociat astfel:

D1 Degetul mare
D2 Degetul arătător
D3 Degetul mijlociu
D4 Degetul inelar
D5 Degetul mic

Gradul de flexare a fiecărui deget este codificat pe 8 biţi deci este cuprins în domeniul 0 la 255. O valoarea mică indică un deget neflexat.

Înclinarea şi **răsucirea** sunt reprezentate pe câte 8 biţi, deci au valori cuprinse între 0 şi 255. Valoarea 128 indică o mână neînclinată (valoare centrală).

Checksum serveşte la verificarea corectitudinii datelor recepţionate.

7.2 Recunoaşterea gesturilor

Prin gesturi se înţelege în general o mişcare a mâinii şi a corpului prin care se pot transmite informaţii de la o persoană la alta. În această lucrare prin gest se va înţelege strict un gest efectuat de mână. Gesturile în sens general pot fi dinamice sau statice. Gesturile statice se mai numesc posturi.

Definiţii:

Postura *este o combinaţie specifică a poziţiei, orientării şi flexării mâinii la un moment de timp dat.*

Gestul *este o secvenţă de posturi conectate prin mişcări într-o perioadă scurtă de timp. Datorită dependenţei de timp şi mişcare, un gest este numit adesea şi gest dinamic.*

Posturile nu variază în timp şi pot fi un element al unui gest sau de sine stătătoare. În continuare se va folosi atât termenul de postură cât şi cel de gest cu referire la un gest static iar gesturile dependente de timp vor fi denumite gesturi dinamice.

Există foarte multe domenii de aplicare a recunoaşterii gesturilor din cele mai diverse, de la manipularea roboţilor subacvatici sau spaţiali, până la recunoaşterea semnelor utilizate pentru comunicare de către persoanele cu dizabilităţi de auz sau vorbire. Dintre acestea se evidenţiază trei domenii mai importante:

- Controlul de la distanţă a roboţilor sau a vehiculelor în medii periculoase sau nocive pentru om, sub apă, în spaţiu, etc. Cerinţa care se pune este de controla simultan un număr mare de parametrii.
- Simularea operaţiilor chirurgicale sau telechirurgia. O aplicaţie specifică o reprezintă manipularea uneltelor endoscopice în interiorul corpului pacientului. De obicei acestea au unul sau două grade de libertate datorită controlului dificil a mai multor grade de libertate. Utilizarea unui sistem de recunoaştere a gesturilor permite controlul unor unelte pentru chirurgie endoscopică mai complexe având mai multe grade de libertate.
- Facilitarea comunicării pentru persoanele cu handicap. Există multe încercări în acest domeniu, dintre care cele mai multe se referă la recunoaşterea alfabetului limbajului prin semne.

Numărul posturilor care sunt necesar a fi recunoscute depinde de domeniul de aplicaţie. Astfel pentru manipularea în spaţiul virtual 3D, 3 posturi sunt în general suficiente: *apucă, start, stop.* În sisteme mai complexe trebuiesc recunoscute 5-10 posturi, în timp ce pentru recunoaşterea limbajului prin semne trebuie luate în considerare 26-51 posturi .

Prezenta lucrare are drept scop realizarea unui sistem autonom, capabil să recunoască 15 posturi care vor fi prezentate în continuare.

7.2.1 Metode pentru recunoaşterii gesturilor.

Există multe metode pentru recunoaşterea gesturilor incluzând filtrarea unghiulară, metode statistice, analiza componentelor principale, reţele neuronale, testarea gradului de similaritate a tiparelor, etc. Aceste metode se pot împărţii în două grupe mari: cele care pot învăţa să recunoască gesturi prin exemple, şi cele care nu pot. Dintre acestea se disting trei mai utilizate [175], [179]:

1. **Compararea tiparelor**. Dintre acestea cea mai cunoscută este metoda tabelară dezvoltată în [250]. Pentru fiecare postură, fiecare senzor are un domeniu de valori valide. La fiecare eşantionare datele citite de la senzor sunt comparate cu datele din tabelul posturilor. Dacă datele citite de la senzori sunt în interiorul domeniului pentru o postură dată, aceasta este recunoscută. Metoda deşi simplă prezintă mai multe dezavantaje. Astfel domeniul valorilor pentru fiecare postură trebuie să fie destul de mare (până la 30 %) datorită impreciziei senzorilor şi a impreciziei posturii adoptate de utilizator. Din această cauză pentru mai mult de 10 posturi domeniile se întrepătrund mult, ceea ce duce la alocarea unui gest al mâinii, mai multor posturi din tabel.

2. **Reţele neuronale.** Cunoscute pentru succesul lor în recunoaşterea tiparelor, reţelele neuronale sunt folosite în multe sisteme de recunoaştere a gesturilor. Avantajele oferite de RN sunt multiple: antrenarea pe baza exemplelor, recunoaşterea posturilor în prezenţa zgomotului sau a datelor incomplete, şi nu în ultimul rând generalizarea. Ultima proprietate joacă un rol crucial în performanţele sistemului deoarece nici măcar acelaşi utilizator nu reproduce gesturile cu acurateţe. Există multe sisteme de recunoaştere a gesturilor cele mai importante fiind [120], [125], [147], [196]-[201].

3. **Clasificare statistică.** În această metodă clasificarea posturilor se face pe baza statisticii vectorilor exemplu. Metoda foloseşte teorii de decizie statistică pentru a clasifica posturile.

Compararea metodelor de recunoaştere este dificil de realizat deoarece sistemele implementate nu lucrează cu aceleaşi gesturi şi nu au fost testate în acelaşi mod, etc. Cu toate acestea este evident că metoda de recunoaştere folosind reţele neuronale furnizează cea mai mare rată de recunoaştere.

7.2.2 Definirea posturilor

Setul curent de gesturi statice implementate de fabricantul mănuşii senzorială 5DT 5 utilizează o configuraţie binară închis/deschis a degetelor, exceptând degetul mare care nu este monitorizat. Nici informaţiile de înclinare şi rotire nu sunt luate în calcul. Există $2^4 = 16$ combinaţii posibile. Gestul 0 este definit cu toate degetele (exceptând degetul mare) închise şi gestul numărul 15 cu toate degetele deschise. Degetul arătător reprezintă bitul cel mai puţin semnificativ. De exemplu, gestul de indicare cu degetul arătător este numărul 1, şi indicarea cu degetul mic este numărul 8.

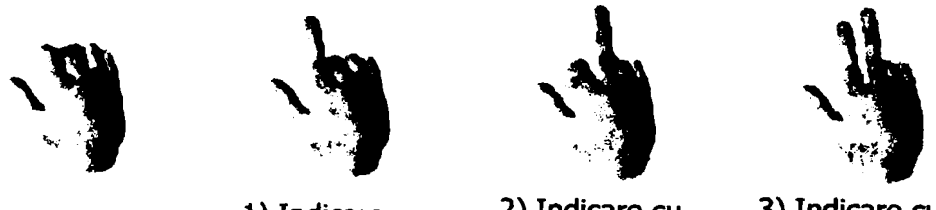
O ieşire a senzorilor mai mare decât pragul de sus indică un deget închis, iar o ieşire cu valoarea mai mică decât pragul de jos indică un deget deschis. O valoare între cele două praguri este invalidă şi generează un gest invalid.

Mănuşa are capabilitatea de a recunoaşte puţine gesturi şi de aceea este indicată folosirea reţelelor neuronale pentru implementarea unor algoritmi mai avansaţi de recunoaştere a gesturilor.

Tabelul 7.2 indică definiţia gesturilor statice implementate pentru mănuşa senzorială 5DT, iar figura 7.8 prezintă grafic aceste gesturi.

Tabelul 6.2 Definiţia gesturilor implementate pentru mănuşa senzorială 5DT comercială

Numărul gestului	Flexare (0=flexat, 1=neflexat)				Descrierea gestului
0	0	0	0	0	Pumn strâns
1	0	0	0	1	Indicare cu degetul arătător
2	0	0	1	0	Indicare cu degetul mijlociu
3	0	0	1	1	Indicare cu două degete
4	0	1	0	0	Indicare cu degetul inelar
5	0	1	0	1	Indicare cu degetele indicator şi inelar
6	0	1	1	0	Indicare cu degetele mijlociu şi inelar
7	0	1	1	1	Indicare cu trei degete
8	1	0	0	0	Indicare cu degetul mic
9	1	0	0	1	Indicare cu degetele indicator şi mic
10	1	0	1	0	Indicare cu degetele mic şi mijlociu
11	1	0	1	1	Indicare fără degetul inelar
12	1	1	0	0	Indicare cu degetele mic şi inelar
13	1	1	0	1	Indicare fără degetul mijlociu
14	1	1	1	0	Indicare fără degetul arătător
15	1	1	1	1	Mâna deschisă



0) Pumn strâns | 1) Indicare cu degetul arătător | 2) Indicare cu degetul mijlociu | 3) Indicare cu două degete

4) Indicare cu degetul inelar | 5) Indicare cu degetele indicator şi inelar | 6) Indicare cu degetele mijlociu şi inelar | 7) Indicare cu trei degete

8) Indicare cu degetul mic | 9) Indicare cu degetele indicator si mic | 10) Indicare cu degetele mic şi mijlociu | 11) Indicare fără degetul inelar

12) Indicare cu degetele mic şi inelar | 13) Indicare fără degetul mijlociu | 14) Indicare fără degetul arătător | 15) Mâna deschisă

Figura 7.8 Ilustrarea gesturilor implementate în mănuşa 5DT-5

Utilizarea reţelelor neuronale implementate în circuite FPGA permite extinderea capabilităţilor acestei mănuşi pe mai multe planuri:

- portabilitatea - algoritmul de recunoaştere fiind implementat hardware într-un circuit FPGA nu necesită un calculator;
- numărul gesturilor recunoscute poate fi mult mai mare, fiind dependent de capacitatea circuitului FPGA utilizat;
- complexitatea mai mare a gesturilor ce pot fi recunoscute – datorită folosirii datelor de la toţi cei 7 senzori, inclusiv cele referitoare la rotirea şi înclinarea mâinii;
- posibilitatea adăugării de noi gesturi sau modificarea celor existente – necesită reantrenarea RNA;
- scăderea timpului de răspuns – nu este importantă în cazul recunoaşterii gesturilor statice dar devine importantă în cazul în care se doreşte recunoaşterea gesturilor dinamice, când trebuie recunoscută o secvenţă rapidă de posturi şi luată o decizie cu privire la gest.

Sistemul de recunoaştere a gesturilor statice realizat de autor a fost testat prin implementarea a 15 gesturi uzuale, prezentate în figura 7.9:

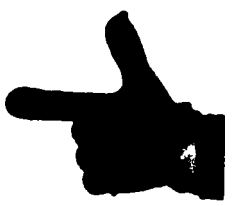
1-5: numerele de la unu la cinci cu palma în poziţie verticală,
6-10: numerele de la unu la cinci cu palma în poziţie orizontală,
11: pumn strâns,
12: semnul „victorie"
13: gestul apucării unui pahar,
14: gestul apucării unui instrument de scris,
15: STOP – palma ridicată vertical.

Tabelul 7.3 prezintă gesturile implementate de către autor şi datele achiziţionate de la ieşirea mănuşii pentru fiecare gest.

Tabelul 7.3 Definiţia gesturilor implementate folosind RNA

Număr gest	D1	D2	D3	D4	D5	Înclinare	Rotire	**Descrierea gestului**
0	44	189	236	211	177	83	247	Indicare nr. 1, palma vertical
1	45	46	211	211	179	79	247	Indicare nr.2, palma vertical
2	43	0	68	195	194	87	246	Indicare nr.3, palma vertical
3	245	7	12	13	17	109	244	Indicare nr.4, palma vertical
4	0	17	14	10	12	116	243	Indicare nr.5, palma vertical
5	46	189	211	211	177	144	112	Indicare nr.1, palma orizontal
6	40	45	206	210	182	139	115	Indicare nr.2, palma orizontal
7	28	0	71	181	182	141	115	Indicare nr.3, palma orizontal
8	255	17	12	23	13	144	107	Indicare nr.4, palma orizontal
9	0	14	14	15	6	148	111	Indicare nr.5, palma orizontal
10	206	152	179	200	206	5	254	Pumn strâns, palma vertical
11	255	12	48	187	189	34	102	Indicarea semnului „victorie"
12	97	113	100	94	61	125	223	Apucare pahar
13	103	21	163	163	173	133	185	Apucare instrument de scris
14	201	0	0	19	29	21	90	STOP – palma ridicată vertical

Gestul 1



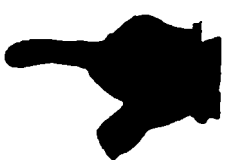
Gestul 2

Gestul 3

Gestul 4

Gestul 5

Gestul 6



Gestul 7

Gestul 8

Gestul 9

Gestul 10

Gestul 11

Gestul 12

Gestul 13

Gestul 14

Gestul 15

Figura 7.9 Ilustrarea gesturilor implementate de autor

7.3 Sistemul de recunoaştere a gesturilor statice

Sistemul de recunoaştere a gesturilor statice se compune dintr-o platformă hardware şi un pachet de componente software care sunt prezentate în continuare.

7.3.1 Hardware

Platforma hardware concepută şi realizată de autorul teze a fost prezentată în paragraful 2 al capitolului 4. Ea permite atât achiziţia vectorilor de antrenare şi de test, necesari în faza de antrenare a RNA, cât şi alimentarea cu date a RNA în faza de propagare. De asemenea această platformă permite implementarea reţelelor neuronale utilizate pentru recunoaşterea posturilor şi a circuitului de afişare a posturii recunoscute.

7.3.2 Software

Pachetul software pentru recunoaşterea posturilor are două componente principale: programul Matlab/Simulink şi programul Xilinx ISE.

Programul Matlab prin extensia Data Acquisition Toolbox permite achiziţia vectorilor de antrenare şi de test. Extensia Neural Network Toolbox permite antrenarea şi simularea reţelelor neuronale pentru recunoaşterea posturilor. Simulink/System Generator este utilizat pentru modelarea RNA cu blocuri Xilinx şi generarea fişierelor VHDL sintetizabile.

Programul Xilinx ISE are rolul de a sintetiza şi implementa într-un circuit FPGA codul VHDL reprezentând modelul RNA creat în System Generator.

7.3.3 Etapele implementării sistemului de recunoaştere a gesturilor

Etapele implementării sistemului de recunoaştere a posturilor sunt cele prezentate în paragraful 4.4 şi ele vor fi detailate în continuare:

1. **Achiziţia datelor de antrenare şi testare.** Prima etapă constă în achiziţia seturilor de antrenare şi testare. Utilizatorul a cărui posturi se doresc a fi recunoscute îmbracă mănuşa senzorială şi adoptă una din posturile prezentate în tabelul 7.3. Achiziţia este iniţiată din mediul Matlab prin rularea unui program conceput de autor (Anexa 1). Se specifică parametrii achiziţiei: dimensiunea vectorului de date şi numărul de eşantioane/vector dorit. În cazul posturilor definite în tabelul 7.3 vectorul de date are 7 componente cu valori cuprinse între 0 şi 255 şi au fost achiziţionate 10 eşantioane/vector. Vectorii sunt furnizaţi de sistemul de achiziţie în format paralel pe 8 biţi. La atingerea numărului prescris de eşantioane/vector utilizatorul este întrebat dacă doreşte să continue procesul de achiziţie a posturilor sau nu. Dacă se doreşte continuarea achiziţiei utilizatorul va adopta o nouă postură şi va răspunde afirmativ la întrebare. În cazul răspunsului negativ procesul se termină şi datele achiziţionate sunt înregistrate într-un fişier împreună cu informaţiile cu privire la numărul de vectori (posturi), dimensiunea unui vector şi numărul de eşantioane/vector. Au fost înregistrate mai multe seturi de vectori de la acelaşi utilizator respectiv de la mai mulţi utilizatori pentru a forma o bază de date necesară antrenării şi testării RNA create (anexa 6).
2. **Implementarea software a fazei de antrenare a RNA.** Modelarea, antrenarea şi simularea RNA se face folosind Neural Network Toolbox printr-un cod scris de autor sau prin intermediul interfeţei grafice Network/Data Manager. Datele achiziţionate servesc ca vectori de instruire pentru orice tip de reţea neuronală care se doreşte a fi

implementată. Se pot face mai multe simulări cu diverse arhitecturi de rețele neuronale, diverse tipuri şi algoritmi de învățare pentru a determina soluția care satisface cerințele impuse cu cele mai puține resurse (număr minim de straturi respectiv neuroni/strat).

3. **Realizarea modelarea hardware a sistemului de recunoaştere.** Modelarea sistemului de recunoaştere se face cu ajutorul Toolboxului System Generator din programul Simulink şi a bibliotecii RNA create de autor. Parametrii RNA (număr de neuroni, ponderi, bias, etc.) create folosind modulele din biblioteca RNA se încarcă automat din mediul de lucru Matlab. Se simulează sistemul şi dacă rezultatele sunt cele dorite se poate trece la generarea codului VHDL.
4. **Generarea descrierii VHDL a sistemului.** Toolboxul System Generator din Simulink permite generarea uşoară a unui cod VHDL dintr-o reprezentare a sistemului în Simulink. Pentru a maximiza predictibilitatea, densitatea şi performanța programul utilizează module sintetizabile, optimizate, de tipul LogicCORE.
5. **Sintetiza şi implementarea**. Codul VHDL este sintetizat şi implementat în dispozitive FPGA din familia Xilinx folosind programul Xilinx ISE 6.2i. sistemul poate fi simulat în oricare din fazele intermediare ale implementării cu ajutorul programului ModelSim.
6. **Încărcarea fişierului de configurare.** Fişierul de configurare "instruire.bit" se încarcă în circuitul FPGA folosind Cablul paralel IV şi programul Xilinx ISE 6.2i.
7. **Testare şi utilizare.** Datele senzoriale sunt furnizate în mod continuu în frecvența de 100 eşantioane pe secundă, în format paralel pe 8 biți de către sistemul de achiziție de date. Sistemul de recunoaştere a gesturilor afişează gestul recunoscut pe un afişaj cu 7 segmente.

7.4 Simularea sistemului pentru recunoaşterea gesturilor

Rețelele neuronale folosite pentru recunoaşterea posturilor diferă de la un autor la altul în funcție şi de posturile care au fost recunoscute [164], [179]. Pentru a putea alege o RNA capabilă să recunoască cât mai corect gesturile statice ale unei mâini, au fost simulate mai multe tipuri de RNA precum şi combinații ale acestora.

Obiectivul sistemului de recunoaştere fiind cea de clasificare a posturilor în 15 clase diferite, apare ca o soluție evidentă utilizarea unei rețele neuronale competitive simple. Stratul de intrare trebuie să fie compus din 7 neuroni, câte unul pentru fiecare componentă a vectorului de intrare. Stratul de ieşire competitiv este format din 15 neuroni, câte unul pentru fiecare din cele 15 gesturi. În urma antrenării rețelei cu datele de intrare se constată că datele nu sunt clasificate în 15 clase diferite, anumite ieşiri sunt active pentru două gesturi diferite, în timp ce alte ieşiri nu se activează niciodată.

O altă posibilitate ce rezultă din lucrările publicate de alți autori [179], [179] o reprezintă utilizarea unor rețele FF BP cu unul sau mai multe straturi ascunse. Au fost testate de autor, rețele având 7 neuroni în stratul de intrare, N_1 neuroni în stratul ascuns şi 15 neuroni în stratul de ieşire. Pentru N_1=7 RNA răspunde corect la 80% din vectorii cu care a fost antrenat şi 73% din vectorii de test, în timp ce pentru N_1=10 acest procent creşte la 93% pentru vectorii de antrenare şi 86 % pentru vectorii de test. Numărul minim de neuroni ai stratului ascuns trebuie să fie N_{1min}=11 pentru ca rețeaua să răspundă fără eroare când la intrarea ei se aplică setul de antrenare. Pentru setul de test răspunsul rețelei este corect în proporție de 88%. Funcția de activare a stratului ascuns este funcția sigmoid, iar stratul de ieşirea are funcția de activare liniară. Arhitectura RNA implementată este prezentată în figura următoare.

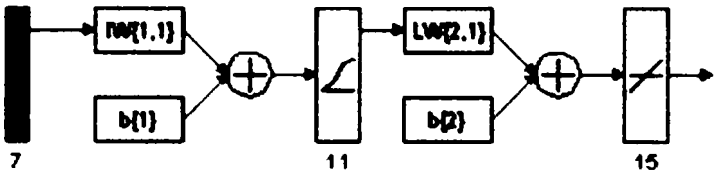


Figura 7.10 Arhitectura RNA FF-BP pentru recunoaşterea gesturilor

Simulând această reţea cu vectorii de antrenare se obţine rezultatul din figura următoare.

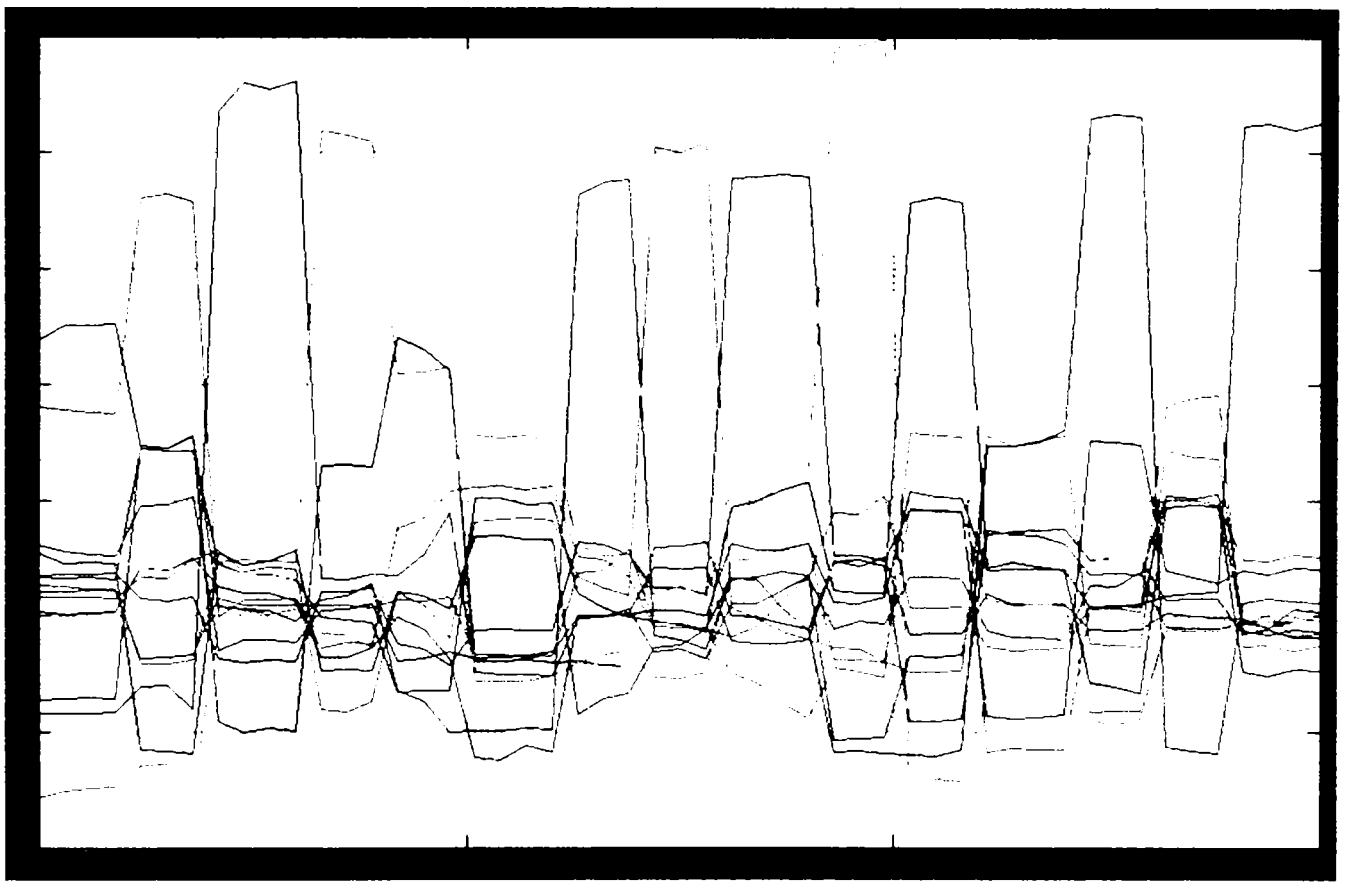


Figura 7.11 Simularea RNA FF-BP pentru recunoaşterea gesturilor

O problemă importantă este cea cu privire la calitatea datelor de intrare. Distribuţia datelor în spaţiul *M* dimensional al datelor de intrare este necunoscută. Deoarece nici interdependenţele dintre datele de intrare nu sunt cunoscute, este necesară utilizarea tuturor datelor senzoriale, chiar dacă doar o mică parte dintre acestea poartă informaţie importantă pentru recunoaşterea anumitor posturi. De asemenea calitatea datelor de intrare poate să difere nu numai de la un utilizator la altul dar chiar şi la acelaşi utilizator care adoptă aceeaşi postură. Datele de intrare pentru aceeaşi postură pot să difere şi datorită slabei repetabilităţi a ieşirii senzorilor. O îmbunătăţire esenţială a calităţii datelor poate fi obţinută prin preprocesarea datelor de intrare.

Mai mulţi autori au propus o preprocesare a datelor de intrare prin diverse metode [16]. O primă posibilitate o prezintă normalizarea datelor, astfel încât media lor devine zero şi eroarea medie pătratică unu

$$x_i^{norm}(t) = a_i x_i(t) + b_i \tag{7.1}$$

unde a_i şi b_i coeficienţi calculaţi anterior.

A doua posibilitate este filtrarea pentru a elimina zgomotele şi a face datele mai stabile. Acest lucru se poate face, de exemplu, folosind un filtru IIR de forma:

$$x_i^{IIR}(t) = kx_i^{norm}(t) + (1-k)x_i^{IIR}(t-1) \tag{7.2}$$

unde k este o constantă de filtrare.

O metodă mai avansată de procesare a datelor care duce la reducerea dimensionalităţii datelor de intrare este analiza componentelor principale.

Toate metodele de mai sus necesită însă implementarea unor algoritmi de calcul rigizi. La schimbarea naturii datelor trebuie modificat şi algoritmul. Din dorinţa de a implementa un mecanism de preprocesare universal şi flexibil autorul a propus şi experimentat cu succes utilizarea unei reţele neuronale pentru preprocesarea datelor.

Soluţia nouă propusă de autor pentru recunoaşterea gesturilor statice ale unei mâini constă în utilizarea a două reţele neuronale. O reţea FF cu rol de preprocesare a datelor, urmată de o reţea competitivă simplă utilizată pentru clasificarea datelor.

