Universitatea "Transilvania" din Braşov
FACULTATEA DE INGINERIE ELECTRICĂ ŞI ŞTIINŢA CALCULATOARELOR

TEZĂ DE DOCTORAT

SISTEME ADAPTIVE CU REȚELE NEURONALE ARTIFICIALE NEUROMORFE. REALIZĂRI CU DISPOZITIVE HARDWARE RECONFIGURABILE

Doctorand:
ing. László BAKÓ

Conducător ştiinţific:
prof. dr. ing. Iuliu SZÉKELY

2009

Cuvânt înainte

Domeniul Inteligenței Artificiale și a mașinilor cu autoinstruire constă în sisteme artificiale și tehnici care manifestă un comportament „inteligent" și pot învăța din exemple. O gamă variată de algoritmi care sunt utilizați în acest domeniu de cercetare sunt inspirați de funcționarea cortexului mamiferelor. Creierul este format dintr-o rețea mare de celule, numite neuroni, care sunt conectate între ele într-un mod complicat. Rețele de funcții non-liniare simple, numite Rețele Neuronale, surprind unele concepte computaționale prezente și în creier. În ultimii ani, în aceste modele au fost incluse din ce în ce mai multe proprietăți de natură biologică. Aceste cercetări se concentrează pe Rețele Neuronale Pulsative (RNP), rețele de sisteme cu dinamică simplă, care surprind caracteristicile temporale simple ale neuronilor biologici, care comunică informația printr-o serie de impulsuri. Aceste serii de impulsuri sunt secvențe de evenimente discrete în timp, în care doar momentul impulsului reprezintă informație transmisă, nu și amplitudinea acestuia. Cercetările recente au arătat că aceste rețele sunt mai puternice și mai eficiente decât rețelele de tip analog folosite înainte, și recent au fost construite câteva aplicații foarte convingătoare cu ajutorul acestor rețele. Deoarece informația este reprezentată în timp, RNP-urile pot procesa informația temporală într-un mod intrinsec. Modelele mai exacte din punct de vedere biologic, folosite în RNP-uri, ne permit să tragem concluzii asupra funcționării creierului si a rețelelor biologice, pe baza rezultatelor obținute prin folosirea modelelor RNP. De asemenea, sugestii pentru construirea de sisteme artificiale pot fi obținute în mod direct din studiile neurobiologice.

Deși RNP-urile au multe proprietăți benefice, și acestea au fost demonstrate cu succes de câteva aplicații și implementări, crearea de aplicații de succes folosind RNP este încă foarte greoaie din punct de vedere ingineresc. În continuare subliniem câteva probleme de actualitate ale RNP: codificarea informației în serii de impulsuri, antrenarea supervizată a RNP, și implementările hardware. Scopul primordial al acestei teze este de a oferi tehnici și instrumente de implementare hardware care ajută RNP-urile să fie general aplicabile ca instrumente inginerești.

Există ipoteze multiple asupra modului în care neuronii biologici codifică informația în serii de impulsuri. Avem două ipoteze principale bazate pe idei opozante: momentul exact al impulsurilor nu este important, deoarece toată informația este înmagazinată în rata de activare și la cealaltă extremă: toată informația este înmagazinată în momentul exact al impulsurilor. O schemă de codificare care apropie diferența de concept dintre aceste două scheme este codificarea cu filtre: informația codificată într-o serie de impulsuri poate fi extrasă cu succes folosind un filtru liniar. Din punct de vedere ingineresc însă, codificarea informației într-o serie de impulsuri, folosind această schemă, nu este deloc simplă.

O problemă foarte importantă a RNP-urilor este lipsa regulilor de învățare supervizate, general aplicabile, care să poată antrena RNP-urile în mod robust și rapid. Există o singură regulă de învățare supervizată, dar aceasta se poate aplica doar în cazul unei scheme de codificare specifice, care de fapt ignoră natura temporală intrinsecă a RNP. De asemenea nu are o convergență de antrenare robustă, este înceată și necesită o topologie de rețea specială, destul de mare.

Putem îmbunătăți viteza de convergență și cerințele de rețea ale acestei reguli prin antrenarea tuturor parametrilor neuronilor pulsativi. Celelalte probleme rămân nerezolvate însă. Din acest motiv vom introduce o nouă regulă de învățare, bazată pe aproximarea analogică a unui neuron pulsativ. Această regulă poate fi aplicată când folosim o versiune generalizată a schemei de codificare a momentelor de activare, converge în mod rapid și robust. Necesită doar topologii de rețea mici și este ușor implementabilă în hardware digital.

La prima vedere se pare că datorită naturii mai complexe a ecuațiilor necesare pentru calcularea RNP, acestea vor necesita mai mult timp de calcul decât rețelele tradiționale analogice. Vom demonstra că RNP-urile sunt bine adaptate pentru a fi implementate pe hardware digital. Mai multe proprietăți, cum ar fi natura binară, pulsativă a comunicației pot fi eficient implementate în hardware. S-a demonstrat deja, că se pot obține creșteri mari în viteză și în eficiența de consum prin implementarea de RNP în hardware analogic. Pe de altă parte, ne vom concentra asupra hardware-ului digital, deoarece folosind dispozitive digitale reconfigurabile, este posibilă implementarea designului RNP real în hardware, chiar și cu reconfigurare parțială în timpul funcționării când este nevoie. Vom arăta că există mai multe arhitecturi hardware posibile pentru a implementa RNP, fiecare având alt domeniu și alte caracteristici de viteză. Acest lucru permite proiectantului să aleagă o implementare optimă după ce s-au luat în considerare diverse constrângeri impuse de aplicația dată. Sunt posibile accelerări foarte mari sau implementări foarte mici, în funcție de ce necesită aplicația dată.

Pentru a demonstra tehnicile prezentate în această teză se vor prezenta mai multe aplicații inginerești și aplicații de test (benchmark).

Aș dori să exprim mulțumiri pe această cale în primul rând d-lui Prof. dr. ing. Iuliu Székely care mi-a fost alături cu sfaturi, m-a îndrumat, sprijinit și m-a ajutat pe toată durata elaborării tezei în calitate de conducător științific.

Mulțumesc domnilor Prof. dr. ing. Dávid László și Conf. dr. ing. Márton László pentru sprijinul acordat în rezolvarea problemelor ivite pe durata elaborării tezei.

De asemenea, mulțumesc colectivului Catedrei de Inginerie Electrică din cadrul Facultății de Științe Tehnice și Umaniste a Universității Sapientia, pentru sprijinul acordat și pentru atmosfera de lucru indispensabilă realizării tezei.

Mulțumesc Institutului Programelor de Cercetare al Fundației Sapientia și Fundațiilor Communitas și Eurotrans pentru bursele acordate, respectiv CNCSIS UEFISCSU, pentru acordarea grantului de cercetare TD în decursul finalizării stagiului doctoral.

În final, dar nu în ultimul rând, doresc să mulțumesc membrilor familiei mele în egală măsură, pentru încurajare, răbdare și pentru suportul fără de care nu aș fi reușit să parcurg acest drum.

CUPRINS

LISTA FIGURILOR

1. INTRODUCERE

1.1. Inteligenţă artificială şi tehnici de machine learning

Cercetătorii au urmărit de multă vreme dezvoltarea conceptului de maşini instruibile şi inteligente. La început s-a considerat că, deoarece orice funcţie se poate calcula cu un dispozitiv arbitrar compatibil cu o maşină Turing, şi inteligenţa trebuie să fie calculabilă cu aceleaşi dispozitive. În urma acestui fapt a luat naştere în anii 1960 domeniul de cercetare al Inteligenţei Artificiale (IA). Scopul IA a fost de a produce programe inteligente, fără a se afla cum reuşeşte acest lucru creierul biologic, considerându-se, că numai comportamentul unui sistem este important şi nu procesul în sine care generează acest comportament. În prima fază rezultatele au fost promiţătoare, deşi performanţele au fost slabe. S-a afirmat însă, că această performanţă slabă se va îmbunătăţi drastic odată cu creşterea exponenţială a puterii de calcul prevăzută de legea lui Moore. În anii 1980 a devenit evident, că această creştere de performanţă foarte aşteptată nu se realiza, chiar după 20 de ani de creştere exponenţială a puterii de calcul. În multe cazuri, la probleme ce par simplu de rezolvat unui subiect uman, soluţiile cu IA întâmpinau greutăţi multiple. Exemple semnificative erau recunoaşterea vorbirii, robotică, procesarea imaginilor, prelucrarea semnalelor biologice, controlul sistemelor nestaţionare şi/sau neliniare. Pentru a putea rezolva o astfel de problemă, în general impunem unui sistem de procesare două cerinţe importante, comportament neliniar şi abilitatea de învăţa din exemple.

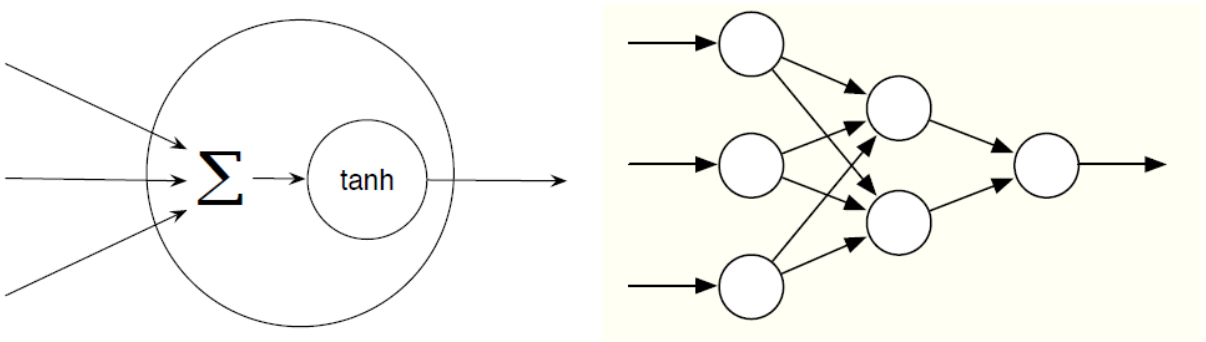


Figura 1.1 Primele modele de neuroni şi reţele neuronale
(stânga – neuron analog cu sumator a intrărilor şi o funcţie neliniară,
dreapta – reţea neuronală feed-forward cu trei straturi, putând reprezenta numai funcţii independente de timp)

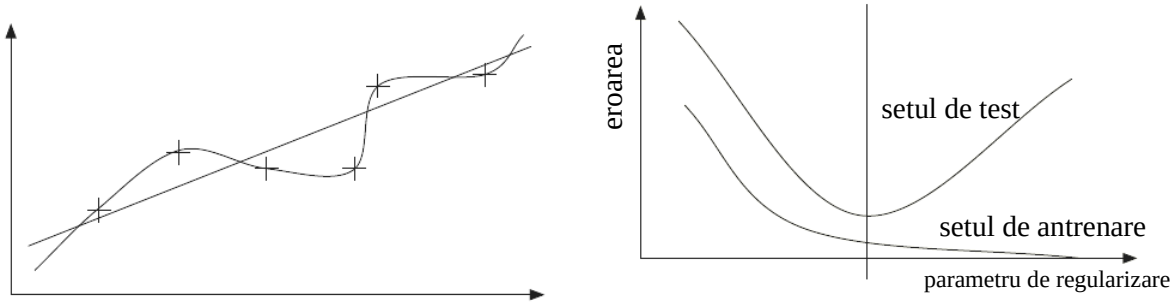
Cercetătorii au început să se orienteze după paradigme şi arhitecturi de calcul alternative pentru a crea sisteme inteligente, în mare măsură inspirate de unicul sistem cunoscut a fi cu adevărat inteligent: creierul biologic. Astfel a luat fiinţă domeniului de cercetare al Reţelelor Neuronale Artificiale (RNA), care de fapt a fost abordat şi în anii 1960, dar a pierdut rapid interes datorită rezultatelor negative în non-separabilitate utilizând neuroni individuali. Acestea sunt modele foarte simplificate ale structurilor prezente în creierul natural, în cele mai multe cazuri compunându-se din numai câteva straturi de noduri simple de calcul interconectate (astfel fiind foarte departe de realitatea biologică). Utilizând aceste structuri (vezi Figura 1.1), însă, a devenit posibil să se *înveţe* un anumit comportament în loc de a fi programat. Acest lucru a fost foarte important, deoarece multe probleme complexe nu sunt definite clar de un model, cum ar fi ecuaţii analitice sau chiar un algoritm, dar pot fi descrise de un set de măsurători zgomotoase ale sistemului împreună cu răspunsurile predefinite.

Modelele de Reţele Neuronale Artificiale sunt ecuaţii neliniare simple, ce au fost parţial inspirate de modul în care cercetătorii anilor 1940 presupuneau că funcţionează neuronii biologici. există multe variante, dar ecuaţia de bază este:

$$a_j = \tanh\left(\sum_i w_{ij} a_i\right) \qquad \textit{Ec. 1}$$

unde ieşirea a_j a neuronului j este calculată de o funcţie neliniară (cum este funcţia tangenta hiperbolică) a sumei ponderate a activărilor a neuronilor sau intrărilor la care este conectat. A fost arătat de

(Cybenko, 1989) că orice reţea cu două straturi creată din astfel elemente cu funcţii neliniare simple, sunt capabile a aproxima orice funcţie continuă cu o precizie arbitrară.



[a] învăţarea din exemple [b] supra-învăţarea (overfitting)

Figura 1.2 Reprezentarea schematică a conceptului de învăţare în reţele neuronale artificiale

Ideea învăţării este bazată pe faptul, că este posibil să se antreneze un sistem să răspundă corespunzător pentru un set de date de antrenare iar apoi demonstreze acelaşi lucru (cu o eroare neglijabilă) şi pentru un set de date de test, neîntâlnite apriori. Această proprietate se numeşte generalizare şi este un aspect crucial al realizării sistemelor inteligente. Ca urmare a introducerii acestei proprietăţi, putem învăţa funcţii complexe fără a cunoaşte explicit funcţia respectivă. Învăţarea supervizată este realizată prin specificarea ieşirii impuse pentru fiecare set de date de intrare, ajustând parametrii funcţiei (proces numit şi antrenare) în aşa fel, încât o metrică de eroare calculată pe setul de date de antrenare să se minimizeze. Un lucru important de remarcat este capacitatea de control: în cazul în care antrenăm o funcţie cu mulţi parametrii utilizând un set de date de antrenare limitat este posibil să învăţăm şi zgomotul prezent în acest set de date. Această problemă, numită şi supra-învăţare (over-fitting) apare, când numărul gradelor de libertate (a parametrilor) funcţiei ce este învăţată este prea mare pentru complexitatea funcţiei care a generat setul de antrenare. Rezolvarea cea mai simplă constă în a limita numărul (efectiv) parametrilor funcţiei până la o valoare pentru care rezultă o generalizare optimă (acest lucru se poate realiza prin stoparea antrenării mai devreme, modificare arhitecturii, etc.) pe un set de date separat, numit de obicei setul de validare.

Mai recent, cercetarea ce a pornit ca reţele neuronale a fost fertilizată de domenii conexe, cum ar fi statistică şi control, astfel obţinându-se metode de învăţare avansate, bazate pe principii matematice puternice. În anii 1990 s-a pus în evidenţă faptul, că deşi RNA sunt bine înţelese ele nu pot spune multe despre modul de funcţionare al creierului natural. Această realizare a dus la o intensificare a interesului cercetătorilor în a găsi modele mai plauzibile din punct de vedere biologic, care să ia în considerare natura pulsativă a neuronilor corticali. Astfel, s-a studiat capacitatea de învăţare a acestor modele mai detaliate, numite Reţele Neuronale Pulsative (RNP) (spiking neural networks), unde informaţia este transmisă prin şiruri de impulsuri. Primele astfel de rezultate (Maas, 1997) (Maas & Natschläger, 1997) au arătat, că unele dintre aceste modele neuronale complexe sunt capabile de performanţe de calcul mai mari decât modelele neuronale clasice.

S-a demonstrat (Delorme & Thorpe, 2001), de asemenea, că se pot implementa cu RNP aplicaţii inginereşti impresionante, cu un necesar de putere de calcul foarte redus. Aici însă, s-au utilizat modele foarte mult simplificate pentru a reprezenta proprietăţile specifice ale RNP pe arhitecturi clasice bazate pe unităţi centrale de prelucrare (CPU). Pe de altă parte, există şi o direcţie de cercetare mai apropiată de neuro-biologie, unde există o nevoie acută de tehnici inteligente pentru a procesa multitudinea de semnale măsurate de la neuroni reali, biologici cu metode in-vitro şi in-vivo. În acest caz, scopul este de a decoda informaţiile stocate în aceste trenuri de impulsuri şi de a înţelege ce procesează acei neuroni. Cercetarea metodelor de codare/decodare a informaţiilor în impulsuri este importantă din acest punct de vedere.

Unul dintre scopurile generale al acestei teze, este de a arăta, că aceste modele neuronale mai complexe pot fi utilizate în aplicaţii inginereşti, obţinându-se performanţe de vârf şi că aceste modele sunt foarte pliabile unor implementări pe suporturi hardware digitale, depăşind chiar modelele clasice de RNA. În continuarea acestei introduceri se va face o descriere mai detaliată a RNP, cu o privire generală asupra neajunsurilor acestora din punctul de vedere ingineresc, respectiv se va deschide tema implementărilor hardware.

1.2. Reţele neuronale artificiale

Structura creierului uman se compune dintr-o reţea complexă de neuroni care se interconectează între ei în mod paralel prin intermediul unor axoni, respectiv dendrite. Schimbul de informaţie între diferiţii neuroni se realizează prin intermediul aşa-numitelor sinapse. Funcţionarea creierului uman se caracterizează prin fenomene electro-chimice relativ simple. S-a pus problema explicării modului în care aceste reţele neuronale complexe, dar constituite din elemente de procesare relativ simple, pot să prezinte performanţe notabile. Aceste idei au condus la crearea diferitelor modele matematice ale neuronilor "artificiali".

Prin studierea reţelelor neuronale artificiale putem încerca înţelegerea operaţiilor esenţiale ce au loc în reţeaua de neuroni dens interconectată a sistemului nervos central natural. Primul model neuronal propus de (McCulloch & Pitts, 1943) a fost bazat pe neuroni binari simplificaţi, care implementează o funcţie de prag simplă, conform căreia un neuron este sau activ sau inactiv, stări calculate pe baza sumei ponderate a stărilor neuronilor la care acesta este conectat. În acest scop conexiunile dintre neuroni sunt direcţionate (de la neuronul *i* la neuronul *j*) şi au ataşate câte o pondere. Legăturile sunt deci nişte conexiuni care transportă semnale excitatorii sau inhibitorii de la un element de procesare la altul. Dacă suma ponderată a neuronilor *i* conectaţi la neuronul *j* depăşeşte o valoare de prag, atunci starea neuronului *j* este activ altfel inactiv. Remarcabil, reţele construite din astfel de elemente de calcul simple, pot implementa o gamă largă de funcţii matematice, corelând stările de intrare cu stările de ieşire şi variind ponderile conexiunilor pe baza unor algoritmi specifici, astfel reţelele de neuroni artificiali învăţând funcţiile respective.

1.3. Fundamente biologice

1.3.1. Neuronul pulsativ ideal

Un neuron natural tipic se poate împărţi în trei componente funcţional diferite: dendrite, soma (corpul celulei) şi axonul. Generalizând, se poate afirma, că dendritele sunt intrările, care achiziţionează semnale de ieşire de la alte celule pe care le transmite către soma neuronului (Figura 1.3). Soma este unitatea centrală de prelucrare, care realizează un pas important de prelucrare neliniară: dacă ieşirea totală calculată de acesta depăşeşte o valoare de prag, se generează un semnal de ieşire – un impuls de potenţial (Gerstner & Kistler, 2002). Acest semnal este transmis spre receptorii altor celule de către axon, dispozitivul de ieşire a neuronului.

Conexiunea dintre doi neuroni este denumită sinapsă. Să presupunem, că un neuron transmite un semnal printr-o sinapsă. Ne vom referi la acest neuron ca neuron pre-sinaptic, iar la cel care va recepţiona semnalul ca neuron post-sinaptic. În cazul unei reţele neuronale naturale, ca cea a unui creier uman, un singur neuron pre-sinaptic poate fi conectat până la 10^4 neuroni post-sinaptici. Deseori o sinapsă se poate afla în imediata vecinătate a axonului dar se poate afla chiar şi la câţiva centimetri depărtare.

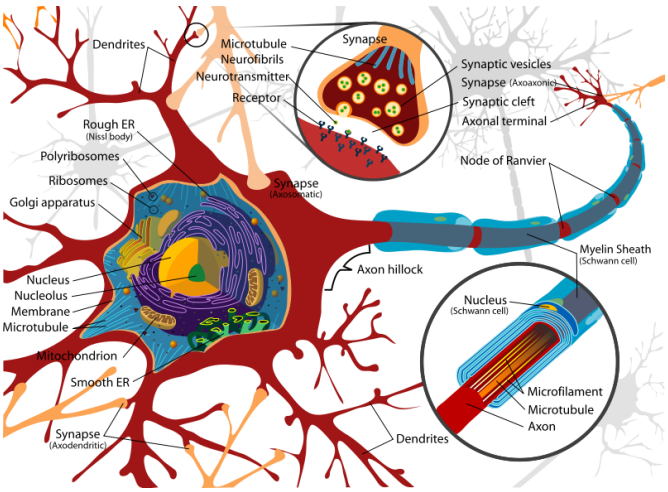


Figura 1.3 Structura unui neuron natural

1.3.2. Impulsuri de activare

Impulsurile de activare sunt impulsuri electrice de scurtă durată, dar care se pot sesiza plasând un electrod sensibil foarte aproape de soma sau axonul neuronului natural. Amplitudinea caracteristică a acestor impulsurile de activare este de aproximativ 100 mV, iar durata lor este de 1-2 ms. Forma impulsului nu se modifică, pe durata propagării acestuia de-a lungul axonului. Un şir de astfel de impulsuri stereotipice, emise

de un neuron, se pot repeta la intervale de timp regulate sau neregulate. Deoarece toate impulsurile emise de către un neuron sunt asemănătoare, forma acestor impulsuri nu este purtătoare de informaţie.

Important este numărul şi temporizarea acestor impulsuri. Impulsurile sau potenţialele de activare sunt elementul de bază în transmisia de date dintre neuroni. Potenţialele de activare se pot delimita cu acurateţe în cadrul unui astfel de şir de impulsuri. Distanţa minimă dintre două activări (două impulsuri) determină perioada absolută de reprimare a neuronului. Perioada relativă de reprimare este cea care urmează celei anterior menţionate, în care este dificil, dar nu imposibil excitarea neuronului până la activare.

1.3.3. Sinapsele

Punctul în care axonul unui neuron pre-sinaptic se conectează la dendrita unui neuron post-sinaptic se numeşte sinapsă. În cazul natural, sinapsele cele mai des întâlnite sunt cele chimice.

La o astfel de sinapsă, terminalul (capătul) axonului se apropie extrem de mult de neuronul post-sinaptic, (la o distanţă de ordinul micrometrilor) spaţiul rămas între membranele celulelor pre- şi post-sinaptice numindu-se interspaţiu sinaptic. (Figura 1.4). Dacă un potenţial de activare ajunge la un interspaţiu sinaptic, determină startul unui proces complex de prelucrare chimică, care are ca efect eliberarea de către terminalul pre-sinaptic în interspaţiu a unei substanţe speciale, numită neuro-transmiţător (Trappenberg, 2002). Moleculele acestei substanţe ajungând la senzorii membranei post-sinaptice deschid canale specifice, prin care anumiţi ioni, aflaţi în fluidul intracelular din interspaţiul sinaptic, pot pătrunde în interiorul celulei. Ca urmare a acestei migrări de ioni, potenţialul membranei post-sinaptice se modifică, astfel, semnalul chimic se transformă din nou în semnal electric.

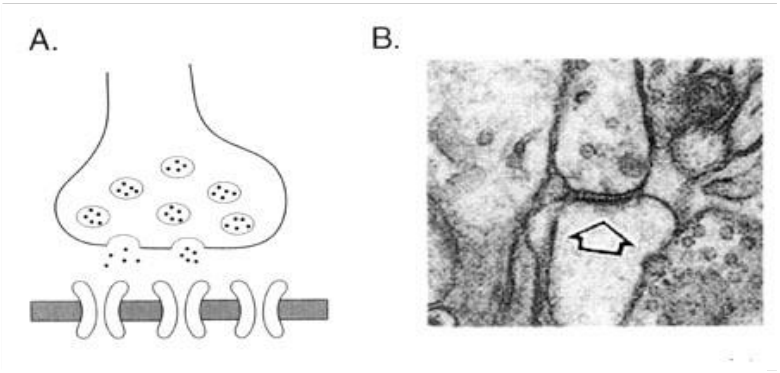


Figura 1.4 ***A.*** *Schema unei sinapse chimice;* ***B.*** *Fotografia electro-microscopică a unei sinapse (Trappenberg, 2002)*

1.3.4. Dinamismul neuronal

Efectul unei activări pre-sinaptice asupra neuronului post-sinaptic se poate măsura cu ajutorul unui electrod intracelular, care măsoară diferenţa de potenţial *u(t)* dintre interiorul celulei şi mediul înconjurător.

Această diferenţă de potenţial este numită potenţialul membranei. În lipsa unui impuls de intrare, neuronul este în stare latentă, având un potenţial de membrană specific acestei stări. După sosirea uni impuls de activare la intrare potenţialul membranei se modifică ca apoi să revină la potenţialul de repaus (vezi Figura 1.5).

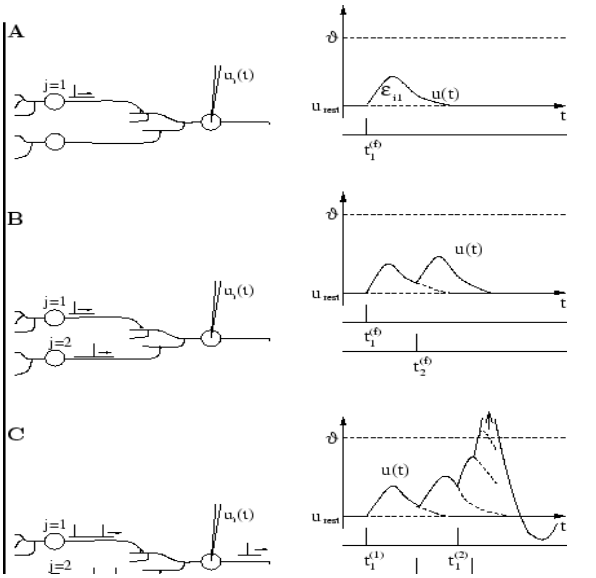


Figura 1.5 Neuronul post-sinaptic nr. i recepţionează impulsuri de activare de la neuronii pre-sinaptici j=1,2. Ambii neuroni emit câte un impuls care modifică potenţialul membranei la neuronul post-sinaptic, modificare care se poate măsura cu un electrod ca diferenţa de potenţial $u_i(t) - u_{rest}$. A. Variaţia potenţialului de membrană post-sinaptic ($t - t_1(f)$). ca răspuns la impulsul de activare excitator recepţionat de la neuronul pre-sinaptic j = 1 B. Sosirea unui al doilea impuls de activare de la neuronul pre-sinaptic j = 2, imediat după cel de la neuronul j = 1, determină un al doilea salt în potenţialul membranei post-sinaptice, care se adună cu primul. C. Dacă $u_i(t)$ atinge pragul , neuronul post-sinaptic emite, la rândul lui, un impuls de activare.

Dacă această variaţie are gradientul pozitiv, atunci este vorba de o sinapsă excitatoare, iar în cazul negativ de una inhibitorie. În stare latentă, sau de repaus, membrana celulei este deja polarizată negativ, având o valoare de aproximativ –65 mV. Efectul unei intrări sosite pe o sinapsă excitatoare, scăderea stării de polarizare negativă a membranei se numeşte depolarizare, iar efectul invers, care duce la accentuarea polarizării negative se numeşte hiperpolarizare.

1.3.5. Pragul de activare sau aprindere şi potenţialul de activare

Să presupunem că avem doi neuroni pre-sinaptici *j = 1, 2*, fiecare transmiţând câte un impuls către neuronul post-sinaptic *i*. Neuronul *j* = 1 se aprinde la momentele de timp $t_1^{(1)}$, $t_1^{(2)}$,..., iar neuronul *j* = 2 la $t_2^{(1)}$, $t_2^{(2)}$,.... . Fiecare impuls de intrare generează separat câte un potenţial post-sinaptic ε_{i1}.şi ε_{i2}. Dacă avem numai câteva impulsuri de intrare, variaţia potenţialului post-sinaptic este aproape liniară, ea putând fi echivalată cu suma salturilor de potenţial post-sinaptic (Figura 1.5B):

$$u_i(t) = \sum_j \sum_f \varepsilon_{ij}\left(t - t_j^{(f)}\right) + u_{repaus} \qquad \textit{Ec. 2}$$

Altfel, dacă avem multe impulsuri de intrare sosite într-un interval de timp relativ scurt, atunci variaţia menţionată îşi pierde liniaritatea. De îndată ce potenţialul post-sinaptic atinge o valoare critică, forma acestui semnal va lua cu o totul altă formă decât cea a sumării impulsurilor intrate, şi ea devenind un impuls abrupt, cu o amplitudine de circa 100 mV. Acest potenţial de aprindere se va propaga de-a lungul axonului neuronului i către sinapsele altor neuroni din reţea. După evenimentul aprinderii, potenţialul membranei nu se ca întoarce imediat la potenţialul de repaus, ci va trece printr-o fază de hiperpolarizare. Amplitudinea impulsurilor de intrare este în domeniul milivolţilor. Nivelul de prag al potenţialului membranei, care trebuie atins pentru a declanşa starea de aprindere, este situat cu cca. 20÷30 mV deasupra potenţialului de repaus. După cum se poate observa în Figura 1.5C, patru impulsuri de intrare nu sunt suficiente, pentru a atinge potenţialul de prag şi implicit pentru a aprinde neuronul. Acest lucru necesită sosirea a cel puţin 20-50 de impulsuri pre-sinaptice într-un interval de timp scurt.

1.4. Sisteme de control neuronal

În prezent sistemelor de reglare li se cer să posede performanţe dinamice ridicate împreună cu proprietăţi de robusteţe şi utilizare în sisteme din ce în ce mai complexe, cu caracteristici dinamice neliniare. Capacitatea de modelare cu metode matematice pierde teren faţă de o modelare şi reglare inteligentă. În ultimul timp s-a manifestat un larg interes în folosirea modelelor bazate pe structuri biologice şi algoritmi de învăţare pentru modelare şi reglare adaptivă. Aplicaţiile de reglare inteligentă au nevoie de algoritmi care sunt capabili (Eldredge & Hutchings, 1994) să:

- opereze într-un domeniu insuficient de bine definit şi variabil în timp
- se adapteze la schimbări în dinamica sistemului ca şi în efectele mediului
- înveţe informaţii semnificative în mod stabil
- considere câteva restricţii pe dinamica sistemului
- să opereze autonom în mediu haotic cu intervenţie minimă.

Căutarea unui algoritm care să ne furnizeze o metodă universală pentru toate problemele de reglare inteligentă nu este un ţel realist. În ingineria reglării, algoritmii de învăţare au importanţă mare. În general, algoritmii de adaptare sunt bazaţi pe sisteme liniare şi modele liniare, numărul parametrilor care trebuie aleşi determinând flexibilitatea schemelor de adaptare. Reţelele neuronale furnizează şi metode cu care aceşti algoritmi pot fi adaptaţi pentru sisteme neliniare. Majoritatea algoritmilor de învăţare supervizată sunt bazaţi pe metoda gradientului dar recent au apărut strategii de adaptare bazate pe concepte de stabilitate (James-Roxby & Blodget, 2000) (Zhu & Gunther, 1999) (Pérez-Uribe & Sanchez, 1996). În general arhitecturile de modelare şi reglare sunt independente de reţea.
In ultima perioadă se manifestă un mare interes în utilizarea reţelelor neuronale în proiectarea sistemelor de control. În proiectele legate de controlul proceselor industriale foarte des se utilizează reţelele neuronale atât pentru identificarea sistemului, cât şi comanda optimală şi adaptivă a lor. În aceste procese de control se utilizează diferite topologii de reţele neuronale cât şi diferite procedee de antrenare a acestor reţele.

Sistemele de control neuronale se pot clasifica aşa cum urmează:

- control neuronal ***indirect***
- control neuronal ***direct***

Faţă de sistemele de control neuronale directe cele cu control indirect nu sunt în legătură directă cu procesul controlat. În general o reţea neuronală este utilizată să modeleze parametrii procesului sau a regulatorului. Dacă parametrii regulatorului sunt furnizaţi de o reţea neuronală atunci avem de a face cu un regulator auto-tuner care adaptează parametrii regulatorului în conformitate cu comportamentul procesului.

Astfel modelele de control neuronal **indirecte** se pot clasifica:

- control pe bază de model proces utilizând reţele neuronale artificiale
- control bazat pe model invers folosind reţele neuronale artificiale
- control bazat pe autoacordare a parametrilor realizat de reţea neuronală

În schemele de control ***directe*** cu reţele neuronale, reţeaua este folosită ca regulator, şi trimite semnalul de comandă direct către proces. În funcţie de conceptul de proiectare a acestor sisteme amintim:

- modelare regulator tip neuronal
- structură de control fără model
- structură de control bazată pe de model
- structură de control robust bazată pe model

În toate clasele descrise mai sus rolul reţelei neuronale este rezolvarea unei probleme de optimizare bazată pe o funcţie de criteriu *J(w)* care depinde de valorile ponderilor reţelei.
O astfel de funcţie de criteriu se poate modela astfel ca:

$$NN : \min_{w} \{J(w)\} \qquad \textit{Ec. 3}$$

unde *NN*: reprezintă faptul că problema de optimizare implică o reţea neuronală.
O reţea neuronală poate fi privită ca o cutie neagră pentru modelul procesului cum se poate vedea Figura 1.6.

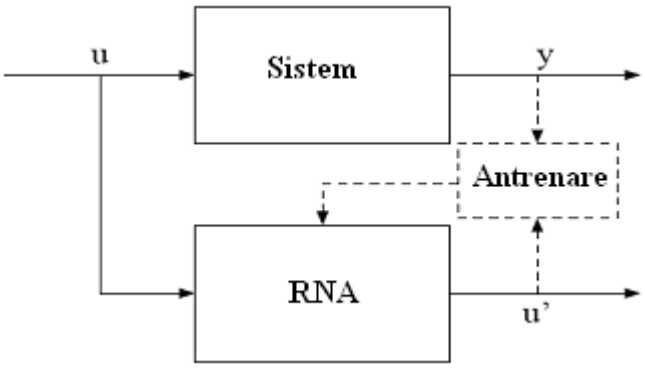


Figura 1.6 Reţea neuronală privită ca o cutie neagră

1.4.1. Control neuronal indirect

Control bazat de modelul procesului, utilizând RNA

Cea mai cunoscută aplicaţie a sistemelor de control cu reţele neuronale este de a utiliza reţeaua neuronală ca un model de proces intrare-ieşire. Reţeaua neuronală este antrenată printr-un algoritm de învăţare supervizată. În mod obişnuit se utilizează ca şi model de sistem modelul NARMAX sau NARX (**N**-neuronal AR-autoregressive MA-moving average X-eroarea estimată) care exprimă funcţii de transfer discrete neliniare.

O alternativă ar fi identificarea sistemului controlat cu o reţea neuronală dinamică Indiferent de structura modelului şi a strategiei de control, modelul cu control neuronal în acest caz poate fi exprimat conform relaţiei din Ec. 4.

$$NN : \min_{w} \{F(y_p - y_n(w,.....))\} \qquad E$$
c. 4

unde y_p reprezintă ieşirea procesului, y_n ieşirea reţelei neuronale, w ponderile reţelei neuronale. $F\{\}$ este o funcţională care măsoară performanţa procesului de optimizare. De obicei F este o integrală din suma erorii de predicţie dintre ieşirea procesului y_p respectiv ieşirea reţelei neuronale y_n. Sunt măsurate, pe o perioadă finită, intrările respectiv ieşirile procesului *{up, yp}*, şi acest set de date măsurate sunt utilizate pentru antrenarea reţelei neuronale.

Deseori, se utilizează pentru funcţia de criteriu:

$$NN : \min_{w} \left\{ \sum_{t} \left| y_p(t) - y_n(t) \right|^2 \right\}; \quad y_n(t) = N(w,...) \qquad \textit{Ec. 5}$$

Odată ce modelul este specificat, se poate utiliza în implementarea regulatorului pe bază de model.

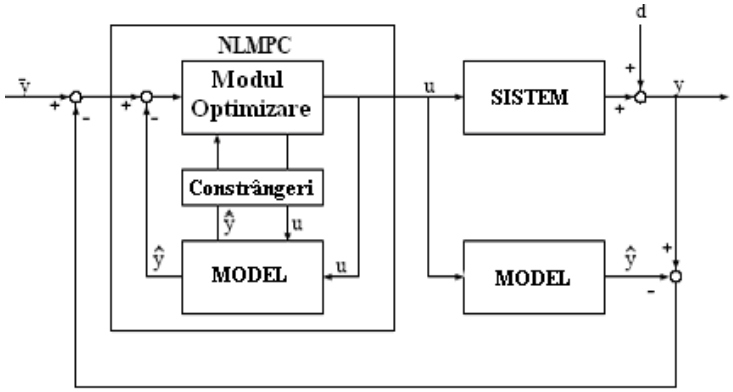


Figura 1.7 O reţea neuronală poate fi incorporată într-o schemă de control predictiv bazat pe model

În stadiul de implementare, de la început modelul de reţea neuronală, nu poate fi utilizat ci trebuie încorporat într-o schemă de control pe bază de model. În procesele industriale chimice reţeaua neuronală este utilizată intr-o schemă de control neliniar cu model predictiv, obţinând un regulator NL MPC. Blocul **Model** din Figura 1.7 reprezintă reţeaua neuronală care este utilizată pentru identificarea procesului controlat şi paralel se desfăşoară şi predicţia stărilor respectiv elaborarea semnalului de comandă pentru eşantionul curent.

De fapt controlul cu model predictiv este tot o problemă de optimizare şi poate fi exprimată prin relaţia:

$$NN : \min_{u} \{F\{y^* - y_n(u,...)\}\} \qquad \textit{Ec. 6}$$

- y^*- ieşirea aşteptată a procesului cu buclă închisă
- y_n–ieşirea estimată (cu modelul de reţea neuronală)
- F funcţia de criteriu pentru evaluarea performanţelor sistemului cu buclă închisă

Problema de optimizare poate fi descrisă prin relaţia din Ec. 7.

$$\min_{u} \left\{ \sum_{t} \left| y^*(t) - y_n(t) - d(t) \right|^2 \right\}; \quad y_n(t) = N(u,...) \qquad \textit{c. 7}$$

unde: y* – reprezintă valorile pentru traiectoria impusă,
d(t)- valoarea perturbaţiei estimate.

1.4.2. Model de reţea neuronală parţială sau parametrică

În multe cazuri este posibil să avem informaţii concrete despre model cum ar fi structura modelului, sau fenomene fizice particulare bine cunoscute care descriu funcţionarea unei părţi a modelului. În acest caz nu este necesar să realizăm în totalitate modelul ca şi o cutie neagră. Dacă

structura cu modelul procesului ne stă la dispoziţie, valorile pentru parametrii asociaţi modelului pot fi determinate cu o reţea neuronală. Aceşti parametri pot fi constante de timp, factori de amplificare, factori de întârziere sau parametrii fizici cum ar fi rata de difuziune, coeficienţi de material etc. Când structura de model nu este cunoscută apriori, reţeaua neuronală poate fi antrenată pentru a selecta elemente din structura de model dintr-un set de modele predefinite. Aceste elemente pot fi compuse într-o structură utilizabilă. În alte cazuri unde structura modelului este parţial cunoscută, reţeaua neuronală se poate folosi împreună cu un model parţial şi astfel procesul poate fi mai bine modelat (Figura 1.8).

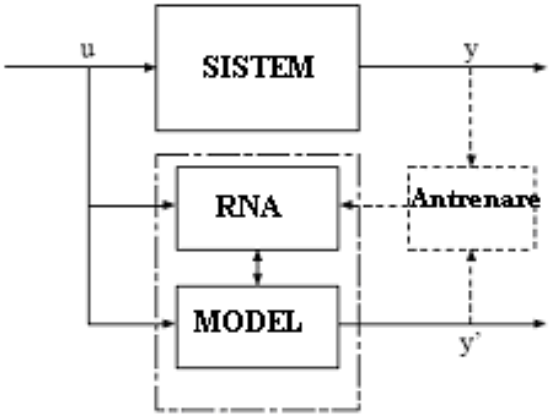


Figura 1.8 Reţeaua neuronală ca estimator de parametrii

Problema de modelare cu reţea neuronală parţială sau parametrică se poate formula astfel:

$$NN : \min_{w} \left\{ F\left\{ y_p - y_m(\theta,...) \right\} \right\}; \quad \theta = N(w,...) \qquad \textit{Ec. 8}$$

- unde y_m este o ieşirea estimată de model

θ - parametrii procesului, informaţii structurale despre model, sau alte elemente pentru completarea modelului.

Din punct de vedere al controlului bazat pe model, această aproximare în esenţă este identică cu modelul de reţea neuronală full black box cu excepţia faptului că reţeaua neuronală nu descrie direct comportamentul procesului.

Modelul sistem invers cu RNA

În acest caz reţeaua neuronală este antrenată pentru a obţine un model invers al procesului Figura 1.9. Ieşirea procesului este intrarea reţelei neuronale, iar intrarea procesului corespunde cu ieşirea reţelei neuronale. În general problema de optimizare se poate formula:

$$NN : \min_{w} \left\{ F\left(u_{p^{-1}} - u_n(w,.....) \right) \right\} \qquad \textit{Ec. 9}$$

Unde $u_{p^{-1}}$ –semnalul de intrare a procesului

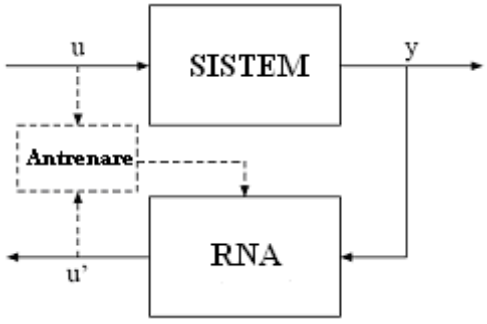


Figura 1.9 Control cu reţea neuronală bazată pe modelul invers al procesului

Pentru o valoare impusă ieşirii sistemului y semnalul de comandă se obţine din ieşirea reţelei neuronale: *(u=u')*

$$u^* = N(y^*,...) \qquad \textit{Ec. 10}$$

Modelul invers există numai când procesul se comportă ca un sistem minimal, altfel această structură este inaplicabilă.

În principiu, o reţea neuronală poate învăţa cu anumite restricţii dinamica inversă a unui sistem. Modelul invers este utilizat într-un mod similar unui controller MPC. În practică, şi în special la un model dinamic discret, modelul invers nu este capabil să înveţe dinamica inversă pe tot spaţiul de comandă inversă.

Modelul invers al sistemelor poate să nu existe dacă sistemul este cu timp mort.

Se poate obţine un model invers a procesului utilizabil cu restricţii impuse pe intrare respectiv parametri de model invers.

Model cu reţea neuronală cu autoacordare.

În acest caz reţeaua neuronală poate fi utilizată pentru estimarea parametrilor de acordare a regulatorului a cărui structură este cunoscută apriori. Deseori estimatorul parametrilor de acordare a regulatorului este denumit autoacordare.

Problema de optimizare în cest caz poate fi formulat astfel:

$$NN : \min_{w} \{F\{\eta^* - \eta_n(w,...)\}\} \qquad \textit{Ec. 11}$$

$\eta*$ - parametrii regulatorului

η_n - parametrii estimaţi de reţeaua neuronală

Parametri η nu se pot unic determina din caracteristica procesului. Aceşti parametrii depind de caracteristicile sistemului închis dorit.

De obicei, parametrii regulatorului se obţin din optimizarea funcţiei de criteriu corespunzătoare sistemului în buclă închisă

$$NN : \min_{\eta} \{F\{y^* - y_{p/m}(u,...)\}\}; \quad u = C(\eta,....) \qquad \textit{Ec. 12}$$

unde C este un regulator cu structură cunoscută $y_{p/m}$ reprezintă un model sau un proces ce poate fi inclus in sistemul cu buclă închisă pentru obţinerea sistemului de reglare cu un C dorit. Avantajul acestei metode este că reţeaua neuronală din buclă se poate antrena în condiţii de simulare. Pentru autoacordare în buclă deschisă este suficientă o simulare în buclă deschisă, altfel necesită o simulare în buclă închisă, Figura 1.10.

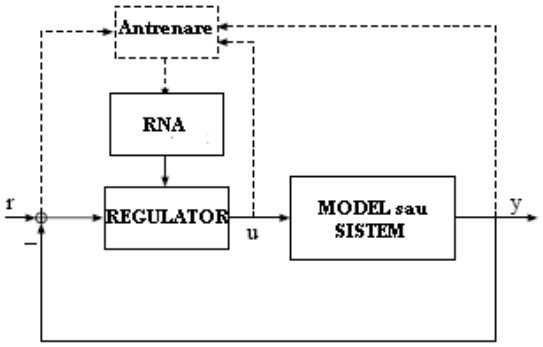


Figura 1.10 Regulator în buclă închisă cu autoacordare cu RNA

Antrenarea trebuie condusă astfel încât să acopere tot spaţiul modelului de proces. În astfel de sisteme regulatoarele PID sunt utilizate pe scară largă.

1.4.3. Control neuronal Direct

Modelarea regulatorului

Dintre cele patru modele de control neuronal direct, cel mai simplu din punct de vedere al proiectării regulatorului este utilizarea unei reţele neuronale pentru modelarea unui regulator existent, Figura 1.11.

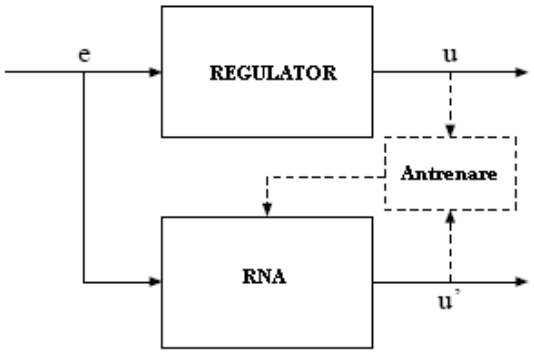


Figura 1.11 Modelare regulator

Setul de antrenare pentru regulatorul neuronal ales este format din perechi de intrare ieşire. Intrarea reprezintă intrarea regulatorului iar ieşirea reprezintă valorile impuse pentru intrările corespunzătoare sistemului.

În mod similar, această structură de control neuronal se poate formula astfel (Ec. 13)

$$NN : \min_{w} \{F\{u_c - u_n(w,...)\}\} \qquad Ec.\ 13$$

unde u_c *este ieşirea regulatorului (curent) existent C**. De obicei regulatorul curent existent C* poate fi şi un operator uman, sau se poate obţine din:

$$NN : \min_{C} \{F\{y^* - y_{p/m}(u,...)\}\}; \quad u = C(....) \qquad Ec.\ 14$$

Ca şi modelul procesului, regulatorul este un sistem dinamic şi deseori conţine elemente integratoare şi/sau elemente de diferenţiere.

Dacă pentru modelarea regulatorului utilizat se foloseşte o reţea feed-forward, la intrarea reţelei atât pentru modelare cât şi în timpul exploatării trebuiesc specificate şi intrări dinamice (modul de variaţie a semnalelor de intrare).

O altă aproximare este similară cu dezvoltarea unui model de proces tip ARX (auto regresiv cu intrări exogene) cu excepţia că ieşirea respectiv intrarea sistemului sunt schimbate cu eroarea feed-back şi ieşirea regulatorului.

1.4.3.1. Neuro-control fără modelul sistemului

În anumite procese in absenţa unui regulator, cercetători s-au inspirat din modul cum un operator uman învaţă să controleze un proces, fără să aibă cunoştinţe amănunţite despre dinamica procesului.

Astfel s-au încercat proiectarea unor regulatoare implementate cu reţele neuronale care prin adaptare şi învăţare pot rezolva probleme dificile de reglare şi control în absenţa unui model al procesului.

În general, acest tip de *control neuronal fără model* se poate exprima astfel:

$$NN : \min_{w} \{F\{y^* - y_p(u,...)\}\}; \quad u = N(w,...) \qquad Ec.\ 15$$

- unde y_p este ieşirea sistemului.

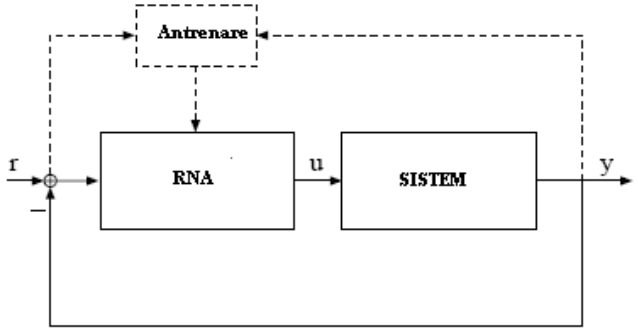


Figura 1.12 Neuro-control fără modelul sistemului

Ideea de mai sus se poate descrie ca o metodă de control adaptivă directă când nici modelul nu se cunoaşte dinainte şi nici în timpul proiectării sistemului de control.

De multe ori la astfel de modelări de sisteme de control foarte des se utilizează metoda de învăţare prin reinformare şi din această cauză metoda se numeşte la fel.

1.4.3.2. Control neuronal bazat pe model

Metoda se bazează pe antrenarea reţelei neuronale pe baza utilizări unui model al sistemului în cauză. Astfel putem evita defectarea sistemului în timpul proiectării, dar metoda este utilizabilă numai în cazul în care putem avea un model foarte exact al procesului de condus.

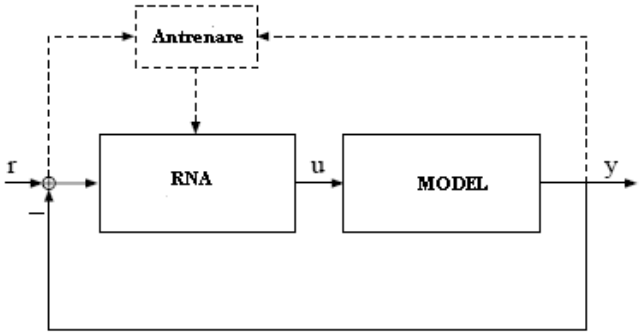


Figura 1.13 Control neuronal bazat pe model

Regula de control în general se poate formula astfel:

$$NN : \min_{w} \{F\{y^* - y_m(u,...)\}\}; \quad u = N(w,...) \qquad \textit{Ec. 16}$$

Dacă nu există model pentru proces, atunci putem utiliza un model al procesului anterior identificat cu o metodă corespunzătoare.

După obţinerea modelului pentru sistem, modelul poate fi utilizat în modelarea structurii de control După o antrenare şi testare riguroasă în mod de simulare a reţelei neuronale, controlerul neuronal poate fi instalat în sistemul de control real.

În realitate, această structură de regulator neuronal bazat pe modelul sistemului nu numai că s-a demonstrat în multe studii dar de asemenea utilizarea lui a condus la utilizări eficiente. Aceste abordări pot fi folosite atât în control on-line cât şi în control off-line.

1.4.3.3. Regulator neuronal robust pe bază de model

Structurile de control cu reţele neuronale discutate în paragrafele anterioare au câteva deficienţe:

- reţeaua neuronală trebuie antrenată pentru fiecare aplicaţie nouă
- Re-antrenarea reţelei neuronale este necesară şi la mici modificări ale criteriilor de reglare sau dacă regulatorul se aplică la un alt proces similar cu procesul anterior reglat.

Pentru a preveni aceste deficienţe în mod natural se introduce conceptul de metodă robustă de proiectare a regulatoarelor neuronale.

În această nouă viziune fiecare proces şi modul de reglare a lui nu este considerat ca un proces nominal ci un proces făcând parte dintr-o familie de procese. În majoritatea cazurilor aceste familii sunt

specificate pe baza unor modele de zgomot şi de modele de variabilitate a parametrilor proceselor din familie.

Controlul neuronal robust pe bază de model se poate formula astfel:

$$NN : \min_{w} \left\{ F\left\{ y* - y_{m_i}(u,...) \right\} \right\}; \quad u = N(w,...) \qquad \textit{Ec. 17}$$

unde m_i reprezintă membrul i din familia de model $\boldsymbol{M_i}$

În caz ideal robusteţea se referă atât la familia de modele cât şi la o anumită variabilitate în modele reale. Stabilitatea sistemelor robuste se referă la stabilitatea tuturor sistemelor care fac parte din aceeaşi familie.

Totodată trebuie să precizăm că metodele robuste de reglare adeseori nu ajung la performanţa regulatoarelor prezentate anterior.

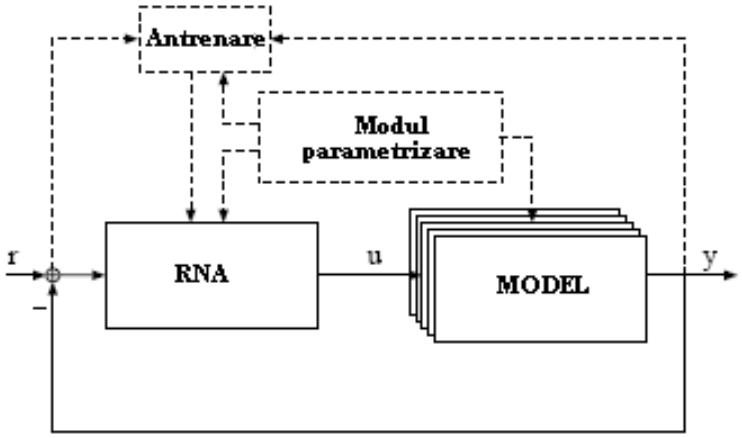


Figura 1.14 Regulator neuronal robust pe bază de model

1.5. Organizarea tezei

În acest subcapitol se trece în revistă câte o introducere succintă asupra conţinutului celor şase capitole, prezentând astfel structura acestei teze de doctorat.

Capitolul 1 constituie o descriere a fundamentelor biologice ale inteligenţei artificiale şi a realizării reţelelor neuronale artificiale bazate pe modele neuromorfe. De asemenea se prezintă stadiul actual al cercetărilor în domeniul aplicării reţelelor neuronale în sisteme de control.

În Capitolul 2 este expusă teoria modelării şi simulării reţelelor neuronale neuromorfe. În cadrul acestei sinteze sunt detaliate metodele de învăţare specifice acestui tip de reţea neuronală şi se introduce şi un model propriu, adaptat pentru implementare în hardware digital. În partea a doua a capitolului se găseşte o prezentare a strategiilor de simulare a reţelelor neuronale pulsative precum şi a celor mai importante sisteme de simulare realizate de cercetătorii domeniului şi prezente în literatura de specialitate. În finalul capitolului se prezintă rezultatele proprii în simularea acestor sisteme neuronale.

Capitolul 3 cuprinde modalităţile de realizare a reţelelor neuronale artificiale pe sisteme digitale reconfigurabile FPGA şi sunt prezentate tehnicile de implementare a logicii de reconfigurare compilată în timp şi reconfigurare în timpul utilizării. În acest capitolului se discută maparea diferitelor algoritmi de reţele neuronale pe sisteme FPGA şi sunt prezentate mai multe sisteme de reţele neuronale hardware la baza cărora stau sisteme FPGA.

Capitolul 4 prezintă motivaţii pro şi contra în alegerea suportului electronic pentru implementarea total paralelă a reţelelor neuronale artificiale. Sunt descrise o serie de astfel de implementări hardware, citând lucrările ştiinţifice care stau la baza acestora. Partea de contribuţii personale ale acestui capitol descrie aplicaţiile realizate în domeniul implementării totale paralele a reţelelor neuronale neuromorfe, detaliind funcţionarea subansamblelor acestora, dar evidenţiind şi rezultatele experimentale obţinute în comparaţie cu realizări software.

În Capitolul 5 sunt prezentate realizări proprii de reţele neuronale pulsative implementate în hardware cu ajutorul incorporării unor procesoare de tip soft-core în circuitele FPGA utilizate. Sunt prezentate patru astfel de aplicaţii, câte una aplicativă şi una de test (benchmark) utilizând procesoare încorporate pe 8 respectiv pe 32 de biţi. Printre acestea se numără o aplicaţie de detectare automată a componentelor de frecvenţă a unui semnal analogic, un sistem de recunoaştere de caractere, respectiv testele benchmark de clasificare a seturilor de date Fischer IRIS şi a bazei de date Wisconsin Breast Cancer.

Capitolul 6 conţine concluziile generale şi lista contribuţiilor originale realizate de autor în cadrul acestei lucrări.

2. MODELAREA ŞI SIMULAREA REŢELELOR NEURONALE NEUROMORFE

2.1. Alegerea modelului optim pentru neuronii pulsativi corticali

2.1.1. Introducere

În ultimii câţiva ani, în comunitatea care se ocupă de reţelele neuronale artificiale, s-a observat o trecere accentuată spre reţele neuronale pulsatorii. Motivate de descoperiri biologice, mai multe studii consideră reţelele neuronale legate de pulsuri cu temporizarea pulsaţiilor ca o componentă esenţială a studierii şi înţelegerii procesării informaţiei din creier.

Reţelele neuronale pulsative dispun de mai multe proprietăţi interesante şi cu potenţial bun pentru a fi utilizate la aplicaţii inginereşti.

Comportament temporal intrinsec: În RNP timpul este utilizat pentru a codifica informaţia, iar calculele sunt bazate de asemenea pe temporizări. Acest fapt face ca astfel de reţele neuronale să fie în mod intrinsec capabile de a fluxuri de valori de intrare variabile în timp. În reţelele neuronale clasice, acest lucru este realizat prin introducerea de întârzieri sau prin adăugarea conexiunilor recurente. În RNP este posibil să se efectueze procesarea temporală chiar şi cu arhitecturi de tip feed-forward.

Comunicaţie utilizând impulsuri: În RNA neuronii comunică prin valori continue, reale. Din punct de vedere al implementărilor, acest lucru poate deveni o problemă importantă, deoarece aceste valori reale trebuie multiplicate cu valori de ponderare, transmise unor alte elemente aflate la distanţe arbitrare (în termeni de implementare hardware) şi necesită o capacitate semnificativă de memorie. Pe de altă parte, neuronii din RNP comunică prin impulsuri, adică prin evenimente „*rare*" reprezentabile pe un singur bit. Astfel, lăţimea de bandă necesară pentru comunicaţia dintre neuroni este drastic redusă.

Capacitate de calcul mai ridicată decât în cazul RNA: S-a demonstrat în (Maas & Natschläger, 1997)că neuronii bazaţi pe modele cu impulsuri sunt mai eficienţi din punct de vedere al puterii de calcul decât neuronii bazaţi pe modele analogice. O RNP perturbată de zgomot poate aproxima orice RNA cu o precizie arbitrară cu un timp de execuţie influenţat numai de numărul de straturi din RNA şi cu un număr de neuroni ce este numai dublul numărul neuronilor din RNA. Pe de altă parte, este necesară o reţea neuronală analogică de dimensiuni mult mai mari (mii de neuroni) pentru a simula unele funcţii ce pot fi implementate utilizând un singur neuron pulsativ. Această asimetrie duce la concluzia, că RNP sunt strict mai eficiente ca putere de calcul decât reţelele neuronale analogice, bazate pe modele clasice.

Inspirat de biologie, neuromorfic: Reţelele neuronale pulsative sunt mai plauzibile din punct de vedere biologic decât alte reţele neuronale. Utilizarea sistemelor de calcul cu arhitectură inspirată de mecanisme biologice este însă un concept foarte dezbătut. Este adevărat, că pentru multe domenii inginereşti a propune ca unic scop al unei lucrări plauzibilitatea biologică nu este o abordare validă. Un exemplu frecvent este cel al unei aeronave Boeing 747. Păsările naturii sunt foarte diferite ca design, dar construind numai o pasăre mare nu obţinem un Boeing 747. Modul de operare al unei astfel de maşini este drastic diferit de cea a unei păsări, deci este nevoie de idei de construcţie complet noi. În cazul sistemelor inteligente însă, o mulţime de tehnici de calcul, teorii matematice şi statistice, respectiv 50 de ani de cercetări în aceste domenii nu au reuşit să se apropie de inteligenţa ce o putem percepe în interacţiunea dintre fiinţele umane. Despre nici un singur sistem artificial curent nu se poate spune că este inteligent, comparat cu un sistem biologic. Mai mult decât atât, nici măcar noţiunea de sistem inteligent nu este clar definită. Turing a spus, că un sistem este inteligent dacă nu-l putem distinge de o fiinţă umană. De aceea, putem afirma cu încredere, că abordarea neuromorfică în domeniul inteligenţei artificiale este cu siguranţă întemeiată. Acest lucru nu însemnă, că fiecare detaliu complicat trebuie înţeles şi simulat, dar cel puţin arhitecturile de procesare importante pot

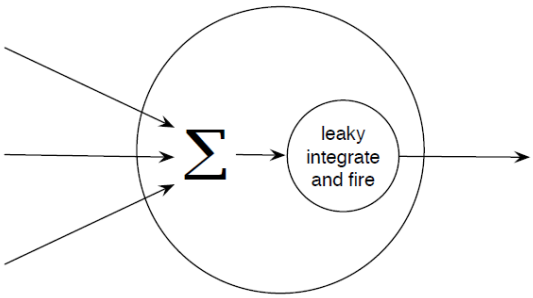


Figura 2.1 Structura de bază a unui neuron pulsativ

fi surse de inspiraţie folositoare. Studierea arhitecturii simplificate ce se află la baza modului de operare al creierului natural poate ajuta de asemenea, la înţelegerea funcţionării acestuia. Aceste concluzii nu ar putea fi extrapolate pentru sisteme nervoase biologice dacă am utiliza modele ce sunt foarte departe de funcţionarea propriu-zisă a creierului, cum ar fi de exemplu maşinile Turing.

De curând, au apărut câteva aplicaţii importante ce utilizează RNP, cum ar fi recunoaşterea de feţe prezentată în (Delorme & Thorpe, 2001) care prezintă o viteză de execuţie mare, propagând un singur val de impulsuri printr-o RNP sau o aplicaţie de procesare de imagini ca o metodă de citire de pe buze în (Booij, 2004), precum şi un sistem detector de epilepsie bazat pe semnale EEG în (Ghosh-Dastidar & Adeli, 2007) sau sistemul iterativ de găsire a rădăcinilor cu o RNP prezentat în (Iannella & Kindermann, 2005).

2.1.2. Dispute curente în domeniul RNP

Reţelele neuronale pulsative sunt modele neuronale noi utilizabile ca instrumente inginereşti foarte promiţătoare şi atrag o atenţie tot mai mare în literatura de specialitate. Totuşi, mai există multe teme deschise care necesită a fi adresate pentru ca aceste RNP să ajungă să fie instrumente bine dezvoltate de la stadiul de studiu neuro-biologic în care sunt de fapt la momentul de faţă.

Primul obstacol de trecut în utilizarea acestor reţele neuronale speciale este faptul că fluxul de informaţii spre şi dinspre acestea se realizează prin impulsuri. Întrucât cele mai multe cantităţi din lumea reală sunt de natură analogică şi variază continuu, reprezentarea acestora cu impulsuri fără amplitudine nu este un lucru simplu. Studiul neuronilor biologici a rezultat în apariţia mai multor ipoteze pentru codarea valorilor reale în trenuri de impulsuri. Acestea pot apărea ca o codare a ratei impulsurilor, care ia în considerare numai valoarea medie a timpului de apariţie a unui impuls sau ca codarea exactă a momentelor temporale de apariţie a acestor impulsuri. Coduri intermediare care utilizează filtre temporale sunt de asemenea utilizate. Totuşi, nu este încă clar care sunt proprietăţile acestor scheme de codare şi dacă este posibilă o codare fără pierdere de informaţie cu aceste metode. Toate aceste întrebări sunt cruciale şi trebuie clarificate înainte de a începe implementarea unor aplicaţii inginereşti cu neuroni pulsativi.

Când vorbim despre RNP există de fapt o supraabundenţă de diferite modele, de la cele simple de tipul integrează şi activează prin neuroni cu diferite modele sinaptice până la modele comportamentale complete descrise prin ecuaţii diferenţiale. Există câteva studii teoretice (Maass, 1996) care arată că unele dintre aceste modele sunt mai eficiente decât celelalte, dar aceste studii se limitează doar la un set redus de modele şi scheme de codare. O explicaţie generală a capacităţii de calcul a diferitelor modele neuronale pulsative şi a schemelor de codare conexe reprezintă şi astăzi o temă deschisă cercetărilor actuale.

Pe de altă parte, schema de codare utilizată pentru a introduce datele într-o RNP şi pentru a evalua ieşirile acesteia nu este întotdeauna aceeaşi cu cea utilizată pentru comunicarea dintre neuronii reţelei. Există chiar şi posibilitatea de a utiliza concomitent mai multe scheme de codare pentru comunicaţiile interne. Acest lucru face ca analiza funcţionării interne a acestor reţele să fie dificilă, precum şi traducerea acesteia în termeni uşor de asimilat de către om, cum este cazul modelelor neuronale, unde există legături puternice cu sisteme bazate pe reguli fuzzy (Mantas, Puche, & Mantas, 2006).

Antrenarea acestor reţele este o altă temă în mod notoriu recunoscută a fi dificilă, deoarece literatura de specialitate ne arată doar o singură metodă supervizată, limitată şi aceasta la o singură schemă de codare, cu frecvente probleme de convergenţă. Deoarece sistemele naturale nu cunosc antrenarea supervizată, majoritatea metodelor derivate din studiile neuro-informatice sunt de natură nesupervizată. Cea mai importantă este cea numită Spike Time Dependent Plasticity (STDP) sau plasticitate dependentă de timpii de activare. În cazul acestei metode, eficacitatea unei conexiuni dintre doi neuroni este mărită, dacă activarea acestuia este corelată cu activarea predecesorului său altfel eficacitatea scade. Această învăţare de tip Hebb a fost observată în multe zone ale creierului biologic. Un alt exemplu pentru astfel de metode de antrenare este plasticitatea homeostatică sau intrinsecă (Intrinsec Plasticity - IP). Această metodă nu este văzută ca o regulă de antrenare, ea vizând de fapt menţinerea la o valoare constantă a ratei medii de activare a unui neuron, funcţionând deci ca şi un mecanism de optimizare energetică. S-a arătat, însă recent (Triesch, 2007), că această metodă simplă poate fi utilizată pentru antrenare şi are un impact drastic asupra proprietăţilor de învăţare a metodei STDP.

2.1.3. Caracteristici neuro-computaţionale

În orice studiu legat de dinamica unei reţele neuronale pulsative, sunt două probleme specifice: 1. ce model descrie dinamica pulsaţiei fiecărui neuron şi 2. cum sunt conectaţi neuronii. O alegere greşită a modelului sau o conexiune greşită, poate duce la rezultate care nu au nimic în comun cu procesarea ce are loc în creier.

În continuare se elaborează prima problemă, comparând neuronii pulsativi. Se vor prezenta diferite modele de neuroni pulsativi precum şi o clasificare după următoarele proprietăţi: 1. numărul de caracteristici neuro-computaţionale pe care le pot reproduce, şi 2. eficienţa lor de implementabilitate, de ex. numărul de operaţii în virgulă flotantă (adunare, înmulţire, etc.) necesare pentru simularea modelului pe o durată de 1ms. Rezultatele comparaţiei sunt rezumate în Figura 2.2.

În Figura 2.2 se prezintă 20 dintre cele mai proeminente caracteristici biologice ale neuronilor pulsativi (Izhikevich, 2004). Ţelul urmărit este ilustrarea bogăţiei şi complexităţii comportamentului pulsativ al neuronilor individuali ca răspuns la impulsuri simple de curent continuu. Ce se întâmplă dacă doar zeci (dar dacă milioane) de astfel de neuroni sunt cuplaţi - depăşeşte tot ceea ce noi putem înţelege. Utilizând câteva dintre modelele amintite în continuare, nu este dificil să simulăm mii de neuroni corticali în timp real cu o rezoluţie de 1ms.

❖ **Pulsaţie tonică (Tonic Spiking)**

Majoritatea neuronilor pot fi excitaţi, ceea ce înseamnă, că sunt pasivi (statici), dar pot să emită impulsuri când sunt stimulaţi. Pentru a testa această caracteristică, cercetătorii de neurofiziologie injectează pulsuri de curent continuu printr-un electrod legat la un neuron şi înregistrează potenţialul membranei. Curentul la intrare şi răspunsul neuronal sunt înregistraţi unul sub celălalt Figura 2.2(A).

Cât timp impulsul de intrare este activ, neuronul continuă să emită un tren de impulsuri. Acest comportament, numit pulsaţie tonică, poate fi observat la trei tipuri de neuroni corticali: neuronii excitatori cu pulsaţie regulată (RS), cu pulsaţii cu prag scăzut sau cu valoare de prag scăzută (LTS), şi neuronii inhibitorii (Gerstner & Kistler, 2002) (Gibson, Belerlein, & Connors, 1999), cu pulsaţie rapidă (FS). O continuă emisie a acestor neuroni indică faptul că sunt supuşi unor impulsuri de intrare persistente.

❖ **Pulsaţie fazică (Phasic Spiking)**

Neuronul poate să emită un singur impuls la pornirea impulsului de intrare, ca în Figura 2.2(B), şi rămâne pasiv după aceea. Acest tip de răspuns se numeşte pulsaţie fazică, şi poate fi utilizată pentru detectarea începutului stimulării.

❖ **Fluxuri de impulsuri tonice (Tonic Bursting)**

Anumiţi neuroni, ca şi neuronii de tip chattering din neocortexul pisicii, emit un rând de impulsuri când sunt stimulaţi, ca în Figura 2.2(C). Frecvenţa apariţiei acestor trenuri poate fi mare, de până la 50 Hz, şi se crede că astfel de neuroni contribuie la oscilaţiile de frecvenţă gamma a creierului.

❖ **Fluxuri de impulsuri fazice (Phasic Bursting)**

Asemănător cu pulsaţiile fazice, anumiţi neuroni emit impulsuri fazice, ca în Figura 2.2(D). Astfel de neuroni raportează începutul unei stimulări prin emiterea unui tren de impulsuri. Există trei ipoteze importante despre rolul unor trenuri de impulsuri al creierului care ar fi: **1.** trenurile de impulsuri sunt necesare pentru a învinge defecţiunea transmisiei şi reducerea zgomotului neuronal, **2.** trenurile de impulsuri pot transmite proeminenţele semnalului de intrare, deoarece efectul unui tren de impulsuri este mai mare decât al unui singur impuls asupra neuronului post-sinaptic şi **3.** fluxurile de impulsuri pot fi folosite ca un mod de comunicare selectivă între neuroni (Izhikevich, Desai, Walcott, & Hoppensteadt, 2003), unde frecvenţa dintre impulsuri să fie codificarea canalului de comunicare. Un model bun al reţelei neuronale corticale nu poate să neglijeze aceste „izbucniri" neuronale.

❖ **Model combinat (Trenuri de impulsuri urmate de pulsaţii - Bursting Then Spiking)**

Neuronii excitatori intrinseci emit trenuri de impulsuri în neocortexul mamar, pot emite o activitate de pulsaţie mixtă, descrisă în Figura 2.2(E). Ei emit trenuri de impulsuri fazice în momentul apariţiei stimulului şi trec la modul de pulsaţie tonică. Nu este clar ce fel de mod de calcul un astfel de neuron poate crea în plus faţă de detectarea momentului de început al stimulării şi raportând lungimea perioadei stimulării.

❖ **Adaptarea frecvenţei pulsaţiei**

Cel mai răspândit tip de neuron excitator în neocortexul mamiferelor, numit RS - celulă cu pulsaţie regulată -, emite impulsuri tonice cu frecvenţă în descreştere, ca in Figura 2.2(F). Înseamnă că frecvenţa este relativ mare în momentul stimulării, şi ulterior se adaptează. Neuronii inhibitori cu pulsaţiile cu valoarea de prag scăzută (LTS), au de asemenea această caracteristică. Frecvenţa dintre impulsuri la aceste celule este o codificare pentru timpii de la apariţia impulsului de stimulare.

❖ **Excitabilitatea de clasa 1**

Frecvenţa pulsaţiilor tonice a neuronilor excitatori neo-corticali RS depinde de puterea impulsului de intrare, şi poate să varieze între 2 Hz şi 200 Hz, sau chiar o valoare mai mare. Abilitatea de a emite impulsuri de frecvenţă mică, când impulsul de intrare este slab (dar deasupra valorii de prag) se numeşte excitabilitate de clasa 1. Neuronii de acest tip pot să codifice puterea impulsului de intrare în rata lor de emisie, Figura 2.2(G).

❖ **Excitabilitate de clasa 2**

Anumiţi neuroni nu pot să emită trenuri de impulsuri de frecvenţă mică. Ei sunt ori pasivi sau emit un tren de impulsuri la o valoare destul de mare de frecvenţă, de ex de 40 Hz, ca în Figura 2.2(H). Aceşti neuroni se numesc de excitabilitate clasa 2, rata lor de emisie este un aproximator slab al puterii stimulării.

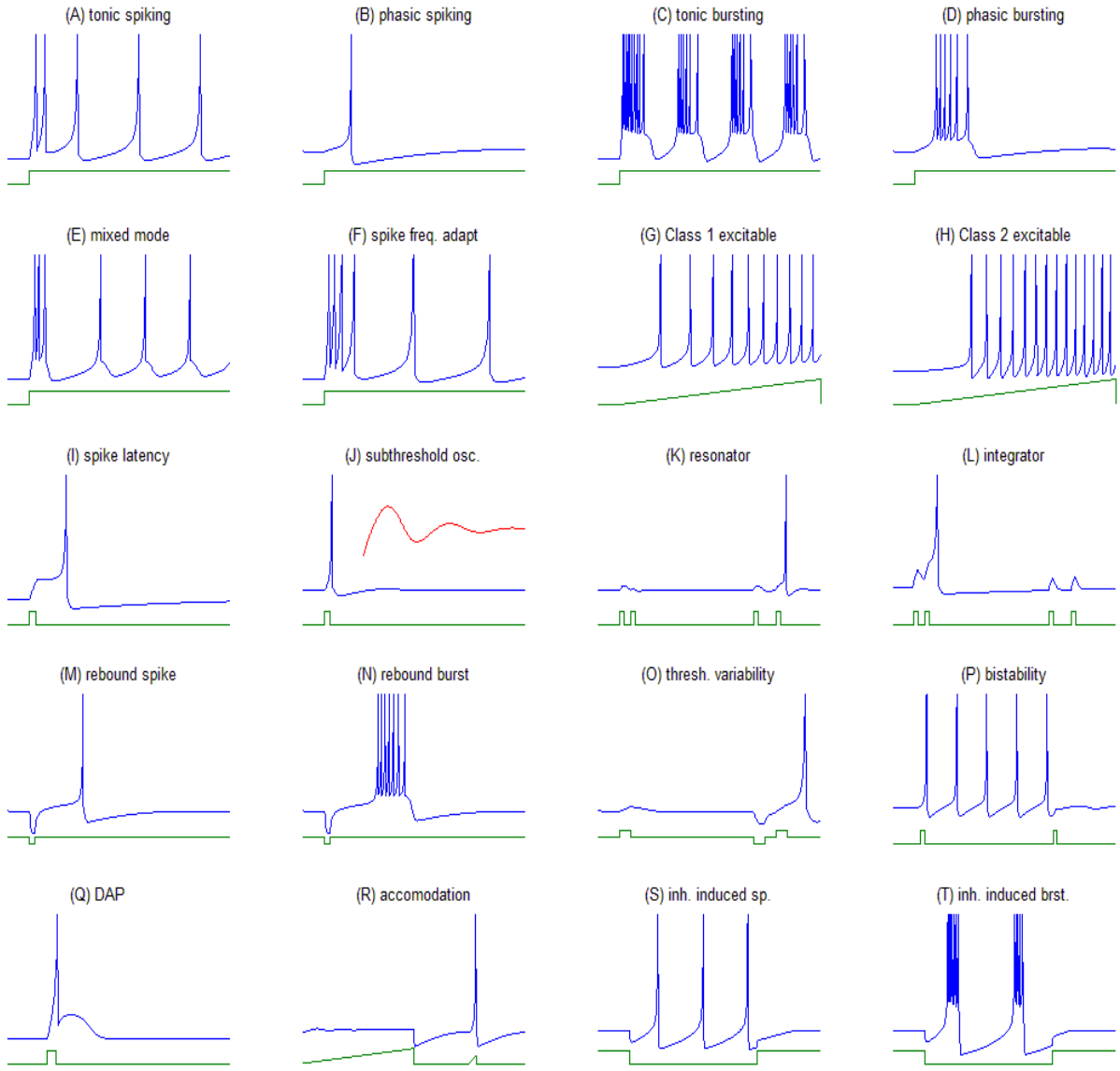


Figura 2.2 Tipuri de neuroni pulsativi

❖ **Latenţa impulsului (Spike Latency)**

Majoritatea neuronilor corticali emit impulsuri cu o întârziere care depinde de puterea semnalului de intrare. Pentru un impuls relativ slab, dar deasupra valorii de prag, întârzierea, numită şi latenţa impulsului, poate fi destul de mare, după cum se vede în Figura 2.2(I). Celulele RS din neocortexul mamiferelor pot avea o latenţă de zeci de milisecunde. Astfel de latenţe (întârzieri) pot asigura un mecanism de temporizare pentru a coda puterea impulsului de intrare.

❖ **Oscilaţii sub valoarea de prag**

Practic orice structură a creierului are neuroni capabili să arate comportament oscilatoriu, Figura 2.2(J). Frecvenţa unor astfel de oscilaţii joacă un rol important şi neuronii de acest tip acţionează ca nişte filtre trece bandă, după cum va fi discutat mai târziu.

❖ **Preferinţă şi rezonanţă a frecvenţei**

Datorită fenomenului de rezonanţă, neuronii având oscilatori, pot răspunde selectiv la valorile de intrare, având conţinut de frecvenţă similară (asemănătoare) cu frecvenţa oscilaţiilor sub valoarea de prag. Astfel de neuroni pot implementa interacţiuni cu frecvenţe modulate (FM) şi multiplica semnalele. În Figura 2.2(K), se prezintă un astfel de neuron stimulat cu o pereche de impulsuri, având frecvenţe diferite ale impulsurilor. Neuronul răspunde numai la una dintre perechi, la impulsul a cărui frecvenţă rezonează cu frecvenţa oscilaţiilor sub nivelul de prag. Aceşti neuroni sunt numiţi rezonatori.

❖ **Integrare şi detectarea coincidenţei**

Neuronii fără potenţial oscilator acţionează ca integratori. Ei preferă impulsuri de intrare de frecvenţă înaltă; cu cât este mai mare frecvenţa cu atât este mai posibilă activarea lor, vezi Figura 2.2(L). Aceaştia pot fi folositori pentru detectarea impulsurilor coincidente sau aproape coincidente.

❖ **Impulsuri de revenire (Rebound Spike)**

Când un neuron primeşte şi după aceea este eliberat de impulsul inhibitor, poate să emită un impuls post inhibitor sau de ricoşare, ca în Figura 2.2(M). Acest fenomen este în legătură cu excitaţia anodică de rupere în membrana excitabilă. Mulţi neuroni pulsativi pot emite ca răspuns la intrări scurte, inhibitorii, astfel estompând diferenţa dintre excitaţie şi inhibiţie.

❖ **Trenuri de impulsuri de revenire**

Anumiţi neuroni, inclusiv cei din celulele thalamo-corticale, pot emite trenuri de impulsuri post – inhibitorii, ca în Figura 2.2(N). Se crede că astfel de trenuri de impulsuri contribuie la oscilaţiile somnului în sistemul thalamo-cortical.

❖ **Variabilitatea valorii de prag**

O concepţie greşită în comunitatea cercetărilor reţelelor artificiale neuronale este că neuronii pulsatorii au un voltaj de valoare fixă a valorii de prag. Este binecunoscut, că neuronii biologici au o valoare de prag care depinde de activitatea apriorică a neuronului. În Figura 2.2(O), prima dată stimulăm neuronul cu un puls excitator mic de curent care produce 10 mV de-polarizaţie. Neuronul nu emite, prin urmare valoarea de intrare este mai mică decât valoarea de prag. După aceea, aplicăm un puls inhibitor scurt, şi după aceea acelaşi puls de curent sub valoarea de prag. Neuronul emite de această dată, deoarece valoarea lui de prag a devenit mai mică din cauza impulsul inhibitor anterior. Prin urmare, aceeaşi depolarizare de 10mV poate fi sub valoarea de prag sau deasupra, depinzând de activitatea anterioară. În mod interesant, un următor puls excitator poate să ridice valoarea de prag şi să facă neuronul mai puţin excitabil.

❖ **Bi-stabilitatea stării de pulsaţie şi pauză**

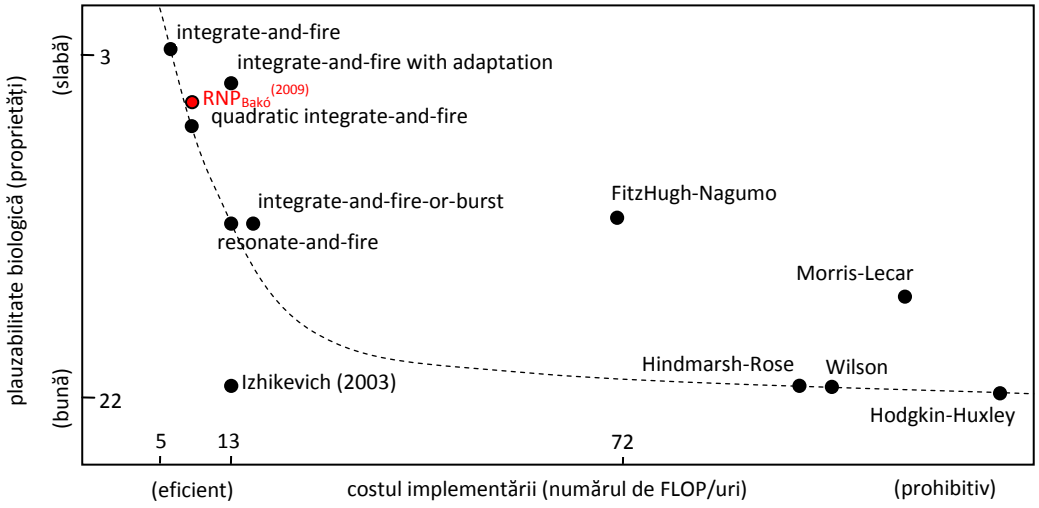
Anumiţi neuroni pot să arate două modele stabile de operare: pauză şi pulsaţie tonică (sau chiar trenuri de impulsuri). Un puls excitator sau inhibitor, poate să schimbe de la un mod la altul, Figura 2.2(P), astfel creând o posibilitate interesantă pentru bi-stabilitate şi memorie de scurtă durată. Trebuie să notăm că, pentru ca schimbarea de la pulsaţie tonică la modul de pauză să aibă loc, impulsurile de intrare trebuie să sosească într-o fază potrivită a oscilaţiei, astfel punând accent pe importanţa temporizării impulsurilor în astfel de procesare a informaţiei.

❖ **Potenţiale ulterioare depolarizante**

După emiterea unui impuls, potenţialul membranei neuronului poate să arate o prelungită hiperpolarizare ulterioară ca în exemplul din Figura 2.2(B), I sau M, sau un prelungit potenţial ulterior (DAP), ca în Figura 2.2(Q). Astfel de DAP-uri pot apărea din cauza influenţei dendritice, din cauza unor curenţi de prag mari, interiori, activaţi în timpul impulsului, sau din cauza unei interacţiuni dintre curenţii sub valoarea de prag. În orice caz, un astfel de neuron are o perioadă refractară mai scurtă şi devine super-excitabil.

❖ **Obişnuinţa (acomodarea)**

Neuronii sunt foarte sensibili la valorile de intrare scurte şi coincidente, dar s-ar putea să nu emită semnal ca răspuns la o valoare de intrare puternică şi încet crescătoare, după cum se vede în Figura 2.2(R). Curentul în creştere uşoară nu provoacă un impuls, pe când un curent mai mic dar cu creştere rapidă provoacă un impuls. În timpul unei rampe de creştere mică, curenţii interni au timp suficient să se inactiveze iar curenţii externi au timp suficient să se activeze. Deci neuronul se obişnuieşte, devine mai puţin excitabil şi nu poate genera un impuls.



Models	biophysically meaningful	tonic spiking	phasic spiking	tonic bursting	phasic bursting	mixed mode	spike frequency adaptation	class 1 excitable	class 2 excitable	spike latency	subthreshold oscillations	resonator	integrator	rebound spike	rebound burst	threshold variability	bistability	DAP	accommodation	inhibition-induced spiking	inhibition-induced bursting	chaos	# of FLOPS
integrate-and-fire	–	+	–	–	–	–	–	+	–	–	–	–	+	–	–	–	–	–	–	–	–	–	5
RNP Bakó (2009)	–	+	–	–	–	–	+	+	–	+	–	–	+	–	–	–	–	–	–	–	–	–	**6**
integrate-and-fire with adapt.	–	+	–	–	–	–	+	+	–	–	–	–	+	–	–	–	–	+	–	–	–	–	10
integrate-and-fire-or-burst	–	+	+		+	–	+	+	–	–	–	–	+	+	+	–	+	+	–	–	–		13
resonate-and-fire	–	+	+	–	–	–	–	+	+	–	+	+	+	+	–	–	+	+	+	–	–	+	10
quadratic integrate-and-fire	–	+	–	–	–	–	–	+	–	+	–	–	+	–	–	+	+	–	–	–	–	–	7
Izhikevich (2003)	–	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	13
FitzHugh-Nagumo	–	+	+	–		–	–	+	–	+	+	+	–	+	–	+	+	–	+	+	–	–	72
Hindmarsh-Rose	–	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	120
Morris-Lecar	+	+	+	–		–	–	+	+	+	+	+	+	+		+	+	–	+	+	–	–	600
Wilson	–	+	+	+			+	+	+	+	+	+	+	+	+	+		+	+				180
Hodgkin-Huxley	+	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	1200

Figura 2.3 Comparaţie a diferitelor modele de neuroni pulsativi

❖ **Impuls indus de inhibiţie**

O caracteristică interesantă a neuronilor thalamo-corticali este că ei sunt pasivi, latenţi, când nu primesc impulsuri, dar emit dacă sunt hiper-polarizaţi de către un impuls inhibitor sau din cauza unui curent injectat, după cum se vede în Figura 2.2(S). Aceasta se întâmplă deoarece curentul injectat activează curentul h şi dezactivează curentul de calciu T, ducând la pulsaţie tonică.

❖ **Trenuri de impulsuri induse prin inhibiţie**

În loc de pulsaţie, un neuron thalamo-cortical poate să emită un tren de impulsuri tonice ca răspuns la o hiperpolarizare prelungită, ca în Figura 2.2(T). Se crede că astfel de trenuri de impulsuri au loc în timpul unor oscilaţii de undă în sistemul thalamo-cortical şi joacă un rol important în ritmurile somnului.

Nici un model nu poate manifesta toate aceste 20 de caracteristici neuro-computaţionale simultan, deoarece câteva sunt caracteristici mutual exclusive. De exemplu, un neuron nu poate fi un integrator şi un rezonator în acelaşi timp. Totuşi, există modele care cu uşurinţă pot fi reglate, să arate fiecare dintre aceste caracteristici. De exemplu, toate răspunsurile neurale din Figura 2.2 au fost obţinute printr-un simplu model pulsativ având patru parametrii uşor de reglat (Izhikevich E. , 2003).

2.2. Realizarea învăţării Hebbiene competitive

Învăţarea Hebbiană, dezvoltarea circuitelor neurale bazate pe activitatea corelată, are două mecanisme de bază. Cea mai bine cunoscută este modificarea sinaptică, bazată pe activitatea pe lângă ideile propuse de Hebb. La fel de important este mecanismul care forţează diferite sinapse să intre în competiţie una cu cealaltă în aşa fel, încât dacă anumite sinapse ale unui neuron post-sinaptic devin mai puternice, altele devin mai slabe.

De exemplu, regulile bazate pe corelaţia modificărilor sinaptice pot asigura un număr mare al aspectelor de dezvoltare în cortexul vizual, dar doar atunci când sunt combinate cu introducerea constrângerilor, ca să se asigure competitivitatea. Cu toate că modificările sinaptice Hebbiene au fost susţinute de experimente cu algoritmii de excitare şi inhibare pe termen lung, se cunoaşte foarte puţin despre mecanismele care generează competiţia dintre sinapse. Evidenţele experimentale de la diferitele preparate, sugerează că atât semnalul cât şi gradul de modificare sinaptică, care apare din repetatele împerecheri ale potenţialelor acţiunilor pre- şi post-sinaptice, depind de temporizarea lor relativă. Acţiunile potenţiale pre-sinaptice care urmează impulsuri post-sinaptice, produc o slăbire îndelungată a sinapselor. Cele mai mari modificări în eficacitatea sinaptică apar, când diferenţa de timp dintre acţiunile potenţiale pre- şi post-sinaptice este foarte mică, şi există o tranziţie bruscă din valori în creştere spre valori în scădere, când această valoare tinde spre 0. Această modificare sinaptică se numeşte plasticitate dependentă de temporizare a impulsului (STDP). Modificarea sinaptică prin reguli STDP a fost studiată prin modele de recunoaştere temporală a modelului (temporal pattern recognition), învăţare de secvenţă temporală de detectarea coincidenţei (Gerstner, Kempter, van Hemmen, & Wagner, 1996). Distribuţiile conductanţei sinaptice produsă prin STDP forţează neuronul post-sinaptic într-un regim de activare echilibrată, cu emisii neregulate de impulsuri, care este sensibil la temporizarea acţiunilor pre-sinaptice pe care le primeşte. O astfel de sensibilitate duce la competiţie între valorile de intrare pentru controlul temporizării impulsurilor post-sinaptice. Aceasta permite STDP să întărească selectiv sinapsele cu întârzieri relativ mai mici sau cu corelări mutuale mai puternice, în timp ce sinapsele rămase devin mai slabe.

2.2.1. Excitaţie echilibrată

Pentru ca un neuron să funcţioneze normal, trebuie să stabilească şi să menţină un nivel potrivit de excitaţie în aşa fel, încât să poată răspunde la impulsurile primite prin acţiuni potenţiale în proporţii rezonabile. Variabilitatea răspunsurilor asigură o constrângere pe impulsurile sinaptice de intrare asupra unui neuron. Răspunsurile primite de la un model care primeşte multe impulsuri independente pre-sinaptice pot fi mai puţin variabile decât răspunsurile observate în realitate (in-vivo). Corelările temporizării impulsurilor de intrare, ca de exemplu sincronizarea, pot contribui la creşterea variabilităţii. Totuşi, mulţi autori au observat că o variabilitate mare poate apărea şi dacă impulsurile excitatorii de intrare asupra unui neuron sunt relativ echilibrate faţă de curenţii sinaptici inhibitori şi ai membranei. Condiţia critică este că impulsul asupra neuronului ar trebui să fie suficient ca să ridice potenţialul

membranei la un punct puţin sub sau deasupra valorii de prag pentru generarea potenţialului de acţiune, astfel încât timpii impulsului să fie determinaţi prin fluctuaţie pozitivă la nivelul total de intrare. După cum se va arăta, STDP asigură un mecanism prin care acest echilibru poate fi stabilit şi întreţinut asupra unui şir de valori de intrare. Din aceasta rezultă o stare în care impulsurile pre-sinaptice pot controla temporizarea impulsurilor post-sinaptice şi o competiţie între sinapse poate să apară.

2.2.2. Discuţii

Cu toate că plasticitatea sinaptică Hebbiană este un concept puternic, ea suferă din cauza apariţiei mai multor probleme. În primul rând, sinapsele sunt modificate de câte ori activităţile pre- sau post-sinaptice sunt corelate. Astfel de activităţi corelate pot să apară doar prin întâmplare, reflectând ineficient o relaţie cazuală care poate fi studiată. Pentru corectarea acestei probleme, modelele reţelelor neurale adesea utilizează o covarianţă faţă de regulile de modificare sinaptice bazate pe corelare. Totuşi, o astfel de regulă nu poate, în general, obţine o competiţie între sinapse. Aceasta creează o nouă problemă a modificărilor pur Hebbiene: nu este competitivă, deci trebuie să se aplice constrângeri asupra lor, pentru a obţine rezultate interesante. STDP pare să rezolve ambele probleme menţionate anterior. Coincidenţele accidentale, non-cauzale slăbesc sinapsele, după cum am presupus, dacă integrala funcţiei modificării sinaptice este negativă. Competiţia apare într-un mod nou, nu din cauza unui echilibru în mod artificial impus al factorilor ne-specifici de creştere sau descreştere a eficienţei sinapselor, ci mai ales prin competiţia pentru controlul temporizării impulsurilor post-sinaptice. Impulsurile de intrare, care în mod consistent sunt cele mai previzibile pentru răspunsuri post-sinaptice, devin cele mai puternice impulsuri pentru neuron. Cauzalitatea este elementul cheie în STDP. Aşa cum a sugerat Hebb, sinapsele devin mai puternice doar dacă impulsurilor lor pre-sinaptice preced şi au contribuit la activarea neuronului post-sinaptic. STDP duce în mod automat la o stare de echilibru neregulat de activare, la care timpii impulsurilor pre- şi post-sinaptici sunt corelaţi.

Acest rezultat depinde de neliniaritatea procesului de generare a impulsului. Într-un model în care probabilitatea impulsurilor depinde de liniaritatea tensiunii de pe membrană, corelaţia dintre încărcarea pre- şi post-sinaptică nu-şi schimbă forma în eficacitatea sinaptică, aşa cum se întâmplă în Figura 2.4.

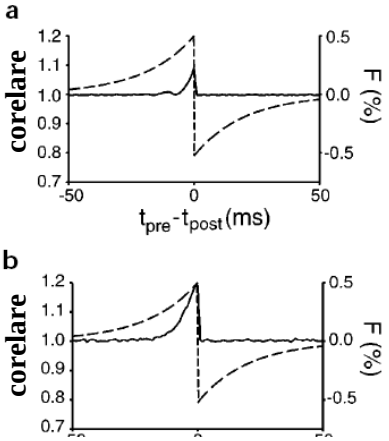


Figura 2.4 Corelaţia dintre încărcarea pre- şi post-sinaptică

Efectele neliniare, care fac corelaţiile cazuale a valorilor de intrare-ieşire să crească relativ faţă de o relaţie cazuală ca o descreştere a întregii eficacităţi sinaptice, sunt cruciale pentru producerea stabilităţii şi ale efectelor competitive al STDP. STDP reglează atât rata cât şi coeficientul de variaţie a încărcării post-sinaptice peste o gamă largă de valori de intrare. Aceasta reprezintă o funcţie regulatoare homeostatică a STDP, ceea ce în mod surprinzător, ca şi la regula lui Hebb, este destabilizatoare pentru sinapsele individuale. STDP poate în mod diferenţial să mărească valorile de intrare de scurtă întârziere date de stimuli. Există evidenţe experimentale, că reducerea întârzierii în răspunsul post-sinaptic apare şi în in-vivo. Un fenomen analog cu reducţia latenţei, discutată aici, prezice că, dacă un şobolan cobai se mişcă într-o anumită parte a spaţiului, acele celule care sunt active pentru spaţiu, s-ar putea să emită mai repede, după ce şobolanul a trecut de câteva ori prin acel spaţiu. Acest efect a fost observat experimental în (Blum & Abbott, 1996). O presupunere a modelului nostru este că slăbirea de către STDP este dominantă faţă de creşterea sinaptică. Aceasta este critică, pentru stabilitate. Dacă această presupunere nu este adevărată, rezultatele pe care le-am raportat, nu ar apărea niciodată, dintr-o combinaţie a STDP şi depresia homo-sinaptică, de lungă durată (slăbirea impulsurilor pre-sinaptice, care emit în lipsa unui impuls post-sinaptic). Cât timp STDP duce la creşterea impulsurilor de intrare efective în timp ce STDP şi/sau alte forme de plasticitate de lungă durată slăbeşte mai tare impulsurile ineficiente, rezultatele de bază întâlnite aici, ar trebui să fie aplicate.

Cea mai eficientă metodă de a mări puterea sinapsei este de a elibera emisia înainte de un impuls post-sinaptic şi după aceea să se oprească din emisie, în aşa fel, încât să nu piardă din putere prin emisii subsecvente care apar în urma activităţii post-sinaptice. O rată mare a depresiei post-sinaptice, care este

o caracteristică a sinapselor puternice din schema de re-distribuţie, asigură aceasta. STDP care acţionează pentru a modifica probabilitatea de emisie şi schimbă rata depresiei sinaptice este astfel un mecanism foarte eficient şi competitiv pentru a conduce sinapsele individuale spre limite maxime şi minime. STDP, cu toate că contribuie la competiţie, probabil nu este singura sursă a plasticităţii în situaţiile de analiză Hebbiene. Ca orice regulă de modificare Hebbiană, STDP nu poate mări eficienţa sinapsei fără o emisie post-sinaptică. Dacă sinapsa excitatoare din anumit motiv este prea slabă ca să emită, STDP nu poate să o salveze. Un mecanism non-hebbian, ca şi scalarea sinaptică 7-9, poate să înlocuiască această funcţie. În modelul STDP utilizăm două seturi de impulsuri care emit la un interval mai mare de timp de 100 ms, generează separat STDP şi totuşi nu sunt în competiţie. Experimentele spun că competiţia nu va avea loc în astfel de situaţii. Astfel de rezultate pot apărea dacă în fereastra temporară STDP, pentru o scădere a valorii sinaptice apare o urmă destul de lungă sau dacă STDP este suplimentat cu o depresie hetero-sinaptică suficient de puternică, sau este indusă competiţia printr-o scalare sinaptică 7-9. Dimensiunea ferestrei temporare prin care mărirea şi scăderea valorii sinaptice are loc, este critică în determinarea efectelor STDP. Ar fi în avantajul dimensiunii ferestrelor să fie diferite în anumite regiuni ale creierului, să se modifice în timpul dezvoltării, şi să fie dinamic ajustabil în timp scurt. Acesta ar permite STDP să rămână compatibilă cu valori de intrare relevante de corelare.

2.3. Metode de propagare înapoi a erorilor (back-propagation) în reţelele neuronale pulsative codificate temporal

Pentru o reţea de neuroni pulsanţi care codifică informaţia în cronometrarea timpilor de impulsuri individuale, se poate formula o regulă de antrenare supravegheată, asemănătoare cu metoda error back-propagation tradiţională. Cu acest algoritm, se poate demonstra cum pot executa reţelele de neuroni pulsanţi cu potenţiale de acţiune biologic rezonabile clasificări neliniare complexe în codificare temporală rapidă la fel de bine ca şi reţelele cu codificarea ratei impulsurilor. Există studii, care arată, aplicabilitatea pentru problema clasică XOR, ridicând problema în termenii unui cadru temporal, şi totodată pentru alte seturi de date reper (benchmark). Comparând numărul neuronilor pulsanţi necesari pentru codificarea problemei interpolate XOR, reţelele antrenate demonstrează că codificarea temporală este o codificare fiabilă pentru o procesare rapidă a informaţiei, şi astfel necesită mai puţini neuroni decât în cazul reţelelor cu codificarea ratei de impulsuri. Totodată, putem observa că un calcul temporal sigur se poate obţine doar prin folosirea funcţiilor-răspuns pulsative cu o constantă a timpului mai mare decât intervalul de codificare, cum se preconiza prin consideraţiile teoretice.

Datorită succesului său în reţelele neuronale artificiale clasice, neuronul cu funcţie de activare sigmoid este cunoscut ca fiind un model de succes a comportamentului neuronal biologic. Prin modelarea gradului la care un singur neuron biologic descarcă potenţialii de acţiune ca şi o funcţie monoton crescătoare a intrării, s-au construit mai multe aplicaţii utile ale reţelelor neuronale artificiale, şi s-au obţinut mai multe puncte de vedere teoretice cu privire la comportamentul structurilor de conectare. Totodată, natura pulsativă a neuronilor naturali a dus la cercetări asupra puterii computaţionale asociată cu codificarea temporală a informaţiei în impulsuri individuale. În (Maas, 1997) s-a dovedit faptul că reţelele de neuroni pulsanţi pot simula reţele neuronale sigmoid feed-forward întâmplător, şi pot aproxima astfel orice funcţie continuă, iar neuronii care propagă informaţia prin timpii impulsurilor individuale, sunt mai puternici din punct de vedere computaţional decât neuronii cu funcţii de activare sigmoid.

Cum impulsurile se pot descrie prin a.n. coordonate de „evenimente” (spaţiu, timp) şi numărul neuronilor activi (pulsanţi) este tipic redus, reţelele neuronale de neuroni pulsanţi artificiale permit implementarea unor reţele neuronale de dimensiuni mari (Mattia & Del Giudice, 2000). Prelucrarea timpilor mai multor impulsurilor individuale s-a propus de asemenea ca o nouă paradigmă pentru implementările de reţele neuronale VLSI care ar putea oferi o accelerare semnificativă. Arhitecturile de reţele bazate pe neuroni pulsanţi care codifică informaţia în timpii individuali ai impulsurilor au oferit, printre altele, o hartă de auto-organizare asemănător SOM Kohonen şi reţelelor de clustering nesupervizat (Natschläger & Ruf, 1998) (Bohte, Kok, & La Poutre, 2000).

Principiul codificării intensităţii de intrare prin timpii relativi de activare a fost de asemenea aplicat cu succes în cazul reţelelor de recunoaştere a caracterelor (Buonomano & Merzenich, 1999). Pentru aplicările reţelelor neuronale pulsative codificate temporal, totuşi, nu s-a dezvoltat un algoritm supervizat practic până în prezent. Chiar şi în (Buonomano & Merzenich, 1999), autorii recurg la

metode back-propagation de tip sigmoidal, tradiţionale pentru a învăţa reţeaua să distingă histogramele diferitelor răspunsuri de timpi ai impulsurilor. Pentru a face posibilă antrenarea supravegheată utilă cu modelul codificării temporale, s-a dezvoltat (Bohte, Kok, & La Poutr, 2002) un algoritm de antrenare pentru impulsuri individuale, care păstrează avantajele neuronilor pulsanţi, şi, în acelaşi timp, permite o antrenare la fel de performantă ca şi în cazul reţelelor neuronale sigmoid.

Se poate obţine astfel un algoritm de antrenare supravegheată bazată pe propagarea înapoi a erorii (error back-propagation), pentru reţelele neuronale pulsative care transferă informaţia în temporizarea unui singur impuls. Pentru a depăşi problema discontinuităţii neuronilor pulsanţi, se va aproxima funcţia limită. Algoritmul poate să înveţe să execute sarcini neliniare complexe, în reţele neuronale pulsative, cu aceeaşi precizie ca şi în cazul reţelelor neuronale sigmoid tradiţionale. Acest lucru se poate demonstra prin experimente pentru problema clasică de clasificare XOR, şi pentru câteva seturi de date reale.

Aceste rezultate pot prezenta interes pentru comunitatea de specialitate, pentru că posibilitatea codificării informaţiei în timpii impulsurilor se bucură de o atenţie din ce în ce mai mare. Se poate demonstra empiric, faptul că reţelele de neuroni pulsanţi cu un raţionament biologic pot să execute clasificarea neliniară complexă printr-o codificare temporală rapidă, la fel ca şi reţelele cu codarea ratei de impulsuri (CRI). Cu toate că acest capitol descrie o regulă de antrenare aplicabilă la o clasă a reţelelor neuronale artificiale, neuronii pulsanţi sunt mai aproape de modelul neuronului biologic, decât cei sigmoid.

Pentru a realiza calcule cu CRI într-un timp foarte scurt, se consideră că în sistemele neuronale biologice se adună răspunsurile unui număr mare de neuroni pulsanţi, pentru a obţine a măsură instantanee a ratei medii de activare. Chiar dacă necesită un număr mare de neuroni, acest răspuns totalizator este robust din cauza numărului mare de neuroni participanţi. Dacă comparăm un astfel de cod de rate instant cu codificarea temporală prin impulsuri individuale, se ştie că sunt necesari mult mai puţini neuroni, chiar dacă şi cu preţul scăderii robusteţii.

Conform unor studii din literatura de specialitate, s-au obţinut rezultate care susţin anticiparea faptului că lungimea segmentului în creştere al potenţialului post-sinaptic trebuie să fie mai mare decât structura temporală relevantă pentru a permite o computaţie temporală de încredere. Pentru neuronii pulsanţi, potenţialul post-sinaptic descrie dinamica unui impuls care influenţează un neuron, şi se poate modela ca şi o diferenţă a două funcţii descrescătoare exponenţial (Gerstner W. , Spiking neurons, 2001).

2.3.1. Propagarea înapoi a erorii

Am obţinut propagarea înapoi a erorii în mod analog cu rezultatele prezentate de (Rumelhart, Hinton, & Williams, 1986). Ecuaţiile au fost obţinute pentru o reţea neuronală pulsativă feed-forward total conectată cu straturi notate cu *H* (strat de intrare), *I* (strat ascuns) şi *J* (strat de ieşire), unde algoritmul rezultat se aplică la fel de bine pentru reţele cu mai multe straturi ascunse.

Scopul algoritmului este să înveţe un set de timpi de activare prescrişi, notaţi $\{t_j^d\}$, la neuronii de ieşire $j \in J$ pentru un set dat de modele de intrare $\{P[t_1..t_h]\}$, unde $P[t_1..t_h]$ ce defineşte un model de intrare individual descris de timpi de impuls individuali pentru fiecare neuron $h \in H$. Am ales ca funcţie de eroare media celor mai mici pătrate, dar alte variante ca entropie sunt de asemenea posibile. Având timpii de activare prescrişi $\{t_j^d\}$ şi timpii de activare măsuraţi $\{t_j^a\}$, această funcţie de eroare este definită de:

$$E = \frac{1}{2}\sum_{j \in J}(t_j^a - t_j^d)^2 \qquad \textit{Ec. 18}$$

Pentru propagarea înapoi a erorii vom trata fiecare terminal sinaptic ca o conexiune separată k cu ponderea w_{ij}^k. Astfel pentru regula propagării înapoi a erorii, trebuie să calculăm:

$$\Delta w_{ij}^k = -\eta \frac{\partial E}{\partial w_{ij}^k} \qquad \textit{Ec. 19}$$

unde η este rata de învăţare şi w_{ij}^k ponderea conexiunii k de la neuronul i la neuronul j.

Cum t_j este funcţie de x_j ce depinde de ponderile w_{ij}^k, derivata din partea dreapta a *Ec. 19* poate fi extinsă în:

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial t_j}\left(t_j^a\right)\frac{\partial t_j}{\partial w_{ij}^k}\left(t_j^a\right) = \frac{\partial E}{\partial t_j}\left(t_j^a\right)\frac{\partial t_j}{\partial x_j(t)}\left(t_j^a\right)\frac{\partial x_j(t)}{\partial w_{ij}^k}\left(t_j^a\right) \qquad \textit{Ec. 20}$$

În ultimii doi factori din dreapta, exprimăm t_j ca o funcţie mărginită a intrării post-sinaptice $x_j(t)$ în jurul lui $t = t_j^a$. Presupunem că pentru un interval suficient de mic în jurul lui $t = t_j^a$, funcţia x_j poate fi aproximată ca o funcţie liniară de t, cum se vede în Figura 2.5. Pentru un astfel de interval mic, aproximăm funcţia de prag $\partial t_j(x_j) = -\partial x_j(t_j)/\alpha$, cu $\frac{\partial t_j}{\partial x_j(t)}$ fiind derivata funcţiei inverse a $x_j(t)$. Valoarea lui α este derivata locală a $x_j(t)$ în punctul t, care este $\alpha = \frac{\partial x_j(t)}{\partial t}\left(t_j^a\right)$.

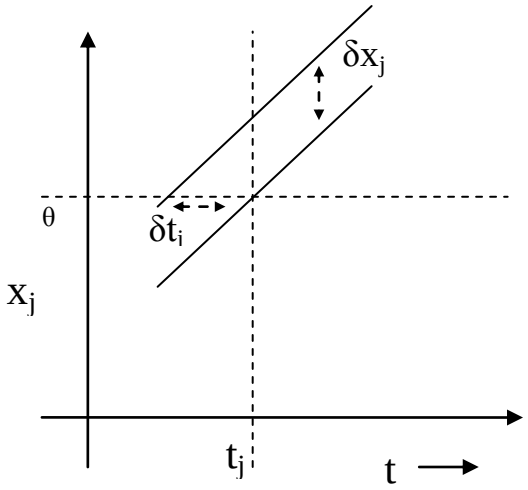


Figura 2.5 Relaţia dintre ∂x_j şi ∂t_j pentru un spaţiu ε în jurul t_j.

Al doilea factor din EEc. 20 poate fi scris ca:

$$\frac{\partial t_j}{\partial x_j(t)}\left(t_j^a\right) = \left.\frac{\partial t_j(x_j)}{\partial x_j(t)}\right|_{x_j=\vartheta} = \frac{-1}{\alpha} = \frac{-1}{\frac{\partial x_j(t)}{\partial t}\left(t_j^a\right)} = \frac{-1}{\sum_{i,l} w_{ij}^l \frac{\partial y_i^l(t)}{\partial t}\left(t_j^a\right)} \qquad \textit{Ec. 21}$$

În următoarele calcule vom scrie termeni ca $\frac{\partial x_j\left(t_j^a\right)}{\partial t_j^a}$ pentru $\frac{\partial x_j(t)}{\partial t}\left(t_j^a\right)$.

Observăm că această aproximare este valabilă numai dacă ponderile spre un neuron nu sunt modificate în aşa fel încât potenţialul de membrană să nu mai atingă valoarea de prag, şi în consecinţa neuronul să nu se mai activeze. Aceasta este o potenţială problemă, dar poate fi combătută prin codificarea intrărilor astfel încât primele impulsuri sunt în mod automat considerate „mai importante" decât următoarele impulsuri. Observăm însă că în cazul în care un neuron nu se activează pentru *oricare* din intrări, nu exista un mecanism care să corecteze ponderile. În experimentele noastre am setat ponderile iniţiale în aşa fel încât fiecare neuron din reţea a răspuns la cel puţin o parte din modelul de intrare. Cu această prevedere, nu am avut nici o problemă cu neuroni inactivi.

De asemenea observăm că această aproximare ar putea sugera că pentru rate de învăţare mari, algoritmul poate fi mai puţin eficient.

Primul factor în EEc. 20, derivata lui E în punctul t_j, este:

$$\frac{\partial E\left(t_j^a\right)}{\partial t_j^a} = \left(t_j^a - t_j^d\right) \qquad \textit{Ec. 22}$$

Avem:

$$\frac{\partial x_j\left(t_j^a\right)}{\partial w_{ij}^k}=\frac{\partial\left\{\sum_{n\in\Gamma_j}\sum_l w_{nj}^l y_n^l\left(t_j^a\right)\right\}}{\partial w_{ij}^k}=y_i^k\left(t_j^a\right) \qquad \textit{Ec. 23}$$

Dacă combinăm aceste rezultate, *Ec. 19* devine:

$$\Delta w_{ij}^k\left(t_j^a\right)=-\eta\frac{y_i^k\left(t_j^a\right)\cdot\left(t_j^d-t_j^a\right)}{\sum_{i\in\Gamma_j}\sum_l w_{ij}^l\frac{\partial y_i^l\left(t_j^a\right)}{\partial t_j^a}} \qquad \textit{Ec. 24}$$

Pentru comoditate, definim δ_j :

$$\delta_j\equiv\frac{\partial E}{\partial t_j^a}\frac{\partial t_j^a}{\partial x_j\left(t_j^a\right)}=\frac{\left(t_j^d-t_j^a\right)}{\sum_{i\in\Gamma_j}\sum_l w_{ij}^l\frac{\partial y_i^l\left(t_j^a\right)}{\partial t_j^a}} \qquad \textit{Ec. 25}$$

şi EEc. 20 poate fi acum exprimat ca:

$$\frac{\partial E}{\partial w_{ij}^k}=y_i^k\left(t_j^a\right)\frac{\partial E}{\partial t_j^a}\frac{\partial t_j^a}{\partial x_j\left(t_j^a\right)}=y_i^k\left(t_j^a\right)\delta_j \qquad \textit{Ec. 26}$$

de aici rezultă:

$$\Delta w_{ij}^k=-\eta y_i^k\left(t_j^a\right)\delta_j \qquad \textit{Ec. 27}$$

Ecuaţiile Ec. 25 şi Ec. 27 oferă funcţia simplă de adaptare a ponderilor pentru neuronii din stratul de ieşire.

Continuăm cu straturile ascunse: pentru propagarea înapoi a erorii în alte straturi decât cel de ieşire, eroarea delta generalizată în stratul I este definită pentru $i\in I$ cu timpii de activare măsuraţi t_i^a :

$$\delta_i\equiv\frac{\partial t_i^a}{\partial x_i\left(t_i^a\right)}\frac{\partial E}{\partial t_i^a}=\frac{\partial t_i^a}{\partial x_i\left(t_i^a\right)}\sum_{j\in\Gamma^i}\frac{\partial E}{\partial t_j^a}\frac{\partial t_j^a}{\partial x_j\left(t_j^a\right)}\frac{\partial x_j\left(t_j^a\right)}{\partial t_i^a}=\frac{\partial t_i^a}{\partial x_i\left(t_i^a\right)}\sum_{j\in\Gamma^i}\delta_j\frac{\partial x_j\left(t_j^a\right)}{\partial t_i^a} \qquad \textit{Ec. 28}$$

unde Γ^i denotă un set de succesori neuronali direcţi în stratul J conectaţi la neuronul i.

Ca şi în (Bishop, 1995), în Ec. 28 dezvoltăm eroarea locală $\frac{\partial E\left(t_i^a\right)}{\partial t_i^a}$ conform erorii ponderate corespunzătoare stratului următor J. Pentru această dezvoltare, aceleaşi reguli ca şi în EEc. 20 sunt folosite cu aceleaşi restricţii, dar pentru $t=t_i$.

Termenul $\frac{\partial t_i^a}{\partial x_i\left(t_i^a\right)}$ a fost calculat în Ec. 21, iar pentru $\frac{\partial x_j\left(t_j^a\right)}{\partial t_i^a}$

$$\frac{\partial x_j\left(t_j^a\right)}{\partial t_i^a}=\frac{\partial\sum_{l\in I}\sum_k w_{lj}^k y_l^k\left(t_j^a\right)}{\partial t_i^a}=\sum_k w_{ij}^k\frac{\partial y_i^k\left(t_j^a\right)}{\partial t_i^a} \qquad \textit{Ec. 29}$$

Deci,

$$\delta_i=\frac{\sum_{j\in\Gamma^i}\delta_j\left\{\sum_k w_{ij}^k\frac{\partial y_i^k\left(t_j^a\right)}{\partial t_i^a}\right\}}{\sum_{h\in\Gamma_i}\sum_l w_{hi}^l\frac{\partial y_h^l\left(t_i^a\right)}{\partial t_i^a}} \qquad \textit{Ec. 30}$$

Astfel pentru un strat ascuns şi conform Ec. 27, Ec. 28, Ec. 29 şi Ec. 30, regula adaptării ponderilor se compune în:

Analog cu algoritmii tradiţionali de propagare înapoi a erorii, regula adaptării ponderilor *Ec. 31*

$$\Delta w_{hi}^k=-\eta y_h^k\left(t_i^a\right)\delta_i=-\eta\frac{\sum_j\left\{\delta_j\sum_k w_{ij}^k\frac{\partial y_i^k\left(t_j^a\right)}{\partial t_i^a}\right\}}{\sum_{n\in\Gamma_i}\sum_l w_{ni}^l\frac{\partial y_n^l\left(t_i^a\right)}{\partial t_i^a}} \qquad \textit{Ec. 31}$$

de mai sus se generalizează pentru o reţea cu mai multe straturi ascunse I numerotate $J-1,...,2$ calculând eroarea delta la stratul i din eroarea delta în stratul $i+1$, propagând efectiv eroarea înapoi.

Algoritmul obţinut mai sus, numit *SpikeProp*, este rezumat în tabelul următor:

Tabelul 1 Algoritmul SpikeProp

1	Se calculează δ_j pentru toate ieşirile conform *Ec. 25*
2	Pentru fiecare strat ulterior $I = J-1,...,2$ Se calculează δ_j pentru toţi neuronii din I conform *Ec. 30*
3	Pentru stratul de ieşire J, ajustăm w_{ij}^k cu $\Delta w_{ij}^k = -\eta y_i^k(t_j)\delta_j$ *Ec. 27*
4	Pentru fiecare strat ulterior $I = J-1,...,2$ Ajustăm w_{hi}^k cu $\Delta w_{hi}^k = -\eta y_h^k(t_i)\delta_i$ *Ec. 31*

O simplă modificare a acestei scheme ar fi să includem un termen de inerţie:

$$\Delta w_{ij}^k = -\eta \Delta w_{ij}^k(t) + p w_{ij}^k(t-1) \qquad \textit{Ec. 32}$$

unde p este variabila de inerţie. Ca urmare a lucrării (Bohte, Kok, & La Poutré, 2000), Xin *et al.* (Xin & Embrechts, 2001) au arătat că asemenea modificări ale algoritmului *SpikeProp* accelerează într-adevăr convergenţa în mod semnificativ.

2.3.2. Problema XOR

În continuare, aplicăm algoritmul *SpikeProp* problemei XOR. Funcţia XOR este un exemplu clasic de problemă neliniară care necesită unităţi ascunse pentru a transforma intrarea in ieşirea dorită.

Pentru a codifica funcţia XOR în modele impuls-timp, am asociat valoarea 0 cu o activare „întârziată" şi valoarea 1 cu o activare „prematură". Cu valorile specifice 0 şi 6 pentru timpii de intrare respectivi, vom folosi următorul XOR codificat temporal:

Tabelul 2 Codificarea modelelor de intrare şi ieşire al problemei XOR rezolvată cu algoritmul Spikeprop

Modele de intrare	Modele de ieşire
0	16
6	10
0	10
6	16

Numerele în tabel reprezintă durata impulsurilor, să spunem în milisecunde. Vom folosi un al treilea neuron de intrare (offset) în reţea care se va activa întotdeauna la $t = 0$ pentru a marca timpul de referinţă (altfel problema devine banală). Definim diferenţa dintre timpii echivalenţi cu *„0"* şi *„1"* ca fiind intervalul de codificare ΔT, care în acest exemplu corespunde cu 6ms.

Pentru reţea vom folosi o arhitectură de tip feed-forward. Conexiunile au un interval de întârziere de 15 ms; astfel vom avea întârzieri sinaptice între 1 şi 16 ms. Potenţialul post-sinaptic (PPS) este definit de o funcţie α ca în

$$\varepsilon(t) = \frac{t}{\tau} e^{1-\frac{t}{\tau}} \qquad \textit{Ec. 33}$$

cu un timp de scădere $\tau = 7$ ms. Valori mai mari până la cel puţin 15 ms rezultă într-o învăţare similară.

Reţeaua a fost compusă din 3 neuroni de intrare (2 neuroni de codificare şi 1 neuron de referinţă), 5 neuroni ascunşi (dintre care un neuron inhibitor generând doar PPS-uri negative) şi 1

neuron de ieşire. Au fost permise doar ponderi pozitive. Cu această configuraţie reţeaua a învăţat în mod fiabil modelul XOR în timpul a 250 de cicluri cu $\eta = 0.01$. Pentru a „învăţa" XOR, au trebuit să fie ajustate 16 x 3 x 5 + 16 x 5 x 1 = 320 de ponderi individuale.

În timpul folosirii algoritmului s-a descoperit că este necesară introducerea specifică a neuronilor inhibitori şi excitatori, neuronii inhibitori şi excitatori fiind definiţi ca generând PPS-uri cu valori negative şi respectiv pozitive, folosind doar ponderi pozitive. De fapt, algoritmul *SpikeProp* nu va converge spre un rezultat satisfăcător dacă se permite ca conexiunile să conţină un amestec de ponderi pozitive şi negative. Cauza acestei probleme ar putea fi că în cazul în care avem valori atât pozitive cât şi negative, efectul unei conexiuni individuale către un neuron ţintă nu mai este o funcţie monoton crescătoare (cum ar fi pentru constante de timp suficient de mari). S-a observat că introducerea de neuroni inhibitori şi excitatori nu este o restricţie: dacă extindem reţeaua în aşa fel încât fiecărui neuron excitator să-i corespundă un neuron inhibitor, vom obţine de fapt o conexiune mixtă. În experimente însă, includerea unui sau a doi neuroni inhibitori a fost suficientă pentru a realiza învăţarea.

2.3.3. Gradientul erorii şi rata de învăţare

Considerăm influenţa ratei de învăţare η şi constanta de timp τ pe capacităţile de învăţare ale algoritmului *SpikeProp* într-o reţea neuronală pulsativă.

Cum s-a văzut în secţiunea anterioară, aproximarea dependenţei de timpul de activare t_j^a pe intrarea post-sinaptică x_j este valabilă doar pentru un interval mic în jurul lui t_j^a. S-a descoperit că pentru rate de învăţare mari probabilitatea convergenţei scade, însă pentru rate de învăţare până la 0.01, creşterea ratei de învăţare corespunde cu timpi mai rapizi de învăţare. Asta se poate vedea în

Figura 2.6, unde numărul mediu de iteraţii de învăţare necesare pentru funcţia XOR este reprezentat pentru un număr de constante de timp τ.

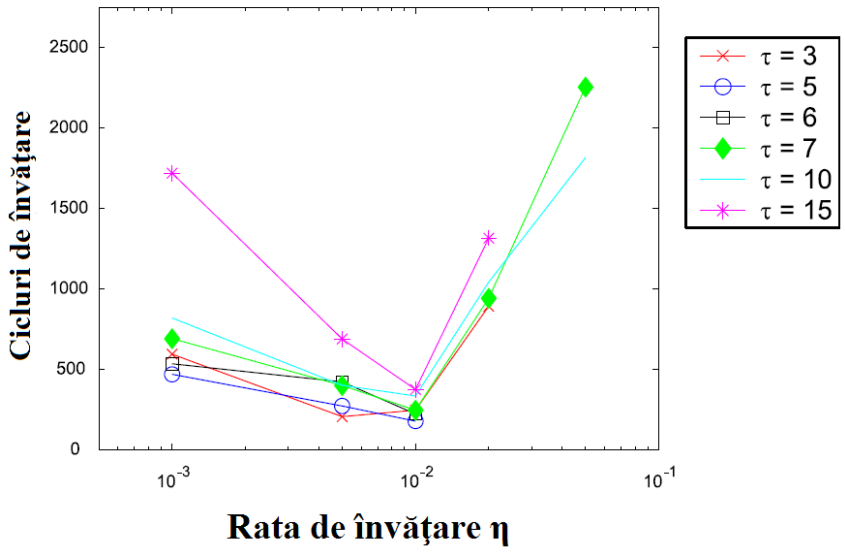


Figura 2.6 Învăţare XOR: numărul mediu necesar de iteraţii de învăţare pentru a atinge suma propusă a erorilor pătratice de 1.0. Media a fost calculată pentru cazurile în care s-a convers la un rezultat satisfăcător.

În Figura 2.7 este reprezentată fiabilitatea învăţării pentru diferite valori ale constantei de timp τ. Reprezentarea arată că cele mai sigure rezultate pentru o convergenţă optimă au fost obţinute pentru valori ale constantei de timp mai mari (cu puţin) decât intervalul de timp ΔT în care se codifică problema XOR (aici $\Delta T = 6$).

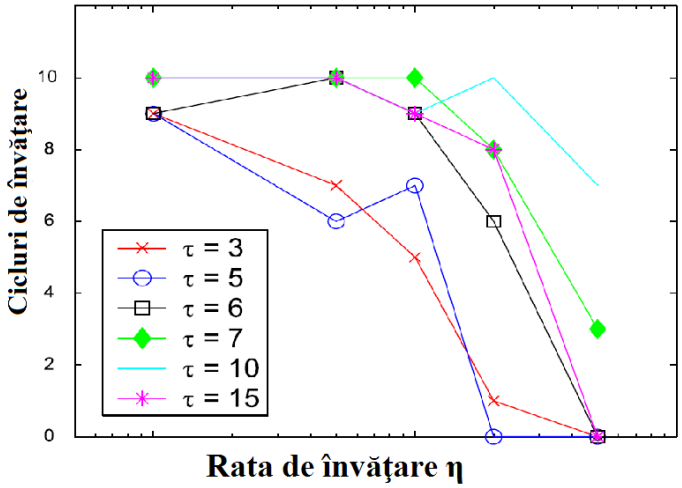


Figura 2.7 Învăţare XOR: Numărul de cazuri din 10 care au convers

Graficele de convergenţă pentru diferite valori ale intervalului de codificare ΔT sunt reprezentate în Figura 2.8 şi arată acelaşi comportament. Aceste rezultate confirmă rezultatele obţinute în (Maass, 1996), unde s-a demonstrat ca teoretic constanta de timp τ trebuie să fie mai mare decât intervalul de codificare relevant ΔT.

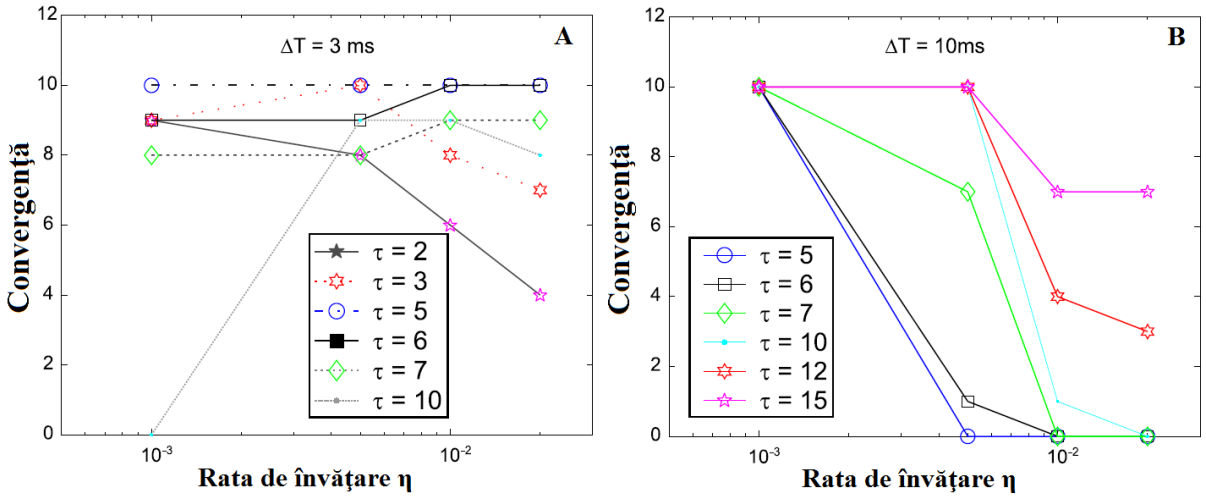


Figura 2.8 Numărul de cazuri din 10 care au convers pentru diferite valori ale τ.

A) Pentru un interval de codificare $t = 0...3, [0,3]^2 \rightarrow [7,10]$. *Suma propusă a erorilor pătratice de 0.7.*

B) Pentru un interval de codificare $t = 0...10, [0,10]^2 \rightarrow [15,25]$. *Suma propusă a erorilor pătratice de 3.0.*

2.4. Dezvoltarea modelelor neuronale neuromorfe

În concordanţă cu cele prezentate în prealabil, putem afirma că în creierul uman elementul de primordial care stă la baza prelucrării informaţiei este neuronul pulsativ. Acestea sunt unităţi specializate în efectuarea unor "calcule" şi sunt total diferite de cele utilizate în dezvoltarea reţelelor neuronale artificiale. Un neuron pulsativ emite impulsuri la un anumit moment în timp, adică se aprinde. Astfel, neuronul transmite un semnal electric prin axonul său – spre neuronii la care este conectat – numit potenţial de aprindere, sau impuls de aprindere sau de activare (IA). Forma şi magnitudinea IA este independentă de semnalul de intrare a neuronului, însă temporizarea acesteia depinde de intrare.

Poate fi vorba de orice fel de calculator artificial, de multe ori ţinem ca o anumită ordine în efectuarea paşilor de calcul să fie respectată, ea încadrându-se într-o schemă temporală globală de lucru. De exemplu, în cazul unei reţele neuronale artificiale de tip *feed-forward*, aşteptăm de la un neuron al

stratului *x*, ca ieşirea lui să se activeze la momentul *x* de calcul, independent de semnalele aplicate la intrarea reţelei. În contrast cu acest aspect, în cazul unei reţele neuronale neuromorfe (similare cu cele naturale) momentele de activare ale neuronilor sunt determinate de intrarea reţelei. Reţelele bazate pe astfel de neuroni sunt capabile să exploateze timpul ca o resursă de codare şi calcul, mult mai eficient decât majoritatea modelelor de calcul existente.
În consecinţă, analiza reţelelor neuronale bazate pe astfel de modele ne determină să reevaluăm rolul timpului şi spaţiului, care sunt dimensiunile de bază ale oricărui calcul. Unele cercetări în acest sens au avut ca efect stimularea dezvoltării de elemente electronice hardware noi, care utilizează impulsuri pentru a efectua calcule. Astfel de elemente sunt de exemplu, circuitele VLSI analogice utilizate în sisteme senzoriale în timp-real.

2.4.1. Modelul unei reţele neuronale artificiale formale cu neuroni pulsativi

Să considerăm o reţea neuronală pulsativă (RNP) (Maas W. , 1997) (Maas W. , 1998), formată dintr-o mulţime finită *V* de neuroni pulsanţi (NP) şi o mulţime $E \subseteq V \times V$ formată de sinapse.

Fiecărui neuron $i \in V$, corespunde o *funcţie de prag* $\vartheta_i : R^+ \to R^+$ (Figura 2.9) şi fiecărei sinapse $\langle i, j \rangle \in E$ corespunde o *funcţie de răspuns* $\varepsilon_{i,j} : R^+ \to R$, unde $R^+ = \{x \in R : x \geq 0\}$ (Figura 2.10)

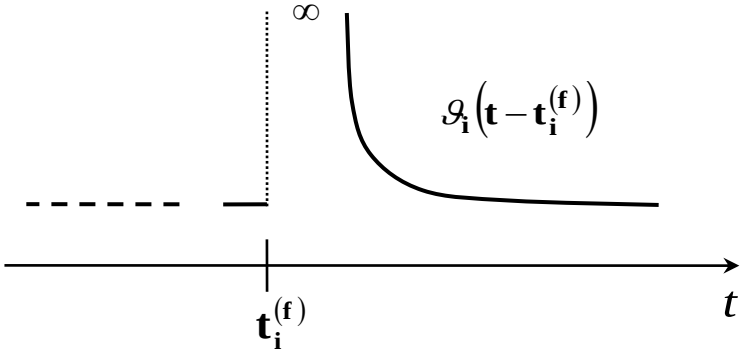


Figura 2.9 Forma funcţiei de prag al unui neuron natural

Un NP *j* emite IA în momentele de timp $t^{(1)}_j$, $t^{(1)}_j$,..., adică emite un impuls stereotipic de-a lungul structurii axonale arborescente. În sinapse, care realizează legătura dintre neuronul *j* şi alţi neuroni, IA-ul emis declanşează producerea unei materii denumite neuro-transmiţător. Ca urmare se generează impulsuri pozitive (potenţiale post-sinaptice excitatorii, PPSE) sau negative (potenţiale post-sinaptice inhibitorii, PPSI) care vor modifica potenţialul membranei neuronului post-sinaptic.
Caracteristica de timp a PPSE şi PPSI-urilor cauzate de IA al neuronului *j* se poate exprima în felul următor $\omega_{ij}\varepsilon\left(t - t_j^{(f)}\right)$, unde valoare funcţiei de răspuns $\varepsilon\left(t - t_j^{(f)}\right)$ este 0 dacă $(t-t_j^{(f)})$ este mai mic decât, timpul de propagare Δ_{ij}, apoi creşte aproape liniar şi în final descreşte exponenţial spre 0. Factorul ω_{ij} este o pondere, care are valoare pozitivă în cazul unor PPSE-uri şi valoare negativă în cazul PPSI-urilor.

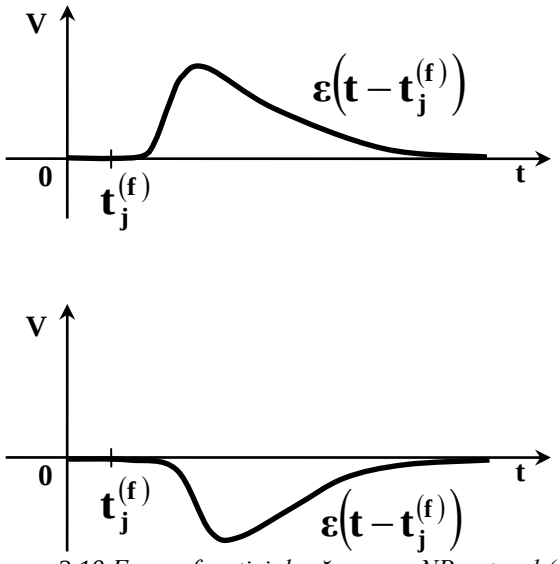


Figura 2.10 Forma funcţiei de răspuns a NP natural (sus PPSE şi PPSI jos)

Există numeroase modele matematice, care încearcă să identifice momentele de timp $t_i^{(1)}, t_i^{(2)}, \ldots$ ale apariţiei de IA-uri la neuronii biologici, în funcţie de momentele $t_j^{(f)}$ în care un neuron pre-sinaptic *j* emite un IA. Datorită relativei simplităţi matematice care duce şi la o mai uşoară implementare software şi hardware, *modelul de reacţie la impuls* este unul dintre modelele cele mai convenabile pentru aplicarea în analize computerizate.

2.4.2. Modelul de reacţie la impuls

În cazul modelului de reacţie la impuls (MRI) potenţialul de membrană $u_i^{(t)}$ al somei unui neuron *i* la un moment dat *t* , se poate exprima în felul următor

$$\sum_{j,f} = \omega_{ij} \varepsilon\left(t - t_j^{(f)}\right) \qquad \textit{Ec. 34}$$

sumă care alcătuită din PPSE şi PPSI-uri emise de neuronul pre-sinaptic *j* în momente de timp $t_j^{(f)}$ premergătoare momentului *t.*

Pe de altă parte în MRI definim şi un *factor de inhibare sau înăbuşire*, $\eta\left(t - \hat{t}_i\right)$, care este o funcţie de $\hat{t}_i$, adică momentul premergător lui *t* în care neuronul *i* a emis ultimul IA. Dacă $\left(t - \hat{t}_i\right)$ este pozitiv, dar de valoare foarte mică (sub 2 ms), atunci factorul $\eta\left(t - \hat{t}_i\right)$ arată o valoare puternic negativă, iar în cazul în care $\left(t - \hat{t}_i\right)$ este mai mare *factorul de inhibare* tinde spre zero.

Deci, în cazul modelului de reacţie la impuls neuronul *i* se aprinde atunci şi numai atunci, când valoarea sumei rezultante atinge un prag ϑ, iar derivata de ordinul unu al acestei expresii este pozitivă, cu alte cuvinte ne apropiem de prag în sens crescător.

Modelul se poate utiliza mai eficient în aplicaţii de calcul, dacă presupunem, că potenţialul de repaus al neuronilor este 0, iar valoarea de prag este întotdeauna mai mare decât zero ($\vartheta > 0$). În realitate, potenţialul de repaus al unui neuron natural are o valoare aproximativă de –70mV iar cel de prag este în jurul valorii de –50 mV. Efectul PPSE şi PPSI-urilor asupra acestor valori este de ordinul câtorva milivolţi.

Comparând acest model cu modelul clasic *Hodkin-Huxley* (Trappenberg, 2002) (Maas & Bishop, 1999) putem spune că MRI este mult mai uşor de utilizat în analize de calcul, deoarece în definiţia acestuia nu am aplicat ecuaţii diferenţiale. Pe de altă parte, dacă alegem corespunzător valorile pentru funcţiile η , ϑ şi ε , MRI este capabil de aproxima foarte bine dinamica neuronilor biologici.

2.5. Simularea reţelelor neuronale pulsative: o sinteză a instrumentelor şi a strategiilor

2.5.1. Introducere

Dovezile experimentale au arătat că sincronizarea impulsurilor poate fi importantă pentru a explica modul în care se realizează calculele la nivel de neuron, motivând astfel folosirea modelelor cu neuroni pulsanţi în locul modelelor tradiţionale bazate pe rata impulsurilor. În acelaşi timp a apărut un număr din ce în ce mai mare de instrumente ce permit simularea reţelelor neuronale pulsative. Aceste instrumente oferă utilizatorului posibilitatea să obţină simulări precise ale unui sistem computaţional dat, precum şi rezultate publicabile într-un timp relativ scurt. Problemele de calcul legate de neuronii pulsanţi sunt foarte variate însă. În unele cazuri este necesară folosirea unei reprezentări biofizice detaliate a neuronilor, de exemplu când se doreşte reproducerea măsurătorilor electrofiziologice intracelulare (Destexhe & Sejnowski, 2001). În aceste cazuri se folosesc modele bazate pe conductanţă (COBA), cum ar fi tipul de model folosit de Hodgkin şi Huxley (Hodgkin & Huxley, 1952). În alte

cazuri nu se doreşte să se surprindă exact mecanismele prin care se generează impulsul, şi sunt suficiente modele mai simple, cum ar fi modelul „integrează şi activează" (integrate-and-fire – IF). Simularea modelelor de tip IF este de asemenea foarte rapidă, făcându-le în special atractive pentru simularea reţelelor neuronale la scară mare.

Există două familii de algoritmi pentru simularea reţelelor neuronale: algoritmi sincroni sau „bazaţi pe tact", în care toţi neuronii sunt actualizaţi simultan la fiecare tact, şi algoritmi asincroni sau „bazaţi pe eveniment", în care neuronii sunt actualizaţi doar când primesc sau transmit un impuls (există de asemenea strategii hibride). Pentru algoritmii sincroni codul este uşor de scris şi se poate aplica oricărui model. Deoarece timpii de impuls sunt legaţi de o grilă discretă de timp, precizia simulării poate fi o problemă. Algoritmii asincroni au fost dezvoltaţi în special pentru simulări exacte, ceea ce este posibil pentru modele simple. Pentru reţele foarte mari, durata simulării prin ambele metode creşte foarte mult odată cu numărul impulsurilor transmise, dar fiecare strategie are propriile avantaje şi dezavantaje.

În cele ce urmează vom trece în revistă diferite strategii de simulare, şi vom sublinia în ce măsură precizia în timp a evenimentelor impulsuri au impact asupra unui sau a unei mici reţele de neuroni de tip IF cu sinapse plastice. Apoi vom trece în revistă simulatoarele sau mediile de simulare disponibile în prezent, punând accent numai pe instrumentele publice si non-comerciale pentru simularea reţelelor neuronale pulsative.

2.5.2. Strategii de simulare

Ne limităm la algoritmi seriali pentru precizie. Secţiunile specifice NEST şi SPLIT oferă informaţii în plus despre conceptele pentru calculul în paralel.

Există două familii de algoritmi pentru simularea reţelelor neuronale: algoritmi sincroni sau „bazaţi pe tact", în care toţi neuronii sunt actualizaţi simultan la fiecare tact, şi algoritmi asincroni sau „bazaţi pe eveniment", în care neuronii sunt actualizaţi doar când primesc sau transmit un impuls. Aceste două abordări au unele proprietăţi comune pe care le vom descrie întâi exprimând problema simulării reţelelor neuronale în formalismul sistemelor hibride, de ex. ecuaţii diferenţiale cu evenimente discrete (impulsuri). În aceste condiţii apar unele strategii comune pentru o reprezentare şi simulare eficiente.

Deoarece vom compara algoritmii din punct de vedere al eficienţei de calcul, să ne punem întâi următoarea întrebare: cât timp Îi poate lua unui algoritm bun să simuleze o reţea mare? Să presupunem că avem N neuroni cu rata medie de activare F şi numărul mediu de sinapse p. Dacă toate impulsurile transmise sunt luate în calcul, atunci o simulare care durează 1s (timp biologic) trebuie să proceseze N x p x F transmisii. Scopul unui algoritm eficient este să atingă acest număr minim de operaţii (desigur până la un factor constant multiplicativ). Dacă simularea nu este restricţionată la interacţiuni imediate de impulsuri, de ex. dacă modelul include fante sinaptice sau interacţiuni dendro-dendritice, atunci numărul optim de operaţii poate fi mult mai mare, dar nu vom lua în seamă aici problema interacţiunilor gradate.

2.5.2.1. Formalismul unui sistem hibrid

Matematic, neuronii pot fi descrişi ca fiind sisteme hibride: starea unui neuron evoluează continuu conform unor ecuaţii biofizice, care sunt de obicei ecuaţii diferenţiale (deterministice sau stocastice, ecuaţii ordinare sau parţial diferenţiale), şi impulsurile recepţionate prin sinapse declanşează schimbări în unele variabile. Astfel dinamica unui neuron poate fi descrisă cum urmează:

$$\frac{dX}{dt} = f(X) \qquad \textit{Ec. 35}$$

$$X \leftarrow g_i(X) \text{ la un impuls recepţionat prin sinapsa i} \qquad \textit{Ec. 36}$$

unde X este un vector care descrie starea neuronului. În teorie, luând în calcul morfologia neuronului ar duce la ecuaţii parţial diferenţiale; însă, în practică se poate aproxima arborele dendritic prin compartimente iso-potenţiale cuplate, ceea ce va duce de asemenea la un sistem diferenţial cu evenimente discrete. Impulsurile sunt emise când o anumită condiţie de activare (threshold) este satisfăcută, de exemplu $V_m \geq \theta$ pentru modelele de tip IF (unde V_m este potenţialul membranei, şi ar fi prima componentă a vectorului X), şi / sau $dV_m/dt \geq \theta$ pentru modelele de tip Hodgkin-Huxley (HH). Putem rezuma spunând că un impuls este emis de fiecare dată când o anumită condiţie $X \in A$ este

îndeplinită. Pentru modelele de tip IF, potenţialul membranei, care ar fi prima componentă a X, se resetează când se produce un impuls. Resetarea poate fi integrată în formalismul sistemului hibrid dacă considerăm de exemplu că impulsurile de ieşire acţionează asupra X printr-o sinapsă adiţională (virtuală): $X \leftarrow g_0(X)$.

Cu acest formalism devine evident că *timpii impulsurilor nu trebuiesc stocaţi* (exceptând desigur cazurile în care sunt incluse întârzierile de transmisie), chiar daca formulările fenomenologice ar sugera contrariul. Spre exemplu, considerăm următorul model de tip IF descris în (Gütig & Sompolinsky, 2006):

$$V(t) = \sum_i \omega_i \sum_{t_i} K(t - t_i) + V_{rest} \qquad \textit{Ec. 37}$$

unde $V(t)$ este potenţialul membranei, V_{rest} este potenţialul de repaus, ω_i este ponderea sinaptică a sinapsei i, t_i sunt timpii impulsurilor venite de la sinapsa i, şi $K(t - t_i) = \exp(-(t - t_i)/\tau) - \exp(-(t - t_i)/\tau_S)$ este potenţialul post-sinaptic (PSP) la care contribuie fiecare impuls primit. Modelul poate fi descris ca un sistem diferenţial de două variabile cu evenimente discrete cum urmează:

$$\tau \frac{dV}{dt} = V_{rest} - V + J \qquad \textit{Ec. 38}$$

$$\tau_S \frac{dJ}{dt} = -J$$

$$J \leftarrow J + \frac{\tau - \tau_S}{\tau} w_i \text{ la un impuls recepţionat prin sinapsa } i$$

Practic putem exprima în acest mod fiecare PSP sau curenţi descrişi în literatură (de ex. funcţii α, funcţii bi-exponenţiale). Mai mulţi autori au descris transformarea de la expresii fenomenologice la formalismul unui sistem hibrid pentru conductanţe sinaptice si curenţi ((Destexhe, Mainen, & Sejnowski, 1994), (Destexhe, Mainen, & Sejnowski, 1994); (Rotter & Diesmann, 1999); (Giugliano, 2000)), depresiuni sinaptice pe termen scurt (Giugliano, Bove, & Grattarola, 1999), şi plasticitatea dependenţei de timpii impulsurilor (Song, Miller, & Abbott, 2000). În multe cazuri, modelul răspunsului la impuls (Gerstner & Kistler, 2002) este de asemenea expresia integrală a unui sistem hibrid. Pentru a deduce formularea diferenţială a unui curent sau conductanţă post-sinaptică dată, o metodă ar fi de a considera ultima ca răspunsul la impuls al unui sistem liniar invariabil în timp (care poate fi considerat ulterior ca un filtru (Jahnke, Roth, & Schoenauer, 1998)) şi să folosim instrumentele de transformare din teoria procesării semnalelor cum ar fi transformata Z (Kohn & Wörgötter, 1998) (a se vedea de asemenea (Sanchez-Montanez, 2001)) sau transformata Laplace (transformata Z este echivalenta transformatei Laplace în domeniul timpului digital, de ex. pentru algoritmi sincroni).

2.5.2.2. Folosirea liniarităţilor pentru simulări sinaptice rapide

În general, numărul de variabile de stare a unui neuron (lungimea vectorului X) creşte odată cu numărul sinapselor, deoarece fiecare sinapsă are propria sa dinamică. Acest fapt constituie o problemă majoră pentru simularea eficientă a reţelelor neuronale, atât în privinţa memoriei consumate cât şi a timpului de calcul. Cu toate acestea, mai mulţi autori au observat că toate variabilele sinaptice caracterizate de aceeaşi dinamică liniară pot fi reduse la o singură variabilă (Wilson & Bower, 1989); (Bernard, Ge, Stockley, Willis, & Wheal, 1994); (Lytton, 1996); (Song, Miller, & Abbott, 2000). De exemplu, următorul set de ecuaţii diferenţiale, descriind un model de tip IF cu n sinapse având conductanţe exponenţiale:

$$C \frac{dV}{dt} = V_0 - V + \sum_i g_i(t)(V - E_S) \qquad \textit{Ec. 39}$$

$$\tau_S \frac{dg_1}{dt} = -g_1 \quad \ldots \qquad \tau_S \frac{dg_n}{dt} = -g_n$$

$$g_i \leftarrow g_i + w_i \text{ la un impuls recepţionat prin sinapsa } i$$

este matematic echivalent cu următorul set de două ecuaţii diferenţiale:

$$C\frac{dV}{dt} = V_0 - V + g(t)(V - E_S) \qquad \textit{Ec. 40}$$

$$\tau_S \frac{dg}{dt} = -g$$

$g \leftarrow g + w_i$ la un impuls recepţionat prin sinapsa i

unde g este conductanţa sinaptică totală. Aceeaşi reducere se aplică sinapselor cu dinamică dimensională mai mare, atât timp cât este liniară şi schimbările determinate de impulsuri ($g_i \leftarrow g_i + w_i$) sunt aditive şi nu depind de starea sinapsei (ex. regula $g_i \leftarrow g_i + w_i * f(g_i)$ ar cauza o problemă). Unele modele de plasticitate dependentă de timpii impulsurilor (cu interacţiuni liniare între perechile de impulsuri) pot fi de asemenea simulate în acest mod (Abbott & Nelson, 2000). Însă, unele modele biofizice importante nu sunt liniare şi astfel nu pot beneficia de această optimizare, în particular interacţiunile mediate NMDA şi sinapsele saturate.

2.5.2.3. Algoritmi sincroni sau bazaţi pe tact

Într-un algoritm sincron sau bazat pe tact (vezi pseudocodul din Figura 2.11), starea variabilelor fiecărui neuron (şi posibil a sinapselor) este actualizată la fiecare tact: $X(t) \rightarrow X(t+dt)$.

```
t=0
while t<duration
      for every neuron                          Actualizări stare
            process incoming spikes
            advance neuron dynamics by dt
      end

      for every neuron
            if vm>treshold
                  reset neuron                  Propagare impulsuri
                  for every connection
                        send spike
                  end
            end
      end

      t=t+dt
end
```

Figura 2.11 Un algoritm simplu bazat pe tact

Având ecuaţii diferenţiale neliniare, se poate folosi o metodă de integrare cum ar fi Euler sau Runge-Kutta (Press, Flannery, Teukolsky, & Vetterling, 1993) sau, pentru modele HH, metode implicite (Hines, 1984). Neuronii cu morfologii complexe sunt de obicei separaţi spaţial şi modelaţi ca compartimente interactive: sunt de asemenea descrişi matematic de perechi de ecuaţii diferenţiale, pentru care s-au dezvoltat metode de integrare speciale (pentru detalii vezi de ex. secţiunea specifică Neuron în cele ce urmează). Dacă ecuaţiile diferenţiale sunt liniare, atunci operaţiunea de actualizare $X(t) \rightarrow X(t+dt)$ este de asemenea liniară, ceea ce înseamnă că actualizarea variabilelor de stare se realizează pur şi simplu înmulţind X cu o matrice: $X(t+dt) = AX(t)$ (Hirsch & Smale, 1974) (vezi de asemenea (Rotter & Diesmann, 1999), pentru o aplicaţie în reţelele neuronale), ceea ce este foarte convenabil în medii ştiinţifice bazate pe vectori cum ar fi Matlab sau Scilab. Apoi, după ce s-au actualizat toate variabilele, condiţia de activare este verificată pentru fiecare neuron. Fiecare neuron ce satisface această condiţie produce un impuls care se va transmite către neuronii ţintă, actualizând variabilele corespunzătoare ($X \leftarrow g_i(X)$). Pentru modele IF, potenţialul membranei fiecărui neuron care pulsează va fi actualizat.

Complexitatea calculelor. Timpul de simulare al unui astfel de algoritm constă în două etape: (1) actualizări de stare şi (2) propagare de impulsuri. Presupunând că numărul de variabile de stare pentru întreaga reţea creşte odată cu numărul de neuroni *N* din reţea (ceea ce se întâmplă când se aplică simplificarea descrisă în secţiunea 2.5.2.2), costul fazei de actualizare este de ordinul *N* pentru fiecare pas, deci este $O(N/dt)$ pe secundă de timp biologic (dt este durata pasului temporal). Această componentă creşte odată cu complexitatea modelelor neuronale şi cu precizia simulării. În fiecare secundă (de timp biologic) o medie de *F* x *N* impulsuri sunt produse de către neuroni (*F* este media ratei de activare) şi fiecare impuls trebuie propagat către *p* neuroni ţintă. Astfel faza de propagare constă în *F* x *N* x *p* impulsuri propagate pe secundă. Acestea sunt în esenţă adăugări de ponderi w_i variabilelor de stare, în consecinţă operaţii simple al căror cost nu creşte odată cu complexitatea modelelor. Însumând, costul total al calculelor efectuate pe secundă de timp biologic este de ordinul

$$\begin{array}{ccc} \textit{Actualizare} & + & \textit{Propagare} \\ c_U \times \frac{N}{dt} & + & c_P \times F \times N \times p \end{array} \qquad \textit{Ec. 41}$$

unde c_U este costul unei actualizări şi c_P este costul unei propagări de impuls; de obicei c_U este mult mai mare decât c_P dar asta depinde de implementare. Astfel, pentru reţele foarte dense, totalul este dominat de faza de propagare şi este liniar în numărul de sinapse, ceea ce e optim. În practică însă, prima fază este neglijabilă numai când se îndeplineşte următoarea condiţie:

$$\frac{c_P}{c_U} \times F \times p \times dt >> 1 \qquad \textit{Ec. 42}$$

De exemplu, rata medie de activare în cortex poate fi redusă ca $F = 1\text{Hz}$ (Olshausen & Field, 2005), şi presupunem că $p = 10000$ sinapse pe neuron şi $dt = 0.1\text{ms}$, vom avea $F \times p \times dt = 1$. În acest caz, considerând că fiecare operaţie în faza de actualizare este mai grea decât în faza de propagare (în special pentru modele complexe), de ex., $c_P < c_U$, este probabil ca faza de actualizare să fie partea dominantă în costul total al calculelor. Astfel se pare că chiar şi în reţele cu conectivitate realistă, creşterea preciziei (dt mai mic) poate fi în detrimentul eficienţei simulării.

2.5.2.4. Algoritmi asincroni sau bazaţi pe eveniment

Algoritmii asincroni sau bazaţi pe eveniment nu sunt la fel de folosiţi ca şi cei bazaţi pe tact deoarece implementarea lor este mai complicată (vezi pseudocodul din Figura 3) şi mai puţin generală. Avantajele cheie ale algoritmilor asincroni constă intr-un posibil câştig în viteză prin faptul că nu vom mai calcula mulţi paşi mici de actualizare pentru un neuron în care nu se produce nici un eveniment şi pentru că timpii impulsurilor sunt calculaţi în mod exact (dar se poate vedea mai jos pentru algoritmi bazaţi pe eveniment aproximaţi); în particular timpii impulsurilor nu mai sunt aliniaţi unei grile (ceea ce este o posibilă sursă de erori).

Problema simulării sistemelor dinamice cu evenimente discrete este un subiect de cercetare bine stabilit în domeniu (Ferscha, 1996); (Sloot, Kaandorp, Hoekstra, & Overeinder, 1999); (Fujimoto, 2000); (Zeigler, Praehofer, & Kim, 2000) (vezi de asemenea (Rochel & Martinez, 2003); (Mayrhofer, Affenzeller, Prähofer, Hfer, & Fried, 2002)), cu structuri de date adecvate şi algoritmi deja disponibili pentru cercetătorii din domeniu. Vom începe prin a descrie cazul simplu în care interacţiunile sinaptice sunt instantanee, de ex. când impulsurile se pot produce doar în momentul recepţionării unui impuls (fără întârzieri); apoi vom aborda cazul cel mai general.

Interacţiuni sinaptice instantanee. Într-un algoritm asincron sau bazat pe eveniment simularea avansează de la un eveniment la următorul eveniment. Evenimentele pot fi impulsuri emise de neuronii din reţea sau impulsuri externe (de obicei impulsuri întâmplătoare descrise de un proces Poisson). Pentru modele în care impulsurile pot fi produse de către un neuron doar în momentul recepţionării unui alt impuls, simularea bazată pe eveniment este relativ uşoară (vezi pseudocodul din Figura 2.12).

Evenimentele sunt stocate într-o coadă de aşteptare (care este un fel de listă ordonată). O iteraţie constă în:

1. Extragerea următorului eveniment
2. Actualizarea stării neuronului corespunzător (de ex. calcularea conform ecuaţiei diferenţiale şi adăugarea ponderii sinaptice)
3. Se verifică dacă neuronul satisface condiţia de activare, caz în care se introduc în coada de aşteptare evenimente pentru fiecare neuron post-sinaptic

```
while queue not empty and t<duration
      extract event with lowest timing          Procesare eveniment
      (= timing t, target i, weight w)
      compute state of neuron i at time t
      update state of neuron i (+w)
      if vm>treshold
            for each connection i -> j          Propagare impuls
                  insert event in the queue
            end
            reset neuron i
      end
end
```

Figura 2.12 Un algoritm simplu bazat pe eveniment cu interacţiuni sinaptice

În cazul simplu al întârzierilor de transmisie identice, structura de date pentru coada de aşteptare poate fi pur şi simplu o listă FIFO (First In, First Out – primul intrat, primul ieşit), care se poate implementa rapid (Cormen, Leiserson, Rivest, & Stein, 2001). Când întârzierile iau valori dintr-o mulţime discretă de dimensiuni mici, cel mai uşor este să folosim o listă FIFO pentru fiecare valoare de întârziere, aşa cum e descris în (Mattia & Del Giudice, 2000). Este de asemenea mai eficient să folosim o listă FIFO separată pentru a gestiona evenimentele externe diverse.

În cazul întârzierilor arbitrare, este nevoie de o structură de date mai complexă. În informatică, structurile de date care menţin în mod eficient o listă ordonată de evenimente marcate în timp sunt grupate sub numele de liste de prioritate (priority queues) (Cormen, Leiserson, Rivest, & Stein, 2001). Subiectul listelor de prioritate este vast şi bine documentat; exemple sunt stive binare, stive Fibonacci (Cormen, Leiserson, Rivest, & Stein, 2001), liste calendar (Brown, 1988); (Claverol, Brown, & Chad, 2002) sau arbori Emde Boas (van Emde Boas, Kaas, & Zijlstra, 1976) (vezi de asemenea (Connollly, Marian, & Reilly, 2003), în care sunt comparate diverse liste de prioritate). Folosirea unei liste de prioritate eficiente este crucială pentru buna funcţionare a unui algoritm bazat pe eveniment. Devine şi mai crucial când interacţiunile sinaptice nu sunt instantanee.

Interacţiuni sinaptice non-instantanee. Pentru modele în care momentul emiterii unui impuls nu coincide neapărat cu momentul recepţionării unui impuls, simularea bazată pe eveniment este mai complexă. Descriem întâi algoritmul simplu fără întârzieri şi fără evenimente externe (vezi pseudocodul din Figura 2.13). O iteraţie constă în:

1. Identificarea neuronului care va emite următorul impuls
2. Actualizarea acestui neuron
3. Propagarea impulsului, adică actualizarea neuronilor către care se transmite impulsul

În general acest lucru se realizează păstrând o listă ordonată a timpilor viitoarelor impulsuri pentru toţi neuronii. Aceşti timpi ai impulsurilor sunt doar provizorii deoarece orice impuls din reţea poate modifica toţi timpii impulsurilor viitoare. Cu toate acestea, impulsul cu cel mai mic timp din listă este garantat ca fiind următorul. Astfel, următorul algoritm pentru o iteraţie garantează corectitudinea simulării (vezi Figura 2.13):

1. Se extrage impulsul cu cel mai mic timp din listă
2. Se actualizează starea neuronului corespunzător şi se recalculează timpul impulsului viitor
3. Se actualizează starea neuronilor ţintă
4. Se recalculează timpii impulsurilor viitoare pentru neuronii ţintă

```
for every neuron i
      compute timing of next spike                    Iniţializare
      insert event in priority queue
end

while queue not empty and t<duration
      extract event with lowest timing                Procesare impuls
      (event = timing t, neuron i)
      compute state of neuron i at time t
      reset membrane potential
      compute timing of next spike
      insert event in queue

      for every connection i -> j                     Comunicare impuls
            compute state of neuron j at time t
            change state with weight w(i,j)
            compute timing of next spike
            insert / change / supress event in queue
      end
end
```

Figura 2.13 Un algoritm simplu bazat pe eveniment cu interacţiuni sinaptice non-instantanee

Pentru simplificare, vom ignora întârzierile de transmisie pentru descrierea de mai sus. Includerea lor într-un algoritm bazat pe evenimente nu este la fel de simplă ca în cazul unui algoritm bazat pe tact, dar este o complicaţie minoră. Când un neuron produce un impuls, evenimentele sinaptice viitoare sunt stocate într-o altă listă de priorităţi, în care timpii evenimentelor nu pot fi modificaţi. Prima fază a algoritmului (extragerea impulsului cu cel mai mic timp) este înlocuită cu extragerea următorului eveniment, care poate fi un eveniment sinaptic sau emiterea unui impuls. Se pot folosi două liste separate sau una singură. Evenimentele externe pot fi tratate în acelaşi mod. Deşi întârzierile aduc complicaţii în codul algoritmilor bazaţi pe eveniment, acestea pot de fapt să simplifice administrarea listei de priorităţi pentru impulsurile emise. Într-adevăr, principala dificultate în simularea reţelelor cu interacţiuni sinaptice non-instantanee constă în faptul că impulsurile programate spre a fi emise pot fi anulate, amânate sau avansate de către impulsurile ce urmează a fi primite. Dacă întârzierile de transmisie sunt mai mari decât o valoare pozitivă $\tau_{\min}$, atunci toate impulsurile programate spre a fi emise în $[t, t+\tau_{\min}]$ (t fiind timpul prezent) sunt garantate. Astfel algoritmii pot exploata natura acestor întârzieri pentru a creşte viteza simulării (Lee & Farhat, 2001).

Complexitatea calculelor. Lăsând la o parte costul tratării evenimentelor externe (care este minor), putem împărţi costul computaţional al tratării unui impuls emis după cum urmează (presupunem că p este numărul mediu de sinapse per neuron):

- Se extrage evenimentul (în cazul interacţiunilor sinaptice non-instantanee)
- Se actualizează neuronul şi ţintele acestuia: $p+1$ actualizări
- Se inserează p evenimente sinaptice în listă (în caz că avem întârzieri)
- Se actualizează timpii de impuls pentru $p+1$ neuroni (în cazul interacţiunilor sinaptice non-instantanee)
- Se inserează sau se reprogramează $p+1$ evenimente în listă (impulsuri viitoare în cazul interacţiunilor sinaptice non-instantanee)

Deoarece avem $F \times N$ impulsuri pe secundă de timp biologic, numărul operaţiilor este aproximativ proporţional cu $F \times N \times p$. Costul computaţional total pe secundă de timp biologic poate fi scris în mod concis după cum urmează:

$$\begin{matrix} Actualizare + Impuls + Listă \\ \left(c_U + c_S + c_Q\right) \times F \times N \times p \end{matrix} \qquad \text{Ec. 43}$$

unde c_U este costul unei actualizări a variabilelor de stare, c_S este costul calculării timpului următorului impuls (interacţiuni sinaptice non-instantanee) şi c_Q este costul mediu al inserărilor în şi al extragerilor din lista (sau listele) de priorităţi. Astfel timpul simulării este liniar cu numărul sinapselor, ceea ce este optim. Cu toate acestea observăm ca operaţiile implicate sunt mai grele decât cele din faza propagării pentru algoritmii bazaţi pe tact (vezi secţiunea anterioară), astfel factorul multiplicativ este foarte probabil să fie mai mare. De asemenea am presupus că c_Q este $O(1)$, adică că operaţiile de intrare şi ieşire din coadă pot fi efectuate într-un timp mediu constant cu structura de date aleasă pentru lista de priorităţi. În cazul simplu al interacţiunilor sinaptice instantanee şi având întârzieri omogene, se poate folosi o simplă listă FIFO (First In, First Out), caz în care inserările şi extragerile sunt foarte rapide şi au durată de timp constantă. Pentru cazul general, există structuri de date pentru care operaţiile de intrare şi ieşire din coadă pot fi efectuate într-un timp mediu constant ($O(1)$), de ex. listele calendar (Brown, 1988); (Claverol, Brown, & Chad, 2002), dar sunt destul de complexe, si deci c_Q este o constantă mare. În implementări mai simple de liste de prioritate cum ar fi stivele binare, operaţiile de intrare şi ieşire din coadă se pot efectua în $O(\log m)$ timp, unde m este numărul de evenimente din coadă. În concluzie, se pare că administrarea cozii este componenta crucială pentru algoritmii bazaţi pe eveniment în general.

2.5.3. Prezentare generală a mediilor de simulare

2.5.3.1. NEURON

Domeniul de utilitate al NEURON. NEURON este un mediu de simulare folosit pentru crearea şi folosirea modelelor empirice a neuronilor biologici şi a circuitelor neuronale. Iniţial şi-a câştigat reputaţia de a fi bine adaptat pentru modele COBA de celule cu anatomie ramificată complexă, inclusiv potenţial extracelular în jurul membranei, şi proprietăţi biofizice cum ar fi tipuri de canale multiple, distribuţie neomogenă a canalelor, acumulare şi difuzie ionică, şi neurotransmitter-i secundari. La începutul anilor 90, NEURON era deja folosit în unele laboratoare pentru modele de reţele cu multe mii de celule, iar în ultima decadă a avut parte de multe îmbunătăţiri care au permis ca construcţia şi simularea reţelelor la scară mare să fie mai uşoară şi mai eficientă.

Până la această dată au apărut mai mult de 600 articole şi cărţi care descriu modele NEURON, de la bucăţi de membrană până la reţele la scară mare cu zeci de mii de celule pulsative artificiale sau COBA[1]. În 2005 au fost publicate peste 50 de articole cu subiecte cum ar fi mecanismele care stau la baza transmisiei şi plasticităţii sinaptice (Banitt, Martin, & Segev, 2005), modularea integrării sinaptice prin intermediul curenţilor activi sub pragul de activare (Prescott & De Koninck, 2005), excitabilitate dendritică (Day, Carr, Ulrich, Ilijic, Tkatch, & Surmeier, 2005), rolul fantelor sinaptice în reţele (Migliore, Hines, & Shepherd, 2005), efectele plasticităţii sinaptice în dezvoltarea şi modul de operare al reţelelor biologice (Saghatelyan, et al., 2005), consecinţele zgomotului de canal şi sinaptic asupra procesării informaţiei în neuroni şi reţele (Badoual, Rudolph, Piwkowska, Destexhe, & Bal, 2005), mecanisme celulare şi ale reţelei pentru codificare şi recunoaştere temporală (Kanold & Manis, 2005), stări şi oscilaţii ale reţelei (Wolf, et al., 2005), efectele îmbătrânirii asupra funcţiilor neuronale (Markaki, Orphanoudakis, & Poirazi, 2005), imprimarea corticală (Moffitt & McIntyre, 2005), stimularea profundă a creierului (Grill, et al., 2005), şi epilepsia rezultată în urma mutaţiilor de canal (Vitko, Chen, Arias, Shen, Wu, & Perez-Reyes, 2005), şi traumatisme cerebrale (Houweling, Bazhenov, Timofeev, Steriade, & Sejnowski, 2005).

[1] http://www.neuron.yale.edu/neuron/bib/usednrn.html

Prin ce diferă NEURON de alte neuro-simulatoare. Principalul motiv pentru care se folosesc simulatoare specializate în locul celor de uz general constă în perspectiva unui control conceptual îmbunătăţit, şi posibilitatea de a exploata structura ecuaţiilor care definesc modelul pentru a obţine robusteţe, acurateţe şi eficienţă computaţională. Câteva diferenţe cheie între NEURON şi alte neuro-simulatoare constă în modul în care se încearcă atingerea acestor scopuri.

Control conceptual. Ciclul formulării ipotezei, testării, şi revizuirii, care stă la baza tuturor cercetărilor ştiinţifice, presupune că se pot deduce consecinţele unei ipoteze. Principalul motiv pentru care se foloseşte modelarea computaţională este utilitatea acesteia în abordarea acelor ipoteze a căror consecinţe nu pot fi nu pot fi determinate intuitiv sau analitic. Valoarea oricărui model ca metodă de evaluare a unei anumite ipoteze depinde în mod critic de potrivirea cât mai exactă dintre model şi ipoteză. Fără o asemenea potrivire rezultatele simulării nu pot fi un bun test al ipotezei respective. Din punctul de vedere al utilizatorului, primul obstacol din calea modelării computaţionale constă în dificultatea cu care se obţine controlul conceptual, adică să ne asigurăm ca modelul computaţional reflectă cu fidelitate ipoteza noastră.

NEURON are mai multe proprietăţi care facilitează controlul conceptual, şi continuă să obţină noi astfel de proprietăţi în timp ce evoluează pentru a satisface nevoile în continuă schimbare ale cercetătorilor din domeniu. Multe din aceste proprietăţi intră în categoria generală a specificaţiilor de „sintaxă nativă" a proprietăţilor modelului: astfel atributele cheie ale neuronilor biologici şi ale reţelelor au echivalente directe în NEURON. De exemplu utilizatorii NEURON pot specifica proprietăţile de obturare ale canalelor ionice prin scheme kinetice sau familii de ecuaţii diferenţiale de tip HH. Un alt exemplu ar fi că modelele pot include circuite electronice construite cu LinearCircuitBuilder, un instrument GUI a cărui paletă include rezistenţe, condensatori, surse de tensiune şi curent, şi amplificatori operaţionali. Cel mai elocvent exemplu de aplicare a conceptului de sintaxă nativă a proprietăţilor NEURON ar fi modul în care tratează proprietăţile de înlănţuire ale neuronilor, care este diferit faţă de orice alt neuro-simulator. Utilizatorii NEURON nu trebuie să trateze în mod direct compartimentele. Astfel, celulele sunt reprezentate prin neurite separate, numite secţiuni, care pot fi asamblate în arhitecturi de tip arbore (topologia unei celule model). Fiecare secţiune are propriile proprietăţi anatomice şi biofizice, plus un parametru de discretizare care specifică rezoluţia locală a grilei spaţiale. Proprietăţile unei secţiuni pot varia încontinuu odată cu lungimea sa, şi variabilele neomogene spaţial sunt accesate în termeni de distanţă normalizată de-a lungul fiecărei secţiuni (Hines & Carnevale, 1997) (Capitolul 5 din (Carnevale & Hines, 2006)). Odată ce utilizatorul a specificat topologia celulei, geometria, proprietăţile biofizice, şi parametrul de discretizare pentru fiecare secţiune, NEURON construieşte automat structurile de date interne care corespund unei familii de ODE-uri pentru ecuaţia discretizată de înlănţuire a modelului.

2.5.3.2. GENESIS

Filozofia de design şi capabilităţile GENESIS. GENESIS (the General Neural Simulation System) a primit acest nume deoarece a fost proiectat de la bun început ca un sistem general de simulare extensibil, folosit la modelarea realistă a sistemelor neuronale şi biologice (Bower & Beeman, 1998). Cu GENESIS s-au executat simulări de la componente subcelulare şi reacţii biochimice (Bhalla, 2004) până la modele complexe de neuroni singulari (De Schutter & Bower, 1994), simulări de reţele mari (Nenadic, Ghosh, & Ulinski, 2003), şi modele la nivel de sistem (Stricanne & Bower, 1998). Aici, „modelele realiste" sunt definite ca fiind acele modele care sunt bazate pe organizarea anatomică şi fiziologică cunoscută a neuronilor, circuitelor şi reţelelor (Bower, 1995). De exemplu, modelele de celulă realiste includ în mod tipic morfologie dendritică şi o mare varietate de conductanţe ionice, iar modelele de reţea realiste încearcă să copieze tiparele cunoscute de proiecţie axonală.

Parallel GENESIS (PGENESIS) este o extensie a software-ului GENESIS care poate rula pe aproape orice cluster paralel, SMP, super-computer, sau reţea de calculatoare care suportă MPI şi / sau PVM, şi pe care poate rula şi software-ul GENESIS serial. Este folosit de obicei la simularea reţelelor mari ce implică zeci de mii de modele de celulă realiste (Hereld, Stevens, Teller, & van Drongelen, 2005).

Utilizatorii GENESIS au la dispoziţie un proces bine documentat prin care îi pot extinde capabilităţile prin adăugarea de noi tipuri de obiecte (clase) GENESIS definite de utilizator, sau comenzi de limbaj de scriptare, fără să fie nevoie să înţeleagă sau să modifice codul de simulare GENESIS.

GENESIS vine deja echipat cu mecanismele necesare pentru a crea uşor modele de reţele la scară mare realizate din modele de neuroni singulari care au fost deja implementate cu GENESIS.
Deşi utilizatorii au adăugat, de exemplu, modelul simplificat de neuron pulsativ (Izhikevich E. M., 2003) (acum încorporat în GENESIS), şi ar putea de asemenea adăuga modele IF sau alte forme abstracte de modele de neuroni, aceste forme de neuroni nu sunt suficient de realiste pentru a satisface interesul majorităţii celor ce modelează cu GENESIS. Din acest motiv, GENESIS nu include în mod normal modele IF de neuroni, şi nu se oferă implementări GENESIS pentru modele IF. Neuronii tipici GENESIS sunt modele multi-compartimentale cu o varietate de conductanţe de tip HH dependente de voltaj şi / sau calciu.

Modelarea cu GENESIS. GENESIS este un sistem de simulare orientat-obiect, în care o simulare se construieşte din blocuri de construcţie de bază (elemente GENESIS). Aceste elemente comunică între ele prin transmiterea de mesaje şi fiecare conţine informaţia propriilor variabile (câmpuri) şi metode (acţiuni) folosite la efectuarea calculelor şi a altor sarcini din timpul simulării. Elementele GENESIS sunt create ca instanţieri ale unui tip particular de obiecte precompilate care se comportă ca un şablon. Neuronii model sunt construiţi din aceste componente de bază, cum ar fi compartimente neuronale şi canale ionice cu conductanţă variabilă, legate între ele prin mesaje. Neuronii pot fi legaţi intre ei prin conexiuni sinaptice pentru a forma circuite neuronale şi reţele. Această abordare orientată-obiect asigură generalitatea şi flexibilitatea sistemului, deoarece permite utilizatorilor să facă schimb sau să refolosească modelele sau componentele modelelor. Mulţi utilizatori GENESIS îşi bazează scripturile pe exemplele care sunt puse la dispoziţie cu GENESIS sau în pachetul GENESIS Neural Modeling Tutorials (Beeman, 2005).

GENESIS foloseşte un interpretor şi un limbaj de simulare de nivel înalt pentru a construi neuronii şi reţelele acestora. Folosirea unui astfel de interpretor cu tipuri de obiecte pre-compilate, în locul unui pas în plus pentru a compila scripturile în cod maşină binar, are avantajul de a permite utilizatorului să interacţioneze cu şi să modifice simularea în timpul derulării acesteia, fără să sacrifice viteza simulării. Comenzile pot fi date fie interactiv prin linia de comandă, fie prin folosirea scripturilor de simulare, sau prin intermediul interfeţei grafice. Cele 268 de comenzi ale limbajului de scriptare şi cele 125 de tipuri de obiecte puse la dispoziţie cu GENESIS sunt suficient de puternice astfel încât este nevoie de doar câteva rânduri de script pentru a specifica o simulare sofisticată. De exemplu, „cititorul de celule” al GENESIS permite construirea modelelor neuronale complexe prin citirea specificaţiilor dintr-un fişier de date.

GENESIS pune la dispoziţie o varietate de mecanisme pentru a modela difuzia calciului şi conductanţe dependente de calciu, precum şi plasticitatea sinaptică. Are de asemenea un număr de „obiecte dispozitiv” care pot fi folosite ca interfaţă în simulare pentru a asigura simulării diverse tipuri de intrări (generatoare de puls şi impuls, circuite de limitare a potenţialului, etc.) sau măsurători (histograme de peristimulus şi interval inter-impuls, măsurători de frecvenţă a impulsurilor, histograme de auto-corelare şi inter-corelare, etc.). Se oferă de asemenea tipuri de obiecte pentru modelarea căilor biochimice (Bhalla & Iyengar, 1999). O listă şi descriere a tipurilor de obiecte GENESIS, cu legături la documentaţia completă, poate fi găsită în secţiunea „Obiecte” a documentului GENESIS Reference Manual, care poate fi descărcat sau vizionat pe site-ul web al GENESIS.

2.5.3.3. NEST

Iniţiativa NEST. Problema simulării reţelelor neuronale de mărime şi complexitate realiste a fost multă vreme subestimată. Acest lucru este reflectat de numărul limitat de articole având ca subiect structuri de date şi algoritmi corespunzători apărute în revistele cu impact mare. Lipsa conştientizării de către cercetători şi agenţiile de finanţare a nevoii de progres în tehnologia simulării şi durabilitatea investiţiilor poate fi cauzată parţial de faptul că un simulator corect din punct de vedere matematic pentru un anume model de reţea neuronală poate fi implementat de un individ în câteva zile. Însă, datorită acestei rutine s-a ajuns la un ciclu de cod program nescalabil fiind continuu rescris într-un mod greu de întreţinut de către novici, obţinându-se un progres minim al fundamentelor teoretice.

Datorită disponibilităţii în creştere a resurselor computaţionale, studiile din domeniul simulării devin din ce în ce mai ambiţioase şi mai populare. Într-adevăr, răspunsurile la multe întrebări din domeniul neuro-ştiinţei sunt în prezent accesibile doar prin intermediul simulării. O consecinţă negativă

a acestei tendinţe este că reproducerea şi verificarea rezultatelor acestor studii devine din ce în ce mai grea. Instrumentele de simulare ad-hoc ale trecutului nu ne pot oferi un grad de cunoaştere potrivit. În locul acestora avem nevoie de simulatoare de reţele neuronale atent dezvoltate, validate, documentate şi expresive având o bază mare de utilizatori. Mai mult decât atât, actualul progres spre modele mai realiste impune necesitatea unor simulări corespunzătoare mai eficiente. Acest lucru este valabil mai ales pentru noul domeniu de studiu asupra modelelor de reţele la scară mare care încorporează plasticitatea. Aceste cercetări nu sunt deloc fiabile fără simulatoare paralele cu proprietăţi scalabile excelente, ceea ce soluţiile ad-hoc nu pot oferi. De asemenea, pentru a fi considerate folositoare de către un număr cât mai mare de cercetători pe o perioadă lungă de timp, aceste simulatoare trebuie să fie uşor de întreţinut şi de extins.

Pe baza acestor consideraţii, iniţiativa NEST a fost fondată ca un proiect de colaborare pe termen lung care să încurajeze dezvoltarea tehnologiei pentru simulatoarele sistemelor neuronale (Diesmann & Gewaltig, 2002). Instrumentul de simulare NEST este implementarea de referinţă a acestei iniţiative. Programul este oferit comunităţii ştiinţifice sub o licenţă open-source prin site-ul web al iniţiativei NEST[2]. Licenţa cere cercetătorilor să facă trimitere către iniţiativa NEST în munca lor derivată de la codul original şi, mai important, în rezultatele ştiinţifice obţinute cu ajutorul acestui program. Site-ul web oferă de asemenea referinţe la material relevant pentru simularea reţelelor neuronale în general, şi vrea să devină o sursă ştiinţifică importantă de informaţii pentru simulări de reţele. Ajutorul este oferit prin intermediul site-ului NEST şi o listă de discuţii prin email. În prezent NEST este folosit în predare la şcoli de vară internaţionale şi în cursurile regulate ale Universităţii din Freiburg.

2.5.3.4. SPLIT

Simulatoare paralele. Dezvoltarea simulatoarelor paralele în ştiinţa neuronală computaţională a fost relativ înceată. În prezent există puţine simulatoare paralele disponibile publicului larg, dar sunt departe de a fi la fel de generale, flexibile şi documentate ca şi simulatoarele seriale folosite în general cum ar fi NEURON (Hines & Carnevale, 1997) şi GENESIS (Bower & Beeman, 1998). Pentru GENESIS există PGENESIS, iar pentru NEURON a început dezvoltarea unei versiuni paralele. De asemenea există simulatoare ca NCS[3] (Ermentrout, 2004), NEST (Morrison, Mehring, Geisel, Aertsen, & Diesmann, 2005), şi simulatorul de paralelizare SPLIT (Hammarlund & Ekeberg, 1998). Însă acestea sunt încă în stadiu experimental sau de dezvoltare din multe puncte de vedere.

2.5.3.5. Mvaspike

Modelarea cu evenimente. S-a discutat de multe ori că potenţialele de acţiune aşa cum sunt produse de multe tipuri de neuroni pot fi considerate ca fiind *evenimente*: ele constă în impulsuri stereotipice care apar suprapuse peste dinamica voltajului intern al neuronilor. Ca rezultat, multe modele de neuroni oferă moduri de definire a timpilor evenimentelor asociaţi cu fiecare potenţial de acţiune emis, deseori prin definirea unui prag de activare[4]. Instrumentele de simulare neuronală au profitat de acest fapt de multă vreme, prin folosirea unui *algoritm bazat pe eveniment* (vezi secţiunea 2). Într-adevăr când se vorbeşte despre *evenimente* în contextul simulării reţelelor neuronale, ne gândim în primul rând la algoritmi *bazaţi pe eveniment*, şi avem impresia că folosirea evenimentelor upstream, în timpul etapei de modelare, este deseori subestimată.

Mvaspike a fost proiectat ca un cadru de modelare şi simulare bazată pe eveniment. Este bazat pe o abordare a modelării teoriei mulţimilor bine stabilită – specificaţie de sistem bazată pe evenimente discrete (discrete event system specification – DEVS) (Zeigler & Vahie, 1993); (Zeigler, Praehofer, & Kim, 2000). Modelele ţintă sunt sisteme bazate pe evenimente discrete: dinamica lor poate fi descrisă

[2] http://www.nest-initiative.org
[3] http://brain.cse.unr.edu/ncsdocs
[4] Pragul de activare aici trebuie înţeles într-un sens foarte larg, de la un simplu prag de detectare impuls într-un model continuu (ex. HH) până la un prag activ care este folosit în expresia matematică a dinamicii (modelul IF).

prin schimbările variabilelor de stare la momente alese arbitrar în timp[5]. Mvaspike realizează apropierea dintre expresia mai familiară a dinamicii continue, folosită în general în neuroştiinţă, şi folosirea de modele centrate pe eveniment în simulator. Lucru convenabil pentru modele simple care reprezintă majoritatea modelelor în Mvaspike (mai ales modele IF sau de fază, şi SRM-uri). Watts (Watts, 1994) a notat deja că multe proprietăţi neuronale pot fi reprezentate în mod explicit şi uşor în sisteme bazate pe evenimente discrete. Dacă ne gândim la *perioade* refractare absolute, *timpi* de creştere ai PSP-urilor, *întârzieri* de propagare axonală, acestea sunt noţiuni legate în mod direct de intervale de timp (şi în concluzie, de evenimente) care sunt folositoare pentru a descrie multe aspecte ale dinamicii neuronale. Astfel fiind în mod evident destul de departe de modelele bine stabilite bazate pe conductanţă, mai corecte din punct de vedere electro-fiziologic, un alt scop al Mvaspike este să ia în considerare cât mai mult posibil aceste modele mai complexe, prin susţinerea explicită a evenimentelor discrete în timp, şi, posibil, discretizarea spaţiului de stare pentru integrarea dinamicilor continue sau hibride.

Formalismul DEVS face posibilă modelarea unor sisteme ierarhice sau modulare mari (de ex. reţele cu populaţii cuplate de neuroni, coloane corticale, etc.), printr-un sistem de cuplare şi compoziţie bine definit. Asta ajută la modelarea reţelelor mari şi complexe, şi favorizează refolosirea codului sursă, crearea de prototipuri, şi folosirea nivelelor diferite de modelare. Au fost implementate instrumente adiţionale în Mvaspike pentru a lua în considerare de exemplu întârzierile de propagare sinaptică sau axonală, precum şi arhitectura reţelelor structurate sau conectate aleator într-un mod eficient pentru a descrie conectivitatea (Rochel & Martinez, 2003).

2.6. Implementări software proprii, validarea modelelor dezvoltate

2.6.1. Idei de bază

Programul de simulare dezvoltat are la bază modelul de reacţie la impuls prezentat anterior, aplicând codarea cu un impuls pe neuron.

Fiind vorba de o simulare într-un mediu digital (realizată pe un calculator personal), a fost necesară dezvoltarea unui model digital adecvat, care să aproximeze cu o precizie suficient de bună modelul original şi în acelaşi timp, să exploateze la maxim capacitatea sistemului de calcul utilizat. În consecinţă, trebuie găsită o soluţie pentru atingerea preciziei dorite a parametrilor şi variabilelor utilizate, precum şi pentru construirea unei simulări bazate pe divizarea timpului în unităţi de timp corespunzător alese. Această divizare temporală necesită o acordare sensibilă, pentru a asigura precizia temporizărilor impulsurilor de activare.

2.6.2. Algoritmul simulării

Scopul principal al procedurilor ce compun mediul de simulare este calcularea acelor parametri ai reţelei care sunt necesari pentru a genera fiecare IA într-un moment dat al procesării continue a reţelei. Impulsurile de activare sunt calculate cu ajutorul potenţialelor de membrană ale neuronilor, respectiv utilizând funcţiile de prag aferente. Este necesar, deci, să prelucrăm în cursul simulării impulsurile pre- şi post-sinaptice, calculând cu ajutorul lor potenţiale de membrană valide pentru fiecare neuron, determinând care dintre aceştia va emite un IA în pasul temporal următor. Totodată trebuiesc procesate ponderile şi decalările temporale valide pentru sinapse. Toţi aceşti parametri sunt stocaţi într-o structură definită în memoria calculatorului, structură ce defineşte starea reţelei neuronale.

Datorită complexităţii modelelor de NP, algoritmii de calcul tradiţionali (ai reţelelor bazate pe modelul perceptronului) care utilizează matrici şi vectori de date nu sunt utilizabili. În acest caz neuronii (somele acestora) şi sinapsele dintre ei necesită calcule separate. Deoarece într-un calculator personal nu avem un număr de unităţi de calcul separate necesare pentru a efectua aceste calcule în paralel, suntem nevoiţi să recurgem – cel puţin parţial – la prelucrare secvenţială de date.

Este necesar, însă, ca aceste prelucrări secvenţiale, să garanteze rezultate temporal consecvente. Pentru a realiza acest lucru, am utilizat procedura de divizare a fiecărui pas temporal în două faze: în prima fază se calculează IA al tuturor neuronilor din populaţia reţelei, IA ce vor fi utilizate în faza a doua pentru a

[5] Spre deosebire de sistemele discrete în timp, în care schimbările de stare se produc în mod periodic, şi sistemele continue unde schimbările de stare se produc încontinuu.

efectua operaţiile de transmitere a acestora conform structurii reţelei.

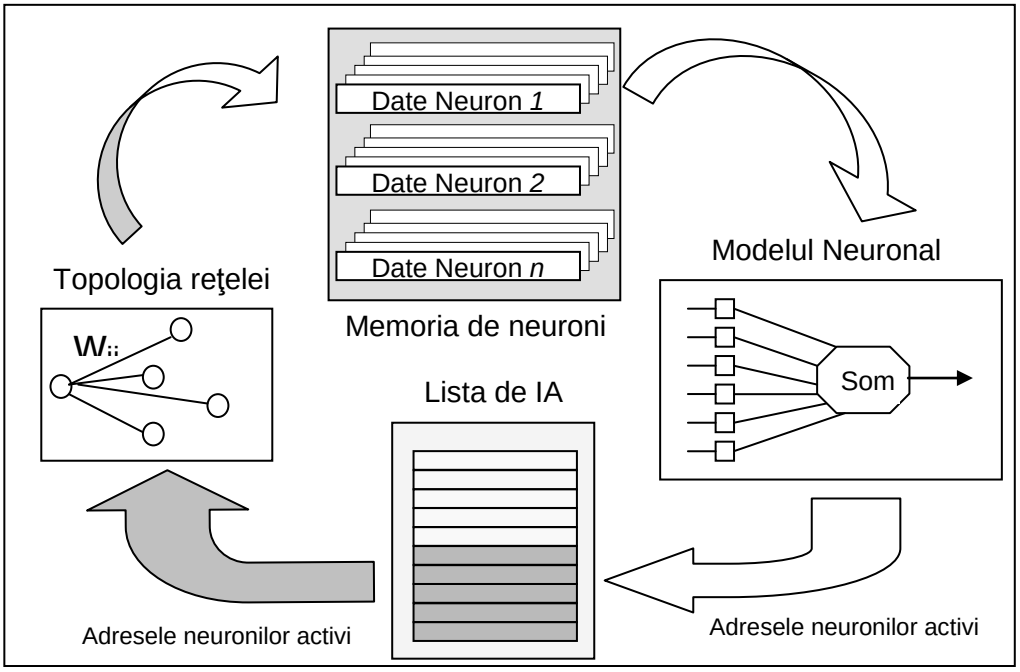


Figura 2.14 Schema algoritmului de simulare

Această divizare duce la realizarea unui algoritm trivial Figura 2.14:

- În primul pas calculez potenţialele de membrană şi de prag pe baza valorilor citite din memoria de neuroni. Dacă o valoare de potenţial de membrană se situează peste valoarea de prag aferentă, atunci se produce un IA care se salvează în lista de IA, împreună cu codul de identificare al neuronului emitent. Toate acestea se efectuează urmând cu stricteţe cerinţele modelului neuronal utilizat.
- În pasul al doilea, valorile w_{ij} sunt utilizate pentru a pondera IA citite din lista în care au fost salvate, urmând a fi transmise spre neuronii post-sinaptici, ţinând cont de topologia reţelei construite. În acest moment se poate aplica algoritmul de învăţare dorit, modificând ponderile şi întârzierile sinapselor.