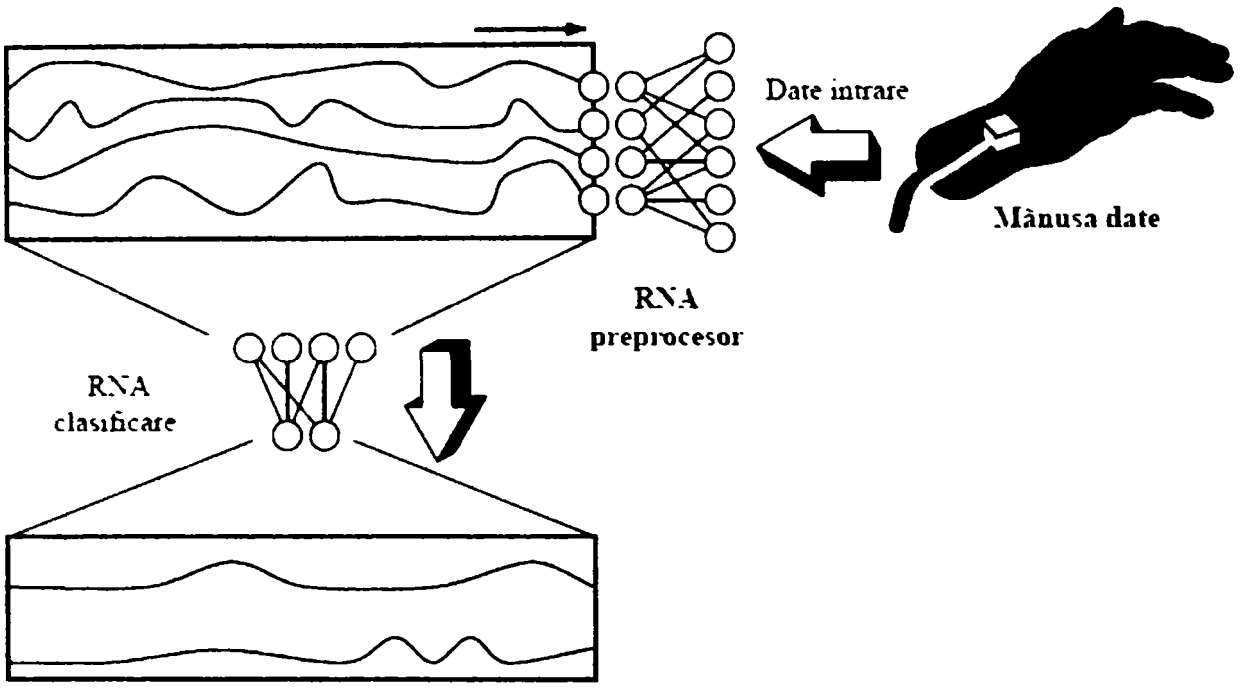


Figura 7.12 Structura sistemului pentru recunoaşterea gesturilor

Deşi mănuşa furnizează informaţii legate de poziţia degetelor cu o precizie de 8 biţi, o precizie de 3 biţi este în general suficientă pentru majoritatea aplicaţiilor. De aceea se poate folosi RNA FF cu rolul de filtrare şi de scalare a datelor de intrare din domeniul [0-255] în domeniul [0-7]. Această RNA ţine cont de toate componentele vectorului de intrare, precum şi de interdependenţele dintre acestea. Defectarea unui senzor nu duce la compromiterea totală a datelor de intrare, existând posibilitatea aproximării acestora. La ieşirea acestei reţele se obţine o estimare a poziţiei degetelor într-una din cele 8 poziţii alese, de la complet întins până la complet îndoit.

Funcţia implementată de neuronii primei reţele este:

$$y_j^{(1)}(x) = f\left(\sum_{i=1}^{M} w_i^{(1)} x_i^{(1)} - \theta^{(1)}\right) \tag{7.3}$$

unde j =1,2,..., N_1, reprezintă neuronii primei reţele, (N_1=7 în cazul nostru), iar i=1,2,...M, (M=7) reprezintă numărul de intrări ai fiecărui neuron.

Pentru funcţia de activare liniară relaţia 7.3 devine:

$$y_j^{(1)}(x) = \sum_{i=1}^{M} w_i^{(1)} x_i^{(1)} - \theta^{(1)} \tag{7.4}$$

Ieşirea netă a neuronilor din cea de a doua reţea este dată de relaţia:

$$net_k = \sqrt{\sum_{j}^{N_1} \left[y_j^{(1)} - w_{kj}^{(2)} \right]^2} \tag{7.5}$$

unde k =1,2,..., N_2, (N_2=15) reprezintă neuronii celei de a doua reţele. În urma simplificării adoptate în paragraful 6.2.2 expresia 7.5 devine:

$$net_k = \sum_{j}^{N_1} \left[y_j^{(1)} - w_{kj}^{(2)} \right]^2 \tag{7.6}$$

astfel că ieşirea netă a neuronilor competitivi este dată de relaţia următoare:

$$net_k = \sum_{j}^{N(1)} \left[\left(\sum_{i=1}^{M} w_i^{(1)} x_i^{(1)} - \theta \right) - w_{kj}^{(2)} \right]^2 \tag{7.7}$$

Funcţia de activare a neuronilor din cea de a doua RNA este funcţia competitivă.

Figura următoare prezintă arhitectura reţelelor neuronale din componenţa sistemului de recunoaştere a gesturilor statice ale unei mâini.

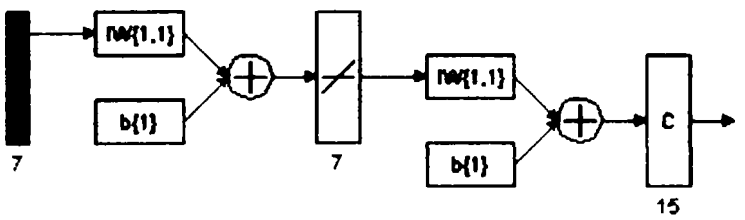


Figura 7.13 Arhitectura reţelelor neuronale pentru recunoaşterea gesturilor

Simularea sistemului format din cele două reţele neuronale în urma aplicării la intrare a setului de date de test compus 10 vectori M dimensionali (M=7) pentru fiecare din cele 15 gesturi, este prezentată în figura următoare. Se poate observa clasificarea datelor de intrare în 15 clase diferite.

Pentru a determina precizia cu care sistemul recunoaşte gesturile învăţate s-a conceput un script Matlab care calculează erorile (Anexa 9). Sistemul conceput recunoaşte setul de vectori de antrenare în proporţie de 100%. Sistemul a fost testat şi cu alte două seturi de vectori de test reprezentând gesturi statice formate de acelaşi utilizator, procentul de recunoaştere fiind tot de 100%. La testarea cu un set de gesturi obţinute de la un alt utilizator

doar 5 vectori din cei 150 au fost clasificaţi eronat, deci procentul de recunoaştere este de 96,67%. Trebuie subliniat că un asemenea procent ridicat de recunoaşterea gesturilor statice nu a mai fost raportat de nimeni. De asemenea merită subliniat faptul că în comparaţie cu soluţiile propuse de alţi autori, de exemplu utilizarea unor reţele FF-BP cu mai multe straturi, pe lângă precizia mai ridicată, resursele necesare sunt mai mici. Astfel, cum s-a arătat şi la începutul acestui paragraf, în condiţiile implementării sistemului de recunoaştere folosind o reţea FF-BP având 11 neuroni în stratul ascuns şi 15 în stratul de ieşire (soluţie propusă de alţi autori), procentul de recunoaştere este de 100% pentru setul de antrenament dar scade 85,33% şi 86,66% pentru seturile înregistrate de acelaşi utilizator, respectiv la 58,66% pentru setul de gesturi înregistrate de un alt utilizator. În concluzie soluţia propusă este superioară altor implementări cunoscute.

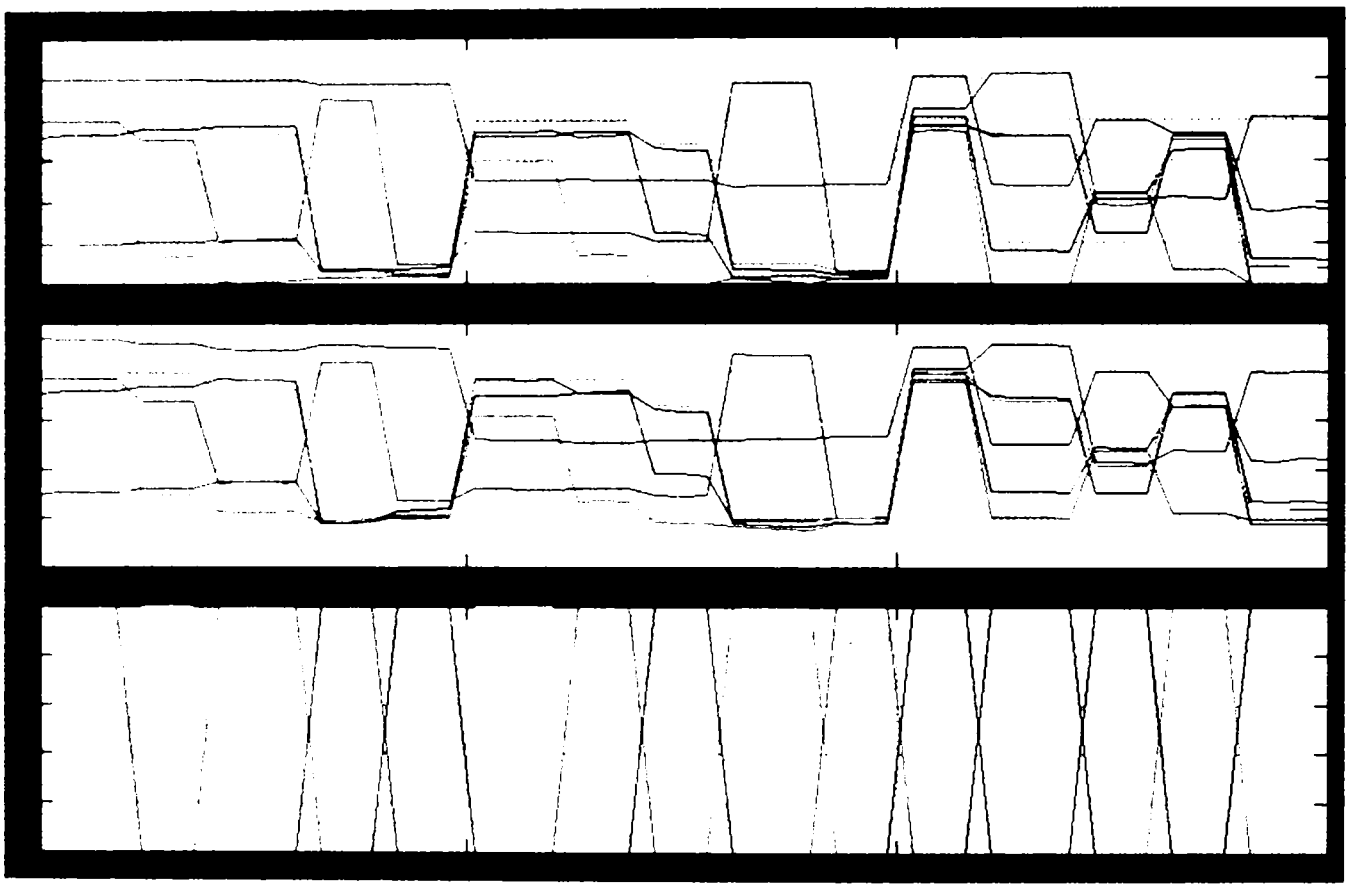


Figura 7.14 Simularea sistemului de recunoaştere a gesturilor statice

Deoarece simulările sistemului conceput pentru recunoaştere a gesturilor statice ale unei mâini corespunde pe deplin cerinţelor, se poate trece la implementarea lui folosind circuite logice programabile FPGA.

7.5 Modelarea sistemului pentru recunoaşterea gesturilor

Modelul hardware al sistemului de recunoaştere a gesturilor este o implementare cu blocuri din biblioteca RNA creată, a modelul Simulink din figura 7.13. Pentru implementarea reţelei de preprocesare s-a implementat un strat neuronal antrenat cu algoritmul Hebbian cu paralelism de strat de tipul celei prezentate în figura 5.42 din paragraful 5.5 Cea de a doua reţea, cu rol de clasificare, s-a implementat folosind modelul hardware a RNA competitive cu paralelism de strat din figura 6.4 a paragrafului 6.2.2. Astfel modelul creat pentru Sistemul de recunoaştere în Simulink/System Generator are structura din figura următoare:

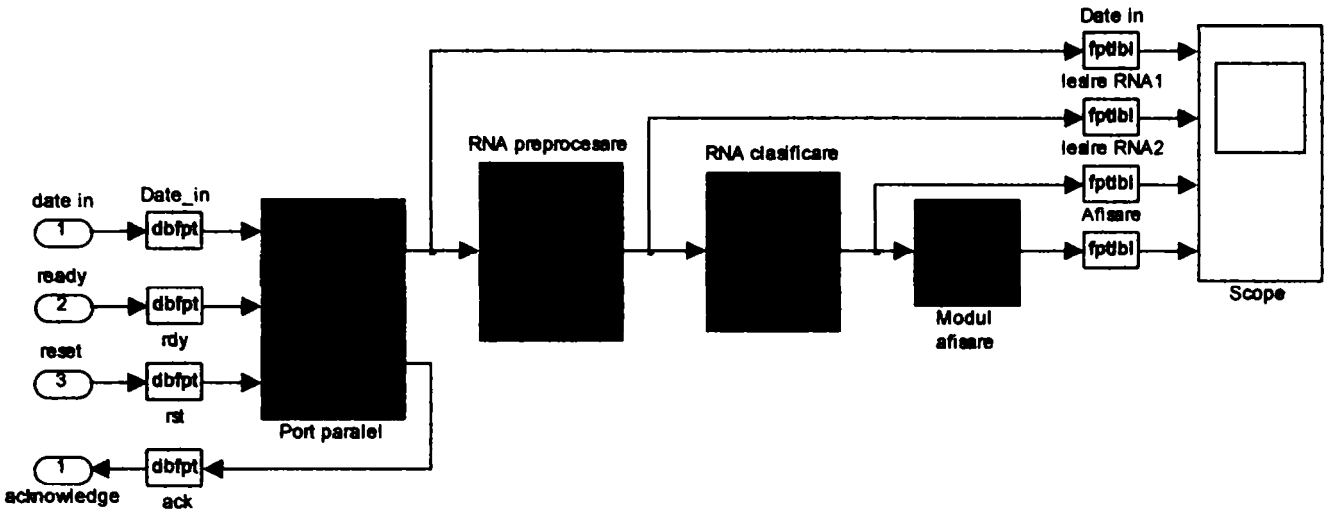


Figura 7.15 Modelul System Generator al sistemului de recunoaştere a gesturilor

Pe lângă cele două reţele neuronale, sistemul are în componenţă un port paralel prin intermediul căruia comunică cu sistemul de achiziţie de date şi un modul de afişare agestului recunoscut. Portul paralel implementează un protocol asincron cu confirmare şi condiţionare totală prin intermediul semnalelor ready şi acknowledge.

În urma simulării modelului hardware din figura 7.15 se obţin procente de recunoaştere foarte bune. Pentru setul de antrenament şi cele două seturi de test înregistrate de la acelaşi utilizator, care a furnizat şi setul de antrenament, procentul de recunoaştere este de 100%, în timp ce pentru setul de gesturi înregistrat de la un alt utilizator au fost recunoscute 138 din cei 150 de vectori, deci 92%. Aceste erori se datorează poziţiei incorecte a degetului mare, ieşirea senzorului fiind de cinci ori mai mare decât valoarea maximă corespunzătoare poziţiei. În consecinţă rata de recunoaştere poate fi considerată ca fiind 100% şi în acest caz. Figura 7.16 prezintă rezultatele simulării modelului hardware a sistemului de recunoaştere, când la intrare s-a aplicat setul de vectori de antrenare compus din câte 10 vectori pentru fiecare din cele 15 gesturi.

Figura 7.16 Simularea modelului HW a sistemului de recunoaştere a gesturilor

Pentru a vedea mai exact întârzierea răspunsului faţa de momentul aplicării datelor corespunzătoare unui gest la intrarea sistemului, în figura următoare se prezintă un detaliu al simulării. Se poate observa că există o întârziere de aproximativ un vector şi jumătate a răspunsului, mai exact 11/7*T (1/T fiind frecvenţa cu care se furnizează datele). Această întârziere se datorează în mai mică parte timpului necesar efectuării calculelor (4/7*T) şi în mai mare parte transferului datelor de la RNA preprocesor la RNA clasificator (T). Având în vedere că rata de eşantionare a sistemului de achiziţie este mai mare decât frecvenţa cu care se modifică posturile, rezultă că există mai mulţi vectori de intrare cu date identice şi deci întârzierea răspunsului faţă de primul vector dintr-o serie reprezentând un gest, nu afectează performanţele sistemului.

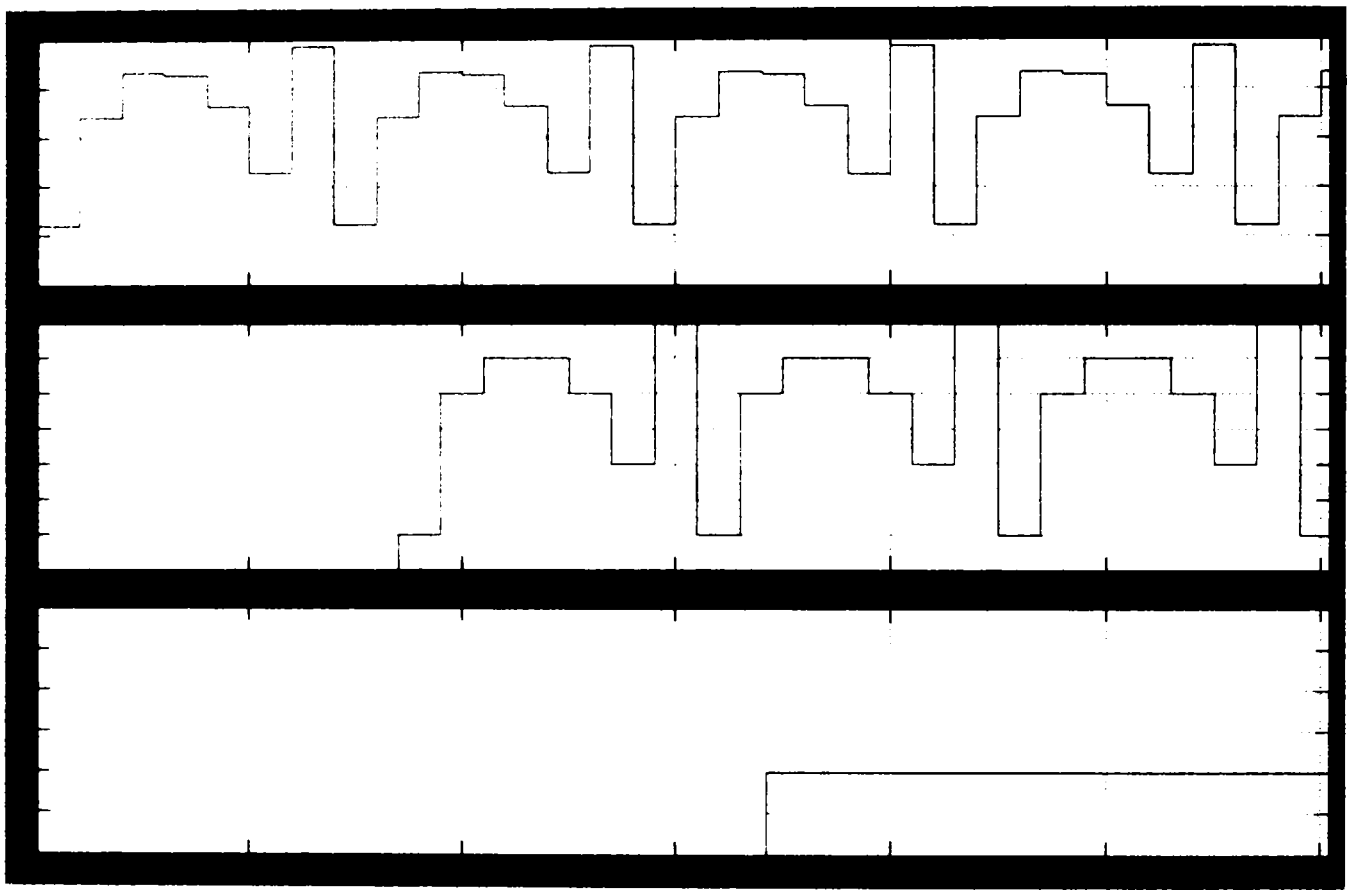


Figura 7.17 Detaliu de simularea pentru sistemul de recunoaştere a gesturilor

Anexa 10 prezintă în detaliu modelul sistemului de recunoaştere a gesturilor statice ale unei mâini artificiale.

7.6 Implementarea sistemului pentru recunoaşterea gesturilor

Sistemul de recunoaştere a gesturilor a fost implementat în diverse circuite FPGA. De exemplu pentru a exploata avantajul oferit de multiplicatoarele dedicate existente în circuitele Virtex s-a folosit circuitul XC2V1000. Sumarul raportului de implementare generat în urma implementării circuitului este prezentat in continuare:

Device utilization summary:

Number of MULT18X18s	2 out of 40	5%
Number of RAMB16s	5 out of 40	10%
Number of SLICEs	137 out of 5120	2%

Se constată că circuitele utilizează foarte puţin din resursele circuitului FPGA, ceea ce conduce la concluzia că sistemul poate fi implementat şi într-un circuit mai mic, chiar dacă acesta nu dispune de multiplicatoare dedicate.

Frecvenţa maximă de tact din raportul de implementare este de 121,86 MHz.

Simularea post implementare a circuitului este prezentată în figura următoare.

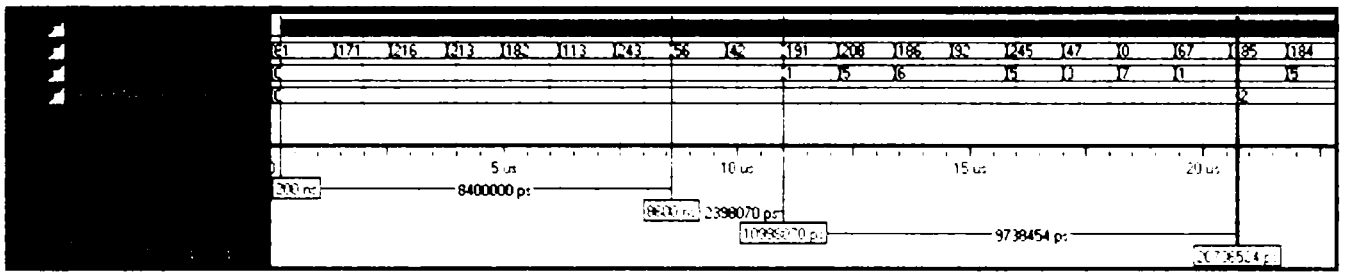


Figura 7.18 Simularea post implementare a sistemului de recunoaştere a gesturilor

Simularea corespunde unui semnal de ceas de 100 MHz. Perioada semnalului de intrare este de 7x120x10 ns = 8,4 μs. Timpul necesar pentru clasificarea gestului este de aproximativ 13 μs, mai mult decât suficient având în vedere că sistemul de achiziţie conceput transmite eşantioanele vectorului de intrare la intervale Ts=10 ms, ceea ce este echivalent cu aproximativ 14 gesturi pe secundă.

O simulare a sistemului în timp real, adică la o furnizare a elementelor vectorului de intrare cu frecvenţa de 100 Hz şi o frecvenţă a semnalului de tact de Fclk = 10 MHz, este prezentată in figura 7.19. Durata unui gest este de 70 ms, iar răspunsul este furnizat în aproximativ 106 ms de la aplicarea ultimului element al vectorului de intrare. Având în vedere că timpul minim pentru a îndoi respectiv întinde complet un deget este de 200 ms şi respectiv 160 ms conform cu [146], răspunsul sistemului poate fi considerat a fi „în timp real".

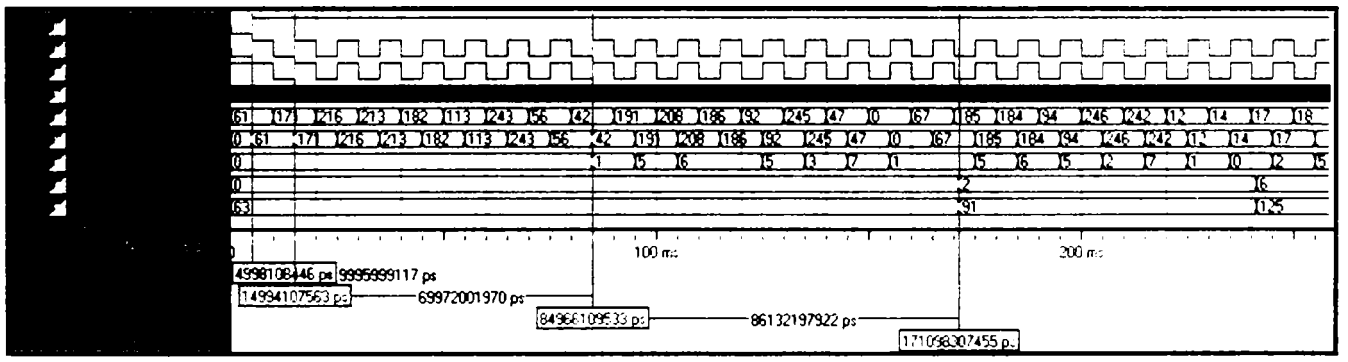


Figura 7.19 Răspunsul în timp real al sistemului de recunoaştere a gesturilor

7.7 Concluzii

Acest capitol prezintă realizările obţinute în recunoaşterea gesturilor statice ale unei mâini. S-a plecat de la dorinţă de a demonstra utilitatea şi buna funcţionare a unor reţele neuronale implementate hardware folosind circuite de tip FPGA. Din acest motiv cerinţele iniţiale impuse sistemului de recunoaştere nu au fost maximale obiectivul fiind utilizarea lui doar ca „platformă de testare". Pe măsura realizării sistemului folosind mediul HW-SW integrat pentru implementare RNA realizat în cadrul capitolului IV şi experienţa acumulată în urma implementării RNA din capitolele V şi VI, au apărut idei care au dus obţinerea unor rezultate notabile.

7.7.1 Comparaţii cu alte metode

Pentru a evalua rezultatele obţinute este util să le comparăm cu rezultatele altor lucrări similare. Acest lucru este dificil de făcut datorită seturilor diferite de posturi adoptate de autori. În figura 7.20 se prezintă o comparaţie a celor mai bune rezultate cunoscute [16], [164], [176], [179].

După cum se poate vedea doar Beale and Edwars au mai raportat un procent de recunoaştere de 100% dar ei au folosit un set de date de test compus din 5 gesturi simulate. Pentru recunoaştere au folosit o reţea FF având straturile compuse din 10-3-5 neuroni.

Murakami şi Taguchi au folosit o reţea FF cu structura 13-100-42, unde cele 42 de ieşiri reprezintă cele 42 de posturi ale alfabetului Japonez pe care au dorit să le recunoască. rata de recunoaştere este de 98% pentru gesturile furnizate de cel care a antrenat mănuşa şi 77% pentru alte persoane.

Fels şi Hilton raportează un procent de recunoaştere de 99,42% pentru datele antrenate şi 96,79% pentru datele de test. Reţeaua utilizată are câte 16, 80 şi respectiv 66 neuroni pe strat.

Prezenta lucrare are procentul maxim de reuşită atât pentru datele antrenate cât şi cele de test în condiţiile în care numărul total de neuroni implementaţi este foarte mic, ceea ce îl face implementabil şi într-un dispozitiv FPGA de cost redus.