Acum starea reţelei este complet determinată, poate începe următorul pas temporal.

Viteza de calcul a parametrilor neuronali şi sinaptici se poate ridica – se poate ajunge chiar şi la calcul în timp real – dacă utilizăm prelucrare paralelă sau aplicăm tehnici pipe-line (Hopfield, 1995). De exemplu, într-un sistem multiprocesor, rolul unei unităţi centrale de prelucrare (CPU) ar fi prelucrarea parametrilor neuronali, iar un al doilea procesor ar rula procedurile de calcul a parametrilor sinaptici. Dacă nu avem la îndemână un astfel de calculator, atunci putem utiliza o unitate specială de prelucrare, foarte rapidă, care utilizează tehnica pipe-line (alimentare continuă cu date) şi reuşeşte să efectueze aceste calcule în doar câteva cicluri de tact.

2.6.3. Încorporarea algoritmilor de învăţare în procesul de simulare

Fiind vorba de reţele neuronale artificiale, ne referim la algoritmii de învăţare ca la mecanisme de modificare a intensităţii conexiunilor dintre neuroni în vederea optimizării funcţionării reţelei. În cazul de faţă însă, lucrul cu reţele neuronale bazate pe NP ne determină să ne concentrăm asupra procedeelor de învăţare observate în reţelele neuronale naturale. Cercetări neurobiologice (Trappenberg, 2002) au scos la iveală mai multe astfel de mecanisme, care modifică eficacitatea sinapselor de la o perioadă de timp foarte scurtă (de ordinul milisecundelor) până la modificări care determină schimbări structurale durabile.

Proiectând un neuro-calculator bazat pe codare prin impulsuri, avem nevoie de un model de acordare a sinapselor care realizează – cel puţin parţial – fenomenele naturale înglobând în acelaşi timp caracteristici considerate importante de către autor. Scopul principal este de a găsi o cale de mijloc, care va conduce la un sistem aplicabil în diferite probleme de control dar care să fie uşor implementabil în

hardware, cu ajutorul circuitelor reconfigurabile FPGA. Aceste consideraţii sunt foarte importante, deoarece în cazul unor probleme complexe de control automat, dimensiunile reţelei neuronale pot creşte semnificativ, rezultând o sarcină de calcul dificil de rezolvat.

În această ordine de idei am recurs la utilizarea *algoritmului de învăţare de tip Hebb.* In reţele neuronale neuromorfe, algoritmul Hebb se poate considera ca o colecţie de reguli care ajustează eficacitatea sinapselor în funcţie de activitatea neuronilor pre- şi post-sinaptici. La modul general, învăţarea de tip Hebb se poate exprima cu următoarele patru componente:

1. O conexiune neuronală este întărită (creşterea eficienţei sinaptice), dacă neuronii pre- şi post-sinaptici sunt simultan activi (emit IA).
2. O conexiune neuronală este slăbită, (scăderea eficienţei sinaptice), dacă numai neuronul pre-sinaptic este activ.
3. O conexiune neuronală este slăbită, (scăderea eficienţei sinaptice), dacă numai neuronul post-sinaptic este activ.
4. O conexiune neuronală rămâne neschimbată, dacă nici unul dintre neuronii între care se află sinapsa nu este activ.

Modificările eficienţelor sinaptice se realizează exclusiv pe baza parametrilor locali, parametrii globali, ca activitatea medie a reţelei, nu influenţează acest proces. Nu există funcţionare *corectă* sau *incorectă* a reţelei, deci nu există un *expert,* care să judece acest lucru. În consecinţă, algoritmul Hebb implementează o învăţare nesupervizată.

Ca rezultat al cercetărilor din acest domeniu au apărut mai multe variante metodei de învăţare Hebb.

$$\Delta w_{ij}(t) = \left(x_i(t) - \overline{x_i}\right)\left(x_j(t) - \overline{x_j}\right) \qquad \textit{Ec. 44}$$

Acordarea sinapselor este dată de corelaţia dintre activităţile de ieşire ale neuronilor x_i şi x_j, expresie ce se poate interpreta şi ca *factorul mediei de activare* a acestor neuroni.

Având ca scop realizarea RNP cu ajutorul echipamentelor hardware, o soluţie mai potrivită în ceea ce priveşte alegerea metodei de învăţare ar fi cea bazată pe observaţii neurofizice, şi anume *excitarea pe termen lung* (ETL) respectiv *inhibare pe termen lung* (ITL). Ca scurtă descriere a metodei ETL-ITL se poate spune că eficienţa unei sinapse active creşte, dacă potenţialul membranei neuronului post-sinaptic se depolarizează, respectiv scade în caz contrar. Primul caz este cel de ETL iar cel din urmă ITL. Ambele cazuri se pot compara cu punctele 1. respectiv 2. din descrierea metodei Hebb. Punctul 3. din descrierea metodei Hebb lipseşte însă la analiza metodei ETL-ITL, deoarece aici luăm în considerare doar sinapsele active.

Pentru a putea dezvolta un mediu de simulare flexibil, adăugarea acestui punct 3. ar fi totuşi, necesară în pofida faptului, că acest lucru nu se conformează cu rezultatele experimentale din literatură (Földiák, 1990), (Natschläger & Ruf, 1998), (Bohte, Kok, & La Poutré, 2002). Autorii acestor lucrări au arătat, că utilizarea algoritmului Hebb în reţele neuronale cu codare al factorului mediu de activare, poate duce la apariţia unor efecte neuromorfe de genul auto-organizării. Este demonstrat, de asemenea, că în mediul natural, sinapsele care nu influenţează pe termen lung activitatea post-sinaptică vor înceta să existe.

Studiind în amănunţime rezultatele experimentale din (Natschläger & Ruf, 1998), (Bohte, Kok, & La Poutré, 2002)în ceea ce priveşte ETL-ITL, putem trage două concluzii importante:

1. O anumită valoare de prag a depolarizării post-sinaptice este aceea care determină activarea ETL sau a ITL.
2. "Retro-propagarea" IA post-sinaptice creează un comportament de învăţare, în funcţie de decalajul temporal dintre IA pre- şi post-sinaptice.

Pe baza acestor rezultate şi considerente am dezvoltat o strategie de învăţare, care va fi prezentată în cele ce urmează.

2.6.4. Învăţare bazată pe valori de prag

Modificarea eficacităţii sinapselor este funcţia măsurii de depolarizare a potenţialului membranei neuronului post-sinaptic. (Între valorile de prag ϑ_{ITL} şi ϑ_{ETL} eficacitatea sinaptică scade.) Dacă potenţialul membranei neuronului post-sinaptic se află în acest interval, atunci are loc ITL [14]. Dacă acest potenţial depăşeşte ϑ_{ETL}, atunci se va activa ETL. Comparând acest model cu modelul de învăţare Hebb, putem spune, că aici activitatea pre-sinaptică este dată de emiterea unui IA pre-sinaptic, iar activitatea post-sinaptică este dată de valoarea potenţialului de membrană (Figura 2.15).

Deoarece această metodă descrie numai felul modificării sinapselor active, un impuls pre-sinaptic se poate percepe ca un semnal de start al procesului de învăţare. Acest fapt îndeplineşte cerinţa unei simulări eficiente, deoarece frecvenţa IA nu este prea mare. Deci, într-un anumit pas al simulării nu este necesar a **calcula**, numai parametrii unei fracţiuni din populaţia de neuroni ai reţelei.

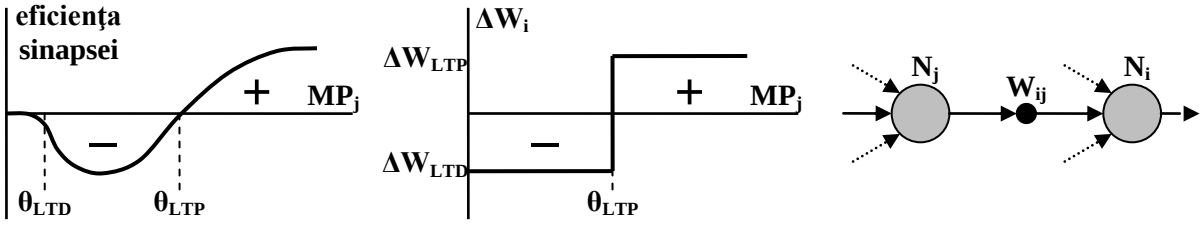


Figura 2.15 *Regula de învăţare bazată pe valori de prag*

(Δw_{ij} variaţia ponderii sinaptice, MP_i potenţialul membranei postsinaptice)

A –modelul biologic, B – modelul adaptat

Având în vedere necesitatea implementării hardware, este posibilă o simplificare adiţională, setând valoarea de prag ϑ_{ITL} la zero. Procesul de învăţare rezultant se poate exprima în pseudo-cod, cu algoritmul următor:

if neuronul pre-sinaptic *j* emite IA
  if $MP_i > \vartheta_{ETL}$
    incrementează w_{ij}
  else
    decrementează w_{ij},

unde w_{ij} este ponderea conexiunii dintre neuronii *j* şi *i*, iar MP_i este potenţialul membranei post-sinaptice.

Acest algoritm, însă nu descrie numai punctele 1. şi 2. din regula de acordare sinaptică Hebb. Punctul 3. care se referă la cazul în care numai neuronul post-sinaptic este activ lipseşte, pentru moment. Pentru a completa algoritmul, va trebui să adăugăm următoarele rânduri:

if $MP_i > \vartheta_{ETL}$
  if neuronul pre-sinaptic *j* **nu** emite IA
    decrementează w_{ij}.

Deci, acest proces de învăţare se activează numai în cazul în care potenţialul de membrană post-sinaptic depăşeşte ϑ_{ETL}, eveniment care este mult mai frecvent, decât apariţia unui IA.

2.7. Rezultate experimentale a simulărilor software

Datorită complexităţii problemei, am acordat o atenţie deosebită alegerii mediului de dezvoltare a programului de simulare. După cum s-a văzut în capitolele anterioare, în reţelele neuronale naturale precum şi în cele artificiale neuromorfe, componenta temporală joacă un rol extrem de important. Era nevoie, deci de o soluţie care să asigure temporizări precise şi previzibile concomitent cu un mediu de programare de nivel înalt, corespunzător sarcinii de rezolvat. În această ordine de idei, şi după câteva experimente efectuate, am renunţat la calea utilizării sistemului de operare Windows şi a unui mediu interpretor (de ex. Matlab) sau compilator (de ex. Visual C++). Am rămas la limbajul de programare C++, acesta oferind mijloacele cele mai dinamice, însă am trecut la utilizarea sistemului de operare RedHat Linux (respectiv Fedora Core 3). Pe lângă suportul substanţial mai bun al temporizărilor în comparaţie cu Windows, în Linux am beneficiat şi de alte avantaje precum reprezentarea grafică 3D mai uşoară a reţelelor neuronale realizate (pachete grafice Povray, Gnuplot) sau salvări de structură a reţelelor în format modern XML.

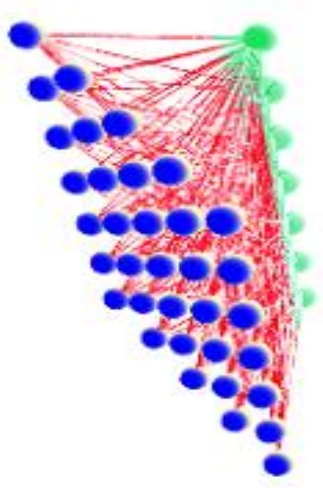


Figura 2.16 O reţea neuronală realizată în mediul de simulare dezvoltat Aplicaţia: recunoaştere de caractere dintr-o matrice de 7x5 puncte (albastru – neuroni de intrare, verde – neuroni de ieşire, roşu – sinapse)

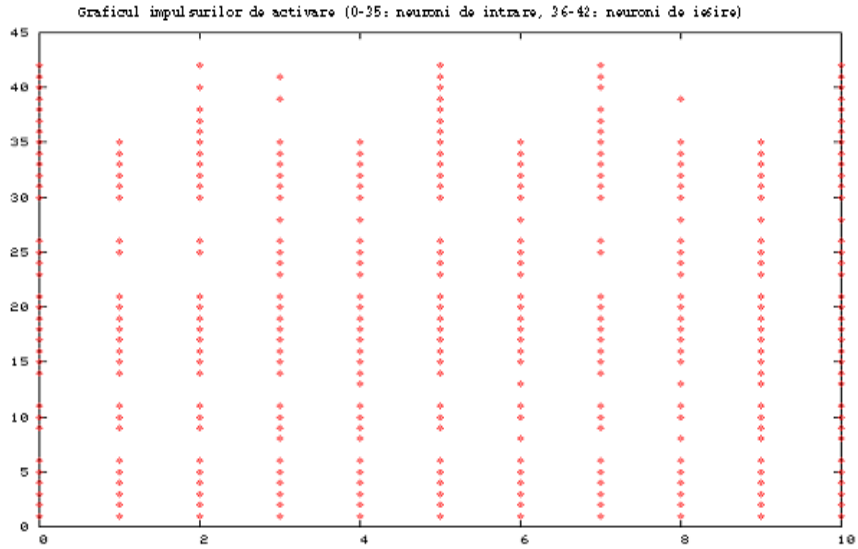


Figura 2.18 IA în timpul funcţionării reţelei neuronale prezentate în Figura 2.16.

Dezvoltarea de software a urmat regulile impuse de *modelul de reacţie la impuls*, descris la subpunctul 2.4.2, respectiv învăţarea bazată pe valori de prag. Trecând în revistă pe scurt componentele sistemului de simulare dezvoltat ele ar fi: modelul neuronal implementat, modelul sinaptic implementat, modulele de construcţie a straturilor respectiv cele ce realizează conexiunile dintre straturi. Orice configuraţie reprezentată în 2D sau 3D este realizabilă, construind reţeaua neuron cu neuron sau strat cu strat. Am implementat şi posibilitatea de realizare de reacţii, astfel putându-se construi reţele compuse din câteva straturi de NP, al căror ieşiri sunt conectate la intrările celui de-al doilea strat de neuroni. Primul strat este format din neuroni speciali, fără sinapse care posedă numai câte o singură intrare. Putem ajunge astfel la structuri care vor emite impulsuri de ieşire la intervale regulate, ele fiind adecvate pentru utilizare ca generator de tact sau pentru sincronizarea altor grupuri de neuroni.

Programul utilizează, deci, trei tipuri de neuroni, şi anume:

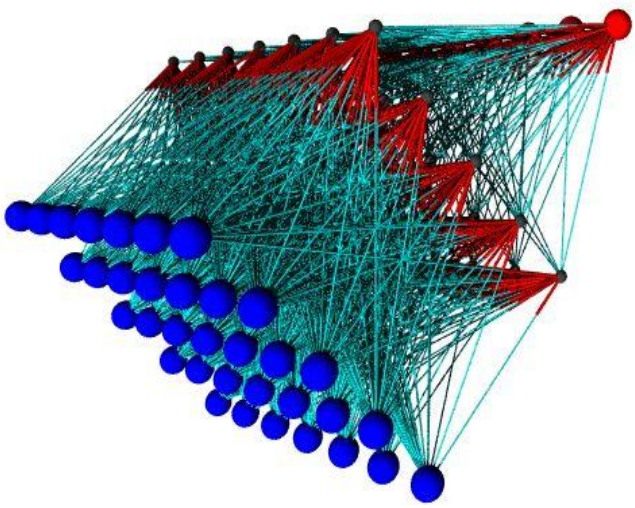


Figura 2.17 Aceeaşi problemă, rezolvată cu o reţea cu un strat ascuns

- neuroni de intrare (pot ocupa loc numai în stratul de intrare)
- neuroni ascunşi (neuronii straturilor interioare)
- neuroni de ieşire (neuronii stratului de ieşire)

Semnalele de intrare si de ieşire ale reţelelor de NP construite sunt considerate evenimente ale căror temporizări se pot regla, ele fiind de fapt valori binare - 1 si 0 logic.

În Figura 2.17 este prezentată o RNP cu un strat ascuns, care este capabilă să categorizeze caractere prezentate chiar si sub formă deteriorată.

Cei şapte neuroni de ieşire corespund segmentelor unui afişor cu 7 segmente. Ieşirile date de reţea la intrările zgomotoase date la intrare se pot urmări în Figura 4.6, luând în considerare, că pe axa X s-a reprezentat numărul pasului temporal în care s-a efectuat măsurătoarea respectiv, pe axa Y identificatoarele neuronilor din reţea. Identificatorii neuronilor sunt: (ID's): 0_34 strat de intrare, 34_69 strat ascuns, 70_76 strat de ieşire.

În fiecare pas temporal la intrarea reţelei este prezentat un nou vector de intrare, după care sunt efectuate calculele sinaptice aferente, care au ca rezultat acordarea ponderilor sinapselor reţelei.

Convergenţa procesului de învăţare este confirmată de faptul că, după un număr suficient de mare de paşi temporali efectuaţi, pentru un model anume de intrare se va activa numai un anumit grup de neuroni de ieşire. În Figura 2.19 au fost reprezentate primii treizeci de paşi temporali din cei o mie ai simulării în care s-a efectuat măsurătoarea.

După ~800 de cicluri de învăţare, caracterele de la 0 la 9 au fost identificate corect în proporţie de 95%.

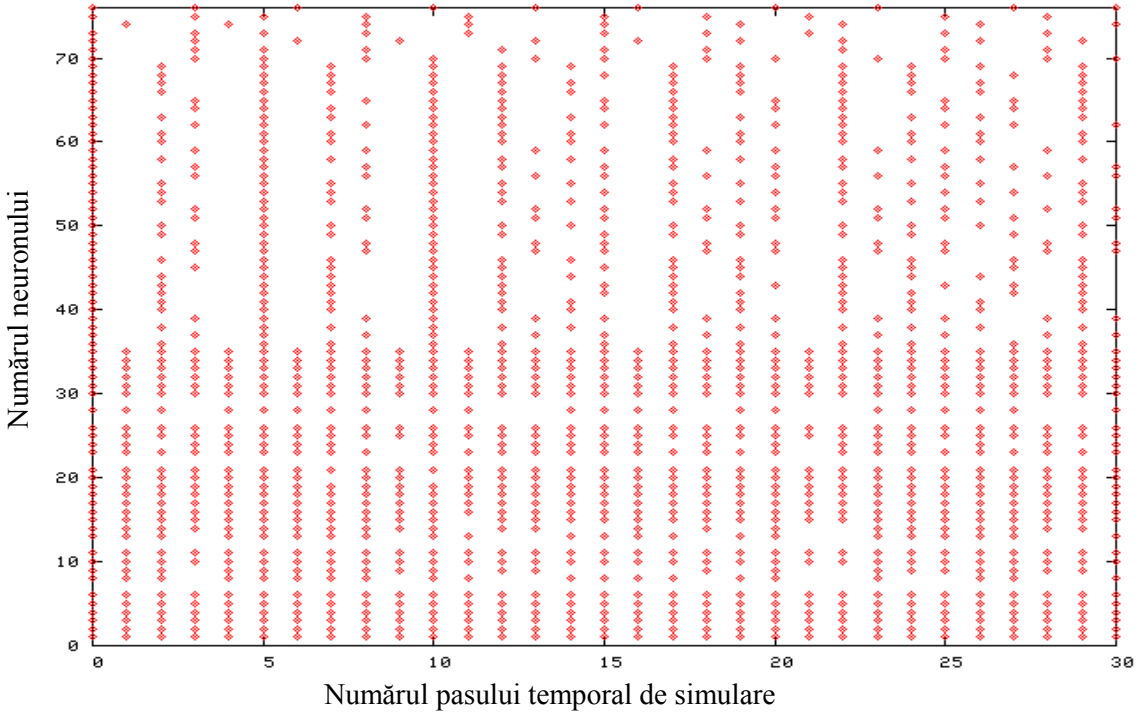


Figura 2.19 Rezultatul măsurătorii efectuate pe ieşirea RNP din Figura 2.17. Primii treizeci de paşi temporali

2.8. Concluzii

În prima parte a acestui capitol s-a discutat modelarea reţelelor neuronale neuromorfe, expunând comportamentul intrinsec temporal al acestora, utilizarea impulsurilor în comunicarea datelor în cadrul acestora şi evidenţiind capacitatea de calcul mai ridicată decât în cazul reţelelor neuronale bazate pe modele clasice.

Studiul neuronilor biologici a rezultat în apariţia mai multor ipoteze pentru codarea valorilor reale în trenuri de impulsuri. Acestea pot apărea ca o codare a ratei impulsurilor, care ia în considerare

numai valoarea medie a timpului de apariţie a unui impuls sau ca codarea exactă a momentelor temporale de apariţie a acestor impulsuri.

Când vorbim despre RNP există de fapt o supraabundenţă de diferite modele, de la cele simple de tipul integrează şi activează prin neuroni cu diferite modele sinaptice până la modele comportamentale complete descrise prin ecuaţii diferenţiale. Există câteva studii teoretice (Maass, 1996) care arată că unele dintre aceste modele sunt mai eficiente decât celelalte, dar aceste studii se limitează doar la un set redus de modele şi scheme de codare. O explicaţie generală a capacităţii de calcul a diferitelor modele neuronale pulsative şi a schemelor de codare conexe reprezintă şi astăzi o temă deschisă cercetărilor actuale.

În orice studiu legat de dinamica unei reţele neuronale pulsative, sunt două probleme specifice, şi anume ce model descrie dinamica pulsaţiei fiecărui neuron respectiv cum sunt conectaţi neuronii. O alegere greşită a modelului sau o conexiune greşită, poate duce la rezultate complet eronate.

În cadrul acestui capitol s-a elaborat prima problemă, comparând neuronii pulsatori. Se vor prezenta diferite modele de neuroni pulsatori precum şi o clasificare a acestora.

În continuare am prezentat cele mai des utilizate metode de antrenare (ne-supervizate şi supervizate) utilizate în învăţarea reţelelor neuronale neuromorfe. Astfel, în această parte a capitolului se regăsesc discuţii asupra realizării învăţării Hebbiene competitive precum şi asupra modalităţilor specifice de aplicare a regulii back-propagation în reţele neuronale pulsative codificate temporal (algoritmul SpikeProp).

După acest studiu amănunţit al acestui domeniu de inteligenţă artificială prezintă modelul neuronal teoretic ales pentru implementările hardware ce urmează a fi prezentate în capitolele următoare.

A doua contribuţie importantă a acestui capitol este prezentare generală a strategiilor de simulare software a RNP împreună cu descrierea celor mai importante astfel de medii existente la ora actuală, dar şi a simulatorului neuronal implementare proprie, cu care am testat funcţionarea modelului dezvoltat. Se detaliază încorporarea algoritmilor de învăţare în procesul de simulare software şi se expun rezultatele experimentale obţinute pe această cale, care au stat la baza primelor publicaţii ale autorului (Bakó, Székely, & Brassai, Development of Advanced Neural Models. Software And Hardware Implementation, 2004) (Bakó, Analiza şi simularea sistemelor cu inteligenţă artificială neuromorfă, 2005) (Bakó, Székely, Dávid, & Brassai, 2004), (Bakó & Brassai, 2004) în acest domeniu de cercetare.

3. DISPOZITIVE FPGA ŞI IMPLEMENTAREA REŢELELOR NEURONALE ARTIFICIALE

În acest capitol sunt prezentate sistemele FPGA şi modalităţile de implementare ale reţelelor neuronale pe aceste sisteme. Urmează să fie examinată evoluţia sistemelor hardware reconfigurabile, în special a sistemelor FPGA.

În continuare vom descrie sistemul de dezvoltare FPGA Xilinx XC3S1000 utilizat la implementarea reţelelor neuronale. Sunt prezentate elementele de bază ale sistemului FPGA, şi accentul se va pune în special asupra modulelor de memorii Block RAM, module care stau de fapt la baza structurilor de reţele neuronale neuromorfe care vor fi prezentate în Capitolul 4.

Sunt prezentate moduri de abordare ale cuplării şi configurării pentru sisteme FPGA care să funcţioneze în combinaţie cu procesorul gazdă. De asemenea sunt introduse cele două moduri de implementare a logicii reconfigurabile: reconfigurarea compilată în timp şi reconfigurarea în timpul utilizării.

În următoarea parte a capitolului se discută despre maparea diferiţilor algoritmi de reţele neuronale pe sisteme FPGA şi sunt prezentate diferite sisteme de reţele neuronale hardware la baza cărora se află sisteme FPGA. Sunt trecute în revistă câteva sisteme, care se pot considera de bază, cum sunt RRANN, GANGLION, sistemele REMAP, RAPTOR2000 şi de asemenea sunt referiri la câteva teze de doctorat şi lucrări ştiinţifice publicate în ultimii ani.

Modalitatea de implementare a funcţiilor de activare respectiv alegerea unei aritmetici corespunzătoare joacă un rol foarte important în implementarea reţelelor neuronale pe sisteme hardware. Ultima parte a Capitolului prezintă aspecte cu privire la modalităţile de implementare a funcţiilor de activare şi aritmetica neuronală utilizată pentru reprezentarea datelor.

Sistemele FPGA sunt circuite integrate construite din configuraţii de blocuri logice (CLB, configuration logic blocks) interconectate printr-o reţea de conexiuni. Atât blocurile logice cât şi interconexiunile sunt configurabile (Bondalpati, 2001). Aceste dispozitive pot fi configurate şi reconfigurate de proiectant printr-un software CAD. Evoluţia sistemelor FPGA seamănă mai mult cu evoluţia structurilor memoriilor ROM. Prima dată au apărut structurile FPGA bazate pe elemente fuzibile analogice structurilor PROM. Alte structuri bazate pe elemente fuzibile au fost Programmable Logic Array (PLA) şi Programmable Array Logic (PAL).

La modul general, toate aceste structuri bazate pe elemente fuzibile sunt clasificate sub denumirea de dispozitive logice programabile (PLD Programmable Logic Device). Limitările acestor structuri PLD au dus la apariţia structurilor FPGA bazate pe memorii RAM. Structurile FPGA disponibile în prezent sunt de acest fel, care permit realizarea unei configuraţii prin programarea memoriei RAM a blocurilor CLB. Structurile FPGA au început să fie disponibile la sfârşitul anilor 80 şi la începutul anilor 90.

Din acel moment cercetătorii şi proiectanţii au ajuns la concluzia că este posibilă realizarea sistemelor de calcul reconfigurabile prin sisteme FPGA. Arhitectura reconfigurabilă a început să-şi arate fezabilitatea însemnată de aplicare numai recent, dar ideile de bază s-au conturat aproape cu ideea calculatorului programabil cu scop general. În anul 1966, János Neumann, recunoscut ca părintele modelului serial al calculatoarelor programabile, a conceput de asemenea un automat de calcul, compus dintr-o reţea simplă de celule, care pot fi configurate pentru executarea unor operaţii computaţionale (Dehon, 2000).

Minnick în 1971 a descris o matrice celulară programabilă utilizând bistabile pentru a memora un context de configuraţie.

Posibilitatea de utilizare eficientă a sistemelor de hardware reconfigurabil realizat într-un FPGA a apărut când era deja posibilă introducerea sutelor de elemente programabile pe un singur chip.

După cum considera Moore "numărul transistoarelor într-un circuit integrat se va dubla aproximativ în 18 luni". Astăzi este posibil să utilizăm sisteme FPGA cu peste 10 milioane de porţi într-un singur chip.

În general, pentru a implementa un circuit dorit, sistemele FPGA sunt reconfigurate de un controller extern. Reconfigurarea se poate realiza cu o viteză corespunzătoare, ce se apropie de o funcţionare în timp real, depinzând de timpul necesar pentru reconfigurare respectiv timpul necesar pentru încărcarea datelor de configurare.

Ca şi la toate sistemele VLSI, densitatea şi viteza în creştere a circuitelor pe bază de siliciu este limitată la modulele de comunicaţie şi de la această regulă nici sistemele FPGA nu constituie o excepţie. Hardware-ul reconfigurabil poate fi clasificat în una din următoarele categorii, presupunând că un astfel de dispozitiv are câteva nivele proprii de configurabilitate:

Hardware-ul configurabil poate fi configurat o dată sau de două ori.

Hardware-ul reconfigurabil –are avantajul că poate fi reconfigurat de mai multe ori conform funcţiilor pe care trebuie să le îndeplinească. Aceste sisteme conţin elemente ca Dispozitive Logice Programabile (PLA), care se comportă ca un tabel de adevăr, funcţii Booleene, automat de stare, sau memorii PROM, care pot fi utilizate ca memorii nevolatile pentru microprocesoare şi microcontrolere. Câteva variante ale sistemelor reconfigurabile sunt *in-system programmable* (ISP) şi pot fi reprogramate în timp ce sunt încă rezidente pe placa de circuit.

Hardware-ul dinamic reconfigurabil a apărut odată cu apariţia sistemelor FPGA bazate pe memorii SRAM. Celulele SRAM permit configurarea logicii unui sistem FPGA prin simpla încărcare a unui şir de biţi în modulul SRAM. Deci acestea sunt sisteme care poate fi rapid reprogramate pentru realizarea unei sarcini. De exemplu la pornire, dispozitivul FPGA poate fi configurat pentru a realiza o diagnostizare a propriilor circuite înainte de reconfigurarea dinamică pentru realizarea sarcinii principale pentru care a fost proiectat.

Hardware-ul parţial reconfigurabil. La majoritatea sistemelor hardware dinamic reconfigurabile problema care apare este că, în timpul reconfigurării, operaţiile sale trebuie întrerupte, şi conţinutul memoriilor de configurare SRAM reîncărcate, rezultând o pierdere irecuperabilă de date din registrele sistemului FPGA. A devenit astfel necesară dezvoltarea unui nou tip de dispozitiv FPGA, care permite reconfigurarea dinamică a unei porţiuni a logicii interne fără deconectarea conexiunilor I/O sau a semnalelor de tact de sistem, şi care asigură funcţionarea continuă a operaţiilor dispozitivului în porţiunile care nu sunt supuse reconfigurării. O caracteristică care oferă un avantaj important este conţinutul registrelor interne care nu se pierd în timpul reconfigurării. Astfel rezultatul unei instanţieri poate fi preluat de instanţierea dinamic reconfigurată.

3.1. Evoluţia dispozitivelor reconfigurabile

Există mai multe metode aflate la dispoziţia proiectanţilor de sisteme reconfigurabile pentru integrarea diferitelor componente discrete într-un singur chip VLSI ASIC (Hauck, 1995), fiecare metodă având o capacitate specifică, performanţă, flexibilitate şi cost unitar atât din punctul de vedere al timpului, cât şi financiar. Acest subcapitol evidenţiază funcţiile diferitelor tehnologii, subliniind noile oportunităţi oferite de folosirea FPGA-urilor.

Un *sistem full-custom* este acel sistem la care întregul circuit este construit după cerinţele proiectantului. Rezultatul este un chip cu performanţe foarte bune, cu o utilizare optimă a substratului de siliciu şi este câteodată singura metodă de implementare luată în calcul în cazul unor aplicaţii speciale, cum ar fi microprocesoarele realizate cu tehnologii de vârf. Costurile de fabricaţie şi timpul iniţial de punere în funcţiune sunt de asemenea mari, deoarece flexibilitatea oferită proiectantului rezultă într-un design al chipului foarte complex şi care necesită mult timp.

O proiectare pe bază *de celule* sacrifică flexibilitatea şi performanţa unui sistem full-custom pentru a accelera procesul de proiectare. Aceasta se realizează prin folosirea unui set de restricţii de proiectare şi celulele de format standard îngăduind folosirea instrumentelor soft la automatizarea procesului de proiectare.

Problema cu ambele metode, full custom şi pe bază de celule, este faptul că ele trebuie începute de la zero. În cazul MPGA-rilor (Mask Programmable Gate Array), marea parte a procesului de fabricaţie se realizează mai înainte. De exemplu, un MPGA poate consta din seturi de transistoare în locaţii specifice care sunt parţial interconectate pentru a forma baza mapării unor tipuri specifice de matrici de porţi. Astfel, tot ce trebuie să facă un proiectant este interconectarea elementelor necesare pentru a atinge funcţionalitatea dorită. Această metodă reduce atât costul, cât şi timpul de producţie, deoarece o astfel de bază poate stoca cantităţi mari de proiect standard, chipuri parţial realizate. Proiectantul are o flexibilitate destul de mare şi este limitat doar de rutările aflate la dispoziţie pe chip. Din păcate, este inevitabil să nu fie anumite ineficienţe, deoarece unele proiecte necesită resurse specifice sau lăţime diferită a benzii de rutare, decât cele care există pe chip.

Unul dintre primele chipuri de uz general prefabricate care au ajuns să fie folosite la scară largă a fost memoria read only programabilă (PROM). Ea constă dintr-un şir de celule preprogrambile în care prima dată se scriu datele. Informaţia de pe un PROM se stochează folosind una dintre următoarele metode:

- *Fuses/antifuses*- se poate folosi pentru a stoca biţi, condiţia cărora va determina care valoare de biţi se va citi. Ele se activează prin aplicarea unei tensiuni de programare.
- *Erasable* PROM (EPROM)- ele sunt programate prin folosirea unei tensiuni mai mari decât tensiunea de alimentare, dar se deosebesc de prima categorie prin faptul că ele se pot şterge prin expunerea la lumină ultravioletă (UV). Acest lucru permite ca EPROM să fie reprogramabil.
- *Electrically erasable* PROMs (EEPROMs)(PROM care se poate şterge pe cale electrică)- sunt programate prin folosirea tensiunii înalte, dar în acest caz chipul se poate şterge prin metode pur electrice. O altă tehnologie care merită amintită, dar care nu este clasificată strict ca şi PROM, este memoria cu acces aleator *(RAM)*. Caracteristicile sale sunt identice cu cele ale PROM-ului, cu deosebirea că acesta se poate citi şi scrie, şi astfel, reprograma în timp ce sistemul este în funcţiune.
- *Dispozitivele logice programabile (PLD)* au fost proiectate pentru a implementa circuite logice şi constau dintr-un şir de porţi AND urmat de un şir de porţi OR într-o sumă-de-produse.
- *Programmable array logic* (PAL) este cel mai larg răspândit tip de PLD cu un plan AND programabil urmat de un plan OR fixat.
- *Programmable logic arrays* (PLA) constituie un tip mai flexibil al PLD-urilor, care permit atât planului AND, cât şi OR să fie programabile.
- *Generic array logic* (GAL) constituie un tip mai avansat de PLD care se poate configura pentru implementarea diferitelor tipuri de PAL cu inversia ieşirii opţională.
- *Complex programmable logic devices* (CPLDs) sunt sisteme complexe, care conectează mai multe elemente PLD folosind o matrice de comutare.
- PLD-urile se pot implementa prin folosirea tehnicilor de programare fuse/antifuse, EPROM sau EEPROM.
- *Field programmable gate arrays (FPGA)* sunt dispozitive prefabricate construite pentru implementarea unor circuite multistrat în loc de termeni sumă-a-produsului simple PLD, singura limitare fiind resursele aflate la dispoziţie în circuitele chipului. Din păcate, efectul acestei tehnologii este apariţia unei întârzieri a propagării care nu se poate estima. Trebuie făcută menţiunea că în unele cazuri se referă la FPGA ca fiind dispozitive CPLD, dar cu toate acestea noi ne vom referii la ele ca FPGA-uri, pentru a evita ambiguităţile. FPGA-urile nu sunt disponibile doar prin tehnologiile fuse/antifuse, EPROM sau EEPROM; dar şi în SRAM sau RAM dinamic (DRAM) (Motomura, Aimoto, Shibayama, Yabe, & Yamashina, 1998). Versiunile SRAM şi DRAM permit reconfigurarea rapidă "in-circuit" a chipului, dar sunt dezavantajate din punctul de vedere al substratului de siliciu necesar pentru tehnologia RAM.

Reconfigurarea poate fi parţială sau completă, ceea ce depinde de dispozitiv, şi nu trebuie să rezulte neapărat în distrugerea blocurilor.

FPGA-ul (Brown, Francis, Rose, & Vranesic, 1993) (Oldfield & Dorf, 1995) s-a introdus pentru prima dată în 1985 de Xilinx şi de atunci multe alte companii au lansat produse similare.

3.1.1. Evaluarea performanţei aplicaţiilor bazate pe circuite FPGA

Inovaţiile remarcabile din domeniul tehnologiei computaţionale (Asanovic, 2006) îndeplinesc previziunile pe termen lung ale NASA (Sobieski & Storaasli, 2004) pentru calcule ştiinţifice şi inginereşti mai rapide. O inovaţie de prim plan este valorificarea circuitelor FPGA (Field Programmable Gate Arrays) pentru accelerarea aplicaţiilor de calcul de înaltă performanţă (High-Performance Computing - HPC) cu unul sau mai multe ordine de mărime comparativ cu procesoarele tradiţionale. FPGA, o formă complet nouă de logică programabilă, a fost inventată în 1984 de către Ross Freeman, co-fondator al Xilinx Corporation. Arhitecturile FPGA sunt extrem de flexibile şi interconectează mii de module încorporate cum ar fi sumatoare, multiplicatoare, module de memorie şi structuri de elemente logice utilizabile de exemplu în prelucrarea digitală a semnalelor şi comunicaţia de mare viteză (Hypertransport, PCI-EXPRESS). Spre deosebire de microprocesoarele programabile convenţionale

„fixe”, circuitele FPGA sunt reconfigurabile. Unele circuite FPGA pot fi parţial reconfigurate chiar în timp ce alte porţiuni ale aceluiaşi circuit funcţionează.

Circuitele FPGA Virtex-4 pot avea unul sau mai multe procesoare PowerPC încorporate. Piaţa circuitelor FPGA de 2 miliarde de dolari, cu o creştere rapidă (15-20% / an) este dominată de Xilinx şi Altera. Folosirea circuitelor FPGA se extinde rapid şi în aplicaţii mai mici, aeronautice şi de calcul încorporat de înaltă performanţă (High-Performance Embedded Computing - HPEC).

3.1.1.1. Caracteristici FPGA

Structura circuitelor FPGA este strict reglementată în comparaţie cu cea a microprocesoarelor, simplificând astfel procesul de fabricaţie, şi a permis circuitelor FPGA să fie printre primele care şi-au redus dimensiunile (90nm => 65nm => 45nm). La fiecare ciclu de tact, algoritmii FPGA (când au fost codificaţi cu scopul de a mări la maximum numărul de operaţii executate în paralel) îşi folosesc aproape 100% din resursele de siliciu, comparativ cu microprocesoarele mai puţin eficiente care îşi folosesc mai puţin de 2% din resursele de siliciu (în timp ce consumă de 10 ori mai mult decât FPGA pentru a executa doar una sau două operaţii).

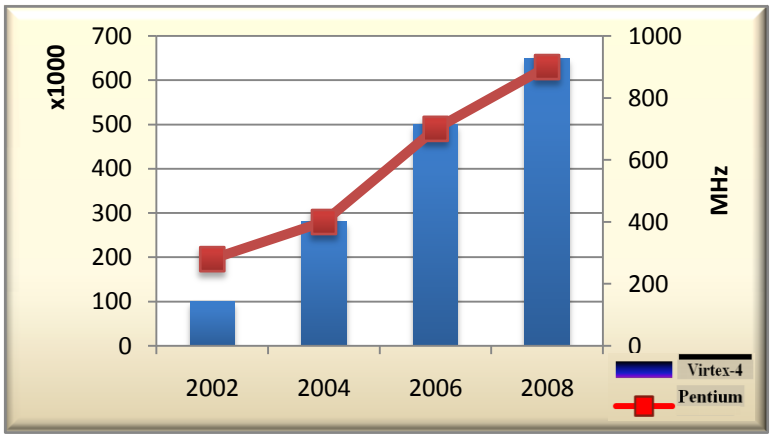


Figura 3.1 Caracteristici FPGA: evoluţia creşterii unităţilor logice şi a vitezei de tact

Figura 3.1 şi Figura 3.2 arată câteva caracteristici cheie ale circuitelor FPGA. Spre deosebire de microprocesoare, circuitele FPGA continuă să avanseze la viteza legii lui Moore, şi mai au mult până să atingă pragul limită de unităţi logice şi viteză de tact Figura 3.1). Viteza de tact a circuitelor FPGA (frecvent în jur de 100-200 MHz pentru multe aplicaţii) mai are mult de crescut până să se lovească de problemele de supraîncălzire ce au determinat microprocesoarele să adopte o structură multi-core cu chipuri rulând la viteză de tact redusă. Când aplicaţiile FPGA sunt programate cu un grad ridicat de paralelism, viteza lor de calcul depăşeşte cu mult pe cea a microprocesoarelor (Figura 3.2). Cum era de aşteptat, pentru sisteme de comunicaţie de mare viteză, lăţimea de bandă pentru memorie şi I/O depăşeşte de asemenea cu mult pe cea a microprocesoarelor (Figura 3.2).

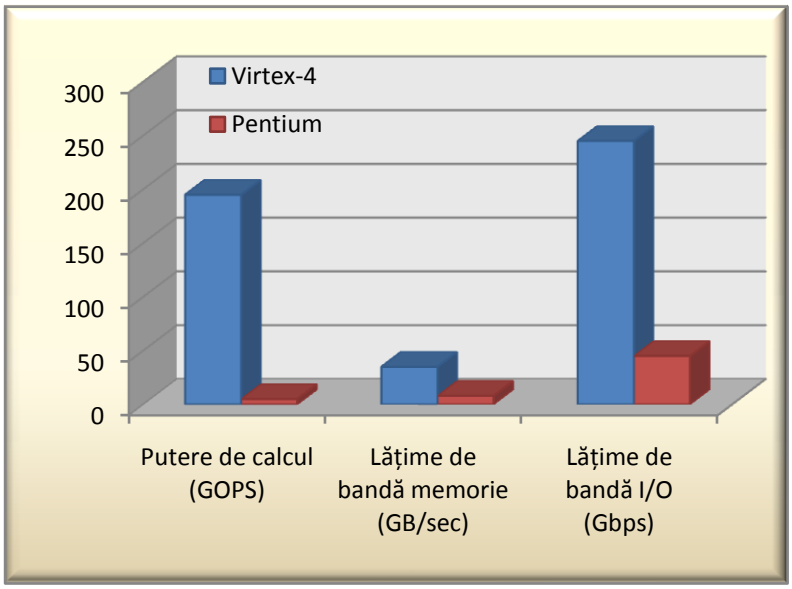


Figura 3.2 Caracteristici FPGA: evoluţia creşterii vitezei de calcul şi a lăţimii de bandă

3.1.1.2. Programarea circuitelor FPGA

Cum circuitele FPGA au fost dezvoltate de proiectanţi de circuite logice, se programează în mod tradiţional folosind limbaje de programare de descriere hardware cum ar fi VHDL şi Verilog. Aceste limbaje necesită cunoştinţele şi experienţa unui proiectant de circuite logice, învăţarea lor necesitând luni de zile. Chiar şi cu experienţa şi cunoştinţele necesare, programarea în VHDL sau Verilog este

extrem de greoaie, dezvoltarea unor prototipuri timpurii necesită luni de programare, iar perfecţionarea şi optimizarea chiar ani. Programarea FPGA, spre deosebire de compilatoarele HPC, este mult încetinită de paşii mari în plus necesari pentru a sintetiza un proiect, a plasa componentele şi a concepe conexiunile dintre acestea.

Odată efortul depus pentru programarea de aplicaţii specifice în VHDL, performanţa lor pe FPGA este greu de atins. În particular, aplicaţii care folosesc doar operaţii simple cu numere întregi sau operaţii logice (comparaţie, adunare, înmulţire) cum ar fi comparaţii de secvenţe ADN, criptografie, logica jocului de şah, rulează extrem de rapid pe FPGA. Cum aplicaţiile care folosesc operaţii de dublă precizie în virgulă mobilă au epuizat rapid numărul de structuri de elemente logice disponibile pe primele circuite FPGA, acestea au fost evitate adesea pentru calcule de înaltă precizie. Această situaţie s-a schimbat însă pentru ultimele modele de FPGA de la Xilinx (Virtex-4 şi Virtex-5) care au suficiente unităţi logice pentru a acomoda în jur de 80 module de multiplicare pe 64 de biţi (Strenski, 2007).

Capabilitaţile primelor circuite FPGA erau foarte potrivite pentru calcul încorporat de înaltă performanţă (HPEC) cu destinaţie specială. Folosirea lor în HPC cu destinaţie generală a fost iniţial restrânsă la supercomputere reconfigurabile de primă generaţie. (de ex. Starbridge Systems, SRC, Cray XD1). Această primă generaţie a fost caracterizată de lipsa unui sistem de I/O de mare viteză şi a unei infrastructuri (compilatoare, librării) care să faciliteze aplicaţii supercomputer cu destinaţie generală, incluzând cod deja existent. Această situaţie se schimbă rapid odată cu ultima generaţie de supercomputere reconfigurabile şi circuitele FPGA pe care le folosesc. DRC Computer, Xtreme Data şi Xilinx în colaborare cu Intel oferă acum pe piaţă module conţinând cele mai noi circuite FPGA care se potrivesc în acelaşi socket folosit şi de microprocesoare şi folosesc aceeaşi linie de comunicaţie de mare viteză. Cray a ales circuitul DRC pentru a accelera performanţele liniei de supercomputere XT.

Alternative concurente la FPGA în domeniul HPC: Creşterea vitezei aplicaţiilor HPC este atât de necesară, astfel au apărut pe piaţă mai multe alternative. Deşi multe coduri bazate pe fizica tradiţională au fost scrise in limbajul secvenţial Fortran acum mai bine de 30 de ani, au supravieţuit remarcabil de bine mai multor generaţii de HPC: vector (via compilatoare), paralel (via MPI, OpenMP) şi acum primele generaţii de microprocesoare multi-core. Despre unele se presupune că ar putea suferi degradări masive ale performanţei, sau chiar va fi nevoie să fie rescrise pentru a folosi la maximum capacitatea a 8 sau mai multe „core”-uri / chip. Marii producători de chip-uri (Intel şi AMD) au depus eforturi substanţiale pentru a satisface cerinţele pieţei cu acceleratoare, concentrându-se în principal pe circuite FPGA ca un mod de a recâştiga performanţă. Cum microprocesoarele multi-core înfruntă probleme cu consumul de curent, răcirea, dimensiunile şi numărul porturilor de I/O, circuitele FPGA par mai atractive.

Alte trei tipuri de acceleratoare sunt disponibile arhitecţilor HPC: Cell (IBM), Array (ClearSpeed) şi procesoare grafice (Graphical Processors - GPUs). Ca şi circuitele FPGA, procesoarele grafice si Cell au o piaţă vastă de desfacere (grafică şi jocuri video), fapt ce duce la scăderea costurilor, promovarea concurenţei şi încurajarea dezvoltării în domeniu, făcându-le din ce în ce mai atractive pentru HPC. În schimb procesoarele Array sunt dispozitive specializate care trebuie amortizate pe un număr relativ redus de utilizatori. Procesoarele GPU necesită multă putere/răcire şi au probleme complexe de programare şi precizie a datelor de rezolvat înainte de a putea intra pe piaţa HPC. Programarea procesoarelor Cell 8+1 e probabil să fie considerată mai greoaie decât programarea circuitelor FPGA în VHDL sau Verilog, care are deja o bază mare de utilizatori. Odată cu dezvoltarea hardware-ului circuitelor FPGA, software-ul şi intrumentele FPGA devin mai uşor de folosit pentru HPC, incluzând spre exemplu MitrionC, Viva, DSPlogic, ImpulseC, Celoxica, Aldec, şi CHiMPS de la Xilinx, toate având ca ţintă specifică piaţa HPC.

3.1.1.3. Lăţimea de bandă şi localizarea datelor

Performanţa multor algoritmi este limitată mai degrabă de lăţimea de bandă decât de puterea de calcul. Astfel de limitări ale lăţimii de bandă pot apărea între microprocesor şi dispozitivul reconfigurabil, sau între dispozitivul de procesare şi propria sa memorie. Lăţimea de bandă dintre microprocesor şi dispozitivul reconfigurabil tinde să fie fixă şi probabil mai eficientă când se transferă un volum mare de date. Lăţimea de bandă a memoriei tinde să crească cu cât memoria este mai aproape de microprocesor. De exemplu, memoria cache internă a microprocesoarelor moderne este substanţial mai rapidă decât memoria cache externă, care de asemenea este mai rapidă decât memoria SDRAM externă. La fel se întâmplă şi în cazul memoriei dispozitivelor reconfigurabile. Algoritmii care sunt în

prezent limitaţi de lăţimea de bandă SDRAM a unui microprocesor pot fi limitaţi în mod similar pe un dispozitiv reconfigurabil.

Mai departe, pentru a putea profita de paralelism, se determină dacă există suficientă lăţime de bandă între microprocesor şi dispozitivul reconfigurabil, şi de asemenea între dispozitivul reconfigurabil şi resursele de memorie ale acestuia. De exemplu, pentru a calcula punctajul maxim de aliniament (maximum alignment score) între secvenţe de ADN, să considerăm o arhitectură în care microprocesorul transmite către circuitul FPGA secvenţa de interogare, secvenţa de date şi mai mulţi parametrii de punctaj. Numărul de punctaje pe care circuitul FPGA trebuie să le calculeze pentru a afla maximul este egal cu lungimea secvenţei de interogare înmulţită cu lungimea secvenţei de date. Pentru fiecare caracter transmis în secvenţa de date, circuitul FPGA trebuie să calculeze un rând întreg de punctaje. Determinarea fiecărui punctaj în parte necesită multe calcule pentru fiecare caracter transmis către circuitul FPGA şi astfel „blocajele" în lăţimea de bandă pe care se transmit secvenţele către circuitul FPGA devin puţin probabile. De asemenea, deoarece singura dată retransmisă de către circuitul FPGA este punctajul maxim, nici lăţimea de bandă de la circuitul FPGA către microprocesor nu va suferi limitări. Astfel, singura limitare a algoritmului este viteza cu care circuitul FPGA poate calcula punctajele.

3.1.1.4. Memorie suficientă?

Analiza exemplului de mai sus este valabilă doar dacă circuitul FPGA poate calcula şi stoca întregul tabel de punctaje într-o singură trecere. Fapt puţin probabil, deoarece ar necesita ca punctajele dintr-un rând întreg al tabelului să fie calculate şi stocate în paralel. Asta ar limita foarte mult dimensiunea secvenţei de interogare. Pentru ca circuitul FPGA să poată calcula tabela de punctaje pe segmente, trebuie să stocheze datele intermediare în memoria locală. Pentru a împărţi tabela de punctaje în segmente verticale, trebuie stocată în memorie ultima coloană a unui segment pentru a o folosi la calculul primei coloane din următorul segment. Deoarece numai o coloană trebuie stocată, e puţin probabil ca lăţimea de bandă a memoriei necesare va fi factorul limitant. Însă, dimensiunea acestei memorii va limita lungimea maximă a secvenţelor de interogare şi de date.

3.1.2. Clase de FPGA

Există patru mari clase de modele FPGA, aşa cum apare în Figura 3.3. Toate aceste patru arhitecturi sunt disponibile în comerţ, fiecare descriind poziţionarea liniilor de rutare şi a blocurilor logice. Se folosesc instrumente CAD sofisticate la proiectarea circuitelor logice care „traduc" desenul, arhitectura aflată la baza lor devenind transparentă. Alegerea arhitecturii depinde de aplicaţie, unele fiind mai potrivite în cazul anumitor sarcini decât altele.

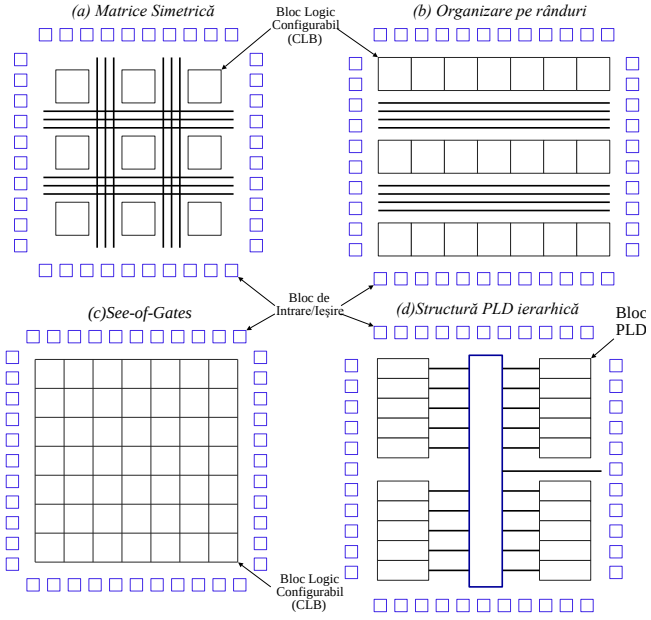


Figura 3.3 Principalele clase de FPGA

Blocurile logice configurabile (CLB) asigură elementele funcţionale pentru construcţia proiectului în FPGA. Cel mai important merit al CLB-ului este funcţionalitatea sa. Funcţionalitatea sa mărită permite implementarea unor funcţii logice mult mai complexe pe un singur CLB, dar totuşi, cum creşte funcţionalitatea unui CLB, tot aşa creşte şi dimensiunea sa, rezultatul fiind mai puţine CLB-uri pe fiecare FPGA. Un CLB conţine un număr de blocuri de construcţie standard, aranjate conform metodei specifice a fabricantului, pentru a permite implementarea diferitelor funcţii logice. Blocurile de construcţie sunt:

• *Tabele de căutare* (LUTs) –acceptă M intrări binare şi dau N ieşiri, unde N este de obicei 1. Se poate nota cu M-LUT-N un LUT pentru a evidenţia caracteristicile sale: ex. 3-LUT-1 este un tabel de căutare cu 3 intrări şi o ieşire. Un astfel de tabel de căutare poate fi folosit pentru implementarea unui port AND cu trei intrări, ilustrat în Tabelul 3. Numărul biţilor de ieşire care trebuie stocaţi pentru orice tabel de căutare sunt o funcţie directă a intrărilor: 2^M. Astfel, un 4-LUT-1 necesită 16 biţi stocaţi, iar un 3-LUT-1 are nevoie de doar 8. S-a arătat faptul că, pentru cele mai multe aplicaţii, soluţia optimă este un 4-LUT-1.

• *Matricile logice programabile* (PLA) au fot introduse datorită faptului că tabelele de căutare se dovedesc a fi ineficiente în cazul M>5. S-a examinat aria de eficienţă pentru PLA, cu M intrări, N ieşiri şi K termeni-produs, şi s-a dovedit că valorile optime sunt K=10, M=3 şi N=12 Acest dispozitiv oferă cu 4% mai multă funcţionalitate decât o implementare 4-LUT-1.

• *Bistabile* – sunt extrem de utile, deşi nu necesare, atunci când trebuie făcută o operaţie logică secvenţială. S-a arătat prin experimente că fără utilizarea unui bistabil, numărul CLB-urilor necesare implementării unei funcţii se dublează.

Tabelul 3 ***3-LUT-1***

A	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
B	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
C	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
X	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**

Exemplu pentru un tabel de căutare configurat pentru a funcţiona cu trei intrări şi un port de ieşire. CLB-urile pot să conţină alte componente, cum ar fi porţi logici, registre şi multiplexere, module sumatoare, totuşi, neexistând o implementare standard.

Problema pe care o întâlneşte orice producător la proiectarea unui FPGA este găsirea echilibrului optim între granularitatea CLB-ului ş funcţionalitatea sa. O posibilă rezolvare a acestei probleme este crearea unui şir neomogen de blocuri logice, dar un astfel de şir va avea întotdeauna elemente care nu se potrivesc unei anumite aplicaţii. Indiferent de configuraţia CLB-ului, eficienţa FPGA-ului depinde în mare măsură de instrumentele CAD. Dacă instrumentele CAD nu mapează circuitul logic în mod optim pe CLB, avantajul arhitectural va fi irosit.

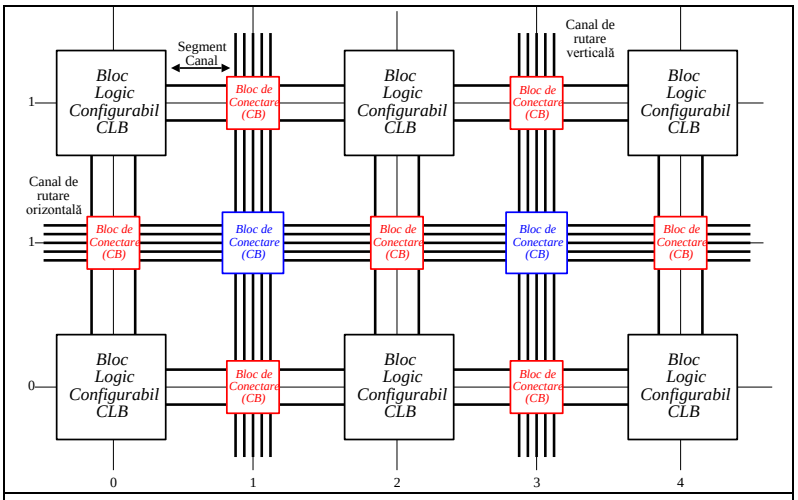


Figura 3.4 Exemplu de rutare tip matrice simetrică

Rutare în FPGA. Rutarea în FPGA se ocupă în mod esenţial de conectarea CLB-urilor cu alte CLB-uri si cu blocurile de intrare/ieşire (IOB), oferind astfel funcţionalitatea necesară. Echilibrul dintre spaţiul oferit rutării şi cel dedicat CLB-urilor este critic: dacă vor fi prea multe CLB-uri, conectarea sistemelor complexe va fi imposibilă, iar dacă vor fi prea puţine, rutarea va rămâne nevalorificată. Pe un FPGA standard, rutarea va ocupa apr. 70-90% din spaţiul total disponibil. Rutarea necesită mult spaţiu şi timp.

Există două tipuri de blocuri pe un FPGA, definite ca şi bloc de conectare (CB) şi bloc comutator (SB). În Figura 3.4 este reprezentată construcţia unei arhitecturi CLB simetrice, care foloseşte

două blocuri adiţionale şi esenţiale. Conexiunile interne posibile atât în CB şi SB depind de producător. Un producător trebuie să ia în considerare rutarea necesară arhitecturii respective, şi să optimizeze conectivitatea CB şi SB pentru a permite o flexibilitate maximă a interconexiunilor, în acelaşi timp păstrând o redundanţă minimă.

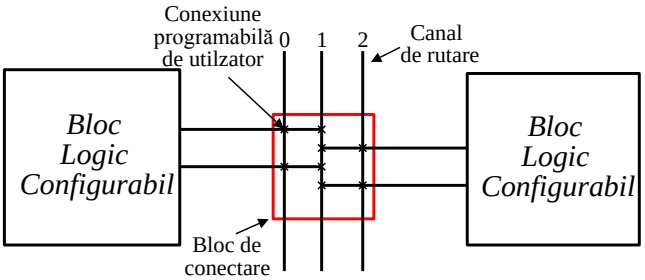


Figura 3.5 Exemplu Bloc de conectare

Blocurile de conexiune (CB) şi comutatoare (SB) se pot folosi pentru a ruta I/O de pe elementele CLB în canale de rutare. Un exemplu de bloc de conexiune apare în Figura 3.5 în care este prezentată un punct de comutare de rutare configurabilă folosind o intersectare, CB-ul având la rândul lui o topologie de interconectare predefinită.

Flexibilitatea este o problemă importantă aici, deoarece prea puţini comutatori sau conexiuni programabile ar putea însemna imposibilitatea interconectării a două linii. În Figura 3.6 vom prezenta un exemplu de bloc comutator.

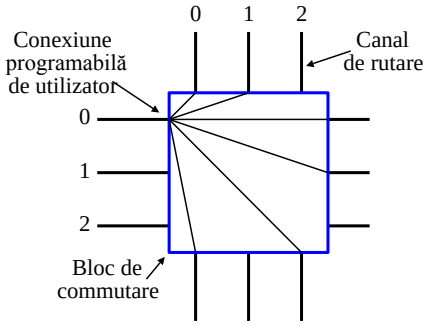


Figura 3.6 Exemplu de bloc comutator

Flexibilitatea unui bloc comutator este definită ca şi numărul segmentelor de cablaj la care orice linie de intrare poate fi conectată, în exemplul arătat aceasta fiind de 6. La proiectarea unei topologii de bloc comutator, producătorul trebuie să aleagă o topologie care să nu împiedice conectarea a două linii CLB.

Resursele de rutare în FPGA se clasifică în general în trei categorii, prin definirea utilizării lor primare. Aceste trei categorii sunt:

- *Interconectare cu scop general.* Se foloseşte pentru interconectări care cuprind una sau mai multe CLB-uri locale. Se implementează prin folosirea canalelor de rutare, blocurilor de conectare şi a blocurilor comutatoare, aşa cum s-a descris mai înainte. Un factor negativ îl constituie faptul că fiecare bloc comutator sau de conectare prin care trebuie să treacă un semnal, are o întârziere RC ca şi o rezultantă a problemelor de propagare a semnalelor la frecvenţe mai înalte ale dispozitivului.
- *Interconectare directă.* Asigură o conexiune directă cu unul sau cu toate CLB-urile învecinate la dreapta, stânga, în sus sau în jos.
- *Liniile lungi* se folosesc pentru rutarea conexiunilor care necesită interconectarea mai multor CLB-uri cu un defazaj minim, care asigură soluţii parţiale pentru semnalele care altfel ar traversa mai mulţi comutatori de rutare, rezultând întârzieri RC cumulative.

Un ultim şi foarte important aspect al rutării în FPGA sunt IOB-urile, fără de care un FPGA ar fi inutil. IOB-urile trebuie alese astfel încât să susţină mai multe moduri de ieşire, cum ar fi TTL (transistor-transistor logic) sau CMOS (complementary metal-oxid semiconductor), şi să asigure curent necesar, dar nu excesiv, pentru a permite funcţionarea corectă a oricărui dispozitiv ataşat. Din păcate, I/O este veriga slabă într-un FPGA şi se dovedeşte a fi disproporţională faţă de densitatea chipului.

FPGA-urile reconfigurabile în mod dinamic (DRFPGA) combină, în principiu, viteza unei soluţii hardware dedicate cu flexibilitatea softului. Astfel de FPGA-uri se pot reconfigura de un controller extern pentru implementarea oricărui set de funcţii Booleene dorit. Acesta este conceptul din spatele computării reconfigurabile dinamice: combinarea flexibilităţii softului cu viteza hardware-ului.
Există, în general, două motive pentru adoptarea unei reconfigurări dinamice:

- *Creşterea vitezei* prin folosirea unei arhitecturi specializate pentru o anumită problemă, rezultând o performanţă mult mai bună
- *Toleranţa faţă de eroare* îmbunătăţeşte procesul de fabricare, prin faptul că permite schimbări ale designului sau depăşirea uşoară a unor probleme.
- Dimensiunile şi complexitatea a celui mai mic bloc dintr-un dispozitiv reconfigurabil se pot clasifica în funcţie de granularitatea sa:
  - *De granularitate fină* - aceste arhitecturi constau din blocuri logice mici şi simple care sunt configurate pentru a realiza operaţii mai complexe.

- *De granularitate medie* - aceste arhitecturi constau din blocuri logice complexe, din care fiecare realizează o anumită parte semnificativă a calculelor. Calculele nestandardizate se pot face prin reconfigurarea reţelei de interconectare.
- *De granularitate dură* - constau dintr-un număr de unităţi de execuţie dintre care fiecare are propriul set de instrucţiuni. Fiecare unitate e mai simplă decât un microprocesor, dar integrată, astfel încât permite o viteză mare de comunicare. (Nagami, Oguri, Shiozawa, Ito, & Konishi, 1998).

Integrarea dispozitivului specifică cât de strâns este cuplat sistemul reconfigurabil la gazda sa:

- *Sistemele dinamice* sunt sisteme de inspiraţie biologică care nu sunt controlate de un dispozitiv extern. Aceste sisteme auto-evoluează.
- *Sisteme statice, cuplate strâns* - leagă elementele reconfigurabile strâns ca şi elemente de execuţie pe calea de date a unui procesor-gazdă.
- *Sisteme statice, cuplate larg* - se situează pe o placă separată de gazdă. Acest lucru se desfăşoară în general în detrimentul vitezei, deoarece datele trebuie transferate spre şi de la sistemele fiică.

Reconfigurabilitatea unei reţele de interconectare externă între unităţi reconfigurabile se poate de asemenea clasifica în două categorii:

- *Reţea externă reconfigurabilă* - extinde conceptul reconfigurării asupra mai multor circuite reconfigurabile în mod eficient, având ca şi rezultat o unitate reconfigurabilă de dimensiune mare. Totuşi, penalităţile depăşirii chipului sunt mari.
- *Reţele externe fixe* - folosesc conexiuni statice între circuitele reconfigurabile, limitând flexibilitatea în vederea creşterii vitezei şi minimizării costului.

3.1.3. Elementele unui circuit FPGA din familia Xilinx Spartan3

Arhitectura circuitului FPGA din familia Spartan-3 conţine următoarele elemente funcţionale programabile:

- **Blocuri logice configurabile** (CLBs) (Configurable Logic Blocks) –conţine tabele de căutare LUT cu scopul implementării de funcţii logice, şi elemente de stocare care se pot utiliza ca bistabile sau registre.
- **Blocuri de intrare ieşire** IOB (Input/Output Blocks) –acestea există pentru controlul fluxului de date între pinii de intrare/ieşire şi logica internă a circuitului FPGA. Fiecare bloc IOB suportă flux bidirecţional de date şi de asemenea permite operaţii cu trei stări. Recunoaşte 65 standarde de semnale diferite. Include registre Double Data-Rate.
- **Bloc de control digital al impedanţei** (DCI Digitally Controlled Impedance) asigură terminaţii on-chip, simplificând astfel complexitatea proiectului.
- **Blocuri RAM** permit stocarea de date în blocuri de memorie cu poartă dual (dual-port blocks) sau cu poartă simplă (single port) de 18-Kbit.
- **Blocuri multiplicator** –acceptă două numere binare pe 18 biţi ca intrare, şi calculează produsul.
- **Manager digital de semnal de tact** - asigură soluţii digitale pentru distribuirea, întârzierea, multiplicarea, divizarea şi modificarea fazei semnalului de tact.

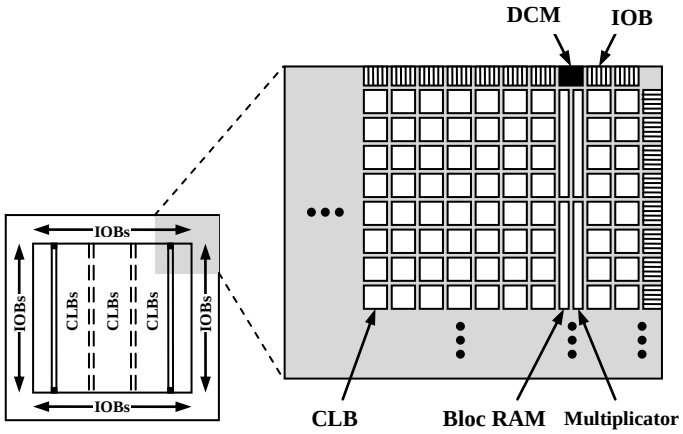


Figura 3.7 Organizarea elementelor în circuitul FPGA

Organizarea acestor elemente în circuitul FPGA este prezentată în Figura 3.7.

Toate aceste elemente constructive sunt conectate printr-o magistrală de control extrem flexibil programabilă, care este realizată pentru a asigura un număr foarte ridicat de procese de reconfigurare. Şirul de biţi care determină configurarea acestor conexiuni între elementele de bază ale FPGA se poate descărca prin porturi seriale, paralele sau dintr-o memorie PROM externă. Un aspect important este faptul că aceste circuite sunt capabile să funcţioneze la frecvenţe de până la 200 MHz, ceea ce permite realizarea unor reţele neuronale artificiale foarte rapide.

Utilizarea memoriilor are un rol foarte important în implementarea reţelei neuronale de tip CMAC şi RBF. Circuitul FPGA descris înglobează două tipuri de memorii:

- blocuri de memorie RAM- de 18Kbit.
- RAM distribuit în blocurile logice, de 64 de biţi cu poartă simplă sau de 32 de biţi cu poartă dublă (Xilinx, DS099, 2006).

Modulele de memorii Block RAM s-au utilizat cu succes la implementarea unei reţele neuronale de tip CMAC şi RBF, reprezentând elementele de bază pentru stocarea ponderilor reţelei şi a funcţiilor de activare. Deoarece atât memoriile Block RAM cât şi memoriile distribuite au un rol esenţial în realizarea reţelelor neuronale, în următorul subcapitol sunt descrise în amănunt aceste două elemente.

3.1.3.1. Elementele Bloc RAM

Pentru aplicaţiile care necesită memorii extinse, on-chip, FPGA-urile de generaţia Spartan™-3 asigură o multitudine de plăci Select RAM™ foarte eficiente. Utilizând diferite opţiuni de configurare, plăcile SelectRAM creează RAM, ROM, FIFO, tabele extinse de căutare, buffer-e circulare, registre, fiecare susţinând o varietate de date în ceea ce priveşte extinderea şi adâncimea lor. Prin utilizarea sistemului Xilinx Core Generator™ VHDL, sau Verilog Instantiation se pot specifica diferite caracteristici de proiectare.

Fiecare placă FPGA de generaţie Spartan™-3, inclusiv Spartan-3, Spartan-3L şi Spartan-3E se conturează ca şi memorii RAM de plăci multiple, organizate în coloane. Cantitatea memoriei RAM depinde de mărimea FPGA-ului de generaţie Spartan-3. Fiecare placă RAM conţine 18432 biţi de RAM static rapid, din care 16 Kbiţi au fost alocaţi pentru stocare de date, şi, în câteva configuraţii de memorie, 2 Kbiţi adiţionali au fost alocaţi ca biţi de paritate sau biţi adiţionali de date.

Sistemul Xilinx CORE Generator susţine module variate care conţin Block RAM pentru mecanisme Spartan-3, incluzând:

A. module RAM înglobate cu poartă simplă sau dublă
B. module ROM
C. module FIFO sincron sau asincron
D. module de memorii adresabile prin conţinut (CAM)

Memoria bloc RAM se poate încorpora în orice proiect prin folosirea unui modul „RAMB16” adecvat din colecţia de proiecte Xilinx.

Locaţia memoriei Block RAM şi mediul înconjurător. Memoria Block RAM este organizată în coloane. XC3S50-ul are doar o singură coloană de memorie bloc RAM, aşezată la două coloane CLB de marginea din stânga a dispozitivului. Dispozitivele Spartan-3 mai mari decât XC3S50 au două coloane de memorie Block RAM, adiacente marginilor din stânga respectiv dreapta ale dispozitivului, la două coloane CLB de I/O (intrare/ieşire) de la margine. Pe lângă aceste coloane de Block RAM de la margine, XC3S4000 şi XC3S5000 au câte două coloane în plus- patru coloane în total- distribuite în mod egal de la coloanele de la margini.

Adiacent la fiecare memorie Block RAM este un multiplicator hardware încorporat pe 18*18 biţi. Aşezarea memoriei bloc RAM şi a multiplicatorului încorporat unul lângă celălalt îmbunătăţeşte performanţa unor funcţii de procesare/prelucrare a semnalelor digitale.

Conectarea specială din jurul memoriei Block RAM asigură o distribuţie eficientă a semnalelor referitoare la liniile de adresă şi de date. Pe lângă aceasta, unele dotări speciale asigură ca memoriile Block RAM multiple să poată fi aşezate în cascadă pentru a crea memorii mai largi sau mai adânci.

Fluxurile de date

Memoria Block RAM Spartan-3 este construită din memorie dual-port şi sprijină simultan următoarele operaţii:

- Porturile A şi B funcţionează separat ca şi un RAM cu port simplu independent, susţinând operaţii simultane de citire şi scriere folosind câte un singur set de linii de adresă şi de date.

- Portul A este un port de scriere cu o adresă de scriere separată şi portul B este portul de citire cu o adresă de citire separată. Lărgimea semnalelor de date poate să difere la cele două porturi.
- Portul B este un port de scriere cu o adresă de scriere separată şi portul A este portul de citire cu o adresă de citire separată. Lărgimea semnalelor de date poate să difere la cele două porturi.

Semnalele conectate la un primitiv al memoriei Block RAM se împart în patru categorii:.

- Intrările şi ieşirile de date.
- Intrările şi ieşirile de ponderi, accesibile când portul de date este de o mărime de un byte sau mai mare.
- Intrări de adrese pentru selectarea unei locaţii specifice a memoriei.
- Semnale de control pentru a administra diferite operaţii de scriere, citire sau de setare/resetare.

Intrări şi ieşiri de date. Lăţimea totală de date a unui port include magistrala de date şi magistrala de paritate.

Magistrala de intrări de date -DI[#:0] (DIA[#:0], DIB[#:0])

Magistrala de intrări de date este sursa datelor care vor fi înscrise (introduse) în RAM.

Datele de la magistrala de intrare DI sunt înscrise în celulele RAM specificate de magistrala de adresă de intrare, ADDR, prin transmiterea unui semnal cu front crescător la semnalul CLK, când intrările de validare semnal de tact EN şi validare scriere WE sunt la nivel logic 1.

Magistrala de ieşire de date -DO[#:0] (DOA[#:0], DOB[#:0])

Magistrala de ieşire de date, DO, reprezintă conţinutul celulelor de memorie selectate de magistrala de adresă, ADDR, la un front crescător al semnalului de tact în cursul unei operaţii de citire. În cursul unei operaţii de scriere simultană, comportamentul registrului de date de ieşire este controlat de atributul WRITE_MODE.

Intrările şi ieşirile de paritate

Cu toate că ne referim la ele aici ca şi biţi de „paritate”, intrările şi ieşirile de paritate nu au o funcţionalitate specială, şi pot fi folosite ca şi biţi adiţionali de date.

Intrări de adrese. Fiind un RAM dual-port, ambele porţi operează în mod independent, accesând acelaşi set de celule de memorie de 18 K-biţi.

Magistrala de adrese - ADDR[#:0] (ADDRA[#:0], ADDRB[#:0]) Magistrala de adrese selectează celulele de memorie pentru operaţiile de citire sau scriere.

Semnale de control

Ceas - CLK (CLKA, CLKB) Fiecare port este sincronizat în totalitate cu semnalele de ceas independente. Toate semnalele de intrare ale portului au fixaţi timpii de validare relativ faţă de semnalul de tact CLK.

Semnalul de validare EN

Intrarea semnalului de validare, EN, controlează operaţiile de citire, scriere, setare/resetare. Când EN este pe nivel logic 0, nu sunt înscrise date, şi ieşirile DO şi DOP păstrează ultima valoare. Polaritatea EN este configurabilă şi este activă pe nivel logic 1.

Semnalul de validare de scriere — WE (WEA, WEB)

Semnalul de validare de scriere, WE, controlează înscrierea datelor în RAM. Când ambele semnale de validare, EN şi WE sunt active la frontul pozitiv al semnalului de ceas, semnalele din magistralele de intrare a datelor şi de paritate sunt înscrise în zona de memorie selectată de magistrala de adresă. Registrul de ieşire a datelor este încărcat sau nu în funcţie de atributul WRITE_MODE.

Setare/resetare sincron - SSR (SSRA, SSRB) Intrarea de setare/resetare sincron, SSR, forţează registrele de ieşire de date să ia valoarea specificată de atributul SRVAL. Când SSR şi semnalul de validare sunt la nivel logic 1, registrele de ieşire pentru ieşirile DO şi DOP sunt setate la „0” sau „1” în funcţie de parametrul SRVAL.

O operaţie de setare/resetare sincron nu influenţează celulele de memorie RAM şi nu perturbă operaţiile de scriere la celălalt port.

Semnale de control de inversare/schimbare. Pentru fiecare port, toate cele patru semnale de control- CLK, EN, WE şi SSR- au fiecare câte o opţiune individuală de inversare. Fiecare semnal de control

poate fi configurat să fie activ la nivel logic ‘1’ sau ‘0’, iar ceasul poate fi activ la un front pozitiv sau negativ fără a fi necesar utilizarea altor resurse logice.

Organizarea memoriei. Organizarea datelor sau aspectul proporţional al unui bloc RAM sunt configurabile, aşa cum apare în Tabelul 4. Dacă calea de date este de o lăţime de 1 byte sau mai largă, blocul RAM asigură biţi adiţionali pentru a susţine paritatea pentru fiecare byte. În consecinţă, o organizare a memoriei de 1Kx18 este de o lăţime de 18 biţi cu 16 biţi (2 byte) alocaţi datelor plus 2 biţi de paritate, câte unul pentru fiecare byte. De asemenea, cantitatea fizică de memorie accesibilă de la un port depinde de organizarea memoriei. Pentru memorii de o lăţime de 1 byte sau mai largi, există 18 Kbiţi de memorie accesibilă. Pentru memorii de o lăţime mai mică, numai 16 biţi de memorie sunt accesibile din cauza lipsei biţilor de paritate în această organizare. De regulă, 16 Kbiţi sunt alocaţi semnalelor de date şi 2 Kbiţi semnalelor de paritate în cazul blocului RAM de 18 Kbiţi.

Tabelul 4 Organizarea memoriei Bloc RAM

Organizare	Nr biţi adrese	Nr biţi date	Nr biţi paritate	DI-DO	DIP/DOP	ADDR	Primitivă port simplu	Total RAM KBit
512x36	512	32	4	(31 : 0)	(3:0)	(8:0)	RAMB16_S36	18K
1Kx18	1024	16	2	(15 : 0)	(1:0)	(9:0)	RAMB16_S18	18K
2Kx9	2048	8	1	(7 :0)	(0:0)	(10:0)	RAMB16_S9	18K
4Kx4	4096	4	-	(3:0)	-	(11:0)	RAMB16_S4	18K
8Kx2	8192	2	-	(1:0)	-	(12:0)	RAMB16_S2	18K
16Kx1	16384	1	-	(0:0)	-	(13:0)	RAMB16_S1	18K

Modurile de scriere/citire

Comportamentul de citire în timpul scrierii simultane - WRITE_MODE

Pentru a maximiza performanţele datelor şi ale utilizării memoriei dual-port, la fiecare front al ceasului, memoria bloc RAM susţine unul dintre cele trei moduri de scriere pentru fiecare port de memorie. Aceste moduri determină datele care sunt disponibile la registrele de ieşire după un front de ceas valid la scriere la acelaşi port. Modul iniţial, WRITE_FIRST, asigură compatibilitatea în retrospectivă cu arhitecturile Virtex™/Virtex-E şi Spartan-IIE FPGA, şi este de asemenea starea iniţială pentru echipamentele Virtex-II/Virtex-II Pro™. Totuşi, modul READ_FIRST este cel mai util deoarece creşte eficienţa memoriei Block RAM la fiecare ciclu al ceasului, permiţând echipamentelor să folosească lărgimea de bandă la maxim. În modul READ_FIRST, un port de memorie susţine operaţii simultane de citire şi scriere la aceeaşi adresă la acelaşi front al ceasului, independent de orice complicaţii de temporizare. Tabelul 5 subliniază modul în care aranjamentul WRITE_MODE influenţează registrele de ieşire la acelaşi port şi cum influenţează registrele de ieşire la portul opus în timpul accesului simultan la aceeaşi adresă.

Tabelul 5 Influenţa modului WRITE_MODE

Modul de scriere	Efectul asupra aceluiaşi port	Efectul asupra portului opus (doar mod dual-port, aceeaşi adresă)
WRITE_FIRST Citeşte după scriere (din oficiu)	Datele de la intrările DI, DIP sunt scrise într-o zonă specificată a memoriei RAM şi apar simultan la ieşirile DO, DOP.	Invalidează datele la ieşirile DO, DOP.
READ_FIRST Citeşte înainte de scriere (recomandat)	Datele din zona specificată RAM apar la ieşirile DO, DOP. Datele de la intrările DI, DIP sunt scrise într-o zonă specificată.	Datele din Zona specificată a RAM apar la ieşirile DO, DOP.
NO_CHANGE Fără citire asupra scrierii	Datele de la ieşirile DO, DOP rămân neschimbate. Datele de la intrările DI, DIP sunt scrise în zona specificată.	Invalidează datele la ieşirile DO, DOP.

Selectarea modului este setat prin configuraţie. Unul dintre aceste trei moduri este setat în mod individual pentru fiecare port printr-un atribut.

Modul WRITE_FIRST este modul de operare din oficiu din considerente de compatibilitate în retrospectivă. Pentru proiecte noi se recomandă modul READ_FIRST.
În acest mod, datele de intrare sunt scrise în zona de memorie RAM adresată şi sunt stocate simultan în registrele de ieşire de date, rezultând în operaţii transparente de scriere, aşa cum apare în Figura 3.9. În Figura 3.8 este prezentată ciclul de semnale pentru o operaţie de scriere WRITE_FIRST.

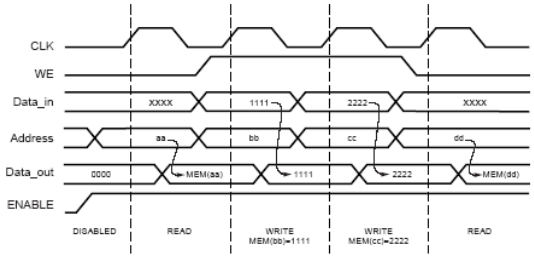


Figura 3.8 Ciclul de scriere in modul WRITE_FIRST

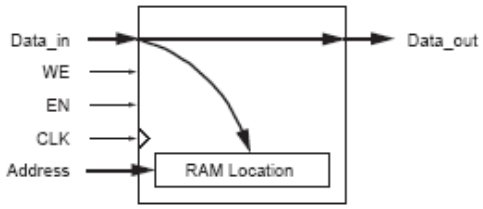


Figura 3.9 Fluxul de date in modul WRITE_FIRST

În modul READ_FIRST, datele stocate anterior la adresa de scriere apar la registrele de ieşire, în timp ce datele de intrare sunt stocate în memorie, rezultând o operaţie de citire-înainte-de-scriere aşa cum apare în Figura 3.10. Datele RAM mai vechi apar la ieşirea de date în timp ce datele RAM noi sunt stocate în zona de memorie RAM specificată.

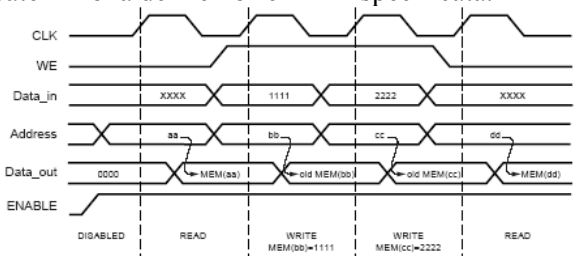


Figura 3.10 Ciclul de scriere în modul READ_FIRST

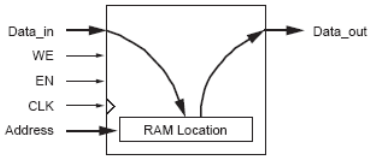


Figura 3.11 Fluxul de date in modul READ_FIRST

Modul NO_CHANGE
În modul NO_CHANGE, registrele de ieşire sunt blocate şi rămân neschimbate în timpul unei operaţii simultane de scriere, aşa cum apare în Figura 3.12. Acest comportament imită cel al unei memorii sincron simple când zona de memorie este citită sau scrisă în timpul unui ciclu de ceas.

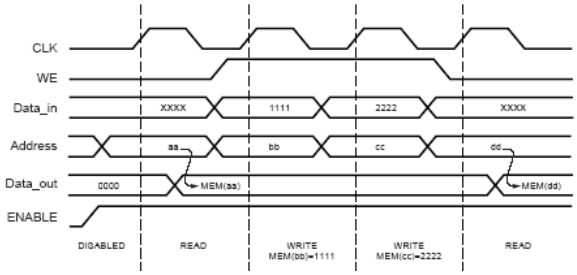


Figura 3.12 Ciclul de scriere în modul NO_CHANGE

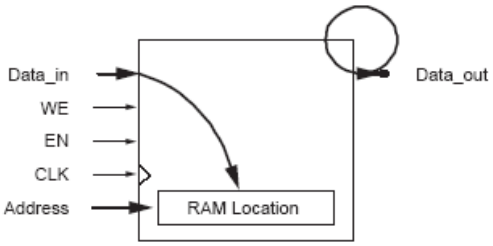


Figura 3.13 Fluxul de date în modul NO_CHANGE

Modul NO_CHANGE este util în cazul mai multor aplicaţii, incluzând acelea la care blocul RAM conţine forme de unde, tabele de funcţii, coeficienţi, etc. Memoria poate fi actualizată fără afectarea ieşirilor memoriei.
Fluxul de date în modul NO_CHANGE arată că ieşirea de date reţine ultima dată citită dacă există o operaţie simultană de scriere la acelaşi port (Xilinx, XAPP463, 2005).

3.1.3.2. RAM distribuit în CLB

Adiţional faţă de Block RAM de 18Kbit FPGA-ul Spartan-3 conţine blocuri de memorie distribuită în fiecare bloc logic configurabil (CLB). Fiecare generator de funcţie SLICEM sau LUT din resursele CLB poate să implementeze opţional un 16x1bit RAM sincron. Tabelele de căutare din SLICEL nu conţin RAM distribuit.

Scrierea datelor în memoriile RAM distribuite se realizează în mod sincron, iar citirea asincron. Fiecare 16x1 bit RAM se poate conecta în cascadă pentru a obţine memorii cu o magistrală mai lată sau îngustă de adrese şi/sau date prin resursele logice aferente cu o penalizare minimă de timp. Blocurile logice configurabile Spartan-3 suportă o varietate largă de primitive până la 64 biţi de adresare x 1 bit lăţime linie de date.

Fluxul de date Single-Port şi Dual-Port RAM distribuite

Memoriile RAM distribuite susţin următoarele tipuri de configurare:

- RAM cu port simplu cu înscriere sincron şi citire asincron. Şi citirea sincron este permisă prin utilizarea bistabilelor asociate memoriei RAM distribuite.
- Dual-port RAM cu o scriere sincron şi două operaţii de citire a porturilor în mod asincron. De asemenea este posibilă citirea sincron.

Cum este ilustrat în Figura 3.14, memoria RAM distribuită conţine un port pentru scriere şi citire şi un port independent pentru citire. Orice operaţie de scriere prin intrarea D sau de citire prin ieşirea SPO poate să apară simultan şi independent de o operaţie de citire din portul secund DPO.

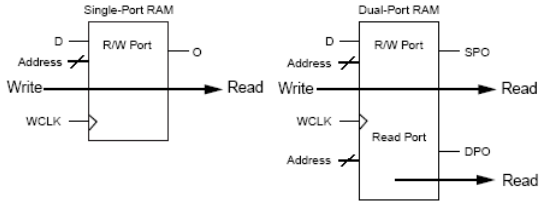


Figura 3.14 Fluxul de date Single-Port şi Dual-Port RAM distribuit

Operaţii de citire/scriere

Operaţia de scriere se realizează pe frontul semnalului de tact controlat prin intrarea validare scriere WE. Din oficiu WE este activ pe '1', dar acesta se poate inversa. Când WE este 1, pe frontul semnalului de tact este validată adresa şi datele din intrarea D sunt înscrise în locaţia de memorie selectată. Dacă WE='0' datele nu sunt înscrise în memorie.

Operaţia de citire este pur combinaţional. Portul de adresare, atât pentru memoria cu port simplu sau port dublu, este asincron cu un timp de acces echivalent cu întârzierile logicii tabelei de căutare LUT.

Citire în timpul scrierii. Când sunt înscrise date noi, ieşirea reflectă datele care au fost înscrise în celula de memorie adresată. În Figura 3.15 este prezentată diagrama de timp, care ilustrează o operaţie de scriere cu datele anterior citite prin portul de ieşire, urmate de datele noi după apariţia frontului crescător pe semnalul de tact.

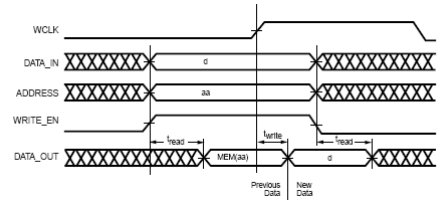


Figura 3.15 Diagrama de timp a unei operaţii de scriere

Caracteristici

A. operaţia de scriere necesită numai un singur front de semnal de tact

B. durata operaţiei de citire depinde numai de timpul de accesare a elementelor logice

C. ieşirile sunt asincron şi depind numai de întârzierea logicii tabelelor de căutare

D. intrările de adrese şi date sunt activate pe frontul pozitiv al semnalului de tact. Nu necesită timp de menţinere.

E. pentru RAM cu port dual, portul A[#:0] este adresă pentru scriere citire, şi DPRA[#:0] este adresă independentă pentru citire

Spartan-3 suportă următoarele primitive prezentate în tabelul Tabelul 6.

Tabelul 6 Primitive pentru RAM distribuit cu Port Simplu şi Port Dual

Primitive	Dimensiune memorie	Tip	Semnale de adrese
RAM16x1S	16x1	Port Simplu	A3,A2,A1,A0
RAM32x1S	32x1	Port Simplu	A4,A3,A2,A1,A0
RAM64x1S	64x1	Port Simplu	A5,A4,A3,A2,A1,A0
RAM16x1D	16x1	Port Dual	A3,A2,A1,A0

Datele de intrare ieşire sunt de un bit. Mai multe memorii RAM distribuite conectate în paralel permit implementarea unor elemente cu funcţie de memorie cu un număr mai mare de biţi de date.

Semnalele de port. Figura 3.16 reprezintă primitivele pentru memoriile RAM distribuite cu port simplu sau port dual.

Semnalele pentru controlul memoriilor distribuite. Ambele porturi de RAM distribuit operează independent unul faţă de celălalt în timp ce citesc acelaşi set de celule de memorie.

Clock — WCLK Semnalul de tact WCLK este utilizat pentru scriere sincron. Liniile de adrese şi de date au timpii de setare specificaţi faţă de semnalul WCLK.

Validare scriere — WE. Semnalul WE validează funcţionalitatea de scriere a portului. Un semnal inactiv de WE previne orice operaţie de scriere a datelor în celulele memoriei.

Un semnal activ WE=’1’ permite ca apariţia unui front pozitiv pe semnalul de tact să înscrie semnalele de date în locaţia de memorie specificată de semnalele de adrese**.**

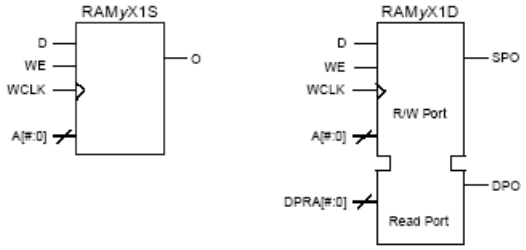


Figura 3.16 Semnalele de control la RAM distribuit

Address — A0, A1, A2, A3 (A4, A5). Liniile de adrese selectează celulele de memorie pentru scriere sau citire. Lăţimea portului determină numărul liniilor de adrese.

Date de intrare — D intrarea de date furnizează valorile noilor date care să fie înscrise în RAM

Date de ieşire — O, SPO, şi DPO. Ieşirea de date O în cazul memoriei RAM distribuite cu port simplu, sau SPO şi DPO în cazul memoriei RAM cu port dual reflectă conţinutul celulelor de memorie selectate prin liniile de adrese. După un semnal de tact activ de scriere, liniile de date de ieşire (O sau SPO) reflectă datele noi care au fost înscrise (Xilinx, XAPP464, 2005).

3.2. Implementări RNA pe FPGA

Progresul în curs al microelectronicii este forţa motrice a dezvoltării continue a produselor tehnice noi. Sistemele FPGA sunt inovaţii de acest fel în sistemele microelectronice pentru aplicaţiile computerizate.

Creşterea continuă în densitate a sistemelor FPGA a făcut posibilă realizarea unor proiecte de sisteme on-chip cu o complexitate mai mare de un milion de porţi şi RAM intern. De aceea, sistemele FPGA s-au dovedit în prezent a fi o platformă hardware atractivă pentru algoritmii RNA care necesită mult spaţiu.

Un alt avantaj al acestui dispozitiv este capacitatea de a combina programabilitatea cu viteza crescută a operaţiilor asociate cu soluţiile hardware paralele (Lysaght, Stockwood, Law, & Girma, 1994).

3.2.1. Moduri de cuplare şi configurări pentru FPGA

Sistemele reconfigurabile sunt de obicei formate din combinarea sistemelor logice reconfigurabile şi a unui microprocesor de uz general. Procesorul efectuează operaţiile care nu pot fi realizate eficient în logica reconfigurabilă, ca de exemplu comenzi în bucle sau în derivaţii, accesări ale memoriei.

Sunt trei moduri diferite de abordare pentru ca un FPGA să funcţioneze în combinaţie cu procesorul gazdă. În prima abordare, o unitate reconfigurabilă poate fi utilizată ca un coprocesor. În acest caz coprocesorul execută calculele fără a apela la interogarea procesorului gazdă. Procesorul gazdă iniţializează hardware-ul reconfigurabil şi trimite datele necesare pentru calcule şi coprocesorul prelucrează independent aceste date (Compton & Hauck, 2000). O astfel de integrare permite, în majoritatea cazurilor, atât pentru procesorul gazdă, cât şi pentru hardware-ul reconfigurabil, să funcţioneze simultan datorită nesuprapunerii comunicării.

În a doua abordare, o unitate reconfigurabilă poate fi utilizată ca o unitate de procesare ataşată. Aceasta este o configuraţie mai puţin strânsă în comparaţie cu prima abordare. Memoria cache a procesorului gazdă nu este vizibilă de către unitatea reconfigurabilă ataşată. De aceea există o întârziere mai mare între comunicarea dintre procesorul gazdă şi hardware-ul reconfigurabil, între datele de intrare şi rezultate (Compton & Hauck, 2000).

Totuşi această configuraţie permite o mai mare independenţă pentru calcule, deoarece se pot transmite mai multe operaţii CPU către hardware-ul ataşat.