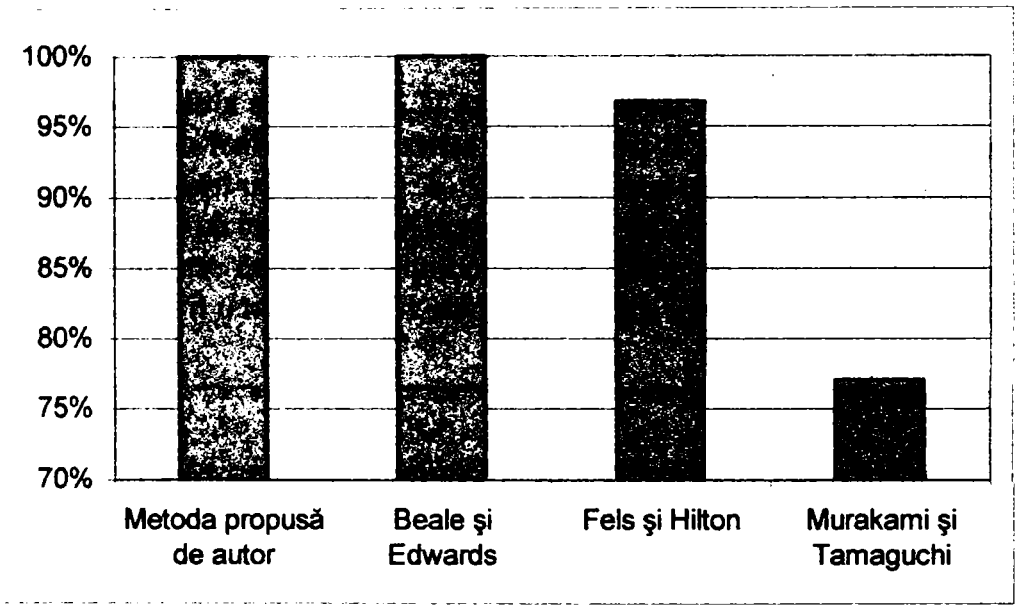


Figura 7.20 Cele mai bune rate de recunoaştere a posturilor

7.7.2 Aplicaţii posibile

Sfera aplicaţiilor posibile este vastă. Dintre posibilităţile de aplicare mai simple se pot aminti dispozitivele de intrare pentru calculator care permit manipularea unor obiecte virtuale sau în cadrul unor prezentări interactive. Utilizatorul purtând o asemenea mănuşă poate să indice pe imaginea proiectată de calculator, poate să treacă la imaginea următoare, etc.

Aplicaţiile mai importante sunt din domeniul reabilitării pacienţilor cu diverse tipuri de handicap. Sistemul poate servi de exemplu pentru asistarea recuperării bolnavilor în urma unor accidente când este necesară verificarea şi chiar cuantizarea abilităţii pacientului de a realiza anumite gesturi [127]. Poate cea mai interesantă aplicaţie este cea de interfaţă între limbajul semnelor şi un sintetizator vocal [198]. Astfel gestul recunoscut de sistem poate reprezenta o comandă pentru un sintetizator vocal. Există câteva încercări de acest gen dar majoritatea se rezumă la utilizarea unui calculator, în timp ce sistemul realizat permite realizarea unui dispozitiv mobil.

7.7.3 Contribuţii

Sistemul de recunoaştere a gesturilor statice, conceput şi realizat se bazează pe câteva idei noi propuse de autor. Astfel s-a propus utilizarea unei reţele neuronale pentru preprocesarea datelor în locul metodelor „clasice" de preprocesare. O altă noutate este utilizarea unor reţele competitive simple pentru clasificarea posturilor. Nu în ultimul rând, trebuie semnalată implementarea hardware a RNA folosind circuite FPGA, ceea ce face ca sistemul să se preteze pentru realizarea unui sistem de recunoaştere portabil. Toate ideile propuse au fost verificate cu succes cu ajutorul mediului de testare hardware/software realizat.

Cele mai importante contribuţii aduse în cadrul acestui capitol sunt următoarele:

- **Alegerea unei soluţii pentru un sistem senzorial minimal pentru o mână care să permită recunoaşterea gesturilor statice**. Pornind de la studiul sistemelor senzoriale folosite de alţi autori s-a stabilit că sistemul minimal pentru o mână trebuie să fie compus din senzori de poziţie (flexare) a degetelor şi mâinii.
- **Alegerea metodei de recunoaştere a gesturilor folosind reţele neuronale.** Metoda a fost aleasă, deoarece dintre metodele utilizate pentru recunoaşterea gesturilor aceasta este cea mai potrivită scopului propus.
- **Stabilirea gesturilor statice care trebuiesc recunoscute.** În urma studierii literaturii s-a optat pentru implementarea unor gesturi uzuale care să utilizeze informaţia senzorială de la cât mai mulţi senzori de care dispune mănuşa.
- **Realizarea practică a sistemului.** Sistemul a fost realizat folosind un sistem de achiziţie cu microcontroler şi o placă de dezvoltare pentru circuite FPGA, la care s-a adăugat o mănuşa senzorială şi circuite de interfaţare.
- **Crearea modelului sistemului pentru recunoaşterea gesturilor structurat pe două niveluri.** Primul nivel este realizat cu o reţea FF, iar cel de al doilea nivel realizat cu o reţea competitivă simplă, care are rolul de clasificare a datelor de intrare (determinarea gesturilor).
- **Utilizarea unei RNA FF pentru preprocesare datelor**. Acesta înlocuieşte circuitele obişnuite de preprocesare a datelor (filtrare, scalare, extragerea componentei principale, etc.).
- **Modelarea hardware a sistemului** pentru recunoaşterea gesturilor, implementarea şi testarea practică a acestuia.

Capitolul VIII

Contribuţii personale şi direcţii de continuare a cercetării

Realizarea unei mâini artificiale pentru un robot sau pentru protezarea unei mâini umane reprezintă o problemă de certă actualitate şi cu consecinţe aplicative deosebite. Abordarea acestei teme în cadrul prezentei teze s-a bazat pe domeniile de interes şi de competenţă ale autorului: senzori inteligenţi, circuitele logice programabile, protetică.

Rezultatele obţinute în prezenta teza aduc contribuţii în principal în următoarele trei direcţii:

1. Dezvoltarea unei metode şi a unui mediu integrat hardware-software, originale, pentru implementarea RNA în circuite FPGA,
2. Implementarea de RNA cu arhitecturi cunoscute în circuite FPGA,
3. Elaborarea unui sistem senzorial pentru recunoaşterea gesturilor statice ale unei mâini utilizând RNA implementate în circuite FPGA.

Implementarea reţelelor neuronale în circuite programabile este un domeniu cu o vechime relativ redusă. Deşi a trecut abia cu ceva mai mult de un deceniu de la primele implementări reuşite, rezultatele obţinute au demonstrat că aceasta procedură reprezintă o alternativă viabilă la metodele cunoscute de realizare hardware a RNA.

Contribuţia cea mai importantă a prezentei teze constă în elaborarea unei noi metode şi a unei mediu integrat HW-SW pentru implementarea RNA în circuite FPGA. Metodologia elaborata creează premisele pentru realizarea unui sistem portabil de recunoaştere a gesturilor statice ale unei mâini, fapt care permite utilizatorului (uman sau robot) să transmită informaţii prin gesturi către lumea înconjurătoare.

Recunoaşterea gesturilor în general şi a limbajului prin semne în mod special, este un domeniu deja tradiţional şi intens cercetat, cu multe teze de doctorat şi contracte de cercetare ştiinţifică elaborate. Domeniul rămâne în continuare de mare interes datorită multiplelor aplicaţii posibile.

8.1 Contribuţii personale

În opinia autorului, prezenta teză aduce contribuţii în principal cu privire la următoarele aspecte:

1. Realizarea unui studiu privind sistemul senzorial al unei mâini artificiale, studiu care a relevat care este configuraţia minimală a unui asemenea sistem precum şi necesitatea prelucrării informaţiilor senzoriale utilizând RNA. Pentru recunoaşterea gesturilor statice, sistemul senzorial minimal propus şi testat cu succes în teză, constă dintr-o mănuşă senzorială echipată cu senzori de poziţie a articulaţiilor degetelor şi a încheieturii mâinii.

2. Realizarea unui studiu comparativ al posibilităţilor de implementare a RNA. Pe baza acestuia se ajunge la concluzia că varianta de implementare hibridă: hardware-software, este cea mai eficientă, deoarece combină avantajul flexibilităţii implementării software a fazei de antrenare, cu cea de viteză, portabilitate şi consum mic de putere, specifice RNA implementate hardware.

3. Realizarea unui studiu aprofundat privind implementarea în circuite FPGA a RNA. Studiul scoate în evidenţă principalele probleme ce trebuiesc urmărite în cazul unor asemenea implementări şi sunt propuse totodată, pe baza lui, câteva criterii pentru evaluarea performanţelor RNA astfel realizate. Alegerea circuitelor FPGA pentru implementarea de RNA (corespunzător fazei de propagare) este o consecinţă logică a facilităţii de procesare paralelă oferite de aceste circuite.

4. Elaborarea unui mediu integrat hardware-software pentru implementarea hibridă a RNA. Platforma hardware se compune în principal dintr-un sistem de achiziţie de date, un sistem de dezvoltare cu circuite FPGA şi un calculator gazdă. Sistemul de achiziţie de date conectat la proces asigură datele de antrenare şi testare necesare în faza de antrenare respectiv achiziţia de date din proces în faza de propagare. Sistemul de dezvoltare cu FPGA permite implementarea efectivă a RNA precum şi a altor circuite aferente acestora. Componenta software a mediului integrat propus utilizează în principal aplicaţii din mediile de programare şi simulare Matlab/Simulink, System Generator şi Xilinx ISE.

5. Elaborarea unei noi metode de implementare hibridă: hardware-software a RNA. Metoda propusă utilizează principii de proiectare clare şi unelte puternice de proiectare a circuitelor digitale. Ea permite proiectarea RNA folosind mediul Matlab/Simulink, atât pentru implementarea software a fazei de antrenare, cât şi pentru implementarea hardware a fazei de propagare. Modelarea software, antrenarea şi simularea RNA se face prin procedeele cunoscute folosind uneltele NNToolbox. Pentru o modelare hardware eficientă se utilizează biblioteca RNA Blockset propusă de autor care include blocuri de reţele neuronale direct implementabile. Transferul parametrilor reţelei (număr de neuroni pe strat, ponderi, etc.) de la modelul software la cel hardware se face automat, în mediul nou elaborat. Cele două modele, software şi respectiv hardware pot fi co-simulate pentru verificarea corectitudinii modelului hardware. Metoda propusa în teză permite conversia automată şi exactă a modelului de nivel înalt din mediul Simulink într-un fişier de configurare a circuitului FPGA.

6. Pentru a exemplifica valabilitatea şi eficienţa noilor unelte de proiectare propuse în teză, autorul a implementat câteva tipuri de RNA mai larg utilizate. Astfel, sunt implementate reţele neuronale feed-forward cu paralelism de neuron având unul sau două straturi. Modelele elaborate folosesc biblioteca RNA Blockset, dezvoltată de autorul tezei şi seturi de ponderi obţinute în urma antrenării cu algoritmul de învăţare Hebbian supervizat şi respectiv algoritmul Levenberg-Marquardt. A fost implementată, de asemenea, o RNA cu paralelism de strat, antrenată cu algoritmul Hebbian care utilizează resurse

hardware foarte mici. Validitatea modelelor nou elaborate a fost verificată cu mai multe seturi de vectori de test.

7. Contribuţii au fost aduse şi în ceea ce priveşte optimizarea modelelor de reţele neuronale de tipul FF, din punctul de vedere a resurselor consumate şi a frecvenţei de lucru. S-a făcut un studiu pentru determinarea numărului optim de biţi pe care trebuie să lucreze blocurile componente ale modelului hardware a RNA şi s-au determinat relaţii de calcul pentru determinarea numărului de biţi necesari reprezentării părţii întregi.

8. O problemă abordată în teză şi în legătură cu care au fost aduse contribuţii proprii, o constituie studiul privind influenţa numărului de biţi utilizaţi la reprezentarea ponderilor asupra erorilor modelelor software şi hardware. Analiza arată că în faza de propagare nu este nevoie de precizie ridicată, 11-12 biţi fiind în general suficienţi pentru obţinerea unor rezultate identice sau uneori chiar mai bune decât cele ale modelului software cu ponderile reprezentate pe 64 biţi în virgulă flotantă. Aceste concluzii au fost stabilite prin simularea comportamentului modelului hardware pentru ponderi reprezentate pe un număr variabil de biţi şi determinarea erorilor corespunzătoare. Cercetările au fost efectuate utilizând pentru simularea şi afişarea erorilor reţelei o aplicaţie - cod Matlab – elaborată de autor.

9. O contribuţie aplicativă majoră o constituie implementarea, utilizând metodologia propusă în teză, a unor RNA competitive simple cu paralelism de neuron respectiv cu paralelism de strat. Modelele create folosesc noile blocuri adăugate bibliotecii RNA Blockset, şi anume blocul de calcul al distanţei euclidiene (simplificată) şi blocul funcţiei de activare competitive. RNA implementată clasifică corect toţi vectorii de antrenare şi două seturi diferite de vectori de test. Resursele necesare implementării unei RNA cu paralelism de strat având 15 neuroni în stratul de ieşire sunt mici (68 de slice-uri) ceea ce permite implementarea pe acelaşi chip a unor reţele de mari dimensiuni. Frecvenţa maximă cu care se pot aplica vectorii ce se doresc a fi clasificaţi este cu ceva mai mare de 1 MHz. RNA implementată cu paralelism de neuron permite creşterea frecvenţei de lucru până la 15 MHz, în condiţiile în care resursele utilizate cresc de aproximativ 4 ori (271 slice-uri). Astfel, într-un circuit de tipul celui utilizat în experimente se pot implementa aproximativ 181 neuroni competitivi.

10. Pentru a valida soluţiile propuse în teză, autorul a realizat un sistem de recunoaştere a gesturilor statice ale unei mâini. Soluţia aleasă pentru sistemul senzorial al mâinii constă dintr-o mănuşă senzorială care furnizează date despre gradul de flexare a degetelor şi respectiv poziţia mâinii. Dintre metodele de recunoaştere cunoscute s-a optat pentru soluţia care utilizează reţelele neuronale, deoarece oferă cele mai bune rezultate sub aspectul preciziei în funcţionare. Sistemul de recunoaştere realizat utilizează platforma hardware-software descrisă în paragraful 7.3.

11. Pentru realizarea sistemului de recunoaştere menţionat mai sus, se propune în teză un model al unui sistem pentru recunoaşterea gesturilor statice ale unei

mâini structurat pe două niveluri. Primul nivel realizat cu o reţea FF antrenată cu algoritmul Hebbian supervizat, are rolul de preprocesare a datelor de intrare, iar cel de al doilea nivel, realizat cu o reţea competitivă simplă, are rolul de clasificare a datelor. Utilizarea unei RNA în preprocesarea datelor oferă flexibilitate în ceea ce priveşte implementarea unei metode de preprocesare. Utilizarea acestei combinaţii de RNA pentru recunoaşterea gesturilor nu este semnalată în literatură şi conduce la rezultate foarte bune. Procentul de recunoaştere pentru setul de antrenament şi două seturi de test este de 100%.

12. Pe parcursul tezei au fost aduse, de asemenea, o serie de contribuţii direct aplicative:

    - Autorul a elaborat aplicaţia soft (cod C si respectiv cod Matlab) pentru sistemul de achiziţie a datelor. Datele achiziţionate servesc ca vectori de antrenare sau de test. Protocolul de comunicaţie implementat este un protocol paralel asincron cu confirmare şi condiţionare totală. Achiziţia este iniţiată din mediul Matlab, prin specificarea dimensiunilor vectorului de intrare şi a numărului de eşantioane dorite din fiecare vector. La terminarea achiziţiei vectorii şi parametrii acestora (dimensiune, număr eşantioane pe vector, număr de vectori) se salvează într-un fişier.

    - Autorul tezei a elaborat o bibliotecă cu blocuri parametrizabile specifice arhitecturii reţelelor neuronale, ca de exemplu: strat de intrare (memorie date intrare), memoria pentru ponderi, blocul de calcul al intrării nete (MAC, Negdist), diverse funcţii de activare, blocul de comandă, etc. Existenţa acestei biblioteci creşte eficienţa metodei de implementare nou propusă.

    - In fine, a fost implementată hardware RNA pentru recunoaşterea gesturilor folosind straturile neuronale create în capitolele V şi VI şi a fost testată practic funcţionalitatea acesteia. Gestul recunoscut de sistem este afişat în timp real pe un afişaj cu 7 segmente.

8.2 Posibilităţi de continuare a cercetării

Cercetarea poate fi continuată în majoritatea domeniilor studiate de prezenta teză, după cum urmează:

- *Extinderea numărului şi a tipului senzorilor utilizaţi*, pentru adăugarea de noi simţuri mâinii artificiale, cum ar fi simţul tactil. Un sistem senzorial mai performant se poate realiza folosind o mănuşă 5DT-16 (care dispune de 16 senzori) şi care poate fi echipată cu senzori de forţă (FSR), senzori de temperatură, etc.

- *Metoda integrată hardware-software.* Extinderea facilităţilor metodei de implementare de RNA elaborată în această teză, prin analizarea automată a ponderilor determinate în faza de antrenare. Această analiză ar permite

determinarea numărului optim de biţi ai ponderilor, multiplicatorului, acumulatorului, etc.

- *Extinderea Bibliotecii RNA Blockset* prin adăugarea de noi blocuri şi straturi de reţele implementate cu un element de procesare pe strat. Este de dorit crearea unei biblioteci care să conţină majoritatea blocurilor bibliotecii Neural Network Blockset, pentru a putea crea modelul hardware al oricărei reţele realizate cu NNToolbox. Adăugarea la bibliotecă de reţele neuronale, cu un strat, implementate cu paralelism de neuron permite implementarea hardware foarte facilă a unor reţele simple cu unul sau două straturi.

- *Implementarea de noi tipuri de RNA*. Folosind noile blocuri din RNA blockset este de dorit să se implementeze şi alte tipuri de reţele ca de exemplu: RNA Kohonen, RNA bazate pe funcţii radiale, etc.

- *Implementarea fazei de antrenare în circuitul FPGA.* Pentru realizarea unui sistem complet independent de calculator ar fi necesară implementarea şi a fazei de antrenare în FPGA. Această facilitate necesită crearea de noi blocuri de calcul. Pentru a evita ocuparea resurselor FPGA cu circuite suplimentare, care se folosesc rar, se poate utiliza proprietatea de reconfigurabilitate a circuitelor FPGA. Astfel circuitul se poate reconfigura în funcţie de faza care se doreşte a fi implementată (antrenare sau propagare).

- *Extinderea numărului de gesturi statice* pe care le recunoaşte sistemul pentru implementarea unui set de gesturi care ar putea ajuta o persoană cu dizabilităţi de vorbire să comunice prin intermediul unui sintetizator de voce.

- *Recunoaşterea gesturilor dinamice* este o problemă a cărei rezolvare ar reprezenta un pas important în realizarea unei mâini artificiale. Viteza de procesare de care este capabil sistemul prezentat pare să fie suficientă şi pentru recunoaşterea gesturilor dinamice.

BIBLIOGRAFIE

[1] A. Buchman, **S. Oniga**, MATLAB Interfaced ADuC 812 Micro Converter Based Data Acquisition System, The 10th International Symposium for Design and Technology of Electronic Packages Conference Proceedings, Bucharest, 23-26 sept. 2004, pp. 175.

[2] A. Buchman, **S. Oniga**, Development Framework for Hardware Implementation of Digital Neural Networks Used in Smart Sensors, Scientific Bulletin of the Politehnica University of Timişoara, Tomul 49(63), Fascicola 1, 22-23 octombrie 2004, pp..232.

[3] A. Buchman, **S. Oniga**. Frequency Domain Analysis of an Ultrasonic Emitter - Receiver System. International Symposium for Design and Technology of Electronic Packages - 11th Edition, Cluj-Napoca, Romania. 2005.

[4] A. Bernatzki, W. Eppler, H. Gemmeke. Interpretation of Neural Networks for Classification Tasks. Proceedings of EUFIT 1996, Aachen, Germany, http:. fuzzy.fzk.de eppler postscript eufit.ps 2005.

[5] A. Cechin, U. Epperlein, W. Rosenstiel, B. Koppenhoefer. The Extraction of Sugeno Fuzzy Rules from Neural Networks. ESANN '96, Bruges, 1996.

[6] A. Ignea. Măsurarea electrică a mărimilor neelectrice. Ed. de Vest, Timişoara, 1996.

[7] A. Krukowski, I Kale. Simulink/Matlab-to-VHDL Route for Full-Custom/FPGA Rapid Prototyping of DSP Algorithms. Matlab DSP Conference DSP'99, Tampere, Finland, 16-17 November 1999.

[8] A. Malinowski, T. J. Cholewo, J. M. Zurada. Capabilities and Limitations of Feedforward Neural Networks with Multilevel Neurons. Proceedings of the IEEE International Symposium on Circuits and Systems, volume 1, pages 131-134, Seattle, Wash., USA, April 1995.

[9] A. Mulder. Human Movement Tracking Technology. Technical Report, NSERC Hand Centered Studies of Human Movement project, Burnaby, B.C., Canada: Simon Fraser University. ftp: fas.sfu.ca: pub cs graphics vmi HMTT.pub.ps.Z.

[10] A. Perez-Uribe, E. Sanchez. FPGA Implementation of an Adaptable-Size Neural Network. Proceedings of the VI International Conference on Artificial Neural Networks - ICANN 96, Bochum, Germany, July 16-19, 1996.

[11] A. Perez-Uribe, E. Sanchez. A Digital Artifcial Brain Architecture for Mobile Autonomous Robots. Proc. IV Intl. Symposium on Artificial Life and Robotics AROB'99, Oita, Japan, Jan., 1999, pp. 240-243.

[12] A. Perez-Uribe, E. Sanchez. FPGA Implementation of a Network of Neuronlike Adaptive Elements. Proceedings of the 7th International Conference on Artificial Neural Networks - ICANN '97, pp. 1247-1252, Lausanne, Switzerland, October 8-10, 1997.

[13] A. Pérez-Uribe. Structure-Adaptable Digital Neural Networks, PhD Thesis Nr. 2052, Swiss Federal Institute of Technology-Lausanne, Switzerland, 1999.

[14] A. Perez-Uribe, E. Sanchez. Structure-Adaptable Neurocontrollers: A Hardware-Friendly Approach. Proceedings of the International Work-Conference on Artificial and Natural Neural Networks IWANN97, Lanzarote, Canary Islands, Spain. 1997. pp 1251-1259.

[15] A. P. Singh. Design & Implementation of Neural Hardware. University School of Information Technology, GGS Indraprastha University, Delhi http://www.geocities.com/aps_ipu/papers/synopsis.pdf, 2005.

[16] A. Sandberg, Gesture Recognition using Neural Networks, Master thesis TRITA-NA-E9727, 1997.

[17] A. P. Paplinski. Self-Organizing Feature Maps. NNets — L. 10, June 26, 2000. http://www.csse.monash.edu.au/courseware/cse5301/04/Lnts/L10.pdf, 2005 .

[18] A. Savran, S. Unsal. Hardware Implementation of a Feedforward Neural Network using FPGAs. International Conference on Electrical and Electronics Engineering. Bursa, December 2003.

[19] A. Sareen. Reconfigurable Design for Pattern Recognition Using Field Programmable Gate Arrays. Master of Science thesis, College of Engineering and Technology Ohio University, August, 1999.

[20] A. K. Sharma. Programable Logic Handbook: PLD, CPLDS and FPGAs. McGraw-Hill Handbooks, 1998.

[21] A. S. Micilotta. The Tracking of Hands for the Development of a Gesture Recognition Algorithm. BSc. Thesys in Electrical and Electronic Engineering at the UNIVERSITY OF CAPE TOWN 18 October 2000 University of Cape Town 2000

[22] A. V. Upasani, C. Kapoor, D. Tesar. Survey of Available Sensor Technology for Robotic Hands. Preoceedingds of DETC99: 1999 ASME Design Engineering Technical Conferences, September 12-15, 1999, Las Vegas, Nevada.

[23] B. Krose, P. van der Smagt. An Introduction to Neural Networks. Eighth edition, November 1996. University of Amsterdam.

[24] B. D. Ripley. Pattern Recognition via Neural Networks, Cambridge: Cambridge University Press. ISBN 0-521-46086-7.

[25] B. K. Bharkhada. Efficient FPGA Implementation of a Generic Function Approximator and its Application to Neural Net Computation. M. S. Thesis. Electrical and Computer Engineering and Computer Science of the College of Engineering University Of Cincinnati. August 06, 2003.

[26] C. Amerijckx, M. Verleysen, P. Thissen, Jean-Didier Legat. Image Compression by Self-Organized Kohonen Map. IEEE Transactions on Neural Networks, Vol. 9, No. 3, May 1998.

[27] Chin-Shyurng Fahn and H. Sun. Development of a Sensory Data Glove Using Neural-Network-Based Calibration. ICAT 2000 Conference, October 27, 2000.

[28] C. Jutten und J. Herault. Blind separation of sources, Part I: An adaptive algorithm based on neuromimetic architecture. Signal Processing, Elsevier, 24:1-10, 1991.

[29] C. De Stefano, C. Sansone, M. Vento. Evaluating Competitive Learning Strategies for Handwritten Character Recognition, Proceedings of 1994 IEEE International Conference on Systems Man and Cybernetics, San Antonio (USA), October 2-5, pp. 759-764, 1994.

[30] C. Laschi, P. Dario, et. al. Grasping and Manipulation in Humanoid Robotics. Scuola Superiore Sant'Anna, Pisa, Italy, Syneragh Project (contract #BE-4505) of the EU community.

[31] C. Lee, D. Yeung, M. Mah, J. Kapasi. Sign Language Glove: American Sign Language Translation Device. EE 552 The S.L.G. Project Final Documentation 2003-04-01.

[32] C. S. Lindsey. Neural Networks in Hardware: Architectures, Products and Applications. Hardware Neural Network Course. Royal Institute of Technology Stockholm, Sweden. 1998. www.particle.kth.se ~lindsey.

[33] C. S. Lovchik. The Design and Development of a Space Compatible Dextrous Robotic Hand with Capabilities Comparable to those of an Astronaut's Hand, NASA Intern Presentation Slides. Automation Robotics and Simulation Division. NASA.

[34] C. S. Lovchik, M.A. Diftler. The Robonaut Hand: A Dexterous Robot Hand for Space. Proceeding of the 1999 IEEE International Conference on Robotics & Automation, Detroit, Michigan, May 1999, pp. 907-912.

[35] C. S. Rai, A. P. Singh. A Review of Implementation Techniques for Artificial Neural Networks, Proceeding of National Conference on Research & Practices in Current Areas of IT, S.H.S. Longowal Central Institute of Engineering & Technology, Longowal, Punjab, India, March 26-27, 2004, pp.178-182.

[36] D. Abramson, K. Smith, P. Logothetis, D. Duke. FPGA Based Implementation of a Hopfield Neural Network for Solving Constraint Satisfaction Problems, Workshop on Computational Intelligence of the 24th Euromicro Conference, Västerås, Sweden, August 25th-27th, 1998.

[37] D. F. Wolf, R. A. F. Romero, E. Marques. Using Embedded Processors in Hardware Models of Artificial Neural Networks. In proceedings of SBAI - Simpósio Brasileiro de Automao Inteligente. 2001. pp 78-83.

[38] D. Gonek, G. Barreiro, A. Ling, S. Chadha, T. Li, R. Orsten. EE552 Project Final Report. Hardware Implementation of a Neural Network Trainer and Associated Neural Network. March 21, 2002.

[39] D. Hammerstrom. A Digital VLSI Architecture for Real-World Applications, An Introduction to Neural and Electronic Networks. S. F. Zornetzer, J. L. Davis, C. Lau and T. McKenna, San Diego, CA, Academic Press, pp.335-358, 1995.

[40] D. Hammerstrom. Implementation of Neuro-Fuzzy Algorithms Using Field Programmable Gate Arrays (FPGAs). Lecture Notes, ECE Department OGI School of Science and Engineering Oregon Health and Science University.

[41] D. Hammerstrom, C. Gao, S. Zhu, Mike Butts. FPGA Implementation of Very large Associative Memories. ECE Department OGI School of Science and Engineering Oregon Health and Science University. 2003.

[42] D. Hammerstrom. Neural networks at Work. IEEE Spectrum July 1993. pp. 26-32.

[43] D. Hammerstrom. Working with neural networks. IEEE Spectrum July 1993. pp. 45-53.

[44] D. DeSieno. Adding a conscience to competitive learning. IEEE International Conference on Neural Networks, IEEE Press, New York, 1988, vol. 1, pp 117-124.

[45] D. J Edell. A Peripheral Nerve Information Transducer for Amputees: Long-Term Multichannel Recordings from Rabbit Peripheral Nerves. IEEE Transactions on Biomedical Engineering, Vol. BME-33, No. 2, pp. 203 - 213, 1986.

[46] D. Mic, **S. Oniga.** A Study Regarding the Implementation with VHDL of a Simulated Memory Used for Testing a Microprocessor, The 6th International Symposium for Design and Technology for Electronic Modules - SIITME '00, Bucharest, Romania, 21-24 Sept. 2000, pp. 132-135.

[47] D. Mic, E. Micu, **S. Oniga**, C. Gavrincea. The FPGA Implementation of a Digital Controller as a Digital Filter, Buletinul Ştinţific al Universităţii "Politehnica" din Timişoara, Transaction on Electronics and Telecomunications, Tom 49(63), Fascicola 1, 2004.

[48] D. Mic, **S. Oniga**. Proiectare asistată cu circuite logice programabil. Risoprint Cluj-Napoca, 2002.

[49] D. Sherman. Measure resistance and capacitance without an A/D. Philips Semiconductors Microcontroller Products Application Note, AN449, December 1993.

[50] D. Johnston et al. A Full Tactile Sensing Suite for Dextrous Robot Hands and Use in Contact Force Control. Proceedings of the 1996 IEEE Conference on Robotics and Automation, Minneapolis, Minnesota, April 1996, pp. 3222-3227.

[51] D. J. Sturman. Whole-hand Input. PhD thesis, Massachusetts Institute of Technology, February 1992. ftp://media.mit.edu/pub/sturman/WholeHandInput/thesis.pix.ps.Z

[52] Dr. M. Turhan Taner. Kohonen's Self Organizing Networks with "Conscience". Rock Solid Images, November 1997.

[53] E. Fiesler, R. Beale. Hanndbook of Neural Computation. Institute of Physics and Oxford University Press, Oxford 1997.
[54] E. Karalis. Digital Design principles and Computer Arhitecture. Prentice Hall, 1999.
[55] E. Sánchez-Nielsen, L. Antón-Canalís, M Hernández-Tejera. Hand Gesture Recognition for Human-Machine Interaction. Journal of WSCG, Vol.12, No.1-3, ISSN 1213-6972 WSCG'2004, February 2-6, 2003, Plzen, Czech Republic.
[56] E. V. Simões, L. F. Uebel, D. A. Barone. Fast Prototyping of Artificial Neural Network: GSN Digital Implementation. Universidade Federal do Rio Grande do Sul. Porto Alegre, RS – Brazil, http://electronica.com.mx/neural/articulos/fast_desig.pdf
[57] F. Aghili and M. Buehler J. M. Hollerbach. A Joint Torque Sensor for Robots. Proc. IEEE Int. Conf. Robotics and Automation, pp. 2865-2870, April 1997.
[58] F. Vahid. Embedded System Design: A Unified Hardware Software Introduction. Editura Wiley 2002
[59] G. A. Bakey et al. Control Architecture for the Belgrade/USC Hand, Dextrous Robot Hands, Springer-Verlag New York Inc., 1990, pp. 136-149.
[60] G. C. Burdea. Haptic Feedback for Virtual Reality. Virtual Reality and Prototyping Workshop, June 1999, Laval (France).
[61] G. C. Burdea, D. Gomez, N. Langrana, E. Roskos, P. Richard. Virtual Reality Graphics Simulation with Force Feedback. International Journal in Computer Simulation 5, pp. 287-303, 1995.
[62] G. Fang, W. Gao, X. Chen, C. Wang, J. Ma. Signer-independent Continuous Sign Language Recognition Based on SRN/HMM. Proceeding of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition (FG'02), Washington, USA, 2002, pp.312-317
[63] G. Goodhue. Determining baud rates for 8051 UARTs and other UART issues. Philips Semiconductors Microcontroller Products Application Note, June 1993.
[64] G. Hirzinger. Status Report 1997, Institute of Robotics and System Dynamics, DLR German Aerospace Research Establishment, 1997.
[65] G. Hirzinger. Multisensory Shared Autonomy and tele-sensor programming – Key issues in space robotics. Robotics and Autonomous Systems 11, 1993, pp. 141-162.
[66] G. Hirzinger, J. Butterfass, S. Knoch, H. Liu. DLR'S Multisensory Articulated Hand. Preprints Fifth Int. Symposium on Experimental Robotics, Barcelona, Spain, June 1997, pp. 28-34.
[67] G. Hirzinger et al. A Mechatronics Approach to the design of light-weight arms and multifingered hands. Proceeding of the 2000 IEEE International Conference on Robotics & Automation. San Francisco, CA, April 24-28, 2000, pp.46-54.
[68] Gh. Trémiolles, P. Tannhof, B. Plougonven, C. Demarigny, K. Madani. Visual Probe Mark Inspection, using Hardware Implementation of Artificial Neural Networks, in VLSI Production. In Proceedings IWANN'97, International Work-Conference on Artificial and Natural Neural Networks Lanzarote - Canary Islands, Spain - June 4-6, 1997, Springer Verlag, p 1374-1383.
[69] G. T. A. Kovacs, C.W. Storment, and J.M. Rosen. Regeneration Microelectrode Array for Peripheral Nerve Recording and Stimulation. IEEE Transactions on Biomedical Engineering, Vol. 39, No. 9, 1992.
[70] H. Demuth, M. Beale. Neural Network Toolbox For Use with MATLAB. User's Guide Version 4. The MathWorks, Inc. www.mathworks.com
[71] H. F. Restrepo, R. Hoffmann, A. Perez-Uribe,Ch. Teuscher, E. Sanchez. A Networked FPGA-Based Hardware Implementation of a Neural Network Application. 2000 IEEE

Symposium on Field-Programmable Custom Computing Machines: Proceedings: April 17-19, 2000 Napa Valley, California.

[72] H. Liu, J. Butterfass, S. Knoch, P. Meusel, G. Hirzinger. A New Control Strategy for DLR's Multisensory Articulated Hand. Conference on Robotics & Automation, Leuven, Belgium, May 1998.

[73] H. Liu, P. Meusel, J. Butterfass, G. Hirzinger, DLR's multisensory articulated Hand Part II: The Parallel Torque/Position Control System. Proceedings of the 1998 IEEE International Conference on Robotics & Automation, Leuven, Belgium, May 1998, pp. 2087 - 2093.

[74] H. Liu, P. Meusel, and G. Hirzinger. A tactile sensing system for the DLR Three-Finger Robot Hand. Proceedings of the ISMCR'95, Slovakia, 1995, pp. 91–96.

[75] H. Ossoinig, E. Reisinger, Ch. Steger, R. Weiss. Design and FPGA-Implementation of a Neural Network. Proceedings of the 7th International Conference on Signal Processing Applications & Technology. Boston, USA, October 1996. pp. 939-943.

[76] H. R. Taylor. Data Acquisition for Sensors System. Chapman&Hall, 1997.

[77] H. Shimodaira, J. Rokui, M. Nakai. Modifed Minimum Classifcation Error Learning and Its Application to Neural Networks. School of Information Science Japan Advanced Institute of Science and Technology Tatsunokuchi, Ishikawa, JAPAN. http: www-ks.jaist.ac.jp index.html

[78] Interlink Electronics, Force Sensing Resistor Integration Guide & Evaluation parts catalog. URL: http: www.interlinkelec.com.

[79] I. Petra, D. J. Holding, K. J. Blow, B. Tam, X. Ma, P. N. Brett. The design of a flexible digit towards wireless tactile sense feedback.pdf. Manuscript received March 12th 2004. This work was supported under ESPRC GR/R35520/01. Aston University, Aston Triangle, Birmingham B4 7ET, U.K.

[80] J. B. Burr. Digital Neural Network Implementations. In Neural Networks, Concepts, Applications, and Implementations. Vol III. Englewood Cliffs, New Jersey, Prentice Hall, 1991.

[81] J. B. Burr. Digital Neurochip Design, Chapter 8 of Parallel Digital Implementations of Neural Networks. Englewood Cliffs, New Jersey, Prentice Hall, 1993.

[82] J. B. Burr. Digital Neural Network Implementations. January 2, 1995, http: bwrc.eecs.berkeley.edu People kcamera neural papers burr95digital.pdf

[83] J. Butterfass, G. Hirzinger, S. Knoch, H. Liu. DLR's Multisensory Articulated Hand Part I: Hard- and Software Architecture. Proceedings of the 1998 IEEE International Conference on Robotics & Automation, Leuven, Belgium, May 1998, pp. 2081 - 2086.

[84] J. Cloutier, E. Cosatto, S. Pigeon, F. R. Boyer, P. Y. Simard. VIP: An FPGA-based Processor for Image Processing and Neural Networks. Proceedings of the fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, Lausane, Suisse, Feb. 1996.

[85] J. Davis, M Shah. Recognizing Hand Gestures. ECCV-94, Stockholm, Sweden, May 2-6 1994.

[86] J. Eisenstein, S. Ghandeharizadeh, L. Huang, C Shahabi, G. Shanbhang, R. Zimmermann. Analysis of Clustering Techniques to Detect Hand Signs. Proceedings of the 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing. Hong Kong. May 2001.

[87] J. Eisenstein, S. Ghandeharizadeh, L. Golubchik, C Shahabi, D. Yan, R. Zimmermann. Device Independence and Extensibility in Gesture Recognition, IEEE Virtual Reality 2003, p. 207.

[88] J. Eisenstein, S. Ghandeharizadeh, L. Golubchik, C Shahabi, D. Yan, R. Zimmermann. Multi-Layer Gesture Recognition: An Experimental Evaluation, University of Southern California Department of Computer Science Technical Report: 02-766, http://people.csail.mit.edu/jacobe/papers/usc-tr.pdf, 2005.
[89] J. F. Wakerly. Digital Design Principles & Practices, third edn, Prentice Hall 2000
[90] J. Fraden. Handbook of Modern Sensors, Springer. Verlag, New York, 1996.
[91] J. G. Ganssle. The Art of Designing Embedded Systems. Newnes.
[92] J. Gunthorpe, D. O'Reilly, R. Lewis. Implementing a Neural Network on an FPGA. March 8, 2000. http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/2000_w/interfacing/NN_on_FPGA/NNappnotes.pdf, 2005.
[93] J. G. Eldredge. FPGA Density Enhancement of a Neural Network Through Run-Time Reconfiguration. M. S. Thesis. Department of Electrical and Computer Engineering Brigham Young University. May 1994.
[94] J. G. Eldredge and B. L. Hutchings. RRANN: A Hardware Implementation of the Backpropagation Algorithm Using Reconfigurable FPGAs. Proc. IEEE Int. Conf. on Neural Networks", Orlando, Florida, June 1994.
[95] J. G. Eldredge and B. L. Hutchings. RRANN: The Run-Time Reconfiguration Artificial Neural Network. Presented at IEEE Custom Integrated Circuits Conference San Diego, CA. May 1-4, 1994. pp. 77-80.
[96] J. G. Eldredge and B. L. Hutchings. RRANN: A Method for Enhancing the Functional Density of SRAM-Based FPGAs. Kluwer Academic Publishers, Boston.
[97] J. Jockusch. Exploration based on Neural Networks with Applications in Manipulator Control. Ph.D. Dissertation, University of Bielefeld, February 2000.
[98] J. J. Blake, LP. Maguire, B. Roche, TM. McGinnity, LJ. McDaid. The Implementation of Fuzzy Systems, Neural Networks and Fuzzy Neural Networks Using FPGAs. Faculty of Engineering, University of Ulster, Northern Ireland, http://citeseer.ist.psu.edu/62029.html, 2005.
[99] J. J. LaViola Jr. Whole-Hand and Speech Input in Virtual Environments. Msc. Thesys Department of Computer Science at Brown University Providence, Rhode Island December 1999.
[100] J. Kangas. On the Analysis of Pattern Sequences by Self-Organizing Maps. PhD Thesis. Helsinki University of Technology. Finland. 6 May 1994.
[101] J. L Noyes. Learning rules. Neural Network Training. Handbook of Neural Computation IOP Publishing Ltd and Oxford University Press. 1997
[102] J. L. Holt, T.E. Baker. Back propagation simulations using limited precision calculations, in Proceedings of International Joint Conference on Neural Networks. 1991. pp 121-126 vol. 2.
[103] J. L. Beuchat, J.O. Haenni, E. Sanchez. Hardware Reconfigurable Neural Networks, 5th Reconfigurable Architectures Workshop, International Parallel and Distributed Processing Symposium IPDPS 1998.
[104] J. M. M. Aróstegui. VLSI Architectures for Evolutive Neural Models. Teză de doctorat. Barcelona, December 1994.
[105] J. M. Hollerbach. Some Current Issues in Haptics Research.. Proc. IEEE Intl. Conf. Robotics and Automation, San Francisco, April 24-28, 2000, pp. 757-762.
[106] J. N. H. Heemskerk. Overview of neural hardware. Capitolul 3 în: Neurocomputers for Brain-Style Processing. Design, Implementation and Application. PhD thesis, Unit of Experimental and Theoretical Psychology Leiden University, The Netherlands. 1995. ftp.mrc-apu.cam.ac.uk/pub/nn/murre/neurhard.ps

[107] J. P. Oliver, et. al. Implementation of Adaptive Logic Networks on an FPGA board, Configurable Computing: Technology and Applications, John Schewel, Editor, Proceedeings of SPIE Vol. 3526, page numbers 264-273 (1998).
[108] J. Starzyk, Y. Guo. A Self-Organizing Learning Array and its Hardware-Software Co-Simulation. Proc. ECCTD, (Krakow, Poland, 2003).
[109] J. Starzyk, J. Pang. Evolvable Binary Artificial Neural Network for Data Classification. the 2000 Int. Conf. on Parallel and Distributed Processing Techniques and Apllications, (Las Vegas, NV, June 2000).
[110] J. Stader. Applying Neural Networks. Artificial Intelligence Applications Institute. University of Edimburg. UK. 1992. http: citeseer.ist.psu.edu stader92applying.html.
[111] J. Torresen. Two-Step Incremental Evolution of a Prosthetic Hand Controller Based on Digital Logic Gates. 4th International Conference on Evolvable Hardware (ICES2001), October 2001, Tokyo, Japan.
[112] J. Torresen, Sh.Tomita. Chapter 2. A Review of Implementation of Backpropagation Neural Networks. Chapter 2 in the book by N. Sundararajan and P. Saratchandran (editors): Parallel Architectures for Artificial Neural Networks, IEEE CS Press, 1998. ISBN 0-8186-8399-6.
[113] J. Torresen, Sh.Tomita. Chapter 7. Implementation of Backpropagation Neural Networks on Large Parallel Computers. Chapter 7 in the book by N. Sundararajan and P. Saratchandran (editors): Parallel Architectures for Artificial Neural Networks, IEEE CS Press, 1998. ISBN 0-8186-8399-6.
[114] J. Torresen, Sh.Tomita. Implementation of Backpropagation Neural Networks on Large Parallel Computers. IEEE Los Alamitos, 2001.
[115] J. U. Meyer, et. al. Perforated silicon dices with integrated nerve guidance channels for interfacing peripheral nerves.pdf. INTER project supported by the European Community under ESPRIT BR project #8897.
[116] J. Zhu, P. Sutton. FPGA Implementations of Neural Networks – a Survey of a Decade of Progress. Proceedings of 13th International Conference on Field Programmable Logic and Applications (FPL 2003), Lisbon, Sep 2003.
[117] J Wachs, U Kartoun, H Stern, Y. Edan. Real-Time Hand Gesture Telerobotic System Using Fuzzy C-Means Clustering. Proceedings of the 5th Biannual World Automation Congress, 2002.
[118] K. Anderson. A new growing algorithm for SOFM. 19 September, 1997
[119] K.A. Kwiat, W. H. Debany Jr. Modeling a Versatile FPGA for Prototyping Adaptive Systems. Sixth IEEE International Workshop on Rapid System Prototyping (RSP'95), June 07 - 09, 1995, Chapel Hill, North Carolina.
[120] K. Boehm, W. Broll, M. Sokolewicz. Dynamic Gesture Recognition Using Neural Networks; A Fundament for Advanced Interaction Construction. SPIE Conference Electronic Imaging Science & Technology, San Jose California, USA, Feb. 1994.
[121] K. B. Shimoga. Robot Grasp Synthesis Algorithms: A Survey. The International Journal of Robotic Research, Vol. 15, No. 3, June 1996, pp. 230-266.
[122] K. F. D. Alotaibi. A High Level Hardware Description Environment for FPGA-Based Image Processing Applications. PhD Thesis Faculty of Science, Queen's University of Belfast. May 1999.
[123] K. Parnell, N. Mehta. Programmable Logic Design Quick Start Handbook. Xilinx, Inc. 2003.
[124] K. Rajagopalan, P. Sutton. A Flexible Multiplication Unit for an FPGA Logic Block. Proceedings of 2001 IEEE International Symposium on Circuits and Systems vol IV (ISCAS 2001), 6-9 May, 2001, Sydney, Australia, p546-549.

[125] K. Symeonidis. Hand Gesture Recognition Using Neural Networks. Master of Science Thesis in Multimedia Signal Processing Communications School of Electronic and Electrical Engineering, August 23, 2000.http://www.ee.surrey.ac.uk/People/T.Windeatt/student_projects/klimis

[126] K. Salisbury, C. Ruoff. The design and control of a dexterous mechanical hand. Proceeding of the 1981 ASME Computer Conference, Minneapolis, MN 1981.

[127] L. Dipietro, A. M. Sabatini, P. Dario. Evaluation of an instrumented glove for hand-movement acquisition. Journal of Rehabilitation Research and Development Vol. 40, No. 2, March/April 2003 Pages 179–190.

[128] L. Eriksson, F. Sebelius, C. Balkenius. Neural Control of a Virtual Prosthesis. Perspective in Neural Computing, ICANN 98.

[129] L. Eriksson, F. Sebelius. Pattern recognition of nerve signals using artificial neural networks. M.Sc. thesis, Department of Solid State Physics, Lund University, 1996.

[130] L. Lavagno, B. Pino, L.M. Reyneri, A. Serra. A Simulink-based Approach to System Level Design and Architecture Selection. Proc. of the 26th EUROMICRO Conference (EUROMICRO 2000), Vol. I, pp. 76-83 ISBN, 0-7695-0780-8, Maastricht, the Netherlands, September 5-7, 2000.

[131] L. Mari, F Renga, L.M. Reyneri. A Simulink-based Hardware/Software Codesign Tool for Rapid Prototyping of Control Systems, research report Politecnico di Torino, 2004

[132] L.M. Reyneri. Implementation Issues of Neuro-Fuzzy Hardware: Going Towards HW/SW Codesign, IEEE Transactions on Neural Networks, January 2003. pp. 176-194.

[133] L.M. Reyneri, M. Chiaberge, L. Lavagno, B. Pino, E. Miranda. Simulink-Based HW/SW Codesign of Embedded Neuro-Fuzzy Systems. Int'l Journal on Neural Systems, June 2000.

[134] L. Montelius, N. Danielsen et. al., The Artificial Hand Project, Lund University, http://lucs.fil.lu.se/ArtHand

[135] L. Montelius, N. Danielsen et. al., Implantable nerve contacts for artificial prosthesis control, Lund University, http://lucs.fil.lu.se/ArtHand/implants.html

[136] L. Montelius, F. Sebelius, et. al. Pattern recognition of nerve signals using an artificial neural network. Proceedings of the 18th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 1996.

[137] L. Sekanina, V. Drabek. The Concept of Pseudo Evolvable Hardware. Department of Computer Science and Engineering Faculty of Electrical Engineering and Computer Science Brno University of Technology. http://www.fee.vutbr.cz/UIVT

[138] Mathworks, Matlab The Language of Technical Computing, Simulink for Model Based on System Level Design. www.mathworks.com

[139] M. A. Figueiredo. Implementation of a Probabilistic Neural Network for Multi-spectral Image Classification on an FPGA Based Custom Computing Machine. Vth Brazilian Symposium on Neural Networks, December 09 - 11, 1998 Belo Horizonte, MG, Brazil.

[140] M Balch. Complete Digital Design. McGraw-Hill. 2003.

[141] M. Bogdan. Intelligent Neural Interface – Signalprocessing of Nerve Signals using Artificial Neural Nets, INTER project supported by the European Community under ESPRIT BR project #8897.

[142] M. Bogdan. Artificial Neural Nets for Peripheral Nervous System - Remoted Limb Prostheses. NeuroNimes 94. pp 193-202.

[143] M. Bogdan, W. Rosenstiel. Classification of Nerve Signals using Kohonen's Self-Organizing Map.pdf. INTER project supported by the European Community under ESPRIT BR project #8897.

[144] M. Bogdan, W. Rosenstiel. Intelligent Neural Interface - Signalprocessing of Nerve Signals using Artificial Neural Nets.pdf. INTER project supported by the European Community under ESPRIT BR project #8897.
[145] M. Bogdan, A. Babanine, J. Kaniecki, W. Rosenstiel. Nerve Signal Processing using Artificial Neural Nets. Information Processing in Cells and Tissues, Liverpool, Seiten 55–68, 1995.
[146] M. Bouzit. Design, Implementation and Testing of a Data Glove with Force Feedback for Virtual and Real Objects Telemanipulation. PhD Thesis, University of Pierre et Marie Curie, november 29, 1996.
[147] M. Brooks. The DataGlove as a Man-Machine Interface for Robotics. The Second IARP Workshop on Medical and Healthcare Robotics, Newcastle upon Tyne, UK 5-7 September 1989.
[148] M. Green, C. D. Shaw. Virtual Reality and Highly Interactive Three Dimensional User Interfaces: A Technical Outline.pdf. Department of Computing Science University of Alberta Edmonton, Alberta, Canada.
[149] M. Gschwind, V. Salapura, D. Maurer. FPGA Rapid Prototyping of Application-Specific Processors. VHDL Forum in Europe, Toledo, Spain, May 1997.
[150] M. Gschwind, V. Salapura, O. Maischberger. A Generic Building Block for Hopfield Neural Networks with On-Chip Learning.pdf. Institut fur Technische Informatik Technische Universitat Wien.
[151] M. Gschwind, V. Salapura. A VHDL Design Methodology for FPGAs.pdf.
[152] M. J. Wirthlin. Improving Functional Density Through Run-Time Circuit Reconfiguration. PhD Thesis. Department of Electrical and Computer Engineering Brigham Young University. November 1997.
[153] M. M. Mano, C. R. Kime. Logic and Computer Design Fundamentals. Prentice Hall PTR, 1997.
[154] M. Matsumiya, H. Takemura, N. Yokoya. An Immersive Modeling System for 3D Free-form Design Using Implicit Surfaces. Proc. ACM Symposium on Virtual Reality Software and Technology 2000 (VRST2000).
[155] M. Ponca, C. Shauer. FPGA Implementation of a Spike-based Sound Localization System. Fifth International Conference on Artificial Neural Networks and Genetic Algorithms 22-25 April 2001, Prague, Czech Republic.
[156] M. Ponca, G. Scarbata. Implementation of Artificial Neurons Using Programmable Hardware. Synopsys User Group Conference - SNUG Europe 2001.
[157] M. P. T. Juvonen, J. G. F. Coutinho, J. L. Wang, B. L. Lo, W. Luk, O. Mencer and G. Z. Yang. Custom Hardware Architectures for Posture Analysis. Department of Computing, Imperial College London. IEEE International Conference on Field Programmable Technology (FPT), Singapore, Dec. 2005.
[158] M. R. Tremblay, M. R. Cutkosky. Estimating Friction Using Incipient Slip Sensing During a Manipulation Task. Proceedings of the 1993 IEEE International Conference on Robotics and Automation, May 2-6, Atlanta, GA, pp. 363-368.
[159] M. Rossmann, T. Jost. K. Goser, A. Bühlmeier, G. Manteuffel. Exponential Hebbian On-Line Learning Implemented in FPGAs. ICANN 96, pages 767--772. Springer Verlag.
[160] M. Skrbek. Fast Neural Network Implementation. Neural Network World, 9, No.5, 1999, p. 375-391.
[161] M. W. Gertz, P. K. Khosla. Onika: A Multilevel Human-Machine Interface for Real-Time Sensor-Based Systems. ASCE/SPACE 94: The 4th Int'l Conf. and Expo. on

Engineering, Construction., and Operations in Space," Feb. 26-Mar. 3, 1994, Albuquerque, New Mexico.

[162] N. Lall. New XtremeDSP Slices Deliver As Much As 10X More GMACs Per Dollar. Xcell Journal. Winter 2004.

[163] N. I. Badler, R. Bindiganavale, J. P. Granieri, S. Wei, X. Zhao. Posture Interpolation with Collision Avoidance. Center for Human Modeling and Simulation University of Pennsylvania Philadelphia. Proc. Computer Animation, pages 13--20, Los Alamitos, CA, 1994. IEEE Computer Society Press

[164] P. A. Harling. Gesture Input using Neural Networks. BSc Thesis, Department of Computer Science, University of York, 1993.

[165] P. A. Harling, A. D.N. Edwards. Hand Tension as a Gesture Segmentation Cue. Department of Computer Science. Philip A. Harling and Alistair D. N. Edwards, editors, Progress in Gestural Interaction: Proceedings of Gesture Workshop '96, pages 75--87, Springer, Berlin et al., 1997.

[166] P. Donachy. Design and Implementation of a High Level Image Processing Machine Using Reconfigurable Hardware. PhD Thesis Faculty of Science, Queen's University of Belfast. September 1996.

[167] Ph. Thissen, Jean-Didier Legat, Ch. Amerijckx, M. Verleysen. Image Compression by Self-Organized Kohonen Map. IEEE Transactions on Neural Networks, Vol. 9, No. 3, May 1998.