A treia abordare, caracterizată de conexiunea cea mai slabă, este un hardware extern, reconfigurabil, de sine stătător. Există o comunicare rară între hardware şi procesorul gazdă.

Fiecare mod de abordare are avantajele şi dezavantajele lui. Cu cât este mai strânsă integrarea hardware-ului reconfigurabil cu procesorul gazdă, cu atât poate fi mai mult utilizat în comunicări mai puţin suprapuse. Cu toate acestea nu poate opera în mod independent o perioadă îndelungată de timp fără întrerupere de către procesorul gazdă. Şi la acest tip de abordare suprafaţa de logică reconfigurabilă este destul de mică. Cu cât este mai liberă cuplarea, cu atât permite mai multe paralelisme în executarea programelor, suferă de suprapunerea comunicării (Compton & Hauck, 2000). Această formă de conexiune se potriveşte pentru aplicaţiile la care o parte a calculului poate fi efectuată cu un hardware reconfigurabil o perioadă de timp destul de îndelungată, fără prea multă comunicare cu procesorul gazdă. Într-un sistem în care există o cuplare strânsă între gazdă şi hardware-ul reconfigurabil, este extrem de necesar ca sarcinile de efectuat să fie împărţite între software şi hardware. Implementarea acestei metode cade în sarcina unui calcul proiectat atât pentru hardware, cât şi pentru software.
În chipurile FPGA există două moduri de implementare a logicii reconfigurabile. Sunt binecunoscute CTR (reconfigurarea compilată în timp) şi RTR (reconfigurarea în timpul utilizării). CTR este o strategie de implementare statică, la care fiecare aplicare constă într-o singură configurare. RTR este o strategie de implementare dinamică unde fiecare aplicaţie constă în multiple configuraţii cooperative (Dehon, 2000).

În CTR, o dată ce o operaţie începe, configuraţia nu se mai schimbă în timpul procesului. Această abordare este asemănătoare cu implementarea unei aplicaţii în ASIC, în timp ce RTR utilizează o schemă de alocare dinamică care realocă partea de hardware în timpul executării aplicaţiei (Hutchings & Writhlin, 1995). Fiecare aplicaţie constă în mai multe configuraţii pentru FPGA.

Aceste configuraţii sunt încărcate succesiv în timpul derulării unei aplicaţii pe chipul FPGA. Acest lucru înseamnă realizarea unui nou hardware pentru fiecare configuraţie pentru porţiuni particulare ale aplicaţiei. Există două modele RTR: globală şi locală. RTR global alocă întreaga resursă a întregului chip FPGA pentru fiecare pas al configuraţiei. Aceasta duce la o nouă realizare pe FPGA în timpul fiecărei configuraţii în timpul utilizării. Proiectantul trebuie să implementeze o aplicaţie în RTR global şi să împartă aplicaţia în părţi aproximativ egale pentru a utiliza în mod eficient resursele FPGA (Hutchings & Writhlin, 1995). Aceasta se numeşte partiţionare temporală a unei aplicaţii. În RTR local, o aplicaţie reconfigurează local nişte subseturi ale logicii în timpul executării aplicaţiei. Poate configura orice procent din resursele reconfigurabile în orice moment. Organizarea aplicaţiilor în RTR local se bazează mai mult pe o divizare funcţională a muncii decât pe partiţionarea utilizată de aplicaţiile RTR global (Dehon, 2000). Odată ce partiţionarea temporală manuală este decisă, poate fi foarte uşor implementată în varianta RTR global. Implementarea RTR local nu este posibilă în instrumente CAD.

3.2.2. Maparea algoritmilor de reţele neuronale artificiale pe sisteme FPGA

Eldredge (Eldredge, 1994) a implementat cu succes algoritmul backpropagation folosind o placă proprie construită din FPGAuri Xilinx XC3090, denumit Run-Time Reconfiguration Artificial Neural Network (RRANN).

Influenţat de arhitectura RRANN a lui Eldredge, Beuchat et al. (Beuchat, Haenni, & Sanhez, 1998) au elaborat o placă FPGA, denumită RENCO (Reconfigurable Network Computer). Sistemul RENCO conţine patru FPGA-uri Altera FLEX 10K130, care se pot reconfigura şi monitoriza printr-o conexiune LAN (ex. Internet sau altul) printr-o interfaţă 10Base-T. RENCO a fost destinat recunoaşterii caracterelor scrise de mână.

Ferucci şi Martin (Ferrucci, 1994) (Marcelo & Martin, 1994) au construit o placă proprie, denumită „Adaptive Connectionist Model Emulator" (ACME) care constă din FPGAuri Xilinx XC4010 multiple. ACME a fost validat cu succes prin implementarea unei reţele cu 3 intrări, 3 unităţi ascunse şi 1 ieşire pentru rezolvarea problemei XOR cu 2 intrări. Skrbek a folosit această problemă pentru a demonstra că propria sa placă FPGA bazată pe backpropagation funcţionează (Skrbek, 1999). Placa FPGA a lui Skrbek, denumită cardul ECX, poate implementa de asemenea reţele neuronale cu funcţii de bază radiale, şi a fost validată prin folosirea unor aplicaţii de recunoaştere a modelelor cum ar fi problema parităţii, recunoaşterea cifrelor, recunoaşterea semnalelor sonare.

GANGLION-ul este o aplicaţie dezvoltată de IBM Research Division, care se bazează pe circuite FPGA programabile de serie Xilinx XC3000. Sistemul se compune din cinci asemenea elemente programabile, care au fost aşezate pe o placă VME. Reţeaua obţinută este un MLP cu 12 intrări, 14 neuroni ascunşi, 4 ieşiri. Aplicaţia este integral paralelă, operează pe date de intrare de 8 biţi şi ponderi de 8 biţi. Astfel puterea atinsă este de 4,48 GCPS.

În ceea ce priveşte funcţiunile sale, un RNA se poate evidenţia prin asociere, clasificare şi aproximare. Autorii (Poormann, Witkowski, Kalte, & Ruckert, 2002) au implementat pe sisteme FPGA un algoritm pentru fiecare funcţie. Pentru acest scop a fost folosit un accelerator hardware reconfigurabil în mod dinamic, RAPTOR2000 care este compus dintr-o placă de bază şi până la şase module specifice aplicaţiei. Placa de bază asigură infrastructura de comunicare pentru modulele găzduite prin magistrala PCI. Cele şase module sunt conectate în inel. Fiecare modul se compune dintr-un chip Xilinx Virtex XCV1000 FPGA şi 256 Mbyte SDRAM. Toate modulele sunt conectate la o magistrală locală comună pentru a comunica cu alte dispozitive sau module şi pentru comunicarea cu gazda printr-o punte PCI.
Puntea de magistrală PCI poate să funcţioneze în mod slave serial în care sistemul gazdă configurează iniţial modulele prin transferarea şirurilor de biţi în FPGA. Se poate folosi o magistrală broadcast în plus pentru comunicarea simultană dintre module.

Pentru accesarea rapidă a memoriei sistemului gazdă, se foloseşte un SRAM cu poartă dublă, care poate fi accesată de fiecare modul. Reconfigurarea modulului pe RAPTOR2000 este iniţiată de calculatorul gazdă. Se susţine că RAPTOR2000 are un timp de reconfigurare de 20 ms pentru fiecare modul. S-au ales reţelele SOM (hartă Kohonen) pentru sarcina de clasificare, NAM (memorie neurală asociativă) pentru problema de asociere şi RBF pentru aproximarea funcţiei la implementarea sistemului RAPTOR2000. Pentru implementarea SOM se folosesc cinci module. Patru module sunt utilizate la implementarea matricei PE, în timp ce cel de-al cincilea modul este folosit ca şi controler de matrice.
SRAM cu port dublu se utilizează pentru stocarea vectorilor de intrare. Folosind dispozitive XCV1000 se poate ajunge la 64 elemente de procesare pe un singur modul.

În acest caz (Poormann, Witkowski, Kalte, & Ruckert, 2002) a atins o performanţă de 11,3 GCPS ceea ce este mult mai bun decât cele 80 MCPS atinse cu un calculator personal (AMD athlon, 800 MHz). Poormann (Poormann, Witkowski, Kalte, & Ruckert, 2002) a implementat de asemenea algoritmul SOM pe sistemul RAPTOR2000, folosind diferite dispozitive Xilinx Virtex.
Operaţiile aritmetice se realizează cu reprezentare în virgulă fixă cu o precizie de 16 biţi. Pentru implementarea NAM, sistemul RAPTOR2000 este echipat cu şase module. În acest caz, cu fiecare modul se pot realiza 512 neuroni.

Unităţile neurale se compun dintr-o unitate de control, o unitate de prelucrare neurală, codificator şi decodor de priorităţi şi 128 Mbyte SRAM. Sinteza rezultată a arătat o folosire a spaţiului de 3200 CLB pentru 512 neuroni. Neuronii din această implementare pot să funcţioneze la 50 MHZ şi necesită 5,4 μs pentru o singură asociere.

A existat o implementare FPGA inovativă bazată pe modelul reţelei neurale care îşi schimbă dimensiunile în mod dinamic, numită FAST (topologie de mărime flexibil adaptabilă) (Pérez-Uribe & Sanchez, 1996).
Cu cele mai multe modele de reţele neurale principalele probleme care se ivesc, sunt: determinarea numărului straturilor, a numărului neuronilor dintr-un strat şi a faptului cum ele se interconectează (Pérez-Uribe & Sanchez, 1996).
Reţelele neurale ontogenice au fost create cu scopul de a înlătura aceste probleme prin oferirea posibilităţii de a schimba în mod dinamic topologia. ART (teoria rezonanţei adaptabile) şi GAR (creştere şi reprezentare) sunt reţele neurale de acest tip, iar FAST derivă din aceste concepte.

Astfel, în lucrarea (Pérez-Uribe & Sanchez, FPGA Implementation of an Adaptable Size Neural Network, 1996) s-a implementat o reţea numită reţea neuronală ontogenică pe o placă FPGA proprie, denumită Flexible Adaptable-Size Topology (FAST). FAST a fost folosit pentru a implementa trei tipuri diferite de reţele neuronale nesupravegheate, ontogenice- adaptive resonance theory (ART), adaptive heuristic critic (AHC) şi Dyna-SARSA.

Primele implementări FAST au folosit reţele neuronale bazate pe ART. Când se foloseşte la o problemă de segmentare a culorilor de imagini, patru neuroni FAST au segmentat cu succes o imagine 294x353, de 61 culori înfăţişând Floarea soarelui de Van Gogh în patru clase de culori.

A doua implementare FAST a folosit o reţea neuronală bazată pe AHC (Pérez-Uribe & Sanchez, 1997). În această implementare, denumită FAST-AHC, s-au folosit opt neuroni pentru a controla un

pendul invers. Problema pendulului invers este exemplul clasic pentru un sistem instabil, folosit la testarea unor abordări ale controlului antrenării (Pérez-Uribe, 1999). FAST-AHC nu a putut să generalizeze la fel de bine ca şi algoritmul de propagare înapoi a erorii, dar antrenarea este mai rapidă şi mai eficientă. Acest lucru se datorează faptului că tehnica de antrenare AHC se poate generaliza ca şi o formă a antrenării locale, unde doar nodurile active ale reţelei neuronale sunt actualizate, contrar algoritmului backpropagation, unde antrenarea este globală.

A treia implementare FAST a folosit a reţea neuronală Dyna-SARSA (Pérez-Uribe, 1999). Dyna-SARSA constituie un alt tip de antrenare de întărire, şi a fost mult mai puţin intensivă din punct de vedere computaţional, în comparaţie cu AHC, şi foarte potrivită pentru implementarea digitală. Placa FAST Dyna-SARSA a fost integrată pe un robot mobil de sine stătător, şi folosită ca şi un neurocontroller pentru a demonstra o sarcină de antrenare-navigare. Neurocontrollerul FAST Dyna-SARSA a fost o încercare reuşită, ajutând robotul mobil să evite obstacolele, care s-au adaptat la schimbări uşoare ale poziţiei obstacolelor.

Arhitectura FAST a fost prima din categoria sa, care foloseşte reţele neuronale cu antrenare nesupravegheate, ontogenice, dar se poate spune că arhitectura FAST are limitările sale, deoarece poate să rezolve doar probleme simple care necesită categorizare dinamică sau clasificare on-line.

Reţeaua FAST se compune din două straturi feed-forward conectate total şi antrenarea este nesupervizată. Această reţea este potrivită pentru gruparea sau clasificarea datelor de intrare.

Se adaugă un neuron în stratul de ieşire dacă se găsesc vectori de intrare suficient de diferiţi. Există o fază de curăţire în care, în funcţie de suprapunerea regiunilor sensibile ale unor neuroni învecinaţi, un neuron din stratul de ieşire este şters.

Reţeaua neurală FAST se compune dintr-un microcontroler 68331 şi patru chipuri FPGA Xilinx XC4013. Pe aceste chipuri se implementează neuroni FAST, registre de mapare, generator de secvenţe, şi I/O.

Neuronul FAST se compune din trei blocuri diferite care execută câte un algoritm: calculul distanţei, antrenare şi curăţire. Fiecare neuron include nouă sumatori de 8 biţi şi un singur multiplicator shift-add de 8 biţi. Conţine de asemenea un generator de numere aleatoare pentru procesul de eliminare. Generatorul de secvenţe este un automat cu număr finit de stări care controlează executarea algoritmilor din cele trei stadii diferite.

Modelul de reţea FAST este aplicat la o problemă de învăţare şi recunoaştere a culorilor la imagini digitale. Aceasta necesită gruparea pixelilor de imagine după proprietăţile de asemănare cromatică. În acest experiment, coordonatele de pixeli ale unei imagini sunt prezentate reţelei în mod aleator, din care rezultă segregarea imaginii în patru categorii, câte unul pentru fiecare neuron de ieşire. Se poate observa că fiecare vector de intrare se poate clasifica în 8 μs. Rezultatul obţinut cu implementarea hardware seamănă foarte mult cu cel obţinut prin simularea software.

Ca o aplicaţie, problema cu N regine cu reţea Hopfield pe FPGA apare în lucrarea (Abramson, Smith, Logothetis, & Duke, 1998), în care este menţionat faptul că implementarea reţelei Hopfield pe FPGA pentru rezolvarea acestei probleme diferă de alte implementări în mai multe aspecte.

În primul rând, ponderile sunt mici şi se pot reprezenta folosind numere întregi. Astfel se reduce semnificativ dimensiunea unităţilor aritmetice. În al doilea rând, valorile ponderilor neurale se reduc la 0 sau 1. Acest lucru înlătură necesitatea unităţilor de multiplicare pentru că produsul vectorial devine simplă operaţie logică ŞI.

Arhitectura acestei implementări constă din 16 XC4010 conectate de 4 comutatori programabili (FPIC) pe o placă reconfigurabilă Aptix AP4. Folosind această configuraţie, este posibilă încărcarea unui proiect de până la 160000 porţi. Neuronii şi interconexiunile lor sunt specificaţi în VHDL, care este sintetizat în Exempler Logic's Galileo System Tools.

Performanţa acestui sistem se compară cu alte două simulări software în „C” pentru problema celor 4, 6 şi 8 regine. Rezultatul indică faptul că este posibilă atingerea unei creşteri în viteză a soluţionării problemei de 2 sau 3 ori prin realizările hardware.

A existat de asemenea o încercare de a implementa o reţea neurală probabilistică pe sisteme FPGA pentru probleme de clasificare. Clasificarea automată a imaginilor multispectru în spaţiu necesită un proces îndelungat din punctul de vedere al calculelor. Imaginile multispectru se obţin prin sateliţi sofisticaţi ale Sistemului de Observare a Pământului (Earth Observing System (EOS)) lansaţi de NASA pentru studierea mediului înconjurător. Unul din scopurile clasificării acestor imagini este divizarea solului pământesc în diferite categorii cum ar fi: urban, agricol, silvic, neproductiv etc.

În articolul (Abramson, Smith, Logothetis, & Duke, 1998) s-a implementat o reţea neurală probabilistă (PNN) pe FPGA pentru clasificarea imaginilor multi-spectru obţinute de satelitul LANDSAT-2 EOS. Imaginile multi-spectru sunt formate de scanere şi reprezintă seturi de imagini, fiecare corespunzând unei bande spectrale. Algoritmul de clasificare PNN implică calcularea probabilităţii pentru fiecare pixel pentru a-l încadra în una dintre clase. Ecuaţia pentru verificarea probabilităţii necesită calcule complexe de scădere, înmulţire şi calcul exponenţial a vectorilor de imagine.

Arhitectura clasificatorului PNN constă din două sisteme FPGA XC 4013E (numindu-le XFPGA şi YFPGA), fiecare cu câte 13000 porţi pe o placă acceleratoare X213. Datorită numărului limitat de porţi, pentru efectuarea calculelor se foloseşte aritmetica în virgulă fixă. Lăţimea căii de date în virgulă fixă este determinată prin simularea unor variabile cu operaţii pe bit în limbajul de programare C. În acest fel, valorile ponderilor de 10 biţi sunt satisfăcătoare. Memoria ponderilor este realizată pe SRAM-ul de pe YFPGA.

Modulul YFPGA este compus din unitatea de scădere, unitatea de ridicare la putere şi unitatea de acumulare. Unitatea XFPGA constă dintr-o unitate exponenţială, o unitate de înmulţire şi o unitate de acumulare a claselor.

Datorită lipsei de spaţiu pe XFPGA, o unitate de comparaţie a claselor a fost mutată pe calculatorul gazdă. Este folosit un tabel de căutare pentru calcularea exponenţialei negative. Ambele sisteme FPGA pot fi interfaţate cu calculatorul gazdă printr-o magistrală PCI. Astfel calculatorul gazdă este capabil să configureze datele iniţiale. Performanţa acestui sistem este comparată cu două simulaţii software efectuate pe sisteme DEC şi Pentium.

Timpul necesar pentru simularea algoritmului scris în „C” pe DEC rulând la 200 MHz şi pe Pentium la 166 MHz s-a dovedit a fi de 22 respectiv 30 minute. Utilizând o placă acceleratoare X213 cu două dispozitive FPGA a redus timpul necesar implementării la 77 secunde, ducând la o accelerare mai mare cu un ordin de mărime.

De Garis et al (de Garis, Gers, & Korkin, 1997) (de Garis & Korkin, 2002) au implementat o reţea neuronală evoluţionară bazată pe tehnici evoluţionare, şi au reuşit să obţină o antrenare on-chip. De Garis a proiectat o placă bazată pe FPGA, denumită CAM-Brain Machine (CBM) unde se foloseşte un algoritm genetic (GA) pentru a antrena o reţea neuronală bazată pe un automat celular (cellular automata, CA) . Chiar dacă CBM este privit ca o reţea care permite antrenare on-chip, nu a fost inclus nici un algoritm de antrenare în CA. În locul acesteia, antrenarea localizată se realizează în mod indirect printr-un algoritm genetic (în acest caz acesta iniţializează datele de configurare pentru fiecare automat celular, care stabileşte cum va creşte reţeaua), urmat de „creşterea” topologiei reţelei neuronale, care este o caracteristică funcţională a automatului celular.

CBM susţine până la 75 milioane de neuroni, fiind cea mai mare reţea neuronală evolutivă a timpului, unde mii de neuroni evoluează în fiecare secundă. CBM-ul s-a dovedit a fi un succes în aplicaţii precum aproximarea de funcţii. Scopul pe termen lung al lui de Garis este folosirea CBM-ului la crearea unor reţele neuronale modulare foarte rapide, la scară largă, care se pot folosi la aplicaţii de construcţie a creierului. De exemplu, de Garis plănuieşte folosirea CBM-ului ca şi un neurocontroller în reglarea în timp real al unei pisici-robot de mărime naturală denumit „Robokitty”.

Susţinerea reţelelor neuronale modulare prin CBM este oarecum limitată, deoarece interacţiunea în aceste module sau conexiunile intermodulare trebuie definite manual, off-line.

Nordstorm (Nordstrom, 1995) a încercat să conceapă o reţea neuronală modulară pe o placă bazată pe FPGA, denumit REMAP (Real-time, Embedded, Modular, Adaptive, Parallel Processor). Nordstorm a considerat că calculul reconfigurabil se poate folosi ca şi o bază potrivită pentru a susţine cu uşurinţă diferite tipuri de module (diferiţi algoritmi de reţele neuronale). Acest tip de mediu se poate folosi în crearea unor reţele neuronale modulare eterogene, cum ar fi „ierarhia experţilor” („hierarchy of experts”) propus de Jordan şi Jacobs.

Cu excepţia CAM-Brain Machine al lui de Garis, observaţiile lui Nordstorm cu privire la acest domeniu rămân valabile chiar până astăzi. Dat fiind faptul că densitatea FPGA-urilor aflate la dispoziţia lui Nordstorm la timpul cercetării sale era limitată, el a reuşit să implementeze doar aplicaţii de modul singular pe REMAP. Încazul ideal, Nordstorm ar fi dorit să susţină folosirea reţelelor neuronale modulare pe REMAP, însă acest scop a rămas neatins.

Diferenţa dintre REMAP-α şi REMAP-β este mărimea densităţii FPGA-ului folosit. La REMAP-α s-a folosit FPGA Xilinx XC3090 pentru prototipizarea diferiţilor algoritmi neuronali, în timp ce la REMAP-β s-a folosit iniţial Xilinx XC4005, dar care a fost înlocuit cu Xilinx XC4025. REMAP-β

s-a folosit ca şi emulator hard de reţele neuronale şi instrument de antrenare, care a putut să efectueze implementarea a următoarelor tipuri de reţele neuronale: reţele Adaline şi Madaline, algoritmul backpropagation, reţele de memorie asociativă bidirecţională, memoria asociativă Hopfield, reţele counterpropagation, hartă autoorganizantă (SOM), teoria rezonanţei adaptive (ART).

Pentru a creşte viteza de procesare, s-a construit un al treilea şi ultim prototip, REMAP-γ, prin folosirea unei abordări bazate pe ASIC.

În următoarele paragrafe sunt prezentate două lucrări recente (Pandya, 2005) (Nichols, 2003) cu privire la posibilităţile de implementare a reţelelor neuronale artificiale.

Reţelele neuronale artificiale, şi algoritmul backpropagation în special, reprezintă o formă a inteligenţei artificiale care este caracterizată de antrenarea lentă şi lipsa unei metodologii clare pentru determinarea topologiei reţelei înainte de începerea antrenării. Cercetătorii anteriori au folosit calculul reconfigurabil ca o metodă a accelerării testării reţelelor neuronale artificiale. În lucrarea (Nichols, 2003) este prezentat modul în care au ajutat avansarea în domeniu îmbunătăţirile recente în instrumentele şi metodologiile folosite la calculul reconfigurabil, şi au susţinut astfel aplicabilitatea lor la accelerarea RNA-urilor. S-a creat o arhitectură RNA nouă, bazată pe FPGA, denumită RTR-MANN, pentru a demonstra performanţele mai bune obţinute prin folosirea instrumentelor şi metodologiilor de generaţie curentă. RTR-MANN are o scalabilitate şi o densitate funcţională de un ordin mai mare decât arhitecturile RNA bazate pe FPGA mai vechi. În plus, folosirea unei metodologii de proiectare a sistemelor noi (limbaj de nivel înalt) a condus la o fază de verificare/validare mult mai intuitivă, care a fost cu un ordin de magnitudine mai rapid decât simulatorii HDL tradiţionali.

Contribuţiile pentru a atinge obiectivele lucrării au fost următoarele:

Analiza şi concluzia faptului că folosirea unei aritmetici în virgulă flotantă de 32 biţi nu este atât de fezabilă ca şi o aritmetică în virgulă fixă de 16 biţi, în cazul proiectelor FPGA de generaţie curentă. Această concluzie nu reprezintă o noutate în domeniu, dar este o concluzie actualizată în ceea ce priveşte FPGA-urile de generaţie curentă.

Aritmetica utilizată s-a proiectat în mod special pentru a fi folosit în RTR-MANN, care a conţinut operatori aritmetici în virgulă fixă de 16 biţi, care au fost optimizate din punctul de vedere al suprafeţei pentru Xilinx XCV2000E Virtex-E FPGA. Această bibliotecă a ajutat la obţinerea unei scalabilităţi şi densităţi funcţionale mult mai bune pentru RTR-MANN.

Au apărut alte probleme în cazul simulatorilor HDL de generaţie curentă. În special, timpii de simulare ai simulatorilor HDL de generaţie curentă s-au dovedit a fi foarte lungi (de ordinul zilelor sau săptămânilor în cazul proiectelor VLSI), ceea ce a dus la faze de verificare/validare anevoioase.

Algoritmul backpropagation (BP), folosit la construirea reţelelor neuronale artificiale a devenit foarte populară de la apariţia sa la sfârşitul anilor 1980. Structura regulară şi lărgimea domeniului de aplicabilitate a algoritmului BP a dobândit interesul cercetătorilor în tentativele lor de a obţine implementări eficiente din punctul de vedere al utilizării timpului. Procesoarele de utilizare generală (General Purpose Processors, GPP) şi circuitele integrate specifice aplicaţiei (ASIC) sunt exemple relevante pentru RNA-uri bazate pe algoritmul BP. Totuşi, aceste dispozitive de calcul suferă în mod constant de căutarea echilibrului între flexibilitate şi performanţă. În ultima decadă s-au obţinut progrese semnificative în dezvoltarea unui anumit tip de hardware, o platformă reconfigurabilă bazată pe FPGA. FPGA-urile au o flexibilitate excelentă în termeni de reprogramabilitate ai aceluiaşi hardware, şi, în acelaşi timp, ating performanţe foarte bune în termeni de calcul paralel.

Cercetarea descrisă în lucrarea (Nichols, 2003) propune arhitecturi parţial paralele şi o arhitectură total paralelă pentru realizarea algoritmului BP pe un FPGA. Proiectele propuse sunt codificate în Handel-C şi verificate din punctul de vedere al funcţionalităţii prin sintetizarea sistemului pe un chip FPGA Virtex2000e. Validarea proiectelor se realizează după diferite considerente. Arhitecturile parţial paralele şi varianta total paralelă se dovedesc a fi de 2,5, respectiv 4 ori mai rapide decât implementările soft.

Într-o altă lucrare sunt propuse mai multe arhitecturi ANN pentru implementarea algoritmului BP pe FPGA Virtex2000e. Pe parcursul cercetărilor efectuate, s-au realizat iniţial două arhitecturi în serie. Scopul dezvoltării acestor arhitecturi în serie a fost identificarea problemelor şi stabilirea bazelor dezvoltării arhitecturilor paralele. După aceea, s-au realizat două arhitecturi parţial paralele. Scopul acestor arhitecturi a fost studiul accelerării în timpul acomodării unor reţele de dimensiuni variabile. Unele din problemele întâlnite la aceste arhitecturi parţial paralele au servit ca şi îndrumare pentru dezvoltarea unei arhitecturi total paralele.

Operaţia de multiplicare-acumulare este operaţia de bază necesară executării algoritmului BP. Conceptele de mod Branch-In şi Branch-Out ale operaţiei de multiplicare-adunare au fost introduse în detaliu. Aceste două moduri formează baza dezvoltării paralele. S-au propus şi s-au prezentat următoarele arhitecturi:

- SIPEX şi SIPOB: abordare serială, diferă în stocarea parametrilor iniţiali şi tabele de căutare (LUT) pentru funcţia sigmoid.
- PAROI, PARPP şi PARIO: abordare parţial paralelă, diferă în atribuirea operaţiilor modului Branch-In şi Branch-Out la fazele algoritmului BP.
- FPAR: un proiect total paralel.

Generarea aleatoare a numerelor prin LFSR şi funcţia sigmoid, aproximată liniar prin trei variabile, a fost prezentată în detaliu.

Trei repere XOR, Iris şi Cancer, respectiv mic, mediu, mare, au fost alese pentru validarea arhitecturilor. Contribuţia majoră adusă este elaborarea proiectelor parţial şi total paralele care ating performanţe de 2,5 şi 4 ori mai rapide decât varianta cu implementare software. Implementările parţial paralele încercă să obţină un echilibru între spaţiul de pe chip şi performanţă, atingând totuşi CUPS de $12x10^6$, mai mare decât la oricare dintre arhitecturile prezentate în bibliografie. O altă contribuţie care merită menţionată aici este implementarea cu succes a reperului Cancer care este o problemă reală pentru toate arhitecturile cu excepţia FPAR. Validarea reperului Cancer pentru arhitectura FPAR necesită un număr foarte mare de module aritmetice, ceea ce are ca rezultat un consum de spaţiu foarte mare.

S-a arătat prin calcule pentru un sistem multi-FPGA ipotetic, că FPAR-ul poate să atingă CUPS de $51x10^6$ şi o viteză de 10 ori mai mare decât versiunea software pentru setul de date Cancer. O altă contribuţie importantă este implementarea cu succes a problemelor XOR şi Iris pentru arhitectura FPAR cu necesităţi hardware reduse de 0,4 respectiv 0,6 milioane porturi pe chipul Virtex XCV2000e FPGA. Pe parcursul cercetărilor prezentate în lucrarea (Pandya, 2005), s-a folosit Handel-C ca şi limbaj de descriere hardware. Structura generală Handel-C contribuie la înlesnirea muncii de dezvoltare pentru un inginer hardware începător.

Handel-C permite o simulare rapidă a unor proiecte de dimensiuni mari în comparaţie cu simulatorul VHDL Modelsim. Timpul de dezvoltare este mult mai scurt decât în VHDL. Operaţiile de depanare în Handel-C nu sunt uşor de realizat pentru arhitecturile RNA. S-a folosit metoda reprezentării numerelor în virgulă fixă. Problema cu compilatorul Handel-C este că nu poate să afişeze numerele în virgulă fixă, în schimb afişează valorile decimale pentru partea întreagă şi partea fracţională. Deoarece algoritmul BP necesită operaţii aritmetice elaborate, depanarea parametrilor în numere în virgulă fixă devine un poces foarte complex.

Caracteristica de reconfigurare a unui FPGA în timpul rulării se poate utiliza pentru implementarea unor RNA-uri de dimensiuni mari (Pandya, 2005). Algoritmul BP este foarte potrivit pentru o astfel de reconfigurare deoarece se poate diviza în trei faze distincte în timp. Totuşi, timpul necesar reconfigurării diferitelor faze pe un FPGA poate fi un impediment în asemenea proiecte. În acelaşi timp, faptul că un anumit FPGA poate să aibă caracteristica de reconfigurabilitate sau nu, este determinat de furnizorul plăcii respective. Arhitecturile descrise execută cele trei faze ale algoritmului BP în mod secvenţial.

Chiar dacă paralelismul complet s-a aplicat la nivel de strat, un progres în creşterea vitezei în cazul versiunii soft nu a fost posibilă. Investigaţiile se pot extinde asupra celor trei faze ale metodei pipeline: calculul ieşirii reţelei, propagarea înapoi a erorii şi adaptarea ponderilor. Posibilitatea combinării parametrilor şi a topologiei iniţiale şi descărcarea sa directă în Block RAM-ul unui FPGA trebuie cercetate. Instrumentul sintetic Xilinx asigură asemenea posibilităţi pentru platforme hardware specifice. Acest lucru va elimina necesitatea de a accesa off-chip RAM-ul ceea ce poate să rezulte într-o execuţie mai rapidă a algoritmului. Arhitecturile prezentate în lucrarea (Pandya, 2005) pot utiliza alte limbaje descriptive hardware cum ar fi VHDL şi performanţele lor în termeni de spaţiu şi viteză se pot compara. Viitoarele lucrări bazate pe arhitecturile FPAR se pot realiza pentru implementarea setului de date Cancer pe un hardware multi-FPGA dedicat.

Sistemele multi-FPGA sunt sisteme utilizate în mod frecvent la executarea algoritmului BP. Hardware-ul multi-FPGA nu doar permite implementarea unor reţele de dimensiuni mari, dar atinge de asemenea viteze mari de procesare şi de adaptare. În lucrarea (Pandya, 2005) se propune în locul

reprezentării numerelor în virgulă fixă oferită de compilatorul Handle-C, reprezentarea în virgulă fixă separat şi investigarea criteriului spaţiu-viteză pentru operaţiile aritmetice în virgulă fixă.

Ca şi o concluzie, se poate spune că reţeaua neuronală folosită la implementarea bazată pe FPGA este o caracteristică importantă la clasificarea diferitelor arhitecturi. Tipul reţelei neuronale implementate depinde de aplicaţia care se va folosi la rezolvarea problemei în cauză. Ultimele tendinţe din acest domeniu au arătat că au existat puţine încercări pentru implementarea reţelelor neuronale bazate pe FPGA. Densitatea şi viteza FPGA-urilor a crescut, până la punctul unde este rentabilă susţinerea implementării reţelelor neuronale pe astfel de sisteme.

3.2.3. Implementarea funcţiilor de activare

Există multe tehnici cu privire la evaluarea funcţiilor elementare sau aproximativ elementare cum sunt aproximările polinomiale, algoritmi CORDIC, aproximări raţionale, aproximări pe bază de tabele şi multe altele (Omondi, 1994) (Muller, 1997). Pentru implementarea hardware a funcţiilor de activare, precizia, performanţa şi costul sunt cei mai importanţi factori. Prin ultimii doi se înţelege că majoritatea tehnicilor performante care au fost dezvoltate în analiza numerică, şi sunt uşor de implementat în software, nu sunt potrivite la implementare hardware.

Dezvoltare în serie Taylor:

Acesta se realizează prin aproximarea funcţiei sigmoid utilizând primele 4, 5 termene dintr-o serie Taylor. Publicaţia (Murtagh & Tsoi, 1992) propune următoarea formulă de serie Taylor pentru aproximarea funcţiei sigmoid, care poate fi utilizată în domeniul [-1, 1].

$$y(x) = \frac{1}{2} + \frac{3}{4}x - \frac{1}{4}x^3 \qquad \textit{Ec. 45}$$

În graficele din Figura 3.17 sunt reprezentate funcţia sigmoid

$$y(x) = \frac{1}{1+e^{-ax}} \qquad \textit{Ec. 46}$$

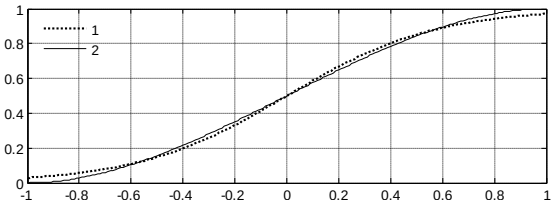


Figura 3.17 Aproximare Taylor a funcţiei sigmoid

a=3.5 respectiv funcţia sigmoid aproximată prin dezvoltare în serie Taylor. Cu linie punctată (1) este reprezentată funcţia sigmoid, iar cu linie continuă (2), aproximarea prin dezvoltare în serie Taylor. Dacă pentru parametrul *a* se ia valoarea 3.5 eroarea de aproximare este 0.0338.

Tabele de căutare (LUT)

Tabelele de căutare este un mod de abordare simplă, valorile funcţiei de activare sunt stocate off-chip sau on-chip în memorii de tip RAM sau Block RAM în FPGA.
Valorile potrivite sunt citite din valorile stocate prin introducerea adresei adecvate, adresă ce poate fi generată de un circuit simplu compus din comparatoare şi registre de deplasare.
Avantajul acestei aproximări este costul redus de logică necesară, iar dezavantajul este suprafaţa mare necesară pentru a realiza LUT-ul în memorie.

Computer digital de rotaţie coordinată (CORDIC)

Schema CORDIC este o metodă iterativă bazată pe operaţii aritmetice şi deplasare la nivel de bit. Necesită suprafaţă de substrat de siliciu mare. Eroarea de aproximare depinde de numărul de biţi utilizaţi pentru codificarea valorii funcţiei de activare.

Aproximarea liniară pe porţiuni (PWL)

Aceasta se realizează prin aproximarea liniară pe domenii a funcţiei de activare. Această aproximare s-a dovedit a fi o alegere bună în ceea ce priveşte cerinţele hardware, respectiv precizia de aproximare.

Aproximările PWL pot fi divizate în aproximări liniare de ordin întâi şi aproximări de ordin superior. Publicaţiile (Alippi & Storti-Gajani, 1991) (Myers & Hutchison, 1989) au fost primele care au prezentat schema pentru aproximarea prin metoda PWL a funcţiei sigmoid. Autorul (Alippi & Storti-

Gajani, 1991) a lucrat foarte mult la simplificarea funcţiei sigmoid pentru a obţine un model realist, capabil să fie implementat în tehnologie VLSI. Publicaţia (Alippi & Storti-Gajani, 1991) descrie aproximarea funcţiei sigmoid cu metoda PWL prin selectarea unui set de valori pentru valorile "x" şi calcularea valorilor "y" ca numere în funcţie de putere a 2. Astfel luăm în considerare un singur segment pentru fiecare interval întreg, funcţia se poate liniariza şi descrie după cum urmează:

$$y(x) = 1 + \frac{1}{2^{|(x)|}}\left(\frac{\hat{x}}{4} - \frac{1}{2}\right) \qquad \textit{Ec. 47}$$

$\hat{x}$: partea zecimală a valorii lui x cu semnul său
|(x)| : partea întreagă a valorii lui x
Cu notaţiile de mai sus, pentru axa negativă:

$$y(x) = 1 + \frac{1}{2^{|(x)|}}\left(\frac{\hat{x}}{4} + \frac{1}{2}\right) \qquad \textit{Ec. 48}$$

În Figura 3.18 este reprezentată funcţia sigmoid (linie punctată 1) şi aproximarea funcţiei prin segmente liniare (linie continuă 2) în intervalul [-4, 4]. Eroarea maximă de aproximare este 0.0189. Versiunea simplificată prezentată anterior poate fi simplu implementată în hardware cu un registru de deplasare controlat de un numărător. Acesta elimină necesitatea unui circuit dedicat de multiplicare. Publicaţia (Alippi & Storti-Gajani, 1991) a derivat formula de mai sus prin descompunerea funcţiei sigmoid în intervalul [-8, 8] în 15 segmente liniare.

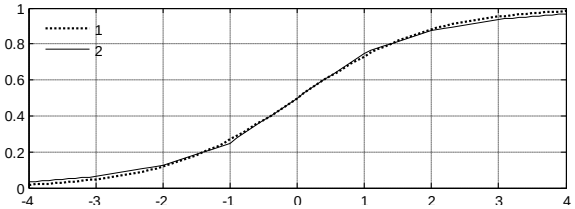


Figura 3.18 Aproximarea liniară pe porţiuni a funcţiei de activare sigmoid

Aproximarea propusă de lucrarea (Myers & Hutchison, 1989) este o curbă modificată care se bazează pe principiul legii A-law pentru sisteme PCM (Pulse Code Modulation). Curba modificată compusă din 7 segmente a fost utilizată la aproximarea funcţiei. Publicaţiile (Alippi & Storti-Gajani, 1991), (Myers & Hutchison, 1989) au prezentat de asemenea formula pentru calculul derivatei funcţiei sigmoid bazată pe segmentarea liniară a curbei.

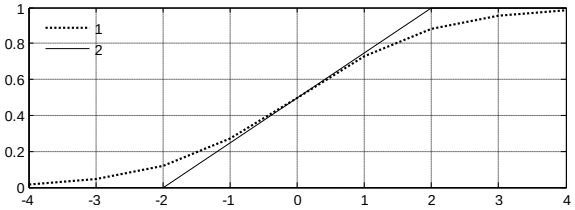


Figura 3.19 Aproximarea funcţiei sigmoid prin metoda CRI

Lucrarea (Basterretxea & Tarela, 2004) a propus o schemă de Aproximare Liniară Centrată (CRI) pentru o implementare optimă a funcţiei sigmoid şi derivata acestuia. CRI este o schemă computaţională secvenţială pentru generarea funcţiei PWL. Autorii susţin că aproximarea PWL bazată pe modelul CRI se poate aplica la aproximarea oricărei funcţii neliniare.

Figura 3.19 prezintă structura minimă iniţială unde metoda CRI este utilizată la aproximarea funcţiei sigmoid. Este o aproximare simplă cu trei segmente, unde două segmente reprezintă saturaţia pentru valori de intrare mici şi mari, în timp ce segmentul intermediar este tangenta funcţiei sigmoid de referinţă în punctul x=0.

Ecuaţiile care descriu aproximarea sunt descrise mai jos.

$$y_1(x) = 0, \quad y_2(x) = \frac{1}{2}\left(1 + \frac{x}{2}\right), \quad y_3(x) = 1 \qquad \textit{Ec. 49}$$

Cum se poate observa şi din ecuaţiile prezentate şi conform Figura 3.19 liniarizarea cu trei segmente este o aproximare dură a funcţiei, deoarece eroarea de aproximare este mare la sfârşitul curbei înainte de saturaţie.

Autorii lucrării (Basterretxea & Tarela, 2004) au observat că numărul segmentelor poate fi crescut şi poate fi calculat pentru fiecare nivel de interpolare astfel:

$$Nr.\deg mente = 2^{q+1} + 1 \qquad \textit{Ec. 50}$$

Numărul de segmente se poate mări prin introducerea de perechi de tangente pe ambele părţi a axei y la diferite valori $|x|$.

Deci numărul de segmente creşte până la 5, 9, 17, 33 pentru q=1, 2, 3 respectiv 4.

În modul de abordare prezentat de lucrarea (Sammut & Jones, 1991) s-a susţinut utilizarea circuitelor de adunare şi înmulţire rapidă pentru aproximarea funcţiei sigmoid prin liniarizare. Curba sigmoid este descompusă în patru segmente, pozitiv saturat, pozitiv nesaturat, negativ nesaturat, şi negativ saturat. Partea neliniară dintre limitele de saturaţie este aproximată prin utilizarea unui polinom pătratic de gradul doi.

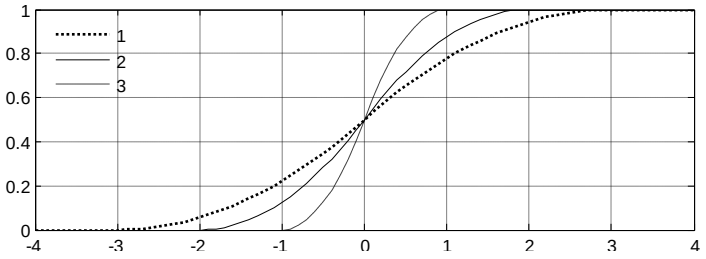


Figura 3.20 Aproximarea funcţiei de activare sigmoid prin polinom pătratic de gradul doi

$$\begin{aligned}
&y(x) = 1 && \textit{dacă} \quad x \geq hl \\
&y(x) = 1 - g(hl - x)^2 && \textit{dacă} \quad x \geq 0 \quad \textit{şi} \quad x < hl \\
&y(x) = 0 + g(hl + x)^2 && \textit{dacă} \quad x < 0 \quad \textit{şi} \quad x \geq ll \\
&y(x) = 0 && \textit{dacă} \quad x < ll
\end{aligned} \qquad \textit{Ec. 51}$$

unde valorile pentru *ll* şi *hl* se pot obţine din limitele de saturaţie a funcţiei sigmoid şi

$$g = \frac{1}{2(hl)^2} \qquad \textit{Ec. 52}$$

Modelul propus de autorul lucrării (Sammut & Jones, 1991) reprezentat în Figura 3.20 este o aproximare de gradul doi a funcţiei sigmoid. În reprezentarea grafică, limitele de saturaţie a curbelor sunt 1, 2 şi 3, corespunzând liniei punctate (1), liniei continue (2), şi liniei întrerupte (3).

Publicaţia (Zhang, Vassilliadis, & Fris, 1996) a propus de asemenea un model care îmbunătăţeşte aproximarea funcţiei în comparaţie cu varianta sa liniară de gradul I.

În particular, reduce eroarea medie şi eroarea maximă a aproximării şi necesită numai o singură operaţie de multiplicare. În acest model, intrările funcţiei sigmoid sunt descompuse pe segmente şi o schemă de aproximare de ordinul II este utilizată pentru calcularea fiecărui segment.

Pentru un segment *[α,β]*, dacă *x* este din intervalul $x \in [\alpha, \beta]$ şi *y(x)* este ieşirea aproximată, în general o aproximare de ordinul doi se poate exprima astfel:

$$y(x) = A + B(x + C)^2 \quad unde \quad |C| = 2^{-n} \qquad \textit{Ec. 53}$$

unde C este o putere a lui 2, multiplicarea cu C se poate calcula prin operaţii de deplasare.

De asemenea este prezentat în detaliu un algoritm pentru definirea valorilor A, B şi C. Autorul a prezentat o formulă de aproximare a funcţiei împărţită pe două segmente (-4,0) şi (0,4) care arată astfel:

$$y(x)=\begin{cases}2^{-1}\left(1-\left|2^{-2}x\right|\right)^2 & -4<x<0\\ 1-2^{-1}\left(1-\left|2^{-2}x\right|\right)^2 & 0\le x<4\end{cases} \qquad \textit{Ec. 54}$$

În modelul de mai sus se poate folosi o reprezentare cu bit de semn sau complement faţă de doi. Pentru reprezentarea numerelor s-a utilizat o reprezentare în virgulă fixă, folosind 4 biţi pentru partea întreagă, un bit pentru semn, zece biţi pentru partea fracţionară. În graficele din Figura 3.21 sunt reprezentate funcţia sigmoid (linia punctată 1) şi aproximarea funcţiei de activare prin două segmente neliniare (linia continuă 2). Eroarea maximă de aproximare pentru funcţiile reprezentate este 0.0180. Eroarea medie în aproximarea funcţiei sigmoid cu această metodă este de ordinul 10^{-3}, mult mai mică comparativ cu metoda de aproximare PWL de gradul întâi, caz în care eroarea medie este de ordinul 10^{-2}.

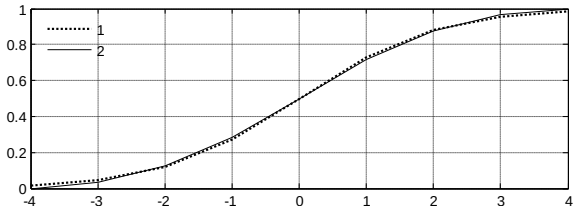


Figura 3.21 Aproximarea funcţiei de activare prin două segmente neliniare

În publicaţia (Beiu, Peperstraete, & Vandewalle, 1994) s-a a subliniat faptul că, chiar dacă există metode de aproximare pentru simplificarea funcţiei sigmoid, calculele implicate sunt totuşi complexe. Alţi autori au introdus o funcţie specială, o funcţie sigmoid particulară. În această lucrare s-a arătat că funcţia sigmoid particulară este echivalentă cu funcţia sigmoid clasică dacă amplificarea este modificată cu o constantă. Funcţia este descrisă astfel:

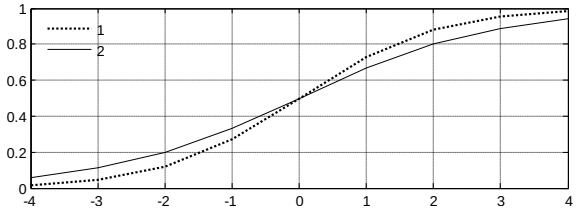


Figura 3.22 Aproximarea funcţiei sigmoid clasic printr-o funcţie particulară pe bază de puterile lui 2

$$f^*(n)=\frac{1}{1+2^{-n}} \qquad \textit{Ec. 55}$$

Diferenţa dintre ecuaţia de mai sus şi funcţia sigmoid clasică este un factor constant de scalare, care se poate obţine prin relaţia

$$e^{-z}=2^{\frac{-z}{\ln(2)}} \qquad \textit{Ec. 56}$$

La implementarea funcţiei, întregul este scalat cu acest factor constant şi adăugat la valoarea obţinută prin formula prezentată mai sus pentru funcţia sigmoid particulară. În graficele din Figura 3.22 sunt reprezentate funcţia sigmoid clasică (linia punctată 1) şi funcţia sigmoid particulară pe bază de puterile lui 2 (linia continuă). Eroarea maximă de aproximare a funcţie sigmoid clasice prin metoda menţionată este 0,0808.

Publicaţia (Simard & Graf, 1993) a experimentat pentru valori de intrare limitate la intervalul [-8,8], reprezentând cu 4 biţi partea întreagă şi cu 8 biţi partea fracţionară. Eroarea calculată, adică diferenţa dintre funcţia continuă şi funcţia cuantizată este în intervalul [-0.16, 0.16].

Unii autori au propus versiuni simplificate ale funcţiilor cum ar fi tangenta hiperbolică şi sigmoida rapidă. Lucrarea (Beiu, Peperstraete, & Vandewalle, 1994) a descris de asemenea un algoritm pentru derivata funcţiei sigmoid şi a extins formula de calcul asupra altor funcţii de activare menţionate.

În cazul aproximărilor funcţiei de activare sigmoid clasice prin metodele prezentate apare o eroare de aproximare care este diferită de zero. Din punctul de vedere al reţelelor neuronale artificiale această eroare de aproximare nu prezintă nici o semnificaţie datorită faptului că reţeaua neuronală este antrenată, şi nu este niciun indiciu din care să rezulte că erorile de aproximare ale funcţiilor de activare să se regăsească la ieşirea reţelei neuronale.

Din descrierile anteriore despre aproximarea funcţiilor de activare, rezultă că soluţiile clasice analizate sau implementate de cercetători sunt următoarele: tabele de căutare în RAM sau ROM, aproximarea PWL, aproximări cu grad superior, dezvoltare în serii Taylor, alte aproximări dedicate cum ar fi CORDIC.

Algoritmul CORDIC este cea mai studiată tehnică în privinţa implementării hardware a funcţiilor de activare, dar rareori o întâlnim utilizată. Avantajul este că acelaşi suport hardware poate fi utilizat pentru implementarea diferitelor tipuri de funcţii de activare, dar performanţa rezultantă de obicei nu este satisfăcătoare.

Aproximările funcţiilor de activare cu polinoame de grad ridicat duc la erori de aproximare reduse, dar nu sunt adecvate pentru implementare hardware din cauza numărului mare de operaţii aritmetice care trebuie efectuate pentru fiecare valoare, sau trebuie utilizate multe elemente hardware sau performanţa este compromisă. O observaţie asemănătoare este valabilă şi pentru implementările bazate pur pe metode tabelare. În cazul utilizării unui tabel de căutare de dimensiune redusă performanţele sunt nesatisfăcătoare, iar implementarea funcţiei printr-un tabel de dimensiuni semnificative costă. Aşa stau lucrurile atât în cazul implementărilor în ASIC, cât şi pe sisteme FPGA. Implementarea funcţiei prin combinarea funcţiilor polinomiale de ordin redus nu este o tehnică nouă. Întrebarea este cum să se aleagă cel mai bine punctele de interpolare în aşa fel încât dimensiunea tabelului de căutare să rămână redusă.

Interpolările cu funcţii polinomiale de grad redus au trei avantaje majore: aceeaşi structură hardware poate fi utilizată pentru implementarea diferitelor funcţii în care numai coeficienţii polinomului (conţinutul tabelei de căutare) trebuie redefinite; pot fi uşor implementate pe sisteme FPGA care conţin module de multiplicare, sumatoare şi tabele de căutare, respectiv se pot implementa relativ uşor prin utilizarea unui număr reduse de elemente logice (tabele de căutare, sumatoare).

3.2.4. Aritmetică neurală/Reprezentarea datelor

Scopul acestei părţi este clasificarea aritmeticii neurale folosite la sistemele FPGA şi examinarea efectului său asupra resurselor sistemelor FPGA. Deoarece executarea algoritmilor de reţele neuronale (de ex. backpropagation) necesită multe operaţii aritmetice, alegerea schemei aritmetice şi a reprezentării numerelor joacă un rol important în cazul în care avem în vedere proiectarea unui hardware de reţele neuronale.

Performanţa RNA pe hardware depinde în foarte mare măsură de domeniul aplicaţiilor în care se utilizează şi precizia numerelor folosite pentru reprezentarea semnalelor. Aceste semnale includ parametrii iniţiali cum ar fi cei de intrare, de ieşire, valorile ponderilor, şi valori intermediare cum ar fi funcţia de activare a fiecărui neuron, derivata funcţiei de activare şi valorile erorii. Scopul creării unei scheme aritmetice pentru implementările hardware este de a prezenta semnalele într-un număr şi cu o precizie suficientă care pot să ducă la convergenţa reţelei pe lângă păstrarea consumului de spaţiu în limitele resurselor FPGA.

Vom descrie mai multe scheme de reprezentare a datelor şi scheme aritmetice aplicate la arhitecturile prezentate în capitolele anterioare:

Aritmetica fluxului de impulsuri (Pulse Stream Arithmetic):

Modularea frecvenţei impulsurilor (Pulse Frequency Modulation, PFM) este o schemă de codificare unde valorile circuitului sunt reprezentate de frecvenţa unor impulsuri de bandă îngustă constantă (Lysaght J. , Stockwood, Law, & Girma, 1994).

Semnalele astfel codificate se pot multiplica şi aduna la fiecare nod folosind porţi logice simple. Această tehnică se numeşte aritmetica fluxului de impulsuri şi funcţionează la sisteme FPGA fin granulate.

Figura 3.23 reprezintă valoarea fracţionară a 7/16 prezentat prin flux de impulsuri. În lucrarea (Lysaght P. , Stockwood, Law, & Girma, 1994) s-a elaborat o arhitectură a neuronului bazată pe flux de impulsuri, pornind de la principiul aritmetic menţionat. Valoarea de intrare (valoarea de activare) pentru fiecare neuron este un flux constant de impulsuri, pe când ponderile sinaptice sunt construite ca şi funcţii restrictive prin mascare OR selectivă a unei serii de semnale de tact.

Semnalele de tact sunt obţinute de la ceasuri binare sincrone, fără suprapuneri, cu cicluri de ½, ¼ etc.

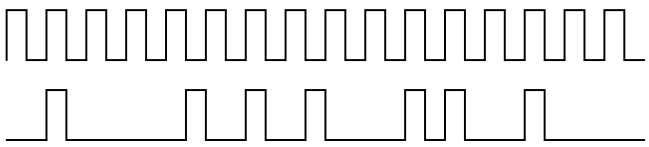


Figura 3.23 Reprezentarea valorii 7-16 prin flux de impulsuri

S-a folosit un generator de tact de 4 biţi care poate fi folosit pentru construirea unor ponderi între 0 şi 15/16.

Înmulţirea fluxului de impulsuri de intrare cu valorile ponderilor se poate face prin efectuarea simplei operaţii de AND între funcţia de intrare şi de mascare.

Reprezentarea în virgulă flotantă

Această reprezentare a numerelor la implementările RNA este foarte asemănătoare cu reprezentările la un calculator de uz general. Reprezentarea în virgulă flotantă a implementărilor hardware ale RNA oferă un domeniu şi o precizie mare la costul spaţiului de circuit valoros de pe FPGA. De aceea, în trecut, cei mai mulţi dintre cercetători au evitat folosirea unei aritmetici bazate pe reprezentarea punctelor în virgulă flotantă din cauza inaccesibilităţii sistemelor FPGA de capacitate mare.

Autorul (Simard & Graf, 1993) a folosit reprezentarea în virgulă flotantă pentru mai multe valori la implementarea algoritmului de propagare înapoi a erorii. Autorii au folosit mantisa de un 1 bit şi exponent de 4 biţi pentru rata de antrenare, mantisa de un 1 bit şi exponent de 3 biţi pentru valorile de stare activare şi mantisa de un 1 bit şi exponent de 5 biţi pentru valorile gradientului. Chiar dacă această reţea a fost validată pe o platformă software prima dată, autorii susţin în mai multe ipoteze că se poate implementa foarte uşor pe un chip.

Publicaţia (Cloutier, Pigeon, & Boyer, 1996) a construit un procesor de imagini virtuale (VIP) care foloseşte sistem FPGA, Altera EPF81500 cu 1,5 MB RAM static pentru a realiza algoritmii neurali, aplicaţiile de prelucrare a imaginilor şi de recunoaştere a formelor.

Autorii (Sahin, Gloster, & Doss, 2000) a verificat flexibilitatea aritmeticii în virgulă flotantă aplicată la sisteme reconfigurabile. Au arătat în această lucrare că dezvoltarea recentă a tehnologiei FPGA oferă utilizatorului mai multe resurse pe un singur dispozitiv şi, astfel, posibilitatea de a construi sisteme reconfigurabile complexe. În aceeaşi publicaţie (Sahin, Gloster, & Doss, 2000) s-au construit modulele lor în virgulă fixă, vectorul de adunare, scădere şi înmulţire folosind VHDL şi realizate pe dispozitiv FPGA Xilinx XC4044XL. Aceste module au viteze de 5 ori mai mari decât modelul software care rulează pe un calculator Pentium II 300MHz.

Reprezentarea în virgulă fixă

Reprezentarea în virgulă fixă este alegerea cea mai răspândită pentru implementarea algoritmilor RNA pe hardware. Aritmetica în virgulă fixă a fost mai efectivă din punctul de vedere al spaţiului utilizat în detrimentul reprezentărilor în virgulă flotantă. Deoarece multe aplicaţii de procesare a imaginilor şi RNA funcţionează în mod satisfăcător într-un domeniu restrâns şi precizie mică, şi reprezentarea în virgulă flotantă necesită o suprafaţă de siliciu mare, cercetătorii au folosit pe scară largă reprezentarea în virgulă fixă pentru aceste tipuri de aplicaţii.

Reprezintă un echilibru ideal între suprafaţa şi necesităţile de domeniu-precizie pentru arhitecturile RNA bazate pe FPGA.

În aritmetica în virgulă fixă, există mai multe posibilităţi prin care calculele de bază pentru algoritmul backpropagation cum ar fi înmulţirea, adunarea şi derivarea funcţiei de activare pot fi făcute.

Aritmetica bit-serial

În continuare sunt prezentate câteva dintre tehnicile aritmetice pentru realizarea de hardware în virgulă fixă. Acest tip de aritmetică calculează câte un bit, în timp ce aritmetica paralelă calculează toţi biţii în acelaşi timp, simultan. Ocupă suprafaţă mică pe chip dar este foarte lentă. Această metodă s-a utilizat la multe implementări RNA pe FPGA. Arhitecturile clasice RRANN (Eldredge & Hutchings, 1994) şi RENCO, binecunoscute implementări de RNA pe sisteme FPGA, au folosit acest tip de reprezentare.

Acest mod de abordare ajută la încadrarea cât mai multor elemente de procesare pe un FPGA de dimensiuni mici. Liniile de comunicare ale operaţiilor aritmetice necesită o lărgime de un singur bit, şi sumatorii necesită doar un bloc logic combinaţional Xilinx (Xilinx Combinational Logic Block).

Operaţii de decalare /fără multiplicare

Această tehnică necesită reprezentarea numerelor în funcţie de putere de 2.

Astfel înmulţirea/multiplicarea este implementată prin operaţii de decalare. A fost introdusă pentru prima dată de autorii publicaţiei (Simard & Graf, 1993). Dezavantajul metodei constă în găsirea unor numere potrivite pentru minimizarea erorii. Publicaţia (Molz, Molz, Moraes, Torres, & Robert, 2000) a folosit acest mod de abordare pentru a prezenta intrările şi ieşirea funcţiei de activare în format în virgulă fixă de 4 biţi.

Aritmetica on-line

Această metodă foloseşte o reprezentare redundantă a numerelor ceea ce face posibilă realizarea unor operaţii aritmetice foarte rapide. Această reprezentare se bazează pe aritmetica în serie. De aceea ea prezintă toate avantajele aritmeticii bit-serial cu o viteză sporită. Ea efectuează o operaţie aritmetică în mod MSDF, prima dată transmiţând cel mai semnificativ bit, spre deosebire de aritmetica în serie obişnuită care transmite prima dată datele cele mai puţin semnificative, printr-un mod de abordare LSDF.

Publicaţia (Girau & Tisserand, 1996) a explicat în detalii algoritmii pentru adăugare, înmulţire şi funcţia tangentă hiperbolică folosind transmisii MSDF (aritmetică on-line) cu reprezentarea numerelor redundantă. Prezentarea numerelor redundantă exprimă numerele în format radix-2.

Un astfel de format, definit ca şi împrumută-salvează, conţine setul de cifre -1,0,1.

În reprezentarea împrumută-salvează fiecare cifră a_i a unui număr a este reprezentat de 2 biţi, a_i^+ şi a_i^- astfel încât $a_i = a_i^+ - a_i^-$.

De exemplu, 1 este reprezentat prin (1,0), pe când pentru 0 există două posibilităţi de reprezentare: (0,0) şi (1,1).

Autorul din publicaţia (Skrbek, 1999) a propus o aritmetică neurală „deplasare adunare”, care oferă un set complet de funcţii aproximate potrivite pentru implementarea MLP pe FPGA. Elementele (calculele) de bază ale acestui mod de abordare sunt 2^x şi $\log_2 x$. Aceste funcţii se bazează pe operaţii de deplasare în combinaţie cu aproximarea liniară.

Reţelele neuronale artificiale au, din natura lor, o arhitectură paralelă şi sunt potrivite pentru implementare pe sisteme FPGA. O importantă problemă de implementare este să se determine formatul numeric utilizat care asigură un optim între precizie şi suprafaţa de implementare. Reprezentarea standard în virgulă flotantă de precizie simplă sau dublă minimizează erorile de cuantizare, dar necesită resurse hardware semnificative. Reprezentarea în virgulă fixă, mai puţin precisă, necesită un număr mai redus de elemente hardware, dar adaugă erori de cuantizare care îngreunează antrenarea reţelei, în special în probleme de regresie.

3.3. Concluzii

Creşterea continuă în densitate (număr porţi/suprafaţă) a sistemelor FPGA a făcut posibilă realizarea unor proiecte de sisteme on-chip cu o complexitate mai mare de un milion de porţi şi RAM intern. De aceea, sistemele FPGA s-au dovedit în prezent a fi o platformă hardware atractivă pentru algoritmii RNA care necesită mult spaţiu. Un alt avantaj al acestui dispozitiv este capacitatea de a combina programabilitatea cu viteza crescută a operaţiilor asociate cu soluţiile hardware paralele.

Arhitecturile bazate pe circuite FPGA pot fi exploatate bine pentru realizarea de reţele neuronale artificiale, deoarece utilizând capacitatea lor de reconfigurare rapidă şi concurentă se poate ajunge la performanţe bune în adaptabilitatea ponderală şi chiar topologică. Cu toate acestea, luând în calcul că prima astfel de aplicaţie (Cox & Blanz, 1992) a apărut numai cu aproximativ un deceniu în urmă, realizarea de reţele neuronale de dimensiuni mari, cu mulţi neuroni în circuite FPGA este o mare provocare chiar şi în zilele noastre. Acest lucru se datorează faptului că majoritatea modelelor neuronale clasice necesită un număr mare de operaţii de multiplicare, ceea ce duce la realizări hardware foarte costisitoare (multe circuite multiplicatoare complexe). În pofida acestei probleme, există tehnologii care

exploatează proprietatea de reconfigurabilitate a acestor circuite şi reuşesc să implementeze ieftin şi eficient RNA în acestea.

Orice realizare de reţea neuronală cu FPGA trebuie să se străduiască să utilizeze într-un anumit fel configurabilitatea acestor circuite, implicit optimizarea lor pentru problema propusă. Această proprietate care se poate aplica într-un timp scurt şi de nenumărate ori, este de folos în special în dezvoltarea de *prototipuri şi simulări*. Această flexibilitate permite realizarea rapidă de reţele neuronale cu diferite structuri şi strategii de învăţare, dar şi demonstrarea unor principii prin simulări primordiale. Proiectul GANGLION (Cox & Blanz, 1992) este un bun exemplu în acest sens.

Procedurile de *creştere a densităţii* realizează o mai bună funcţionalitate raportată la unitatea de arie FPGA, utilizând reconfigurarea. Acest lucru se poate valorifica dacă dispunem de un circuit FPGA capabil de reconfigurare – cel puţin parţială – în timpul funcţionării (run-time/partial reconfigurability). Primul tip de astfel de procedură este *multiplexarea în timp,* ceea ce înseamnă că simularea RNA este descompusă în mai multe faze secvenţiale, care corespund fiecare unei configurări diferite ale aceluiaşi circuit FPGA. Astfel se poate realiza un sistem în care circuitul este întotdeauna optimizat pentru faza care este în execuţie. De exemplu, autorii din lucrarea (Eldredge & Hutchings, Density enhancement of a neural network using FPGAs and run-time reconfiguration, 1994) au construit o reţea neuronală feed-forward care utiliza regula de învăţare back-propagation. Ei au împărţit algoritmul simulării în faza de reprezentare a intrărilor feed-forward, şi faza de învăţare back-propagation, care au rulat pe acelaşi circuit FPGA reconfigurat secvenţial.

Al doilea tip de procedură pentru creşterea densităţii de implementare se bazează pe execuţia de operaţii cu multiplicatori constanţi şi se numeşte *schimbare dinamică de constante*. Această metodă a fost utilizată în lucrarea (James-Roxby & Blodget, 2000), unde s-a arătat că, folosind ponderi de valoare constantă pentru a efectua foarte rapid operaţiile necesare, s-a putut efectua ulterior acordarea sinapselor, în timpul reconfigurării circuitului ceea ce a durat mai puţin de 69 μs. Deoarece aceste două proceduri au avut ca scop primordial creşterea densităţii de implementare, în ceea ce priveşte o performanţă superioară, nu putem avea pretenţii faţă de aceste sisteme, numai în cazul în care timpul de reconfigurare a circuitului este neglijabil în comparaţie cu timpul utilizat pentru efectuarea calculelor.
În circuitele FPGA reconfigurabile dinamice, există posibilitatea de implementare a reţelelor neuronale cu *adaptare topologică*. Altfel spus, putem construi RNA care suferă modificări arhitecturale succesive. În faza de învăţare, pe lângă topologia reţelei avem posibilitatea de a acorda şi precizia calculelor, conform funcţiei de criteriu utilizate de algoritmul de învăţare.

Câteva grupuri de cercetători şi-au propus realizarea RNA utilizând numere întregi pentru valorile ponderilor. Interesul faţă de această strategie a crescut deoarece complexitatea circuitelor de multiplicare necesare pentru efectuarea calculelor scade vertiginos, dacă utilizăm numere întregi în locul celor în virgulă mobilă. Se cunosc chiar şi reguli de învăţare, care utilizează puterile lui doi ca valori ale ponderilor. Avantajul mare al acestei idei este că operaţiile de multiplicare necesare algoritmilor de învăţare a RNA se pot efectua – în acest caz – foarte simplu, doar cu operaţii de deplasare a biţilor la stânga sau dreapta. Acele încercări, care au vizat implementarea RNA pe FPGA utilizând reprezentare în virgulă mobilă, au ajuns doar în faza de experimentare.

Neurohardware-ul digital diferă de asemenea în reprezentarea numerică a operaţiilor aritmetice. Implementările în virgulă flotantă oferă o precizie mai bună, dar mai complexă şi cu necesităţi de suprafaţă mai mare. Aritmetica în virgulă fixă este mai puţin complexă, cu necesităţi de suprafeţe mai reduse, dar oferă o precizie scăzută faţă de implementarea în virgulă flotantă. Aritmetica în virgulă fixă poate fi utilă în cazul în care un algoritm RNA nu necesită precizie ridicată pentru implementarea unor aplicaţii simple.

Alegerea preciziei ponderilor este un pas foarte important în procesul de construcţie a RNA cu ajutorul FPGA. Măsura preciziei ponderilor presupune întotdeauna un compromis între creşterea performanţelor reţelei şi costul implementării acesteia. Aplicarea de ponderi cu precizie ridicată duce la erori de cuantizare mai reduse în implementarea finală, iar utilizarea de ponderi cu precizie mai scăzută rezultă în viteze de execuţie mai mari, mai puţine resurse utilizate şi consum de energie mai moderat. În mod tradiţional, valoarea optimă se află prin testări şi evaluări de erori succesive (metoda empirică). Implementarea directă a unor funcţii de activare neuronale, neliniare, în circuite FPGA este prea costisitoare. Există însă soluţii mai practice, care garantează o precizie suficientă în cazul implementării acestor funcţii în FPGA.

4. POSIBILITĂŢI DE IMPLEMENTARE TOTAL PARALELĂ A MODELELOR NEURONALE PULSATIVE CU CIRCUITE FPGA

4.1. Consideraţii iniţiale

Există sisteme neuro-adaptive în timp real – clasificatoare, de control – la care învăţarea este proces critic, a cărui finalizare trebuie garantată într-un timp finit, bine determinat. Exemple elocvente de astfel de sisteme pot fi roboţii de recunoaştere, sisteme senzoriale ale sateliţilor aflaţi în medii necunoscute care caută obiecte, semnale interesante. Într-un astfel de mediu străin, sunt imposibil de prevăzut toate modelele posibile de tipuri de obiecte pe care automatul nostru clasificator le va întâlni. Rezultă, că sistemul nostru trebui să posede capacitatea de a încorpora cunoştinţe noi, de a învăţa să clasifice în timp real tipurile noi de obiecte întâlnite de-a lungul misiunii de recunoaştere.

În mod evident, s-a demonstrat de multe ori, că RNA pot fi utilizate excelent ca clasificatoare de precizie rapide. Cu toate acestea, însă RNA bazate pe modele de tip perceptron sau RBF (Radial Basis Function), necesită algoritmi de învăţare iterativi. Dacă acestea sunt implementate pe maşini de calcul secvenţiale, atunci însăşi procesul de învăţare va deveni un impediment în calea obţinerii unor performanţe edificatoare (Chakrabarti, Roy, & Soundalgekar, 2001). De aici rezultă că procesul de învăţare nu va fi convergent sau va dura prea mult pentru a fi de folos în sisteme în timp real. Acesta este motivul, pentru care RNA bazate pe învăţare iterativă nu se pot aplica în sisteme în timp real la care procesul de învăţare este unul critic. Concluzionând, putem afirma, că RNA oferă o paradigmă promiţătoare: rezolvarea de probleme învăţând din exemple. Ele realizează prelucrări de date foarte rapide, utilizând elemente masiv paralelizate. În timp ce o implementare software pe o platformă de tip Neuman este foarte utilă în evaluarea performanţelor unui anumit tip de reţea neuronală, în cazul unei implementări hardware este imperativ a ne concentra pe exploatarea paralelismului lor natural, pentru a le folosi în aplicaţii în timp real.

O rezolvare a problemei puse este algoritmul Kak (Tang & Kak, 2002), care permite învăţarea instantanee. De asemenea, un algoritm plauzibil din punct de vedere al implementării hardware este şi algoritmul FAST (Flexible Adaptable Size Topology) (Chakrabarti, Roy, & Soundalgekar, 2001) (Tang & Kak, 2002) (Pérez-Uribe, 1999). Reţeaua neuronală FAST este o reţea cu învăţare nesupervizată, cu o topologie flexibil adaptabilă. De fapt este o reţea feed-forward, formată din două straturi complet conectate, unul de intrare şi unul de ieşire. Constituirea claselor în care împarte reţeaua intrările primite se face pe baza corelaţiei dintre acestea. În cazul unei intrări suficient de diferite de cele de până atunci, dimensiune reţelei se poate mări, adăugând un neuron la stratul de ieşire. Dimensiune reţelei se poate diminua eliminând neuronii cu activitate redusă din stratul de ieşire.

4.2. Caracteristici generale ale sistemelor hardware pentru implementarea reţelelor neuronale. Motivaţii pro şi contra

4.2.1. Implementări analogice

4.2.1.1. Avantajele implementărilor analogice:

- viteză de procesare foarte mare
- integrabilitate foarte bună, densitate mare, neuronul analog are o complexitate mult mai redusă
- asigură o funcţionare total paralelă, este foarte uşor de implementat din cauza suprafeţei de circuit integrat utilizată (chip) reduse

4.2.1.2. Dezavantajele implementărilor analogice:

- **precizie redusă** - (şi la cea mai bună tehnologie precizia maximă este de 8 biţi) fapt ce rezultă din următoarele cauze:

- curgerile de curenţi în cazul capacitiv de stocare a ponderilor, suprafaţa condensatorului este redusă 30x30µm de o capacitate 1pF, unde un curent de ordinul 1pF produce o modificare a tensiuni de 1V/s
- tranzistoarele utilizate în modulele multiplicare analogică la curenţi reduşi nu sunt liniare, la curenţi mai ridicaţi liniaritatea este mai bună dar apare o nouă problemă rezultată din cauza drift-ului termic.
- imperfecţiunea procesului de fabricare a circuitelor analogice

- **sensibilitate**, zgomot, diafonie, asuprirea tensiunii de alimentare, drift-ul de temperatură,
- **modificarea ponderilor** ridică probleme
- deficienţa adaptivităţii - din cauza modificării ponderilor şi din lipsa algoritmilor continue de antrenare (majoritatea algoritmilor cunoscuţi de antrenare discretă în timp)
- posibilitatea de **legare în cascadă** a chip-urilor este greoaie
- cost ridicat de fabricare - modificarea parametrilor de fabricaţie trebuie să fie foarte redus. Testul după fabricare este complex, şi este posibil numai cu introducerea unui modul de autotestare care ridică suprafaţa chip-ului şi costurile de fabricare şi proiectare
- **proiectare greoaie** - necesitând experienţă de proiectare ridicată, şi o mulţime de simulări din cauza numărului ridicat de parametrii a căror efect trebuie luat în considerare.

Precizia obţinută cu circuitele analogice rămâne sub nivelul preciziei ce se poate obţine cu sistemele digitale. Deoarece în sistemele analogice prezenţa zgomotului influenţează direct valoarea semnalului purtător de informaţie trebuie avut grijă la reducerea sensibilităţii la zgomot a reţelei şi algoritmului de antrenare. La implementările analogice una dintre problemele majore constă în stocarea ponderilor modificabile. Aceste unităţi de stocare trebuie să satisfacă simultan următoarele condiţii:

- rezoluţie şi precizie ridicată
- stabilitate ridicată pe toată durata de exploatare, posibilitatea de stocare şi în lipsa tensiunii de alimentare
- posibilitate de modificare rapidă şi accesibilitate rapidă din exterior

La cele mai multe sisteme ponderile reţelei sunt constante sau pot fi modificate numai din exterior prin intermediul unui calculator.

4.2.2. Implementări digitale

Implementările digitale sunt mult mai răspândite faţă de sistemele analogice. Implementările digitale se pot realiza cu sisteme hardware relativ simple, astfel verificarea sistemului obţinut poate fi mai uşor efectuată faţă de sistemele analogice. Totodată cu implementare hardware se obţine o viteză de procesare ridicată, dar totuşi sub viteza de procesare obţinută prin sistemele analogice sau optice.

O parte din dezvoltările de hardware digitale de reţele neuronale de fapt înseamnă proiectarea unor sisteme de procesare paralelă multiprocesor, a căror arhitectură ia în considerare proprietăţile speciale a structurii reţelei neuronale utilizate. O altă posibilitate a reţelelor neuronale este implementarea neuronilor prin circuite digitale ca elemente de circuite logice programabile respectiv circuite VLSI cu un grad ridicat de integrare.

4.2.2.1. Avantajele implementărilor digitale

- precizie mare - preciza este în funcţie de numărul de biţi utilizaţi care teoretic poate fi foarte mare.
- este uşoară stocarea şi modificarea parametrilor reţelei
- în varianta digitală pot fi implementate sisteme mai mari faţă de sistemele analogice, permite o proiectare flexibilă
- proiectare simplă şi rapidă - proiectarea şi testarea este simplificată prin utilizarea unor programe evoluate pot fi utilizate elemente logice programabile, circuite „gate array”, microprocesoare
- stabilitate ridicată – rezultată din cauza modului de funcţionare sincronă
- insensibilitate la zgomot.

4.2.2.2. Dezavantajele implementărilor digitale

- viteză de funcţionare mai redusă faţă de sistemele analogice
- elementele de procesare (multiplicatoarele) ocupă o suprafaţă de chip mare, realizarea conexiunilor, şi a unui număr ridicat de semnale digitale ocupă o suprafaţă însemnată din suprafaţa circuitului integrat
- Proprietăţile mai importante a sistemelor optice din punctul de vedere a realizării reţelelor neuronale
- proprietăţile de liniaritate a sistemelor optice relativ sunt bune
- prin utilizarea luminii ca purtător de informaţie se ajunge la viteză ridicată de procesare
- realizarea operaţiilor de combinări liniare rapide este simplă
- se poate realiza un număr mare de conexiuni
- implementarea neliniarităţilor este problematică
- sistemul este încetinit de interfaţa dintre sistemul optic şi mediu
- realizarea ponderilor cu valori modificabile este greoaie

4.2.3. Calificarea implementărilor

Evaluarea implementărilor neuronale prin indicatoare standard (MFLOPS, MIPS) ne dă informaţie limitată despre sistem. Caracteristicile mai importante a chip-urilor neuronale este suprafaţa chip-ului, consumul şi viteza. Pentru caracterizarea implementării din punctul de vedere a vitezei s-au introdus următoarele indicatoare:

- Connections Per Second (CPS), - calculul sinapselor într-o secundă
- Connection Update Per Second (CUPS) - Numărul de ponderi modificate în faza de învăţare, este caracteristic vitezei de învăţare a reţelei.

Faţă de cele două indicatoare cel mai mult răspândite se mai obişnuieşte utilizarea acestor indicatori normaţi cu numărul sinapselor, respectiv înmulţit cu numărul de biţi din reprezentarea intrărilor şi a ponderilor.

4.2.4. Implementări RNA pe FPGA

Progresul în curs al microelectronicii este forţa motorică a dezvoltării continue a produselor tehnice noi. Sistemele FPGA sunt inovaţii de acest fel în sistemele microelectronice pentru aplicaţiile computerizate. În ultima decadă a devenit o metodă realizarea unor algoritmi bogaţi din punctul de vedere al calculului pe FPGA-uri.

Creşterea continuă în densitate a sistemelor FPGA a făcut posibilă realizarea unor proiecte de sisteme on-chip cu o complexitate mai mare de un milion de porţi şi memorii RAM interne. De aceea, sistemele FPGA s-au dovedit în prezent a fi o platformă hardware atractivă pentru algoritmii RNA care necesită mult spaţiu.

Un alt avantaj al acestui dispozitiv este capacitatea de a combina programabilitatea cu viteza crescută a operaţiilor asociate cu soluţiile hardware paralele.

În chip-urile FPGA există două moduri de implementare a logicii reconfigurabile. Sunt binecunoscute CTR (reconfigurarea compilată în timp) şi RTR (reconfigurarea in timpul utilizării). CTR este o strategie de implementare statică, la care fiecare aplicare constă într-o singură configurare. RTR este o strategie de implementare dinamică unde fiecare aplicaţie constă în multiple configuraţii cooperative.

În CTR, odată ce o operaţie începe, configuraţia nu se mai schimbă în timpul procesului. Această abordare este asemănătoare cu implementarea unei aplicaţii in ASIC. Pe când RTR utilizează o schemă de alocare dinamică care re-alocă partea de hardware în timpul executării aplicaţiei. Fiecare aplicaţie constă în mai multe configuraţii pentru FPGA.

Aceste configuraţii sunt încărcate succesiv în timpul derulării unei aplicaţii pe chip-ul FPGA. Acest lucru înseamnă realizarea unui nou hardware pentru fiecare configuraţie pentru porţiuni particulare ale aplicaţiei. Există două modele RTR: Globală şi locală. RTR global alocă întreaga resursă a întregului chip FPGA pentru fiecare pas al configuraţiei. Aceasta duce la o nouă realizare pe FPGA în timpul fiecărei configuraţii în timpul utilizării. Proiectantul trebuie să implementeze o aplicaţie în RTR global şi să împartă aplicaţia în părţi aproximativ egale pentru a utiliza în mod eficient resursele

FPGA. Aceasta se numeşte partiţionare temporală a unei aplicaţii. În RTR local, o aplicaţie reconfigurează local nişte subseturi ale logicii în timpul executării aplicaţiei. Poate configura orice procent din resursele reconfigurabile în orice moment. Organizarea aplicărilor in RTR local se bazează mai mult pe o divizare funcţională a muncii decât pe partiţionarea utilizată de aplicaţiile RTR global. Odată ce partiţionarea temporală manuală este decisă, poate fi foarte uşor implementată în varianta RTR globală.

Implementarea RTR local nu este posibilă în instrumente CAD.

Tabelul 7 de mai jos oferă o recapitulare a celor discutate în această secţiune.

Tabelul 7 Performanţele diferitelor platforme pentru implementări de reţele neuronale hardware

	ASIC analog	***ASIC digital***	***FPGA***	***Bazat pe procesor***	***Computer paralel***
Viteză	+++	++	+	–	+
Resurse	+++	++	+	–	––
Cost	––	––	++	++	––
Timp de design	––	––	++	+++	+
Fiabilitate	––	+	++	++	++

4.3. Sinteză asupra diferitelor implementări hardware de reţele neuronale artificiale

Acest capitol va prezenta câteva exemple de implementări hardware de reţele neuronale artificiale, cu accent pe cele neuromorfice, acestea din urmă reprezentând obiectul primar al tezei de doctorat.

În primii ani ai cercetărilor legate de reţelele neuronale artificiale, s-a presupus, că va fi nevoie de implementarea unor sisteme hardware specializate, pentru a beneficia de funcţiile promiţătoare ale acestor modele. De asemenea, s-a prevăzut, că aceste sisteme vor fi probabil circuite analogice, şi vor conţine multiple elemente de procesare paralele cu un număr mare de conexiuni între ele.

Aceste presupuneri nu s-au adeverit în totalitate, însă, deoarece creşterea imensă a capacităţii de calcul a maşinilor convenţionale von Neuman în domeniul digital a permis implementărilor software să atingă succese răsunătoare în nenumărate aplicaţii.

Între timp, dezvoltarea echipamentelor hardware de reţele neuronale a fost mult mai lentă, având un succes comercial mult mai modest. Sunt însă câteva contraexemple, dintre care voi prezente pe scurt, în cele ce urmează, cele mai relevante.

S-ar putea pune întrebarea: de ce să implementăm hardware reţele neuronale, ştiind că procesoarele convenţionale PC (de ex. cele von Neuman, seria Pentium) continuă să arate un ritm de dezvoltare dramatic? Evident, motivaţia este în continuare VITEZA, care poate fi dusă dincolo de limita procesării în timp real doar de circuitele neurale specializate, ASIC-urile analogice, digitale sau cele hibride.

Aplicaţiile de reţele neuronale software au un succes comercial din ce în ce mai semnificativ. De exemplu, orice scanner nou cumpărat se livrează cu un software de recunoaştere a caracterelor (OCR - Optical Character Recognition) care cel mai probabil utilizează algoritmi bazaţi pe reţele neuronale. Aceşti algoritmi nu sunt însă realizaţi exclusiv cu inteligenţă artificială, procesul de recunoaştere a caracterelor fiind mult mai complex, presupunând mai mulţi paşi de prelucrare a imaginii scanate (vezi Figura 4.1).

În termeni generali se poate afirma, că sunt foarte puţine acele probleme care se pot rezolva exclusiv cu o singură reţea neuronală.

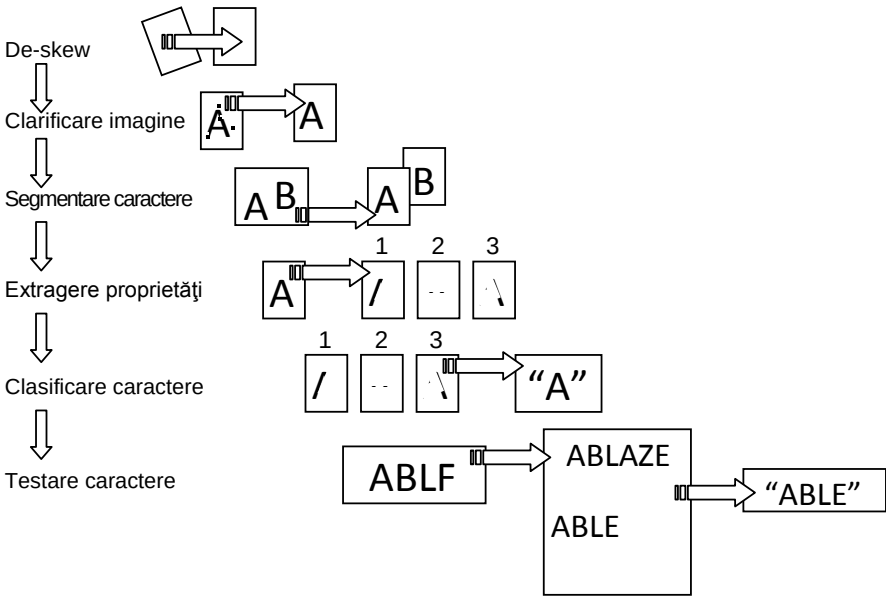


Figura 4.1 Paşii OCR, conform manualului de utilizare al Adaptive Solutions CNAPS

4.3.1.1. Legea lui Amdahl

Prezentată în 1967 (Amdahl, 1967) iar apoi versiunea acesteia conform lui Gustafson din 1988 (Gustafson, 1988), au arătat, că paralelizarea soluţiei unei probleme este eficientă doar dacă partea cea mai semnificativă a acesteia se poate implementa astfel. Spre exemplu, să presupunem, că 50% din aplicaţia mai sus amintită, programul OCR, poate fi executată de un sistem paralel. În acest caz factorul de accelerare a execuţiei este de numai 2. Numai dacă 90%, sau mai mult din program este executat în paralel, va ajunge factorul de accelerare la valori de peste 10. Aceste studii au sta la baza dezvoltării rapide a sistemelor cu procesare masiv paralelizată (MPP - massively parallel processing).

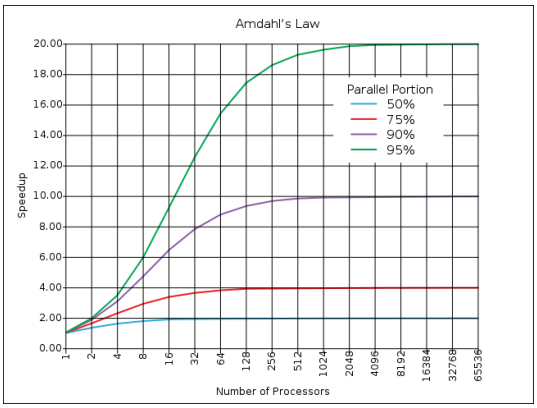


Figura 4.2 Legea lui Amdahl

4.3.1.2. Legea lui Gustafson

Cunoscută de asemenea ca legea Gustafson-Barsis, este o lege din domeniul ingineriei calculatoarelor care afirmă ca orice problemă de dimensiuni suficient de mari poate fi paralelizată în mod eficient. Legea lui Gustafson este strâns legată de legea lui Amdahl, care dă o limită gradului cu care un program poate fi accelerat cu ajutorul paralelizării. Această lege a fost prima dată descrisă de John L. Gustafson în 1988 (Gustafson, 1988).

$$S(P) = P - \alpha \cdot (P - 1) \qquad \text{Ec. 57}$$

unde P este numărul de procesoare, S este accelerarea, iar α este partea procesului ce nu poate fi paralelizată.

Legea lui Gustafson abordează deficienţele legii lui Amdahl, care nu poate scala astfel încât să egaleze disponibilitatea puterii computaţionale odată cu creşterea în dimensiuni a maşinii. Elimină problema dimensiunilor fixe sau a sarcinii computaţionale fixe pe procesoarele paralele: în locul acestora propune un concept de timp fix care duce la scalarea accelerării pentru probleme de dimensiuni mai mari (adică scalare *slabă* sau *moale*).

Legea lui Amdahl (Amdahl, 1967) se bazează pe volum de lucru fix, sau dimensiuni ale problemei fixe (adică scalare *puternică* sau *tare*). Ea implică faptul că partea secvenţială a unui program nu se schimbă în raport cu dimensiunea maşinii (adică numărul de procesoare). Însă partea paralelă este distribuită în mod egal la *n* procesoare.

Impactul legii a fost simţit prin schimbările în cercetare pentru dezvoltarea de compilatoare care paralelizează şi reducerile în partea serială a soluţiei pentru a impulsiona performanţele sistemelor paralele.

Implementarea legii lui Gustafson

Fie *n* o măsura a dimensiunilor problemei.

Execuţia programului pe un computer paralel se descompune în:

$$a(n)+b(n)=1 \qquad \textit{Ec. 58}$$

unde *a* este fracţiunea secvenţială şi *b* este fracţiunea paralelă.

Pe un computer secvenţial, timpul relativ ar fi $a(n)+p\cdot b(n)$ unde *p* este numărul de procesoare în cazul paralel.

Astfel, accelerarea este:

$(a(n)+p\cdot b(n))$ în cazul paralel, şi luând în considerare relaţia $a(n)+b(n)=1$, obţinem:

$$S=a(n)+p\cdot(1-a(n)) \qquad \textit{Ec. 59}$$

unde $a(n)$ este funcţia serială.

Presupunând că funcţia serială $a(n)$ se micşorează odată cu dimensiunea problemei *n*, atunci accelerarea se apropie de *p* când n se apropie de infinit, exact cum s-a dorit.

Legea lui Gustafson argumentează că chiar şi folosirea sistemelor de computere masiv paralele nu influenţează partea serială, şi tratează această parte ca o constantă. În comparaţie, ipoteza legii lui Amdahl rezultă din ideea că influenţa părţii seriale creşte odată cu numărul de procese.

O *metaforă* de conducere auto pentru a explica aceste concepte ar fi următoarea:

Legea lui Amdahl sugerează aproximativ că:

„Presupunem că un autoturism circulă între două oraşe aflate la o distanţă de 60 mile între ele, şi a petrecut deja o oră parcurgând jumătate din distanţă la viteza de 30 mph. Indiferent de cât de rapid veţi conduce pe ultima jumătate de drum, este imposibil să atingeţi viteza medie de 90 mph înainte de a ajunge la destinaţie. Deoarece a trecut deja o oră şi distanţa este de 60 mile în total; chiar şi circulând cu o viteză infinit de mare nu veţi putea depăşi viteza medie de 60 mph."

Legea lui Gustafson sugerează aproximativ că:

„Presupunem că un autoturism a circulat deja pentru o anumită perioadă de timp cu o viteză mai mică de 90 mph. Având la dispoziţie suficient timp şi distanţă de parcurs, viteza medie a autoturismului poate eventual atinge oricum valoarea de 90 mph, indiferent de cât de mult sau de încet a circulat deja până acum. De exemplu dacă autoturismul a petrecut deja o oră la viteza de 30 mph, ar putea atinge media dorită circulând cu 120 mph pentru încă două ore, sau cu 150 mph pentru încă o oră, şi aşa mai departe."

4.3.2. Implementări hardware neuronale analogice

Implementarea hardware analogică se remarcă prin abilitatea de a efectua calcule (adunări, înmulţiri, transformări neliniare) extrem de rapid. *Viteza ridicată de procesare* şi de eficienţa calculelor bazate pe principiile clasice ale teoriei circuitelor electrice contrabalansează principalul dezavantaj al metodelor analogice: *imposibilitatea* obţinerii unei *precizii* ridicate.

Unul din elementele cele mai importante pentru o reţea îl reprezintă *interconexiunile* dintre neuroni. În mod tipic numărul de neuroni ce pot fi integraţi pe un chip este limitat de numărul de conexiuni posibile. Aceste interconexiuni pot să fie *fixe*, implementate prin *rezistori* de o anumită valoare sau *programabile*. Prima alternativă se foloseşte în aplicaţiile în care este cunoscută arhitectura RNA şi valorile ponderilor.

Implementările analogice a sistemelor de reţele neuronale sunt stimulate de viteza mare de prelucrare. De obicei în aceste sisteme semnalele sunt funcţii continue de timp, iar purtătorul de informaţie este o mărime analogică - de exemplu intensitatea curentului. VLSI analog este un instrument potrivit pentru implementarea sistemelor neuronale inspirate din modele biologice(de exemplu sisteme neuronale de procesare sunet şi imagine). Cu instrumente analogice se poate realiza foarte uşor însumarea ponderată respectiv caracteristica neliniară.

Pentru implementarea analogică a neuronului tipic se utilizează un amplificator operaţional, iar ponderarea intrărilor este realizată de raportul rezistenţelor legate în serie corespunzătoare intrării. Ponderile sinaptice sunt fixe (realizate prin rezistenţe cu valori fixe), potrivite pentru realizarea reţelelor fixe.

Principiul de bază a construirii sistemelor neuronale analogice antrenabile şi utilizării lor în sistemele adaptive este implementarea setului de ponderi cu valori modificabile. În literatura de specialitate se regăsesc două tehnici specifice:

- ❖ valorile ponderilor sunt stocate în condensatoare. În cazul utilizării condensatoarelor ca elemente pentru stocarea ponderilor precizia de implementare şi timpul de stocare este redus din cauza scurgerilor de curenţi, scurgeri care pot descărca condensatorul.
- ❖ utilizarea tranzistoarelor MOS cu poartă grilă flotantă: sunt tranzistore cu mai multe porţi grilă, iar una dintre porţi este izolată. Sarcina trimisă pe poarta izolată se conservă, iar rezistenţa canalului MOSFET poate fi continuu modificată în funcţie de sarcină. Timpul de stocare este mult mai mare şi scurgerile de curenţi sunt mai mici.

Această tehnologie este utilizată şi la chip-urile ETANN proiectate de INTEL.

Pentru multe aplicaţii precizia de stocare a ponderilor prin metodele analogice amintite mai sus nu este îndeajuns. Din acest motiv ponderile sunt stocate digital. Ponderile stocate digital sunt trimise pe intrarea multiplicatorului analogic printr-un convertor digital analog, sau se utilizează convertoare digital analogice cu înmulţire MDAC (Multiplyaing D/A converter).

Abordări VLSI analogice ("AVLSI") mai recente marchează o orientare spre mimarea în detaliu a comportamentului neuronului biologic. Implementarea s-a realizat în tehnologie de 10µm nMOS.

În acelaşi context ca si abordarea precedentă se regăseşte şi implementarea în siliciu a dendritelor unui neuron. Conductanţa axială este implementată prin intermediul unei capacităţi comutate iar conductanţa trans-membrană este realizată printr-un amplificator transconductanţă.

4.3.3. Implementări hardware neuronale digitale

Există două principale avantaje ale acestei implementări faţă de versiunea analogică:

Uşurinţa proiectării şi realizării. Folosirea tehnologiei VLSI digitale oferă o precizie ridicată, stocarea facilă a ponderilor şi un raport performanţă/cost mai bun decât varianta analogică.

Flexibilitatea. Implementarea VLSI digitală permite utilizarea unor algoritmi complecşi de o mare diversitate, mărind astfel gama aplicaţiilor posibile.

Unul dintre *dezavantajele* majore ale tehnologiei VLSI digitale îl constituie **consumul ridicat** de putere şi arie de siliciu a blocurilor de multiplicare. Una dintre soluţiile pentru reducerea suprafeţei necesare o constituie multiplexarea interconexiunilor.

În cazul implementărilor digitale ajungem la sisteme cu eşantionare, fiecare semnal este discret în timp şi amplitudine. Implementările digitale sunt motivate de concepţia conecţionistă vizavi de realizările analogice motivate de modelele biologice. Se poate privi ca o realizare digitală acel hardware, pe care rulează un program neuronal. Avantajele oferite de reţelele neuronale pot fi exploatate cu adevărat numai în cazul unui hardware special proiectat. Din punctul de vedere de al arhitecturii chip-urilor neuronale acestea se pot clasifica astfel:

- **chip neuronal multiprocesor -** chip-urile neuronale multiprocesoare sunt realizat cu ajutorul mai multor procesoare care pot funcţiona în diferite moduri de lucru: SIMD, MIMD, matrici sistolici.
- **Arhitectură** felie (slice). Arhitectura de proiectare a sistemelor digitale bazată pe slice-uri este o metodă bine testată şi utilizată cu succes. Elementele funcţionale sunt descompuse pe slice-uri care uşor pot fi combinate în cascadă. Din slice-uri se pot construi elemente de dimensiuni preferabile. Avantajul principal al acestei abordări este că permite o proiectare modulară, reduce timpul şi costurile de proiectare.
- **LSI waffer-scale**. În cazul tehnologiei WSI, analog fabricaţiei circuitelor integrate, slice-urile sunt descompuse pe chipuri dar se realizează şi conexiunile dintre chipuri şi de alimentare. Aceste resurse la sistemele WSI LSI sunt supra-proiectate şi cu redundanţă ridicată. La sfârşitul procesului de fabricare chipurile cu defecte sunt neutralizate, şi circuitul este utilizat întreg fără dezmembrare în chip-uri.
- **Chip-uri speciale -** Aceste chip-uri au fost dezvoltate pentru reţele şi aplicaţii specifice. Avantajul lor este că numai blocurile şi unităţile necesare sunt implementate faţă de neurocalculatoarele de uz general.
- **Soluţii dedicate.** Scopul soluţiilor dedicate este creşterea vitezei de procesare. Posibilitatea de creştere a vitezei de procesare depinde de utilizarea unei proiectări orientată pe metoda ţintă prin

care se măreşte viteza de prelucrare a unităţilor individuale totodată este posibilă amplasarea mai multor unităţi de procesare pe un singur chip

Soluţia optimală poate fi remarcată prin proiectarea de către utilizator a unor circuite orientate pe echipament. Fabricarea acestora însă este rentabilă numai în cazul unui număr ridicat de unităţi. Din această cauză - şi în principal la sistemele fabricate în număr de bucăţi reduse - a crescut rapid utilizarea sistemelor logice programabile. Deoarece ştiinţa reţelelor neuronale în mare parte este în faza de cercetare, dezvoltarea soluţiilor hardware dedicate sunt bazate pe sisteme logice programabile.

În faza de cercetare-dezvoltare dintre sistemele logice programabile în primul rând sunt utilizate acelea care sunt reprogramabile. Din această categorie fac parte chipurile FPGA XILINX conţinând blocuri logice reconfigurabile şi conectarea acestora prin comutatoare programabile.

Fiecare tip de reţea neuronală conţine unităţi aritmetice, unitatea de comunicare dintre neuroni , unitatea de comandă a întregii reţea, etc. Problema cea mai mare constă în realizarea circuitului de înmulţire, necesitând un număr foarte mare de elemente logice şi astfel acesta ocupă o suprafaţă însemnată din suprafaţa chip-ului. Scopul este introducerea unui număr ridicat de unităţi de procesare pe un singur circuit integrat, astfel şi acele reţele care nu conţin elemente de înmulţire pot fi implementate eficient.

4.3.4. Aplicaţii ale implementărilor hardware de reţele neuronale

Deşi nu atât de reuşite precum reţelele neuronale software, există şi reţele neuronale hardware care funcţionează la parametri optimi zi de zi.

4.3.4.1. Accurate Automation Corp (AAC)

Descriere: Procesorul de reţele neuronale al Accurate Automation (Neural Network Processor - NNP) utilizează o adevărată arhitectură cu instrucţiuni şi date multiple (multiple-instruction multiple-data – MIMD), capabilă de a rula în paralel pe mai multe chipuri fără ca performanţa sa să aibă de suferit.

Fiecare chip conţine un procesor de o viteză înaltă de 16 biţi cu stocare de date în circuitul integrat (CI) pentru ponderile sinaptice. Procesorul efectuează doar nouă instrucţiuni lingvistice de asamblare. De exemplu, numai o singură instrucţiune este necesară pentru a transmite ieşirea unui neuron prin funcţiile de transfer bazate pe valori de prag, specifice majorităţii metodelor de învăţare neuronale." Comunicarea interprocesorală dintre multiple NNP-uri este dirijată de către o magistrală interprocesorală proprie. Un toolbox software este inclus care deja a codificat programe în limbaj de asamblare pentru toate metodele cunoscute de învăţare neuronale optimizate pentru procesoare paralele multiple. De exemplu, toolbox-ul implementează algoritmi de back-propagation, reţele Hopfield, metode de învăţare Adaline şi Madaline, învăţarea recurentă şi cea Kohonen fiind în curs de dezvoltare în cadrul Accurate Automation.

Chipurile sunt disponibile doar de la Accurate Automation fiind deja montate pe circuite imprimate. O versiune ISA pentru PC ($4,995) vine împreună cu un chip şi spaţiu pentru a adăuga circuite imprimate de extensie cu până la încă nouă chipuri ($3,995 bucata). Versiunea VME ($19,995) a circuitului – cea folosită pentru LoFLZTE – include de asemenea două TMS340 DSP-uri plus spaţiu pentru a instala încă nouă chipuri neuronale. DSP-urile permit circuitului să efectueze o preprocesare de semnale pentru reţeaua neuronală, cum ar fi filtrarea digitală, recunoaştere de model, optimizare şi procesare de matrice rară.

Învăţare: Poate fi programat pentru a implementa orice algoritm de antrenare neuronală.

Performanţe: 140 MCPS pentru un singur chip. Până la 1.4 GCPS pentru un sistem de 10 procesoare.

4.3.4.2. Adaptive Solutions CNAPS

Descriere: sistemul CNAPS este un sistem de dezvoltare a reţelelor neuronale, bazat pe circuitul propriu CNAPS-1064, Procesor Paralel Digital care are 64 de sub-procesoare operând în mod SIMD. Fiecare sub-procesor are proprii 4k biţi de memorie locală şi o unitate aritmetică în virgulă fixă pentru a efectua operaţii cu parametrii întregi reprezentaţi pe 1-bit, 8-biţi sau 16-biţi. Fiecare sub-procesor poate emula unul sau mai mulţi neuroni şi mai multe circuite pot fi conectate împreună.

Serverul CNAPS II este mediul de dezvoltare software şi hardware. Are o magistrală VME cu 4 slot-uri şi un controller Ethernet. Un singur card VME al serverului CNAPS II poate fi setat pentru 64, 128, 256 şi 512 de configuraţii, iar un card are un spaţiu de stocare de 16M Bytes.

Procesoarele trebuie să fie programate pentru a executa un anumit algoritm de reţea neuronală. Utilităţile CNAPS includ CNAP-C (un compilator C şi un debugger), Quicklib (funcţii codate manual apelabile din CNAPS-C), BuildNet (algoritmi pre-codaţi ai reţelelor neuronale) şi CodeNet (debugger pentru limbaj de asamblare).

Cardul CNAPS/PC ISA foloseşte 1,2 sau 4 circuite integrate paralele ale noului procesor CNAPS-1016 sau două din circuitele integrate 1064 pentru a obţine procesoare CNAPS de 16, 32, 64 sau 128. În cazul acestora, ratele de executare a operaţiilor de multiplicare şi adunare sunt de 320M, 640M, 1.28G şi 2.56G pe secundă. Viteza datelor magistralei ISA este de 20MByte/s. Sistemul de dezvoltare CNAPS este similar cu cel pentru staţia de lucru: compilator CNAPS-C, modul de asamblare, Debugger, API, QuickLib şi programe demonstrative (McCartor, 1991).

Software-ul BrainMaker a fost de asemenea portat pentru a rula card-uri CNAPS PC.

Învăţare: Algoritmii de învăţare pot fi programaţi. Cei clasici ca back-propagation şi alţi algoritmi de acest fel vin cu pachetul BuildNet.

Performanţe: Cu un singur chip, se poate atinge 1.28 miliarde ca rată de multiplicare şi adunare pe secundă. Cu 4 chipuri, se ajunge la 10.24 miliarde rată de multiplicare şi adunare pe secundă. Algoritmii backpropagation pentru reţele de tip feedforward au performanţe de 1.16 miliarde ca rată de multiplicare şi adunare pe secundă şi 293 milioane modificări de ponderi pe secundă cu un *chip*.

4.3.4.3. IBM ZISC036

Descriere: ZISC este un circuit integrat (CI) cu 64 intrări de 8-biţi şi 36 neuroni RBF. Mai multe CI pot fi puse laolaltă pentru a crea reţele de mărime arbitrară. Vectorul de intrare V este comparat cu un vector prototip P stocat pentru fiecare neuron. Ieşirea neuronului de 14-biţi (pentru posibile categorii de 16K) este valoarea de distanţă calculată conform a două norme selectabile: (1) dist = sum of abs(V_i-P_i), i=1,64; (2) dist = Max((V_i-P_i)), i=1,64. Intrarea este serială pe 8-biţi. Durează 3,5 microsecunde pentru a încărca cele 64 de elemente şi încă 0,5 microsecunde pentru ca semnalul de clasificare să apară.

Învăţare: Învăţare on-chip, stocare a vectorilor de instruire ca prototipuri. Procesarea învăţării a unui vector durează încă 2 microsecunde în afara celor 4 microsecunde pentru încărcare şi evaluare.

Performanţe: La 16MHz, clasificarea unui vector de 64 componente reprezentat pe 8-biţi se efectuează în 4 microsecunde.

Comentarii: Prima generaţie de chip-uri utilizează o tehnologie VLSI foarte conservatoare. Generaţiile următoare sunt planificate şi vor folosi progresiv o tehnologie mai agresivă pentru a mări viteza şi densitatea neuronilor.

4.3.4.4. Intel 80170NX Electrically Trainable Analog Neural Network (ETANN)

Descriere: reţele neuronale analogice cu 64 de ieşiri (0÷3V), 16 offseturi interne şi 64 neuroni cu funcţii de transfer sigmoidal. Reţelele de tip feedforward cu două straturi pot fi implementate cu 64 de intrări, 64 de neuroni din stratul ascuns şi 64 neuroni de ieşire, folosind două matrici de ponderi de 80x64. Ieşirile stratului ascuns vor trece printr-o altă matrice de ponderi pentru a procesa stratul de ieşire. Alternativ, o singură reţea de 64 de straturi şi 128 ieşiri poate fi implementată folosind ambele matrici şi prezentarea intrărilor în 2 faze de câte de 64 de intrări. Ponderile sunt stocate în sinapse non-volatile de tip *floating gate* synapses (multiplicatoare analogice Gilbert). Ponderile au o precizie de numai 6-biţi.

Sistemul de învăţare INNTS este un mediu de dezvoltare PC care include software (emularea şi interfaţa chip-ului), o extensie PC a magistralei card-ului şi un modul de instruire pentru un singur chip. De asemenea, un circuit imprimat de 8 chip-uri este disponibil.

Învăţare: Fără învăţare on-chip. Emularea este făcută în software şi este descărcată pe circuit. Cu toate acestea, pentru a aduce performanţa aproape de nivelul emulării este necesară introducerea unei faze speciale de învăţare.

Performanţe: Un timp de propagare de aproximativ 8 microsecunde pentru o reţea cu 2 straturi. Acesta este echivalent cu numai 2 miliarde de rate de multiplicare/adunare pe secundă.

4.3.4.5. Irvine Sensors 3DANN

Descriere: Irvine Sensors au dezvoltat o tehnologie de stratificare FET de mare densitate care permite circuitelor să fie construite atât pe dimensiuni verticale cât şi pe dimensiuni orizontale. Folosind aceste circuite stratificate se poate construi o reţea neuronală artificială 3DANN - 3D (NNW) de foarte mare densitate. Compania a câştigat un contract cu Armata SUA pentru a "demonstra fezabilitatea unui sistem

de interconectare cu densitate ridicată pentru a îmbunătăţi tehnologia reţelei neuronale artificiale". Noua schemă de interconectare face parte din planul de îmbunătăţire a tehnologiei Irvine Sensors în vederea atingerii unui "Silicon Brain", un sistem de recunoaştere conceput pentru a emula performanţele sistemului nervos central al omului.

Bazându-se pe o dezvoltare similară al CalTech, care emula cortexul vizual utilizând circuite FPGA, s-a ales o structură stratificată de FPGA-uri pentru proiectul de emulare Silicon Brain Architecture SBIR Phase II. Utilizarea de FPGA este importantă, deoarece se pot implementa multe configuraţii de circuite integrate fără costuri de design şi executare a acestora.

Dificultatea impusă de circuitele FPGA convenţionale la implementarea unor funcţionalităţi electronice complexe este faptul, că conexiunile de care dispun nu sunt suficiente. Conceptul 3DFET permite interconexiuni dense între diferite straturi.

În noua implementare a Silicon Brain Architecture SBIR Phase II se va inlcude 3DFET, precum şi circuite FPID (Field Programmable Interconnect Devices), care vor permite ca interconxiunile dintre circuitele FPGA stratificate să fie şi ele total programabile.

Astfel, se vor putea implementa diferite scheme de interconectare, cum ar fi cea de fluture, crossbar sau mesh. Cu o asemenea flexibilitate a sistemului stratificat, se poate construi orice prototip de procesor digital. Potenţialul de piaţă al unui asemenea produs este, evident, imens.

4.3.4.6. National Semiconductor NeuFuz/COP8 Microcontrollers

Descriere: Acest sistem foloseşte o combinaţie de reţea neuronală şi software cu logică fuzzy pentru a genera cod către microcontrolerele National Semiconductors COP8. Logica fuzzy oferă o metodă foarte flexibilă şi puternică pentru aplicaţiile de control. Atunci când regulile "if-then" ale sistemului sunt cunoscute, ele pot fi exprimate într-un mod sistematic şi direct utilizând funcţia de apartenenţă fuzzy şi metodologia regulilor fuzzy. Totuşi, pot exista multe situaţii când nu se cunosc regulile "if-then" dinainte. În asemenea cazuri, o reţea neuronală poate fi folosită pentru a învăţa regulile şi apoi "cunoştinţele" reţelei neuronale sunt înglobate în regulile fuzzy şi funcţiile de apartenenţă. National Semiconductors oferă mai multe pachete:

- NeuFuz Kit de Învăţare (NF2-C8A-Kit): software cu reţea neuronală PC/AT (2 intrări, 1 ieşire) şi generator de legi şi funcţii de apartenenţă fuzzy (maximum 3 funcţii), generator de cod COP8, assm/linker COP8.
- NeuFuz4 (NF2-C8A): software cu reţea neuronală PC/AT (4 intrări, 1 ieşire) şi generator de legi şi funcţii de apartenenţă fuzzy (maximum 3 funcţii de apartenenţă), generator de cod COP8, assm/linker COP8.
- NeuFuz4 Sistem de Dezvoltare (NF2-C8A): software cu reţea neuronală PC/AT (4 intrări, 1 ieşire) şi generator de legi şi funcţii de apartenenţă fuzzy (maximum 3 funcţii de apartenenţă), generator de cod COP8, assm/linker COP8, emulator în circuit cu programare PROM COP8.

Învăţare: doar învăţare software.

4.3.4.7. Nestor NI1000

Descriere: Aceasta este o reţea cu neuroni Radial Basis Function. În timpul învăţării, vectorii prototip sunt stocaţi prin presupunerea faptului că aceştia sunt aleşi aleatoriu din distribuţia principală. Pot fi stocaţi până la 1024 de prototip cu 256 de dimensiuni, 5-biţi pe dimensiune. Apoi fiecare prototip este desemnat unui anumit neuron din stratul mijlociu. Acest neuron, la rândul lui, este desemnat unui neuron de ieşire care reprezintă clasa specifică pentru acel vector (până la 64 de clase admise). Toţi neuronii din stratul de mijloc, care corespund aceleiaşi clase sunt desemnaţi aceluiaşi neuron de ieşire. În modul de redare, un neuron de intrare este comparat cu fiecare prototip, în paralel, şi dacă distanţa dintre ei este deasupra pragului stabilit, acesta se activează, şi activează la rândul lui ieşirea, clasa sau neuronul corespondent.

Algoritmii de învăţare pot varia şi nu salvează pur şi simplu fiecare prototip. De exemplu, algoritmul aparte poate decide dacă un vector de intrare este salvat ca un nou prototip sau e considerat a fi prea asemănător unui prototip deja existent. Setarea pragului poate să depindă şi de algoritm.

Învăţare: Doi algoritmi de învăţare on-chip sunt disponibili: Reţeaua Neuronală Probabilistică (RNP) şi Restricted Coulomb Energy (RCE). De asemenea, micro-codarea poate fi modificată pentru algoritmi definiţi pentru utilizatori.

Performanţe: 40k modele a 256 elemente pe secundă (25 micro-secunde/model).

4.3.4.8. Philips L-Neuro chips

Descriere: Philips a oferit anterior chipul L-Neuro 1.0 şi a anunţat dezvoltarea următoarei generaţii, chipul L-Neuro 2.3.

L-Neuro 1.0 conţine 16 elemente de procesare (EP) cu registre de 6-biţi. Totuşi, cei 16-biţi pentru un EP pot fi consideraţi ca 16 neuroni de 1-bit, 4 de 4-biţi, 2 de 8-biţi sau 1 de 16-biţi. Aşadar, un singur chip poate implementa, spre exemplu, o reţea neuronală cu 256 neuroni de 1-bit. Există o memorie tampon pentru ponderi de 1kByte pe chip pentru a implementa 1024 de ponderi reprezentate pe 8-biţi sau 512 ponderi reprezentate pe 16-biţi. Funcţiile de transfer sunt implementate în afara chip-ului pentru ca mai multe chip-uri să poate fi conectate unul de altul. Cele 16 ieşiri EP sunt citite în serie. Chip-urile sunt cel mai uşor interfaţate pe procesoare gazdă Transputer. O rată de procesare de 100MCPS pentru modul cu reprezentare pe un 1-bit şi una de 26MCPS sunt raportate pentru modul cu reprezentare pe 8-biţi pentru un singur chip. Circuitul poate fi programat să înveţe şi rate de 160MCUPS, respectiv 32 MCUPS este dat pentru modul de 1-bit şi 8-biţi. Un sistem bazat pe transputer cu până la 112 L-Neuro 1.0 este disponibil şi de la Telmat care vinde sisteme Transputer. De asemenea, este disponibil şi un circuit imprimat PC cu un singur circuit L-Neuro.

L-Neuro 2.3 este a doua generaţie care a beneficiat de experienţa lui L-Neuro 1.0. Este conectabil în cascadă ca şi L-Neuro 1.0 şi constă din 12 procesoare care pot opera fie în mod paralel (SIMD), fie în mod pipelined. Are ponderi pe 16-biţi şi ieşiri neuronale în modul basic. Fiecare din cele 12 procesoare conţine 128 de registre cu reprezentare pe 16-biţi pentru a stoca ponderi şi stări, un multiplicator de 16 pe 32 biţi, o unitate Aritmetică şi Logică (ALU) pe 32-biţi. Micro-codarea circuitului integrat este disponibilă utilizatorului pentru a putea fi proiectat conform unei anumite aplicaţii. Cu o viteză de tact de 60MHz, circuitul poate calcula ieşirile, la fiecare 17ns, ponderile fiind reprezentate pe 32 de biţi, iar valorile de intrare pe 16. Acesta asigură 720MCPS, adică de 27 ori modul cu reprezentare pe 8-biţi al L-Neuro 1.0. În procesul învăţării, cele 12 ponderi sunt updatate în paralel în 34ns. Circuitul integrat se poate folosi nu doar ca aplicaţie pentru reţele neuronale, dar şi ca aplicaţie pentru procesare de semnal, procesare de imagine şi logică fuzzy (folosindu-şi abilitatea de a extrage valori minime si maxime din vectorii celor 12 elemente).

4.3.4.9. Sensory Circuits RSC-164 Speech Recognition Chip

Descriere: RSC-164 este un CI recunoaştere a vorbirii cu cost redus, proiectat pentru uzul produselor electronice. Acesta combină un procesor pe 8-biţi cu algoritmi de reţele neuronale pentru o recunoaştere a vorbirii, atât independentă de interlocutor cât şi dependentă de acesta, de cea mai bună calitate. Circuitul integrat conţine de asemenea funcţiile de înregistrare a vocii, redare, sinteză de muzică, sinteză de vorbire şi sistem de control. Dispozitivul CMOS include memorii RAM şi ROM înglobate pe chip, 16 linii I/O, convertor D/A şi un procesor dedicat de 4-MIPS.

RSC-164 utilizează o reţea neuronală apriori antrenată să execute recunoaşterea vorbirii, în timp ce sinteza de înaltă calitate a vorbirii este dobândită folosind o schemă de compresie timp-domeniu care îmbunătăţeşte ADPCM-ul de tip obişnuit. Filtrarea digitală pe chip ameliorează precizia recunoaşterii, pre-procesând semnalele primite. Controlul dinamic AGC compensează persoanele care nu stau la o distanţă corespunzătoare faţă de microfon sau pe cele care vorbesc prea încet sau prea tare.

RSC-164 include o interfaţă externă de memorie care permite conectarea cu dispozitivele de memorie pentru înregistrare audio/playback, recunoaşterea dependentă de vorbitor şi lungimi extinse de mesaje pentru sinteza de vorbire.

4.3.4.10. Siemens MA-16 chip, SYNAPSE-3 Neurocomputer

Descriere: circuitul integrat Siemens MA-16 este un multiplicator rapid de matrici care poate fi combinat pentru a forma *serii sistolice*, adică intrările şi ieşirile sunt trimise de la un modul la altul ca într-o linie de asamblare. Un singur modul poate procesa 4 modele, de 16 elemente fiecare (pe 16-biţi), cu 16 valori neuronale (pe 16-biţi) la o rată de 800 multiplicări-adunări/secundă la 50Mhz. Ponderile sunt încărcate de pe RAM-ul din afara circuitului iar funcţiile de transfer neuronal sunt calculate cu tabele de căutări (look-up table) externe circuitului integrat.

SYNAPSE-1 este un sistem hardware/software complet care foloseşte 8 circuite integrate MA-16. Acesta se află în propria lui carcasă şi comunică via Ethernet cu o staţie de lucru gazdă. Cardul magistralei de accelerаţie SYNAPSE2*PC conţine 1 MA-16, în timp ce card-ul SYNAPSE3-PC PCI conţine 2 MA-16.

SYNAPSE3-PC: Până la trei circuite SYNAPSE3*PC pot fi inserate într-un PC Pentium. Gazda PC şi circuitul(ele) pot executa operaţii simultane sau independente una de cealaltă. Performanţele cele mai înalte ale unui circuit sunt de aproximativ 2560 MOPS (1,28 x 10 E 09 MultAcc/sec); 7160 MOPS (3,58 x 10 e 09 MultAcc/sec) când se folosesc 3 circuite! O asemenea configuraţie obţine performanţe mai bune decât SYNAPSE1, având de asemenea performanţe I/O mult mai ridicate şi un raport optim preţ/performanţe. SYNAPSE3-PC este utilizat ca accelerator pentru reţelele neuronale MediaInterfaces ale SynUse·Base Software.

Învăţare: Circuitul integrat nu are implementat hardware nici un algoritm specific dar poate fi programat să facă calcule, cum ar fi pentru back-propagation. Cardurile SYNAPSE pot fi programate pentru aproape orice fel de algoritm de reţele neuronale.

Performanţe: SYNAPSE-1 cu 8 MA-16 a avut performanţe de vârf de 3,2 miliarde de multiplicări (16-biţi x 16-biţi) şi adunări (48-biţi) pe secundă la o frecvenţă de bază de 25Mhz. SYNAPSE3-PC: 2560 MOPS (1,28 x 10 E 09 MultAcc/sec); 7160 MOPS (3,58 x 10 E 09 MultAcc/sec) când se folosesc trei circuite.

4.4. Implementări software şi hardware ale reţelelor neuronale bazate pe impulsuri

S-a demonstrat deja de mulţi cercetători, pornind de la studiile sistemelor biologice, că structura temporală a şirurilor de impulsuri din aceste modele neuronale neuromorfe are o putere de procesare demnă de luat în considerare. Aplicaţiile acestor modele, spre exemplu în probleme de segmentare de imagini, necesită simularea unor reţele largi într-un timp rezonabil de mic. În cele ce urmează se vor prezenta diferite platforme hardware pe care aceste reţele neuronale au fost deja implementate. După cum se va vedea, majoritatea acestor sisteme sunt foarte complexe şi costisitoare, acest lucru făcând necesară dezvoltarea de acceleratoare neuronale neuromorfe implementate hardware mult mai accesibile atât ca preţ, cât şi ca timp de dezvoltare.

4.4.1. Neuronii pulsativi

Să examinăm în primul rând particularităţile de bază ale neuronilor pulsanţi (Pérez-Uribe, 1999) (Gerstner W. , 2001) (Trappenberg, 2002):

- Neuronii comunică unul cu celălalt doar prin impulsuri temporizate
- Impulsurile care intră sunt ponderate şi induc un potenţial post-sinaptic conform unei funcţii impuls-răspuns.
- Funcţiile impuls-răspuns sunt construite din unul sau mai multe integratoare permeabile care sunt descrise într-o versiune discretă de PI (n+1) = r* PI (n), unde PI este un modul potenţial intern şi r este un factor de dispersie.
- Modulele potenţiali interni (PI) sunt combinaţi în funcţiile de ieşire conţinând adunare, scădere şi/sau multiplicare şi cel puţin o funcţie de prag.
- Reţelele pot consta din 10^5 neuroni şi pot fi distribuiţi în straturi de neuroni cu proprietăţi identice. Fiecare neuron este conectat cu până la 10^4 alţi neuroni.
- Structura de conectivitate a unei reţele poate fi împărţită în două grupuri: 1.) *conexiuni regulate (CR)* care urmează nişte reguli deterministice simple (ex. câmpuri receptive) şi 2.) *conexiuni neregulate (CNR)* (conexiuni rare, aleatorii).

4.4.2. Probleme de implementare

Când implementăm asemenea reţele pe computere digitale, timpul *t* decurge în unităţi temporale T proprii discrete (de obicei T = 1ms). Simularea unei etape de timp proprie pentru întreaga reţea ar trebui să fie un *pas temporal* (Time Step - TS). A calcula un pas temporal în mai puţin de o milisecundă va fi numită *simulare în timp real*. Procesarea în fiecare pas temporal implică trei operaţii:

Intrare: impulsurile trebuie să fie distribuite şi neuronii-receptori de impulsuri trebuie să incrementeze potenţialele lor interne, PI.

Ieşire: fiecare neuron trebuie să-şi combine potenţialii interni şi să decidă dacă să trimită un impuls la ieşire sau nu.

Dispersie: potenţialii interni trebuie să fie dispersaţi pentru fiecare neuron. Trebuie menţionat faptul că datorită necesităţii stocării PI-urilor, calcularea acestor reţele este I*/O-limitată* (nr. I/O > nr. de operaţiuni). Putem aplica diferite metode pentru a mări viteza simulării:

- *Activitatea reţelei* (numărul mediu al impulsurilor pe TS / numărul de neuroni) este de obicei foarte scăzută. O schemă foarte eficientă de comunicare pentru aceste reţele este *protocolul listă-eveniment* (event-list protocol). Doar adresele neuronilor pulsanţi trebuie înregistrate într-o listă-eveniment a impulsurilor şi trebuie să distribuie impulsuri pentru intrare (Lazarro & Wawrzynek, 1993).
- În mod caracteristic, reţelele neuronale pulsative nu sunt complet conectate. O metodă de a reprezenta această conectivitate risipită (sparse connectivity) este folosirea listelor, una pentru fiecare neuron n_i. Unităţile de informaţie din liste sunt seturi de date conţinând ponderi şi adrese. Adresele a_j din listă pot indica neuronul n_j căruia n_i îi trimite un impuls sau din partea căruia n_i primeşte un impuls. Cel dintâi reprezintă o *conectivitate emiţător-orientată*, iar cel din urmă o *conectivitate receptor-orientată* (Frank & Hartmann, 1995). Pentru o activitate de reţea scăzută metoda emiţător-orientată este avantajoasă: doar neuronii receptori de impulsuri trebuie să calculeze funcţia de intrare (Jahnke, Roth, & Klar, 1995).
- Pentru o conectivitate uzuală putem calcula conexiunile unui neuron independent de poziţia sa în spaţiu. *Calcularea on-line* a conexiunilor reduce drastic cantitatea necesară de stocare pentru listele de conexiune. Presupunând că vectorii de pondere sunt similari pentru toţi neuronii, atunci nu trebuie să stocăm decât un vector de pondere pentru întreaga reţea.
- În loc de a utiliza aritmetica în virgulă mobilă, este recomandată folosirea aritmeticii rapide în virgulă fixă. Conform rezultatelor analizei de rezoluţie (Roth, Jahnke, & Klar, 1995), acurateţea de la 8 biţi (ponderi) până la 18 biţi (potenţiali interni) este suficientă pentru problemele de vizibilitate redusă.

În ceea ce priveşte simularea pe un computer paralel, vom face o deosebire între trei metode de mapare a unei reţele care pot fuziona una cu cealaltă:

- **n(euron)-paralel:** toate sinapsele ale unui neuron sunt mapate aceluiaşi element de procesare PE. Diferiţi neuroni sunt procesaţi în paralel, dar sinapsele sunt procesate în serie.
- **s(inapsă)-paralelă:** sinapsele unui neuron sunt distribuite către PE-uri diferite. Acum sinapsele unui neuron sunt procesate în paralel şi neuronii în serie.
- **m(odel)-paralel:** PE-urile calculează răspunsul pentru un model de intrare în paralel, într-un segment anume al reţelei, în timp ce diferite segmente de reţea vor fi procesate în serie.

4.4.3. Implementare pe calculator paralel

SP2 este un computer MIMD cuplat în paralel, cu până la 256 R6000 PE-uri, arhitectură cu memorie locală şi comutatoare 16 pe 16 de viteză înaltă pentru comunicare inter-PE. Planificarea n-paralel şi conectivitatea emiţător-orientată au fost folosite pentru implementare (Delorme & Thorpe, 2003). Pe de-o parte, un singur PE manifestă o mult mai slabă performanţă decât un P90 care poate fi rezultatul unor setări nesatisfăcătoare ale compilatorului şi necesită investigaţii suplimentare. Pe de altă parte, observăm o accelerare de numai 1.12/1.23/1.25 folosind PE-uri de 2/4/8 peste implementarea unui PE individual. Aşadar, performanţa este îndeosebi limitată de către comunicare inter-PE (calculul şi comunicarea nu pot fi realizate în paralel pe SP2) care rezultă în comportamentul slab la scalarea impulsurilor PE observate.

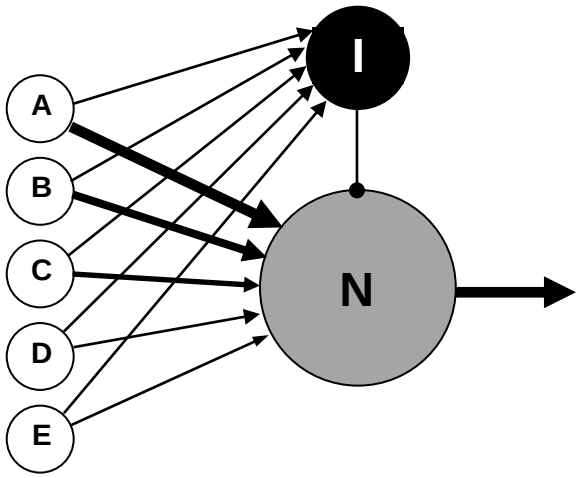


Figura 4.3 Modelul neuronal conceput de Thorpe

4.4.4. SpikeNET

Simon Thorpe şi echipa sa (Arnaud Delorme, Rufin VanRullen, Marie Fabre-Thorpe) în Toulouse, Franţa, sunt unii dintre acei cercetători care au ajuns să fie la cel mai înalt nivel – în zilele noastre – în ceea ce priveşte studiul reţelelor neuronale neuromorfe (spiking). Alături de multe alte realizări, ei au inspirat apariţia primului produs comercial bazat pe aceste cercetări, şi anume **SpikeNET**, un sistem de recunoaştere vizuală.

În lucrarea lor, se argumentează, că un sistem bazat pe codarea ordinii rangurilor, *rank order coding,* - unde primele impulsuri recepţionate de neuroni sunt luate în considerare, iar cele care sosesc mai târziu sunt inhibate – poate livra o cantitate extraordinară de informaţi (Delorme & Thorpe, 2003). Pe lângă acest fapt, simplitatea modelului de dinamism neuronal, care poate realiza ordonarea este foarte atractivă.

În acest model (vezi Figura 4.3), neuronul ***I*** inhibă neuronul ***N***. Astfel, o acumulare de impulsuri ce activează ***N,*** va stimula spre activare şi neuronul ***I***, a cărui factor de inhibare, va împiedica ***N*** să mai emită impulsuri, chiar dacă va mai recepţiona intrări ulterioare.

Alte cercetări foarte importante în acest domeniu, se desfăşoară în Austria, la Graz, sub conducerea lui Wolfgang Maass, împreună cu Berthold Ruf, Heinrich Schmitt şi alţii.

Se poate vedea, că această tematică este foarte tânără, dar cu mare potenţial, conturându-se deja, mai multe direcţii promiţătoare de cercetare.

4.4.4.1. Sistemul de simulare reţele neuronale neuromorfe SpikeNET

În prezent sunt disponibile o multitudine de sisteme, care oferă posibilităţi variate de simulare a reţelelor neuronale de orice fel. Unele dintre acestea au fost dezvoltate pentru a aproxima în medii artificiale, detaliile biofizice ale funcţionării neuronilor biologici. Tipic pentru unităţile ce s-au născut astfel este o structură cu o ieşire simplă (de multe ori 0 sau 1). Aplicare acestora este destul de răspândită, în domenii inginereşti şi financiare. La celălalt capăt al spectrului se află programe sofisticate cum ar fi GENESIS (Bower & Beeman, 1998) sau NEURON (Hines & Carnevale, 1997). Acestea au o excelentă capacitate de a modela şi simula procese biofizice, spre exemplu structura dendritică sau cinematica canalelor ionice ale membranelor celulelor. Acest nivel ridicat al detalierii funcţiunilor reprezintă însă, un impediment în calea implementării eficiente a unor reţele mari conform acestor modele.

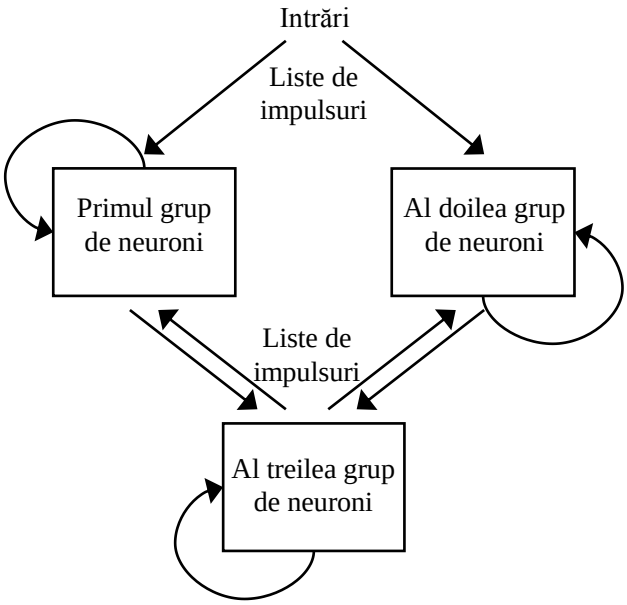


Figura 4.4 *Structura de bază a sistemului SpikeNET*
Se redirectează liste de impulsuri între diferite grupe de neuroni, organizaţi într-o matrice bidimensională.

Deoarece numai un număr mic de neuroni emit impulsuri în fiecare pas temporal, traficul comunicaţiei este minimalizat.

Unul dintre avantajele SpikeNET (Figura 4.4) este viteza. Utilizând un calculator G3 Macintosh (PowerPC 750, cu procesor la 266 Mhz), SpikeNET poate updata aproximativ 20 de milioane de conexiuni pe secundă, chiar şi în cazul utilizării unui parametru de sensibilitate pentru a modula efectul fiecărei intrări sinaptice. Astfel se poate realiza simularea în timp real a unei reţele formate din 400.000 neuroni, cu un pas temporal de 1ms (presupunând 49 de conexiuni pe neuron şi o rată medie de activare de 1 pe secundă, acestea fiind o aproximare rezonabilă a ratei de activare a neuronilor corticali). De notat este faptul, că utilizând o simulare mai convenţională de reţele neuronale, ar fi nevoie de procesarea fiecărui element la fiecare pas temporal. Acest lucru ar însemna, că utilizând aceeaşi putere de calcul, numai 20.000 de conexiuni ar putea fi procesate pe milisecundă, ceea ce ar limita mărimea reţelei simulate în timp real la circa 400 de neuroni (49 de conexiuni/neuron).

4.4.5. Reţele neuronale pulsative, dezvoltări hardware

Cel mai bun model neuronal, care asigură fidelitatea cea mai ridicată faţă de fenomenele biofizice din celulele nervoase biologice, este cel propus de Hodgkin şi Huxley (Hodgkin & Huxley, 1952). Acest model se bazează pe patru ecuaţii diferenţiale neliniare cuplate, care implică timpi de simulare software mari, iar implementările hardware ar fi total ineficiente din punct de vedere al raportului preţ/performanţă (Gerstner & Kistler, 2002). Implementările hardware necesită deci, un model mai simplu, cum ar fi modelul numit Integrare-şi-Activare, I&A, (Integrate-and-Fire) studiat în (Koch, 1999) (Dayan & Abbott, 2001) şi care a stat si la baza implementărilor realizate şi prezentate în primul referat al acestui program de doctorat (Bakó, Analiza şi simularea sistemelor cu inteligenţă artificială neuromorfă, 2005). Modelul se bazează pe o ecuaţie diferenţială de ordinul unu, unde rata de schimbare a stării unui neuron *v*, este corelată cu curenţii de membrană. Modelul I&A utilizat este următorul:

$$c_m \frac{dv(t)}{dt} = g_l(E_l - v(t)) + \sum_j \frac{w^j g_s^j(t)}{A}(E_s^j - v(t)) \qquad \textit{Ec. 60}$$

Pentru o sinapsă anume j, apariţia unui potenţial pre-sinaptic la un moment t_{ap} declanşează activarea acesteia la momentul $t_{ap} + t^j_{delay}$, prin efectuarea unei schimbări discontinue în conductanţa sinaptică de forma:

$$g_s^j(t_{ap} + t_{delay}^j + dt) = g_s^j(t_{ap} + t_{delay}^j) + q_s^j \qquad \textit{Ec. 61}$$

altfel, $g^j_s(t)$ este controlat de:

$$\frac{dg_s^j(t)}{dt} = \frac{-1}{\tau_s^j} g_s^j(t) \qquad \textit{Ec. 62}$$

Folosind schema de integrare directă Euler cu un pas temporal de *dt=0.125ms* pentru a rezolva ecuaţiile modelului I&A, rezultă:

$$v(t+dt) = v(t) + \frac{1}{c}((g_l(E_l - v(t)) + \sum_j \frac{w^j g_s^j(t)}{A}(E_s^j - v(t))))dt \qquad \textit{Ec. 63}$$

$$g_s^j(t+dt) = g_s^j(t) + (\frac{-1}{\tau_i} g_s^j(t))dt \qquad \textit{Ec. 64}$$

Acest model a fost utilizat de către un ***grup de cercetători din Irlanda de Nord, sub conducerea lui B. Glackin***, pentru a implementa reţele neuronale pulsative pe circuite FPGA. Tabelul 8, de mai jos prezintă explicaţii, respective valori utilizate în această lucrare (Glackin, McGinnity, Maguire, Wu, & Belatreche, 2005), pentru ecuaţiile modelului I&A.

Tabelul 8. Descrieri şi valori ale simbolurilor utilizate în ecuaţiile modelului I&A

Parametru	Valori uzuale	Valori în celule piramidale	Valori în celule inhibitorii	Descriere
c_m	8nF/mm^2			Capacitatea membranei
E_l	-70mV			Potenţialul de inversare a membranei
v_{th}	-59mV			Tensiunea de prag
v_{reset}	-70mV			Tensiunea de reset
τ_{ref}	6.5ms			Perioada de hiperpolarizare
t^j_{delay}	0.5ms			Decalajul temporal de propagare
g_l		1μs	1μs	Conductanţa de scurgere a membranei
A		0.03125mm^2	0.015625mm^2	Suprafaţa membranei
E^j_s		0mV	-75mV	Potenţialul de inversare a sinapsei
τ^j_s		1ms	4ms	Timpul de descreştere al sinapsei

Implementarea hardware prezentată utilizează o reprezentare a datelor în virgulă fixă. Au fost utilizaţi 18 biţi pentru reprezentarea potenţialului de membrană, din care 10 biţi sunt partea fracţionară. Parametrii de multiplicatori şi divizori au fost aleşi ca valori multiple ale puterilor lui 2, pentru a putea implementa aceste operaţii cu registre de deplasare.

Algoritmul de antrenare implementat în hardware în această lucrare este unul cu fidelitate biologică ridicată, şi anume algoritmul de Plasticitate dependentă de temporizările impulsurilor (Spike Timing Dependant Plasticity - STDP), bazat pe studiul realizat de Song şi Abott în (Song, Miller, & Abbott, 2000) (Song & Abbott, 2001). Astfel, utilizând o rată de conectare de 1:1 între neuroni şi sinapsele STDP, s-a reuşit implementarea a unui total de 168 de neuroni şi 168 de sinapse pe un circuit FPGA Xilinx, din familia Virtex, XC2V8000, al cărei structuri se poate vedea în Figura 4.5. Din capacitatea acestui circuit această reţea neuronală a ocupat aproximativ 60% din resursele reconfigurabil. Dacă s-a aplicat o rată de conectare mai realistă de 1:10, atunci reţeaua a avut 41 de neuroni şi 410 sinapse, iar dacă rata s-a ridicat la un nivel de 1:100, atunci valorile erau de 4 neuroni şi 400 de sinapse.

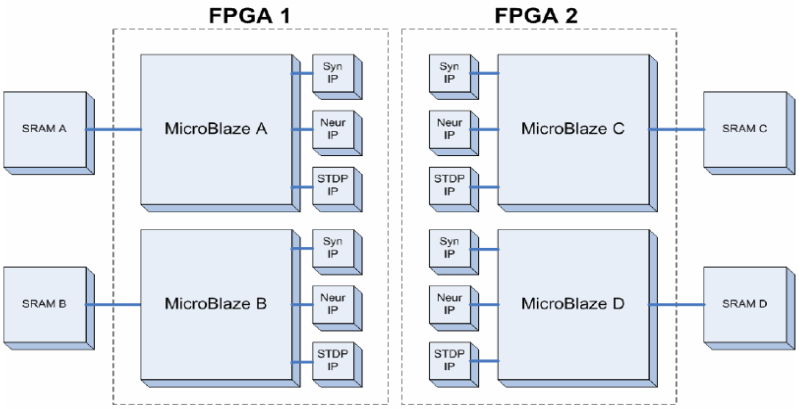


Figura 4.5.- Structura sistemului din (Glackin, McGinnity, Maguire, Wu, & Belatreche, 2005) pe un circuit FPAG Xilinx Virtex XC2V8000

Utilizând această arhitectură complet paralelizată la o viteză de tact de 100 MHz şi un pas de integrare Euler de 0.125ms pe ciclu de tact, autorii acestei lucrări (Glackin, McGinnity, Maguire, Wu, & Belatreche, 2005) au reuşit să atingă executarea unei perioade de funcţionare în timp real de o secundă în numai 0.8ms. Acest lucru înseamnă o rată de accelerare de 12500 faţă viteza de timp real. O alternativă pentru mărirea mărimii reţelei implementate este partajarea sistemului între o implementare hardware şi una software, însă această formulă implică un compromis între utilizarea mai eficientă a resurselor reconfigurabile şi viteza de execuţie a reţelei. Astfel, s-ar putea implementa pe una sau mai multe dintre controllerele MicroBlaze înglobate în circuitul Virtex, un algoritm de multiplexare în timp a unui singur neuron sau a unei sinapse. O diagramă a paşilor de execuţie a acestui sistem este dată în Figura 4.6.

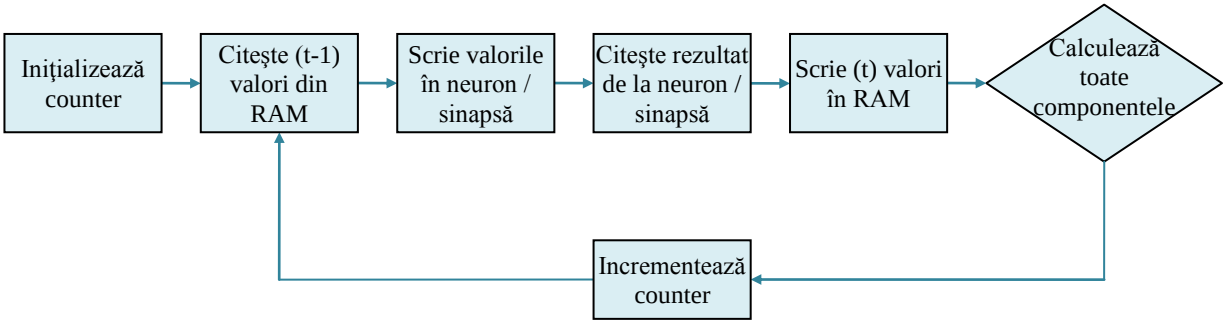


Figura 4.6 Paşii de execuţie a sistemului cu elemente de reţea multiplexate prezentată în (Glackin, McGinnity, Maguire, Wu, & Belatreche, 2005)

Versiunea cu mai multe controlere utilizate din FPGA, atinge o îmbunătăţire de performanţă de 107.25 faţă de o simulare în Matlab implementată pe un PC Pentium 4 la 2GHz (vezi Tabelul 9).

Tabelul 9 Timpii de simulare pentru o secundă de timp real

Mărimea reţelei	*PC (Matlab)*	*1 MicroBlaze*	*4 MicroBlaze*
*100*100*100*	*2917.2 s*	*115.7s*	*27.2s*
*1400*1400*1400*	*454418s*	*N/A*	*4237s*

Un sistem care reprezintă o categorie diferită de implementări de reţele neuronale de tip spiking a fost dezvoltată de un grup de cercetători din Spania, (Agis, Díaz, Ros, Carrillo, & Ortigosa, 2006).

Acest motor de simulare bazată pe evenimente, implementat pe un circuit FPGA Xilinx Virtex-II, poate procesa 2.5 milioane de impulsuri pe secundă, el conţinând şi cinci memorii SRAM externe. Acest lucru face ca acest sistem să poată fi folosit cu succes în experimente de simulare care necesită motoare de simulare de tip embedded (de exemplu experimente robotice cu agenţi autonomi). Această schemă este implementată de obicei în software (Delorme & Thorpe, SpikeNET:an event-driven simulation package for modelling large networks of spiking neurons, 2003) (Ros, Carrillo, Ortigosa, Barbour, & Agis, 2005) (Mattia & Del Giudice, 2000) (Reutimann, Guigliano, & Fusi, 2003), dar a fost abordată şi hardware (Makino, 2003) (Schoenauer, Atasoy, Mehrtash, & Klar, 2002) (Mehrtash, Jung, Hellmich, Schoenauer, Lu, & Klar, 2003). Astfel, motorul de simulare poate face salturi de la un impuls de activare la celălalt. Ca urmare, toată activitatea reţelei se organizează în liste cronologice de evenimente (impulsuri) care se procesează în mod secvenţial. Această arhitectură se pretează foarte bine calculatoarelor convenţionale, datorită naturii sale secvenţiale.

În ceea ce priveşte performanţele simulării şi utilizarea de resurse hardware, se poate afirma, că procesarea totală a căii de date consumă 74 cicluri de tact (în cazul în care lista de evenimente are mai puţin de 1024 de elemente). Utilizând o structură pipe-line, pentru a procesa aceste liste, se reduce sarcina la un impuls de procesat pe fiecare 13 cicluri de tact. Astfel, la o viteză de tact de 33 MHz se obţin cele aproximativ 2.5 milioane de procesări de impulsuri pe secundă.

Comparaţia acestor implementări cu altele din domeniu este dificilă, deoarece fiecare utilizează alt model neuronal.

Un alt centru important de cercetare al reţelelor neuronale neuromorfe este laboratorul de Sisteme Logice al *Swiss Federal Institute of Technology*, Lausanne, Elveţia. Autorii lucrării (Upegui, Peñ a-Reyes, & Sánchez, 2004), pun accent pe proprietăţile de reconfigurare dinamică parţială (dynamic partial reconfiguration - DPR) a celor mai recente circuite FPGA, care permite refolosirea aceloraşi resurse logice pentru diferite configuraţii. Se poate reduce astfel necesarul de resurse hardware pentru un anumit proiect şi se poate optimiza numărul neuronilor precum şi conexiunile dintre aceştia. Există şi încercări de a realiza adaptare topologică utilizând DPR, cum ar fi cea descrisă în (Upegui, Peña-Reyes, & Sánchez, 2003)[66]. O topologie modulară, care utilizează DPR, permite mai multe configuraţii pentru fiecare modul, făcând posibilă şi transmiterea bidirecţională a impulsurilor în cazul reţelelor recurente.

Modulele conţin straturi de neuroni cu conexiuni predefinite, iar un algoritm genetic ar putea căuta combinaţia optimă de straturi.

În (Upegui, Peñ a-Reyes, & Sánchez, 2004) se prezintă un model neuronal funcţional, bazat pe învăţare Hebbiană, implementat în hardware pe baza căruia s-a implementat şi o reţea neuronală, cu scopul studierii necesităţii de resurse logice pe un anumit suport. Aplicaţia de test a fost discriminare între două semnale de frecvenţe diferite.

Autorii lucrării mai sus menţionate, au utilizat un circuit FPGA Xilinx Spartan II XC2S200 pe care au implementat un neuron cu 30 de intrări, cu şi fără învăţare ale căror necesităţi de resurse sunt date de Tabelul 10.

Tabelul 10 Resurse utilizate dintr-un circuit XC2S200 de către un neuron implementat de (Upegui, Peñ a-Reyes, & Sánchez, 2004)

Neuroni fără învăţare			**Neuroni cu învăţare**	
Neuron cu 14 intrări	Neuron cu 30 intrări	Neuron cu 62 intrări	Neuron cu 30 intrări	Întreaga reţea (30 de neuroni cu 30 intrări)
17 slice-uri 0.72%	23 slice-uri 0.98%	46 slice-uri 1.95%	41 slice-uri 1.74%	1350 slice-uri 57.4%

Utilizând modelul neuronal cu 30 de intrări ei au implementat o reţea cu trei straturi, (Figura 4.8) fiecare strat conţinând 10 neuroni, complet conectaţi. Fiecare neuron are câte zece intrări, din stratul propriu, din stratul precedent şi din stratul următor. Această reţea a ocupat 57.4%din resursele circuitului FPGA, un strat necesitând 19.13%.

În continuare se prezintă un procesor de reţea neuronală cu fidelitate biologică şi funcţionare în timp real, bazat pe circuit FPGA.

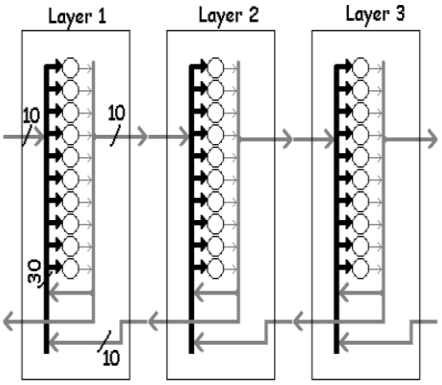


Figura 4.8 Structura reţelei din (Upegui, Peñ a-Reyes, & Sánchez, 2004)

Modelul neuronal utilizat în acest procesor – dezvoltat în Anglia (Pearson, şi alţii, 2005) – este unul singular (uni-compartimental), aparţinând clasei de excitabilitate de ordinul I, conform clasificării date în (Izhikevich E. , 2004). Ponderea fiecărei sinapse poate fi supusă influenţei unui zgomot multiplicativ cu distribuţie Rayleigh. De asemenea, se poate injecta un zgomot cu distribuţie Gaussiană, în magnitudinea valorii de prag a potenţialului de membrană. Aceste distribuţii ale zgomotelor utilizate au fost alese astfel încât ieşirea modelului să aproximeze cât mai fidel datele empirice biologice. Pentru a modela reţele spaţial dispersate, s-au implementat decalaje inter-neuronale specifice fiecărei sinapse. Atât perioada de dispersie absolută, cât şi cea relativă au fost înglobate în modelul dezvoltat. Utilizarea reprezentării acestor parametrii în virgulă mobilă ar necesita un volum prea mare de resurse al circuitului FPGA. Din acest motiv şi acest sistem utilizează reprezentări în virgulă fixă, pe 16 biţi.

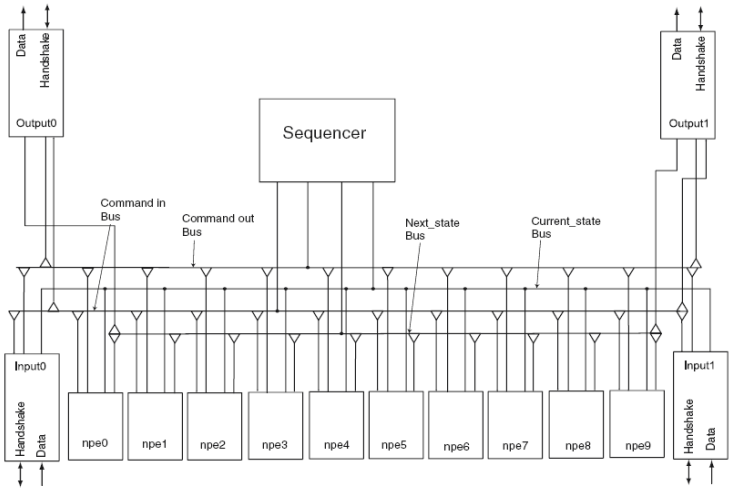


Figura 4.7 Diagrama bloc a topologiei procesorului neuronal SIMD

Arhitectura acestui procesor s-ar putea clasifica, ca unul SIMD (vezi Figura 4.7). Este compus dintr-o matrice de elemente de procesare (Processing Elements – PE), care operează asupra aceleiaşi instrucţiuni, date de un secvenţiator central, utilizând însă date locale. Stimularea procesorului se face prin două module de intrare seriale care pot citi datele de intrare în mod asincron. În mod similar, există şi două module de ieşire.

Neuronii şi sinapsele sunt implementate în elementele de procesare neuronale (PE's). Procesorul neuronal prezentat în articolul de (Izhikevich, 2004) a fost implementat cu 10 PE, fiecare emulând 120 de neuroni virtuali şi 912 sinapse. Perioada de reîmprospătare a procesorului a fost setată la 500 μs, frecvenţă generată de un numărător din modulul de secvenţiere (Sequencer).

Descrierea modulelor procesorului:

- **modulul de secvenţiere:** menţine performanţa de funcţionare în timp real şi coordonează activitatea tuturor PE ale procesorului, care operează concurent. Această coordonare este asigurată de comunicaţia desfăşurată pe un canal de control pe 2 biţi şi o magistrală de date de 16 biţi care conectează toate elementele de procesare.
- **modulul de intrare:** dispune de un port de intrare de 64 de biţi (multiplexat pe 384 canale de intrare) care poate fi conectat la conectoare fizice de I/O sau la o interfaţă internă corespunzătoare, utilizând logica circuitului FPGA gazdă. Linii de handshake facilitează operaţia asincronă şi permit comunicarea între diferite domenii a frecvenţei de tact. Cele 384 canale de intrare sunt transmise pe magistrala de date internă de 16 biţi pentru a fi stocate ulterior în memoria de stare.
- **modulul de ieşire:** are o structură şi funcţionare similară cu modulul de intrare, dar conţine un bloc de memorie RAM intern circuitului FPGA, pentru stocarea listei ieşirilor reţelei neuronale.

Elementele de procesare Neuronale conţin implementările hardware ale unui neuron şi o singură sinapsă (Figura 4.9). Informaţia contextuală despre cei 120 de neuroni "virtuali" şi cele 912 sinapse este stocată local în 4 bancuri de memorii RAM. Această informaţie este multiplexată secvenţial pe hardware-ul implementat cu o viteză foarte mare. O funcţionalitate interesantă, implementată de autori, este că o copie a stării întregii reţele este stocată local în fiecare PE, care reprezintă stimulul de intrare pentru neuronii/sinapsele virtuale. Starea reîmprospătată a fiecărui neuron din PE este de asemenea stocată în memoria locală a stării următoare, care va fi transmisă ulterior celorlalte PE-uri.

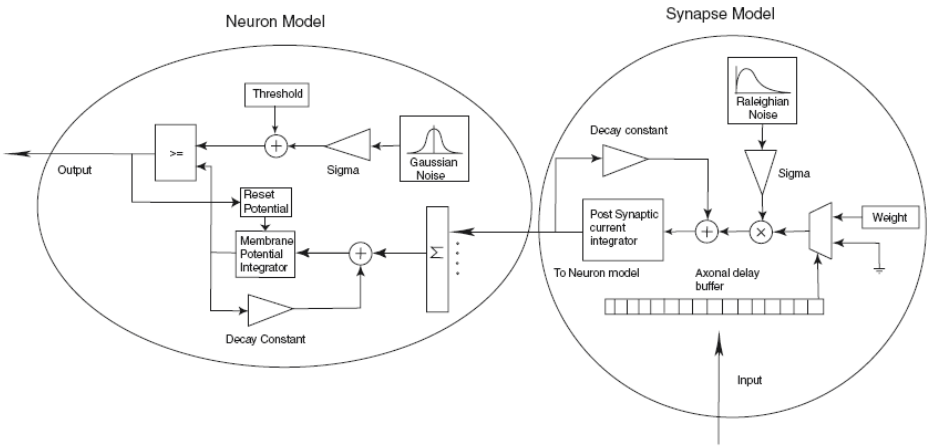


Figura 4.9 Diagramele bloc ale unui neuron şi a unei sinapse din SIMD

Circuitul FPGA utilizat a fost Xilinx Virtex-II (XC2V1000-4), cu 1 milion de porţi logice echivalente, funcţionând la o frecvenţă de tact de 50 Mhz, situate pe placă de dezvoltare Celoxica RC200, pe care s-au implementat 1200 de neuroni.

În general, pentru a putea estima dacă o reţea neuronală se poate implementa hardware să funcţioneze în timp real – ceea ce este o măsură a activităţii maxime – este necesar a se lua în calcul dimensiunea reţelei, determinându-se apoi rata de împrospătare necesară. Aceasta mărime poate fi enunţată a fi *cota de activitate a reţelei* adică, numărul maxim de evenimente impuls/neuron/perioadă de reîmprospătare. În cazul de faţă, dacă ar fi implementaţi 7000 de neuroni, ei ar genera o activitate pulsatorie de 175 de evenimente pe o perioadă de reîmprospătare, reţeaua putând fi categorizată ca necesitând o cotă de activitate de 0.025, pentru ca modalitatea de procesare implementată să poată funcţiona în timp real. Reţeaua implementată, de 1200 de neuroni, are momente de activitate de peste 30 de evenimente de impulsuri pe perioadă de reîmprospătare. Autorii susţin că procesorul prezentat a fost conceput să menţină performanţa de funcţionare în timp real până la o cotă de activitate de 1, altfel spus, performanţa procesorului nu depinde de activitatea reţelei.

4.5. Implementarea cu circuit FPGA a modelului neuronal propriu dezvoltat, validat prin simulare software

Modelul neuronal prezentat în subcapitolul 2.4 a fost conceput în aşa fel, încât să poată fi utilizat în aplicaţii în timp real. Implementarea practică acestui model în hardware, a fost bazată şi pe rezultatele simulării software prezentate la subpunctul 2.6.

Acest model exploatează bine posibilităţile oferite de sistemul de dezvoltare FPGA, asigurând o funcţionare complet paralelă reţelelor neuronale construite. Totodată, datorită realizării speciale a algoritmului de învăţare de tip Hebb, am reuşit implementarea de neuroni pulsativi (spiking neurons) cu circuite digitale ***fără module multiplicatoare.*** Multiplicarea seriilor de impulsuri este efectuată de numărătoare şi modulele de control ale acestora.

Iniţial, am realizat circuite simple, testând dotările sistemului de dezvoltare FPGA utilizat (Ashenden, 1990) (Cîrstea, Dinu, & Nicula, 2001), cum ar fi numărătoare, unităţi logice şi aritmetice, pe care le-am utilizat ulterior în implementarea modelului neuronal.

Proiectarea neuronului realizat în FPGA a fost divizată în două faze – care vor fi prezentate într-un capitol următor – şi anume, proiectarea modulului sinaptic şi proiectarea corpului celular, adică al somei.

În continuare se va prezenta sistemul de dezvoltare FPGA utilizat şi procesul proiectării-implementării hardware.

4.5.1. Descrierea sumară a sistemului de dezvoltare FPGA

4.5.1.1. Structura plăcii de dezvoltare

Părţile componente cele mai importante ale plăcii de dezvoltare FPGA, XESS XSA100, utilizate sunt (Figura 4.10):

- XC2S100 Spartan II FPGA - circuit integrat logic programabil (FPGA XC2S100 Spartan II cu 100 de mii de porţi logice echivalente 144 pini QFP)
- XC9572XL CPLD – circuit integrat configurabil care face legătura între portul paralele al calculatorului personal şi celelalte componente ale plăcii XSA
- Oscilator Dallas DS1075 – Oscilator programabil, responsabil pentru furnizarea semnalului de tact principal plăcii XSA cu intermediul CPLD

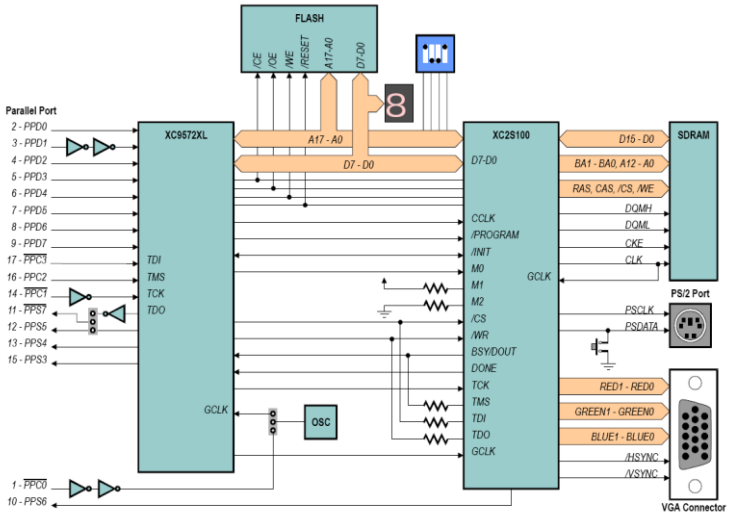


Figura 4.10 Schema arhitecturală a plăcii de dezvoltare FPGA XSA100, produsă de firma XESS (SUA), cu un circuit FPGA Xilinx XC2S100

- Flash RAM: 256 Kbyte memorie Flash pentru salvarea de date şi program de configurare FPGA
- SDRAM: 16 Mbyte memorie SDRAM, adresabil de către FPGA
- LED - afişor cu şapte segmente, pentru vizualizare variabile în cursul procesului de debug al circuitului realizat în FPGA
- Parallel Port (interfaţă paralelă): interfaţa prin care se încarcă biţii de configurare FPGA iar apoi utilizabilă pentru eventualele date şi/sau semnale de stare/control.
- Port PS/2 – port de conectare mouse sau tastatură PS/2 la placa XSA
- Port VGA – Se poate conecta un monitor VGA la placa XSA

Oscilatorul programabil Dallas DS1075 furnizează semnalul de tact pentru circuitele integrate FPGA şi CPLD. Frecvenţa maximă a DS1075 este de 100MHz, care se poate diviza, obţinând frecvenţe de 50 MHz, 33,3 Mhz, 25 MHz.... 48.7KHz. Semnalul de tact este conectat la o intrare dedicată a circuitului CPLD care o transmite spre FGPA.

Comunicaţia între PC şi placa XSA se desfăşoară prin portul paralel (PP). Linia de control C_0 a acestui port este direct conectată la oscilatorul DS1075 şi este utilizată pentru programarea divizorului acestuia. Bitul de stare S_6 al PP realizează o conexiune directă între circuitul FPGA şi calculator. Restul de 15 semnale ale PP sunt controlate de circuitul CPLD. Programarea circuitului CPLD se efectuează prin trei linii de control ale PP (C_1-C_3), C_1 furnizează un semnal de ceas de sincronizare, prin C_2 se transmit datele seriale iar prin C_3 starea circuitului CPLD. Starea CPLD-ului se poate citi prin bitul de stare S_7.

Prin biţii registrului de date al PP se pot trimite date spre FPGA, iar din acesta se pot transmite semnale spre calculator prin biţii rămaşi liberi în registrul de stare. Producătorul a furnizat câte un program utilitar pentru testarea şi plăcii XSA şi încărcarea biţilor de configurare FPGA respectiv CPLD.

4.5.1.2. Elementul central al plăcii XSA100, circuitul FPGA Xilinx XC2S100

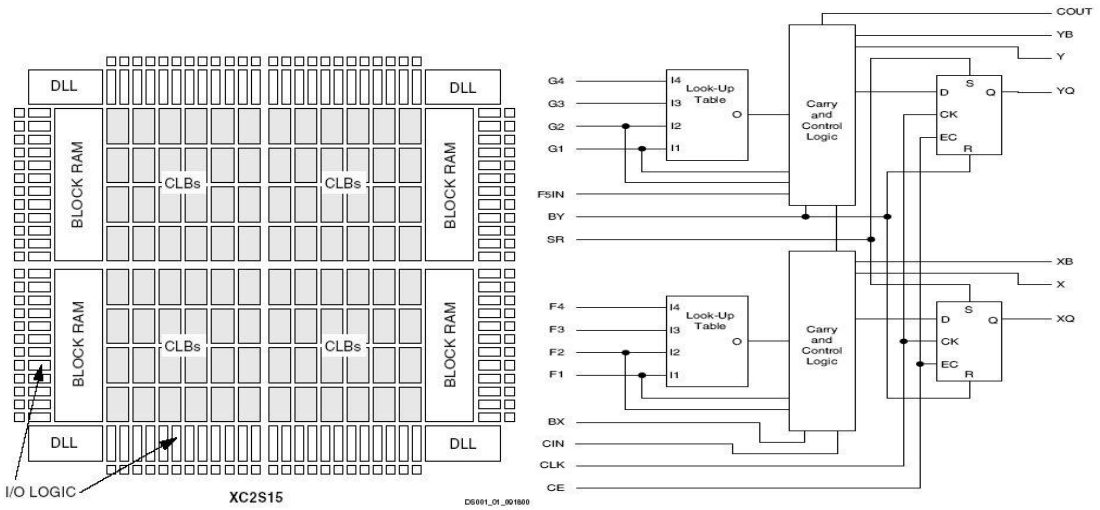


Figura 4.11 Structura internă a circuitelor FPGA din familia Spartan (stânga), părţile componente al unui bloc logic programabil (dreapta) (CLB)

Circuitul Xilinx XC2S100 este membru al familiei Spartan-II FPGA. Circuitele FPGA din această familie au o structură uşor de înţeles şi de utilizat, precum se poate urmări şi în Figura 4.11.

Această arhitectură de bazează pe *Blocuri Logice Programabile (CLB)*, care sunt înconjurate de Unităţi Logice Configurabile de Intrare/Ieşire (I/O Logic). Ca elemente arhitecturale se mai pot aminti blocuri RAM interne. Toate aceste elemente constructive sunt conectate printr-o magistrală de control extrem de flexibil programabilă, care este realizată pentru a asigura un număr infinit de procese de reconfigurare. Şirul de biţi, care determină configurarea acestor conexiuni între elementele de bază ale FPGA, se poate descărca prin porturi seriale, paralele sau dintr-o memorie PROM externă. Un aspect important este faptul că aceste circuite sunt capabile să funcţioneze la frecvenţe de până la 200 MHz, ceea ce permite realizarea unor reţele neuronale artificiale foarte rapide cu ajutorul acestora.

4.5.1.3. Mediul de dezvoltare software VHDL Xilinx ISE Webpack

Firma Xilinx, producătorul circuitului FPGA utilizat, pune la dispoziţia utilizatorilor acest mediu gratuit de dezvoltare al proiectelor VHDL. Acest proces de dezvoltare, cu ajutorul programului Xilinx Webpack 6.3 se compune din mai mulţi paşi, după cum arată şi Figura 6.3.

În primul pas se descrie circuitul logic de realizat cu ajutorul unui limbaj HDL (hardware description language) ales (VHDL sau VERILOG) sau se

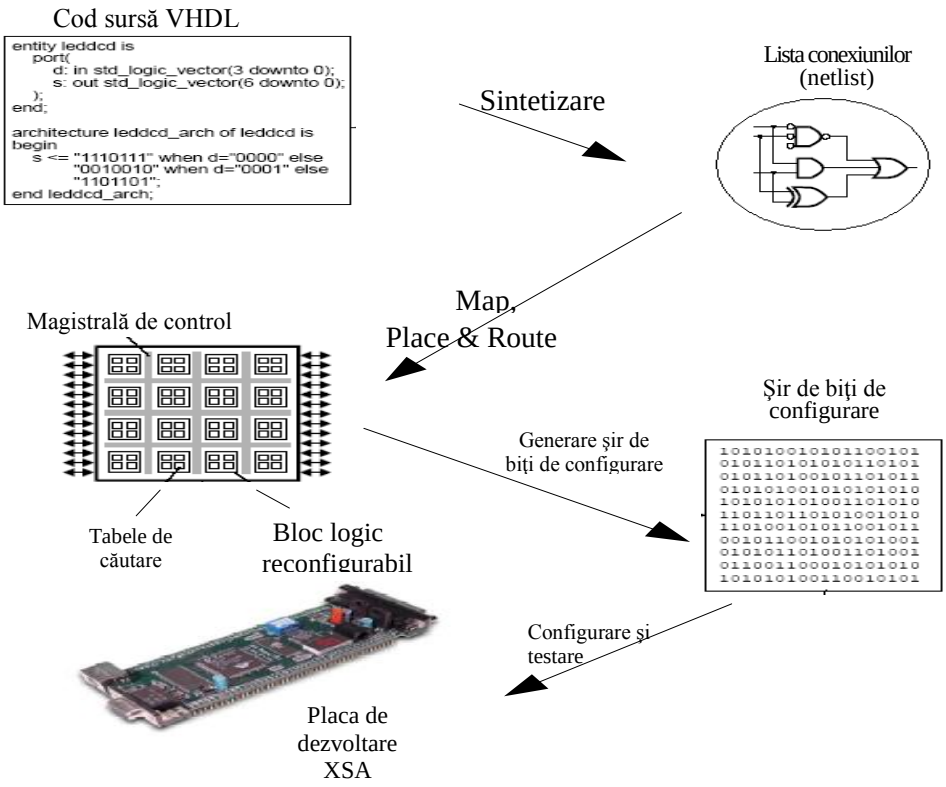


Figura 4.12 Fazele procesului de proiectare FPGA în mediul Xilinx Webpack

realizează o schemă cu ajutorul utilitarului grafic inclus

Programul de sintetizare logică translatează fişierul sursă scris sau schema desenată în formă de "*netlist*". Un fişier netlist conţine descrierea tipurilor de elemente de bază utilizabile pentru realizarea unui circuit care se comportă conform descrierii date şi a conexiunilor dintre aceste elemente.

Utilitarul de implementare (Implementation Tool) realizează o hartă a acestor elemente, pe baza celor aflate efectiv în circuitul FPGA şi a conexiunilor posibile între acestea. Blocurile logice programabile din care se compune XC2S100 sunt de fapt formate din subansamble ca tabele look-up (LUT) care pot realiza operaţii logice. Un utilitar numit Mapping Tool grupează porţile logice descrise de fişierul netlist, în aşa fel ca să se poată implementa cu cât mai puţine LUT-uri. În fine utilitarul de aşezare *Place and Route* asignează aceste grupuri de porţi logice unor CLB-uri, deschizând sau închizând "comutatoarele" corespunzătoare în matricea de comandă a magistralei de conexiuni a circuitului FPGA, conectând porţile logice în modul descris în program.

Din această matrice se genrează un fişier binar (şirul de biţi de configurare).

Odată cu descărcarea acestui fişier în circuit, proiectul realizat începe să funcţioneze. Toate aceste utilitare sunt parte integrantă a mediului de dezvoltare Xilinx ISE WebPack 6.3.

4.5.2. Procesul de realizare practică a modelului neuronal, implementarea sinapsei

4.5.2.1. Fazele implementării sinapsei în FPGA

După cum am amintit, construcţia unui circuit în integratul FPGA se poate realiza în două feluri:

- Proiectare cu ajutorul circuitelor elementare secvenţiale şi combinaţionale oferite de mediul de dezvoltare.
- Proiectare prin descriere (algoritm) cu ajutorul codului VHDL

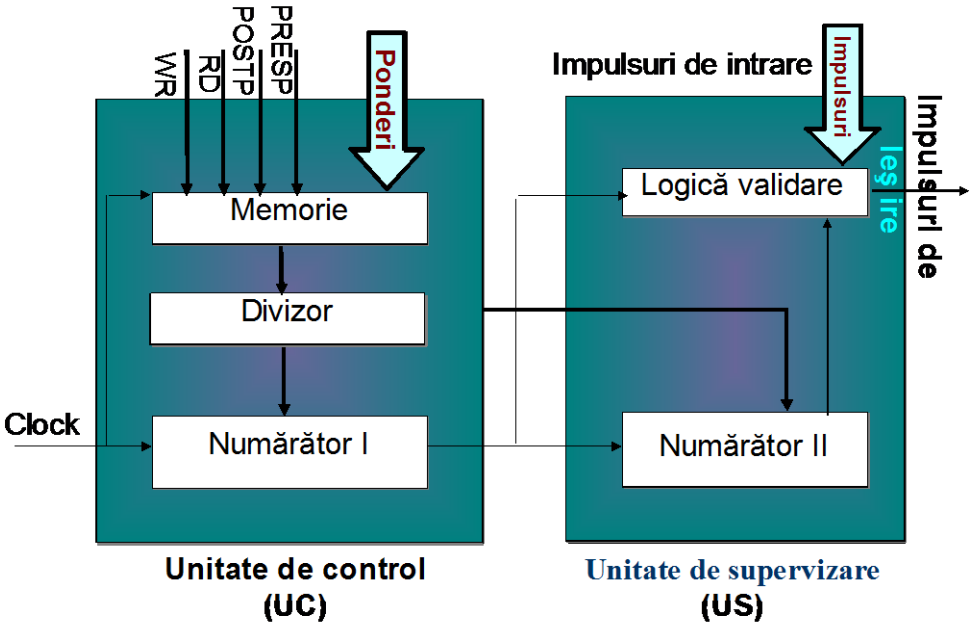


Figura 4.13 Schema structurală a sinapsei

În cursul proiectării a fost necesar împărţirea sinapsei în mai multe circuite funcţional diferite, şi definirea exactă a intrărilor şi ieşirilor acestora.

În concordanţă cu funcţionarea sinapsei biologice şi a modelului teoretic dezvoltat pentru un neuron pulsativ artificial, sinapsa electronică realizată preia impulsurile pre-sinaptice, le multiplică conform valorii ponderii actuale şi le transmite, ca impulsuri post-sinaptice, către corpul neuronal (somă). Impulsurile de intrare sosesc în sinapsă prin intrarea *spike_in* şi cele de ieşire o părăsesc prin ieşirea *spike_out*. Sinapsa va fi de fapt un circuit logic secvenţial, în care este necesar a sincroniza diferite părţi componente, cu ajutorul unui semnal de tact.

Valorile ponderale sunt stocate într-un registru de N biţi, ele trebuind iniţializate înainte, respectiv citite în timpul şi după procesul de învăţare. Aceste operaţii se efectuează cu ajutorul semnalelor RD şi WR (Figura 4.13), sincronizate pe semnalul de tact exterior.

Deoarece este vorba despre mai multe sinapse în cadrul aceluiaşi neuron, pentru a putea executa operaţiile menţionate anterior am implementat o logică de adresare şi o magistrală de ponderi. Selecţia unei sinapse se realizează setând semnalul CE la valoare logică ”1”, ceea ce înseamnă permiterea citirii sau programării ponderii acelei sinapse. În cazul în care CE este 0 logic, nu pot avea loc aceste operaţii, deoarece neuronul este în stare funcţională (reţeaua din care face parte neuronul respectiv este activă).

Scrierea ponderilor în sinapse se compune din următorii paşi:

1. Se selectează sinapsa setând CE=1,
2. Se activează semnalul WR,
3. Se aşează o valoare pe magistrala de ponderi (de exemplu prin portul paralel),
4. La frontul crescător al semnalului de ceas această valoare se va înscrie în registrul intern al sinapsei,
5. Se şterge CE, separând sinapsa de la magistrala de ponderi.
6. Operaţia de citirea a ponderilor din sinapse are următorii paşi:
7. Se selectează sinapsa setând CE=1,
8. Se activează semnalul RD,
9. La frontul crescător al semnalului de ceas sinapsa va aşeza valoarea ponderii sale pe magistrala de ponderi,
10. Se citeşte ponderea prin PP al PC,
11. Se şterge CE, separând sinapsa de la magistrala de ponderi.

Conform modelului utilizat, învăţarea se efectuează pe baza IA pre- şi post-sinaptice. Rezultă, că sinapsa trebuie să fie dotată cu două intrări care să semnaleze valorile acestor impulsuri.

Din punct de vedere funcţional, sinapsa se împarte în:

- ✓ Unitate de control
  - o Stochează valorile ponderii într-un registru intern,
  - o Asigură citirea şi scrierea ponderilor,
  - o Recepţionează semnalele impulsurilor pre- şi post-sinaptice utilizate în învăţare,
  - o Conţine un circuit divizor realizat din porţi logice, care calculează partea întreagă a câtului divizării frecvenţei de tact cu valoarea ponderii
  - o Conţine un numărător care divizează frecvenţa de bază (tact).
- ✓ Unitate de supervizare
- ✓ Numărător
- ✓ Logică de validare.

Divizorul calculează valoarea cu care trebuie divizată frecvenţa de bază, pentru a realiza un număr de impulsuri de ieşire proporţional cu ponderea sinapsei.

Numărătorul unităţii de supervizare va număra impulsurile de ieşire, iar dacă acest număr a atins valoarea ponderii atunci interzice emiterea altor astfel de impulsuri mână la sosirea unui nou impuls pre-sinaptic.

Deoarece resursele hardware ale FPGA sunt finite, ordinul de mărime al unor variabile a fost limitat. În faza iniţială a proiectării, valorile ponderilor au fost reprezentate pe patru biţi, ceea ce înseamnă, că sosirea unui impuls de intrare poate determina emiterea a maxim cincisprezece impulsuri post-sinaptice.

Multiplicatorul de impulsuri implementat funcţionează după idea următoare: două numărătoare conectate în serie, divizează frecvenţa de bază cu N_1*N_2, în aşa fel, încât frecvenţa semnalului de tact de ieşire să rămână constantă. Valoarea numărătorului N_2 este definită de pondere, deci cunoscând frecvenţa de bază putem calcula valoarea N_1.

Semnalul de tact de intrare este de 32 de ori mai rapid decât cel de ieşire. Impulsurile de ieşire nu sunt distribuite în mod absolut liniar. Liniaritatea completă putea fi atinsă, dacă tactul de intrare ar fi cu mult mai mare decât cel de ieşire şi ar fi cel mai mic multiplu comun al valorilor 2,3,5,7,9,11,13. Am renunţat la această cale, deoarece astfel, numărătorul N_1 este mai simplu de implementat şi utilizează mai puţine resurse.

4.5.2.2. Descrierea detaliată a funcţionării sinapsei

Valorile ponderilor sunt înscrise în Registrul unităţii de control. De aici, aceste valori vor ajunge la intrarea numărătorul 1, prin circuitul de divizare. Circuitul divizor realizează o împărţire întreagă a frecvenţei de bază (predefinită) cu valorare ponderii. Astfel am aflat la al câtelea front crescător al semnalului de ceas va trebui să generez impuls de ieşire.

Numărătorul unităţii de control va număra de la zero până la valoarea dată de divizor. Dacă a atins valoarea aceasta, atunci va emite un impuls de durată egală cu perioada semnalului de tact, se va reseta şi va reîncepe numărarea la fronturile pozitive de tact.

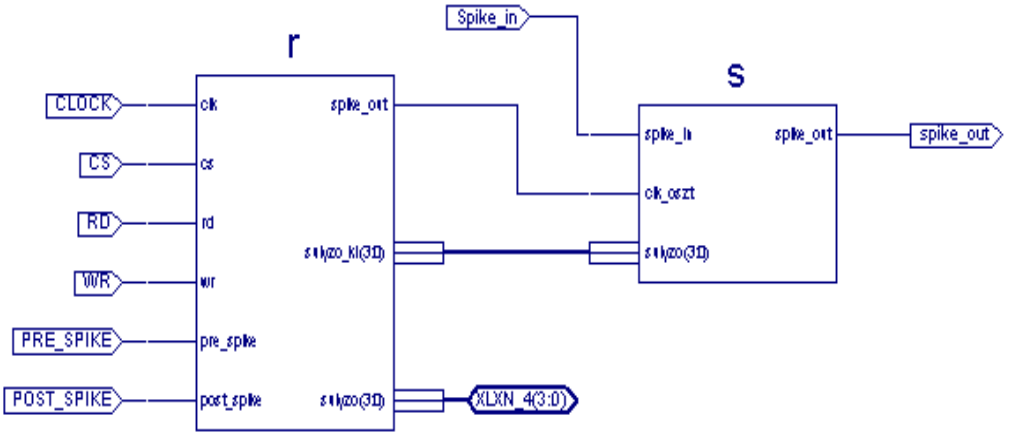


Figura 4.14 Schema de conectare interioară a sinapsei

Numărătorul unităţii de supervizare numără de la zero, la fiecare front crescător al impulsurilor emise de unitatea de control, până la valoarea ponderii. Acest numărător este resetat şi validat de un impuls pre-sinaptic. Dacă acest numărător a atins valoarea dată de pondere, atunci se opreşte şi prin logica de supervizare împiedică emiterea altor impulsuri post-sinaptice, până la sosirea unui nou impuls pre-sinaptic.

Modificarea ponderii duce la modificarea valorii limită a numărătorului 1. Astfel pentru o pondere mare impulsurile de ieşire din acest numărător vor fi mai dense precum şi cele post-sinaptice, iar pentru pondere mică ambele vor fi mai rare.

Unitatea de control (r), precum şi cea de supervizarea (s) a sinapsei a fost scrisă în cod VHDL, din care am generat câte un modul, utilizat în schema de conectare din Figura 4.14.

4.5.2.3. Îmbunătăţirea sinapsei

După efectuarea de teste cu implementarea sinaptică prezentată, am ajuns la concluzia, că necesită unele modificări, pentru a îmbunătăţi capacitatea de învăţare a reţelei.

Imaginile din Figura 4.15 şi Figura 4.16 prezintă implementarea îmbunătăţită a sinapsei. Se poate observa, că a apărut un modul de memorie, care are ca scop menţinerea stării semnalului pre-sinaptic de-a lungul procesului de învăţare. Procesul de învăţare începe cu frontul crescător generat de apariţia unui IA pre-sinaptic şi durează un număr de cicluri de tact date de valoarea ponderii. Reprezentarea ponderilor a rămas neschimbată, pe patru biţi, deoarece experimentele efectuate, după cum se va vedea mai târziu, au arătat că această precizie este suficientă pentru ca RNP construită să poată rezolva problema propusă. Rezultă, că modulul de memorie amintit, va trebui să menţină informaţia pe o durată de maximum 15 cicluri de tact, timp în care nu poate sosi alt impuls de intrare. După expirarea acestui interval, un semnal de control (Clearmem, Figura 4.15) va şterge conţinutul modulului de memorie.

În sinapsă am mai introdus un semnal de control suplimentar, care permite sau nu rularea procesului de învăţare, în funcţie de nevoia de a învăţa RNP sau a-i testa performanţele după învăţare.

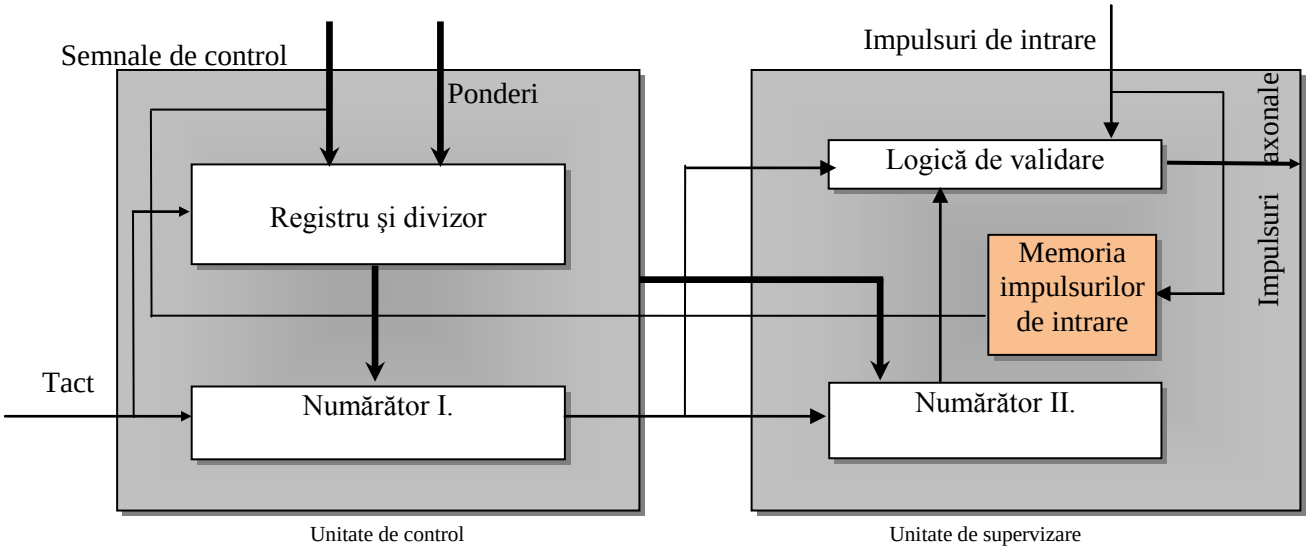


Figura 4.15 Schema de principiu a sinapsei

Figura 4.16 prezintă schema de conectare interioară a sinapsei modificate, unde *r* este UC, *s* este US, iar FDC modulul de memorie (bistabil D).

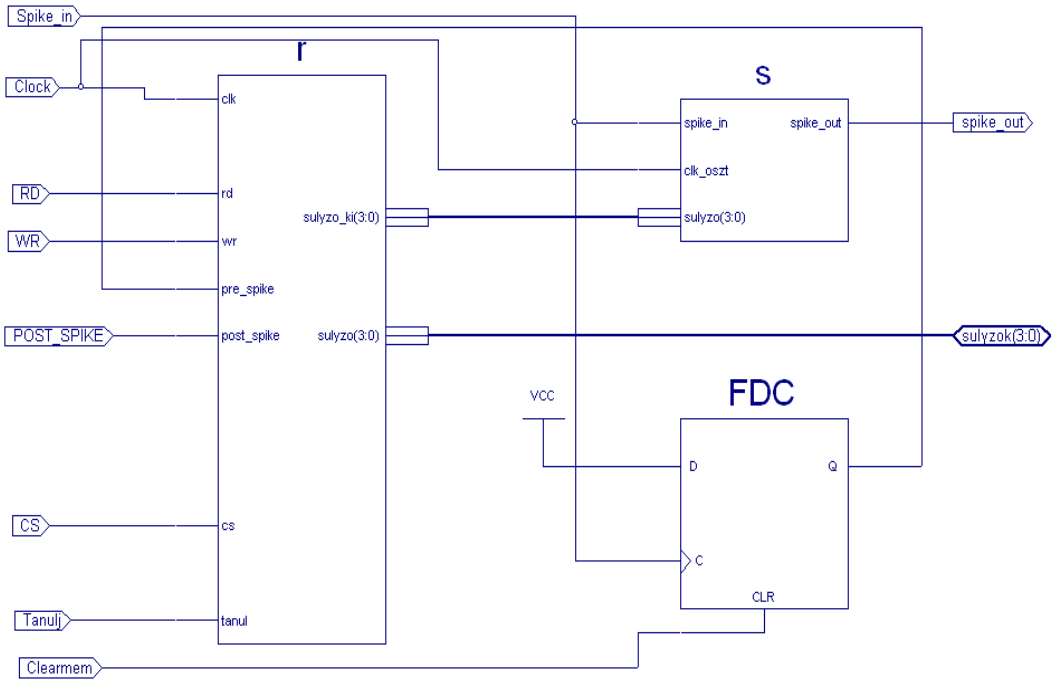


Figura 4.16 Schema de conectare a sinapsei modificate

4.5.2.4. Unitatea serială de intrare date (USID)

Această unitate asigură transferul valorilor ponderilor de la calculator la reţeaua aflată în FPGA, respectiv livrarea impulsurilor de intrare reţelei neuronale.

Unitatea serială de intrare date, citeşte semnalul de tact al transferului serial (CLOCK), biţii de date transmişi serial (DATA), respectiv semnalele de activare a sinapselor pentru citire/scriere (CEN – Chip Enable) şi semnalul de validare a impulsurilor de intrare (SPE – Spike Enable) de la portul paralel al calculatorului (Figura 4.17).

Datele sunt înscrise într-un registru de deplasare, prin linia de date la frontul pozitiv al semnalului de tact serial, respectând următorii paşi:

- Un bit de date este înscris pe linia de date,
- Este generat un front crescător pe linia CLOCK (Serial_clock),

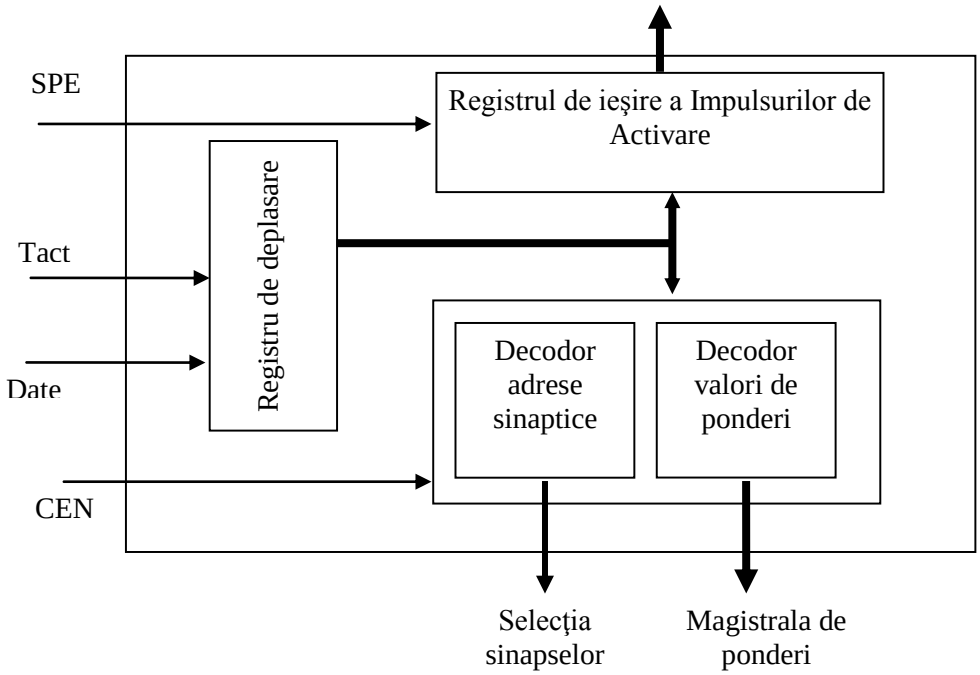


Figura 4.17 Schema constructivă a USID

Transferul unui bit se realizează într-un singur ciclu de tact. În Figura 4.18 se poate vedea o diagramă de timp a unei transmisii de patru pachete de date pe linia serială către registrul de deplasare din USID FPGA (ambele semnale sunt reprezentate în formă negată).

Acest proces este identic atât în cazul citirii/scrierii valorilor ponderilor, cât şi în cazul transmisiei impulsurilor de intrare şi a fost folosit numai pentru experimente în care s-a urmărit evoluţia reţelei fără pretenţii în ceea ce priveşte performanţele în viteza de prelucrare.

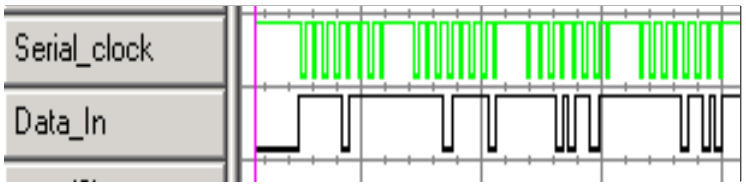


Figura 4.18 Transmisia serială de date către FPGA

În cazul transferului de ponderi, pe lângă valorile acestora se trimite şi adresa fiecărei valori. Astfel biţii cei mai puţini semnificativi conţin adresele (reprezentate pe 3 sau 4 biţi, în funcţiei de numărul de sinapse utilizate în neuronul respectiv), urmaţi de cei patru biţi reprezentând ponderile.