[168] P. K. Allen, A. T. Miller, P. Y. Oh, B. S. Leibowitz.Using Tactile and Visual Sensing with a Robotic Hand. Proc. of the 1997 IEEE Int. Conf. on Robotics and Automation, April 1997. http: citeseer.ist.psu.edu/article/allen97using.html

[169] P. Lysaght, J. Stockwood, J. Law, D. Girma. Artificial Neural Network Implementation on a Fine-Grained FPGA. In R. W. Hartenstein, M. Z. Servit (Eds.) Field-Programmable Logic, Springer Verlag pp. 421-431, Prague, Czech Republic, Sept. 1994

[170] P. Moerland, E. Fiesler. Neural Network Adaptations to Hardware Implementations. E. Fiesler and R. Beale, editors, Handbook of Neural Computation, E1.2:1-13. Institute of Physics Publishing and Oxford UniversityPublishing, New York. January 97.

[171] R. D. Turney, C. Dick, D. B. Parlour, J. Hwang. Modeling and Implementation of DSP FPGA Solutions. Xilinx Inc.

[172] R. Der, G. Balzuweit, M. Herrmann. Building Nonlinear Data Models with Self-Organizing Maps. Artificial Neural Networks - ICANN 96, pp. 821 – 826, Springer 1996.

[173] R. F. Molz, Paulo M. Engel, Fernando G. Moraes, Lionel Torres, Michel Robert. Codesign of Fully Parallel Neural Network for a Classification Problem. International Conference on Information Systems, Analysis and Synthesis, Orlando, USA, 2000.

[174] R. Frank. Understanding Smart Sensors, Artech House, 1996.

[175] R. H. Liang, M. Ouhyoung. A Sign Language recognition System Using Hidden Markov Model and Context Sensive Search, Proc. of ACM VRST'96, ACM International Symposium on Virtual Reality and Software Technology, pp. 59-66, Hong Kong, July 1996.

[176] R. H. Liang, M. Ouhyoung. Taiwanese Sign Language recognition System. http://www.cmlab.csie.ntu.edu.tw/~f1506028/

[177] R. Ohba. Intelligent Sensors Technology, John Wiley & Sons, 1992.

[178] R. Tessier, W. Burleson. Reconfigurable Computing for Digital Signal Processing: A Survey. Journal of VLSI Signal Processing 28, 7–27, 2001Kluwer Academic Publishers. Manufactured in The Netherlands.

[179] R. Watson. A Survey of Gesture Recognition Techniques. Technical Report TCD-CS-93-11. Department of Computer Science, Trinity College, Dublin 2, 1993. ftp: ftp.cs.ted.ie pub tet tech-reports reports.93 TCD-CS-93-11.ps.gz

[180] S. C. Jacobsen et al. Design of The Utah/MIT Dextrous Hand, Center for Engineering Design, University of Utah. Proceedings of the 1996 IEEE Conference on Robotics and Automation, Minneapolis, Minnesota, April 1996, pp. 1520-1532.

[181] S. Draghici. On the capabilities of neural networks using limited precision weights. Neural Networks, 2002. **15**: p. 395-414.

[182] S. Hollar, J. K. Perng, K. S. J. Pister. Wireless Static Hand Gesture Recognition with Accelerometers - The Acceleration Sensing Glove. Berkeley Sensor & Actuator Center University of California, Berkeley. www-bsac.eecs.berkeley.edu/archive/ users/hollar-seth/publications/journalpapforglove4.pdf

[183] Shinoda H., K. Matsumoto and S. Ando, "Tactile Sensing Based on Acoustic Resonance Tensor Cell," Proc. TRANSDUCERS '97, Vol. 1, pp. 129-132, 1997.

[184] SIDSA Semiconductor Design Solutions. CARMeN Core ARM emulation for Embedded SOC and Software Co-Development. www.sidsa.com. 2003.

[185] **S. Oniga**, A. Tisan, C. Gavrincea, D. Mic. Implementări digitale ale rețelelor neuronale artificiale. Simpozionul de Electronică şi Telecomunicații, ETc. 2002, Septembrie, 2002, Timişoara, România, pp. 43-47.

[186] **S. Oniga**, A New Method for FPGA Implementation of Artificial Neural Network Used in Smart Devices, International Computer Science Conference microCAD 2005, Miskolc, 7-8 March. 2005. Hungary.

[187] **S. Oniga**, A. Buchman, A new method for hardware implementation of Artificial Neural Networks Used in Smart Sensors, The 10th International Symposium for Design and Technology of Electronic Packages Conference Proceedings, Bucharest, 23-26 septembrie 2004, pp. 215.

[188] **S. Oniga**, A. Tisan, Sensor System for an Artificial Hand, International Multidisciplinary Conference, 5th edition,Baia Mare, 23-24 May 2003, Scientific Bulletin, serie C, Volume XVII, pp. 391-396.

[189] **S. Oniga**, Survey of Sensor Systems for Robotic Hands, International Computer Science Conference microCAD 2002, Miskolc, 7-8 March. 2002. Hungary.

[190] **S. Oniga**, D. Mic. Application possibilities of a development board with FPGA in signal processing, International Computer Science Conference microCAD 2001, Miskolc, 1-2 March. 2001, Hungary.

[191] **S. Oniga**, D. Mic. Some Aspects Regarding VHDL Description for Program Memory Used in Functional Simulation of Microcontroler, The 6th International Symposium for Design and Technology for Electronic Modules - SIITME '00, Bucharest, Romania, 21-24 Sept. 2000, pp. 135-139.

[192] **S. Oniga**, V. Tiponuț, A. Buchman, D. Mic, Adaptive Interface Based on FGA implemented Artificial Neural Network, Scientific Bulletin of the Politehnica University of Timişoara, Tomul 49(63), Fascicola 1, 22-23 octombrie 2004, pp.236.

[193] **S. Oniga**, Sistemul senzorial pentru o mîna artificiala. Configuratie hard si soft, Buletinul Ştiințific, Seria C, Volumul X, Fascicola Electronică, Electrotehnică, Automatizări, Simpozionul Ştiințific Național, Baia Mare, 8 - 9 Mai 2003, pp. 73 - 78.

[194] **S. Oniga**, A. Tisan, A. Buchman, D. Mic, C. Gavrincea, Sistemul senzorial experimental pentru o mână artificială, Buletinul Ştiințific, Seria C, Volumul X, Fascicola Electronică, Electrotehnică, Automatizări, Simpozionul Ştiințific Național, Baia Mare, 8 - 9 Mai 2003, pp. 79 - 84.

[195] S. Rüping, M. Porrmann, U. Rückert. A High Performance SOFM Hardware-System. Proc. of the International Work-Conference on Artificial and Natural Neural Networks IWANN'97, Lanzarote, Spain, June 1997, pp. 772 – 781.
[196] S. S. Fels. Glove-TalkII: Mapping Hand Gesture to Speech Using Neural Networks – An Approach to Build Adaptive Interfaces. PhD Thesis. Computer Science University of Toronto. 1994. http: psy.uq.oz.au michaeln/erg/abstracts/fels.glove.html
[197] S. S. Fels. Neural Networks for Computer-Human Interfaces: Glove-TalkII.Progress in Neural Information Processing Vol.2, pages 1299--1304. ICONIP Hong Kong, Springer, September 1996.
[198] S. S. Fels, G. Hinton. Glove-TalkII: A neural network interface which maps gestures to parallel formant speech synthesizer controls. IEEE Transactions on Neural Networks, 9 (1998), No. 1, 205-212
[199] S. S. Fels and G. Hinton. Glove-TalkII: Mapping hand gestures to speech using neural networks. In G. Tesauro et. al., editor, Advances in Neural Information Processing Systems, volume 7, Denver, 1995. The MIT Press.
[200] S. S. Fels and G. Hinton. Glove-Talk: A Neural Network Interface Between a Data-Glove and a Speech Synthesizer. IEEE Transaction on Neural Networks, 4:2-8, 1993.
[201] S. S. Fels and G. Hinton. Glove-Talk: An Adaptive Gesture-to-Format Interface. In Proceedings of CHI95 Human Factors in Computing Systems, 1995
[202] S Hauck. The Roles of FPGAs in Reprogrammable Systems. *Proceedings of the IEEE*, Vol. 86. No. 4, pp. 615-639, April, 1998.
[203] S. Lam. Implementing DSP Algorithms in FPGAs. Xcell Journal. Winter 2004.
[204] S. Soloman. Sensors Hanbook. McGraw-Hill Handbooks. 1998.
[205] T. Akin and K. Najafi. A Micromachined Silicon Sieve Electrode for Nerve Regeneration Applications. IEEE, 1991.
[206] T. Iberall, AH. Fagg. Neural Network Models for Selecting Hand Shapes. A.M. Wing, P. Haggard, and J.R. Flanagan (Eds) Hand and Brain: The Neurophysiology and Psychology of Hand Movements, NY: Academic Press.
[207] T. Kohonen. Self-organization and associative memory. Springer-Verlag, Berlin, 1984.
[208] T. Kohonen. Self-Organizing Maps. Third Edition. Springer-Verlag Berlin, 2001.
[209] T. Kuroda, Y. Tabata, A. Goto, H. Ikuta, M. Murakami. Consumer price data-glove for sign language recognition. Proc. 5th Intl Conf. Disability, Virtual Reality & Assoc. Tech., Oxford, UK, 2004 ICDVRAT/University of Reading, UK; ISBN 07 049 11 44 2
[210] T. Schoenauer, A. Jahnke, U. Roth and H. Klar. Digital Neurohardware: Principles and Perspectives. Neuronal Networks in Applications - NN'98 - Magdeburg 1998.
[211] T. S. Huang, V. I. Pavlovic. Hand Gesture Modeling, Analysis, and Synthesis. Proc. International Workshop on Automatic Face- and Gesture- Recognition", pp. 73-79, 1995. 31 http: citeseer.ist.psu.edu huang95hand.html.
[212] U. Meyer-Baese. Digital Signal Processing with Field Programmable Gate Arrays. Springer-Verlag Berlin Heidelberg. 2001.
[213] V. F. Cimpu. Hardware FPGA Implementation of a Neural Network. Proceedings International Conference in Technical Informatics, 1996.
[214] V. G. Popescu, G. C. Burdea, M. Bouzit, V.R Hentz. A Virtual Reality-based Telerehabilitation System with Force Feedback. Proceedings of MMR 7, IOS Press, Amsterdam, Vol. 62, pp. 261-267.
[215] V. G. Popescu, G. C. Burdea, M. Bouzit,Virtual Reality Simulation Modeling for a Haptic Glove.pdf. Department of Electrical and Computer Engineering Rutgers – The State University of New Jersey.

[216] V. Salapura, M. Gschwind, O. Maischberger. A Fast FPGA Implementation of a General Purpose Neuron. Proceedings of the Fourth International Workshop on Field Programmable Logic and Aplications, September 1994.
[217] V. Tiponuţ, C. D. Căleanu. Reţele neuronale. Arhitecturi şi algortmi. Editura Politehnica Timişoara 2002.
[218] V. Trifa, E. I. Gaura. Reţele Neuronale Artificiala. Editura Mediamira, 1996.
[219] Xilinx Inc. Xilinx System Generator for DSP Tutorials. Xilinx Development System. www.xilinx.com.
[220] Xilinx Inc. Nabeel Shirazi, Jonathan Ballagh, Put Hardware in the Loop with Xilinx System Generator for DSP, Fall 2003.
[221] Xilinx Inc. Spartan-IIE FPGA Family, Complete Data Sheet, DS077, July 2004.
[222] Xilinx Inc. Spartan-3 FPGA Family: Functional Description. DS099-2 (v1.3) August 24, 2004
[223] Xilinx Inc. Virtex-II Platform FPGAs: Complete Data Sheet. DS031 (v3.3) june 24, 2004.
[224] Xilinx Inc. Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet. DS083 (v4.2) March 1, 2005.
[225] Xilinx Inc. Virtex-4 Family Overview. DS112 (v1.2) December 8, 2004.
[226] Xilinx Inc. Synthesis and Simulation Design Guide. www.xilinx.com
[227] Xilinx Inc. Xilinx Design Reuse Methodology for ASIC and FPGA Designers. System-on-a-Chip Designs Reuse Solutions. www.xilinx.com
[228] X. Zeng, D. S. Yeung, Sensitivity Analysis of Multilayer Perceptron to Input and Weight Perturbations. IEEE Transactions on Neural Networks, Vol. 12, No. 6, November 2001.
[229] Yihua Liao. Neural Networks in Hardware: A Survey. Department of Computer Science, University of California, Davis.
[230] Y. Wu, T. S. Huang. View-independent Recognition of Hand Postures. In Proc. of IEEE Conf. on CVPR'2000, Vol.II, pp.88-94, Hilton Head Island, SC, 2000.
[231] Z. A. Ye, A. Moshovos, S. Hauck, P. Banerjee. CHIMAERA: A High–Performance Architecture with a Tightly–Coupled Reconfigurable Functional Unit. Proc. of ISCA (Symposium on Computer Architecture), June 2000.
[232] W. Craelius. The Artificial Hand. Rutgers' College of Engineering, April 1999, http://uc.rutgers.edu/medrel/newhand/.
[233] W. J. Tompkins, J. G. Webster, editors. Interfacing Sensors to the IBM PC. Prentice-Hall, Inc. 1998.
[234] W. Kadous. GRASP: Recognition of Australian Sign Language using Instrumented Gloves. Master's Thesis, University of new Soutf Wales, October 1995. http://www.cse.unsw.edu.au/~waleed/thesis.html
[235] W. Philipsen. Optimization with Potts Neural Networks in High Level Synthesis. Ph.D. Thesis Eindhoven University of Technology, Eindhoven, The Netherlands, 1995.
[236] W. Putnam, R. B. Knapp. Input/Data Acquisition System Design for Human Computer Interfacing. October 17, 1996.
[237] ***. Bourns Inc., URL: http://www.Bourns.com
[238] ***. EE563 Project Document: "Using A Neuroprocessor Model for Describing Artificial Neural Nets", http://www.compusmart.ab.ca.
[239] ***. AMP Incorporated, Valley Forge. Piezo Film Sensors Technical Manual, Dec. 1993.
[240] ***. EE563 Project Document: "Using A Neuroprocessor Model for Describing Artificial Neural Nets", http://www.compusmart.ab.ca

[241] ***. EE602 Project Document: "A Stack Processor: Synthesis",
[242] ***. EE635 Project Document: "A Writable Computer", http://www.compusmart.ab.ca
[243] ***. Fifth Dimension Technologies. 5DT Data Glove 5. User manual, version 2.00, February 2000.
[244] ***. Grasp Planning and Mechanics, The University of Texas at Austin, http://www.robotics.utexas.edu.
[245] ***. Grasping and master Gloves, http://www.cs.utah.edu/~jmh/UMDH.html. McGill University.
[246] ***. MicroStrain Inc., MicroStrain Product Catalog
[247] ***. http://www.compusmart.ab.ca.
[248] ***. Overview of the EU-Project GRIP (An Integrated System for the Neuroelectric Control of Grasp in Disabled Persons), http://www.grip-europe.org/.
[249] ***. Simulating a Neuroprocessor, http://www.compusmart.ab.ca.
[250] ***. VPL Research DataGlove Model 4. Operation Manual. 1993
[251] ***. ACM Transactions on Computer-Human Interaction, Vol. 12, No. 1, March 2005.
[252] ***. Primary User Input Interfaces. http://www.hitl.washington.edu/scivw/scivw-ftp publications IDA-pdf PRIMARY.PDF . pp.97-116.
[253] ***. Neural Network Hardware. http://www1.cern.ch/NeuralNets/nnwInHepHard.html
[254]***. Datasheet of SMT355. Sundance. http://www.sundance.com .
[255] ***. Datasheet of E5 and A7. Triscend. http://www.triscend.com.
[256] ***. Datasheet of Excalibur. Altera http://www.altera.com

ANEXE

Anexa 1. Codul Matlab pentru achiziţia datelor pe portul paralel

```
% Acest program permite achiziţia datelor de la portul paralel

% setarea portului paralel
% matport('Outport', 890, 252)  % setarea bitului 5 din portul de comanda al portului paralel pe 1
                                % pentru ca portul de date sa devină bidirecţional
% matport('Outport', 880, 255)  % setarea registrului de date al portului paralel
                                % pe "FF" pentru ca portul de intrare sa poate fi citit

% Setarea parametrilor achiziţiei
n=13                            % numărul de senzori = numărul de neuroni
s=50                            % numărul de eşantioane care se înregistrează pentru fiecare gest
g=0;                            % iniţializarea numărului de gesturi care se înregistrează

% delete(dio);
dio = digitalio('parallel','LPT1');
addline(dio,0:7,'in');          %index 1:8, pini 2:9
addline(dio,9:10,'in');         %index 9:10, pini 13:12
addline(dio,13:14,'out');       %index 11, 12 pini 1,14
a=zeros(s*g,n);                 %loc pentru s linii de cate n octeţi
reply = 'Y';
while   reply == 'Y';
putvalue(dio.line(12),0);
putvalue(dio.line(12),1);
ack = 0;
putvalue(dio.line(11),ack);
  for i=1:s,
    for j=1:n,
      while getvalue(dio.line(9))==1; %aştept rdy
      end
      b=getvalue(dio.line(1:8));
      k=g*s+i;
      a(k,j)=binvec2dec(b);
      ack = 1;
      putvalue(dio.line(11),ack);
      while getvalue(dio.line(9))==0; %astept rdy
      end
      ack = 0;
      putvalue(dio.line(11),ack);
    end
  end
  reply = input('Do you want more? Y/N [Y]: ','s');
    if isempty(reply)
     reply = 'Y';
    end
  g=g+1;
end
a;
```

Anexa 2. Programul de achiziţie şi de transfer al datelor

```
//receptie date manusa , transmisie date la PC

#pragma DEBUG OBJECTEXTEND CODE
#include <stdio.h>
#include <ADuC812.h>

//_____________________________________________________________________
// SUBRUTINA RECEPTIE CARACTER
char getch(void)
        {
        char ch; // int i=0;
        while (!RI);
        ch = SBUF;
        RI = 0;
        return (ch);
        }
//_____________________________________________________________________
// PROGRAMUL PRINCIPAL

void main (void)
        {
         char start='C', reset='A', date[9], u=0, canal_curent, conv_val[6];
         int i,j;
        T0=0;
//CONFIGURARE PORT SERIAL
        PCON = 0X80 ;                           // 2*9600
        SCON = 0x50 ;                           // 8bit, fara paritate, 1stopbit
        TMOD = 0x20 ;                           // configurare Timer1..
        TH1 = 0xFD ;                            // ..pentru 9600baud..
        TR1 = 1 ;                               //start TIMER1
        TI = 1;
//CONFIGURARE INT0 - ACTIV PE NIVEL
//      TCON.0=1;
//CONFIGURARE ADC
        ADCCON1 = 0x50 ;                        //configurare ADC mod normal
                                                //clk / 8, t de ach.= 4clk,start prin soft

//PROGRAMARE T2 PENTRU 7,5ms
                        RCAP2L = 0x00 ;         // perioada temporizare =
                        RCAP2H = 0xE5 ;         //  = 2*(10000h-E500h)*1.085us
                        TL2 = 0x00 ;            //  = 7,5ms
                        TH2 = 0xE5 ;

//ACHIZITIE DE DATE
        printf ("%c",reset) ;                   // reset manusa

        while (u != 0x55)                       //astept confirmare reset
```

```
                {
                printf ("%c",reset) ;
                u = getch();
                }

        printf ("%c",start) ;                   // cer date de la manusa
        u = getch();                            //sterg SBUF

        for (i=0; i<9; i++)                     //citesc in gol 18 date
                {u=getch();}                    //pentru revenire dupa reset
        printf ("%c",start) ;
        for (i=0; i<9; i++)
                {u=getch();}

                        T1=1;                   // ridic rdy=1;
start1: //while (!IE0);
                        T2CON = 0x04 ;          //start temporizare 7,5 ms

                T0=!T0;
        printf ("%c",start) ;
        for (i=0; i<9; i++)                     // citesc 9 date manusa
                {
                date[i]=getch();
                }

                        while (!TF2);
                        TF2=0;


//TRANSMIT DATE PARALEL
        for (i=1; i<7; i++)                     // transmit 7 date manusa
                {
                        P2=date[i];             //pun date
                        RCAP2L = 0x00 ;         // perioada de transmisie =
                        RCAP2H = 0xF7 ;         //  = 2*(10000h-EE00h)*1.085us
                        TL2 = 0x00 ;            //  = 2,5ms
                        TH2 = 0xF7 ;
                        T2CON = 0x04 ;          //start temporizare 2,5 ms

                        T1=0;                   // pun rdy=0;
                        while (WR==1);          // astept ack

                        //ASTEPT 2,5ms

                        while (!TF2);
                        TF2=0;

                        T1=1;                   // ridic rdy=1;
                        while (WR==0);
                        T0=!T0;
```

```
                    //ASTEPT 7,5ms
                    RCAP2L = 0x00 ;          // perioada de pauza = per de achizitie
                    RCAP2H = 0xE5 ;          //  = 2*(10000h-E500h)*1.085us
                    TL2 = 0x00 ;          //  = 7,5ms
                    TH2 = 0xE5 ;
                    T2CON = 0x04 ;
                    while (!TF2);
                    TF2=0;
                }
// trimit ultimul esantion de la manusa:
                    P2=date[7];                              //pun ultimul esantion
                    RCAP2L = 0x00 ;          // perioada transmisie =
                    RCAP2H = 0xF7 ;          //  = 2*(10000h-EE00h)*1.085us
                    TL2 = 0x00 ;          //  = 2,5ms
                    TH2 = 0xF7 ;
                    T2CON = 0x04 ;

                    T1=0;                                    // pun rdy=0;
                    while (WR==1);                    // astept ack

                    //ASTEPT 2,5ms
                    while (!TF2);
                    TF2=0;
                //  while (WR==1);                    // astept ack
                    T1=1;                                    //rdy=1;
                    while (WR==0);



                    RCAP2L = 0x00 ;          // perioada de transmisie =
                    RCAP2H = 0xF7 ;          //  = 2*(10000h-EE00h)*1.085us
                    TL2 = 0x00 ;          //  = 2,5ms
                    TH2 = 0xF7 ;

goto start1;
}
```

Anexa 3. Codul Matlab pentru determinarea erorilor modelului software a unui strat Hebbian la diverse precizii de reprezentare a ponderilor

```
  Instruire si simulare RNA Hebbian
  Instruire
set_instruire_15x1      % incarca programul care genereaza vectorul
                        % de intrare c si vectorul tinta d
for p=8:16
cnb=p;
A=d*pinv(c);            % regula hebbiana de instruire supervizata
A=round(2^p*A)/2^p;       reduce precizia de reprezentare matricii
                        % ponderilor la aprox. 11 biti
  Simulare
net=newlin(minmax(c),n,0,0.01);
net.b{1}=zeros(n,1);
net.iw{1,1}=A;
y=round(sim(net,c));
e=y-d;
ea=abs(e);
em=zeros(n,g*s);
%Calculul numarului total de erori
r(p-7)=0;
for m=1:g*s*n
    em(m)=ea(m);
    ev=sum(em);
    if      ea(m)>0
    r(p-7)=r(p-7)+1;
    end
end
r1(p-7)=0;
for k=1:g
    if      ev(k)>0
    r1(p-7)=r1(p-7)+1;
end
end
end
    x=1:1:9;
    plot(x+7,r1(x), x+7,r(x),'r')
```

Anexa 4. Codul Matlab pentru determinarea erorilor modelului Simulink/System Generator pentru un strat Hebbian la diverse precizii de reprezentare a ponderilor

```
for p=8:16
cnb=p;
cbp=p;
nob=8+cnb+ceil(log2(n));
sim('Paralelism_neuron_15x1.mdl')

load output.mat;
out=out';
dl=matcol(d);
e=out(9:n*g*s+8,2)-dl;
d3=[out(9:n*g*s+8,2) dl e];
ea=abs(e);
em=zeros(n,g*s);
%Calculul numarului total de erori
r(p-7)=0;
for m=1:g*s*n
    if     ea(m)>0
    r(p-7)=r(p-7)+1;
    em(m)=ea(m);
    ev=sum(em);
    end
 end
rl(p-7)=0;
for k=1:g*s
    if     ev(k)>0
    rl(p-7)=rl(p-7)+1;
    end
end
%    x=1:1:g*s*n;
%    plot(x,ea(x),'r*')
clear m;
clear k;
end
    x=1:1:9;
    plot( x+7,r(x),'r', x+7,rl(x))
```

Anexa 5. Codul Matlab pentru determinarea erorilor modelului RNA-LM 2x7 la diverse precizii de reprezentare a ponderilor

```
for p=8:16

cnb=p+5;
cbp=p;
q = quantizer('fixed', 'round', 'saturate', [cnb cbp]);
coef1=round(q,coef_o1);
coef2=round(q,coef_o2);
sim('Paralelism_neuron.mdl')

Yo1=sigmoid(Y); % iesirea calculata a stratului 1

X2=[Yo1;ones(1,150)];  %intrari + bias, strat 2
Yo2=A2'*X2;    % iesirea neta calculata a stratului 2
Yo=round(Yo2);  % iesire strat neuronal (rotunjit)
E=d-Yo;        % eroare la iesire (calculata)


z1=simout1.signals.values;  % iesirea strat 1 (simulare)
Zo1=(z1(10:1059,1));        % iesirea strat 1 (corectata)

Eo1=matcol(Yo1(1:1050))-Zo1;       % eroare iesire strat 1 (calcul-simulare)
Ec1=[matcol(Yo1(1:1050)) Zo1 Eo1];     % comparatie calcul-simulare iesire

z2=simout2.signals.values;    % iesirea neta simulare strat 2
Z2=(z2(21:1070,1));          % iesirea neta simulare (corectata) strat
Eo2=d1(1:1050,1)-Z2;          % eroare la iesirea neta: calcul-simulare
Ec2=[d1(1:1050,1) Z2 Eo2];    % comparatie calcul-simulare iesire neta

ea=abs(Eo2);
em=zeros(n,g*s);
ea1=abs(Eo1);
em1=zeros(n,g*s);
eda=abs(E);
edm=zeros(n,g*s);

%Calculul numarului total de erori
r2(p-7)=0;
r1(p-7)=0;
rd(p-7)=0;
for m=1:1050
   if    ea(m)>0
   r2(p-7)=r2(p-7)+1;
   em(m)=ea(m);
```

```
    ev=sum(em);
    end
    if ea1(m)>0.025
       r1(p-7)=r1(p-7)+1;
       em1(m)=ea1(m);
       ev1=sum(em1);
    end
    if eda(m)>0
       rd(p-7)=rd(p-7)+1;
       edm(m)=eda(m);
       edv=sum(edm);
    end

end
rv2(p-7)=0;
rv1(p-7)=0;
rdv(p-7)=0;
for k=1:g*s
    if    ev(k)>0
    rv2(p-7)=rv2(p-7)+1;
    end
    if   ev1(k)>0
    rv1(p-7)=rv1(p-7)+1;
    end
    if   edv(k)>0
    rdv(p-7)=rdv(p-7)+1;
    end
end
```

```
clear m;
clear k;
end
r1  % numar esantioane eronate la iesirea stratului 1
rv1 % numar vectori eronati la iesirea stratului 1
r2  % numar esantioane eronate la iesirea stratului 2
rv2 % numar vectori eronati la iesirea stratului 2
rd
rdv
   x=1:1:9;

   plot( x+7,rv2(x),'r', x+7, r1(x),  x+7, rdv(x), 'g')
```