O logică din USID decodifică adresele şi activează numai o singură sinapsă pentru programarea ponderii, valoare acestei din urmă fiind de asemenea decodificată din registrul de deplasare în care a intrat şi înscrisă pe magistrala de ponderi.

Pentru ca ponderile să poată fi citite într-un moment ulterior, USID este deconectată de la magistrale de ponderi.

4.5.3. Fazele proiectării somei (al corpului neuronal)

Ca şi în cazul natural, unde soma este Unitatea Centrală de Prelucrare a neuronului biologic, în cadrul NP artificial, realizat în FPGA, circuitul care implementează soma este cel care efectuează prelucrarea impulsurilor de intrare. Circuitul somei se compune din patru părţi componente majore, după cum se poate vedea şi în Figura 4.19:

- Unitatea de citire şi modulare a impulsurilor de intrare sosite de la sinapse (SYNIN)
- Unitatea de calcul şi reîmprospătare a valorii potenţialului de membrană (MPOT)
- Registrul de stocare a potenţialului de prag încărcat (THRH)
- Circuitul comparator de generare a impulsurilor de activare a neuronului (AXON)

Prima versiune testată a neuronului pulsativ artificial a fost implementată cu opt sinapse încorporate, deci un astfel de neuron putea fi conectat la cel mult opt neuroni învecinaţi prin sinapsele sale. Pe baza unor calcule efectuate, a parametrilor neuronilor naturali şi a estimării necesităţii de stocare a informaţiei a algoritmilor utilizaţi în viitor, am decis reprezentarea valorii potenţialului de membrană pe opt biţi. Această decizie a fost influenţată de către considerentul economisirii resurselor sistemului de dezvoltare FPGA, în vederea posibilităţii realizării de reţele mai mari cu ajutorul acesteia.

Impulsurile post-sinaptice, care ajung la somă sunt modulate de unitatea de intrare a acesteia, cu un anumit factor de multiplicare (în primele experimente această valoare a fost setată la cinci sau zece), ele fiind transmise apoi unităţii MPOT. Pentru ca potenţialul de membrană să fie compatibil, adică complet comparabil cu potenţialul de prag (threshold potential), şi acesta din urmă a fost reprezentat pe opt biţi.

Neuronul pulsativ artificial astfel realizat va emite un IA atunci şi numai atunci, când circuitul de ieşire al somei (AXON-ul) – un comparator pe 8 biţi – va semnala, că potenţialul de membrană calculat a depăşit potenţialul de prag.

4.5.3.1. Descrierea detaliată a funcţionării unităţii centrale a somei, adică a modulului de calcul a potenţialului de membrană

Intrarea acestui modul, scris în VHDL, este de fapt – după cum se vede şi în Figura 4.19 – ieşirea modulului de citire a impulsurilor post-sinaptice. Modulul MPOT primeşte ca intrare şi semnalul de tact principal, semnalul de reacţie a stării axonale (dacă s-a emis IA) şi semnalul de autorizare a citirii valorii potenţialului de membrană (OE – Output Enable). Ieşirea acestui modul este valoarea nou calculată a potenţialului de membrană. Calcularea acestei valori se efectuează în felul următor:

La fiecare front crescător al semnalului de tact, modulul MPOT adună valoarea potenţialului de membrană stocat în ciclul anterior cu valorile ce se prezintă la intrare. În primele experimente, valoarea iniţială a fost setată la zero, iar potenţialul de repaus la valoare de zece.

Dacă potenţialul de membrană a depăşit cel de prag şi s-a emis IA, atunci această adunare nu se mai efectuează, ci se resetează potenţialul de membrană la zero, conform fazei de hiperpolarizare, prescrisă de modelul de urmat. Dacă valoare de intrare este egală cu zero, adică nu a fost prezent nici un impuls post-sinaptic, atunci potenţialul de membrană scade cu cinci (valoare variată în diferitele experimente) unităţi în fiecare ciclu de tact.

După cel de-al 32-lea ciclu de tact, dacă semnalul de reacţie de la ieşirea axonală este de valoare neschimbată, logic zero, potenţialul de membrană va scade până la potenţialul de repaus. Acest aspect este necesar pentru a implementa proprietatea neuronilor naturali care spune că ne se poate genera IA numai dacă într-un anumit interval de timp sosesc destule impulsuri post-sinaptice pentru a ridica potenţialul de membrană peste cel de prag.

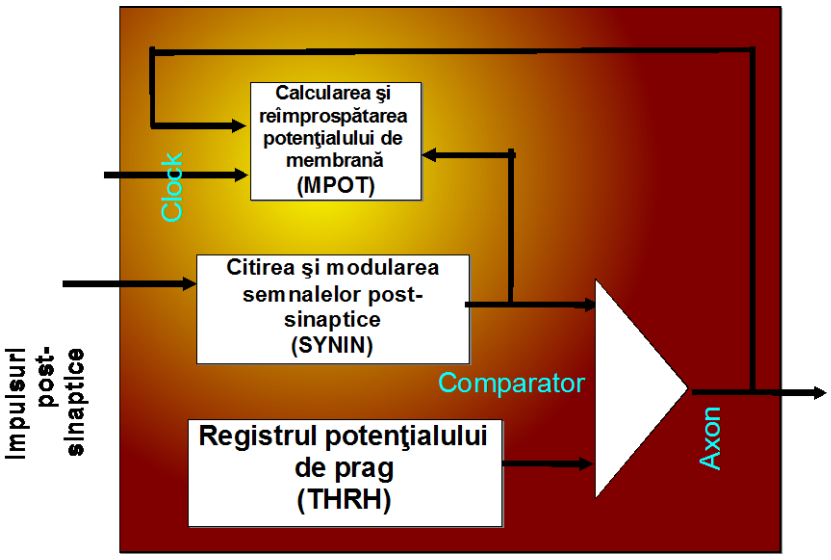


Figura 4.19 Schema constructivă a circuitului ce realizează soma neuronului

Figura 4.20 prezintă schema de conectare internă a somei, într-o fază intermediară a proiectării. În această fază modulul de calculare a potenţialului de membrană era compus din două părţi (Figura 4.20.) şi anume modulele Regiszt şi ADD8. Acesta din urmă a fost mai târziu inclus în Regiszt pentru a rezolva unele probleme de sincronizare ivite. Modulul mux_8_of_8x8 este de fapt un multiplexor 8 din 2x8 scris în VHDL, care serveşte la citirea sau reiniţializarea din exterior a valorii actuale a potenţialului de membrană. Modulul SR8CE este un registru de deplasare utilizat pentru iniţializarea potenţialului de prag, iar COMPM8 este circuitul comparator al cărei ieşire coincide cu axonul neuronului.

Figura 4.21 prezintă un circuit de testare a funcţionării modulului MPOT în care s-a utilizat o versiune a neuronului cu doar şase sinapse, pentru o transparenţă mai bună a testului. Tot în acest sens

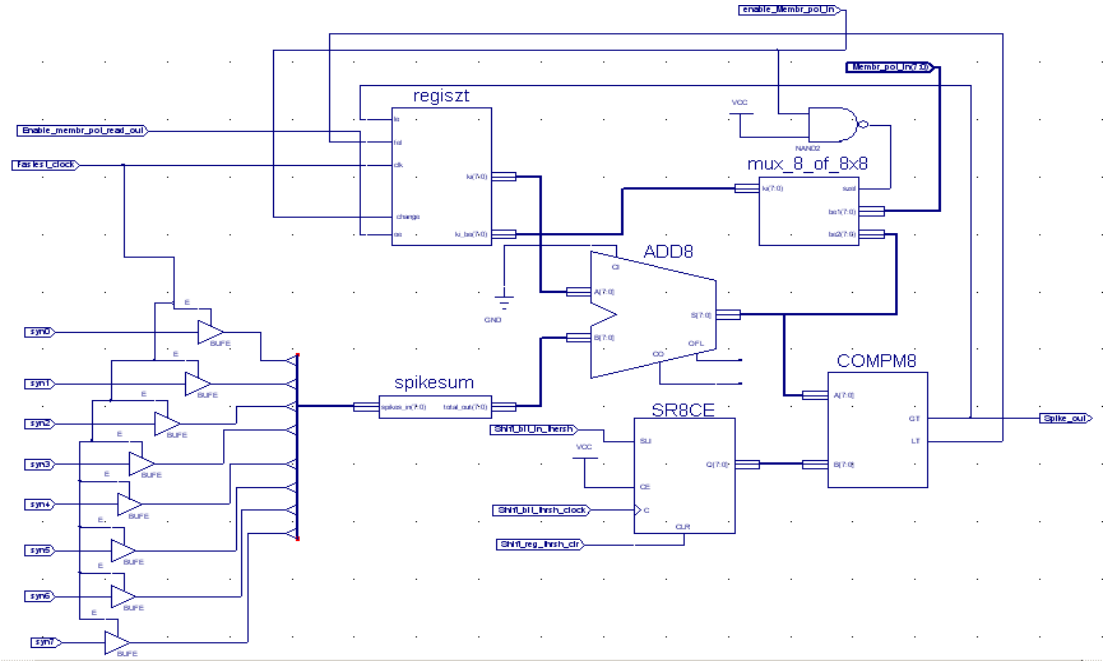


Figura 4.20 Modelul somei implementate, într-o fază intermediară a proiectării

am utilizat un comutator extern – aflat pe placa de dezvoltare – în locul reacţiei axonale, pentru a verifica corectitudinea efectuării operaţiilor propuse de către modulul MPOT. Am procedat la realizarea de astfel de circuite de testare, pentru a avea certitudinea funcţionării corecte a subansamblelor modelului neuronal, precauţie necesară datorită relativei complexităţi a implementării finale.

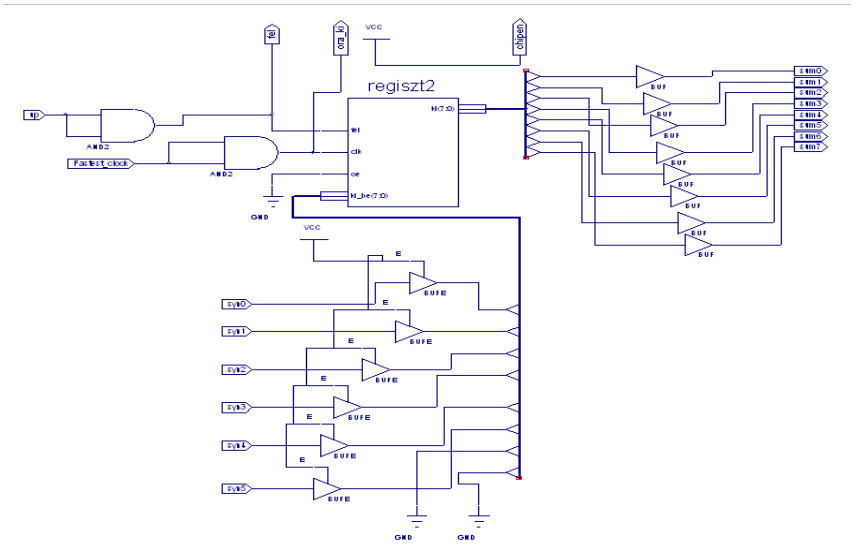


Figura 4.21 Circuitul de testare a modulului central al somei, MPOT

Semnalele din Figura 4.21. au fost eşantionate cu ajutorul unui Analizor Logic Tektronix cu 36 de canale şi arată variaţia potenţialului de membrană al unui neuron în cursul unui experiment de testare a funcţionării somei. Valoarea

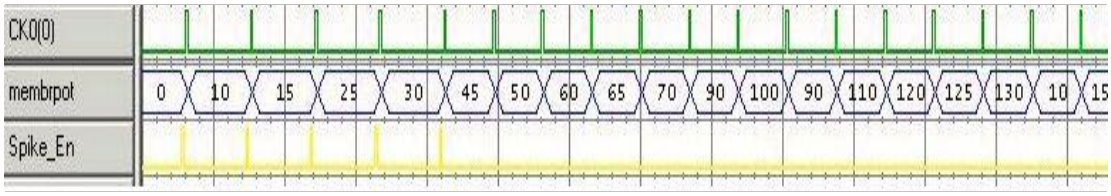


Figura 4.23 Valori ale potenţialului de membrană, eşantionate în timpul testării

reprezentată atunci, când sunt prezente impulsuri post-sinaptice iar dacă acestea dispar sau expiră intervalul de 32 de cicluri de tact, potenţialul scade treptat respectiv scade brusc la valoarea de repaus.

După efectuarea a numeroase astfel de teste am ales un model mai simplificat al somei (Figura

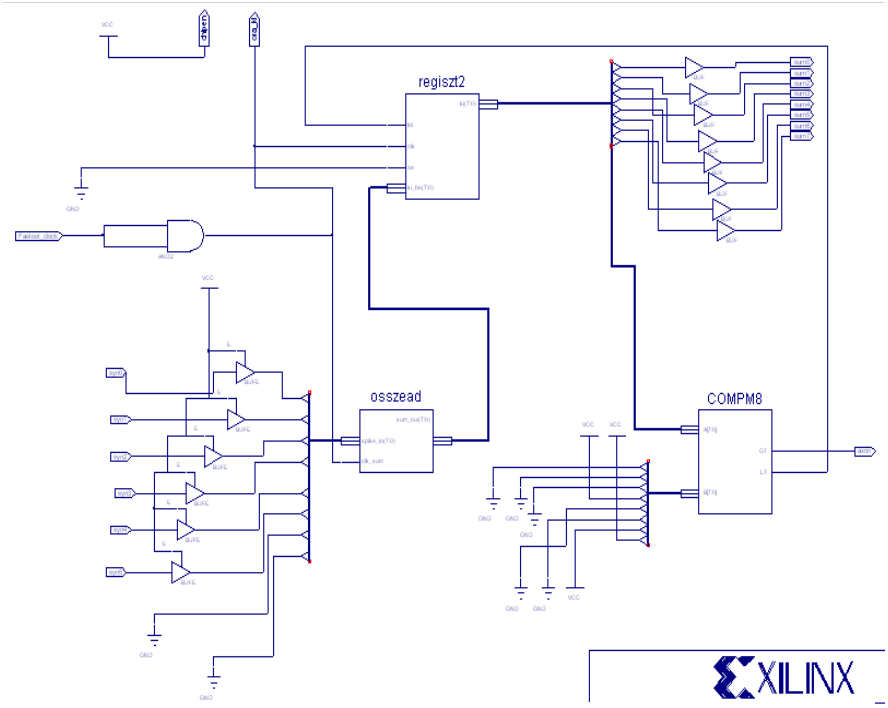


Figura 4.22 Schema somei implementate în neuronul pulsativ complet

4.22.) pe care am utilizat-o la implementarea completă a unui NP.

Diferenţa faţă de schema din Figura 4.20. este lipsa multiplexorului de citire a potenţialului de membrană, care se va utiliza ulterior, respectiv versiuni optimizate ale modulelor MPOT şi SYNIN. Valoarea potenţialului de prag a fost fixată la 200.

4.5.4. Stadiul actual de dezvoltare a somei

Forma finală a implementării somei, care a fost utilizată la realizarea RNP prezentate în subcapitolul următor are următoarele componente (Figura 4.24):

- Unitatea de citire şi modulare a impulsurilor de intrare sosite de la sinapse (SYNIN)
- Unitatea de calcul şi reîmprospătare a valorii potenţialului de membrană (MPOT)
- Registrul de stocare a potenţialului de prag încărcat (THRH)
- *Modulul de stocare a stării ieşirii axonale (AXOUT)*
- Circuitul comparator de generare a impulsurilor de activare a neuronului (AXON)

Se poate observa, că acest model s-a îmbogăţit cu un modul nou – faţă de cel din Figura 4.19, şi anume modulul de stocare a stării ieşirii axonale (AXOUT). Introducerea acestuia a fost necesară pentru ca intrările POSTSPIKE ale sinapselor, care este de fapt ieşirea axonală a aceluiaşi neuron, să semnaleze starea de activare (de aprindere) a neuronului pe întreaga durată a procesului de învăţare. Astfel s-a asigurat (urmând regula Hebb) creşterea ponderilor în cazul în care neuronii pre- şi post-sinaptice sunt simultan activi.

Pe de altă parte, s-a utilizat o configuraţie cu nouă sinapse pe neuron, aceasta fiind ideală pentru aplicaţia propusă.

Acest nou model al somei a fost rescrisă în întregime în VHDL şi implementată ca o singură componentă arhitecturală.

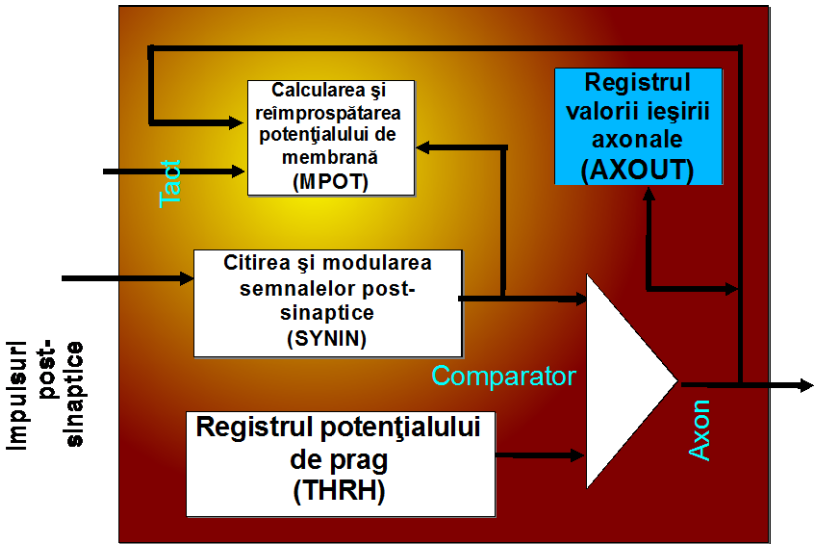


Figura 4.24 Schema de principiu a somei implementate hardware în FPGA

4.6. Prezentarea modelului neuronal implementat complet

După proiectarea, implementarea şi testarea diferitelor subansamble ale modelului neuronal am ajuns în faza realizării ei integrale în circuitul FPGA. Am implementat un neuron complet cu opt sinapse (ANEXA 2- Schema de implementare a unui neuron pulsat) care a fost testat în mai multe experimente. Furnizarea impulsurilor de intrarea către neuronul implementat s-a realizat prin portul paralel al unui calculator personal care rula un program de comunicare cu unitatea serială de intrare date (USID) prezentată la subpunctul 4.5.2.4.

În afară de impulsurile de intrare, a fost nevoie şi de furnizarea unor semnale de control prin portul paralel al PC, pentru a asigura adresarea sinapselor în vederea iniţializării ponderilor. Un pas delicat cu privire la importanţa acordării şi sincronizării din punct de vedere funcţional a sinapselor şi a somei, alegerea corespunzătoare a unor parametrii cum ar fi factorul de modulare din SYNIN, potenţialul de repaus şi cel de prag, respectiv valorile iniţiale ale ponderilor.

4.6.1. Programul de control al circuitului neuronal prin portul paralel al PC

Programarea ponderilor şi furnizarea impulsurilor de intrare se realizează de un program C++, care utilizează biţii D_0-D_7 ai registrului de date din portul paralel. În cazul ponderilor s-au utilizat patru biţi de date şi trei biţi de control. Valorile ponderilor însă sunt transmise serial, prin bitul D_0 şi sunt înscrise în registrul de deplasare din USID la frontul crescător al semnalului de tact serial furnizat prin bitul D_1. Semnalele de control pentru transmiterea ponderilor, CEN şi WR (D_2 şi D_3) sunt eşantionate de USID la frontul crescător al semnalului de tact principal (D_6), care sincronizează întreaga funcţionare a neuronului.

Dacă biţii de date transmişi serial sunt impulsurile de intrare, atunci este setat bitul corespunzător semnalului SPE (D_4), care permite trecerea intrărilor spre sinapsele adresate. Adresa sinapselor este transmisă similar, imediat înaintea impulsurilor de intrare.

4.6.2. Câteva rezultate experimentale de testare a NP implementat, cu diferite configuraţii parametrice

În Figura 4.25. este prezentat un grup de semnale măsurate cu analizorul logic, pe care se pot observa următoarele:

- ✓ Semnalul trasat cu linie verde (Serial-clock), este tactul transmisiei seriale, la al cărui front crescător se înscriu impulsurile de intrare ale neuronului. Se poate vedea, că la acest experiment am transmis impulsuri de intrare simultan pentru mai multe sinapse.
- ✓ Cu linei galbenă s-a trasat semnalul Spike-En, care în stare logică 1 validează înscrierea intrărilor în sinapse.
- ✓ Semnalele de culoare mov (Spikes-out[0÷7]) reprezintă ieşirile celor opt sinapse, emiţând un număr de impulsuri post-sinaptice egale cu valoarea ponderii sinapsei respective
- ✓ Cu linei albastră sunt reprezentate valorile potenţialului de membrană a neuronului, în diferite momente ale rulării testului

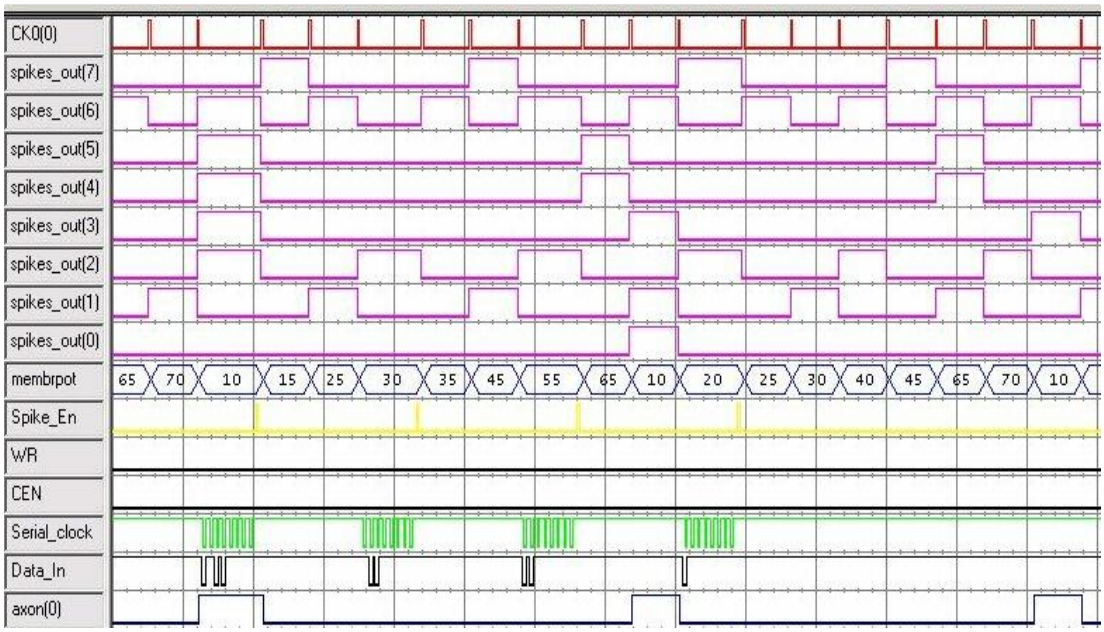


Figura 4.25– Impulsuri axonale într-un experiment cu potenţial de prag scăzut

Aici se poate observa, că la frontul descrescător al tactului principal (roşu, CK0) potenţialul de membrană scade cu de atâtea ori 5 unităţi (aceasta fiind valoare setată a factorului de modulare din SYNIN la acest experiment), câte sinapse emit impuls post-sinaptic în acelaşi ciclu de tact.
Totodată se pot observa momentele de activare a neuronului (emiterea de impulsuri axonale, semnalul axon(0)). Acestea apar la momentele în care potenţialul de membrană depăşeşte cel de prag (setat aici la 80). La frontul crescător al tactului CK0 din ciclul imediat următor, potenţialul de membrană scade la 10.

Figura 4.26. respectiv Figura 4.27. prezintă două segmente ale unor semnale eşantionate într-un alt test asemănător, al cărui parametrii au fost setaţi astfel:

În comparaţie cu cazul anterior, unde valorile ponderilor au fost iniţializate cu valori între 1 şi 3, aici am utilizat valori mai mari, aleator alese (ponderi[8]={4,8,14,6,7,5,9,10});

- ✓ Potenţialul de prag a fost setat la 143,
- ✓ Potenţialul de repaus era 10,
- ✓ Factorul de modulare SYNIN=5.

Pe aceste două figuri se poate urmări un ciclu complet. În Figura 4.26, se poate observa, cum după un IA emis, potenţialul de membrană intră în faza de hiperpolarizare, scăzând brusc la zero, sub

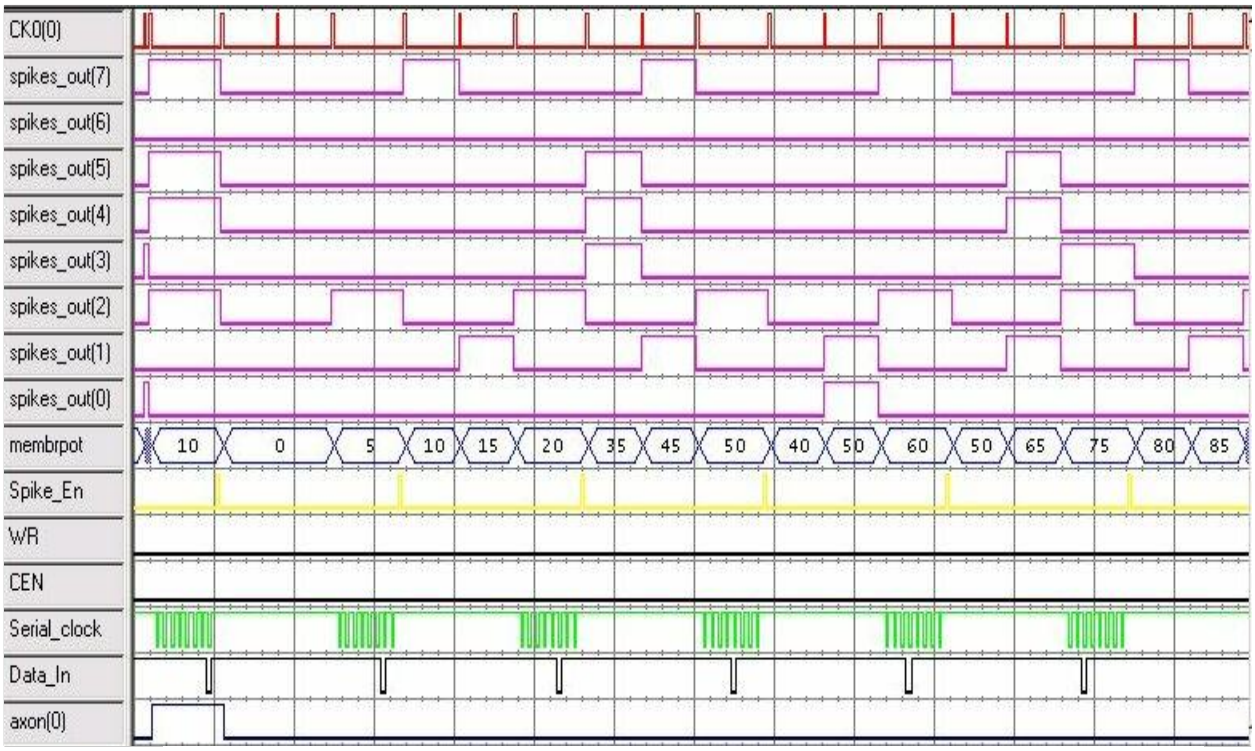


Figura 4.26 IA utilizând valori iniţiale mari ale ponderilor şi impulsuri de intrare distribuite – prima parte

valoarea de repaus. Apoi, în funcţie de numărul sinapselor care emit impulsuri post-sinaptice, potenţialul de membrană creşte cu cinci sau multiplu de cinci unităţi. Dacă într-un anumit moment, nici una dintre sinapse nu a fost activă, atunci această valoare scade cu zece unităţi în fiecare ciclu de ceas.

Partea a doua a măsurătorii este prezentată în Figura 4.27., unde se poate urmări, cum evoluează valoarea potenţialului de membrană, până când neuronul se aprinde din nou şi emite IA.

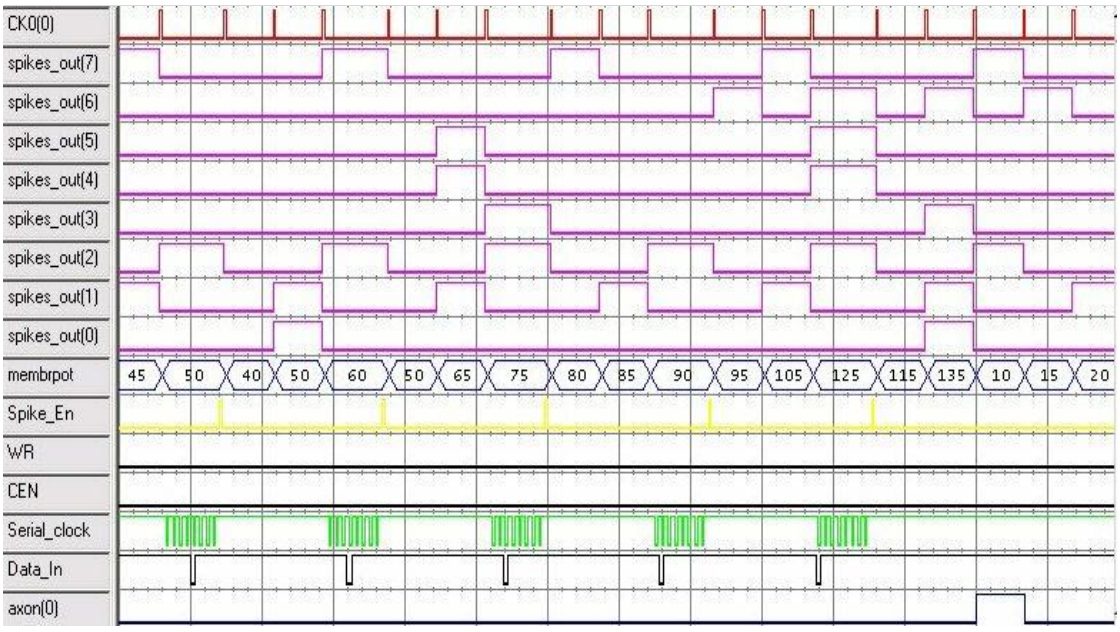


Figura 4.27– IA utilizând valori iniţiale mari ale ponderilor şi impulsuri de intrare distribuite – partea a doua

4.7. Implementarea unei reţele neuronale pulsative artificiale în circuitul FPGA. Rezultate experimentale

4.7.1. Implementarea unei RNP în circuit FPGA

După mai multe luni de experimentări, am decis să implementez neuronul pulsativ cu versiune de nouă sinapse. Cu ajutorul a numai doi astfel de neuroni, am reuşit să construiesc o reţea neuronală pulsativă artificială, capabilă de a diferenţia orice două modele de intrare – chiar şi în prezenţa zgomotului – prezentate într-o matrice de 3x3 puncte. Pentru a utiliza cât mai eficient resursele circuitului FPGA în care s-a efectuat implementarea, fără repercusiuni asupra performanţelor reţelei, modelul neuronal a fost puţin modificat. Astfel, potenţialul de membrană al somei şi potenţialul de prag au fost reprezentate pe şapte biţi, ponderile sinapselor rămânând reprezentate neschimbat pe patru biţi.
Topologia reţelei construite este prezentată de Figura 7.1. Ca şi la testarea neuronilor individuali, comunicarea cu calculatorul s-a realizat cu ajutorul portului paralel. Comunicaţia în sens direct (PC → FPGA) se desfăşoară pe şapte biţi iar cea inversă pe patru biţi. Câte un bit din cele disponibile în registrele de date şi stare ale portului paralel nu se pot utiliza ca funcţie I/O generală din cauza unor restricţii impuse de place de dezvoltare FPGA.

Am utilizat, deci, din nou, USID (Unitatea serială de intrări de date – un circuit inteligent de conversie a comunicării serial-paralele) puţin modificată pentru a comunica cu software-ul de comandă scris în C++ şi rulat pe PC. Cel mai puţin semnificativ bit al registrului de date din PP a fost asignat ca linie de transmisie date seriale, bitul următor este ceasul de sincronizare a transmisiei seriale, restul biţilor fiind semnale de control. Cu ajutorul acestor biţi s-a rezolvat atât iniţializarea ponderilor sinapselor din reţea, cât şi furnizarea modelelor de intrare acesteia. Cei patru biţi ai registrului de stare al PP îndeplineşte două sarcini diferite în funcţie de poziţia unui comutator aflat pe placa de dezvoltare XSA utilizată.

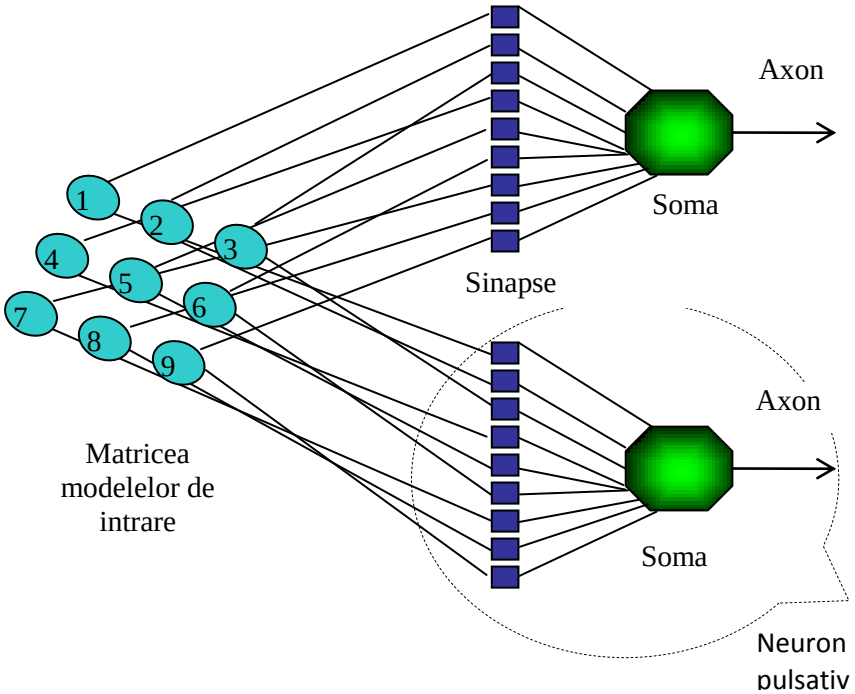


Figura 4.28 Arhitectura reţelei neuronale pulsative implementate

Dacă acest comutator este pe valorare logică 0, atunci aceşti patru biţi vor fi conectaţi la magistrale de ponderi a reţelei neuronale, astfel putându-se citi valorile tuturor ponderilor în orice moment al testării reţelei. In caz contrar, dacă comutatorul este pe poziţie logică 1, atunci doi dintre cei patru biţi reprezintă ieşirea axonală a celor doi neuroni pulsativi implementaţi în reţea. La transmiterea ponderilor un pachet trimis pe linia serială de date D_0 este formată din patru + cinci biţi corespunzători valorii ponderii respectiv adresei sinapsei respective. Aceste pachete sunt decodificate de USID şi cu ajutorul semnalelor de comandă CEN şi WR se adresează apoi se înscriu valorile în registrele interne ale sinapselor prin magistrala de ponderi a reţelei.

Algoritmul de desfăşurare a testului reţelei neuronale este următorul:

1. Programul de control (C++) rulat pe PC generează 18 valori aleatoare între 0 şi 15 ca valori de iniţializare pentru ponderile celor 2x9 sinapse ale neuronilor reţelei. Apoi aceste valori sunt

transmise şi înscrise la locul potrivit, timp în care reţeaua încă nu este funcţională, furnizarea semnalului de tact principal fiind sistată.

2. Programul de control calculează valorile binare pentru modelul de intrare care urmează. Fiecare punct întunecat din matricea de intrare corespunde unui bit de valoare logică 1 iar punctele inactive (albe) au valoare logică 0 (Figura 4.29). Acest şir de biţi astfel calculat este transmis apoi reţelei prin intermediul PP respectiv USID. Acesta din urmă distribuie valorile decodificate şi le plasează la sinapselor corespunzătoare ca impulsuri de intrare. Nici în această fază reţeaua nu funcţionează, din cauza lipsei semnalului de tact principal.
3. În acest pas, startul funcţionării este dat de primul front crescător al semnalului de tact pornit, care duce la generarea de un număr de impulsuri post-sinaptice egal cu valoarea iniţială a ponderii respective de către sinapsele care au primit impuls de intrare. Acestea ajungând la somă se modulează şi se însumează, modificând potenţialul de membrană al acesteia.

Paşii de la punctele 2 şi 3 se repetă de mai multe ori, în funcţie de setările efectuate asupra programului de control în experimentul respectiv. Repetarea modelelor de intrare se face în aşa fel încât probabilitatea de apariţie a ambelor modele să fie egală (De exemplu fiecare model se repetă la fiecare al doilea pas).

Rularea testului se întrerupe prin oprirea semnalului de tact, pentru a putea citi valorile ponderilor prin registrul de stare al PP, respectiv ieşirile axonale ale neuronilor (pentru a afla dacă au emis IA sau nu). Toate aceste date sunt salvate în fişiere pentru a putea fi prelucrate ulterior.
Testul se termină odată cu închiderea programului de control pe PC.

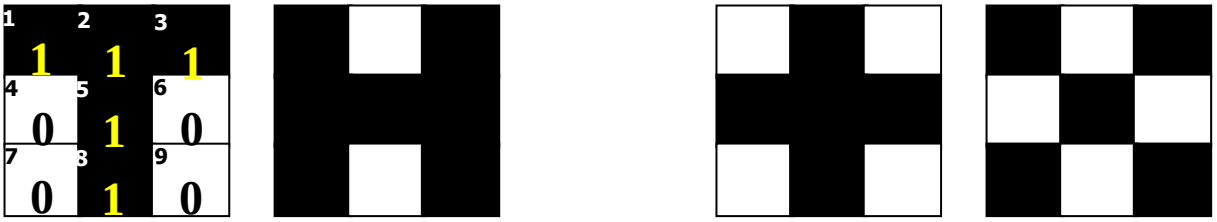


Figura 4.29 Perechile de modele de intrare, utilizate în cursul învăţării (notaţi ordinea şi valorile biţilor

Schema de conectare a RNP implementate, împreună cu circuitele auxiliare (convertorul serial-paralel, USID, modulul de sincronizare, unitatea de control a magistralei de ponderi) este prezentată în *ANEXA 3*- Schema de implementare a unei reţele neuronale pulsative, a acestei teze.

4.7.2. Descrierea algoritmului de învăţare supervizată implementat

Perechile de intrare-ieşire utilizate la învăţare au fost stocate în microcontroler. Reţeaua neuronală implementată poate funcţiona în două regimuri de lucru şi anume regim de învăţare şi regim de testare a reacţiei la intrări. Regimul de lucru este setat de semnalul *learn_or_test* iar selecţia algoritmului de învăţare este efectuată cu ajutorul semnalului *sel_learn*. De fapt regulile algoritmilor de învăţare sunt aceleaşi (regulile Hebb) numai interpretarea impulsurilor post-sinaptice şi a celor axonale sunt interpretate în mod diferit. Algoritmul Hebb implementat se poate rezuma astfel:

- ✓ Valorile ponderilor sinaptice se măresc în cazul prezenţei simultane a impulsurilor pre- şi post-sinaptice.
- ✓ Dacă numai un impuls pre- sau numai unul post-sinaptic este prezent, atunci valorile ponderilor sinaptice trebuie să scadă. În cazul învăţării nesupervizate, semnalele post-sinaptice proprii ale reţelei sunt conectate ca o reacţie în modulul de învăţare a sinapselor.

- ✓ În cazul în care nu este prezent nici impuls pre-sinaptic nici unul post-sinaptic, valoarea ponderii sinaptice respective rămâne neschimbată.

În cazul învăţării supervizate în loc de ieşirile proprii ale reţelei sunt introduse forţat aşa numite IA virtuale din exterior în modulele de învăţare ale sinapselor. Aceste IA virtuale vor fi de fapt ieşirile prescrise reţelei pentru modelul de intrare prezentat.

După cum a fost prezentat şi mai sus, intrarea este o matrice de 3x3 elemente. Scopul este de a antrena reţeaua în mod supervizat, în aşa fel încât să recunoască, de exemplu, literele T şi H prin activarea neuronului numărul 1 respectiv 2 la apariţia secvenţială a celor două litere la intrare.

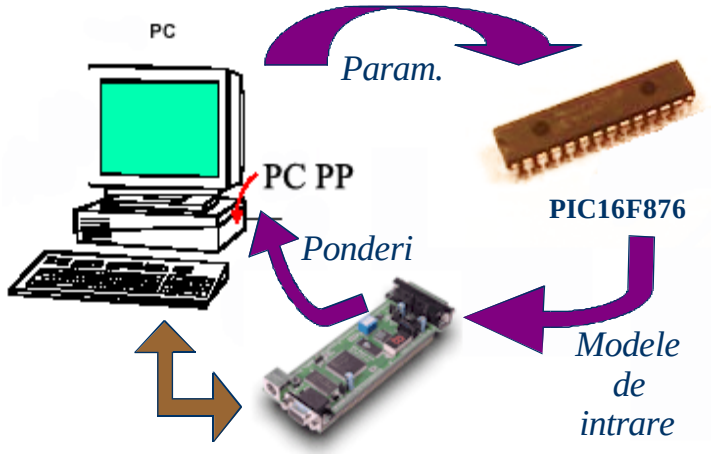


Figura 4.30 Comunicaţia dintre diferitele echipamente ale sistemului

Conectarea fizică a plăcii de dezvoltare FPGA XILINX XC2S100 cu mini-sistemul conţinând microcontrolerul – ca supervizor – PIC16F876 s-a realizat montând ambele pe placa de extensie XST, parte auxiliară a pachetului de dezvoltare FPGA. Realizarea comunicării de date dintre cele două plăci (intrări, ieşiri prescrise, setarea modului de lucru) s-a efectuat conectând porturile B şi C ale PIC16F876 la pinii corespunzători ai plăcii XSA (Figura 4.30).

Programul în limbaj de asamblare implementat pe microcontroler realizează următorul algoritm. În memoria de date a supervizorului s-au stocat valorile intrărilor pentru literele T şi H, respectiv ieşirile impuse pentru acestea. În mod asemănător, tot în memoria de date a microcontrolerului s-a stocat un set de valori de intrare de test, conţinând combinaţiile repetate ale literelor T şi H, alterate sau nu de zgomot. Programul rulat de supervizor selectează modul de lucru al reţelei (învăţare) şi tipul acesteia (supervizată) cu ajutorul liniilor de control ale circuitului din FPGA, *learn_or_test*, respectiv *sel_learn* (*ANEXA 3*- Schema de implementare a unei reţele neuronale pulsative). În ciclul de învăţare, acest program prezintă reţelei neuronale din FPGA, în mod alternant, combinaţiile pentru literele de învăţat (T=111010010, H=101111101), respectiv împreună cu acestea, valorile de ieşire impuse fiecăruia (T=10, H=01). La următorul front crescător de tact se setează semnalul *spike enable (SP)* la 1 logic, semnalând reţelei, că impulsurile de intrare sunt pregătite, apoi urmează 15 cicluri de tact în care reţeaua prelucrează aceste intrări, neuronii emiţând sau nu IA. Se procedează la fel pentru următorul caracter de prezentat.

4.7.3. Măsurători şi rezultate experimentale

Experimentele iniţiale au fost efectuate utilizând calculatorul prin PP pentru a furniza intrările circuitului din FPGA şi pentru a citit date de la aceasta. Scopul primordial în aceste cazuri a fost de a observa evoluţia reţelei în timpul procesului de învăţare, lucru uşor de realizat dat fiind faptul, că comunicare prin portul PP a rezultat într-o funcţionare relativ lentă – cu un tact de ~100 Hz – uşor de urmărit şi eşantionat cu ajutorul analizorului logic Tektronix cu 36 de canale. Evident această cale nu a fost cea care să permită o funcţionare în timp-real, ea îndeplinindu-şi însă ţelul de a verifica convergenţa algoritmului de învăţare. Rezultatele fără echivoc pozitive ale acestor experimente au determinat trecerea la pasul următor şi anume trecerea la un program de control mai rapid, prin utilizarea microcontrolerului PIC16F876. Acesta poate lucra la o frecvenţă maximă de 20 MHz. Astfel am reuşit să prezint reţelei câte un nou model de intrare la fiecare aproximativ 1,5µs (în doi paşi temporali, de 16 cicluri de ceas fiecare), aplicând algoritmul de învăţare nesupervizat.

Figura 4.31 respectiv Figura 4.32 prezintă semnale eşantionate cu analizorul logic, care arată reacţia celor doi neuroni, impulsurile axonale emise de aceştia, la cele două modele de intrare furnizate reţelei. Semnalul trasat cu culoare roşie, *CK0* – în Figura 4.31 – este semnalul de tact principal al sistemului. Se poate observa, că înainte de apariţia ciclurilor de tact pe acest semnal, semnalele

Serial_Clock şi *Data_In* desfăşoară o transmisie serială a valorilor de intrare. Ca efect al acestei intrări, impulsurile post-sinaptice generate, determină schimbări rapide de valori ale potenţialului de membrană (aceasta a fiind afişată numai pentru nunul dintre neuroni, semnalul *membrpot*). Într-un moment dat potenţialul de membrană depăşeşte cel de prag, deoarece neuronii şi-au acordat – în cursul antrenării – ponderile sinaptice pe câte una din modelele de intrare, astfel se emite un impuls de activare (*axon2(0)*). Urmează apoi transmiterea serială a următorului model de intrare la care răspunde în mod asemănător celălalt neuron.

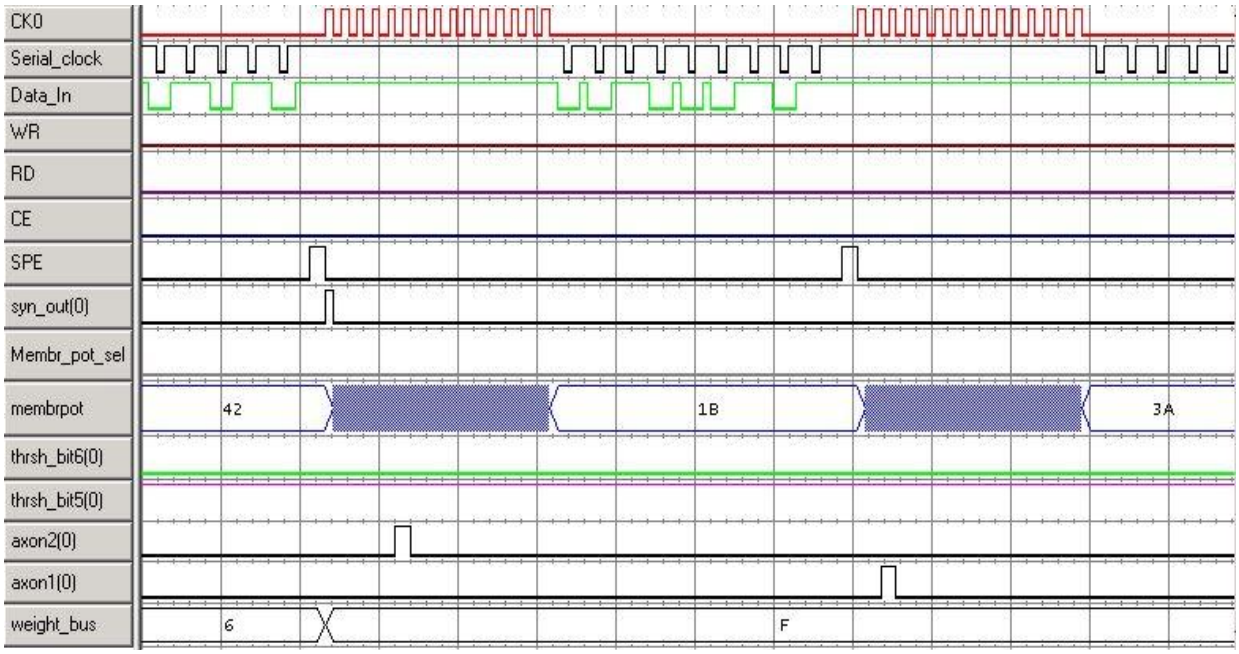


Figura 4.31 Transmisii seriale ale valorilor de intrare cu răspunsurile axonale date de neuroni la acestea

Figura 4.32 marchează un alt moment important al experimentului şi anume procedeul de citire a valorilor ponderilor după două intrări transmise reţelei. Aceste valori apar pe magistrala de ponderi (*weight_bus*) după ce au fost activate semnalele *RD* (semnalarea unei operaţii de citire) respectiv *CE* (activare sinapselor pentru scriere/citire) şi pe linia serială s-a transmis adresa sinapsei citite în fiecare ciclu.

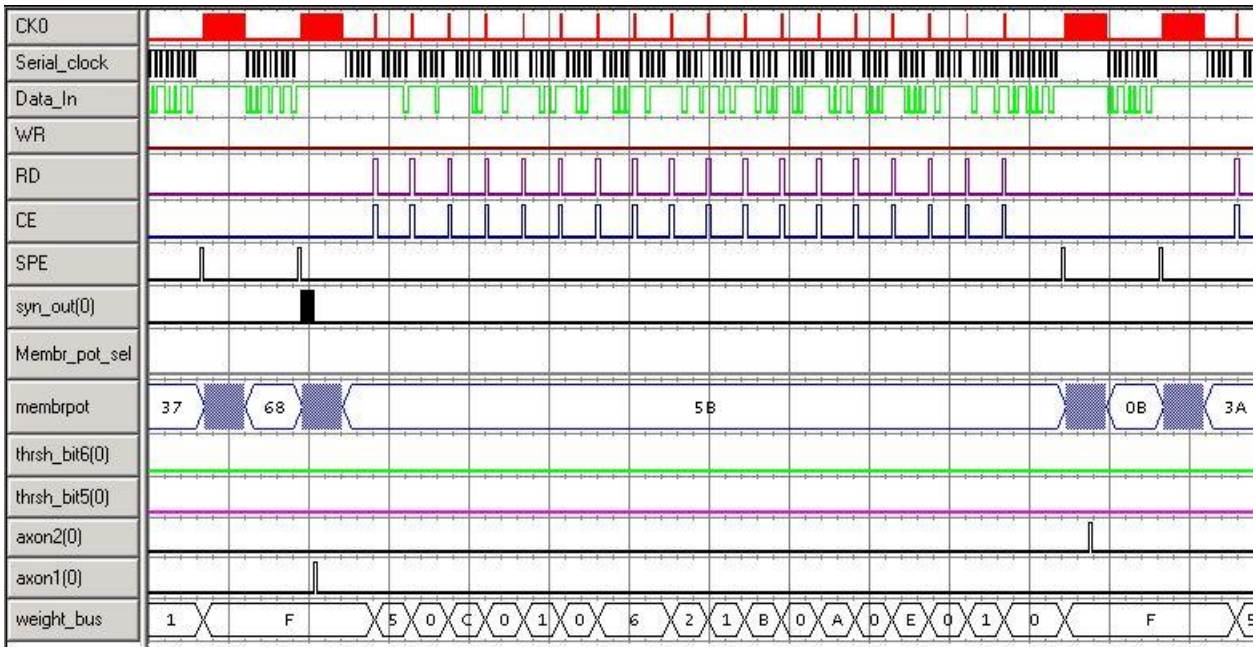


Figura 4.32 Citirea valorilor de pondere sinaptică (hexadecimal) între două faze de transmitere a perechii de modele de intrare

4.8. Analiza performanţelor reţelei neuronale neuromorfe implementate

4.8.1. Rezultatele obţinute de reţeaua neuronală implementată în hardware

Unul din rezultatele majore ale reţelei neuronale neuromorfe pulsative (spiking neural network) implementate în circuitul FPGA este atingerea funcţionării în timp real. De asemenea este de notat faptul că reţeaua este capabilă să rezolve foarte rapid şi cu o eroare minimă, problema propusă, deşi parametrii ei au o precizie relativ mică. Toate acestea sunt adevărate şi în cazul unor modele de intrări care suferă distorsiuni, adică apar cu erori introduse de zgomote.

Aceste afirmaţii sunt susţinute de graficele din Figura 4.33 şi Figura 4.34 care prezintă variaţia ponderilor sinaptice ale celor doi neuroni de-a lungul procesului de învăţare. Se poate observa, că aceste valori (axa Y) se stabilizează în jurul valorii maxime (15) respectiv minime (0) în funcţie de locul de apariţie a punctelor întunecate (valori logice 1) din modelul de intrare asimilat de neuronul respectiv. Acest proces se realizează în aproximativ 20 de paşi (axa X), unde fiecare pas durează 16 cicluri de tact. Calculând la frecvenţa de lucru a microcontrolerului PIC utilizat, de 20 MHz, obţinem 16x20x100ns=*32μs*, ceea ce reprezintă timpul de învăţare a reţelei pentru orice două modele de intrare.

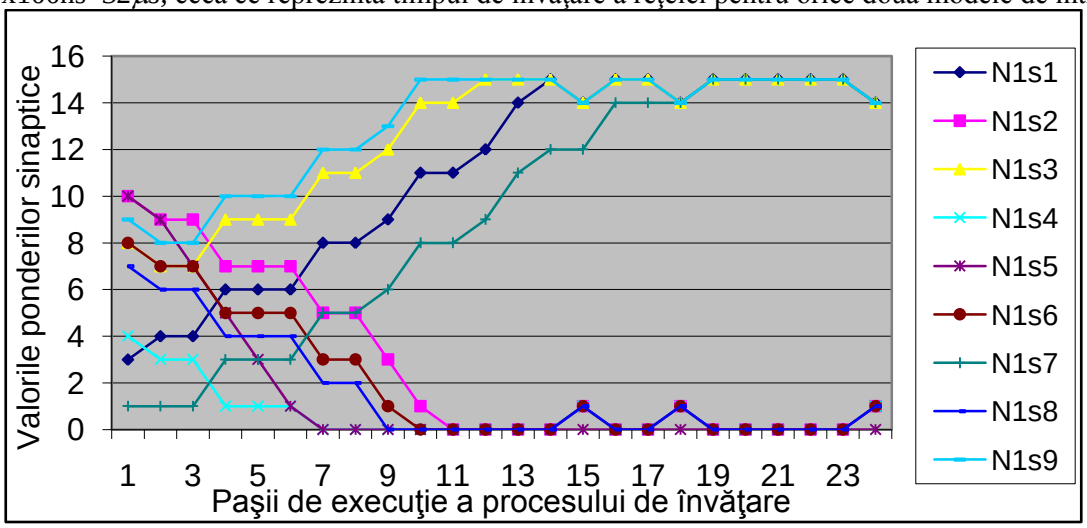


Figura 4.33 Variaţia ponderilor sinaptice a primului neuron în timpul antrenării

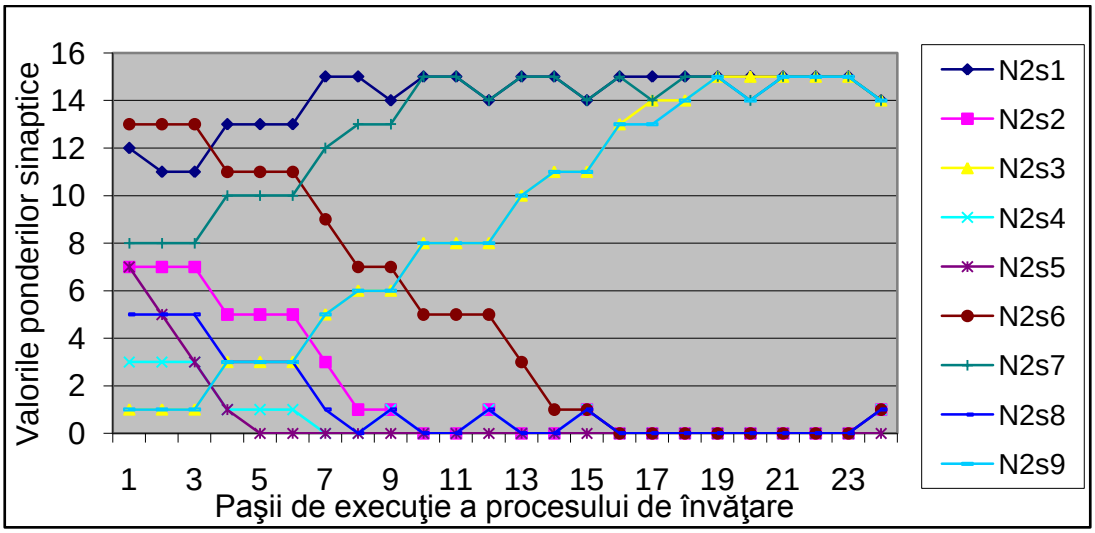


Figura 4.34 Variaţia ponderilor sinaptice a celui de-al doilea neuron în timpul antrenării

După cum am prezentat într-un capitol anterior, structura circuitului neuronal implementat, permite oprirea procesului de învăţare, astfel rolul microcontrolerului dispare. Utilizând însă pentru generarea semnalului de tact, circuitul oscilator programabil aflat pe placa de dezvoltare FPGA XSA100

putem ridica frecvenţa de funcţionare a reţelei la 100MHz. Astfel am ajuns la un timp de identificare a modelelor de intrare de către RNP de numai ~3,2μs. Evident această valoare reprezintă un rezultat important, care permite utilizarea de astfel de reţele neuronale în aplicaţii de control automat în timp real.

4.8.2. Comparaţie cu o reţea neuronală clasică realizată în software

Cu scopul de a confrunta performanţele obţinute prin procedeele noi, descrise în capitolele anterioare, am realizat o simulare de reţea neuronală feed-forward, cu ajutorul mediului Matlab, implementând aceeaşi structură ca a reţelei din FPGA. Astfel, şi neuronii reţelei simulate aveau câte nouă intrări cu ponderile aferente. În acest caz, însă ponderile au fost reprezentate pe 16 biţi în virgulă mobilă, cu o plajă de valori cuprinse între [-1, 1].

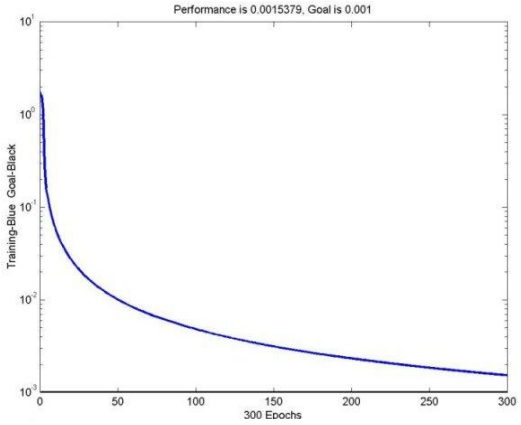


Figura 4.35 Curba de variaţie a erorii reţelei neuronale feed-forward, realizate cu scop de comparaţie în Matlab, în timpul procesului de antrenare

Setul de valori de intrare/ieşire ale algoritmului de învăţare au fost alese în aşa fel, încât să formeze un model pe o matrice de 3x3 ca şi cel implementat hardware. Ieşirile prescrise reţelei neuronale pentru cele două modele de intrare au fost setate la 1 respectiv –1. Reţeaua neuronală astfel creată a fost antrenată în 300 de cicluri (~300ms) cu algoritmul de învăţare backpropagation de tip Levenberg-Marquardt. Rata de învăţare a fost setată la 0.1, iar criteriul de oprire a fost atingerea unei erori de 0.001 faţă de valoarea prescrisă.

Figura 4.35 prezintă graficul variaţiei erorii de-a lungul procesului de antrenare a reţelei neuronale simulate în Matlab.

Comparând această curbă cu cele din Figura 4.37 respectiv Figura 4.36 – care prezintă eroarea medie pătratică a primului şi respectiv al celui de-al doilea neuron pulsativ al reţelei implementate în FPGA – putem nota performanţa net superioară a variantei hardware.

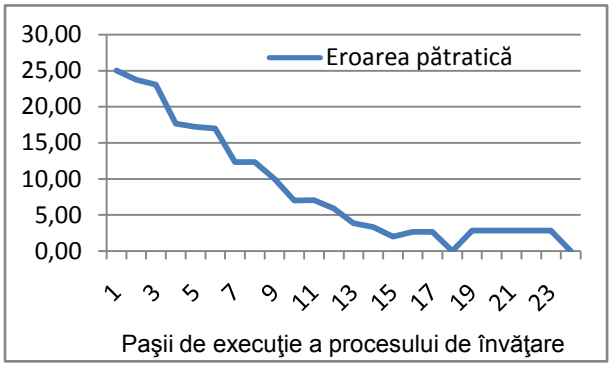


Figura 4.37 Eroarea medie pătratică a primului neuron al RNP implementate hardware

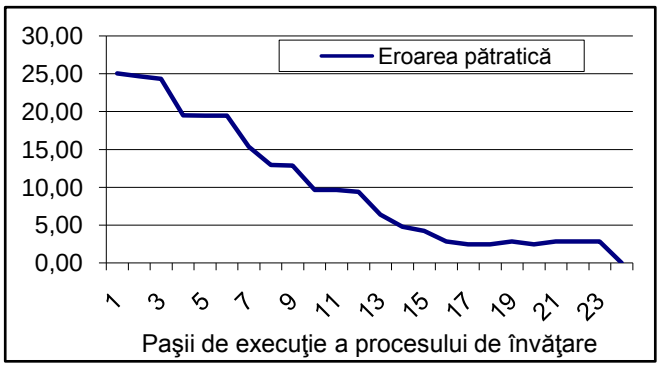


Figura 4.36 Eroarea medie pătratică a celui de-al doilea neuron al RNP implementate hardware

4.9. Concluzii

Acest al patrulea capitol al tezei de doctorat începe prin a prezenta caracteristici generale ale sistemelor hardware analogice şi digitale în ceea ce priveşte implementarea reţelelor neuronale artificiale, aducând motivaţii pro şi contra în fiecare caz. De asemenea sunt prezentate cele mai importante implementări hardware de reţele neuronale din literatura de specialitatea actuală cu accent asupra realizărilor pe diferite platforme digitale. Se trece apoi la domeniul specific temei acestei lucrări, prezentând implementări software şi hardware ale RNP realizate pe calculatoare paralele sau chiar în circuite FPGA.

După această parte de sinteză se prezintă contribuţiile proprii în ceea ce priveşte implementarea complet paralelă de RNP în circuite FPGA. Iniţial se expune implementarea modelului teoretic adaptat realizării hardware, validat prin simulare software în capitolele anterioare. Aceste subcapitole conţin descrieri amănunţite ale proiectării diferitelor subansamble ale acestor reţele neuronale, cum ar fi sinapsele şi modulele de somă sau corp celular. Acestea sunt testate prin simularea funcţionării circuitelor proiectate pentru a implementa funcţionalitatea specifică acestora.

O aplicaţie de test a acestor circuite neuronale realizate reprezintă continuarea logică a acestui capitol, aplicaţia fiind cea clasică de diferenţiere a două forme simple cum ar fi literele T şi H sau semnele + şi X. Se descrie reţeaua neuronală care implementează în circuit FPGA prin resurse hardware dedicate fiecărui subansamblu al acestuia (implementare complet paralelă), specificând algoritmul de învăţare utilizat şi enumerând rezultatele măsurătorilor experimentale. În cadrul unei analize a performanţelor (*32μs* - timpul de învăţare a reţelei, *32μs timp de recunoaștere a formelor de intrare*) acestei RNP se prezintă şi o comparaţie cu o rezolvare pur software (în mediul Matlab) a problemei propuse, rezultând performanţe net superioare ale versiunii hardware, care funcţionează în timp real.

Rezultatele şi contribuţiile personale prezentate în acest capitol au fost publicate într-un număr de lucrări ştiinţifice ale autorului tezei, apărute în publicaţii ale unor conferinţe de specialitate (Bakó & Brassai, Hardware spiking neural networks: parallel implementations using FPGAs, 2006), (Bakó & Brassai, 2005), (Brassai & Bakó, 2007), în reviste de specialitate (Bakó & Brassai, Spiking neural networks built into FPGAs: Fully parallel implementations, 2006) (Bakó, Székely, & Brassai, Development of Advanced Neural Models. Software And Hardware Implementation, 2004), precum şi în rapoartele unor granturi de cercetare în care autorul a fost membru.

5. IMPLEMENTĂRI PARŢIAL PARALELE ALE RNP UTILIZÂND MICROCONTROLERE ÎNCORPORATE ÎN CIRCUITE FPGA

5.1. Prezentarea rezultatelor actuale ale domeniului din literatura de specialitate

O propunere interesantă referitoare la modul de utilizare a reţelelor de neuroni pulsanţi a fost făcută în lucrarea *Lichid State Machine*, de către Maass, Natschlager şi Markram (Maass, Natschlager, & Markram, 2002). Cercetătorii din grupul condus de W. Maass au înţeles că o reţea de neuroni pulsanţi conectată aleatoriu poate implementa în mod eficace un filtru temporal complex prin intermediul elementelor complexe ale activităţii cu reverberaţie şi ale dinamicii sinaptice. Ţinând seama de informaţii primare extinse în timp, precum vorbirea, activitatea colectivă a reţelei poate fi descrisă ca o traiectorie printr-un spaţiu de stări multi-dimensional, iar această traiectorie ar trebui să fie recognoscibil caracteristică pentru intrarea la dispoziţie. Un simplu decodor „read-out" ar trebui să fie suficient pentru a clasifica tiparul (modelul) temporal.

În lucrarea „Isolated Word Recognition With The *Liquid State Machine*: A Case Study", (Verstraeten, Schrauwen, Stroobandt, & Van Campenhout, 2005) autorii studiază recunoaşterea limbajului în Liquid State Machine (LSM). Pentru un set lexical standard, se testează o varietate de codificări temporale care sunt introduse într-o LSM şi un următor decodor liniar simplu clasifică fiecare tipar în parte. Surprinzător, Verstraeten et al. au găsit doar performanţe (încurajator de bune) pentru LSM atunci când back-end-ul codificator aproximează schema de codificare a urechii interne.

Reguli de învăţare echivalente celor folosite în reţelele neuronale sigmoidale tradiţionale au provenit de asemenea din reţele (stratificate) de neuroni pulsanţi, de exemplu o regulă error-backpropagation bazată pe un gradient precum Spikeprop (Bohte, Kok, & La Poutré, 2000). Cu o astfel de regulă de învăţare, s-a demonstrat că o reţea neuronală pulsativă poate în fapt calcula funcţii non-liniare separabile precum reţelele neuronale tradiţionale.

Booij şi Nguyen prezintă regula de învăţare bazată pe propagarea înapoi a erorii (Booij & Nguyen, 2005), care elimină importante limitări ale regulii Spikeprop în lucrarea intitulată „A gradient descent rule for spiking neurons emitting multiple spikes". Înainte de toate, în derivaţia lor gradientul erorii este calculat pentru un neuron emiţând multiple impulsuri. În plus, simple metode euristice sunt discutate care diminuează efectul disruptiv al discontinuităţilor în potenţialul de membrană pentru algoritmi de minimizare a gradientului. Eficacitatea algoritmului îmbunătăţit este demonstrată pe o versiune temporală a clasicei probleme XOR şi pe clasificarea transmisiilor (prin angrenaje) de impulsuri Poisson. În mod interesant, pentru cea mai simplă problemă XOR Booij şi Nguyen ne arată că există o soluţie „hair trigger" într-o reţea fără straturi ascunse.

Un alt domeniu unde neuronii pulsanţi sunt consideraţi a fi deosebit de puternici este cel al *memoriei associative*. Knoblauch studiază tehnologia de ultimă oră în lucrarea „Neural Asssociative Memory for Brain Modeling and Information Retrieval" (Knoblauch, 2005). Începând cu o expunere asupra reţelelor neuronale asociative tradiţionale sub forma clasicului model Willshaw, soluţii folosind neuroni pulsanţi sunt sugerate şi este discutat faptul dacă memoriile asociative neuronale distribuite au avantaje practice faţă de memoria locală/limitată.

Multe aplicaţii folosesc reţele neuronale tradiţionale ca aproximatori de funcţii când funcţia necesară este necunoscută. Având un set de date, reţelele neuronale s-au dovedit a fi foarte bune în a conecta acestea, interpolând valorile dintre ele, apoi aproximând funcţia.

Aşa cum Ianella şi Kindermann subliniază în articolul lor „Finding Iterative Roots with a Spiking Neural Network", o sarcină mai complexă dar adesea şi mai profitabilă va fi cea care va găsi componentele (recurente) care alcătuiesc funcţiile ce urmează a fi aproximate. În lucrarea lor (Iannella & Kindermann, 2005), ei demonstrează că progresul stă în învăţarea unor astfel de soluţii iterative într-o reţea de neuroni pulsanţi. Doi algoritmi pentru învăţare sunt prezentaţi, unul semi-supravegheat în care funcţia este cunoscută şi sarcina fiind aceea de a găsi valorile funcţiei, iar celălalt ne-având această extra-informaţie.

O problemă importantă în utilizarea reţelelor neuronale pulsative este faptul că ele sunt în mod tipic computaţional mai intensive decât reţelele neuronale tradiţionale. A fost deja stabilit că evenimentul bazat pe natura impulsurilor cronometrate reduce în mod drastic randamentul de comunicare între neuroni, permiţând în principiu implementări paralele eficiente.

Aceasta încă lasă deschis modul de a calcula eficient în ce moment un neuron se activează având în vedere impulsurile sale de intrare. În „Spiking Neural Nets with Symbolic internal State", O'Dwyer şi Richardson prezintă un algoritm pentru a calcula eficient această activare bazată pe manipularea simbolică şi intervalul aritmetic (O'Dwyer & Richardson, 2005).

Sunt de părere că aceste lucrări arată că se face un progres substanţial spre aplicarea reţelelor neuronale pulsative: studiul lui Verstraeten et al. demonstrează că reţelele de neuroni pulsanţi ar putea avea un real viitor în recunoaşterea vorbirii; Booij şi Nguyen obţin nişte reguli de învăţare mai eficiente; Knoblauch tratează atât capacităţile cât şi limitările reţelelor neuronale pulsative pentru memoria asociativă; munca lui Ianella şi Kindermann are ca scop găsirea soluţiilor funcţionale, iar O'Dwyer şi Richardson prezintă un algoritm care ajută la calcularea eficientă a activităţii neîntrerupte într-o reţea de neuroni pulsanţi.

Pe lângă acest progres - deja foarte palpabil - care duce spre reţele neuronale pulsative practice, trebuie remarcat faptul că diferite lucrări au fost prezentate recent care fac ca reţelele de neuroni pulsanţi să fie privite cu un si mai mare interes din punct de vedere practic: lucrări de Deneve (Deneve, 2005), Rao (Rao, 2005) şi Zemel et al. (Zemel, Huys, Natarajan, & Dayan, 2005) propun diferite moduri de a realiza inferenţa Bayesiană în reţele neuronale pulsative. Aceste lucrări dezvăluie o perspectivă reală şi ar putea fi baza a mai multor aplicaţii. Desigur, aplicaţiile reţelelor de neuroni pulsative prezentate aici reprezintă doar începutul celor ce vor urma.

5.2. Clasificarea nesupervizată de mulţimi complexe (complex clusters) cu reţele de neuroni pulsativi

Întrucât neuronii produc în creier potenţiali de acţiune „tot sau nimic", importanţa coordonării acestor impulsuri a fost recent recunoscută ca o mijloc de codare a informaţiei neuronale. Datorită acumulării de probe biologice, s-a arătat teoretic că o astfel de codificare temporală permite procesarea puternică de informaţii de către neuroni şi că ar putea fi esenţială în rezolvarea problemelor dinamice combinatorice. Aceste considerente au generat un interes considerabil în domeniul reţelelor neuronale artificiale secvenţiale. Hopfield prezintă un model de neuroni pulsanţi folosiţi pentru a descoperi fascicule într-un spaţiu de intrare asemănător reţelelor bazate pe funcţiile radiale. Natschläger şi Ruf prezintă (Natschläger & Ruf, 1998) un algoritm de învăţare pentru reţele neuronale pulsative care efectuează clasificări nesupervizate de fascicule folosind temporizări de impulsuri ca intrare. Acest model codifică şabloanele de intrare în decalaje temporale în propagarea impulsurilor de-a lungul sinapselor şi s-a demonstrat că este fiabil în a găsi centre ale zonelor multi-dimensionale, dar este limitat în capacitatea şi precizia de clasificare.

Inspirat fiind de domeniile receptive locale ale neuronilor biologici, am codificat variabile de intrare de către neuroni cu profiluri de senzitivitate gradate şi suprapuse. Mai mult decât atât, fiecare dimensiune de intrare a unui set de date de dimensiune mare este codificat independent, evitându-se o creştere exponenţială a numărului neuronilor de intrare. Aceasta s-a arătat că validează în mod eficient reţeaua pentru a clasifica cu succes un număr de probleme de clasificare de fascicule la costuri relativ joase în ceea ce priveşte neuronii. Codificarea multiplă are în vedere detectarea fiabilă a fasciculelor asupra unei game remarcabile şi flexibile de scale spaţiale. Acest aspect este în mod special important în sarcinile de clasificare necontrolată, unde informaţia de scalare este a-priori necunoscută.

Extinzând reţeaua la straturi multiple, putem demonstra cum anume aspectul secvenţial al neuronilor pulsanţi poate fi exploatat pentru a valida clasificarea corectă a fasciculelor non-sferice sau suprapuse. Într-o reţea RBF multistrat, putem demonstra că neuronii din primul strat sunt specializaţi pe componente de fascicule. Sincronismul codificării neuronilor pentru componente apropiate este apoi arătat a fi uşor remarcabil de către un strat RBF, rezultând într-o formă de clasificare ierarhică. Adăugarea unor conexiuni excitatoare laterale se spune că validează o astfel de reţea multistrat pentru a clasifica în mod corect structuri integrate complexe, sincronizând codificarea neuronilor pentru elemente ale aceluiaşi fascicul. Astfel, conexiunile laterale adiţionale produc un număr scăzut de neuroni, în acelaşi timp mărind complexitatea fasciculelor clasificabile şi folosind pentru aceasta doar o regulă de învăţare Hebbiană locală.

5.3. Implementarea unei reţele de neuroni pulsativi cu întârziere

5.3.1. Structura reţelei

Structura reţelei este formată din neuroni pulsanţi conectaţi în buclă deschisă (feedforward) de tipul integrate-and-fire permeabil (leaky integrate-and-fire type). Cum este ilustrat în Figura 5.1A, conexiunile din reţea constă dintr-un set de sinapse *k*, fiecare cu o greutate w^k_{ij} şi o întârziere d^k. Un impuls de intrare generează un set de potenţiale post-sinaptice ponderate (PSP, o funcţie modelând impactul unui impuls asupra neuronului ţintă ca o funcţie temporală). Mărimea ponderii sinaptice determină înălţimea PSP-ului. La fiecare pas temporal al algoritmului, valorile acestor PSP-uri sunt adunate la neuronul ţintă, iar când suma depăşeşte pragul θ, un impuls de ieşire este generat.

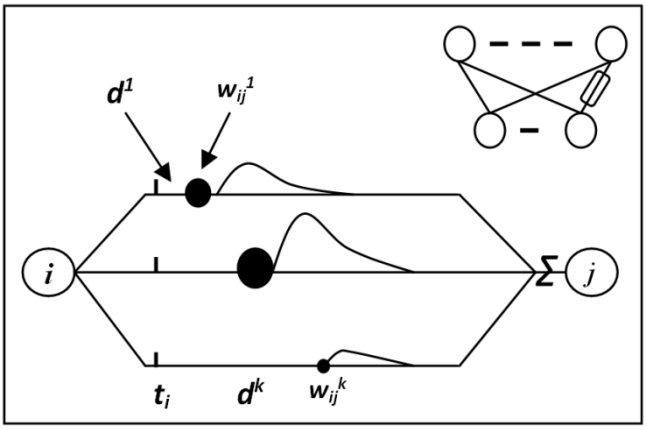


Figura 5.1 Structura de bază a reţelei implementate

Semnalele de intrare pot fi codificate în ponderile sinaptice prin **învăţare întârziată**. După învăţare, momentul de activare al unui neuron de ieşire reflectă distanţa de la mostra de evaluat până la mostra de intrare învăţată (realizând astfel un fel de neuroni RBF). Pentru învăţarea nesupervizată, ponderile dintre neuronii sursă şi cei de ieşire din stratul ţintă (Winner-Take-All) sunt modificate folosind o variantă temporală a regulii de învăţare Hebbiană. Dacă startul unui PSP la o sinapsă precedă cu puţin un potenţial de acţiune în neuronul ţintă, ponderea sinapsei creşte, deoarece a exercitat o influenţă semnificativă asupra momentului activării printr-o contribuţie relativ mare către potenţialul de membrană. Sinapsele anterioare şi ulterioare scad în pondere, reflectând impactul lor scăzut asupra momentului de emisie al neuronului ţintă. Pentru o pondere cu o întârziere d_k de la neuronul *i* la neuronul *j* vom folosi:

$$\Delta w_{ij}^k = \eta L(\Delta t) = \eta(1-b)\exp\left[\frac{(\Delta t - c)^2}{\beta^2}\right] + b \qquad \textit{Ec. 65}$$

după (Natschläger & Ruf, 1998) (ilustrat în Figura 5.1B). Ponderile sunt limitate de o valoare minimă şi una maximă. O intrare în reţea este codificată de un tipar de timpi de activare în cadrul unui interval de codificare şi fiecare neuron de intrare trebuie să se activeze o dată în timpul acestui interval de codificare.

5.3.2. Variabile de intrare continue codificate în decalaje temporale

Pentru a extinde precizia de codificare şi capacitatea de clasificare, am investigat folosirea multiplelor domenii receptive locale pentru a distribui o variabilă de intrare asupra mai multor neuroni de intrare. Un astfel de cod de distribuţie unde variabilele de intrare sunt codificate cu funcţii de activare gradate şi integrate, reprezintă o mult studiată metodă pentru a reprezenta parametri cu valori reale (Snippe, 1996). În aceste studii, funcţia de activare al unui neuron de intrare este modelată ca un domeniu receptiv local care determină rata de activare. O traducere a acestei paradigme în timpi de activare relativi este simplă: un neuron stimulat la un nivel optim se aprinde la $t = 0$, în timp ce o valoare de până la $t = 9$ este atribuită unor neuroni stimulaţi la un nivel mai puţin optim (ilustrat în Figura 5.2).

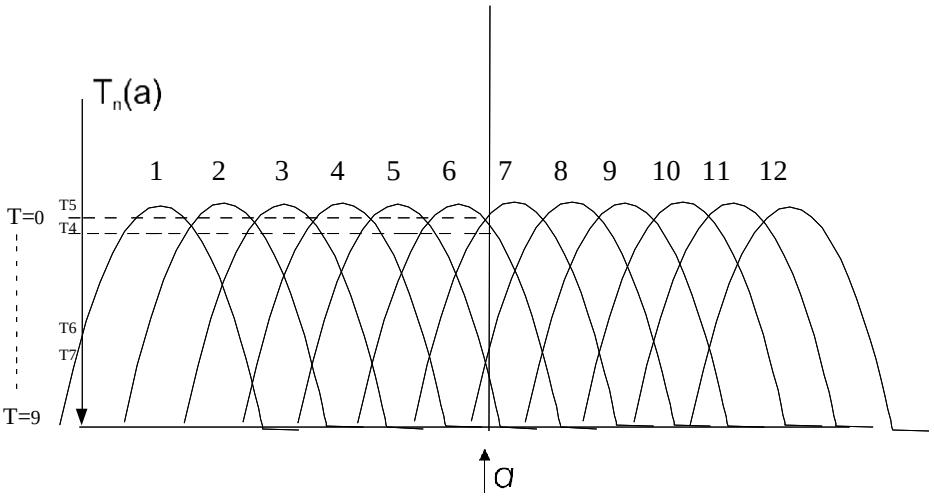


Figura 5.2 Conceptul de codificare a variabilelor de intrare în impulsuri decalate temporal

Pentru a codifica date multidimensionale în maniera descrisă mai sus, o alegere trebuie făcută în ceea ce priveşte dimensionalitatea domeniilor receptive ale neuronilor. Codificarea cea mai puţin costisitoare din punctul de vedere al neuronilor este aceea în care fiecare intrare este codificată independent cu domenii receptive 1-D. Din moment ce suntem interesaţi de clasificare multidimensională, această codificare este foarte oportună deoarece se comportă liniar în ceea ce priveşte numărul de neuroni necesari pe dimensiune şi este de asemenea adaptabilă, permiţând codificarea unor dimensiuni cu precizie mare fără costuri neuronale excesive.

5.3.2.1. Implementarea codării valorilor de intrare în decalaje temporale pe FPGA – Neuronii de intrare, de tip pseudo-RBF

Am experimentat mai multe variante de astfel de codări, luând în calcul resursele disponibile în circuitul FPGA Xilinx Spartan 3 XC3S1000 utilizat. Pentru a utiliza cât mai eficient blocurile logice re-configurabile din acest circuit, respectiv, pentru a păstra cât mai multe astfel de resurse implementării neuronilor pulsativi, am încercat mai multe variante de stocare a valorilor funcţiilor RBF ale intrărilor în module BlockRAM. Acestea sunt disponibile în mai multe variante de configuraţie, cu diferite lăţimi ale magistralelor de adrese şi de date. După cum se vede şi din tabelul de mai jos (Figura 5.3), totalul de 54Kbytes (432Kbiţi în tabel) de memorie RAM disponibilă este împărţită în 24 de blocuri (Figura 5.4). Dilema proiectantului este în a decide cum să utilizeze aceste resurse, sacrificând – chiar dacă parţial – natura pur paralelă a implementării, sau utilizând mai multe BlockRAM-uri decât ar fi necesare din punct de vedere al cantităţii de informaţie ce necesită stocare.

Part Number	XC3S50	XC3S200	XC3S400	XC3S1000
System Gates	50K	200K	400K	1000K
Logic Cells	1,728	4,320	8,064	17,280
Dedicated Multipliers	4	12	16	24
Block RAM Blocks	4	12	16	24
Block RAM Bits	72K	216K	288K	432K
Distributed RAM Bits	12K	30K	56K	120K
DCMs	2	4	4	4

Figura 5.3 Parametrii principali al circuitelor Xilinx FPGA din familia Spartan 3

După cum se poate vedea şi în Figura 5.5 (generată în Matlab), funcţiile domeniilor receptive locale nu sunt Gaussiene ci triunghiulare, pentru a fi uşor implementate în hardware. Pe axa x a acestei diagrame am reprezentat valorile intrării reţelei, care sunt valori pozitive, întregi şi pot varia între 0 şi 192.

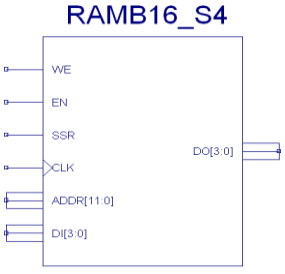


Figura 5.4 Modul BRAM, 1024 cuvinte de 4 biţi

Fiecare valoare a acestui interval este distribuită pe mai mulţi neuroni de intrare (min. 2 – max. 4) datorită suprapunerii funcţiilor de activare ale acestora. Cu cât este mai mare valoarea unei funcţii de activare ale unuia dintre cei 12 neuroni de intrare implementaţi în acest proiect, cu atât mai repede se va activa acest neuron, adică decalajul temporal al apariţiei impulsului de activare faţă de momentul apariţiei valorii de intrare va fi cu atât mai mică. Evident, există multe posibilităţi de a varia aria de suprapunere a triunghiurilor, modificând astfel numărul neuronilor de intrare care se vor activa pentru fiecare valoare de intrare, adică a celor care vor coda această valoare. Totodată, pentru fiecare neuron de intrare există o gamă redusă de valori de intrare pentru care se activează cu decalaj temporal corespunzător. Aceste valori pot fi

calculate şi în timpul funcţionări circuitului neuronal, dar acest lucru ar consuma timp şi resurse preţioase. Pentru a evita aceste impedimente, am decis calcularea acestor valori în prealabil şi stocarea lor în memoriile RAM disponibile în FPGA.

Utilizarea optimă a memoriilor BRAM ar fi fost atinsă, dacă s-ar fi stocat valori ale mai multor neuroni de intrare în acelaşi modul de memorie. Astfel însă ar fi fost imposibilă accesare simultană a acestora. Din acest motiv, am ales să utilizez mai multe memorii BRAM, din tipul ***RAMB_16_S4***, a cărui capacitate este de 1024 cuvinte de 4 biţi şi se comportă ca o memorie statică, sincronă (Figura 5.4).

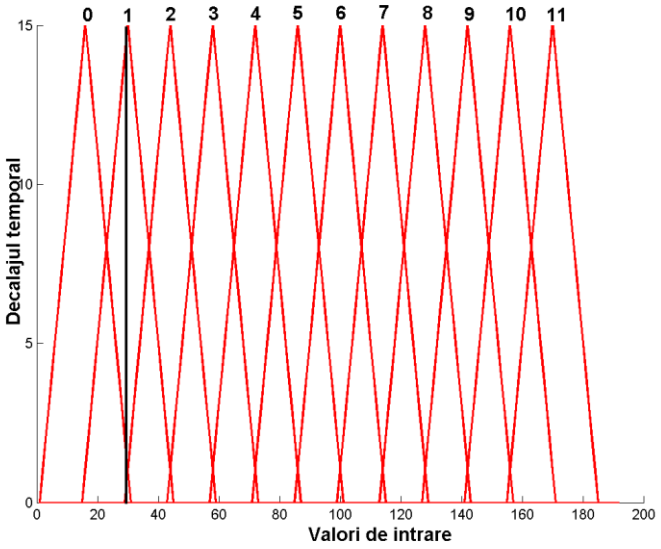


Figura 5.5 Funcţiile triunghiulare ale domeniilor receptive utilizate la implementare

Aceste valori au fost calculate cu ajutorul unui program scris în Matlab (vezi Anexa). Programul permite modificarea uşoară a intervalelor suprapuse şi a numărului de neuroni de intrare utilizaţi. De asemenea, programul Matlab generează codul VHDL necesar iniţializării memoriilor BRAM, cu valorile calculate, după cum arată Figura 5.6.

```
INIT_00 : bit_vector :=   x"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEDCBA98765432101234567890ABCDEF";
INIT_01 : bit_vector :=   x"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF";
INIT_02 : bit_vector :=   x"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF";
INIT_03 : bit_vector :=   x"FFFFFFFFFFFFFFFFFFFFEDCBA98765432101234567890ABCDEFFFFFFFFFFFFFFF";
INIT_04 : bit_vector :=   x"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF";
INIT_05 : bit_vector :=   x"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF";
INIT_06 : bit_vector :=   x"FFFFFFEDCBA98765432101234567890ABCDEFFFFFFFFFFFFFFFFFFFFFFFFFFF";
```

Figura 5.6 Valorile funcţiilor triunghiulare ale domeniilor receptive cu care s-au iniţializat memoriile BRAM

5.4. Realizarea unei reţele de neuroni pulsativi cu antrenare on-chip şi aplicarea sa practică

Inspirat din fenomenele biologice, un nou tip de model de reţea neuronală, reţeaua de neuroni pulsanţi, a atras atenţia cercetătorilor din domeniul calculului neuronal în ultimii ani. Reţeaua implică neuroni cu funcţionare prin impulsuri ca şi unităţi de calcul. Foloseşte cronometrarea impulsurilor individuale (folosiţi în literatura de specialitate sub denumirea de potenţial total-sau-nul de acţiune sau spike) sau frecvenţa instantanee a şirului de impulsuri sau alte caracteristici temporale ale impulsurilor, produse de neuroni pentru codificarea informaţiei.

Această nouă generaţie a modelelor neuronale, spre deosebire de modelele neuronale convenţionale (cu funcţii sigmoidale, etc.), poate să ofere explicaţii în ceea ce priveşte fenomenul procesării informaţiei în creier printr-o interpretare biologică mult mai realistă, şi pe lângă aceasta, pot să atingă performanţe computaţionale competitive. Pe lângă acestea, aceste modele neuronale au avantaje ale implementării hardware, datorită faptului că codificarea bazată pe cronometrare oferă o

potenţială compactitate şi este mai robustă decât implementarea analogică. În cadrul modelelor neurale pulsative, modelul lui Hopfield este cel proiectat ca şi un set de detectori de coincidenţe pentru impulsuri de intrare pe canale multiple.

Poate fi folosit pentru a rezolva recunoaşterea analogică de tipare în mod natural. Hopfield a indicat faptul că aceste modele neuronale funcţionează de fapt ca unităţi de funcţii de bază radiale (RBF). Această idee de bază a fost apoi extinsă asupra mai multor scheme sofisticate. Natschläger şi Ruf au propus o reţea RBF folosind neuroni pulsanţi (Natschläger & Ruf, 1998), care codifică tiparele în întârzierile sinaptice. Fiecare întârziere este tratată ca şi o întârziere medie a populaţiei sub-sinaptice, şi este ajustată indirect prin ajustarea eficacităţii populaţiei sub-sinaptice. Momentul de activare al neuronului pulsativ are o dependenţă regulată de distanţa euclidiană dintre vectorul de intrare şi vectorul central reprezentat prin întârzierile sinaptice.

Bothe et al. (Bohte, Kok, & La Poutre, 2000) combină arhitectura reţelei neurale pulsative (RNP) cu o metodă de codificare temporală care foloseşte câmpuri receptive Gaussiene suprapuse pentru a traduce valorile de intrare analogice în decalaje temporale ale impulsurilor de activare.

Am proiectat un algoritm de antrenare Hebbian modificat pentru această RNP pentru a implementa o aplicaţie de clustering (clasificare) nesupervizată. În ambele lucrări (Natschläger & Ruf, 1998) şi (Bohte, Kok, & La Poutre, 2000), întârzierea sinaptică este tratată ca şi o întârziere medie a unei populaţii sub-sinaptice. În timp ce această arhitectură reflectă realităţile biologice, ea creşte capacitatea de calcul a modelului şi se delimitează de la algoritmul de antrenare nesupervizată mult mai flexibil.

În această lucrare, am propus realizarea unei reţele neuronale pulsative bazate pe ideea originală a lui Hopfield. Acest model este construit astfel încât timpii de activare ai impulsurilor de ieşire depind de sincronizare şi de timpul incidental al sincronizării impulsurilor de intrarea. Sincronizarea în rotaţie se poate optimiza prin coordonarea relaţiei dintre modelul impulsurilor de intrare şi modelul întârzierii sinaptice. Pentru a înlesni calculul şi realizarea circuitului, vom abstractiza întârzierea sinaptică ca şi o variabilă direct ajustabilă în acest model neuronal.

Cele mai multe aplicaţii ale reţelelor neuronale pulsative şi convenţionale se implementează prin folosirea unui simulator soft. Pe lângă acestea, calculele în cazul SNN menţionat anterior sunt destul de complexe pentru o implementare hardware.

Această constituie o limită a uneia dintre cele mai importante avantaje ale reţelelor neuronale, paralelismul masiv. Fiind şi scopul principal al disertaţiei de doctorat, am proiectat circuite hardware pentru reţeaua neuronală propusă folosind circuite Field Programmable Gate Array (FPGA).

Cum circuitul FPGA este un sistem digital, avantajele tehnicii digitale, cum ar fi robusteţea la zgomot şi flexibilitatea designului, sunt eficiente în mod natural. Dar o tehnică digitală necesită de obicei o suprafaţă mare atunci când se foloseşte în cazul circuitelor neuronale paralele. Gâtuirea se recunoaşte de obicei în apariţia unui multiplicator în cazul cadrul calculelor necesare. S-au raportat de asemenea mai multe scheme ale reţelelor neuronale fără multiplicatori. Marchesi et al. (Marchesi, Orlandi, Piazza, & Uncini, 1993) foloseşte registre mobile pentru implementarea multiplicatorilor de aproximaţie într-o reţea neuronală de valori reale, precum şi în publicaţiile proprii (Bakó & Brassai, 2006) (Bakó, Brassai, & Székely, 2006) (Bakó, Székely, & Brassai, 2004) (Brassai & Bakó, 2007) în care am implementat reţele neuronale pulsative bazate pe modele cu prelucrarea ratei impulsurilor realizate fără circuite multiplicatoare.

Totuşi, circuitele hardware sunt disponibile doar în cazul calculului forward. Reţeaua antrenată este dirijată folosind un simulator offline. Hikawa (Hikawa, 1999) a propus o schemă hardware bazată pe frecvenţe cu antrenare on-chip. Ambele scheme (Marchesi, Orlandi, Piazza, & Uncini, 1993) şi (Hikawa, 1999) sunt reţele back-propagation (BP), ai căror algoritmi de antrenare BP ar putea să ajungă într-un minim local. Spre deosebire de acestea, modelul neuronal pulsativ propus în această lucrare evită această problemă, rezultând din esenţa sa de reţea RBF. Folosind natura de sincronizare a impulsurilor temporale, atât calculele forward cât şi cele de antrenare sunt efectuate cu circuite fără multiplicatori.

5.4.1. Reţeaua de neuroni pulsativi realizată în FPGA

Arhitectura reţelei constă dintr-o reţea feed-forward de neuroni pulsanţi cu terminale sinaptice întârziate multiple Figura 18. Neuronii reţelei generează potenţiali de acţiune, sau impulsuri, când variabila de stare internă a neuronului, numit „potenţialul membranei", depăşeşte un prag γ. Relaţia dintre impulsurile de intrare şi variabile stării interne este descrisă prin modelul de răspuns al impulsului (spike response model SRM), introdus de Gerstner (Gerstner W. , Spiking neurons, 2001). În funcţie de alegerea funcţiilor de răspuns ale impulsurilor potrivite, se poate adapta acest model pentru a reflecta dinamica unei mari varietăţi de neuroni pulsanţi.

5.4.1.1. Modelul neuronal RBF pulsativ implementat

Modelul reţelei neuronale pulsative se poate vedea în Figura 5.7. Fiecare neuron, care funcţionează ca şi o unitate RBF, primeşte impulsuri de intrare prin conexiuni de la neuronii de intrare. Segmentul intermitent între fiecare neuron de intrare la terminalul sinaptic corespunzător denotă întârzierea, ceea ce este definită ca şi diferenţa dintre momentul de activare al neuronului pre-sinaptic şi momentul activării neuronului post-sinaptic.

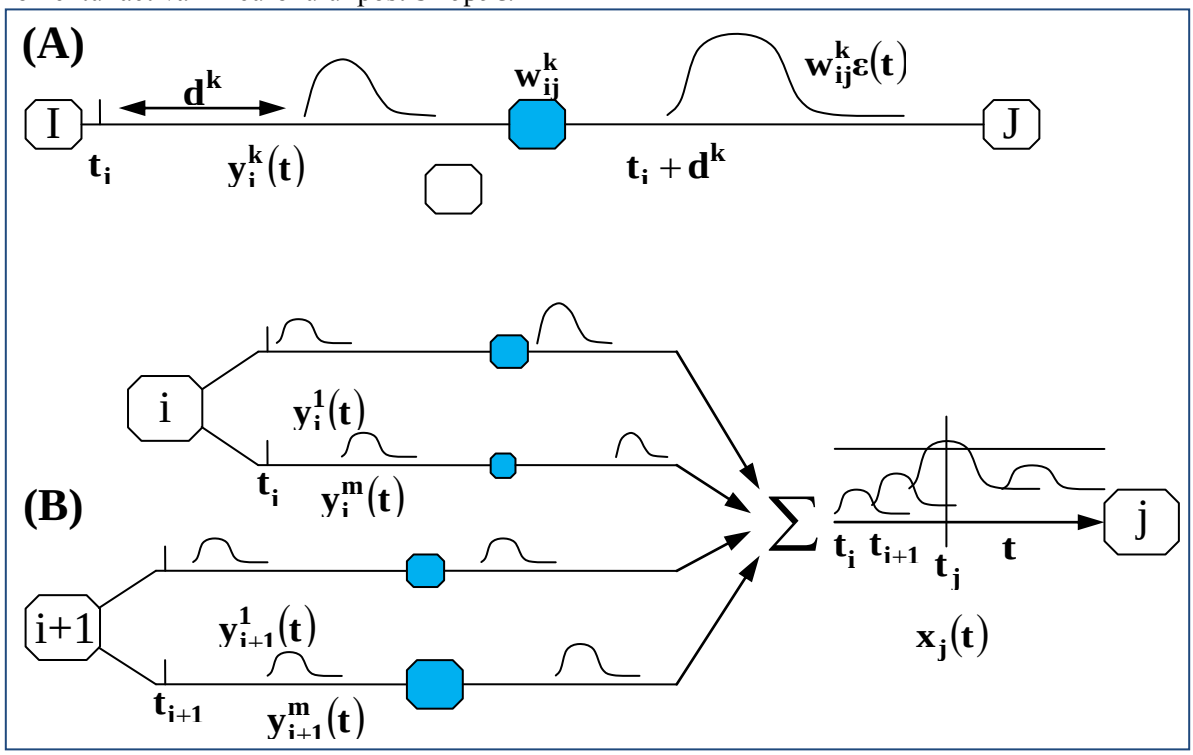


Figura 5.7 Principiul de funcţionare al modelul neuronal implementat

Din punct de vedere formal, un neuron j, având un set Γ_j de predecesori imediaţi (neuroni pre-sinaptici), primeşte un set de impulsuri cu timpul de activare t_i, $i \in \Gamma_j$. Fiecare neuron generează cel mult un impuls în timpul intervalului de simulare, şi emite când variabila de stare internă atinge o valoare de prag γ. Dinamica variabilei de stare internă $x_j(t)$ este determinată de impulsurile incidente, ale căror influenţă se poate descrie prin funcţia-răspuns a impulsului $\varepsilon(t)$ ponderat cu eficacitatea sinaptică („ponderea") w_{ij}:

$$x_j(t) = \sum_{i \in \Gamma_j} w_{ij}\,\varepsilon(t - t_i) \qquad \text{Ec. 66}$$

Funcţia-răspuns a impulsului în Ec. 66 modelează eficient potenţialul post-sinaptic ne-ponderat (PSP) al unui singur impuls incident asupra unui neuron. Înălţimea PSP este modulată de către ponderea post-sinaptică w_{ij} pentru a obţine potenţialul post-sinaptic efectiv. Funcţia-răspuns a impulsului folosită în experimente este definită prin Ec. 68. În reţeaua realizată, o conexiune individuală constă dintr-un număr fix de terminale sinaptice, unde fiecare terminal constituie o sub-conexiune asociată unei întârzieri şi unei ponderi diferite (Figura 5.7). Întârzierea d^k a terminal sinaptic k se defineşte prin diferenţa dintre momentul de activare a neuronului pre-sinaptic şi momentul în care potenţialul post-sinaptic începe să crească.

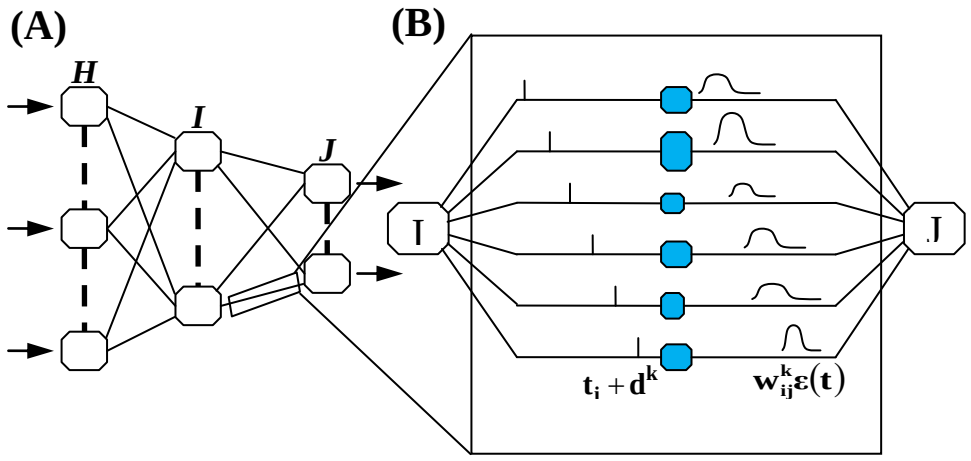


Figura 5.8 Structura logică a reţelei neuronale implementate

Vom descrie un impuls pre-sinaptic la un terminal sinaptic *k* ca şi un PSP de înălţime standard cu o întârziere d^k. Contribuţia ne-ponderată a unui singur terminal la variabila de stare se dă prin formula:

$$y_i^k(t) = \varepsilon\left(t - t_i - d^k\right) \qquad \text{Ec. 67}$$

cu o funcţie-răspuns a impulsului de forma unui PSP, cu $\omega(t) = 0$ pentru $t<0$.

Timpul t_i este timpul de activare a neuronului pre-sinaptic *i*, iar d_k este întârzierea asociată terminalului sinaptic *k*. Funcţia răspuns a unui impuls care descrie un PSP standard are forma:

$$\varepsilon(t) = \frac{t}{\tau} e^{1-\frac{t}{\tau}} \qquad \text{Ec. 68}$$

modelând o funcţie α simplă pentru $t>0$, iar τ modelează constanta de timp a întârzierii potenţialului membranei care determină timpul de creştere şi de întârziere a PSP. Extinzând Ec. 66 pentru a include sinapse multiple pe conexiune şi inserând Ec. 67, variabila de stare a unui neuron x_j a unui neuron *j* care primeşte intrări de la toţi neuronii *i* se poate descrie ca şi suma ponderată a contribuţiilor pre-sinaptice:

$$x_j(t) = \sum_{i \in \Gamma_j} \sum_{k=1}^{m} w_{ij}^k y_i^k(t) \qquad \text{Ec. 69}$$

unde w_{ij}^k înseamnă ponderea asociată terminalului sinaptic k (Figura 5.8 B). Timpul de activare t_j a unui neuron *j* este determinat ca şi primul moment în care variabila de stare depăşeşte pragul $\vartheta : x_j(t \geq \vartheta)$. Astfel, timpul de activare t_j este o funcţie neliniară a variabilei de stare $x_j : t_j = t_j(x_j)$. Pragul ϑ este constant şi egal pentru toţi neuronii reţelei.

5.4.2. Arhitectura reţelei implementate pentru clasificarea de fascicule cu domenii receptive

Reţeaua neuronală hibridă RBF-spiking a fost implementată cu două intrări, având un spaţiu de intrări bidimensional, format din 192*192=36864 puncte. Scopul implementării este ca reţeaua să poată identifica punctele din jurul unor centre definite. Numărul centrelor este determinat de numărul neuronilor de ieşire. Fiecare neuron de ieşire va tinde să se activeze atunci, când intrările sunt destul de aproape de centrul fasciculului (cluster) pe care l-a învăţat.

Pentru a studia capacitatea şi precizia sporită asigurată de codificarea descrisă, am examinat clasificarea unui număr de probleme generate artificial. Când am clasificat intrări constând din două variabile separate, am aflat că o reţea formată din 24 de neuroni de intrare (fiecare variabilă codificată de 12 neuroni) este capabilă să clasifice corect 17 fascicule distribuite în mod egal, prezentând o creştere semnificativă în capacitatea de acumulare a fasciculelor. Astfel de rezultate pozitive au fost obţinute şi în cazul unor acumulări de fascicule spaţiate mai puţin regulat. Am aflat de asemenea că, diminuând

întinderea câmpurilor domeniilor receptive, se pot observa acumulări de fascicule distribuite tot mai compact.

5.4.2.1. Structura proiectului VHDL care implementează reţeaua neuronală

Proiectul VHDL are următoarele elemente componente principale, aranjate conform ierarhiei de mai jos:

- Reţeaua neuronală
  - Bloc de intrare (Input0block.vhdl)
    - Neuronul de intrare (InputNeuron.sch)
      - Blocul funcţiilor domeniilor receptive locale (myBRAMbasefunc.vhd)
      - Modulul funcţional al neuronului de intrare (psRBFInputNeuron.vhd)
  - Modulul sinapselor (Synapse.vhd)
  - Modulul neuronilor de ieşire (RBPoutputSpikingNeuron.vhd)
  - Generatorul paşilor temporali de funcţionare (Timestepgenerator.vhd)
- Modulul de comunicaţie cu calculatorul (DownloadUnit.vhd)

Figura 5.9 respectiv Figura 5.10 prezintă schema bloc şi cea de implementare a întregii reţele neuronale.

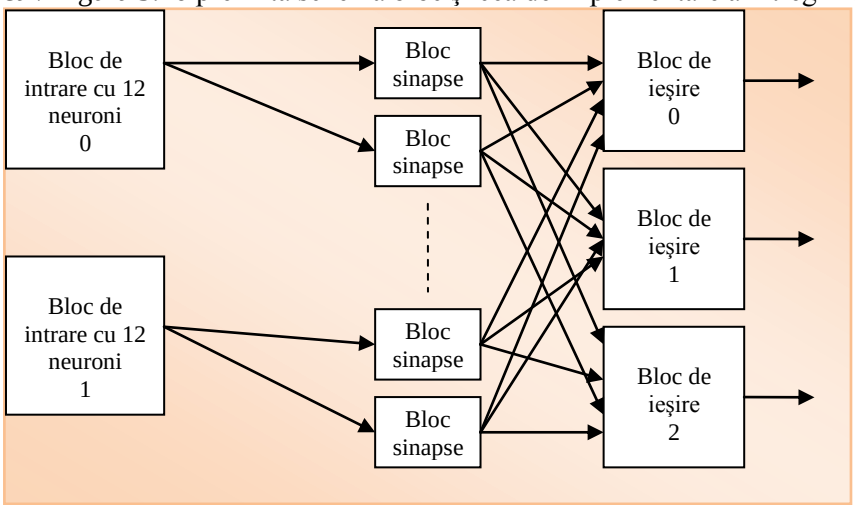


Figura 5.9 Schema bloc a structurii reţelei neuronale implementate

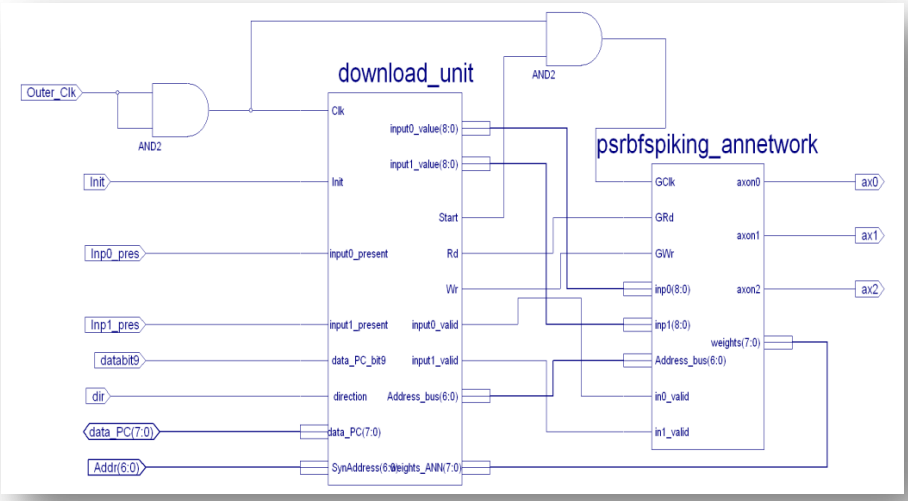


Figura 5.10 Schema de conectare a modulelor principale, în vederea punerii în funcţiune a reţelei neuronale

Simulările au fost executate pentru fiecare modul, pentru care a fost posibil, în cadrul programului Modelsim XE Starter, care este o versiune limitată, dar gratuită al unui pachet de programe de simulare extrem de puternice.

În continuare, se va prezenta fiecare modul în parte, detaliind rolul şi funcţionare acestora, cu câteva exemple de rezultate de simulare.

5.4.2.2. Neuronul de intrare

Acest modul este generat de două ori, pentru fiecare variabilă de intrare. El conţine un modul de memorie BRAM iniţializată cu valorile funcţiilor domeniilor receptive locale, respectiv un modul funcţional. Schema din Figura 5.11, este o variantă a acestui modul de neuron de intrare, care a fost folosit pentru testarea funcţionării, şi conţine multe ieşiri, care sunt prezente doar pentru a putea urmări în cadrul simulării variabilele interne.

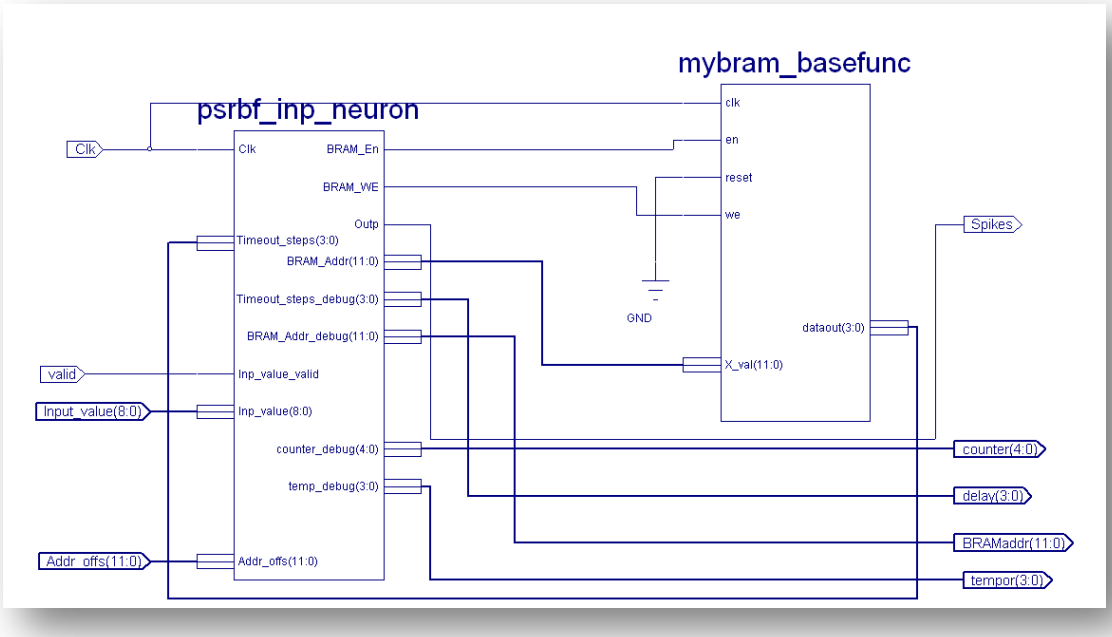


Figura 5.11 Schema de conexiune a unui neuron de intrare

Modulul de control funcţional (Figura 5.12) este un automat cu mai multe stări şi execută următoarele operaţii: pentru valoarea de intrare sosită [Inp_value (8:0)], adresează memoria BRAM, activând în ordinea necesară şi semnalele de control ale acesteia (Rd, En) şi extrage valoarea decalajului corespunzător, dat de funcţia stocată. Folosind această valoare se generează un impuls pre-sinaptic cu decalaj temporal d^k. Acest lucru se întâmplă în mod paralel pentru fiecare dintre cei 24 de neuroni de intrare ai celor două blocuri de intrare.

Diagramele din figurile de mai jos prezintă testarea prin simulare – conform stimulilor întocmiţi – a subansamblelor neuronilor de intrare. Se poate urmări cum modulul de memorie livrează valorile de decalaj, iar apoi generarea impulsurilor decalate, adică codificarea unei valori de intrare.

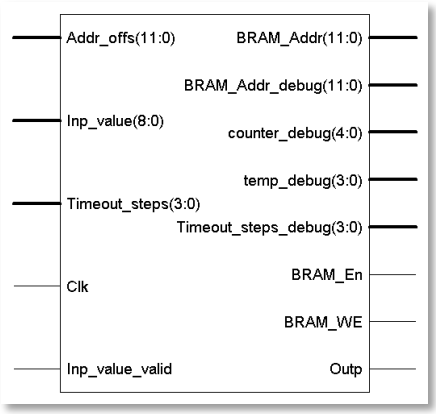


Figura 5.12 Modulul de control funcţional

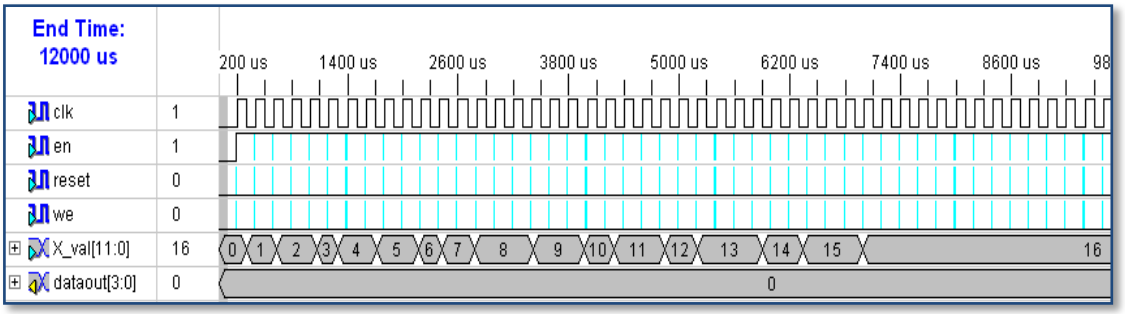


Figura 5.13 Stimul de intrare pentru simularea modulului BRAM

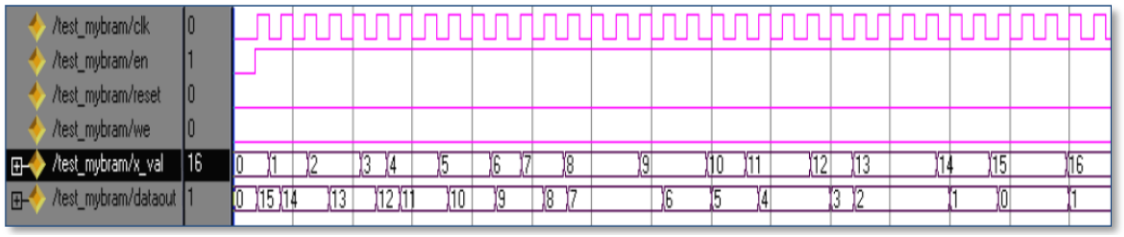


*Figura 5.14 Rezultat de simulare, valori ale funcţiilor domeniilor receptive locale (semnalul **dataout**) pentru valori crescătoare ale variabilei de intrare (semnalul **xval**)*

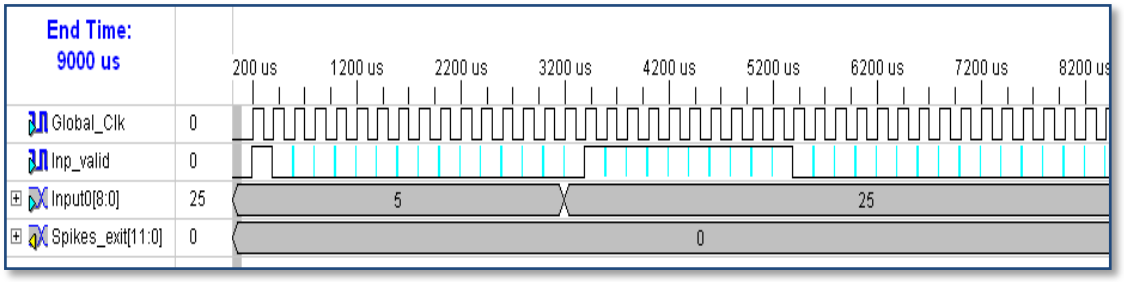


Figura 5.15 Stimul pentru testarea codării intrării în impulsuri decalate temporal

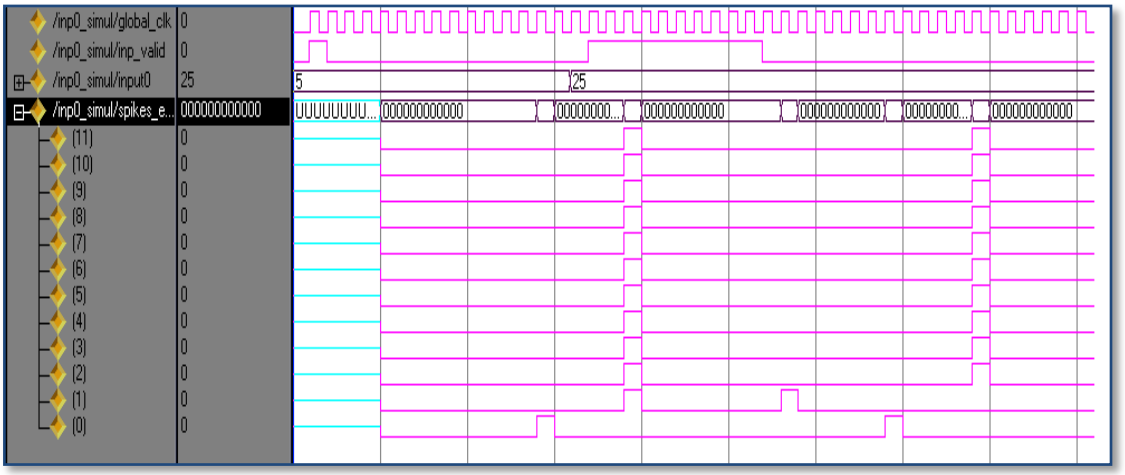


Figura 5.16 Rezultat de simulare al codării intrării în impulsuri. De exemplu, pentru valoarea de intrare 25, se vor activa neuronii ne intrare 0 şi 1, cu decalaje de 5 respectiv 12 paşi (cicluri de tact), iar ceilalţi neuroni emit impuls doar la sfârşitul ciclului de funcţionare (după 16 paşi), generat de un modul dedicat pentru acest scop.

5.4.2.3. Modulul sinapsă

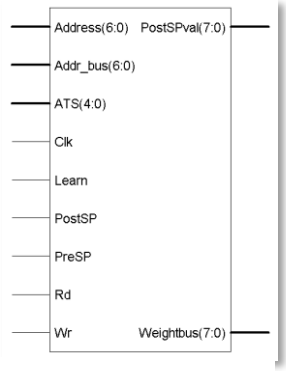


Figura 5.17 Modulul sinapsă

Acest modul este responsabil pentru amplificarea sau atenuarea impulsului pre-sinaptic, transmiţând o valoare ponderată către soma neuronului. Totodată aici este implementat şi algoritmul de învăţare. Învăţarea se face ne-supervizat, conform unei versiuni adaptate a regulilor Hebb, descrise în capitolele anterioare. Sinapsele verifică eticheta temporală a fiecărui impuls pre-sinaptic, şi modifică valoarea ponderii conform regulilor Hebb în varianta modificată. Astfel, de-a lungul unui ciclu de funcţiune de 16 perioade ale semnalului de tact, pentru impulsurile, care au sosit cu maxim 5 perioade înaintea activării post-sinaptice, fortificare ponderală est drastică. În cazul acelor impulsuri, care au sosit cu un decalaj temporal mai mare de 5 dar mai mic de 10 perioade de tact, pondera creşte moderat, iar pentru restul impulsurilor ponderea scade uşor. Dacă impulsul pre-sinaptic soseşte după apariţia impulsului axonal, ponderea sinapsei respective descreşte abrupt.

Intrările şi ieşirile acestui modul sunt semnale ce servesc la controlul procesului de învăţare, respectiv la adresare sinapsei în vederea iniţializării şi citirii valorilor ponderilor.

5.4.2.4. Modulul de some al reţelei neuronale pseduo-RBF-pulsative

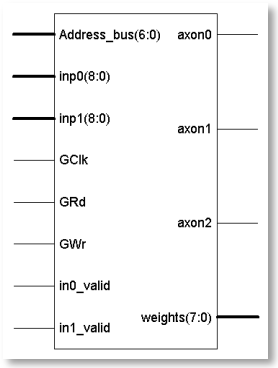


Figura 5.18 Modulul de some

Soma este modulul care este responsabil pentru sumarea impulsurilor incidente, decalate temporal şi amplificate de către sinapse. Fiecare modul de somă primeşte 24 de intrări, de câte 8 biţi, din care calculează valoare potenţialului de membrană. La depăşirea valorii de prag, se emite un impuls de activare, iar valoarea potenţialului de membrană se resetează la valoarea de hiperpolarizare prestabilită.

5.4.2.5. Modulul de comunicaţie cu calculatorul

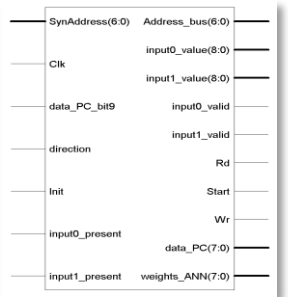


Figura 5.19 Modulul de comunicaţie cu calculatorul

Pentru a putea urmări evoluţia procesului de învăţare, precum şi pentru realizarea iniţializării reţelei, s-a implementat un modul special, destinat comunicaţiei dintre un calculator şi placa de dezvoltare FPGA. Această comunicaţie se desfăşoară paralel pe două căi, şi anume prin portul paralel şi printr-o placă de achiziţii date National Instruments conectabilă pe magistrala USB. Tot prin acest modul se prezintă şi variabilele de intrare reţelei neuronale hardware ce funcţionează în circuitul FPGA.

5.5. Realizări de reţele neuronale neuromorfe utilizând microcontrolere soft-core înglobate în circuitele FPGA

În trecut, sistemele de calcul au fost construite din componente separate cum ar memorii, elemente de I/O, circuite FPGA, etc. Totodată, diferitele programe de proiectare au fost (unele sunt şi astăzi) utilizate pentru a specifica funcţionalitatea individuală a acestor componente, precum şi pentru a le integra să funcţioneze ca un sistem complet.

Componenta centrală a acestor sisteme, procesorul, a fost (şi mai este) cel mai des o unitate separată. Interfaţarea acestuia cu celelalte componente – logică, memorii, I/O – este realizată cu ajutorul pachetelor software mai sus menţionate. Partea software a acestor sisteme de calcul a fost (şi este) proiectată cu programe specializate pentru dezvoltarea se soft-uri şi sisteme de operare. Interfaţarea hardware-software este realizată prin utilizarea unei combinaţii ale acestor unelte menţionate şi în unele cazuri unelte de co-verificare sau co-simulare.

În a doua parte a anilor 1990 multe funcţii logice au fost integrate în dispozitive singulare cum ar fi familiile de circuite 4K sau Virtex™ ale firmei Xilinx. Memoriile de capacitate mare nu sunt încă disponibile în aceste circuite, dar memoriile încorporate se dezvoltă odată cu apariţia noilor familii de astfel de circuite şi cu dezvoltarea tehnologiei de fabricaţie a acestora. Aceste progrese (Figura 5.20) au făcut posibilă ca o parte mai importantă a proiectării şi implementării logice să se realizeze într-o singură unitate, eliminând nevoia de interfaţare dintre componente. Totuşi nu s-a ajuns la schimbări radicale în ceea ce priveşte dezvoltarea în domeniul unităţilor centrale de prelucrare (CPU).

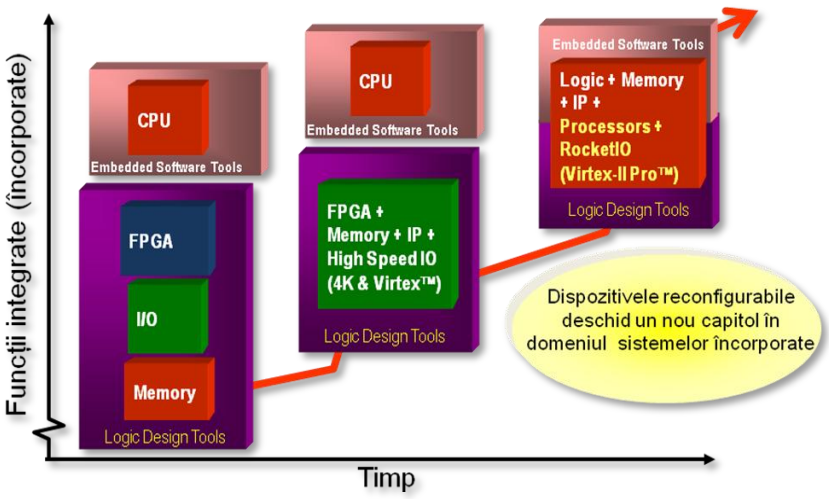


Figura 5.20 Evoluţia funcţiilor integrate în circuitele FPGA

Odată cu apariţia circuitelor Platform FPGA şi a celor SoC ASIC (System-on-Chip Application Specific Integrated Circuit), fuzionarea tuturor acestor funcţionalităţi discrete a devenit realitate. Utilizând potenţialul acestor sisteme, care sunt complet reconfigurabile hardware şi software, se obţine nu numai un produs finit excepţional dar şi o platformă de dezvoltare foarte avantajoasă care necesită, însă cooperarea dintre elementele de siliciu, de module predefinite (proiecte de circuite şi programe) şi îmbinarea coezivă a acestora cu o suită de unelte de proiectare bine puse la punct.

Dezvoltarea şi progresele recente din acest domeniu, împreună cu apariţia procesoarelor încorporabile în circuitele FPGA, au făcut ca alegerea acestora ca platformă pentru implementarea hardware a RNP parţial serializată să fie evidentă. În continuare se vor prezenta contribuţii ale tezei în acest domeniu, şi anume implementări de exemple aplicative şi teste benchmark clasice utilizând procesoare încorporate pe 8 respectiv pe 32 de biţi.

5.5.1. Microcontrolerul soft-core Xilinx PicoBlaze

PicoBlaze, cunoscut şi ca KCPSM3, este un microcontroler RISC cu o amprentă de siliciu extrem de redusă, de numai 96 de slice-uri dintr-un FPGA din familia Xilinx Spartan3 (optimizat pentru această familie) ceea ce reprezintă aproximativ 1,25% din totalul de 7680 de astfel de structuri logice disponibile de exemplu în circuitul XC3S1000. Cea mai mare frecvenţă de tact la care poate funcţiona este de 87MHz, ceea ce rezultă o capacitate de calcul respectabilă de ~43.5 MIPS.

Toate aceste sunt performanţele unui microcontroler de tip soft-core, care este livrat de furnizorul Xilinx în forma unui cod VHDL, putând fi încorporat în orice proiect destinat implementării pe un circuit FPGA.

După cum se poate urmări şi în Figura 5.21, PicoBlaze dispune de o memorie RAM internă de uz general de 64 de octeţi şi este capabil să comunice cu restul componentelor din proiectul în care este încorporat prin 256 porturi de intrare şi 256 porturi de ieşire de câte 8 biţi fiecare. Portul accesat este adresat prin ieşirea de 8 biţi PORT_ID. Operaţiile de citire porturi sunt validate de impulsuri pe ieşirea READ_STROBE iar cele de scriere porturi de impulsuri pe ieşirea WRITE_STROBE. Valorile astfel scrise/citite pot fi valori constante sau conţinutul unuia dintre cei 16 regiştrii (de un octet) interni ai procesorului.

PicoBlaze dispune de un sistem de întrerupere ne-vectorizat foarte simplu, cu o singură intrare de semnalizare a acesteia. Toate cele ~50 de instrucţiuni sunt executate întotdeauna în două cicluri de tact şi pot utiliza o stivă cu adâncimea de 31.

Programele pentru acest procesor sunt scrise într-un limbaj de asamblare cu instrucţiuni specifice acestuia. Asamblorul livrat de producător generează un cod VHDL care instanţiază o memorie ROM (utilizând modulele BRAM din FPGA) iniţializată cu codurile de instrucţiuni şi datele care

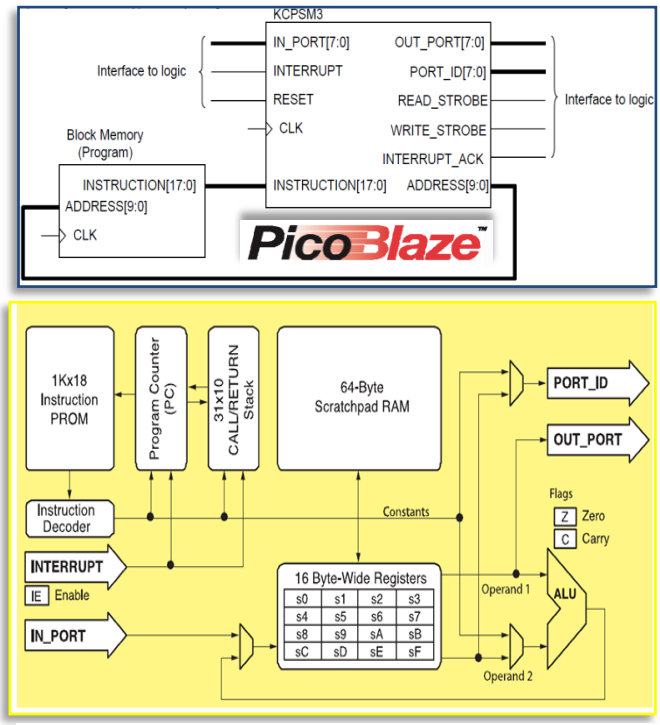


Figura 5.21 Diagrama bloc a microcontrolerului soft-core Xilinx PicoBlaze încorporat în RNP implementate

implementează programul scris. Acest cod VHDL se poate integra cu uşurinţă în proiectul care conţine deja cel puţin un microcontroler PicoBlaze, funcţionând ca memorie de program pentru acesta.

PicoBlaze poate adresa 1024 locaţii a memoriei program, aceasta fiind una din limitările sale principale. Totuşi, în multe cazuri este foarte util în a executa algoritmi ce ar necesita multe resurse hardware reconfigurabile. Pentru testarea şi simularea funcţionării programelor scrise, am utilizat software-ul Mediatronix pBlaze IDE v3.6 care este un debugger simplu dar eficient. Problema majoră a acestui utilitar este faptul, că nu este 100% compatibil din punct de vedere al sintaxei programului cu asamblorul PicoBlaze.

5.5.2. Implementare aplicativă: Detector de componente de frecvenţă

Scopul acestei aplicaţii este de a demonstra, că reţelele neuronale pulsative implementate pot rezolva probleme inginereşti. Rezultatele prezentate în cele ce urmează trebuie văzute ca o demonstraţie a capabilităţilor acestor sisteme, şi nu ca o expunere a unei rezolvări ieşite din comun a problemei în cauză.

Formularea problemei: sistemul dezvoltat trebuie să fie capabil să înveţe a separa dintr-un semnal analog (perturbat de zgomot) discretizat, trei componente de frecvenţă dominante, în jurul a trei valori predeterminate.

5.5.2.1. Reţeaua neuronală pulsativă implementată pe FPGA

S-a implementat o RNP cu două intrări şi trei ieşiri, după cum arată şi schema bloc din Figura 5.22. Această reţea trebuie să clasifice corect perechile de valori (numerice, întregi, pozitive) care definesc un spaţiu de intrări de 256x256=65536 puncte (Figura 5.23). Fiecare punct al acestui spaţiu de intrări reprezintă câte o componentă de o anumită frecvenţă şi putere din componenţa semnalului analizat. Antrenarea reţelei s-a efectuat în aşa fel, încât fiecare ieşire să se activeze atunci, când semnalul respectiv conţine o componentă a cărei frecvenţă şi putere se află în vecinătatea valorii prescrise (punct de focus), adică să aproximeze prezenţa acestei componente cu o anumită histereză.

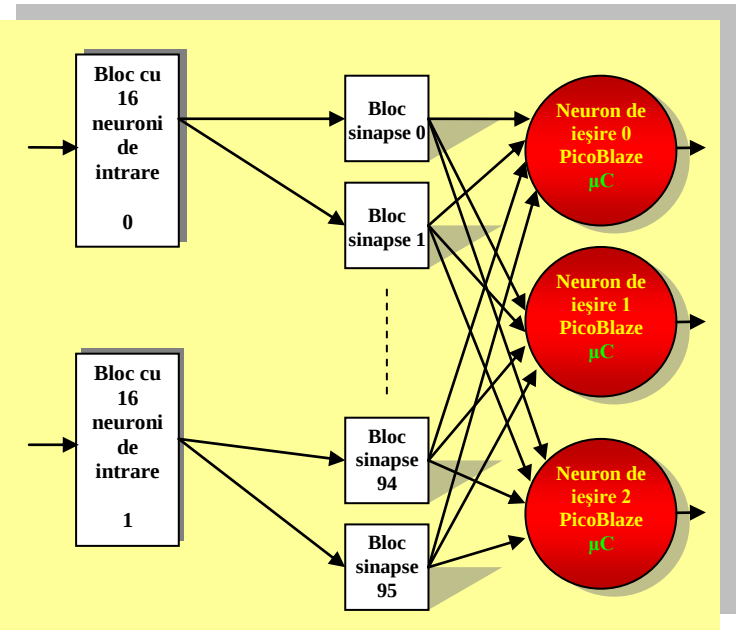


Figura 5.22 Schema bloc a reţelei pseudoRBF-spiking implementată

Parametrii reţelei neuronale implementate:

- *Numărul intrărilor:* $n = 2$
- *Nr. neuronilor de intrare/intrare:* $n_{intr}=12$
- *Nr. neuronilor de intrare activi/intrare:* $n_{act} = 3$
- *Numărul paşilor dintr-un cadru temporal:* $T=16$

5.5.2.2. Implementarea neuronilor de intrare – codificarea variabilelor de intrare în impulsuri decalate temporal

Pentru a putea specifica valorile de intrare reţelei neuronale o decizie a fost luată asupra algoritmului de codificare a acestora în impulsuri. Această codificare putea fi implementată în aşa fel încât să fie executată on-chip, dar s-a considerat, că această metodă ar utiliza excesiv de multe resurse reconfigurabile sau ar fi executată prea lent de un procesor încorporat. Aşadar codificarea s-a efectuat parţial a-priori, după cum se

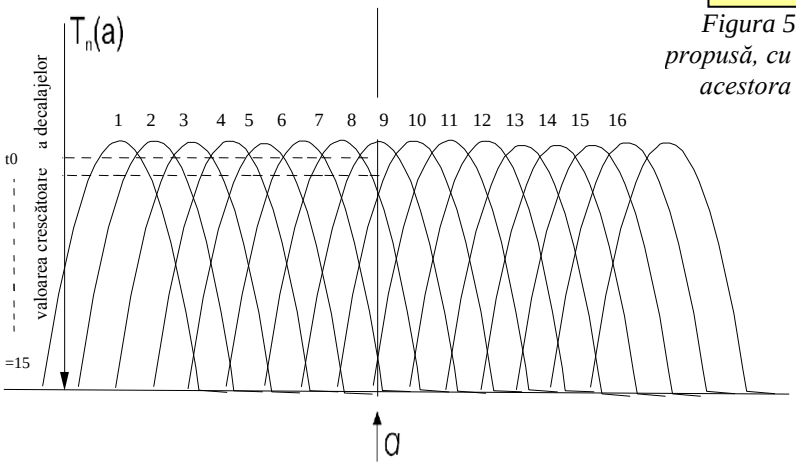


Figura 5.24 Metoda de codificare a valorilor de intrare în impulsuri decalate temporal

pentru a simplifica implementarea am ales o aproximare a acestora cu funcţii triunghiulare. Figura 5.25 arată, cum o valoare arbitrară (din domeniul de intrare cu valori întregi între 0-255) activează patru domenii receptive (marcate cu linii punctate), funcţia corespunzătoare neuronului 7 fiind activat cu valoarea cea mai mare. Acest lucru va însemna, că acest neuron de intrare va emite primul impuls de activare, iar restul de neuroni activi pentru această valoare se vor activa într-un pas ulterior. Codificarea este realizată într-un cadru temporal de 16 paşi, corespunzători cu un

Reţeaua se poate considera, ca fiind una hibridă, putând fi numită pseudo-RBF-spiking, dat fiind faptul, că modalitatea de codificare a valorilor de intrare utilizează funcţii de bază ca şi reţelele neuronale de tip Radial Basis Function (RBF).

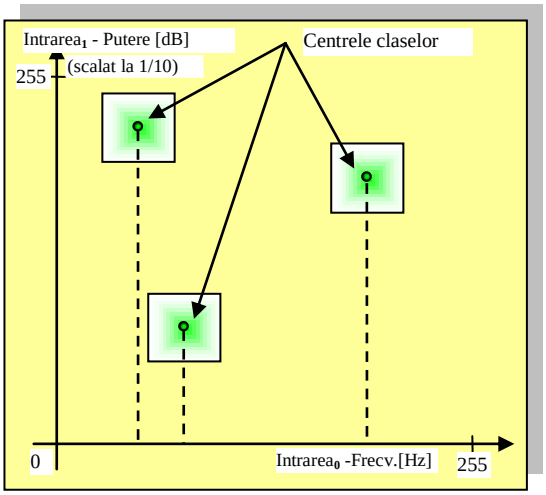


Figura 5.23 Spaţiul intrărilor pentru problema propusă, cu exemple de centre de focus şi vecinătăţile acestora care vor fi învăţate de RNP din FPGA

prezintă în continuare.

Scopul este de a găsi o variantă de acoperire a spaţiului intrărilor cu domenii receptive (funcţii de activare) - variind aria de suprapunere a acestora, în aşa fel, încât pentru fiecare valoare de intrare să se activeze un număr redus, dar egal de funcţii (de exemplu 4). În cazul ideal, aceste funcţii ar trebui să fie Gaussiene, după cu arată Figura 5.24, dar

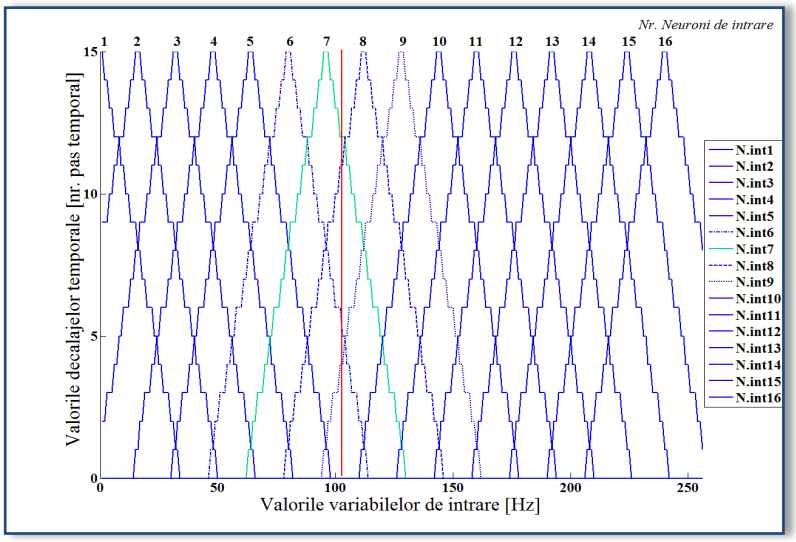


Figura 5.25 Funcţiile triunghiulare ale domeniilor receptive a neuronilor de intrare, cu scalarea decalajelor temporale

număr egal de perioade de tact la care funcţionează acest circuit. Astfel, un neuron de intrare activat cu valoarea maximă posibilă a funcţiei triunghiulare ataşate de 15 va emite impuls de ieşire în primul pas temporal al cadrului (pasul 0).

Aşadar, această alegere a valorilor de decalaj temporal, rezultă într-un necesar de patru biţi/pe valoare pentru stocarea acestora. Valorile au fost calculate cu ajutorul unui program scris în Matlab prin care se poate varia numărul funcţiilor de activare şi aria acestora.

Acelaşi program Matlab generează şi codul VHDL necesar pentru iniţializarea modulelor BlockRAM din circuitul FPGA, utilizate pentru stocare valorilor de decalaj temporal.

După cum a fost prezentat şi la subpunctul 3.1.3.1 elementele BlockRAM ale circuitelor FPGA pot fi configurate în mai multe feluri, variind numărul biţilor de pe magistrala de adrese respectiv de pe cea de date. Pentru a stoca valorile de decalaj temporal am ales componenta cu numele RAMB16_S36_S36, care este un modul BRAM cu port dual şi o capacitate de 512x32 biţi. Arhitectura cu port dual permite ca două locaţii de memorie să fie accesate simultan. Deoarece valorile stocate sunt iniţializate odată cu programarea circuitului FPGA, acest modul va fi utilizat ca şi cum ar fi de fapt o memorie ROM, efectuând numai operaţii de citire, evitând astfel eventualele conflicte de adresare la scriere. Conectând două astfel de module în paralel, devine posibil să se obţină în aceeaşi perioadă de tact cei 128 biţi reprezentând decalajele temporale pentru cele două intrări ale RNP, rezultat obţinut conform ecuaţiei următoare:

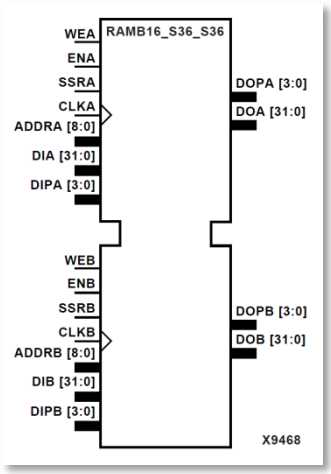


Figura 5.26 Modulul BlockRAM utilizat la stocarea decalajelor temporale

$$\sum \left\langle d^{ij} \right\rangle = \sum_{n} n_{\text{intr}} * \upsilon \qquad \textit{Ec. 70}$$

unde am introdus următoarele notaţii:

n – este numărul intrărilor RNP, în cazul acestei aplicaţii 2

n_{intr} – este numărul neuronilor de intrare care codifică valorile unei intrări ale RNP, aici 16

υ - este numărul de biţi pe care este reprezentat un decalaj temporal, în cazul de faţă 4

$<d^{ij}>$ – este numărul de biţi necesari pentru reprezentarea tuturor decalajelor temporale ale impulsurilor ce se propagă de la neuronii pre-sinaptici i alocaţi unei intrări ale RNP spre neuronul post-sinaptic j, (4x16=64).

Valoarea variabilei de intrare este folosită pentru adresarea memoriei, accesând locaţii diferite pe cele două porturi ale acesteia, utilizând un offset corespunzător (Figura 5.27) la calcularea adresei pe portul B. Am realizat, astfel cea mai eficientă utilizare posibilă a unităţilor BlockRAM, deoarece pentru a acoperi spaţiul valorilor de intrare am folosit 100% din capacitatea acestora.

Aceste valori citite din modulul de memorie - a cărei scheme bloc este prezentată de Figura 5.26 - vor fi folosite de circuitele care implementează neuronii de intrare pentru a codifica valoarea de intrare cu un impuls emis în pasul temporal cu numărul egal cu valoarea decalajului (paşii cu numărul 0 la 15).

Pentru a garanta, însă, că pentru fiecare valoare de intrare se va activa un număr egal de neuroni de intrare, a fost nevoie de un artificiu. Odată cu mărirea ariei de suprapunere a funcţiilor, ar fi fost nevoie sau de mărirea numărului acestor funcţii pe intrare sau de mărirea numărului paşilor dintr-un

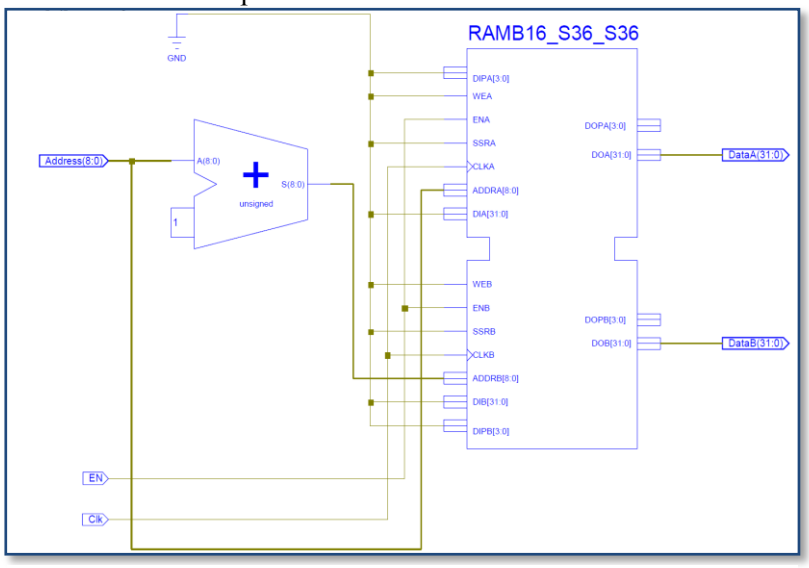


Figura 5.27 Schema de conexiune unui bloc BRAM al unui neuron de intrare

cadru temporal, pentru a menţine forma strict triunghiulară a acestora. Aceste modificări, însă ar fi rezultat o situaţie, în care un modul BRAM nu ar fi fost suficient pentru stocarea tuturor valorilor de decalaj temporal, pentru că valoarea υ din Ec. 70 ar fi crescut. Aşadar, scalarea valorilor de decalaj din intervalul [0...35] necesar garantării condiţiei mai sus amintite în intervalul original de [0...15] s-a dovedit a fi cea mai simplă şi eficientă soluţie. Aceasta a dus la modificarea uşoară a formei funcţiilor de activare, după cum se poate vedea şi în Figura 5.25 ceea ce nu influenţează, însă precizia codificării.

În continuare se va prezenta testarea prin simulare a circuitelor ce implementează un neuron de intrare respectiv un întreg bloc de neuroni de intrare pentru o variabilă de intrare a RNP. Simulările au fost realizate cu simulatorul integrat în mediul de dezvoltare Xilinx ISE 10.1.

În Figura 5.28 se pot urmări semnalele de intrare utilizate pentru testarea funcţionării unui modul de codificare a intrărilor RNP. Se poate observa, că la intervale de 16 cicluri de tact s-a modificat valoarea semnalului de intrare (valorile 0, 2 şi 14) reprezentând decalajul temporal cu care trebuie să apară impulsul de ieşire, valoare citită în modul normal de operare din memoria BlockRAM. A se nota de asemenea prezenţa semnalului *delay_valid* care este inactiv pe perioade de câte două cicluri de tact, pentru a semnala că valoarea de decalaj nu este validă pe timpul citirii acesteia din memoria internă.

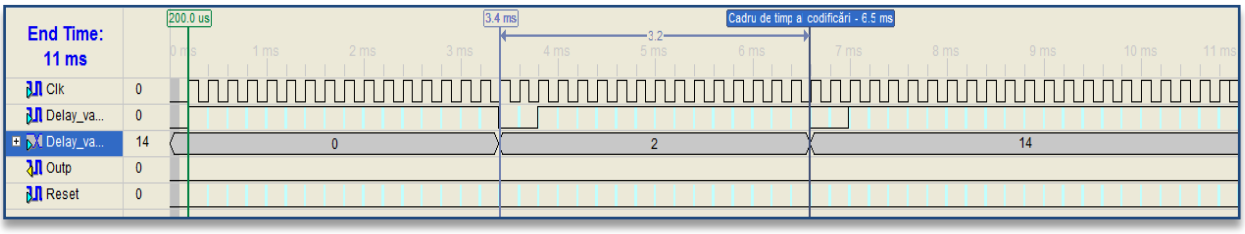


Figura 5.28 Stimul de intrare pentru simularea funcţionării unui modul de codificare a intrărilor RNP

Ieşirile circuitului ce implementează modulul de codificare a valorilor de intrare a RNP la stimulul prezentat anterior (Figura 5.28) sunt afişate pe Figura 5.29. Semnalul de notat în această figură este *outp* pe care vor apare impulsurile decalate cu atâtea cicluri de tact cât a arătat valoarea de decalaj specificată pe intrarea *delay_value*.

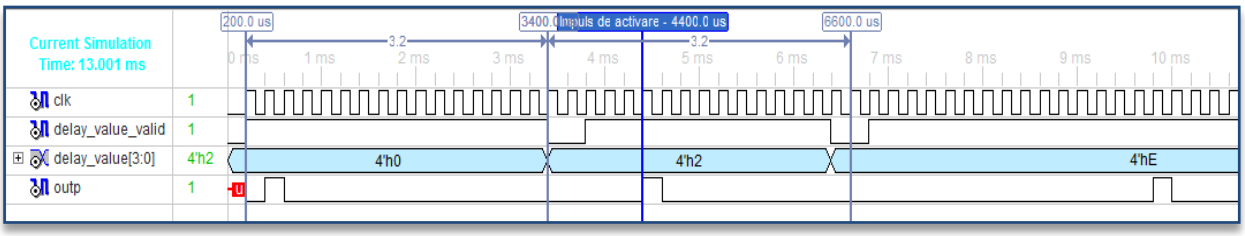


Figura 5.29 Rezultatul simulării unui modul de codificare a valorilor de intrare a RNP

5.5.2.3. Blocul neuronilor de intrare

În acest circuit sunt instanţiate câte 16 neuroni de intrare pentru fiecare intrare a RNP. Reţeaua neuronală ce implementează aplicaţia curentă având două intrări (valori întregi reprezentate pe 8 biţi), acest modul a fost generat în două exemplare. Structura acestui circuit (Figura 5.30) înglobează şi cele două memorii BRAM care stochează valorile pre+calculate ale funcţiilor de activare a celor 2x16 neuroni de intrare.

Modulul funcţional al acestui circuit (Figura 5.31) este de fapt un automat cu mai multe stări, care extrage din memoriile BRAM valorile de decalaj d^k corespunzătoare funcţiilor de activare iar apoi generează câte un impuls pre-sinaptic decalat în timp cu aceeaşi valoare (un număr egal de paşi temporali din cadrul

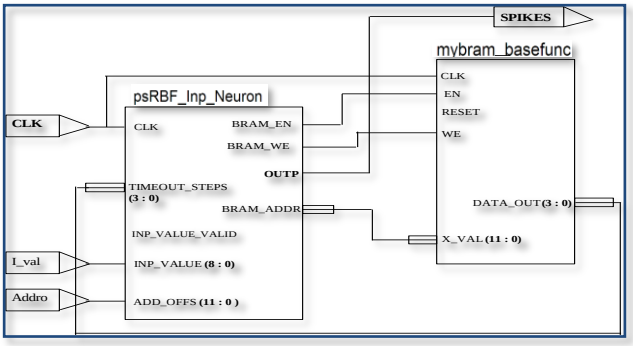


Figura 5.30 Structura blocului neuronilor de intrare

temporal, adică un număr de cicluri de tact) în paralel pentru fiecare dintre cei 16 neuroni de intrare ce codifică variabila de intrare. Pentru a ilustra funcţionarea acestui modul, diagramele de timp ce urmează vor arăta rezultatele simulării acesteia. Astfel, în Figura 5.32 se prezintă semnalele de intrare pentru blocul de neuroni de intrare ce codifică una din variabilele de intrare. Se pot observa perechile valorilor de intrare prezentate reţelei neuronale pe intrările *Inp0* şi *Inp1* respectiv semnalele de validare ale acestora.

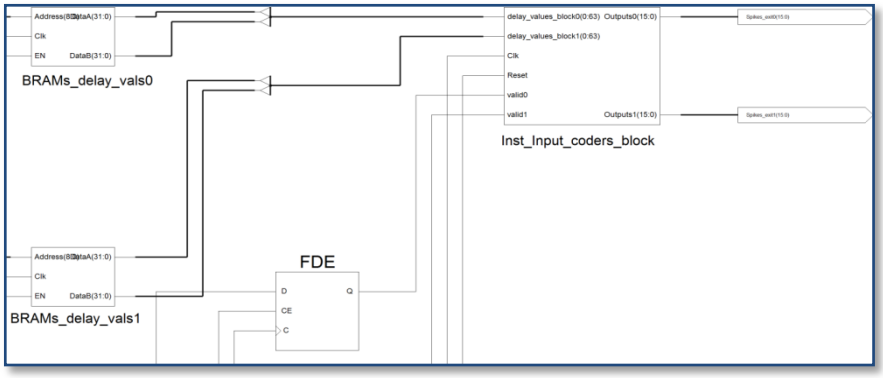


Figura 5.31 Schema de conexiune blocului neuronilor de intrare

Tabelul 11 Utilizarea FPGA a blocului neuronilor de intrare

Logic Utilization	Used	Available	Utiliz.
Nr. of Slices	917	8672	10%
Nr. of Slice Flip Flops	1582	17344	9%
Nr. of 4 input LUTs	1255	17344	7%
Nr. of bonded IOBs	52	250	20%
Nr. of BRAMs	2	28	7%
Nr. of GCLKs	1	24	4%

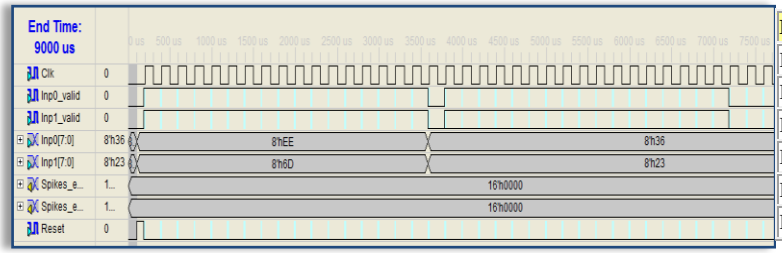


Figura 5.32 Stimul de intrare pentru blocul de neuroni de intrare

Figura 5.33 prezintă rezultatul simulării blocului de neuroni de intrare ce codifică una din variabilele de intrare a RNP, putându-se observa impulsurile de ieşire decalate apărute în urma prezentării valorilor de intrare, precum şi offset-ul de adresare a portului B al memorie BRAM, conform funcţionării descris la punctul anterior.

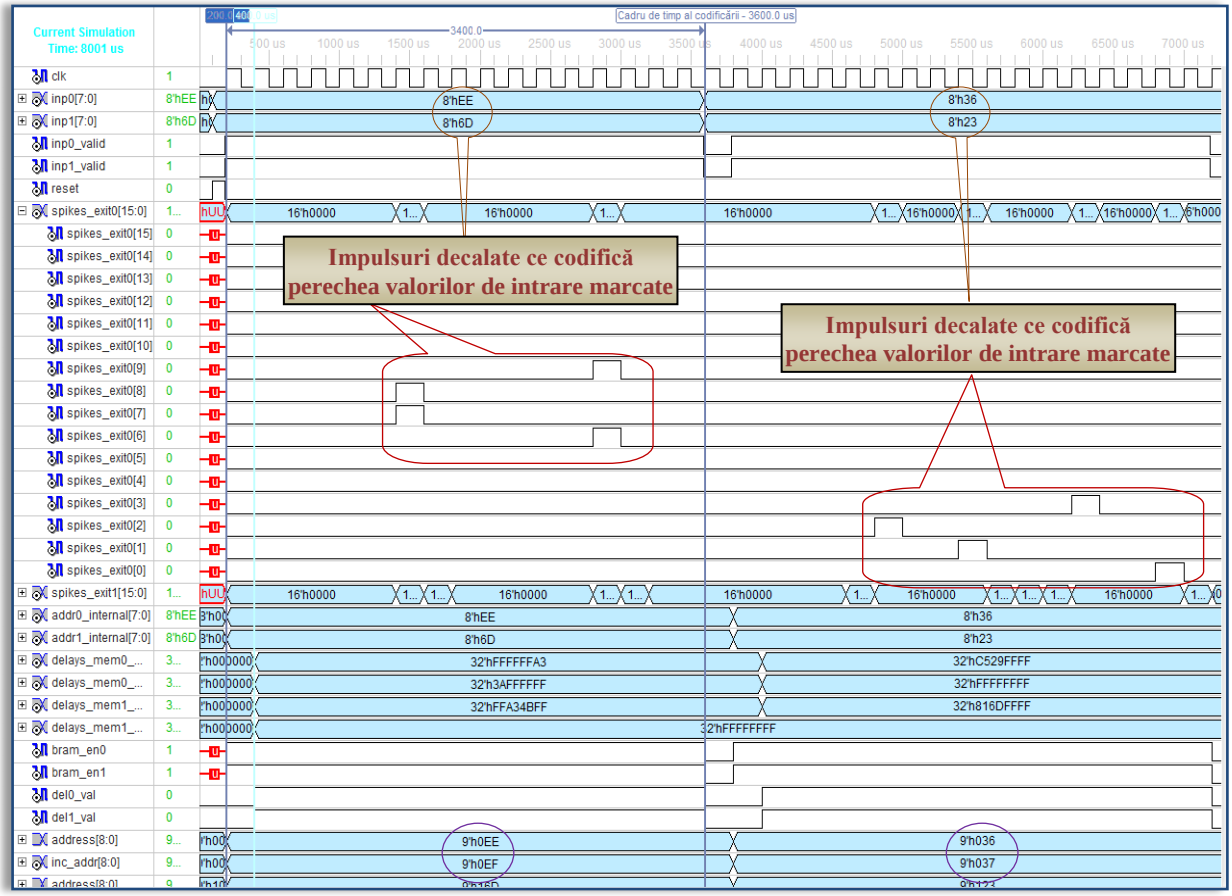


Figura 5.33 Rezultatul simulării blocului de neuroni de intrare ce codifică una din variabilele de intrare a RNP

5.5.2.4. Modulul de implementare a sinapselor

Modulul ce implementează sinapsa unui neuron este responsabilă pentru amplificarea sau atenuarea impulsurilor pre-sinaptice emise de către neuronul de intrare la care este conectat. Astfel se generează o valoare post-sinaptică ponderată ce va fi transmisă spre soma neuronului de ieşire la care aparţine sinapsa respectivă.

Acest modul este de asemenea cel, care implementează regulă nouă de învăţare supervizată care este o variantă adaptată a metodei Hebb, bazată pe reguli de decalaje temporale, după cum este prezentat în cele ce urmează. Sinapsele vor testa amprenta temporală (numărul pasului temporal, notat cu *TS* în Tabelul 12) a fiecărui impuls pre-sinaptic şi vor ajusta valorile ponderilor stocate în concordanţă cu regulile de învăţare prezentate în Tabelul 12, luând în calcul şi valorile prescrise ale stării (activate sau nu) a neuronilor de ieşire (valoare axonală) pentru valorile variabilelor de intrare actuale.

Parametrii modelului neuronal:

- Valoarea maximă a ponderii: $\beta_i^{jk}{}_{max} = 255$
- *Potenţialul de membrană – valoarea maximă în aplicaţia curentă:* $\mu_{actmax} = n_{act} * \beta_i^{jk}{}_{max} * T * n$
- *Potenţialul de repaus:* $\mu_{rep} = \mu_{actmax} * 0.2$
- *Potenţialul de prag:* $\theta = \mu_{actmax} * 0.8$
- *Potenţialul de hiperpolarizare:* $\mu_{hip} = \mu_{actmax} * 0.1$
- *Potenţialul de scurgere:* $\mu_{scu} = \mu_{actmax} * 0.05$
- *Potenţialul de inhibare:* $\mu_{inhib} = \mu_{rep} - \mu_{hip} / 2$

Figura 5.34 Parametrii modelului neuronal

Aşadar, conform tabelului de mai jos, şi a condiţiilor descrise în Ec. 71, Ec. 72 respectiv Ec.73, în decursul unui cadru de timp de 16 cicluri de tact, pentru acele impulsuri pre-sinaptice care au sosit cu cel mult 5 paşi temporali mai devreme decât activarea post-sinaptică (a neuronului de ieşire corespunzător) şi dacă valoarea prescrisă este de activare (pentru a corela activarea neuronului de ieşire cu valorile de intrare dintr-o clasă anume), atunci ponderea sinapsei respective este mărită drastic.

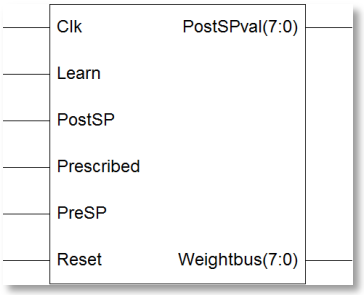


Figura 5.35 Intrările şi ieşirile modulului sinapsă

Tabelul 12 Algoritmul de învăţare propriu, implementat în hardware

Impuls pre-sinaptic prezent	Valoare axonală - ieşire NP	Reguli tem-porale	Valoare de ieşire prescrisă	Adaptare pondere	Calculare model somă
0	0	-	0	-	-
0	0	-	1	-	Σ
0	1	-	0	-	-
0	1	-	1	-	Σ
1	0	-	0	-	-
1	0	PreTS << PostTS	1	↑↑	Σ
1	0	PreTS < PostTS	1	↑	Σ
1	1	PreTS > PostTS	0	↓	-
1	1	PreTS >> PostTS	0	↓↓	-
1	1	-	1	-	Σ

$$if \cdots\cdots \sum \beta_i^{jk} = 0 \cdots\cdots then \cdots\cdots \gamma_i = \lambda_i \qquad \text{Ec. 71}$$

$$if \cdots\cdots \sum \beta_i^{jk} \geq \theta \cdots\cdots then \cdots\cdots \gamma_i = \delta_i \qquad \text{Ec. 72}$$

$$if \cdots\cdots \gamma_{i\pm m}{}^k = \gamma'_{i\pm m}{}^k \cdots\cdots then \cdots\cdots \gamma_i = \lambda^k \qquad \text{Ec.73}$$

În cazul impulsurilor care au un decalaj temporal între 5 şi 10 cicluri de tact şi condiţii prescrise identice, ponderea este mărită moderat. Pentru acele impulsuri pre-sinaptice care sosesc după apariţia activării axonale şi în prezenţa activării prescrise a acesteia, ponderea sinapsei va fi diminuată proporţional cu diferenţa dintre aceste două momente.

Intrările şi ieşirile acestui modul (Figura 5.35) sunt utilizate pentru a controla procesul de învăţare, respectiv pentru a putea citi şi iniţializa valorile ponderilor. În aplicaţia de faţă, proiectul

implementat pe circuit FPGA conţine un total de 96 de sinapse, câte una pentru conexiunile dintre cei 32 de neuroni de intrare către cei trei neuroni de ieşire ale RNP, fiecare stocând valoarea ponderii pe un număr de 8 biţi. Aceste ponderi sunt iniţializate cu valori aleatore ca apoi să fie adaptate conform regulilor de învăţare descrise mai sus în decursul mai multor cicluri de antrenare, în decursul cărora la intrările reţelei neuronale implementate hardware se prezintă perechi de variabile de intrare – de către o Unitate de Monitorizare şi Control, aceasta citindu-le dintr-o memorie BRAM separată – reprezentând elementele unui set de antrenare.

5.5.2.5. Modulul neuronilor de ieşire – implementarea somei cu multiple microcontrolere încorporate Xilinx PicoBlaze

Modulul ce implementează soma neuronală este responsabilă pe de o parte pentru sumarea valorilor post-sinaptice ponderate şi decalate temporal. Toate cele trei astfel de module ale aplicaţiei de detectare a componentelor de frecvenţă recepţionează câte 32 de impulsuri a reprezentate pe 8 biţi fiecare.

Însumând toate aceste valori şi urmând acelaşi ritm al paşilor temporali ca şi modulele de codificare a intrărilor şi sinapsele, somele calculează potenţialul de membrană (PM) a neuronului artificial, notat cu μ în prezentarea parametrilor modelului neuronal din Figura 5.34. Dacă acest PM – iniţializat la o valoare de repaus – depăşeşte o valoare de prag predefinită (THS-thershold), neuronul va emite un impuls axonal (sau impuls post-sinaptic), apoi PM va fi resetat la o valoare mai mică decât valoarea de repaus, plasând neuronul într-o stare specială, numită de hiperpolarizare. În această stare neuronul nu va fi capabil să emită un nou impuls axonal, revenirea la starea de repaus făcându-se într-un număr predefinit de paşi temporali.

Notaţii ale variabilelor şi stărilor somei implementate

- θ_k - *Potenţialul de prag al somei* **k**
- μ_k - *Potenţialul de membrană al somei* **k**
- γ'_k - *Soma* **k** *este activată*
- δ_k - *Soma* **k** *este în stare de hiperpolarizare*
- λ_k - *Soma* **k** *este in stare de drenare curent*

Figura 5.36 Notaţii în descrierea funcţionării somei

Fazele procesării paşilor temporali în RNP implementată

- ✓ ***Faza α:*** *neuronii de intrare generează impulsurile decalate temporal care codifică valorile de intrare*
- ✓ ***Faza β:*** *sinapsele ponderează aceste impulsuri incidente şi transmit valorile post-sinaptice modulelor somă*
- ✓ ***Faza γ:*** *somele acumulează setul actual de intrări post-sinaptice, apoi rulează algoritmul corpului neuronal*

Figura 5.37 Divizarea unui pas temporal în procesarea RNP implementate hardware

Dacă nu se recepţionează nici o valoare post-sinaptică diferită de zero într-un anumit pas temporal, atunci valoarea PM se va diminua cu valoarea de drenare (starea λ_k conform notaţiei din Figura 5.36), realizând o descreştere liniară a acesteia, aproximând astfel funcţionarea modelului neuronal de bază numit *leaky integrate-and-fire*, care prevede o descreştere exponenţială.

Implementarea eficientă acestui modul este una din provocările acestei realizări hardware. Însumarea a 32 de variabile reprezentate pe 8 biţi precum şi implementarea celorlalte funcţionalităţi impuse de modelul neuronal aplicat – care presupun multiple comparaţii ale valorilor unor registre şi ajustarea corespunzătoare a valorilor acestora – realizată într-o variantă complet paralelă, presupune un cost foarte ridicat în ceea ce priveşte utilizarea resurselor reconfigurabile disponibile în circuitul FPGA. Dacă am fi propus deci, o implementare complet paralelă a acestei RNP, am fi ajuns la nevoia de a restrânge mult dimensiunile acesteia, pentru a nu depăşi limitele capacităţii platformei hardware utilizate, ceea ce ar fi rezultat o scădere sigură a performanţelor şi a preciziei de clasificare. În decursul studiului teoretic realizat anterior implementării parţial serializate a acestei aplicaţii, am prevăzut utilizarea ca platformă un sistem de dezvoltare FPGA cu un circuit Xilinx Virtex4FX12 FPGA, deoarece acesta conţine un microprocesor încorporat pe 32 de biţi, de tip hard-core (care dispune de arie ne-reconfigurabilă proprie în cadrul chip-ului FPGA), numit PowerPC. Acest PowerPC ar fi putut fi o unitate ce putea procesa cu uşurinţă toate calculele necesare funcţionării tuturor modulelor din proiect. Totuşi, am considerat, că nu este justificat îndeajuns a urma această cale de implementare, deoarece PowerPC este mult prea complex şi nu ar fi fost exploatat într-o măsură care să contrabalanseze faptul că această metodă ar fi indus o serializare mai profundă a execuţiei, deci şi un timp de execuţie mai mare.

Cu toate acestea, am găsit o soluţie mai fezabilă pentru serializarea parţială a RNP, prin introducerea utilizării microcontrolerului soft-core Xilinx PicoBlaze, pentru implementarea modulelor de somă. Acest procesor pe 8 biţi este evident mult mai simplu decât PowerPC, dar are şi o amprentă digitală mult mai compactă, utilizând puţine resurse reconfigurabile, precum a fost prezentat mai devreme în acest capitol, la punctul 5.5.1. Totuşi, pentru a nu pierde în domeniul performanţelor, sistemul dezvoltat trebuie să conţină mai multe astfel de controlere – pentru fiecare somă în parte –, devenind astfel un ***sistem de tip multi-core.*** Pentru a compensa viteza de execuţie mai lentă faţă de componentele RNP implementate în mod paralel, am conectat procesoarele PicoBlaze încorporate la un semnal de tact mai rapid, obţinut utilizând circuitele DCM (Digital Clock Manager) din FPGA.

Implementând prin software rulat (şi scris în limbaj de asamblare specific) de procesoarele PicoBlaze – vezi diagrama de execuţie în Figura 5.38 – funcţionalitatea modulelor somă au fost eliberate importante resurse reconfigurabile, făcând posibilă extinderea RNP pentru a implementa şi alte aplicaţii, cum ar fi testul benchmark de clasificare a setului de date Fischer IRIS, prezentat în subcapitolul 5.5.3. Cu această modificare ar fi posibile şi implementaţii de alte aplicaţii, de exemplu controlul inteligent şi adaptive al unui braţ de robot sau a unui robot mobil.

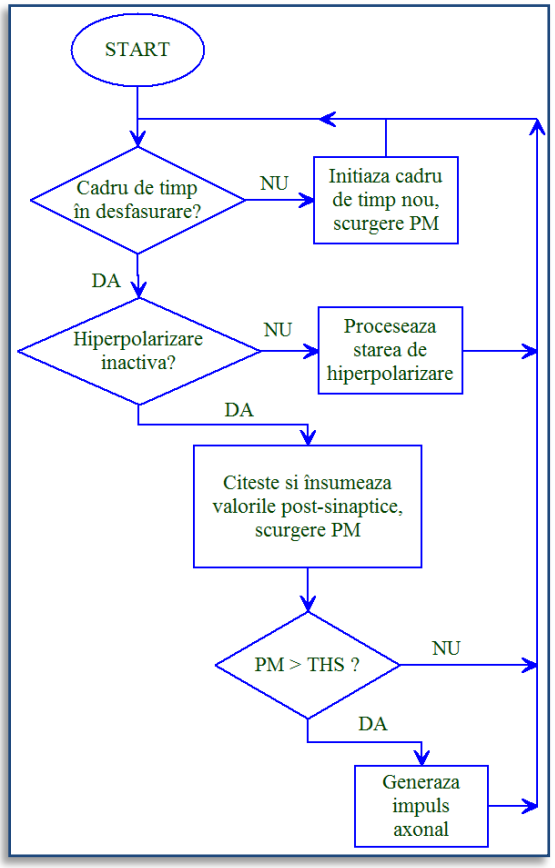


Figura 5.38 Diagrama de execuţie a programului în limbaj de asamblare rulat de procesoarele PicoBlaze care implementează modulele de somă

5.5.2.6. Dinamismul funcţionării modulului somă – Rezultate experimentale

Extrem de important şi crucial de rezolvat în cadrul implementării acestei aplicaţii a fost şi rezolvarea problemei de temporizare şi sincronizare a diferitelor componente ce execută diferiţi algoritmi ai RNP (elemente construite din resurse reconfigurabile dedicate respectiv procesoare încorporate). Pentru a putea soluţiona această dilemă, a fost introdus un al patrulea procesor PicoBlaze (notat cu PB-UMC, fiind componenta centrală a Unităţii de Monitorizare şi Control) care va genera semnalele de tact celorlalte componente ale sistemului şi va sincroniza funcţionarea celor trei module de somă implementate tot prin procesoare PicoBlaze. Pe lângă rularea algoritmului somei, procesoarele PicoBlaze-somă transmit prin porturi configurate în acest sens către PB-UMC parametrii principali ai modelului somei (de ex. potenţialul de membrană). PB-UMC va citi periodic aceste valori, precum şi valorile ponderilor de la cele 96 de sinapse şi le va transmite mai departe printr-un circuit USART ataşat unui calculator personal la care este conectat. Pe acest calculator datele vor fi stocate şi apoi afişate grafic cu ajutorul unui program Matlab.

Pentru a putea trage concluzii asupra funcţionării corecte sau pentru a putea efectua corecţii implementării, este necesară această vizualizare grafică, precum şi o prelucrare numerică prealabilă a fluxului de date ce soseşte de la circuitul neuronal implementat în FPGA spre PC.

Diagramele din Figura 5.39 şi Figura 5.40 prezintă două seturi de măsurători asupra evoluţiei potenţialelor de membrană în două dintre cele trei some implementate în aplicaţia curentă, în decursul unui proces de antrenare. În primul experiment (Figura 5.39), am implementat şi conexiuni laterale inhibitoare de la o somă la celelalte, pentru a preveni activarea simultană a mai multor some pentru aceleaşi valori de intrare, diferenţiind astfel componentele individuale de frecvenţă ce trebuie învăţate. Eliminând aceste conexiuni inhibitoare laterale, somele se vor comporta diferit (Figura 5.40). Este de notat în figurile de mai jos, că activarea unuia dintre neuronii de ieşire rezultă în resetarea PM al celorlalţi neuroni la valoarea de repaus.

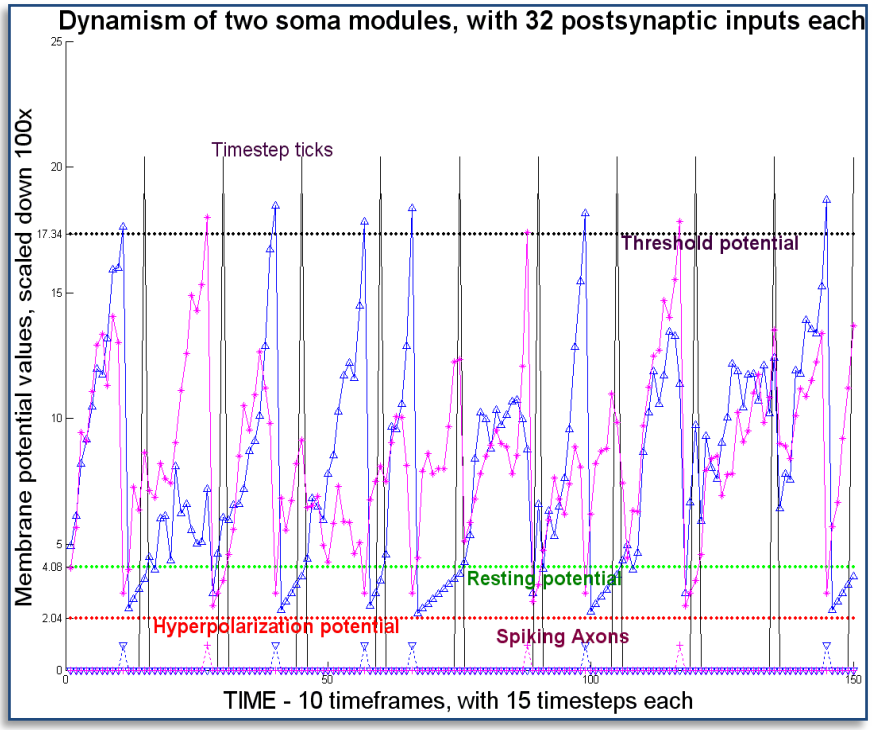


*Figura 5.39 Dinamismul somei **cu inhibare laterală***

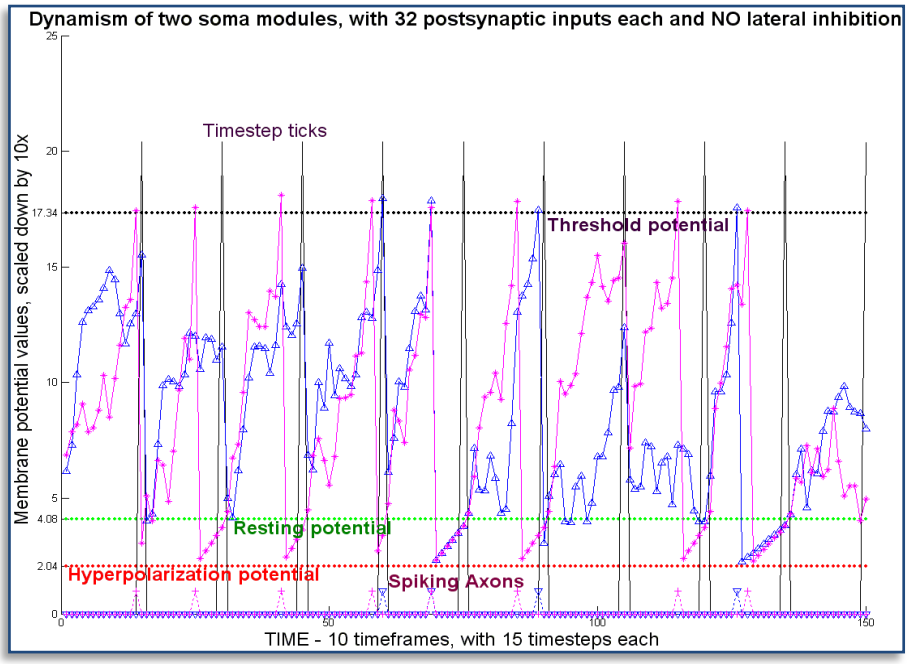


*Figura 5.40 Dinamismul somei **fără inhibare laterală***

După activare, somele vor intra în stare de hiperpolarizare, conform modelului neuronal utilizat, pentru a nu putea emite un nou impuls axonal în acelaşi cadru de timp. Pentru a garanta acest lucru, PM va creşte liniar, cu valori mici, pentru a ajunge doar la valoarea de repaus până la sfârşitul cadrului temporal.

5.5.2.7. Descrierea sistemului test-bench dezvoltat. Prepararea setului de date de antrenare

După cum am afirmat şi la formularea problemei, aplicaţia curentă are ca scop identificarea a până la două component de frecvenţă dintr-un semnal analogic zgomotos. Un astfel de semnal analog, perturbat de zgomot Gaussian este prezentat în Figura 5.43 şi a fost utilizat în experimentele rulate.

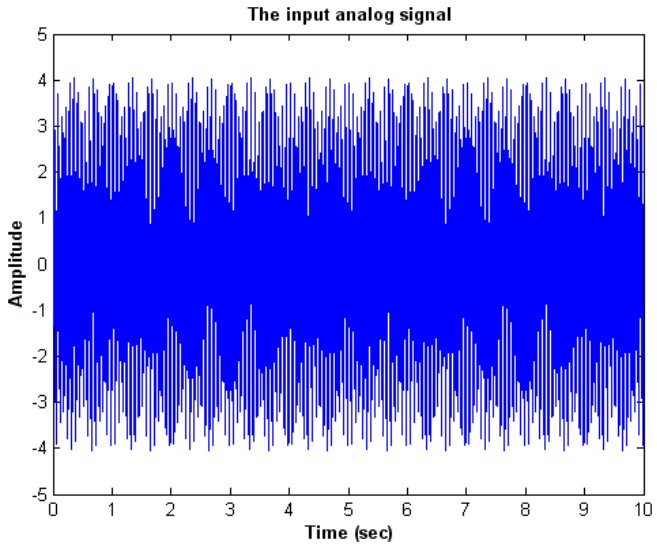


Figura 5.43 Semnalul analog prelucrat

Pentru a putea prezenta la intrările RNP perechi de variabile, reprezentând eşantioane din semnalul vizat (valorile fiind: frecvenţa şi puterea componentei) a fost nevoie de calcule prealabile asupra acestui semnal. Primul pas a fost efectuarea unei transformări Fourier cu algoritmul FFT, al cărei rezultat se poate vedea în Figura 5.42, urmând ca Figura 5.41 să prezinte secţiunea de interes a acesteia. Valorile din Figura 5.41 reprezintă de fapt spaţiul valorilor de intrare

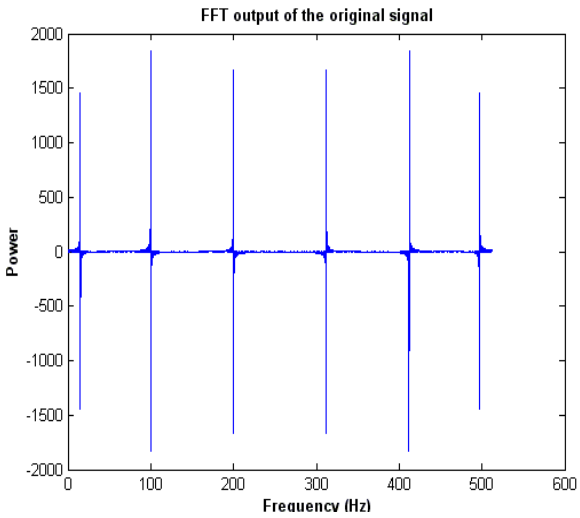


Figura 5.42 Transformata FFT a semnalului analog din Figura 5.43

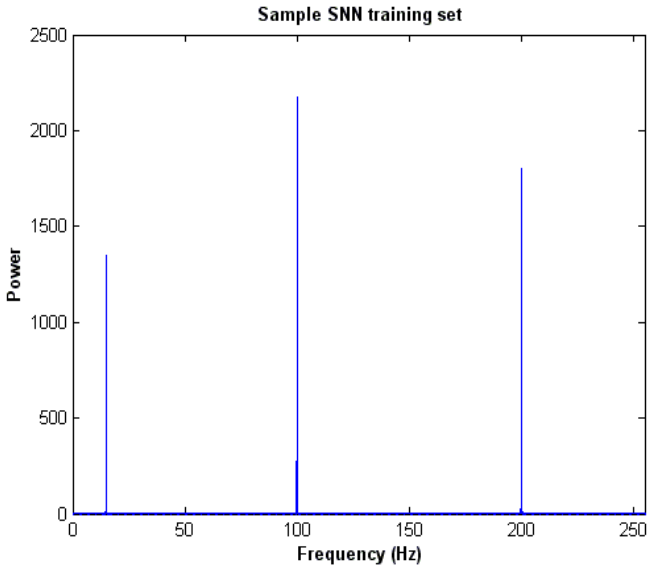


Figura 5.41 Spaţiul valorilor de intrare utilizate pentru antrenarea RNP implementate în FPGA

utilizate pentru antrenarea RNP implementate în FPGA.

Graficul din Figura 5.44 arată valorile de antrenare generate în mod aleator, ele fiind concentrare în vecinătatea punctelor de reper (target focus points) care sunt de fapt valorile prescrise în timpul învăţării neuronilor de ieşire ai RNP.

Pentru fiecare punct din acest spaţiu, valorile coordinatelor reprezintă variabilele de intrare ale RNP, acompaniate de valoarea prescrisă fiecărui neuron de ieşire pentru acel punct, rezultând un algoritm de antrenare

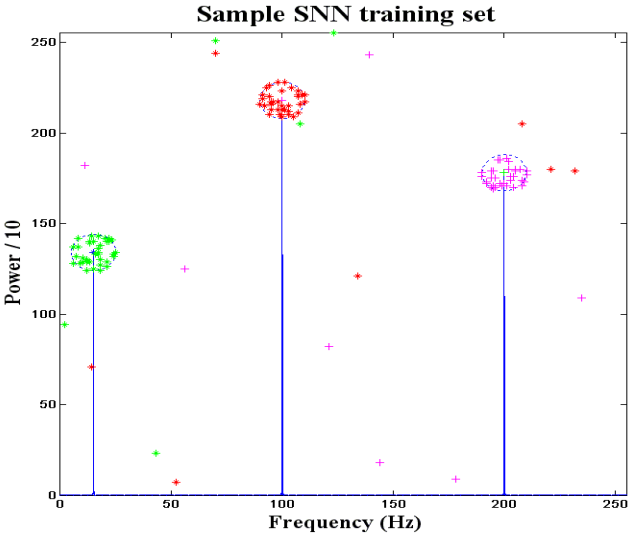


Figura 5.44 Valori de antrenare concentrare în jurul punctelor de focus prescrise, şi altele aleator alese ca zgomot

supervizat. În Figura 5.41 se mai pot observa şi puncte ce apar în poziţii aleatoare în afara vecinătăţilor punctelor de reper. Acestea au fost introduse pentru a evita blocajul algoritmului în minime locale şi a garanta convergenţa acestuia. Toate valorile setului de antrenare sunt prezentate sistemului neuronal în ordine aleatoare.

Reţeaua neuronală pulsativă hardware a învăţat separarea acestor component de frecvenţă, activând după antrenare acelaşi neuron de ieşire pentru punctele aparţinând aceleiaşi componente. În mod evident, acest lucru se întâmplă cu o anumită histereză, definind o vecinătate detectată a componentelor pentru care s-a efectuat antrenarea.

5.5.2.8. Testarea sistemului neuronal implementat

După cum s-a menţionat şi la subpunctul anterior, întreaga RNP hardware este controlată de cel de-al patrulea procesor PicoBlaze (PM-UMC), componenta principală a Unităţii de Monitorizare şi Control din Figura 5.45 de mai jos. Acest modul controlează funcţionarea pas cu pas a sistemului neuronal implementat, aşadar şi a procesului de antrenare.

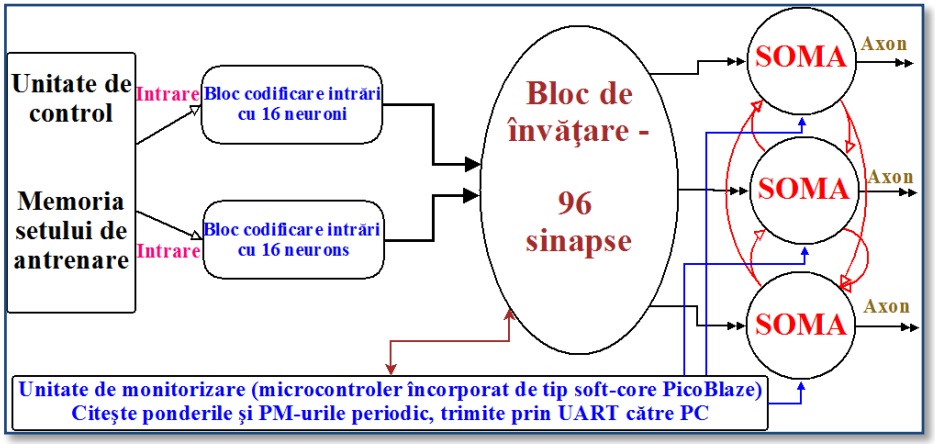


Figura 5.45 Diagrama bloc a sistemului testbench realizat

Diagrama de execuţie a algoritmului rulat de PB-UMC, program scris în limbaj de asamblare specific Xilinx PicoBlaze, se prezintă în Figura 5.46 iar fazele şi paşii de operare a RNP sunt prezentate pe scurt în Tabelul 13.

Tabelul 13 Paşii de procesare a calculelor RNP

Cicluri de tact/ faze de execuţie	a		b	c	d	e
Nr. ciclu de tact	Control Antrenare	Intrări valide	Tact Bloc Intrări	Tact Sinapse	Tact Some (PBlaze)	Transmisie UART
1	front crescător	NU	fără tact	fără tact	HOLD	
2	front crescător	NU	fără tact	fără tact	HOLD	
3	front crescător	DA	fără tact	fără tact	HOLD	
4	front crescător	DA	fără tact	fără tact	HOLD	valori de intrare
5	fără tact	DA	front crescător	fără tact	HOLD	
6	fără tact	DA	front crescător	front crescător	HOLD	ponderi
7	fără tact	DA	fără tact	fără tact	RULEAZĂ	potenţiale de membrană
if TS_nr<16 goto 5				if TS_nr=15 then →		ponderi, valori axonale
if Epoch<129 goto 1						

În primul ciclu de antrenare (epoch) se prezintă pe rând reţelei neuronale fiecare element al setului de antrenare, generând numărul necesar de fronturi ale semnalului de tact către modulul de generare a intrărilor, pentru a permite acestuia să citească aceste valori din memoria BRAM proprie în care sunt stocate. Apoi, Unitatea de Control va livra primul front de tact modulului Blocului Intrărilor, dintre cele 16 ale unui cadru temporal (TS în Tabelul 13) care va începe astfel codificarea valorilor de intrare în impulsuri decalate temporal. La ciclul de tact numărul 7, somele au pe intrările lor primul set

de date post-sinaptice, deci pot fi activate, pentru a rula un ciclu al algoritmului implementat de modulele PicoBlaze.

La momentele corespunzătoare, când valorile respective sunt valide, UMC colecta datele de la componentele sistemului şi va trimite pe portul serial cei mai importanţi parametrii (ponderi, potenţiale de membrană, valori axonale) ai RNP spre calculator pentru procesare şi vizualizare grafică (Figura 5.39, Figura 5.40). Implementarea acestei reţele neuronale pulsative utilizând procesoare încorporate, s-a dovedit a fi mai eficientă în ceea ce priveşte necesarul de resurse hardware, decât alte aplicaţii, implementând RNP cu codarea ratei impulsurilor, realizate complet paralel. Acest lucru a justificat continuarea cercetărilor în această direcţie – cu procesare parţial serializată şi stocarea variabilelor de stare în memorii BRAM – metodă prezentă şi în literatura de specialitate (Maass, Natschlager, & Markram, 2002) (Schrauwen & Van Campenhout, 2006) (Bakó & Brassai, 2006) (Floreano, Dürr, & Mattiussi, 2008).