Anexa 6. Vectorii de antrenare şi test

6.1 Vectorii de antrenare: date_a.mat

g=15,
n=7,
s=10,
a=

1	60	170	216	213	182	114	243
2	61	170	216	213	182	114	243
3	61	170	216	213	182	113	243
4	61	170	216	213	182	113	243
5	61	171	216	213	182	113	243
6	60	171	216	213	182	113	243
7	60	170	216	213	182	113	243
8	60	171	216	213	182	113	243
9	60	170	216	213	182	113	243
10	61	171	216	213	182	113	243
11	57	44	191	208	186	87	246
12	57	44	190	208	186	88	246
13	57	43	190	208	186	87	246
14	57	43	190	208	186	88	246
15	57	43	190	208	186	88	246
16	57	43	190	208	186	89	246
17	57	42	190	208	186	89	246
18	56	42	190	208	186	89	246
19	57	42	190	208	186	91	246
20	56	42	191	208	186	92	245
21	48	0	65	185	185	83	246
22	48	0	65	185	185	84	247
23	47	0	65	185	184	84	247
24	47	0	66	185	184	87	246
25	48	1	67	185	184	89	246
26	48	1	67	185	184	90	246
27	48	1	68	185	184	93	246
28	47	0	67	185	184	92	245
29	47	0	67	185	184	94	246
30	47	0	67	185	184	94	246
31	245	13	10	18	22	112	243
32	245	13	10	17	21	112	243
33	244	13	11	17	20	111	243
34	244	13	11	17	20	111	243
35	244	13	12	17	20	110	244
36	244	12	12	18	20	110	243
37	243	12	13	17	19	109	244
38	243	12	13	17	19	109	244
39	241	12	13	17	19	110	244
40	242	12	14	17	18	109	244
41	1	12	14	8	11	100	245
42	1	11	14	7	10	96	245
43	1	11	14	7	10	98	245
44	1	11	14	7	11	95	246
45	0	11	14	7	10	97	245
46	0	10	14	7	10	96	245
47	0	9	15	7	10	97	245
48	0	10	15	7	11	96	245
49	1	10	15	7	10	97	245
50	1	10	14	7	11	96	245
51	61	186	224	208	185	143	122
52	61	186	225	208	185	142	122
53	61	185	225	208	185	143	123
54	62	185	225	208	185	143	122
55	62	185	225	208	185	143	122
56	62	185	225	208	185	143	122
57	62	185	225	208	185	143	123
58	62	185	225	208	185	143	122
59	62	185	225	208	185	142	122
60	62	185	225	208	185	143	122
61	56	47	200	206	191	140	126
62	56	47	200	206	191	139	125
63	56	47	199	205	190	140	126
64	57	46	199	206	190	140	125
65	57	46	199	206	191	140	125
66	58	46	199	206	191	140	125
67	58	46	199	205	191	140	125
68	58	46	199	205	190	139	125
69	58	45	199	205	191	141	125
70	58	45	199	205	190	140	125
71	50	0	66	176	179	141	123
72	51	0	66	176	178	140	123
73	51	0	66	175	178	140	123
74	51	0	66	175	178	140	124
75	51	0	66	175	178	141	123
76	51	0	66	176	178	141	123
77	51	0	66	176	177	141	124
78	51	0	65	176	177	140	123
79	52	0	64	175	177	141	123
80	52	0	65	175	177	141	124
81	249	16	12	20	19	144	110
82	248	15	12	20	19	145	110
83	249	15	13	20	19	145	111
84	250	15	13	20	19	145	111
85	250	14	13	20	19	145	111
86	251	14	13	20	19	144	111
87	250	14	14	20	19	144	111
88	250	15	14	20	19	145	111
89	250	15	13	20	19	145	112
90	251	15	14	20	19	145	112
91	10	15	14	13	7	148	116
92	10	15	15	12	7	147	116
93	10	15	15	12	7	147	116
94	11	14	15	12	7	147	115
95	11	15	16	12	8	147	116
96	10	15	16	12	8	147	115
97	10	15	16	12	8	147	115
98	9	15	17	13	9	147	115
99	9	15	17	13	9	146	115
100	9	15	17	13	9	146	115
101	205	170	176	198	196	62	249
102	205	169	175	198	196	61	249
103	205	169	175	198	196	63	249
104	205	168	175	198	195	63	249
105	204	167	176	198	195	64	249
106	204	167	176	197	195	66	248
107	204	167	176	197	195	67	248
108	204	167	176	198	194	67	248
109	204	167	176	198	194	67	248
110	204	167	176	198	194	68	248
111	252	1	46	188	180	16	117
112	253	0	46	188	180	16	117
113	253	0	46	188	180	16	117
114	252	1	45	188	180	16	117
115	253	1	45	188	180	16	116
116	252	2	45	187	180	16	118
117	252	2	45	188	179	16	116
118	252	2	45	187	179	16	116
119	254	2	45	187	179	17	115
120	254	2	45	187	179	16	118
121	127	118	101	89	60	128	235
122	125	135	102	90	61	128	236
123	121	139	113	94	63	123	237
124	122	122	105	93	63	128	230
125	122	120	104	93	63	126	232
126	122	120	104	93	63	127	228
127	120	120	104	92	63	127	227
128	120	120	104	92	63	127	225
129	119	120	103	92	62	127	225
130	119	120	104	92	62	128	224
131	112	14	184	178	171	138	188
132	111	14	184	177	170	138	188
133	111	14	184	178	170	138	189
134	111	14	184	178	170	138	189
135	111	14	185	178	170	137	189
136	110	14	184	178	169	138	188
137	109	14	185	178	169	137	189
138	109	14	184	177	169	137	189
139	109	14	185	178	169	137	189
140	109	14	185	178	169	137	190
141	220	0	2	11	25	25	79
142	219	0	2	11	24	24	84
143	219	0	2	12	24	23	83
144	217	0	3	12	25	25	80
145	216	0	3	12	25	23	83
146	216	0	3	12	25	23	85
147	214	0	4	12	24	23	84
148	213	0	4	13	25	23	84
149	212	0	4	13	25	23	85
150	203	0	4	13	23	22	87

6.2 Vectorii de test: date_t.mat

g=15,
n=7,
s=10,
a=

1	49	154	200	205	181	49	250
2	49	154	200	205	182	48	250
3	49	154	200	205	182	49	250
4	49	154	200	205	182	49	250
5	49	155	200	205	181	49	250
6	49	154	200	205	182	50	250
7	49	154	200	205	182	50	250
8	49	154	200	205	182	50	250
9	49	155	200	205	182	50	250
10	50	155	200	205	182	50	250
11	52	35	176	203	190	46	250
12	52	35	176	203	190	46	250
13	52	35	176	203	190	46	250
14	52	35	176	203	190	45	250
15	52	34	176	203	190	46	250
16	51	33	175	203	190	44	250
17	51	33	175	203	190	44	250
18	52	33	176	203	190	43	250
19	52	33	176	203	190	42	251
20	52	34	175	203	189	43	250
21	54	2	52	182	193	54	249
22	54	1	53	182	193	56	249
23	53	1	53	182	193	58	249
24	54	1	54	182	193	58	249
25	54	1	54	182	193	59	249
26	54	2	54	182	192	58	249
27	54	2	55	182	192	59	249
28	54	4	55	182	192	60	249
29	55	4	55	182	192	60	249
30	54	4	55	182	192	62	249
31	225	7	15	19	15	107	244
32	225	7	15	19	16	107	244
33	225	7	15	19	16	108	244
34	224	7	16	19	16	107	244
35	224	7	16	19	16	108	244
36	224	7	16	19	16	108	244
37	224	7	17	19	17	108	244
38	224	7	17	19	17	108	244
39	224	8	17	19	18	108	244
40	223	6	17	19	17	109	244
41	25	9	11	12	18	112	243
42	24	9	12	12	18	111	243
43	24	9	12	12	18	111	244
44	24	9	12	12	19	111	243
45	24	9	12	12	19	111	243
46	24	9	12	12	19	111	243
47	24	9	12	11	20	112	243
48	24	8	12	11	20	111	243
49	24	8	12	11	21	112	243
50	24	8	12	11	21	111	243
51	63	150	185	197	179	139	126
52	63	150	185	197	179	138	126
53	63	150	185	198	179	138	125
54	63	150	185	197	179	139	125
55	62	150	185	197	179	139	125
56	62	150	185	197	178	139	125
57	62	149	185	198	179	139	125
58	62	149	185	198	179	138	125
59	62	150	186	198	179	138	125
60	62	150	186	198	179	138	125
61	62	35	177	197	185	135	127
62	62	34	179	197	185	135	127
63	62	34	178	197	185	135	127
64	62	36	180	197	185	136	126
65	62	35	180	197	185	136	126
66	62	35	180	197	184	135	126
67	62	34	181	197	184	135	127
68	61	34	181	197	184	136	126
69	61	34	181	197	184	136	126
70	62	34	181	197	185	136	126
71	52	0	61	170	163	136	126
72	52	0	62	170	164	135	126
73	52	0	62	170	163	135	125
74	51	0	62	170	162	135	126
75	51	0	62	170	162	136	126
76	52	0	61	170	162	135	125
77	53	0	60	170	162	135	126
78	52	0	59	170	162	136	126
79	53	0	58	170	161	135	126
80	54	0	58	170	161	135	126
81	244	5	7	23	18	135	118
82	244	4	8	24	18	135	118
83	244	4	8	23	17	135	119
84	243	4	8	24	17	135	119
85	244	3	8	23	17	135	118
86	244	2	8	23	16	135	118
87	243	2	9	23	16	136	119
88	243	1	10	23	17	135	118
89	243	0	9	23	16	136	119
90	243	0	9	23	16	136	119
91	12	6	5	15	10	140	121
92	12	6	5	15	10	140	121
93	12	5	5	15	10	140	120
94	12	6	5	15	10	140	120
95	12	5	6	15	10	140	120
96	12	5	5	15	9	140	120
97	12	5	6	15	9	140	120
98	12	5	6	15	10	140	120
99	12	5	7	15	10	140	120
100	12	5	7	15	10	139	120
101	212	186	202	192	191	39	250
102	212	185	203	193	191	40	251
103	212	186	202	192	191	39	251
104	212	186	203	192	191	39	251
105	212	186	202	193	191	40	251
106	212	186	202	192	191	41	251
107	212	186	202	192	191	41	250
108	212	185	203	193	191	42	251
109	212	185	203	193	191	42	251
110	212	186	202	192	191	44	250
111	255	0	40	178	181	50	121
112	255	0	41	178	181	50	119
113	255	0	41	178	180	51	120
114	255	0	40	179	180	49	119
115	255	0	40	179	180	52	119
116	255	0	40	179	180	51	119
117	255	0	40	179	180	50	119
118	255	0	40	179	180	52	119
119	255	0	40	179	180	50	119
120	255	0	39	179	179	51	119
121	104	109	109	95	61	123	198
122	104	109	109	95	61	123	198
123	103	109	110	96	62	123	198
124	104	109	110	96	62	123	197
125	104	109	109	96	62	123	198
126	103	109	110	95	61	123	198
127	102	109	110	96	61	123	199
128	102	109	110	96	61	124	198
129	102	109	110	96	61	123	198
130	102	109	110	95	61	124	197
131	104	17	162	174	178	132	183
132	105	17	162	174	179	132	183
133	104	17	163	174	179	131	183
134	104	17	163	174	178	131	183
135	104	17	163	174	179	131	183
136	104	17	163	174	179	131	183
137	104	17	163	174	179	131	183
138	103	17	163	174	179	131	182
139	103	17	163	174	179	131	182
140	103	17	164	174	179	131	182
141	203	0	0	20	30	22	89
142	203	0	0	20	30	22	87
143	201	0	0	21	30	22	88
144	202	0	0	21	30	22	87
145	203	0	0	19	29	21	90
146	203	0	0	20	30	21	90
147	202	0	0	20	30	21	92
148	204	0	0	20	29	21	91
149	202	0	0	19	29	21	91
150	202	0	0	19	29	21	91

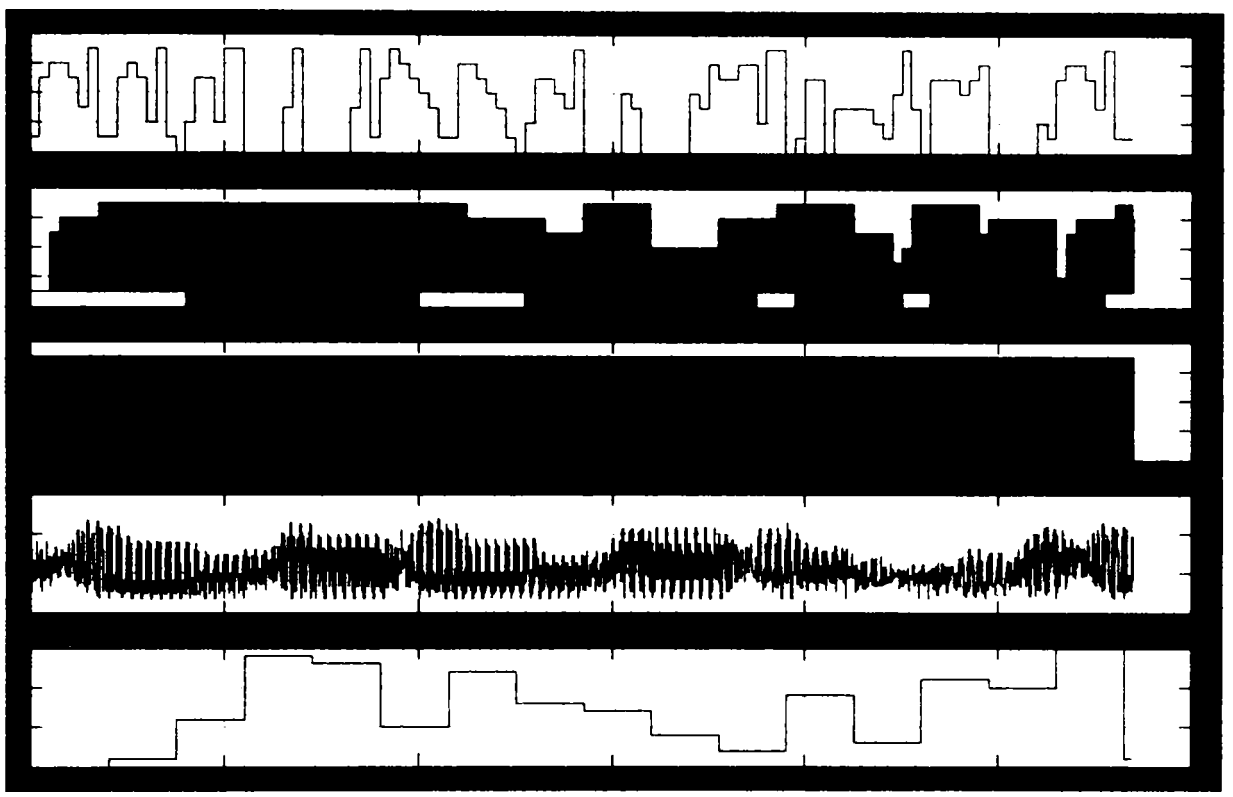
6.3 Vectorii de test: date_t1.mat

g=15,
n=7,
s=10,
a=

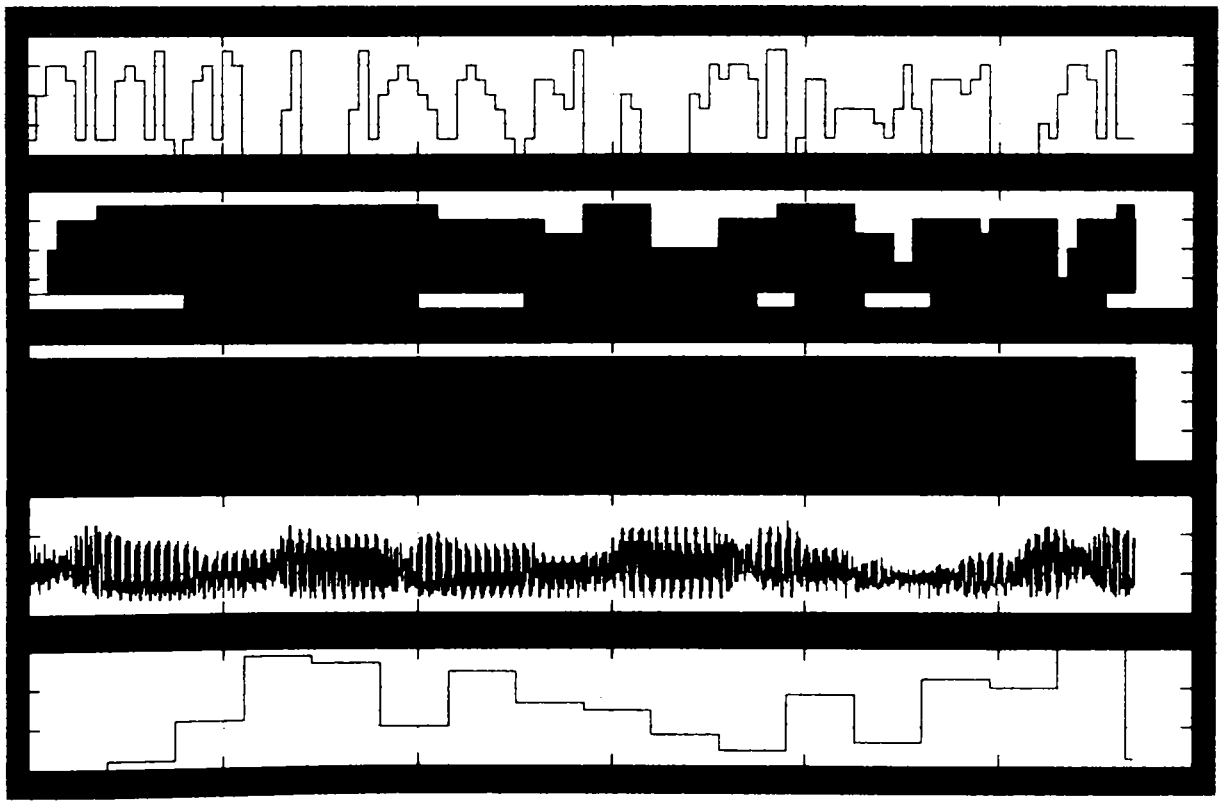
1	27	211	233	200	182	109	244
2	28	211	233	200	182	109	244
3	27	210	233	200	182	108	244
4	27	210	233	200	182	110	244
5	27	210	233	200	182	109	244
6	27	211	233	200	182	110	244
7	27	211	233	200	182	109	244
8	27	210	233	200	182	110	244
9	27	211	233	199	182	109	244
10	28	211	233	200	182	109	244
11	17	51	222	206	183	101	245
12	17	51	223	206	183	101	245
13	19	49	222	206	183	102	244
14	20	49	222	206	183	102	244
15	21	48	222	206	183	103	244
16	24	46	222	206	182	104	244
17	24	45	222	206	182	104	244
18	22	45	222	206	182	103	244
19	22	45	222	206	183	103	244
20	22	46	222	206	182	105	244
21	25	0	89	194	183	103	244
22	26	0	89	194	184	103	244
23	26	0	90	194	183	105	244
24	27	0	90	193	183	105	244
25	27	0	90	193	183	104	244
26	26	0	89	193	182	106	244
27	25	0	89	193	182	105	244
28	25	0	89	193	182	105	244
29	24	0	88	193	182	105	244
30	23	0	89	193	182	106	244
31	255	6	25	13	34	108	244
32	255	6	25	13	34	108	244
33	255	6	26	13	34	108	244
34	255	5	25	13	34	108	244
35	255	6	25	13	34	109	244
36	255	6	25	13	34	108	244
37	255	6	25	13	34	108	244
38	255	6	25	13	34	109	244
39	255	5	25	13	34	108	244
40	255	5	25	12	34	109	244
41	0	0	13	0	12	93	246
42	0	0	13	0	12	90	246
43	0	0	13	1	12	92	245
44	0	0	13	1	12	90	246
45	0	0	13	1	12	93	246
46	0	0	14	1	12	89	246
47	0	0	14	0	12	90	246
48	0	0	14	1	12	90	246
49	0	0	14	1	12	89	246
50	0	0	14	1	13	90	246
51	39	212	241	209	179	143	117
52	39	212	241	209	179	142	117
53	40	213	240	209	179	143	118
54	41	213	240	209	178	142	118
55	41	213	241	209	178	143	119
56	41	213	240	209	178	142	118
57	42	213	240	209	178	142	118
58	43	213	240	209	178	141	118
59	43	213	240	209	178	142	118
60	43	212	240	209	179	141	118
61	28	53	222	212	181	142	120
62	28	52	223	213	181	142	120
63	29	53	223	212	181	143	120
64	29	53	223	213	181	142	121
65	29	53	223	213	180	142	120
66	30	53	223	213	180	142	120
67	30	53	223	213	180	143	120
68	29	53	224	213	181	142	120
69	29	53	224	213	180	142	119
70	30	53	224	213	181	143	119
71	26	0	72	189	201	144	121
72	27	0	71	189	200	143	119
73	28	0	71	189	200	144	120
74	28	0	72	188	199	144	119
75	29	0	72	189	200	143	118
76	29	0	71	188	199	144	119
77	30	0	72	188	199	144	119
78	30	0	71	188	199	144	119
79	30	0	70	188	199	143	118
80	31	0	70	188	199	144	119
81	255	14	13	21	21	144	112
82	255	14	13	21	22	143	111
83	255	12	13	20	21	144	111
84	255	12	13	20	21	143	110
85	255	12	14	19	21	143	111
86	255	12	14	20	21	143	111
87	255	12	14	19	21	144	111
88	255	11	13	18	20	144	112
89	255	11	13	18	20	144	112
90	255	11	12	18	20	144	112
91	0	0	9	8	10	150	116
92	0	0	9	8	10	149	116
93	0	0	9	8	10	149	116
94	0	0	10	8	10	149	116
95	0	0	10	8	10	149	116
96	0	0	10	8	9	149	116
97	0	0	10	7	9	149	115
98	1	0	10	8	10	149	116
99	0	0	11	8	9	149	115
100	0	0	11	7	9	149	116
101	212	186	202	192	191	39	250
102	212	185	203	193	191	40	251
103	212	186	202	192	191	39	251
104	212	186	203	192	191	39	251
105	212	186	202	193	191	40	251
106	212	186	202	192	191	41	251
107	212	186	202	192	191	41	250
108	212	185	203	193	191	42	251
109	212	185	203	193	191	42	251
110	212	186	202	192	191	44	250
111	255	0	40	178	181	50	121
112	255	0	41	178	181	50	119
113	255	0	41	178	180	51	120
114	255	0	40	179	180	49	119
115	255	0	40	179	180	52	119
116	255	0	40	179	180	51	119
117	255	0	40	179	180	50	119
118	255	0	40	179	180	52	119
119	255	0	40	179	180	50	119
120	255	0	39	179	179	51	119
121	104	109	109	95	61	123	198
122	104	109	109	95	61	123	198
123	103	109	110	96	62	123	198
124	104	109	110	96	62	123	197
125	104	109	109	96	62	123	198
126	103	109	110	95	61	123	198
127	102	109	110	96	61	123	199
128	102	109	110	96	61	124	198
129	102	109	110	96	61	123	198
130	102	109	110	95	61	124	197
131	104	17	162	174	178	132	183
132	105	17	162	174	179	132	183
133	104	17	163	174	179	131	183
134	104	17	163	174	178	131	183
135	104	17	163	174	179	131	183
136	104	17	163	174	179	131	183
137	104	17	163	174	179	131	183
138	103	17	163	174	179	131	182
139	103	17	163	174	179	131	182
140	103	17	164	174	179	131	182
141	203	0	0	20	30	22	89
142	203	0	0	20	30	22	87
143	201	0	0	21	30	22	88
144	202	0	0	21	30	22	87
145	203	0	0	19	29	21	90
146	203	0	0	20	30	21	90
147	202	0	0	20	30	21	92
148	204	0	0	20	29	21	91
149	202	0	0	19	29	21	91
150	202	0	0	19	29	21	91

Anexa 7. Simularea RNA competitiv cu paralelism de strat

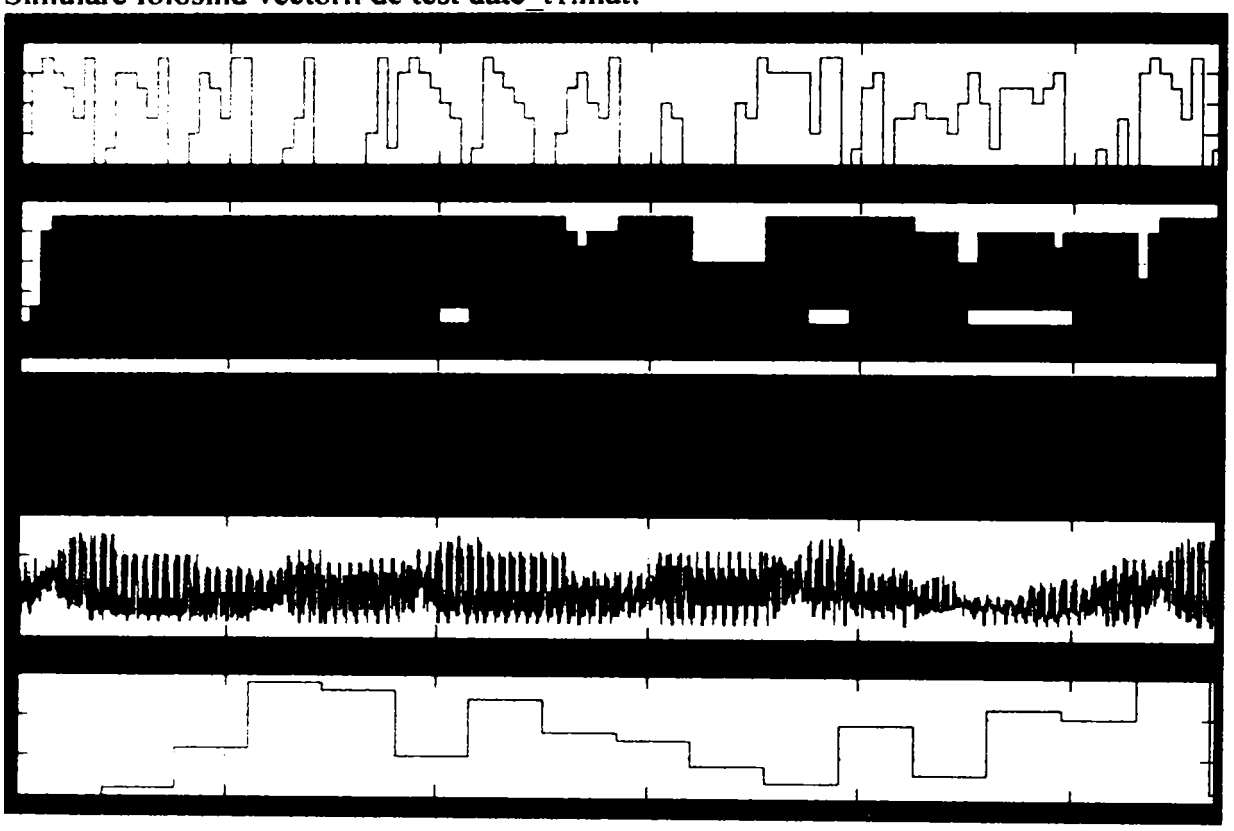
Simulare folosind vectorii de antrenare date_a.mat:



Simulare folosind vectorii de test date_t.mat:

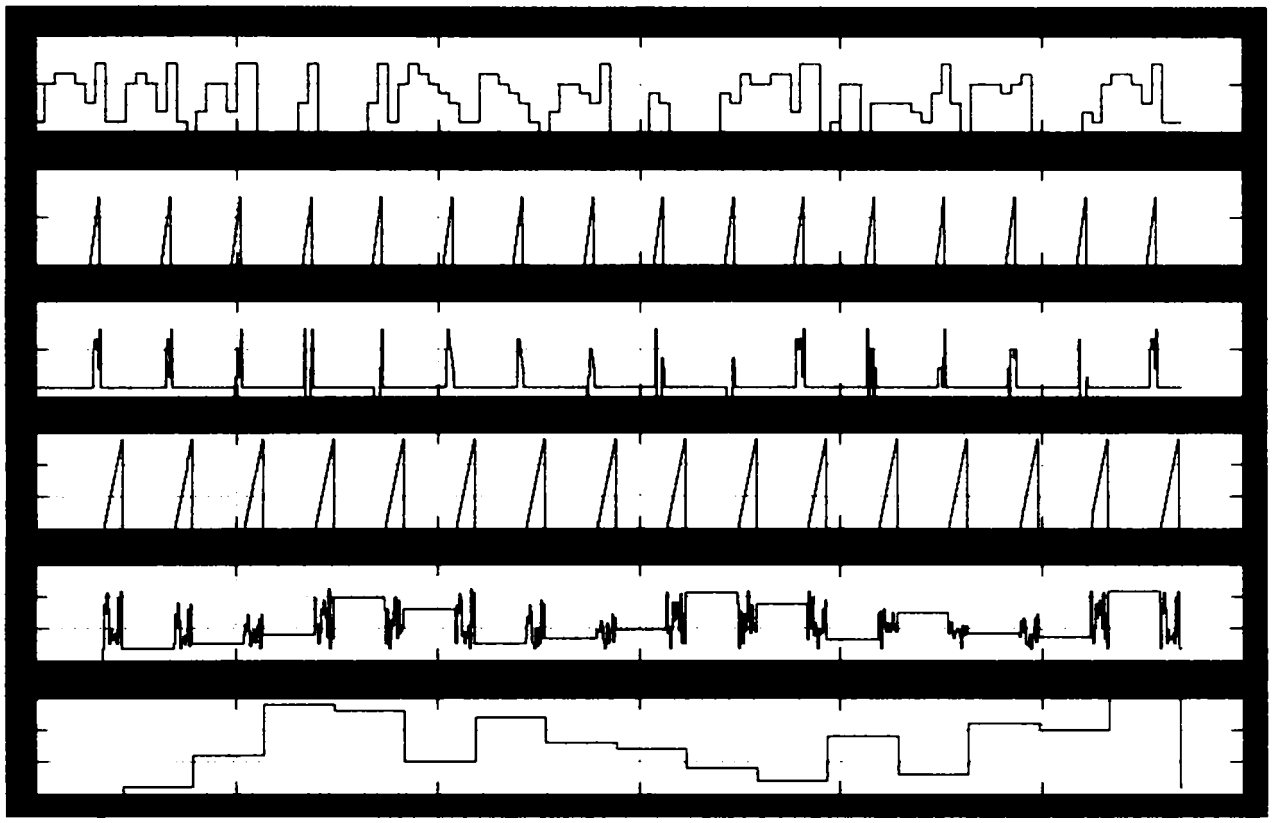


Simulare folosind vectorii de test date_t1.mat:

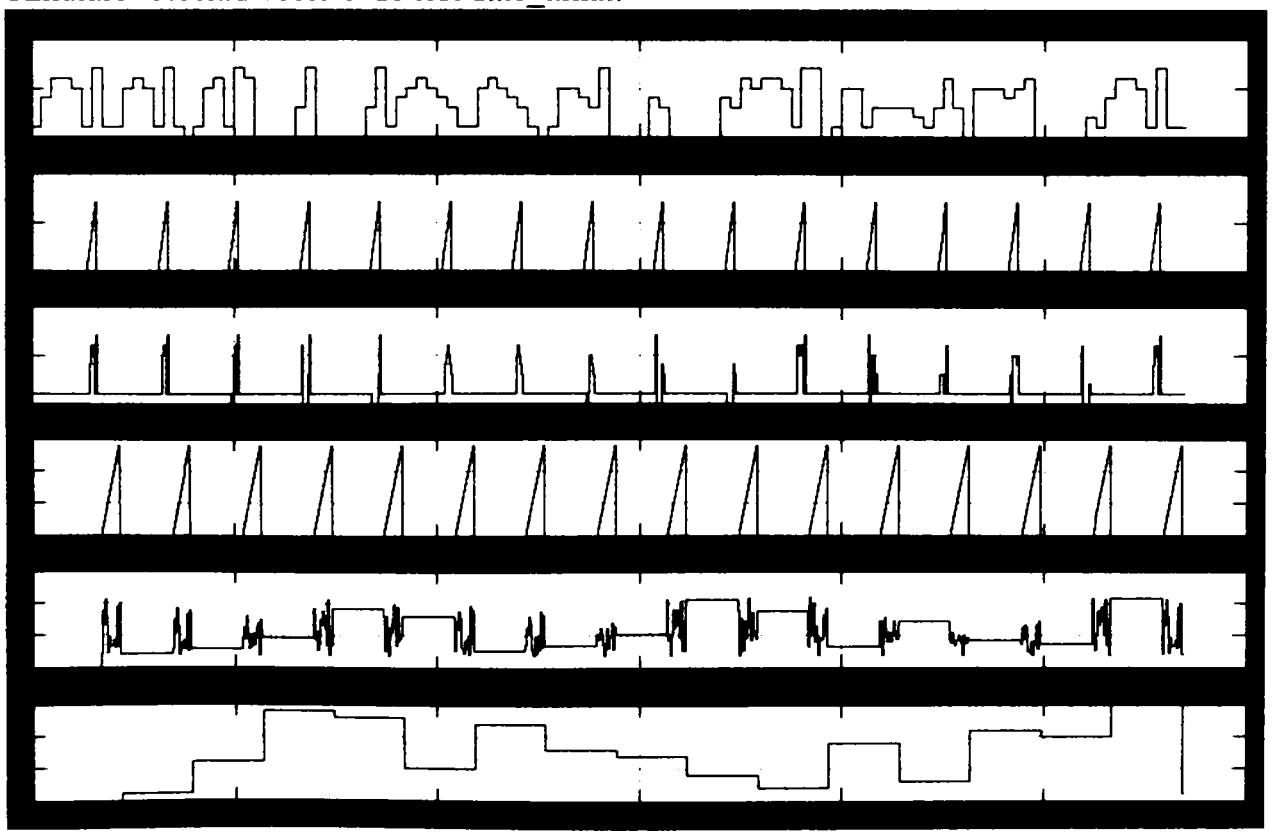


Anexa 8. Simularea RNA competitiv cu paralelism de neuron

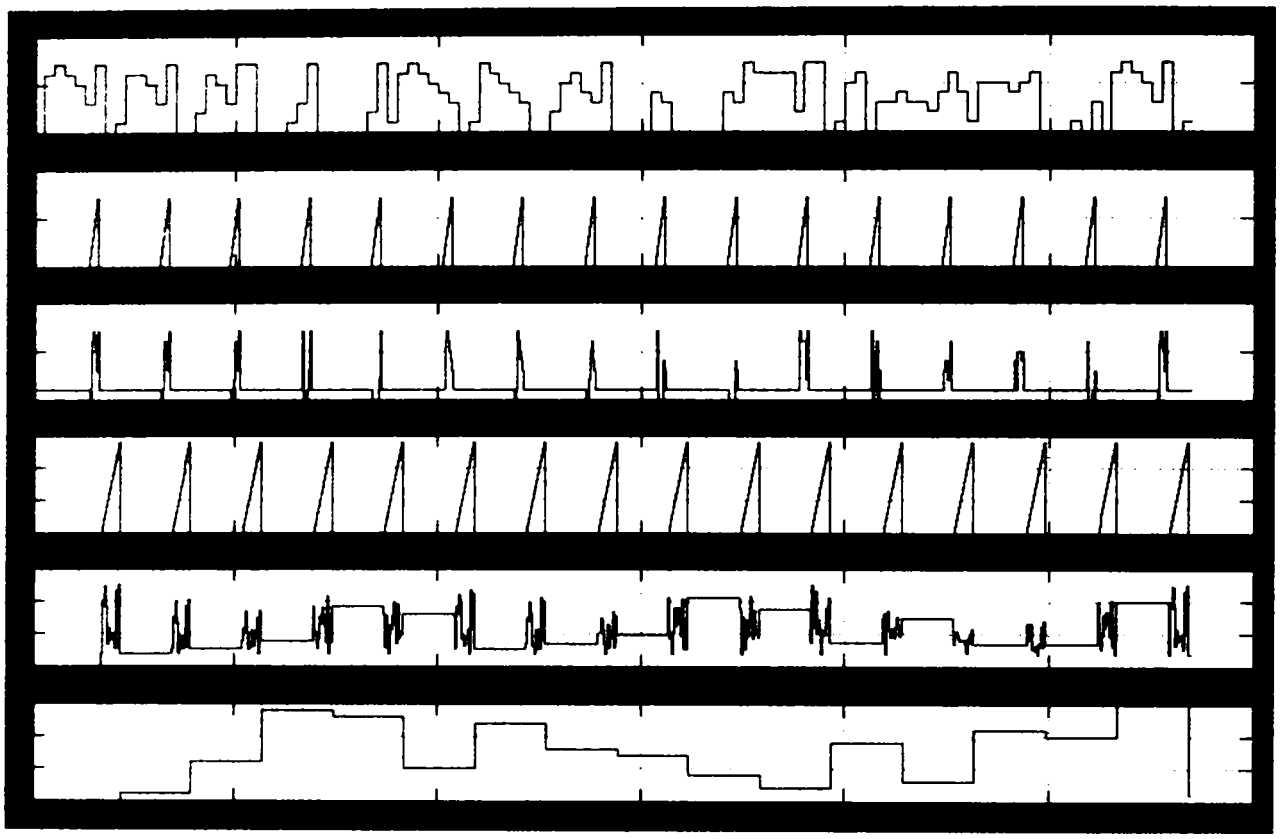
Simulare folosind vectorii de antrenare date_a.mat:



Simulare folosind vectorii de test date_t.mat:



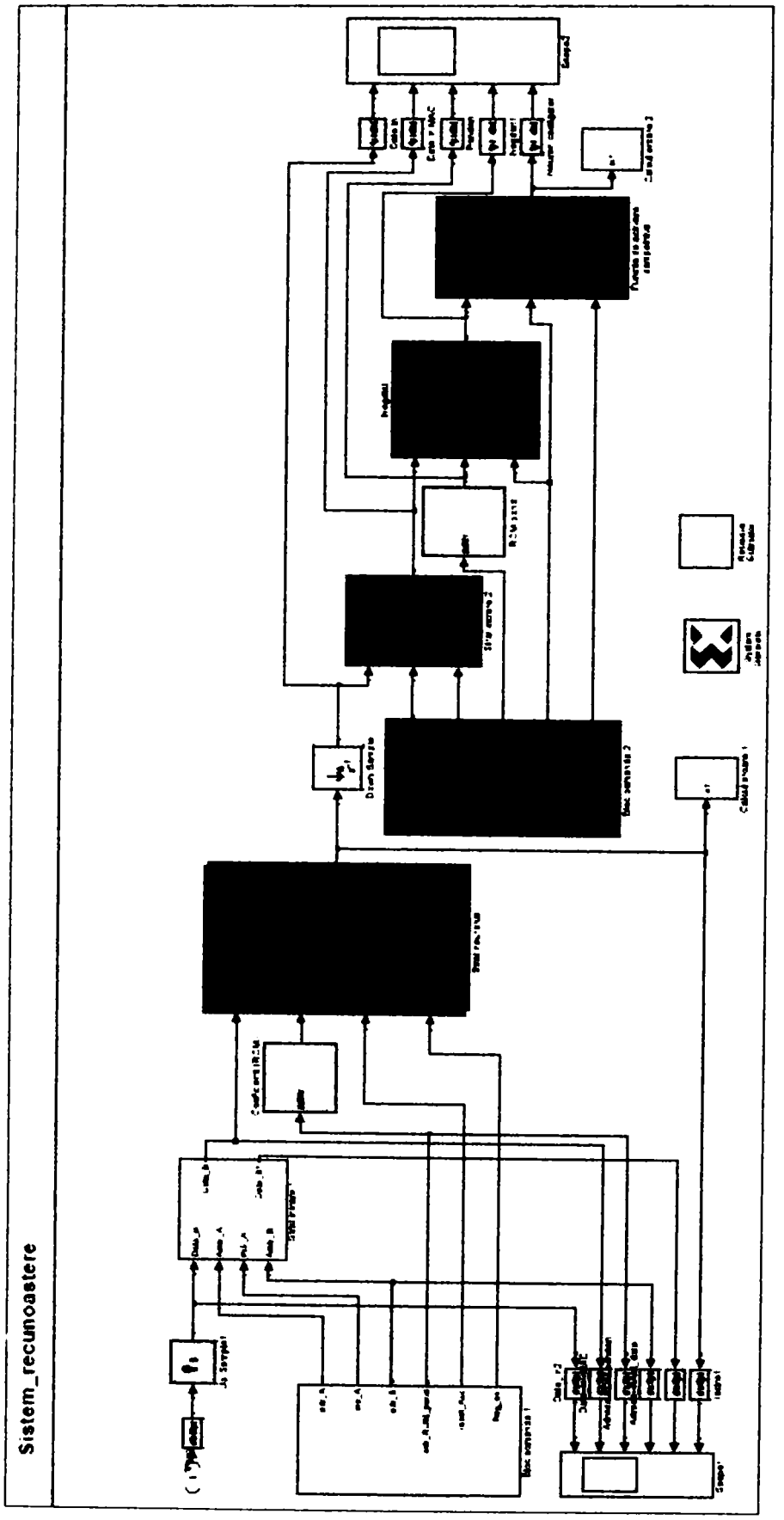
Simulare folosind vectorii de test date_t1.mat:

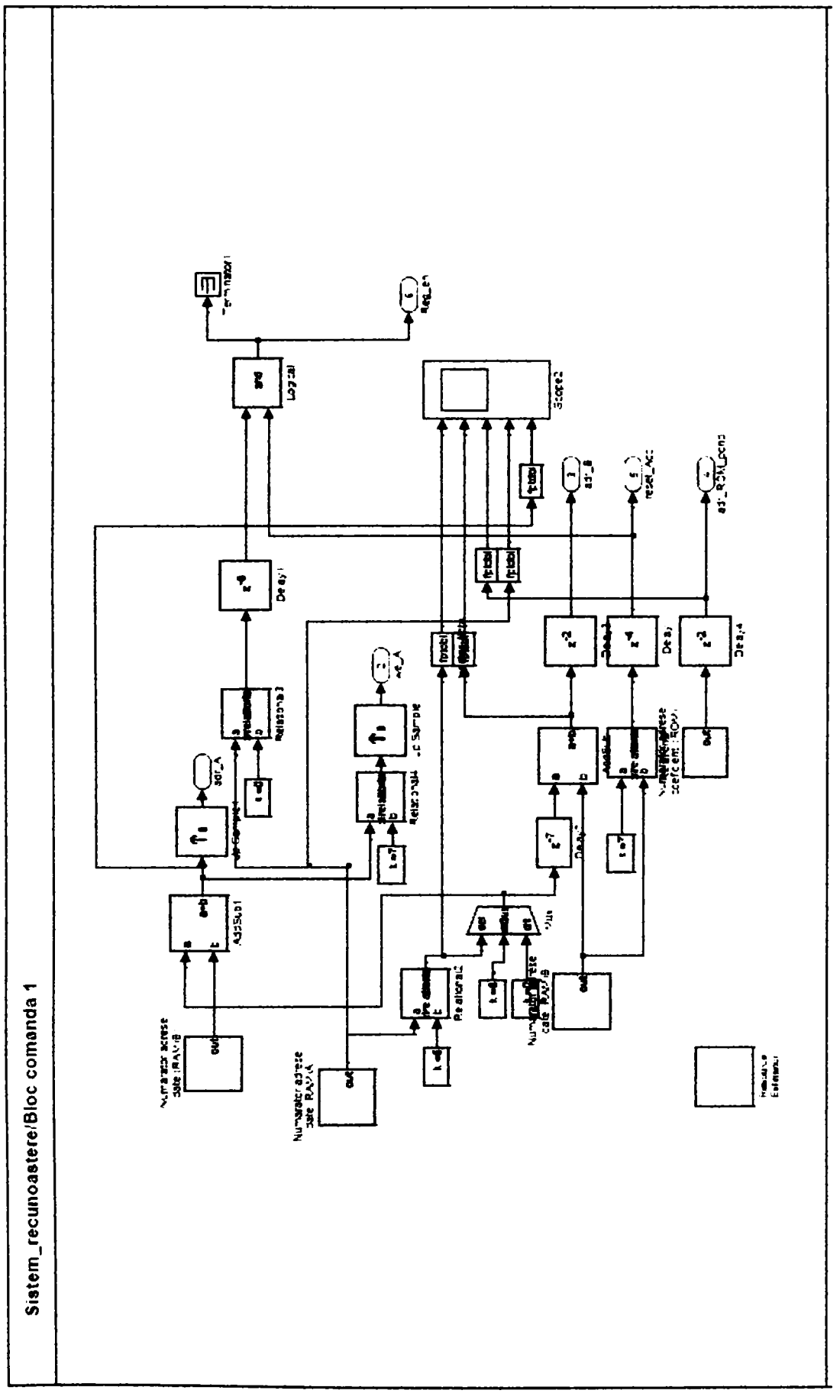


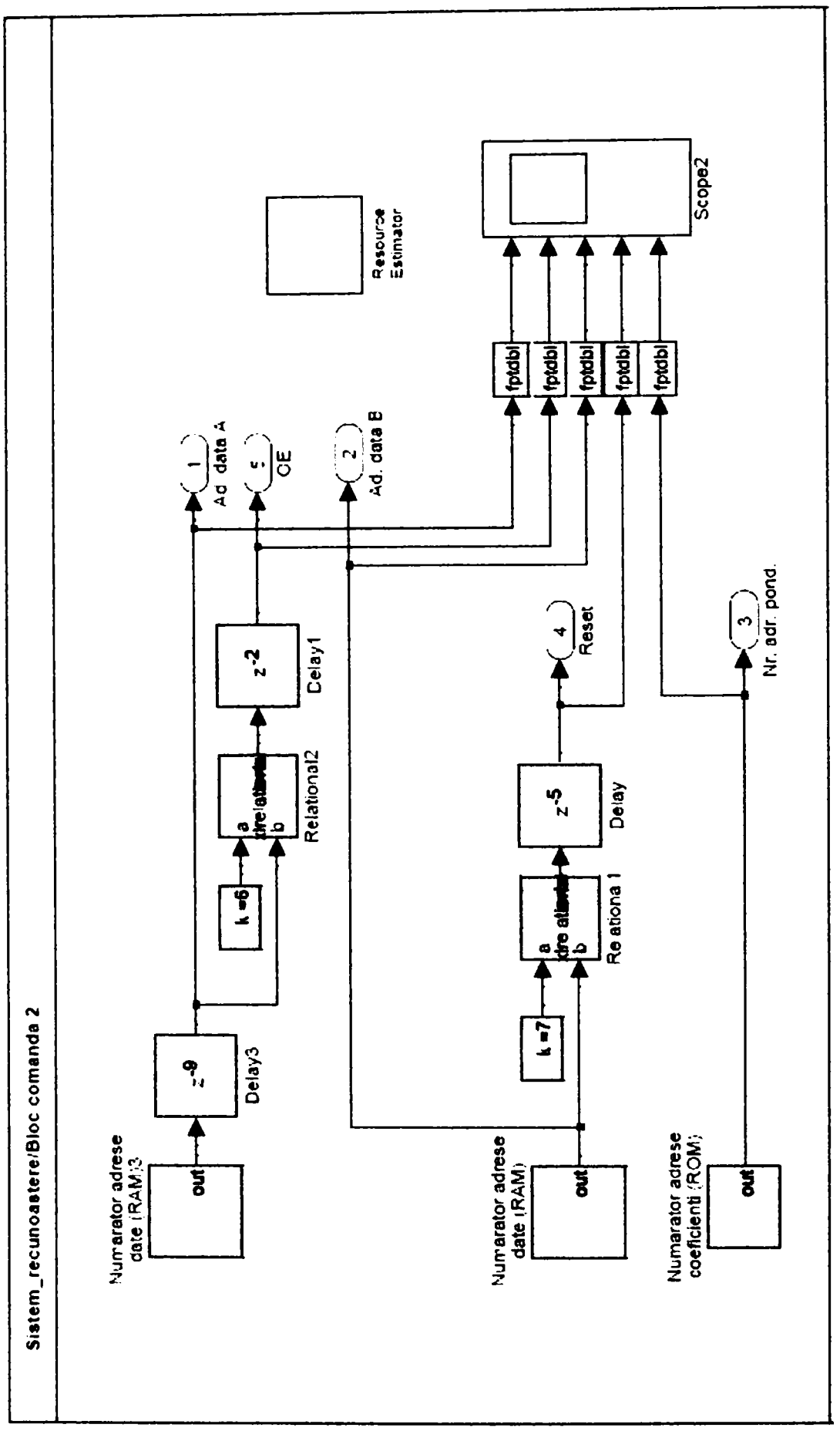
Anexa 9. Codul Matlab pentru determinarea erorilor sistemului de recunoaştere

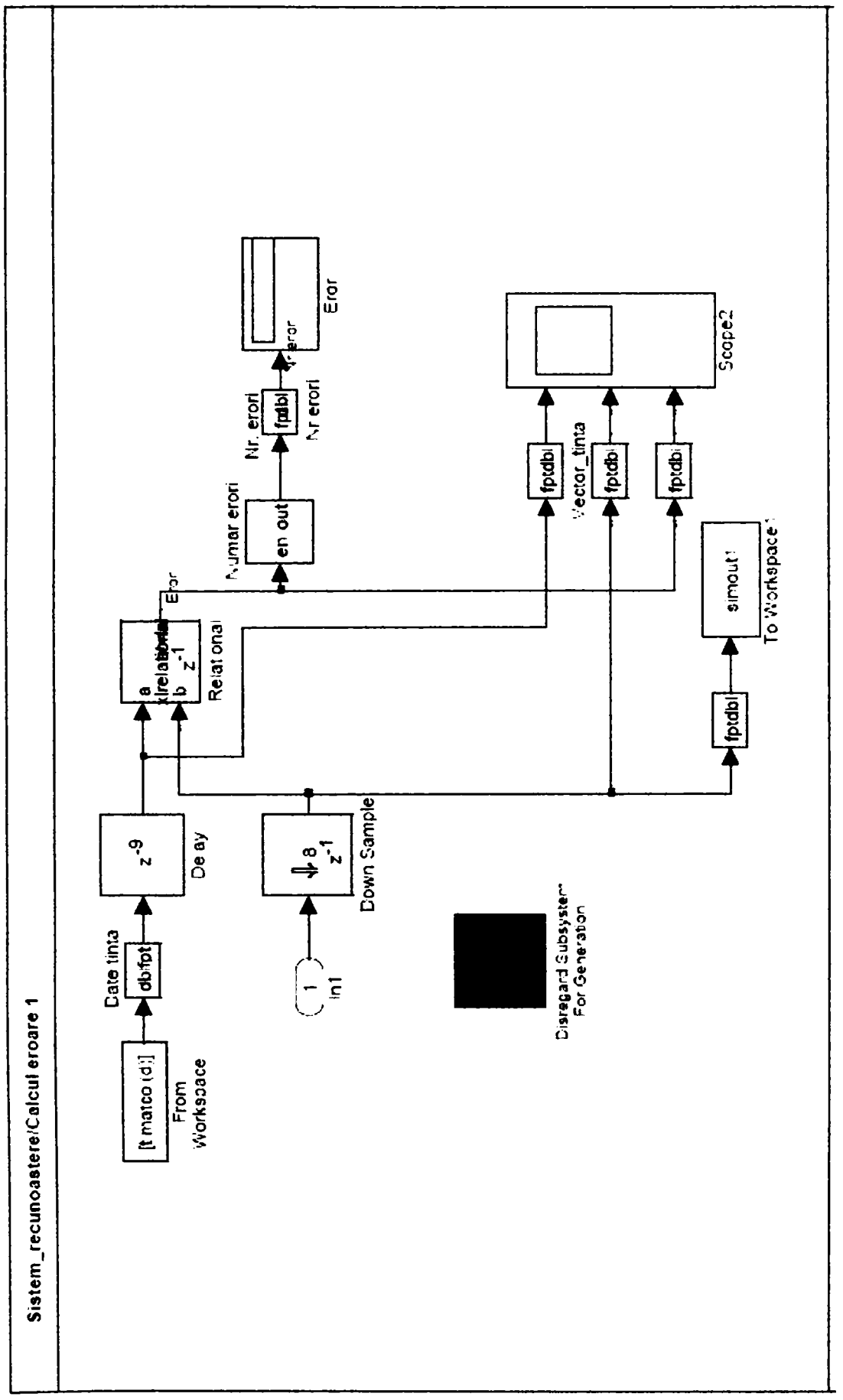
```
% Calculul numarului total de erori
sim ('sistem_recunoastere.mdl')
em=zeros(n,g*s);
r=0;
for m=1:n*s*g
   em(m)=simout(m);
   ev=sum(em);
   if    simout(m)>0
   r=r+1;
   end
 end
r1=0;
for k=1:g*s
   if    ev(k)>0
   r1=r1+1;
   end
end
%
  x=1:1:g*s;
   plot(x,ev(x),'rx')
clear m;
clear k;
end
r
r1
procent_recunoastere=(g*s-r1)*100/(g*s)
x=2:1:9;
plot(x+7,r1(x), x+7,r(x),'r')
```

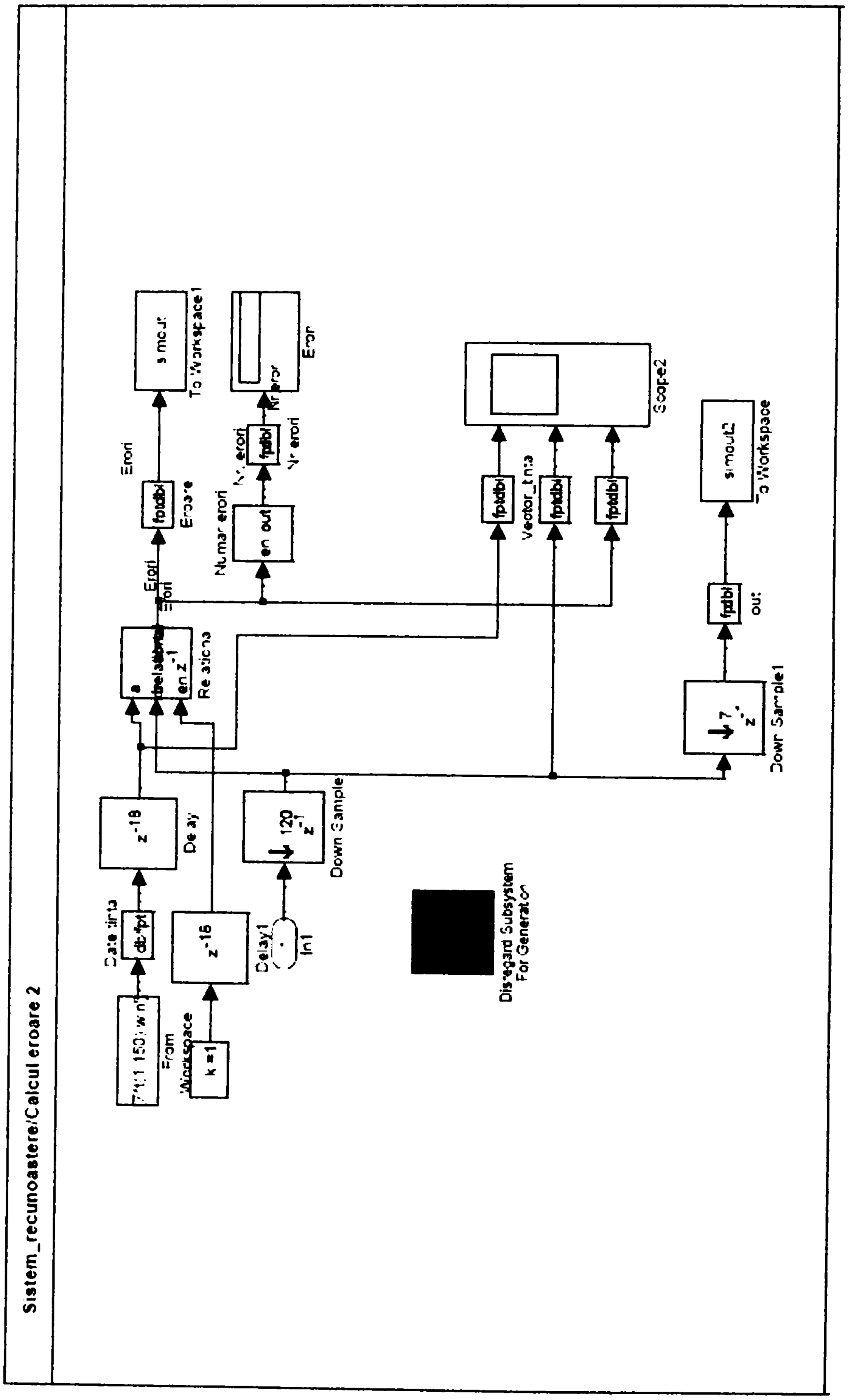
Anexa 10. Modelul sistemului de recunoaştere a gesturilor

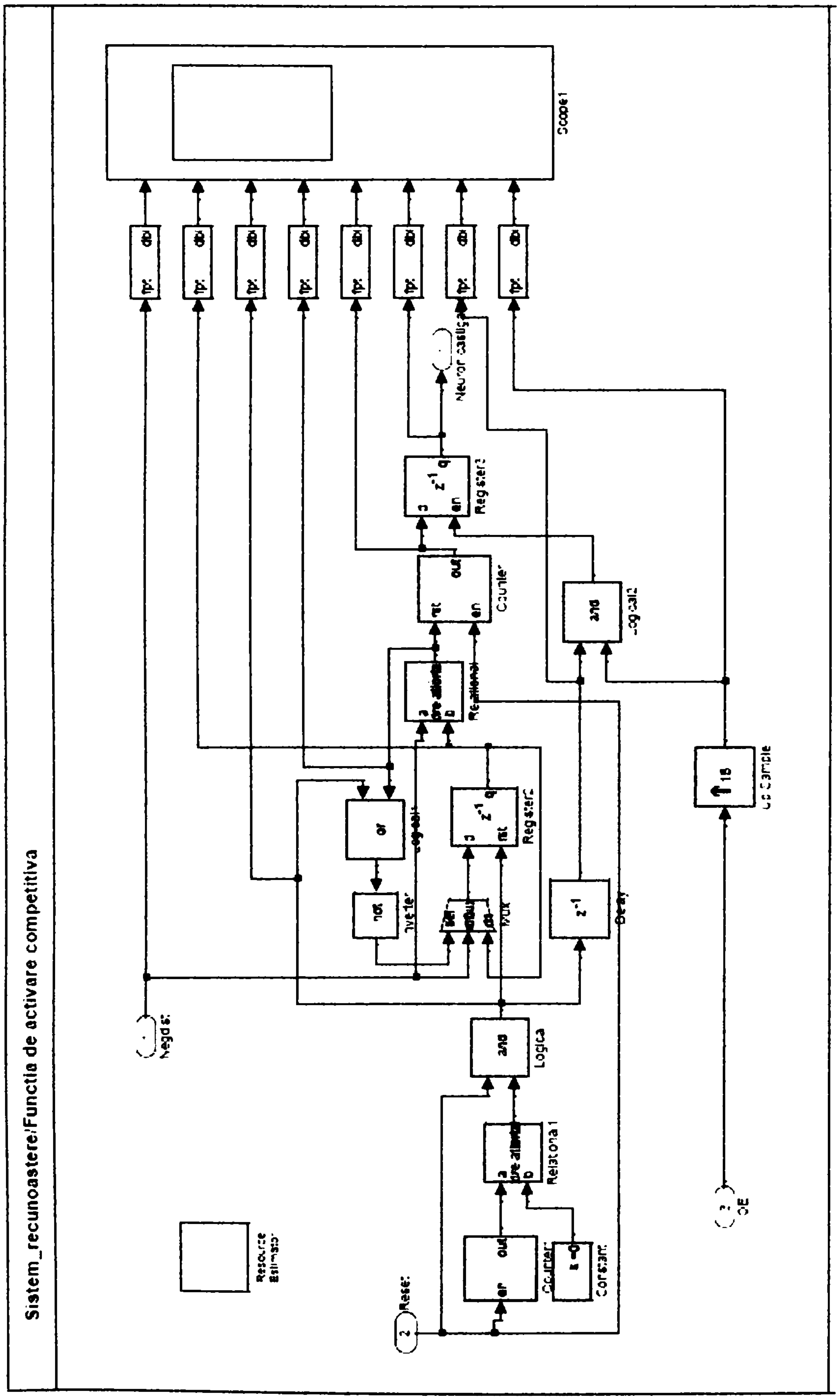


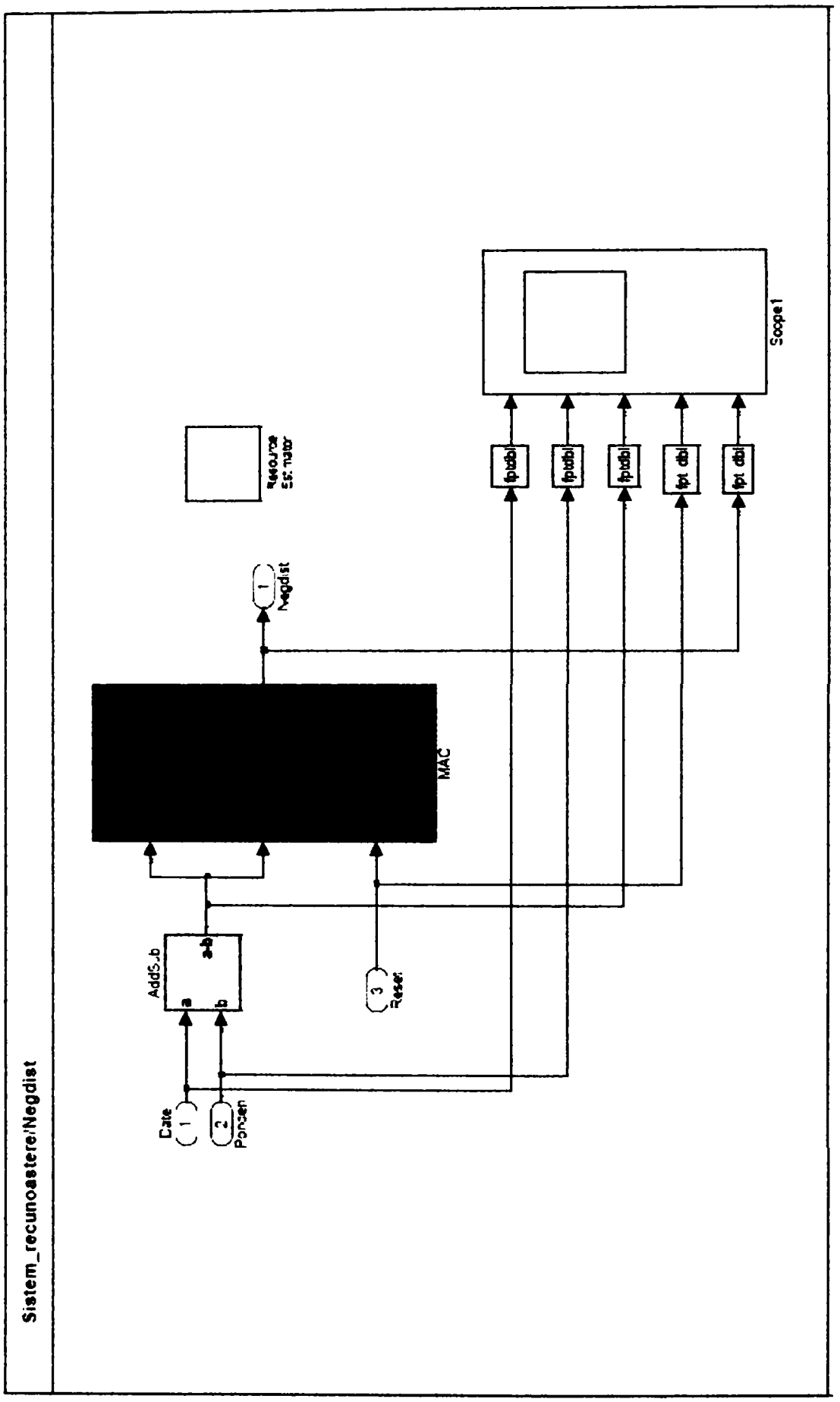


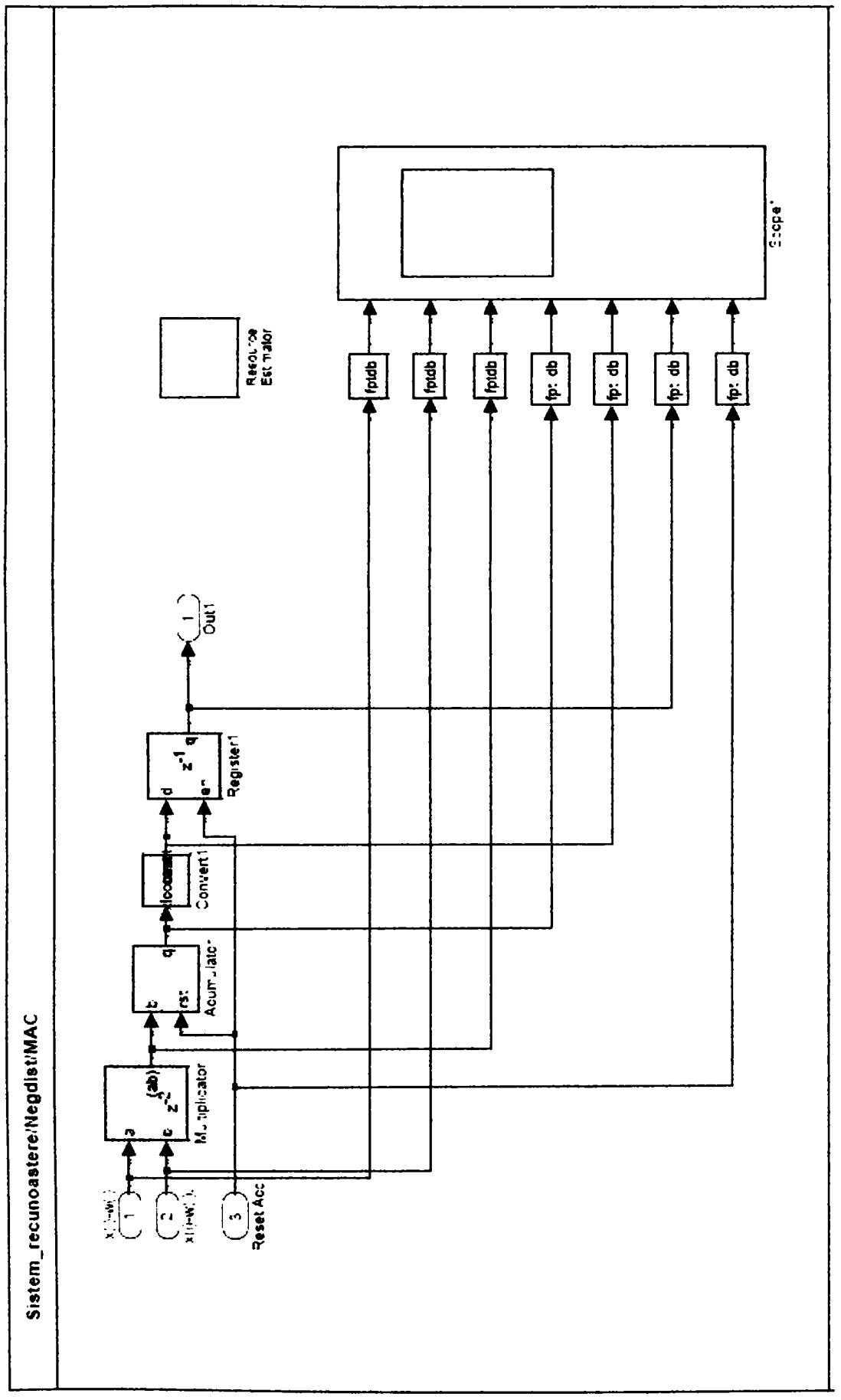

Sistem_recunoastere/Bloc comanda 2
Resource Estimator
Scope2
fpdbl
1
Ad. data A
2
Ad. data B
3
Nr. adr. pond.
4
Reset
z-2
Delay1
z-5
Delay
z-9
Delay3
k =6
k =7
Relational2
Relationa1
Numarator adrese date (RAM)3
Numarator adrese date (RAM)
Numarator adrese coeficienti (ROM)
out

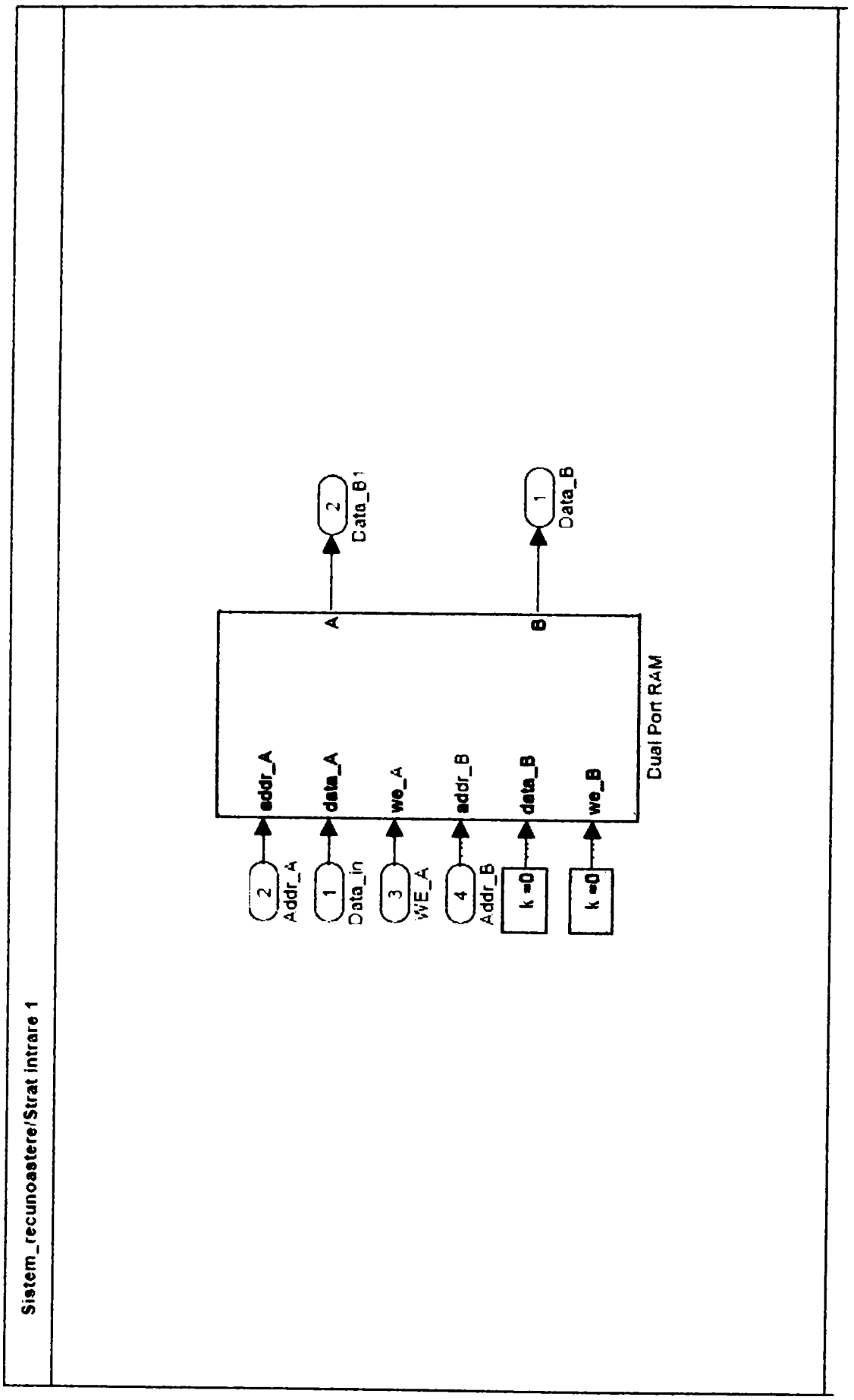

Sistem_recunoastere/Calcul eroare 1
Eror
Nr. erori
Nr erori
Numar erori
en out
Scope2
Vector_tinta
simout1
To Workspace1
Eror
Relational
Date tinta
[t matco (di)]
From Workspace
db/fot
Delay
Down Sample
In1
Disregard Subsystem For Generation

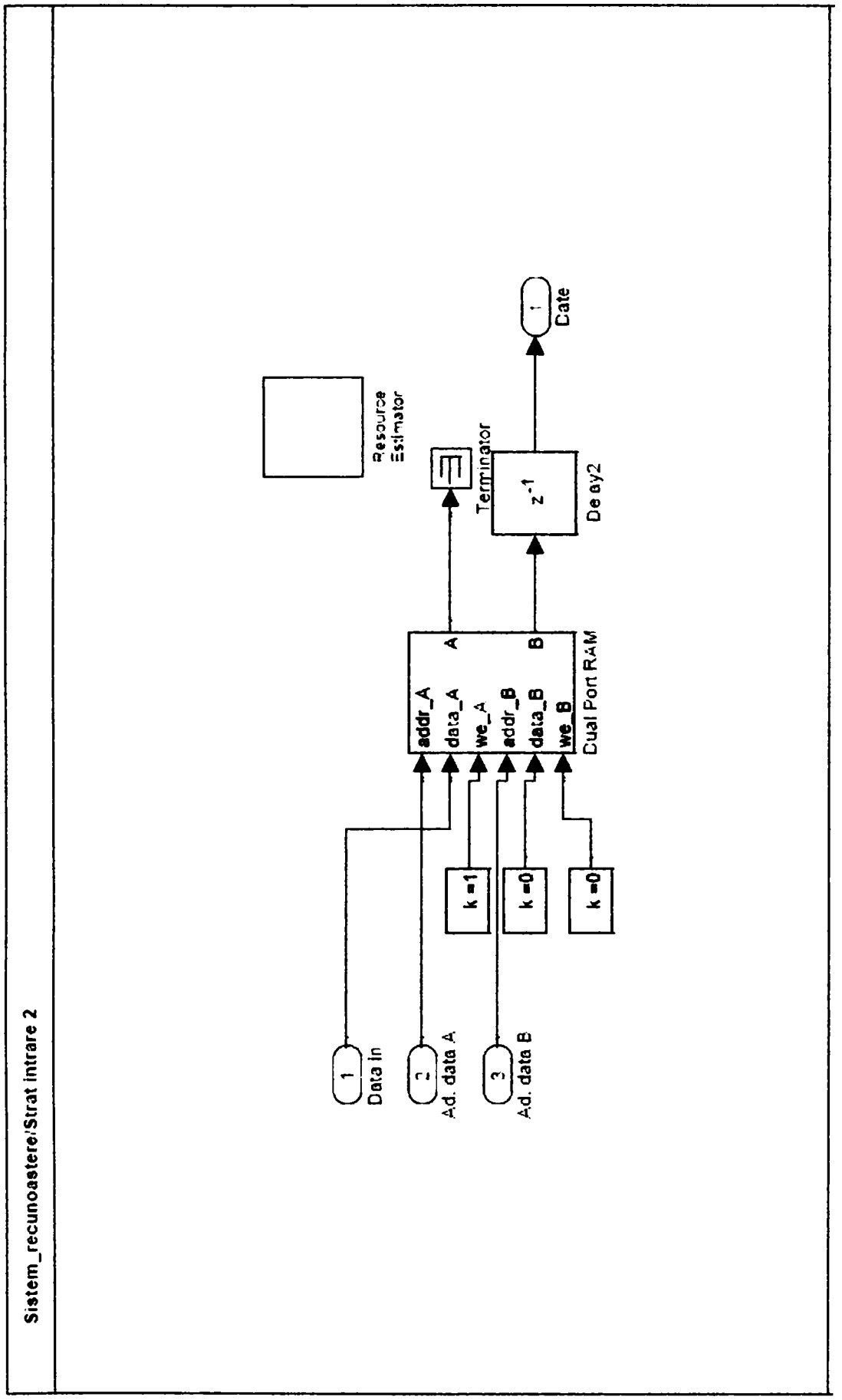

Sistem_recunoastere/Calcul eroare 2
To Workspace1
Scope2
To Workspace
Down Sample1
Down Sample
Delay
Delay1
In1
From Workspace
Disregard Subsystem For Generation
Vector_tinta
Numar erori
Erori
fptdbl

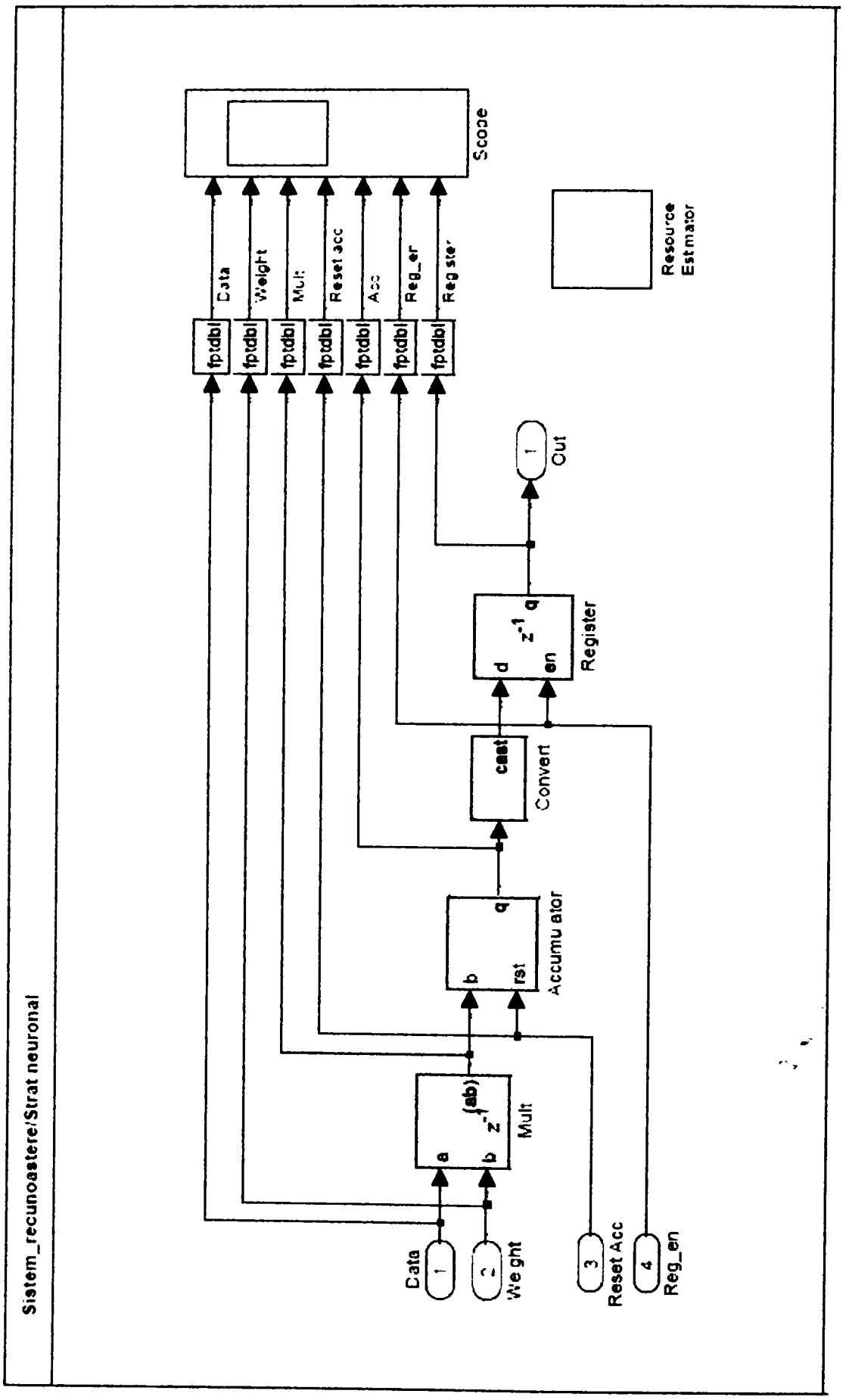

Sistem_recunoastere/Functia de activare competitiva
Scope1
Register1
Counter
Relational
Register2
Delay
Logical
Relational1
Constant
Resource Estimator
Reset
Neuron castigator


Sistem_recunoastere/Negdist
Scope1
MAC
Add5_b
a
b
a+b
Date
Ponsen
Reset
1
2
3


Sistem_recunoastere/Negdist/MAC
Scope
fptdb
fptdb
fptdb
fp: db
fp: db
fp: db
fp: db
Out1
Register1
Convert1
Acumulator
Multiplicator
Reset Acc


Sistem_recunoastere/Strat intrare 1
Dual Port RAM
addr_A
data_A
we_A
addr_B
data_B
we_B
A
B
2
Data_B1
1
Data_B
2
Addr_A
1
Data_in
3
WE_A
4
Addr_B
k =0
k =0


Sistem_recunoastere/Strat intrare 2
Resource Estimator
Terminator
Cate
z-1
Delay2
A
B
addr_A
data_A
we_A
addr_B
data_B
we_B
Dual Port RAM
k =1
k =0
k =0
1
Data in
2
Ad. data A
3
Ad. data B


Sistem_recunoastere/Strat neuronal
Scope
Data
Weight
Mult
Reset Acc
Acc
Reg_er
Register
fpt.dbl
Resource Estimator
Out
1
d
z-1
q
en
Register
cast
Convert
b
q
rst
Accumulator
a
b
(ab)
z-1
Mult
Data
1
Weight
2
Reset Acc
3
Reg_en
4