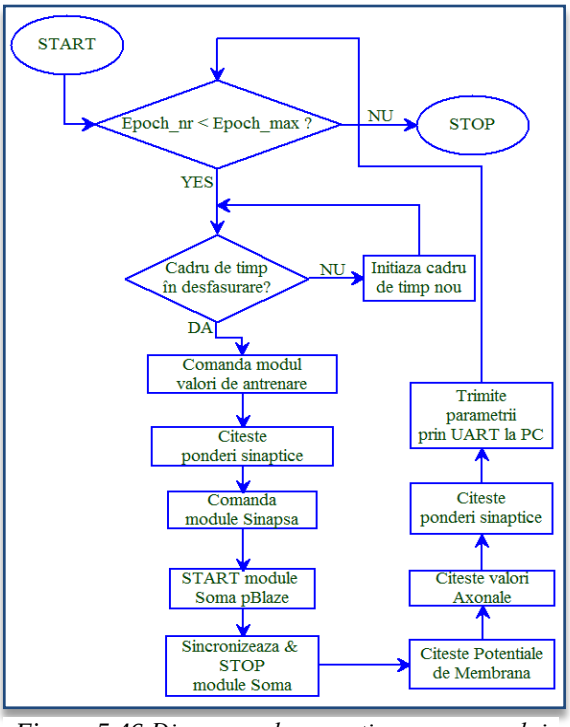


Figura 5.46 Diagrama de execuţie a programului în limbaj de asamblare rulat de procesorul PicoBlaze care implementează Unitatea de Monitorizare şi Control

Tabelul 14 trece în revistă utilizarea de resurse reconfigurabile FPGA în proiectul implementat, detaliat pentru componentele principale ale acestuia. Se poate nota, că pentru a optimiza utilizarea resurselor de tip BRAM s-a folosit o component de memorie cu porturi duale, care stochează programul rulat de către procesoarele PicoBlaze ce implementează somele, ea fiind accesată de două astfel de modul simultan, ele rulând acelaşi program dar cu date de intrare diferite. Deoarece şi această aplicaţie este de fapt una de clasificare, se poate compara cu aplicaţii de test clasice (Fischer's Iris, Wisconsin Breast Cancer). Implementarea unui astfel de test se va prezenta în subcapitolul următor, numărul 5.5.3.

Tabelul 14 Utilizarea de resurse reconfigurabile în proiectul implementat

Nume modul	Slices	Slice Reg.	LUTs	LUT RAM	BRAM
[-] Spiking NN project	9408	6886	12755	0/281	5/28
Control & Monitoring Unit	54/54	41/41	34/34	0/0	0/1
[-] Inst. Spiking ANN	602/ 8592	43/ 6845	1106/ 12323	0/281	0/5
PicoBlaze shared mem. bl.	78	88	49	0	0
Input block	761	693	1277	0	2
Master PicoBlaze	126	69	199	68	1
Synapse Unit (one of 96)	67/ 6432	59/ 5664	95/ 9120	0/0	0/0
SOMA 1 (PicoBlaze uC)	99	67	169	68	2
SOMA 2 (PicoBlaze uC)	97	67	169	68	
SOMA 3 (PicoBlaze uC)	97	67	169	68	

Reţeaua neuronală pulsativă implementată hardware arată performanţe similare sau mai bune, decât o reţea asemănătoare bazată pe modelul MLP (Multi-Layer-Perceptron), implementat în Matlab, antrenat conform regulilor Levenberg-Marquardt (LM) într-un număr de paşi egali. În mod explicit, în timp ce algoritmul LM a avut nevoie de ~1500 de cicluri de antrenare pentru a ajunge la o clasificare corectă de ~97% a componentelor de frecvenţă a semnalului din Figura 5.43 RNP implementată hardware a avut nevoie de numai ~250 de cicluri de antrenare pentru a atinge aceeaşi acurateţe de clasificare.

O altă caracteristică valoroasă de notat a RNP implementate în FPGA este faptul că funcţionează în timp real, cu toate că procesarea unor subansamble este serializată parţial prin utilizarea procesoarelor soft-core încorporate. Aceste procesoare PicoBlaze sunt animate de un semnal de tact de 50MHz şi execută o instrucţiune în două cicluri de tact. Algoritmul somelor este implementat printr-un program de asamblare de aproximativ 200 de instrucţiuni iar celelalte componente ale RNP funcţionează în paralel. Efectuând un calcul simplu, obţinem un timp de calcul/ciclu de antrenare de 64µs, şi un timp de învăţare de 16ms. După terminarea fazei de învăţare, circuitul neuronal va putea detecta o component de frecvenţă într-un timp mai mic de 1µs.

În concluzie, se poate afirma, că această abordare a implementărilor reţelelor neuronale neuromorfe are un potenţial foarte promiţător în domeniul aplicaţiilor în timp real.

5.5.3. Implementare test benchmark: Clasificarea setului de date Fisher IRIS

Scopul acestei aplicaţii este de a testa acest mod de a implementa Reţele Neuronale Pulsative, şi a evalua potenţialul pentru aplicaţii similare şi puterea de calcul a acestora. Testul reper (benchmark) de clasificare a setului de date Fischer IRIS este probabil cel mai cunoscut în domeniu, fiind astfel o metodă bună de verificare.

5.5.3.1. Descrierea setului de date. Formularea problemei de clasificare

Articolul lui Fisher (Fisher, 1936) este unul de referinţă în domeniu, şi este citat în mod frecvent chiar şi în prezent. Acest set de date conţine 3 clase a câte 50 de elemente fiecare, unde fiecare clasă se referă la un tip de plantă Iris. Una din clase poate fi separată liniar de celelalte două; celelalte două clase NU pot fi separate liniar între ele. Lungimea şi lăţimea sepalelor şi petalelor pentru cele trei specii nord-americane de iris. Datele referitoare la *iris setosa canadensis* şi *iris versicolor* au fost folosite de către R.A. Fisher pentru a ilustra analiza discriminării liniare. Datele referitoare la *iris virginica* au fost adăugate ca o extensie. Lungimea şi lăţimea sepalelor şi petalelor sunt doar coordonate convenabile rezultate datorită faptului ca sunt uşor de măsurat.

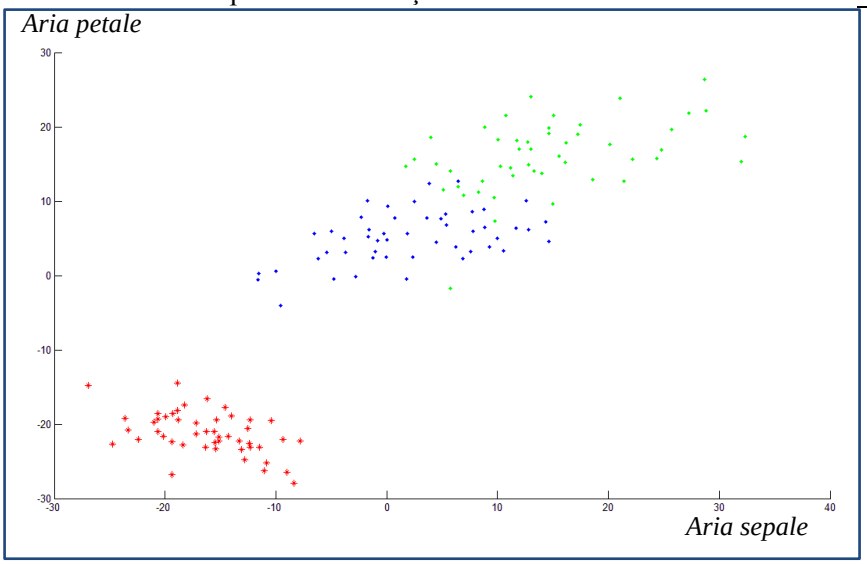


Figura 5.47 Reprezentarea valorilor bazei de date Fisher IRIS cu ajutorul proiecţiei Sammon (vezi subcap.5.6.3.7) 4D→2D.

Valorile afişate pe coordinate sunt de fapt valori calculate de acest algoritm de reprezentare, dar se pot considera ca valorile ariilor celor două tipuri de petale ale florilor IRIS.

Roşu => IRIS-setosa
Albastru=>IRIS-versicolor
Verde => IRIS-virginica

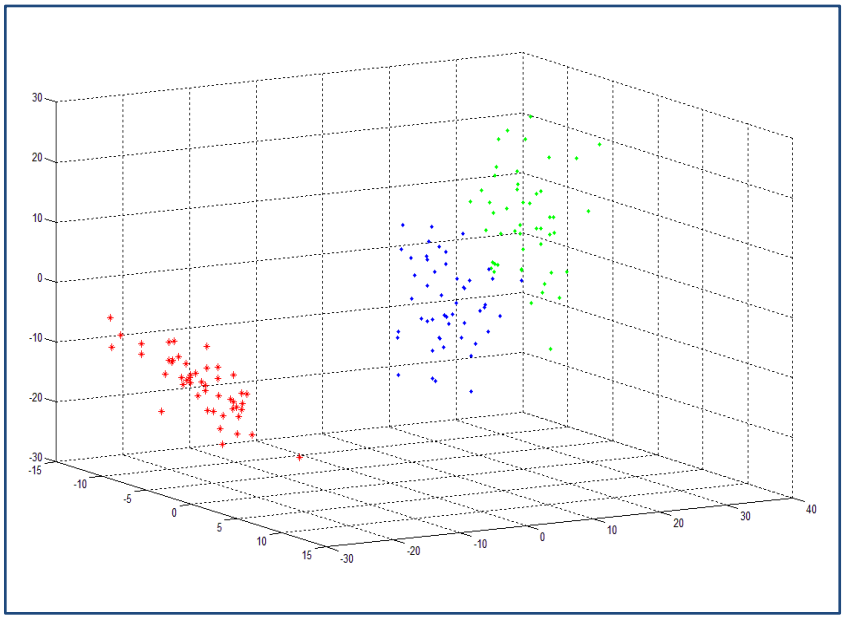


Figura 5.48 Reprezentarea valorilor bazei de date Fisher IRIS cu ajutorul proiecţiei Sammon (vezi subcap.5.6.3.7) 4D→3D.

Roşu=>IRIS-setosa
Albastru=>IRIS-versicolor
Verde=>IRIS-virginica

Folosind lungimea sepalei * lăţimea sepalei şi lungimea petalei * lăţimea petalei ca o aproximare neprelucrată a suprafeţei acoperite de sepale şi petale, obţinem o pereche de coordonate are ar putea fi mai relevante pentru botanică. Folosind aceste coordonate, toate cele trei specii din acest set de date pot fio separate, având doar trei clasificări greşite. *Iris setosa* şi *iris versicolor* sunt separate perfect. Numărul de atribute: 4 atribute numerice, numerotare şi clasa. Distribuţia claselor: 33.3% pentru fiecare din cele 3 clase.

5.5.3.2. Reţeaua neuronală pulsativă implementată pe FPGA

S-a implementat o RNP cu patru intrări şi trei ieşiri, după cum arată şi schema bloc din Figura 5.49. Această reţea trebuie să clasifice corect elementele bazei de date Fisher IRIS, care sunt valori cu o zecimală cuprinse între 0 şi 8, dar au fost scalate între 0-80 pentru a putea fi reprezentate ca valori întregi în hardware-ul digital. Antrenarea reţelei s-a efectuat în aşa fel, încât fiecare neuron ieşire să se activeze atunci, când valorile de intrare identifică un element care aparţine clasei asociate neuronului respectiv. Arhitectura RNP este foarte similară cu ce a aplicaţiei anterioare, dar având un număr diferit de neuroni de intrare pentru codificarea celor patru intrări, neuronii de ieşire fiind implementaţi şi în acest caz cu trei procesoare încorporate de tip soft-core Xilinx PicoBlaze.

Parametrii reţelei neuronale implementate:

- *Numărul intrărilor:* $n = 4$
- *Nr. neuronilor de intrare/intrare:* $n_{intr}=13$
- *Nr. neuronilor de intrare activi/intrare:* $n_{act} = 4$
- *Numărul paşilor dintr-un cadru temporal* $T=16$

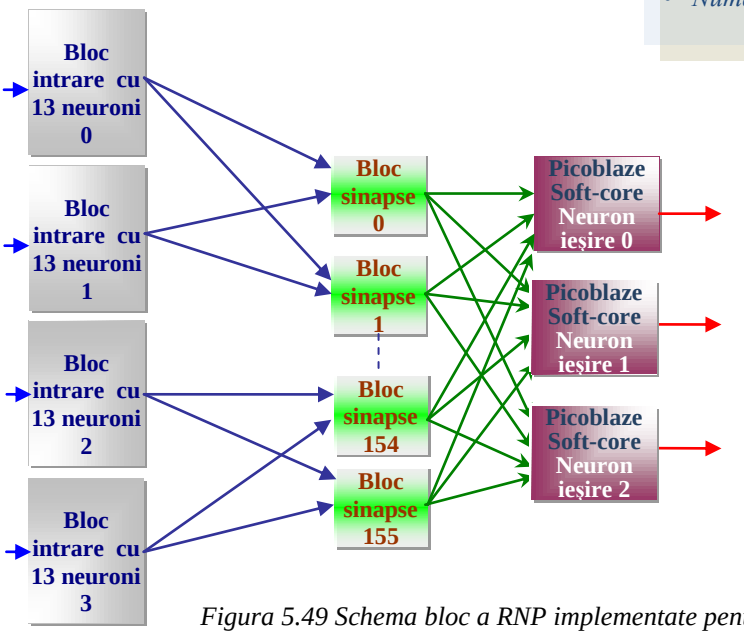


Figura 5.49 Schema bloc a RNP implementate pentru clasificarea setului de date Fisher IRIS

5.5.3.3. Implementarea neuronilor de intrare – codificarea variabilelor de intrare în impulsuri decalate temporal

Deoarece plaja valorilor din baza de date vizată (IRIS) defineşte un spaţiu al parametrilor de intrare de dimensiuni mai reduse decât la aplicaţia de la punctul 5.5.2, s-a scăzut numărul ($n=13$) neuronilor de intrare, dar s-a menţinut constantă ($n_{act}=4$) numărul celor activate într-un cadru de timp. Acoperirea acestui spaţiu cu funcţii de activare triunghiulare este

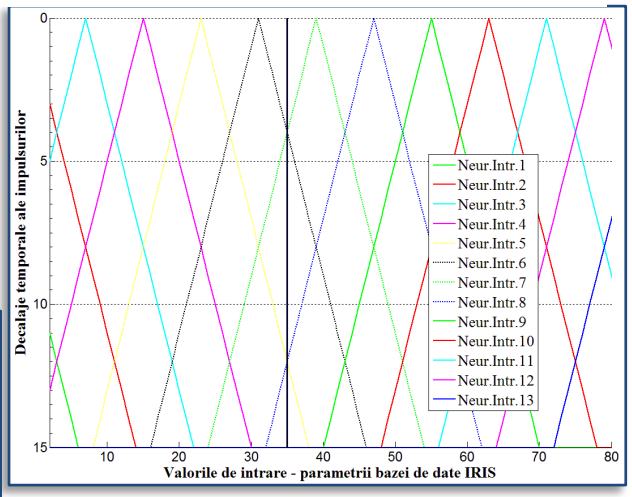


Figura 5.50 Funcţiile triunghiulare asociate neuronilor de intrare şi decalajele temporale

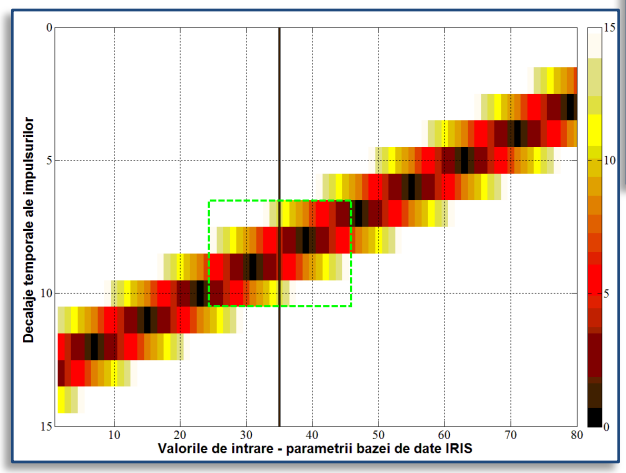


Figura 5.51 Domeniile receptive ale neuronilor de intrare

reprezentată în Figura 5.50 iar domeniile receptive corespunzătoare acestora în Figura 5.51. Se poate observa în ambele figuri, că valoarea de intrare 35 luată ca exemplu va activa patru neuroni de intrare.

După cum a fost prezentat şi la subpunctul 3.1.3.1 elementele BlockRAM ale circuitelor FPGA pot fi configurate în mai multe feluri, variind numărul biţilor de pe magistrala de adrese respectiv de pe cea de date. Pentru a stoca valorile de decalaj temporal am ales componenta cu numele RAMB16_S36_S36, care este un modul BRAM cu port dual şi o capacitate de 512x32 biţi. Arhitectura cu port dual permite ca două locaţii de memorie să fie accesate simultan. Deoarece valorile stocate sunt iniţializate odată cu programarea circuitului FPGA, acest modul va fi utilizat ca şi cum ar fi de fapt o memorie ROM, efectuând numai operaţii de citire, evitând astfel eventualele conflicte de adresare la scriere. Conectând câte două astfel de module în paralel (4 în total), devine posibil să se obţină în aceeaşi perioadă de tact cei 4*4*13=208 biţi reprezentând decalajele temporale pentru cele două intrări ale RNP, utilizând numai o parte a capacităţii unei locaţii astfel adresabile de 4*2*32=256 biţi.

5.5.3.4. Modulul de implementare a sinapselor

Modulul ce implementează sinapsa unui neuron este responsabilă pentru amplificarea sau atenuarea impulsurilor pre-sinaptice emise de către neuronul de intrare la care este conectat. Astfel se generează o valoare post-sinaptică ponderată ce va fi transmisă spre soma neuronului de ieşire la care aparţine sinapsa respectivă.

Acest modul este de asemenea cel, care implementează regulă nouă de învăţare supervizată care este o variantă adaptată a metodei Hebb, bazată pe reguli de decalaje temporale, după cum a fost prezentat în secţiunea 5.5.2.4. Sinapsele vor testa amprenta temporală (numărul pasului temporal, notat cu *TS* în Tabelul 12) a fiecărui impuls pre-sinaptic şi vor ajusta valorile ponderilor stocate în concordanţă cu regulile de învăţare prezentate în Tabelul 12, luând în calcul şi valorile prescrise ale stării neuronilor de ieşire (valoare axonală) pentru valori variabilelor de intrare actuale.

5.5.3.5. Dinamismul funcţionării modulului somă – Rezultate experimentale

Utilizarea PicoBlaze pentru implementarea modulelor somă a fost cea mai utilă alegere şi în acest caz. Aceste procesoare rulează acelaşi algoritm ca şi în cazul precedent (Figura 5.38), dar cu parametrii somei adaptate la arhitectura schimbată a RNP. Astfel, numărul de 13 a neuronilor intrare într-un bloc aparţinând unei dintre intrări rezultă valori pentru maximul PM, potenţialul de prag, valoarea de hiperpolarizare respectiv valoarea de scurgere.

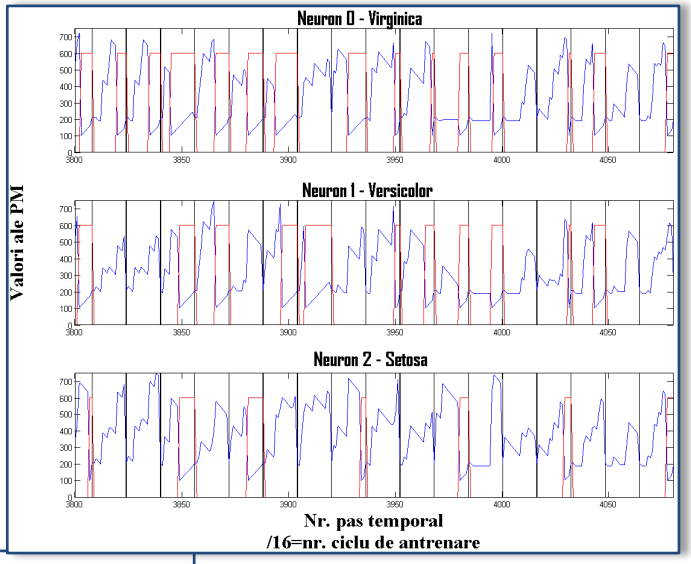


Figura 5.52 Măsurători în timpul antrenării RNP (albastru-variaţia PM celor trei some, roşu valoarea axonală a acestora)

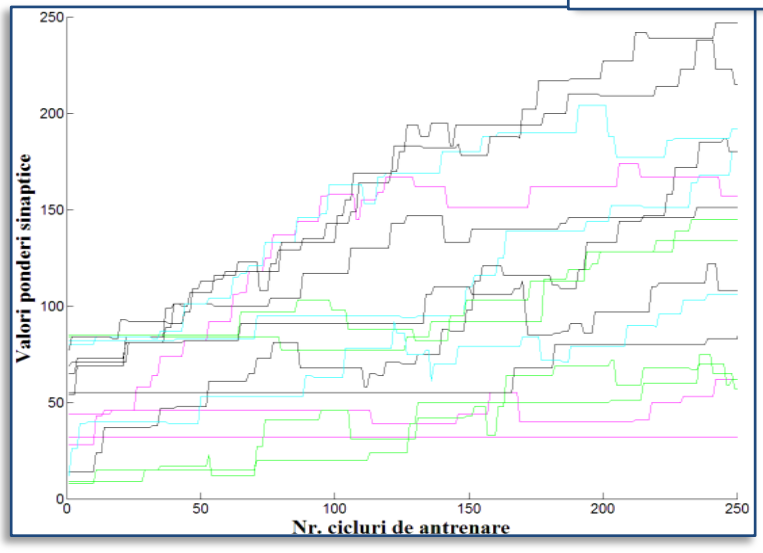


Figura 5.53 Variaţia ponderilor în timpul antrenării (primele 15)

Se poate nota în Figura 5.53 – în care s-a reprezentat numai un număr redus dintre cele 156 de ponderi sinaptice ale reţelei neuronale pulsative hardware – că unele valori rămân neschimbate. Acest fenomen apare datorită faptului, că procesare sinapselor depinde de prezenţa impulsurilor incidente, iar dacă neuronul de intrare la care este

conectată sinapsa respectivă nu s-a activat pentru valoarea de intrare actuală, sinapsa va fi inactivă.

5.5.3.6. Descrierea sistemului testbench dezvoltat. Prepararea setului de date de antrenare

După cum am afirmat şi la formularea problemei, aplicaţia curentă are ca scop identificarea clasei de floare IRIS pe baza a patru proprietăţi prezentate ca intrări ale reţelei neuronale hardware.

S-a menţionat şi la descrierea aplicaţiei anteriore, că întreaga RNP hardware este controlată de cel de-al patrulea procesor PicoBlaze (PM-UMC) încorporat, componenta principală a Unităţii de Monitorizare şi Control din Figura 5.54 de mai jos. Acest modul controlează funcţionarea pas cu pas a sistemului neuronal implementat, aşadar şi a procesului de antrenare.

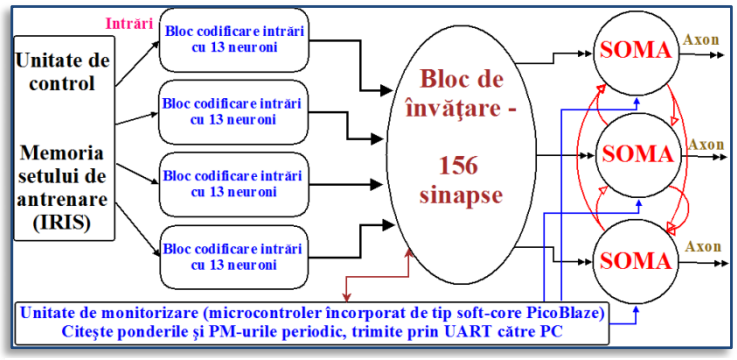


Figura 5.54 Diagrama bloc a sistemului testbench realizat

Tabelul 15 Utilizarea resurselor FPGA a proiectului FPGA IRIS dataset

Device Utilization Summary (XC3S1200E)			
Logic Utilization	**Used**	**Avail.**	**Utiliz.**
Nr. of Slices	6941	8672	80%
Nr. of Slice Flip Flops	5785	17344	33%
Nr. of 4 input LUTs	12806	17344	73%
Nr. of BRAMs	7	28	25%
Nr. of GCLKs	4	24	16%

Din setul de date Fisher IRIS a fost separat o mulţime de antrenare, formată din 60% din toate cele trei clase, alese în mod aleatoriu. Aceste valori de antrenare se prezintă pe rând reţelei neuronale, generând numărul necesar de fronturi ale semnalului de tact către modulul de generare a intrărilor, pentru a permite acestuia să citească aceste valori din memoria BRAM proprie în care sunt stocate. Apoi, Unitatea de Control va livra primul fronturi de tact modulului Blocului Intrărilor, care va începe astfel codificarea valorilor de intrare în impulsuri decalate temporal. La momentele corespunzătoare, când valorile respective sunt valide, UMC colecta datele de la componentele sistemului şi va trimite pe portul serial cei mai importanţi parametrii (ponderi, potenţiale de membrană, valori axonale) ai RNP spre calculator pentru procesare şi vizualizare grafică (Figura 5.52 şi Figura 5.53).

5.5.4. Sistemul de dezvoltare utilizat pentru realizarea proiectelor cu procesoare încorporate

Digilent Nexys2 este o platformă de dezvoltare completă, gata de utilizare, bazată pe un circuit FPGA de tip Xilinx Spartan 3E. Portul USB2 de mare viteză integrat, cei 16Mbytes de memorie RAM şi ROM, şi mai multe dispozitive şi porturi de I/O, fac din Nexys2 o platformă ideală pentru sisteme digitale de toate tipurile, inclusiv sisteme cu procesor integrat bazate pe MicroBlaze de la Xilinx. Portul USB2 asigură plăcii alimentarea şi o interfaţă de programare, astfel încât placa Nexys2 poate fi folosită împreună cu un computer de tip notebook pentru a crea o staţie de proiectare cu adevărat portabilă. Nexys2 aduce tehnologii de vârf pe o platformă care poate fi folosită de oricine pentru a câştiga experienţă în proiectarea digitală. Poate găzdui nenumărate sisteme digitale bazate pe FPGA, şi sistemele proiectate pot uşor creşte dincolo de dimensiunile plăcii prin folosirea oricăruia sau a tuturor celor 5 conectoare de expansiune. Patru conectoare de module periferice cu 4 pini pot găzdui până la opt module periferice cu cost redus pentru a adăuga extensii ca control motor, conversie A/D sau D/A, circuite audio, şi o gamă de interfeţe de senzor şi elemente de execuţie. Toate semnalele accesibile utilizatorului pe placa Nexys2

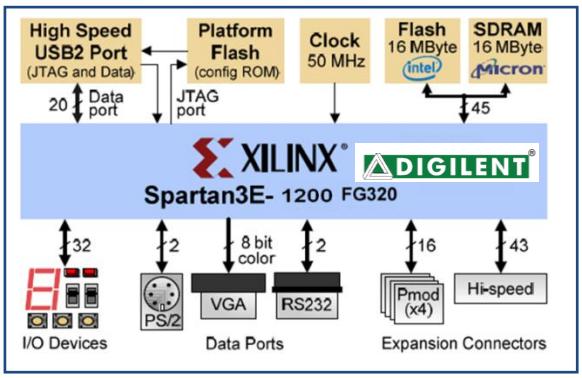


Figura 5.55 Sistemul de dezvoltare Digilent Nexys2 (XC3S1200E)

sunt protejate împotriva perturbărilor electrostatice şi a scurtcircuitelor, asigurând un timp îndelungat de funcţionare în orice mediu. Elementele principale ale Nexys2 sunt următoarele:

- circuit FPGA de tip Xilinx Spartan 3E cu 500Kgate sau 1200Kgate
- 16MB de PSDRAM Micron şi 16MB ROM Intel StrataFLash
- Xilinx Platform Flash pentru configuraţii FPGA non-volatile
- oscilator de 50MHz plus socket pentru încă un oscilator60 I/O-uri FPGA directate la conectori de expansiune,
- 8 LED-uri, afişaj de 7 segmente şi 4 caractere, 4 butoane, 8

5.6. Realizări de RNP cu microcontroler pe 32 de biţi, încorporat în circuitele FPGA

În acest subcapitol se vor prezenta două aplicaţii de clasificare care necesită o putere de calcul mai ridicată, ele fiind recunoaştere de caractere respectiv realizarea testului benchmark de clasificare a setului de date Wisconsis breast cancer.

5.6.1.1. Microcontrolerul Xilinx MicroBlaze

Nucleul încorporat de tip soft-core MicroBlaze este un procesor cu un set redus de instrucţiuni, optimizat pentru implementările bazate pe circuite FPGA realizate de Xilinx. Figura 5.56 arată o diagramă bloc a nucleului MicroBlaze. Nucleul încorporat de tip soft-core MicroBlaze are următoarele proprietăţi:

- 32 de regiştrii pe 32 biţi cu destinaţie generală
- Instrucţiuni pe 32 biţi cu trei operanzi şi două moduri de adresare
- Magistrale separate pe 32 biţi pentru instrucţiuni şi pentru date conform specificaţiei OPB (On-Chip Peripheral Bus) de la IBM
- Magistrale separate pe 32 biţi pentru instrucţiuni şi pentru date cu conexiune directă către blocul de memorie RAM încorporat pe circuitul reprogramabil accesat prin LMB (Local Memory Bus)
- Magistrală de adresare pe 32 biţi
- Cache pentru instrucţiuni şi pentru date
- Elemente logice pentru depanare hardware
- Suportă FSL (Fast Simplex Link)
- Multiplicator hardware (în dispozitivele din familia Virtex-II).

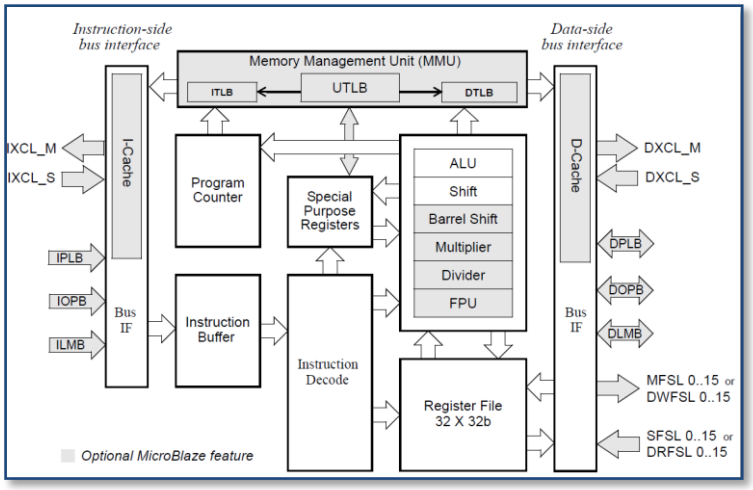


Figura 5.56 Structura logică a procesorului Xilinx MicroBlaze

Nucleul MicroBlaze implementează o arhitectură de tip Harvard. Asta înseamnă că are interfeţe de magistrală separate pentru acces de date şi de instrucţiuni. Fiecare unitate de interfaţă magistrală se împarte la rândule ei în Local Memory Bus (LMB) şi On-Chip Peripheral Bus (OPB) de la IBM. LMB asigură acces într-un singur ciclu de tact la blocul de memorie RAM pe două porturi încorporat în circuitul reconfigurabil. Interfaţa OPB asigură conexiunea către periferice şi memoria încorporată sau alte periferice din afara circuitului reconfigurabil. Nucleul MicroBlaze dispune de asemenea de 8 interfeţe de intrare şi 8 interfeţe de ieşire la magistralele de tip Fast Simplex Link (FSL). Magistralele FSB sunt canale de comunicaţie dedicate unidirecţionale fără arbitrare. MicroBlaze utilizează adresare pe 32 de biţi, harta memoriei adresabile fiind prezentată în Figura 5.57.

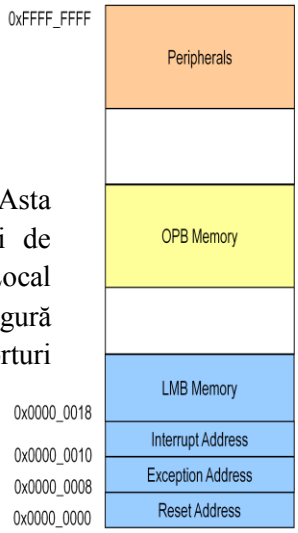


Figura 5.57 Harta memoriei procesorului MicroBlaze

5.6.2. Implementare aplicativă: Recunoaştere de caractere cu o reţea neuronală neuromorfă multistrat

5.6.2.1. Formularea problemei de recunoaştere de caractere

Scopul acestei aplicaţii este de a realiza un sistem neuronal bazat pe o RNP implementată într-un mediu mixt hardware-software pentru a evalua performanţele unui astfel de sistem.

În acest sens am construit o RNP cu două straturi care va fi descrisă în cele ce urmează. Caracterele ce vor fi învăţate de RNP s-au considerat a fi reprezentate într-o matrice de 6x8, asemănător cu modul de afişare pe un afişaj alfanumeric (Figura 5.58). Fiind vorba de o reţea neuronală bazată pe impulsuri, valorile de 0 şi 1 logic, care reprezintă punctele inactive respectiv active pe afişajul LCD se vor transforma în impulsuri ce sosesc sau nu pe una dintre cele 6x8=48 intrări ale acesteia.

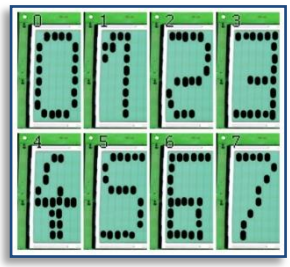


Figura 5.58 Forma caracterelor învăţate

Valorile prescrise pentru ieşirile reţelei neuronale depind de caracterele prezentate la intrări într-un anumit ciclu de învăţare.

5.6.2.2. Dezvoltarea componentelor hardware ale sistemului încorporat

În cadrul acestei aplicaţii am utilizat numai module hardware predefinite ale mediului de dezvoltare Xilinx Platform Studio. Elementul principal al implementării este procesorul Xilinx MicroBlaze, la care am conectat mai multe module de memorie respectiv dispozitive periferice prin magistrala de 32 de biţi Processor Local Bus (PLB).

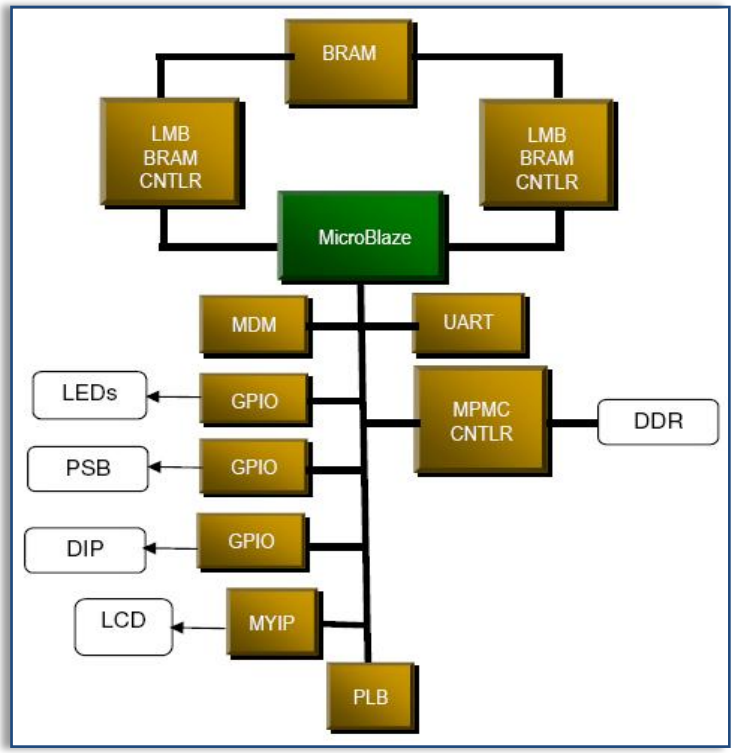


Figura 5.59 Diagrama bloc a sistemului încorporat care implementează RNP pentru recunoaşterea de caractere

În decursul implementării, s-a constatat, că pentru a acomoda programul scris în limbaj C, rulat pe acest procesor, care implementează de fapt această reţea neuronală pulsativă, capacitatea memoriilor interne de tip BRAM ale circuitului FPGA care sunt folosite ca memorie de instrucţiuni şi de date (conectate la magistrala Local Memory Bus - LMB) ale MicroBlaze nu a fost suficientă. Pentru a soluţiona această problemă, am conectat la proiectul încorporat un controler de memorie dinamică (MPMC CNTLR în Figura 5.59) cu scopul de a extinde capacitatea memoriei interne cu modulul DDR ce se găseşte pe placa de dezvoltare utilizată.

Dispozitivul standard de intrare/ieşire a sistemului dezvoltat a fost considerat a fi portul serial RS232, ataşat proiectului încorporat din biblioteca de dispozitive predefinite a mediului EDK (UART în Figura 5.59). Controlul şi setările funcţionării sistemului implementat au fost realizate prin butoanele şi comutatoarele plăcii de dezvoltare, conectate la MicroBlaze prin modulul standard de accesare a perifericelor generice (General Purpose I/O – GPIO în Figura 5.59). Un alt periferic – implementare proprie – este cel care comandă afişajul LCD al plăcii de dezvoltare (MYIP în Figura 5.59) care este utilizat pentru afişarea stărilor RNP din FPGA, respectiv pentru urmărirea procesului de învăţare. Acest din urmă modul a fost implementat în limbaj VHDL.

5.6.2.3. Structura reţelei neuronale pulsative dezvoltate

Reţeaua neuronală dezvoltată are o structură cu trei starturi. Primul strat este cel care primeşte impulsurile de intrare reprezentând caracterele de învăţat. Cel de-al doilea strat este cel care va fi antrenat să înveţe anumite caracteristici ale acestor caractere, iar stratul de ieşire va conţine neuroni pulsativi care se vor antrena să se activeze la apariţia unor caractere cu un set de caracteristici bine definite. Astfel, şase dintre cei paisprezece neuroni pulsativi ai stratului ascuns (Figura 5.60) vor fi antrenaţi cu o regulă care le va acorda să înveţe dacă majoritatea punctelor dintr-o coloană a matricii de intrare sunt active. Aceşti neuroni au câte opt intrări conform numărului de puncte dintr-o coloană a intrării. Restul de opt neuroni pulsativi din stratul ascuns vor fin antrenaţi să răspundă cu impuls axonal atunci, când majoritatea celor şase intrări al fiecăruia (conform numărului de puncte dintr-o linie a matricii de intrare) sunt active.

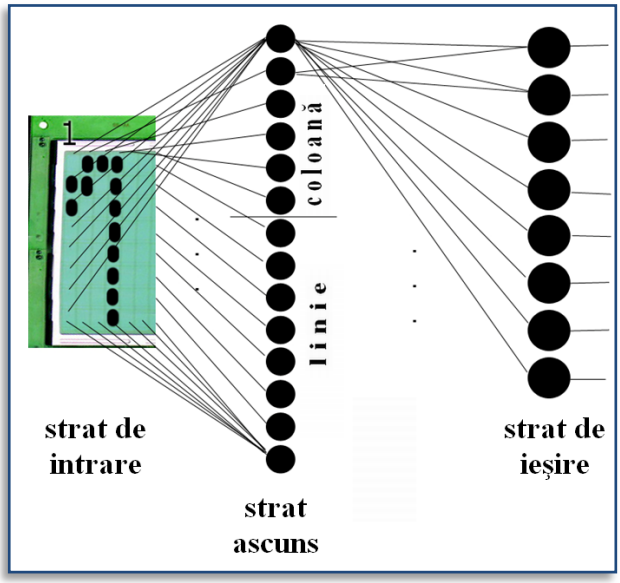


Figura 5.60 Structura RNP pentru recunoaştere de caractere, implementată cu procesor MicroBlaze încorporat în circuit FPGA

Aceste proprietăţi sunt apoi propagate spre neuronii din stratul de ieşire, în număr egal cu cel al caracterelor care vor fi învăţate de RNP, în aplicaţia de faţă aceasta fiind opt. Stratul de ieşire este complet conectat cu stratul ascuns, iar neuronii pulsativ din acesta se vor activa atunci când combinaţia caracteristicilor raportate de activarea celor paisprezece neuroni din stratul ascuns va corespunde cu caracteristicile prescrise pentru caracterul prezentat la intrarea reţelei neuronale. Aşadar, această RNP se poate considera a fi implementată cu codarea ratei impulsurilor deoarece nu sunt considerate decalajele temporale dintre acestea ci doar corelaţia dintre momentele de apariţie pe cele 48 de intrări.

5.6.2.4. Modelul neuronal utilizat

Cei mai importanţi parametrii ai modelului *leaky integrate-and-fire* sunt prezente şi în modelul utilizat în această aplicaţie, adaptat pentru a putea fi implementat în procesorul MicroBlaze încorporat în hardware-ul reconfigurabil FPGA. Astfel, variabila potenţialului de membrană (PM) are în continuare un rol primordial în funcţionarea neuronului. Fiecărei intrări se asociază o pondere sinaptică care se vor însuma pentru a obţine PM. Valoarea de saturaţie a PM va fi notată în continuare cu THS (ca şi în Figura 5.61, care ilustrează modelul neuronal) reprezentând potenţialul de prag, care dacă este depăşit, neuronul se va activa, emiţând un impuls axonal. În urma unui astfel de eveniment, neuronul va trece în faza de hiperpolarizare prin resetarea PM la o valoare mai mică decât cea de repaus. Această din urmă valoare este cea până la care va scade PM cu valoarea de scurgere (LEAK) la fiecare pas de procesare în lipsa impulsurilor de intrare.

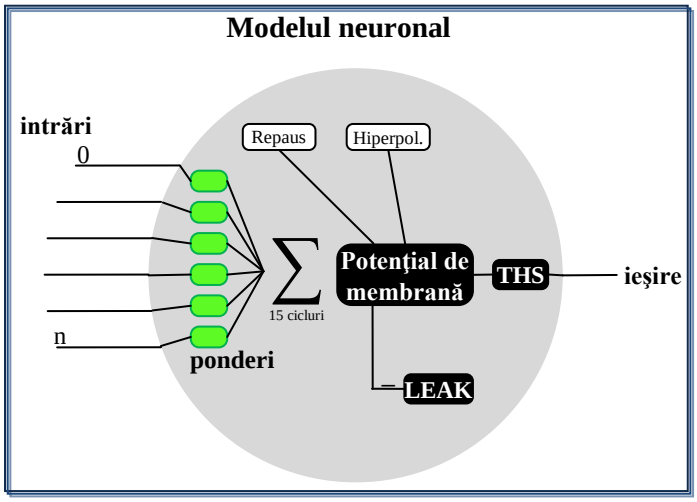


Figura 5.61 Modelul neuronal pulsativ utilizat în aplicaţia de recunoaştere caractere

Funcţionarea neuronului (Figura 5.62) este procesată în 15 paşi. Potenţialul de membrană este iniţializat cu valoarea de repaus. Valoarea de hiperpolarizare este setată la aproximativ 1-2% din valoarea THS.

În fiecare pas se evaluează PM conform ecuaţiei de mai jos

$$\mu = \mu + \sum_i x_i * \beta_i^{jk} - \mu_{scu} \qquad \text{Ec. 74}$$

cu notaţii conform celor din Figura 5.34 şi Figura 5.36.

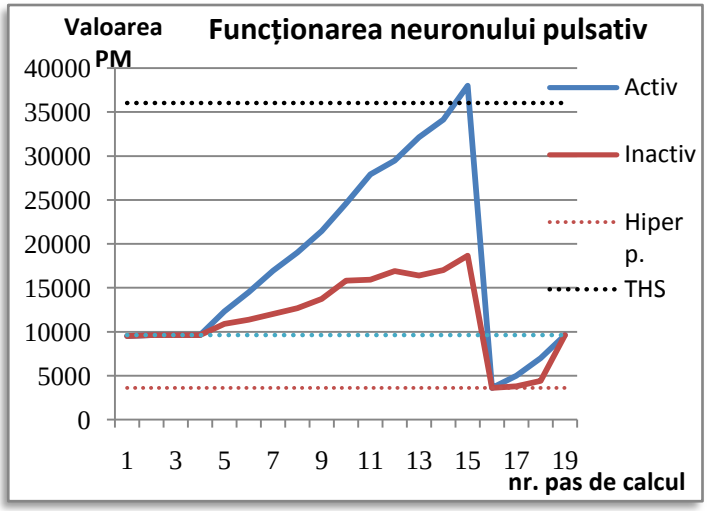


Figura 5.62 Principiul de funcţionare a modelului neuronal utilizat în aplicaţia de recunoaştere caractere

Aşadar, PM va creşte la fiecare pas cu valoarea sumei ponderilor sinapselor care au recepţionat impuls de intrare şi va scădea cu valoarea de scurgere. După epuizarea celor 15 paşi, se va evalua valoarea PM, comparându-se cu cea de prag (THS) şi se va decide dacă neuronul se va activa sau nu. În pasul următor PM se v reseta la valoarea de hiperpolarizare.

Structura şi funcţionarea neuronilor stratului ascuns antrenaţi pentru coloanele matricii de intrare

Aceşti neuroni sunt realizaţi cu opt intrări pre-sinaptice fiecare. Fiecare pondere sinaptică a fost iniţializată cu valori aleatoare generate în Matlab ele fiind în domeniul inferior de 10-20% al intervalul valorilor ponderilor care a fost setat între 0 şi 1200 şi stocat în variabile de tip întreg în procesorul MicroBlaze. Valoarea de prag a acestor neuroni a fost aleasă astfel încât aceştia se vor activa numai atunci, când cel puţin 50% dintre intrările lor primesc impuls pre-sinaptic (4 în cazul de faţă). Ecuaţiile care au fost utilizate pentru calculul valorilor PM de repaus respectiv al THS (θ) sunt date mai jos:

$$\mu_{rep} = n_{act} * \frac{\beta_i^{jk}{}_{max} - \beta_i^{jk}{}_{min}}{i} * 4 = 9600 \qquad \textit{Ec. 75}$$

$$\theta = n_{act} * \frac{\beta_i^{jk}{}_{max} - \beta_i^{jk}{}_{min}}{i} * 15 = 36000 \qquad \textit{Ec. 76}$$

unde $n_{act}=4$, este numărul intrărilor active.

Valoarea de hiperpolarizare s-a setat la 3200, deoarece această valoare aproximează 1% din valoarea de prag THS iar valoarea de scurgere este 600.

Structura şi funcţionarea neuronilor stratului ascuns antrenaţi pentru liniile matricii de intrare

Neuronii din această categorie dispun de şase intrări pre-sinaptice. Valorile ponderilor sinaptice sunt iniţializate în mod identic cu cazul anterior, având aceeaşi plajă de valori. Similar, valoarea de prag a fost setată ca neuronii să se activeze dacă cel puţin trei intrări sunt active (50%). În acest caz parametrii modelului sunt calculaţi astfel:

$$\mu_{rep} = n_{act} * \frac{\beta_i^{jk}{}_{max} - \beta_i^{jk}{}_{min}}{i} * 4 = 7200 \qquad \textit{Ec. 77}$$

$$\theta = n_{act} * \frac{\beta_i^{jk}{}_{max} - \beta_i^{jk}{}_{min}}{i} * 15 = 27000 \qquad \textit{Ec. 78}$$

unde $n_{act}=3$, este numărul intrărilor active.

Notaţiile din ecuaţiile de mai sus sunt cele din Figura 5.34 şi Figura 5.36. Valoarea de hiperpolarizare s-a setat la 2400, deoarece această valoare aproximează 1% din valoarea de prag THS iar valoarea de scurgere este la fel 600.

Structura şi funcţionarea neuronilor stratului de ieşire

Neuronii stratului de ieşire nu diferă mult de cei din stratul ascuns, cu excepţia numărului de intrări – paisprezece în cazul celor de ieşire – respectiv a valorilor unor parametrii. Valorile ponderilor au fost iniţializate şi mărginite în mod identic cu cele prezentate anterior.

Totodată s-a considerat, că un caracter dintre cele opt pentru care va fi antrenat RNP – ceea ce a determinat şi numărul neuronilor de ieşire, egal cu şapte – va activa în stratul ascuns maxim patru neuroni, adică va deţine patru proprietăţi detectabile simultan de acest strat. De aici rezultă, că valorile θ respectiv μ_{rep} vor fi calculate în mod identic cu cele descrise în Ec. 75 respectiv Ec. 76. Valoarea de hiperpolarizare s-a setat la 3200, deoarece această valoare aproximează 1% din valoarea de prag THS iar valoarea de scurgere este la fel 600. Valoarea prescrisă pentru aceşti neuroni depinde de caracterul prezentat la intrări în ciclul actual de învăţare.

5.6.2.5. Prezentarea algoritmului de învăţare implementat în RNP

Caracterul 0: 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1
Caracterul 1: 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0
Caracterul 2: 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1
Caracterul 3: 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1
Caracterul 4: 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0
Caracterul 5: 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1
Caracterul 6: 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1
Caracterul 7: 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0

Figura 5.63 Structura setului de antrenare a stratului ascuns (a se nota, că primele 6 valori sunt pentru neuronii de coloană iar următoarele 8 pentru cele de linie)

Reţeaua neuronală pulsativă implementată pentru această aplicaţie dispune de 48 de intrări şi 8 ieşiri. Neuronii din stratul ascuns respectiv cel de ieşire vor fi antrenaţi cu algoritmi asemănători. În acest sens am construit seturi de antrenare separate, care conţin valori de intrare şi valorile de ieşire prescrise fiecărui neuron. Aceste valori prescrise sunt ilustrate în Figura 5.63. Aceste valori vor fi prezentate pe rând RNP în decursul ciclurilor de antrenare.

Astfel, se vor prezenta valorile setului de antrenare neuronilor din stratul ascuns, iar apoi ieşirile acestora vor fi propagate spre neuronii de ieşire care vor primi în acelaşi timp şi valorile prescrise, conform setului de antrenare propriu. Un ciclu de antrenare constă în prezentarea fiecărui caracter la intrările RNP.

În cadrul funcţiilor ce implementează neuronii individuali se pot separa 15 paşi de calcul în cadrul cărora se însumează valorile ponderate primite, generate de un caracter prezentat la intrare. Dacă suma rezultantă depăşeşte valoarea de prag, neuronul se va activa. În acest moment are loc adaptarea ponderilor sinaptice conform unui set de reguli bazate pe cele descrise de HEBB, într-o variantă supervizată, după cum se prezintă în Tabelul 16. Aceste condiţii prezentate determină dacă şi în ce direcţie se vor modifica ponderile.

Caracterul 0: 1, 0, 0, 0, 0, 0, 0, 0
Caracterul 1: 0, 1, 0, 0, 0, 0, 0, 0
Caracterul 2: 0, 0, 1, 0, 0, 0, 0, 0
Caracterul 3: 0, 0, 0, 1, 0, 0, 0, 0
Caracterul 4: 0, 0, 0, 0, 1, 0, 0, 0
Caracterul 5: 0, 0, 0, 0, 0, 1, 0, 0
Caracterul 6: 0, 0, 0, 0, 0, 0, 1, 0
Caracterul 7: 0, 0, 0, 0, 0, 0, 0, 1

Figura 5.64 Structura setului de antrenare a stratului de ieşire

Valoarea cu care creşte sau scade o pondere este egală cu 3% din valoarea maximă a ponderilor $\beta_{i\ max}^{jk}$.

Tabelul 16 Algoritmul de învăţare supervizat utilizat

Intrare	Ieşire	Valoare prescrisă	Adaptare ponderi
0	0	0	-
0	0	1	-
0	1	0	*Scade?*
0	1	1	-
1	0	0	-
1	0	1	**Creşte**
1	1	0	**Scade**
1	1	1	*Creşte?*

În cazul în care valoarea de intrare este 1 iar ieşirea neuronului nu se activează în timp ce valoarea prescrisă ar cere acest lucru, sinapse intrării respective va fi ajustată prin creşterea ponderii

corespunzătoare. Dacă intrarea şi ieşirea sunt ambele active dar activarea axonală nu a fost prescrisă, ponderea respectivă va fi diminuată. În celelalte cazuri ponderile vor rămâne neschimbate.

Ar exista, însă, încă două cazuri în care s-ar părea că ar fi nevoie de adaptarea ponderilor. Primul astfel de caz este acela în care atât intrarea, ieşirea cât şi valoarea prescrisă este egală cu 1, dar creşterea ponderii în acest caz ar duce la o supra-învăţare a reţelei neuronale, deoarece putem considera, că antrenarea şi-a atins deja scopul. Al doilea caz asemănător este acela în care ponderea ar trebui să scadă, dacă intrarea este inactivă, ieşirea este activă iar valoarea prescrisă nu cere activare. Aici putem considera, că valoarea respectivă de intrare nu a contribuit la învăţarea caracterului respectiv, deci nu va fi luat în considerare. Conform acestor observaţii în aceste cazuri ponderile nu au fost modificate.

5.6.2.6. Proiectarea componentelor software ale sistemului încorporat

Toate funcţionalităţile prezentate în secţiunile anterioare au fost implementate într-un program C compilat pentru şi rulat pe procesorul Xilinx MicroBlaze. Mediul de dezvoltare utilizat, Xilinx EDK (prezentat în subcapitolul 5.6.4) generează drivere şi funcţii necesare pentru accesarea perifericelor introduse în proiect.

Faza de analiză şi proiectare a unui proiect trebuie să fie gata înainte de realizarea codului, pentru a obţine o atenţie mărită din partea dezvoltatorilor. Orice dezvoltator recunoaşte importanţa acestor faze deoarece s-a dovedit că de acestea depinde producerea şi refolosirea de software. Pentru analiza şi proiectarea programelor s-au creat limbajele de modelare. Unul din aceste limbaje de modelare este limbajul de modelare unificat - UML (The Unified Modeling Language).

UML nu este un simplu limbaj de modelare orientat pe obiecte, ci în prezent, este limbajul universal standard pentru dezvoltatorii software din toată lumea. UML este succesorul propriu-zis al celor mai bune trei limbaje de modelare anterioare orientate pe obiecte (Booch, OMT, şi OOSE). UML se constituie din unirea acestor limbaje de modelare şi în plus deţine o expresivitate care ajută la rezolvarea problemelor de modelare pe care vechile limbaje nu o aveau.

Limbajul de modelare modificat (UML - The Unified Modeling Language) oferă arhitecturi de sisteme ce funcţionează pe analiza şi proiectarea obiectelor cu un limbaj corespunzător pentru specificarea, vizualizarea, construirea şi documentarea artefactelor sistemelor software şi de asemenea pentru modelarea în întreprinderi. UML este un limbaj de modelare care oferă o exprimare grafică a structurii şi comportamentului software. Pentru această exprimare grafică se utilizează notaţiile UML.

Notaţiile UML constituie un element esenţial al limbajului pentru realizarea propriu-zisă a modelării şi anume partea reprezentării grafice pe care se bazează orice limbaj de modelare. Modelarea în acest limbaj se realizează prin combinarea notaţiilor UML în cadrul elementelor principale ale acestora denumite diagrame. În cadrul UML-ului descoperim nouă tipuri de diagrame: diagrama cazurilor de utilizare, diagrama de secvenţă, diagrama de colaborare, diagrama de clase (cea mai utilizată), diagrama de stări, diagrama de componente, diagrama de construcţie, diagrama de obiecte, diagrama de activităţi

Analiza unei aplicaţii implică realizarea mai multor categorii de modele, dintre care cele mai importante sunt:

- Modelul de utilizare: realizează modelarea problemelor şi a soluţiilor acestora în maniera în care le percepe utilizatorul final al aplicaţiei. Diagramă asociată: ***diagramă de cazuri de utilizare***
- Modelul structural: se realizează pe baza analizei statice a problemei şi descrie proprietăţile statice ale entităţilor care compun domeniul problemei. Diagrame asociate: ***diagramă de module, diagramă de clase***
- Modelul comportamental: priveşte descrierea funcţionalităţilor şi a succesiunii în timp a acţiunilor realizate de entităţile domeniului problemei. Diagrame asociate: ***diagrama (harta) de stări, diagrama de colaborare, diagrama de interacţiune***

O diagramă oferă utilizatorului un mijloc de vizualizare si de manevrare a elementelor de Modelare. Majoritatea diagramelor se prezintă sub forma unor grafuri, compuse din elemente si arce.

Diagramele pot arăta o parte sau toate caracteristicile elementelor de modelare, conform nivelului de detaliu util în contextul unei diagrame date. Diagramele pot grupa informaţii interdependente, pentru a arăta, de exemplu caracteristicile moştenite de o clasă.

Diagramele UML sunt:

- diagramele cazurilor de utilizare, care prezintă funcţiile sistemului din punct de vedere al utilizatorului;
- diagrame de clasă , care prezintă structura statica în termeni de clase si asocieri (relaţii);
- diagrame de colaborare, care sunt reprezentări spaţiale ale obiectelor, legăturilor şi interacţiunilor;
- diagrame de secvenţă , care prezintă temporal obiectele şi interacţiunile lor;

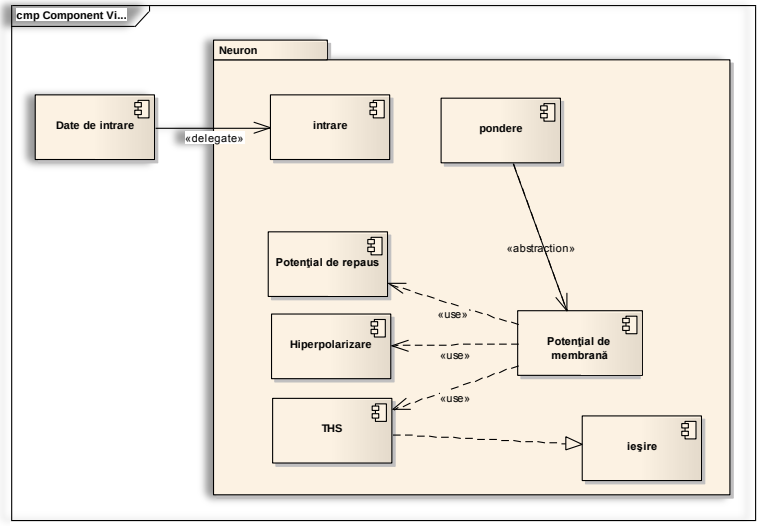


Figura 5.65 Diagrama de componente a modulului neuronului pulsativ implementat software

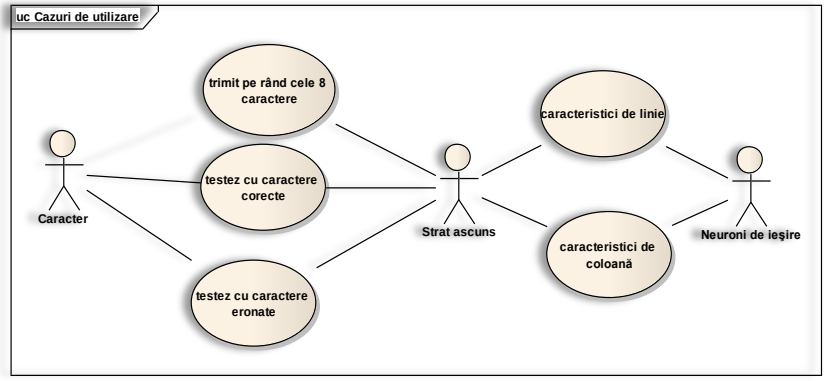


Figura 5.66 Diagrama cazurilor de utilizare (use-case) a RNP implementate pe procesorul MicroBlaze

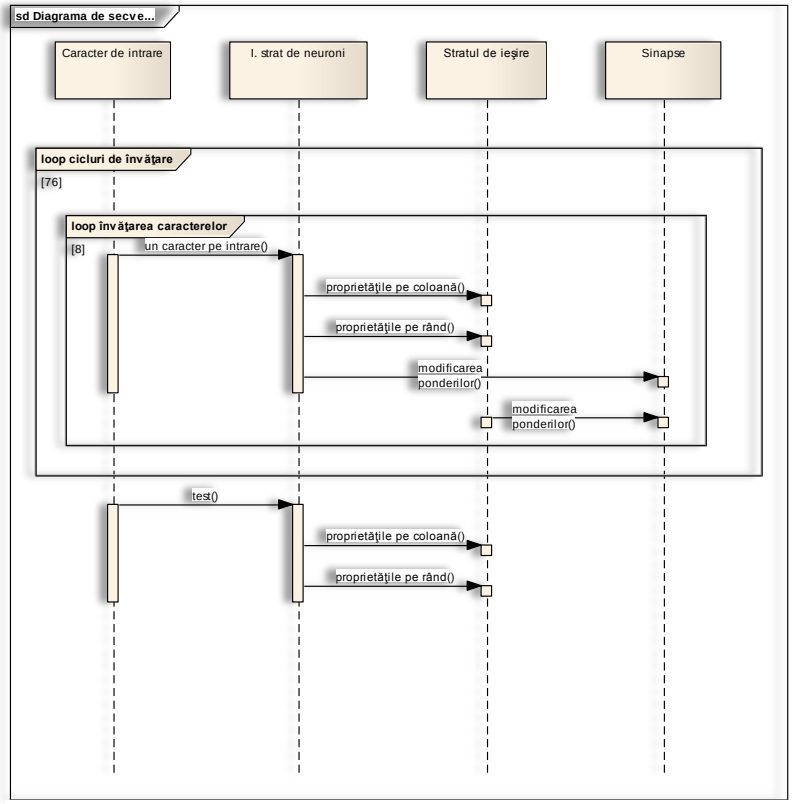


Figura 5.67 Diagrama de secvenţă a RNP implementate pe procesorul MicroBlaze

- diagrame de componente, care prezintă componentele fizice ale unei aplicaţii;
- diagrame de construcţie , care prezintă construcţia componentelor pe dispozitivele hardware;
- diagrame de stări-tranziţii, care prezintă comportamentul unei clase în termeni de stări;
- diagrame de obiecte, care prezintă obiectele şi relaţiile lor, fiind nişte diagrame de colaborare simplificate, fără reprezentarea mesajelor trimise între obiecte;
- diagrame de activităţi, care reprezintă comportamentul unei operaţii în termeni de acţiuni.
- diagramele de colaborare împreună cu cele de

secvenţă se numesc diagrame de interacţiune pe când diagramele de stare mai sunt denumite maşini cu stări finite, automate, etc.

Prima fază a implementării software a fost realizarea modelului neuronal în care putem regăsi toate variabilele prezentate, după cum arată şi Figura 5.65

Programul ilustrat în diagrama de componente din Figura 5.66 (generată cu software-ul Enterprise Architect) are trei actori.

Primul este actorul caracter care livrează rând pe rând caracterele neuronilor stratului ascuns. Actorul din stratul ascuns analizează aceste informaţii apoi trimite spre neuronii din stratul de ieşire caracteristicile de linie şi coloană. Neuronii de ieşire vor învăţa să identifice un anumit caracter.

Scopul principal al diagramei de secvenţă din Figura 5.67 este ilustrarea interacţiunii dintre obiectele afişate şi secvenţa temporală a desfăşurării acestora. În diagramă sunt reprezentate patru obiecte. În primul pas, obiectul *caracter* predă un caracter obiectului *strat ascuns*, asupra căruia se execută operaţiile şi apoi livrează mai departe caracteristicile de linie şi coloană obiectului *neuron de ieşire*. După evaluarea informaţiilor vom trece la modificarea ponderilor din obiectele *sinapse*. Acest proces este executat pentru cele opt caractere completând un ciclu de antrenare care este repetat până când neuronii de ieşire identifică corect caracterele prezentate la intrare.

După antrenarea RNP cu setul de antrenare urmează testarea acesteia. În primul pas se trimite un caracter fără erori obiectului *strat ascuns*. Acest caracter propagându-se spre obiectul *neuron de ieşire* obţinem rezultatul învăţării. După cum se vede în diagramă, în decursul testării ponderile sinaptice nu sunt alterate. Această testare se va repeta apoi şi cu caractere de intrare zgomotoase (cu erori de reprezentare).

5.6.2.7. Rezultate experimentale

În decursul antrenării am calculat eroarea de aproximare comparând valorile de ieşire şi valorile prescrise şi am incrementat un numărător dacă acestea nu corespund. Dacă gradientul acestui numărător va fi zero, însemnă că reţeaua a învăţat problema propusă.

În diagrama din Figura 5.68 putem observa că eroarea descreşte, având valori mari în primele cicluri, apoi începând cu aproximativ ciclul numărul 5 scade abrupt ca apoi să descrească lent spre zero.

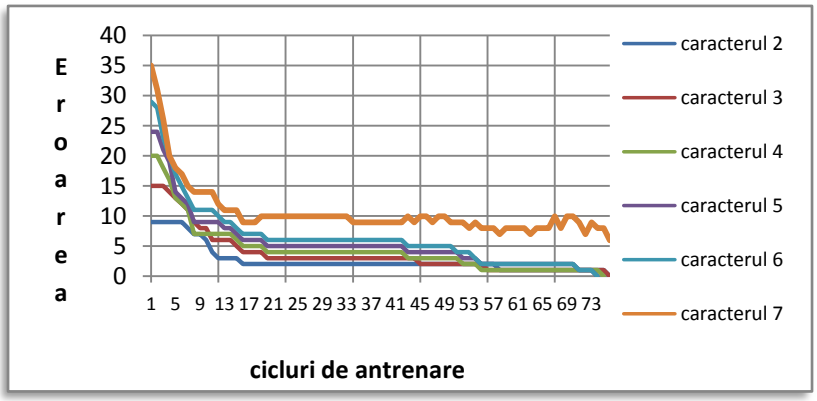


Figura 5.68 Variaţia erorii în timpul antrenării (pentru caracterele 2,3,4,5,6,7)

În Figura 5.69 se poate vedea cum variază potenţialul de membrană al neuronului nr. 1 în timpul primului ciclu (de 15 paşi) de antrenare al caracterului 0, neuron care are prescris activare pentru acest caracter. Se observă că PM scade din cauza lipsei impulsurilor de intrare. Inflexiunea din pasul 4 se datorează faptului că în experimentul respectiv am iniţializat PM cu valoarea de hiperpolarizare, şi am prevăzut ca şi în lipsa impulsurilor de intrare PM să atingă valoarea de repaus cel târziu în patru paşi.

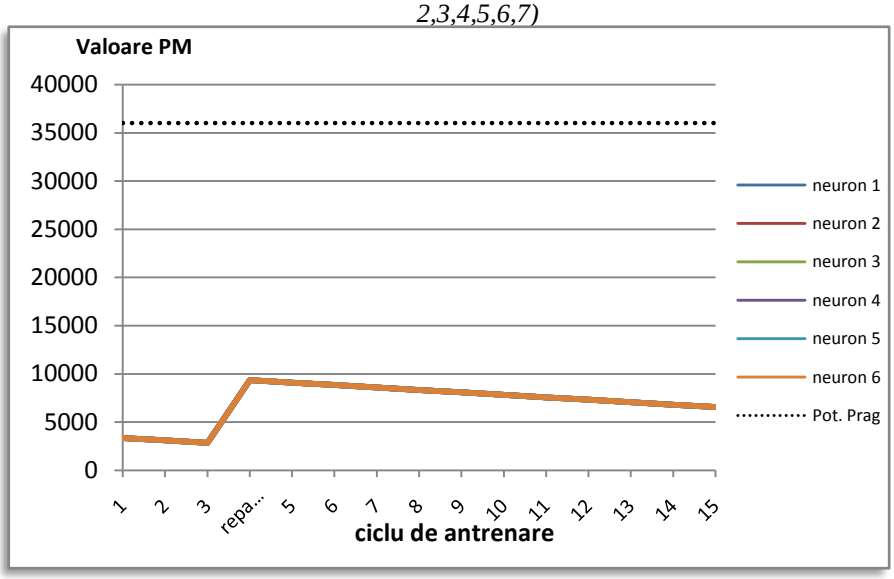


Figura 5.69 Variaţia potenţialelor de membrană a 6 neuroni în timpul primului ciclu de antrenare (al caracterului 0)

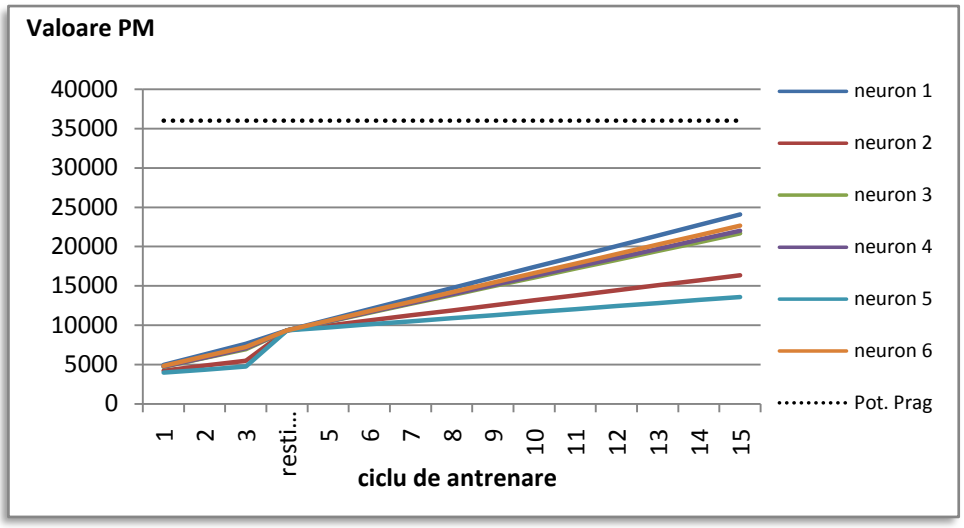


Figura 5.70 Variaţia potenţialelor de membrană în timpul ciclului 32 de antrenare (al caracterului 0)

Se poate vedea în Figura 5.70 că procesul de învăţare va ridica treptat valorile potenţialelor de membrană, astfel în ciclul nr. 32 PM al neuronului 1, are deja valoarea cea mai mare, dar mai trebuie să crească pentru a atinge valoarea de prag.

Variaţia potenţialelor de membrană în timpul ultimului ciclu de antrenare (al caracterului 0) din Figura 5.71 arată că neuronul 1 se activează, învăţând cu succes caracterul 0.

În cele ce urmează se vor prezenta reultate experimentale similare, obţinute prin măsurători efectuate în timpul antrenării celorlalte caractere.

Procesorul MicroBlaze trimite unui calculator prin portul serial aceste date ale RNP implementate, unde sunt salvate şi vitualizate de un program Matlab.

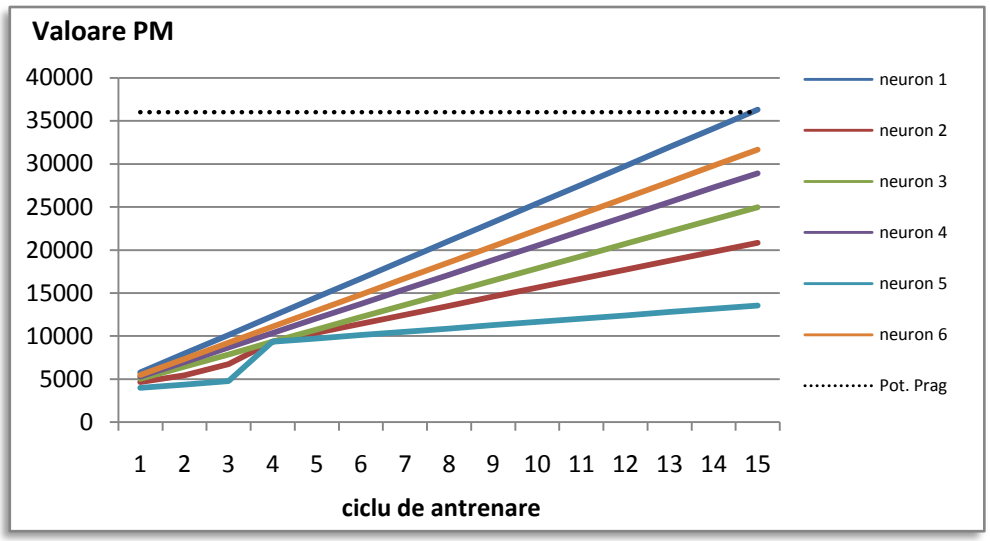


Figura 5.71 Variaţia potenţialelor de membrană în timpul ultimului ciclu de antrenare (al caracterului 0)), care este învăţat de neuronul 1

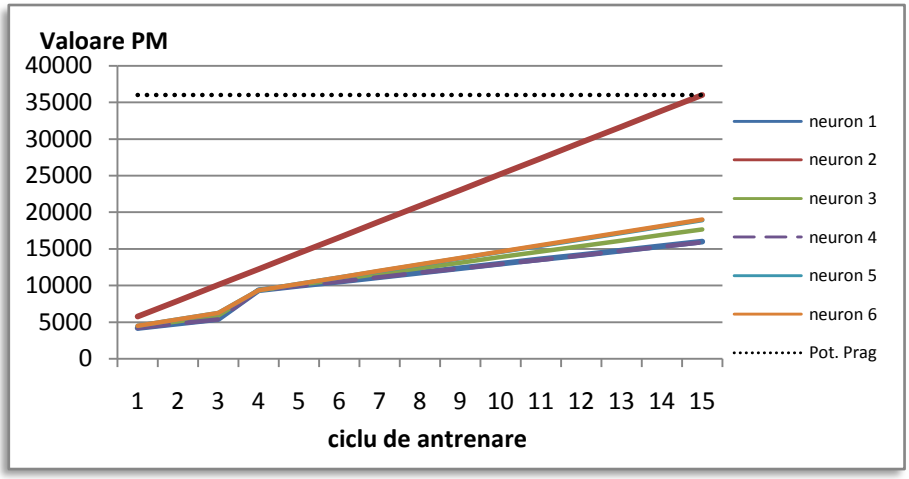


Figura 5.72 Variaţia potenţialelor de membrană în timpul ultimului ciclu de antrenare (al caracterului 1), care este învăţat de neuronul 2

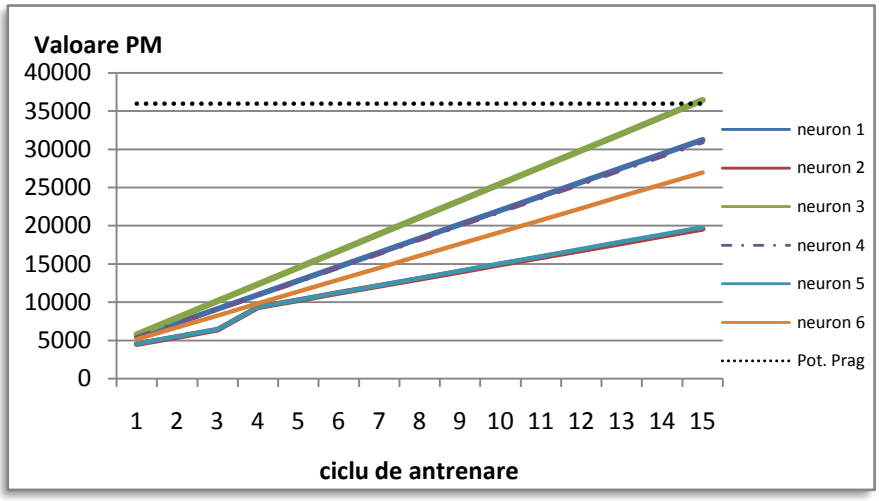


Figura 5.73 Variaţia potenţialelor de membrană în timpul ultimului ciclu de antrenare (al caracterului 2), care este învăţat de neuronul 3

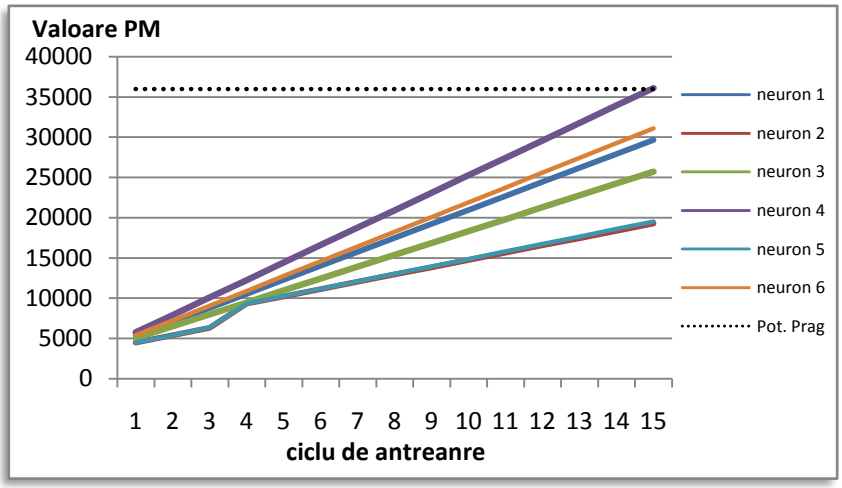


Figura 5.74 Variaţia potenţialelor de membrană în timpul ultimului ciclu de antrenare (al caracterului 3), care este învăţat de neuronul 4

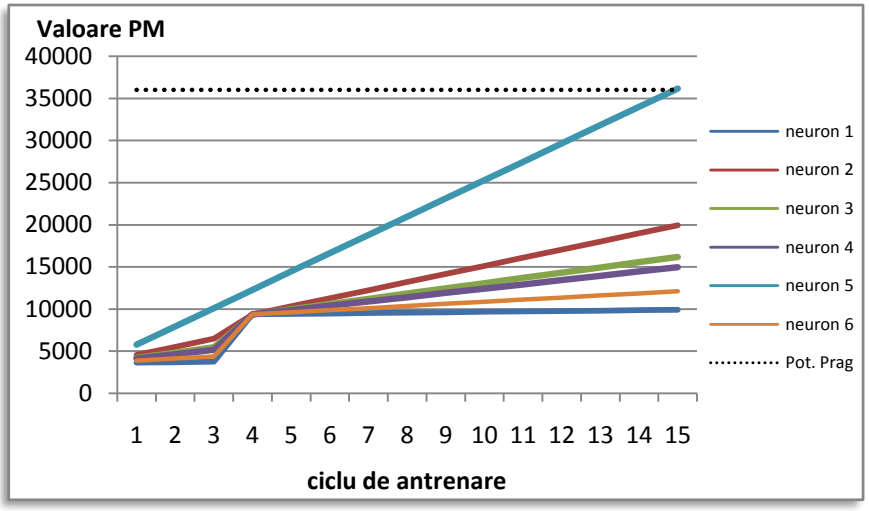


Figura 5.75 Variaţia potenţialelor de membrană în timpul ultimului ciclu de antrenare (al caracterului 4), care este învăţat de neuronul 5

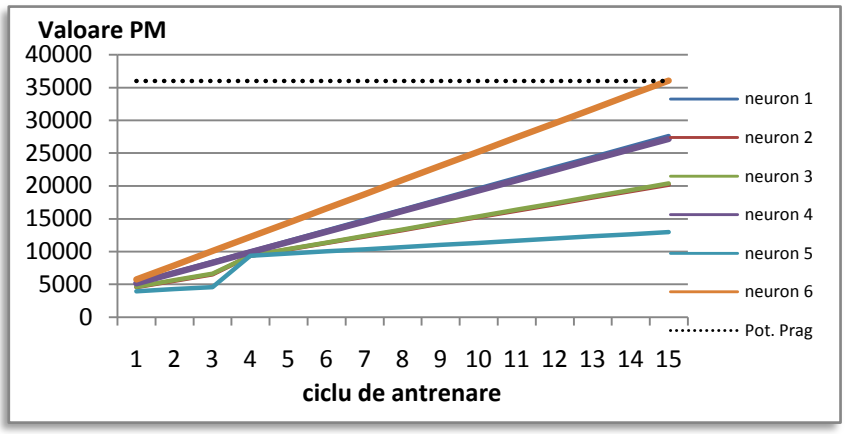
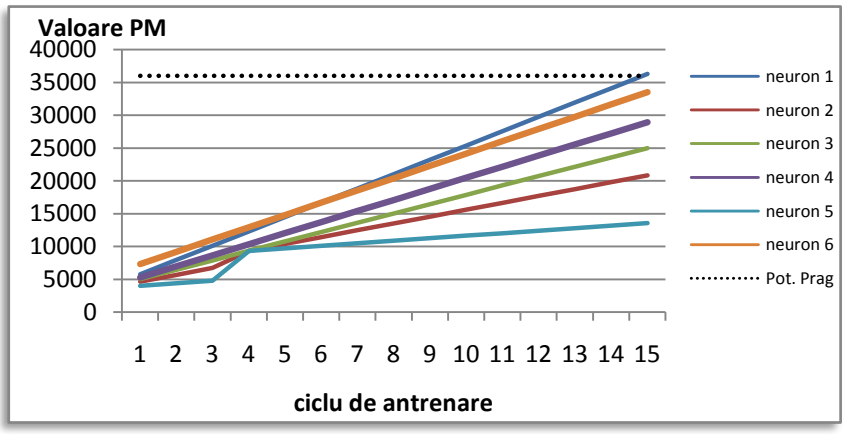


Figura 5.76 Variaţia potenţialelor de membrană în timpul ultimului ciclu de antrenare (al caracterului 5), care este învăţat de neuronul 6

În decursul testării RNP după învăţare am obţinut rezultate de recunoaştere pentru caracterele de la 0-7 corecte în 98% din cazuri. Cele mai multe erori au fost la diferenţierea caracterelor 5 şi 6.

Pentru a arăta robusteţea aplicaţiei am testat RNP implementată în FPGA şi cu intrări perturbate de zgomot. În testul ilustrat de Figura 5.77 am prezentat la intrarea RNP un caracter 0 cu erori de reprezentare, dar se vede că sistemul a recunoscut caracterul fără probleme.



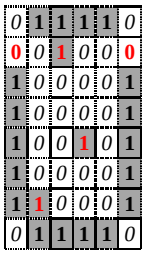
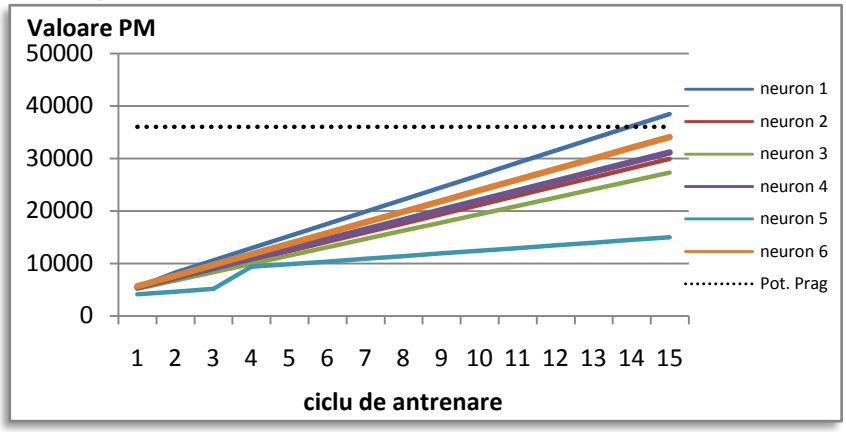


Figura 5.77 Testarea RNP cu caracterul 0 perturbat de zgomot

Dacă introducem o diferenţă mai mare între caracterul original de intrare şi cel zgomotos de test, alterând astfel caracteristicile de linie sau coloană ale acesteia, RNP va reuşi şi în acest caz să identifice corect caracterul ().



0	1	1	1	1	0
1	1	1	1	1	1
1	0	0	0	0	1
1	0	0	0	0	1
1	0	0	0	0	1
1	0	0	0	0	1
1	0	0	0	0	1
0	1	1	1	1	0

Figura 5.78 Figura 5.79 testarea RNP cu caracterul 0 perturbat de zgomot accentuat

5.6.3. Implementare benchmark: Clasificarea setului de date Wisconsin Breast Cancer

5.6.3.1. Formularea problemei de clasificare

Testul utilizează a baza de date Wisconsin Breast Cancer Database (WBCD), care urmăreşte clasificarea malignă sau benignă a unor ţesuturi mamare, pe baza a nouă proprietăţi ale nucleelor celulare prelevate. Această bază de date a fost compilată de Dr. William H. Wolberg (Mangasarian & Wolberg, 1990) de la University of Wisconsin Hospitals, Madison, SUA. Conţine 699 de elemente cu câte 9 atribute, fiecare reprezentând o caracteristică a ţesuturilor cercetate (conform sursei (Open source data visualization and analysis for novice and experts, 2005) respectiv (National Science Foundation, 2007)):

- *Clump thickness*
- *Unif_Cell_Size*
- *Unif_Cell_Shape*
- *Marginal_Adhesion*
- *Single_Cell_Size*
- *Bare_Nuclei*
- *Bland_Chromatine*
- *Normal_Nucleoli*
- *Mitoses type*

5.6.3.2. Elementele hardware ale sistemului încorporat dezvoltat

În cadrul acestei aplicaţii am utilizat module hardware predefinite ale mediului de dezvoltare Xilinx Platform Studio precum şi module proprii implementate în limbaj VHDL (Figura 5.80). Elementul principal al implementării este şi în acest caz procesorul Xilinx MicroBlaze, la care am conectat mai multe module de memorie respectiv dispozitive periferice prin magistrala de 32 de biţi Processor Local Bus (PLB).

Perifericul principal care a fost dezvoltat este acela care implementează codificarea valorilor de intrare a reţelei neuronale pulsative care va descris detaliat în subcapitolul următor.

Acest periferic de generare a impulsurilor de intrare decalate temporal conţine 5 registre prin care se realizează comunicare dintre acesta şi procesorul

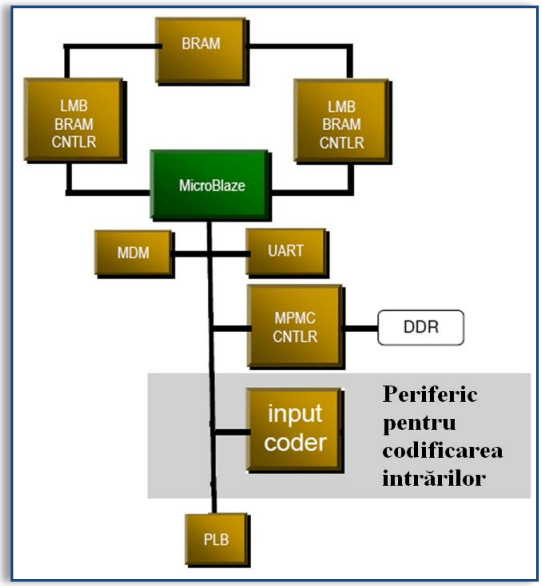


Figura 5.80 Structura hardware a sistemului implementat

MicroBlaze prin magistrala PLB. Pentru a evita coliziuni în scrierea acestor registre dintre aceşti doi iniţiatori de transferuri ai magistralei, am separat rolul registrelor. Astfel, MicroBlaze va putea scrie numai în unul dintre registre celelalte 4 fiind configurate ca putând fi numai citite de către procesor.

5.6.3.3. Principiul de funcţionare perifericului de codificare a intrărilor

Scopul acestui circuit este de a codifica valorile de intrare ale reţelei neuronale – care sunt de fapt cele nouă parametrii ale fiecărei mostră din setul de date – în impulsuri decalate temporal, pentru a putea fi prelucrate de către RNP cu procesor încorporat implementată.

Elementele bazei de date WBCD, care sunt valori cu două zecimale (scalate, pentru a le putea reprezenta ca numere întregi în FPGA) sunt stocate într-o memorie BRAM încorporată în acest periferic. Procesorul MicroBlaze trimite o valoare de 10 biţi într-unul dintre registrele perifericului care va fi utilizată pentru a adresa memoria valorilor setului WBCD. Astfel vom obţine o valoare de 64 biţi care va reprezenta o mostră, conţinând toate cele 9 proprietăţi, pe care baza de date le reprezintă în intervalul 0-10 cu o precizie de o zecimală. Pentru a stoca numere întregi în FPGA aceste valori au fost scalate între 0-100 astfel necesitând 7 biţi pentru reprezentare, ceea ce rezultă în 7*9=63 biţi. Ultimul bit din cei 64 va fi folosit pentru a stoca clasa în care aparţine mostra respectivă (1-malign, 0-belign).

Perifericul este programat să aştepte în stare inactivă sosirea unui front crescător pe semnalul Slv_WR_ack care va apare pe magistrala PLB atunci, când MicroBlaze va efectua prima operaţie de scriere la adresa perifericului. În urma acestui eveniment, perifericul va intra în stare de iniţializare, ce durează 3 cicluri de tact după care va începe secvenţa celor 15 paşi în care cele 9 valori de intrare se vor coda în impulsuri decalate temporal. Semnalul de intrare *Inp_valid* al perifericului de codificare (Figura 5.82) trebuie menţinut activ timp de 18 cicluri de tact pentru ca în cel de-al patrulea ciclu să obţinem la ieşiri

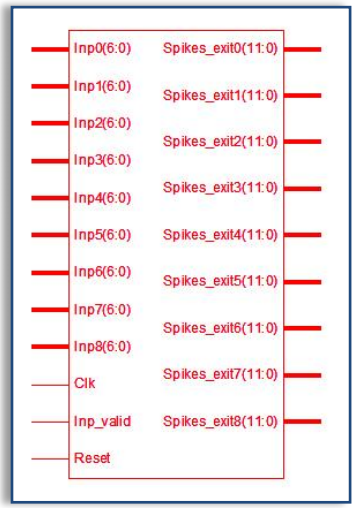


Figura 5.82 Simbolul bloc al perifericului de codificare a intrărilor

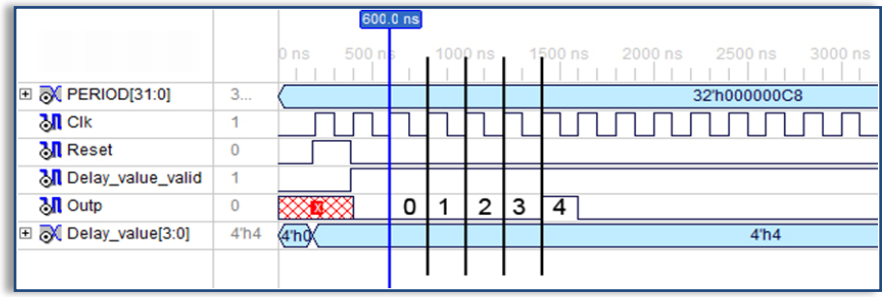


Figura 5.81 Generarea impulsului decalat din valoarea pe 4 biţi citită din BRAM

*Spikes_exit***X** primul set de impulsuri.

Modulul de codificare a intrărilor conţine 9 memorii BRAM de tipul RAMB16_S36_S36 (modul cu port dual pe 32 de biţi, astfel putând accesa 64 de biţi simultan), care stochează valorile decalajelor conform funcţiilor de activare cu care a fost acoperită spaţiul valorilor de intrare.

În cele ce urmează vor fi prezentate două versiuni ale RNP pentru această aplicaţie, una fără şi alta cu strat ascuns. Versiunea mai simplă (Figura 5.86) conţine 12x9=108 neuroni de intrare (Figura 5.84) şi având doi neuroni de ieşire

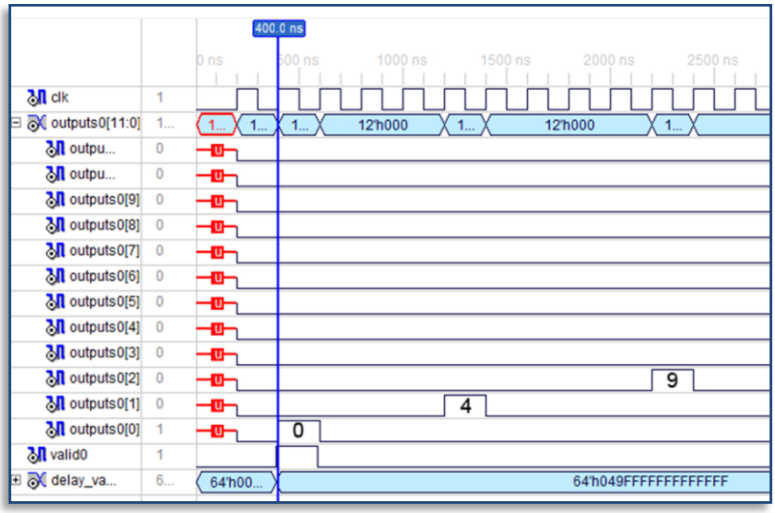


Figura 5.83 Codificarea valorii de intrare în trei impulsuri decalate temporal în decursul unui pas temporal de 15 cicluri de tact

rezultă 216 sinapse. Varianta cu strat ascuns a RNP (Figura 5.87) a fost implementată codificând valorile de intrare cu 16 neuroni de intrare – pentru o precizie ridicată – având 6 neuroni în startul ascuns la care aparţin 16*9*6=864 sinapse şi doi neuroni de ieşire cu 6*2=12 sinapse.

Rezultatele simulării unui neuron de intrare şi a blocului de neuroni de intrare al uneia dintre intrări care codifică valorile sosite pe aceasta în impulsuri decalate temporal sunt prezentate în figurile de mai jos (Figura 5.81 respectiv Figura 5.83).

5.6.3.4. Codificarea valorilor de intrare

Similar cu cele prezentate la aplicaţiile cu procesor PicoBlaze în acest caz cadrul temporal de codificare a intrărilor este de 15 cicluri de tact. În figura alăturată (Figura 5.84) am reprezentat acoperirea cu funcţii triunghiulare a spaţiului intrărilor (valori de la 0 la 100) de către 12 neuroni de intrare. Se poate observa exemplul marcat pe figură, şi anume că valoarea 30 va activa trei neuroni de intrare. Dintre aceşti trei neuroni care codifică valoarea de intrare respectivă, neuronul 3 va emite impuls pre-sinaptic în pasul temporal 1 al cadrului temporal actual, neuronul 2 va emite impuls în pasul 7 iar neuronul de intrare 4 în cel de-al 9-lea ciclu de tact din cadru. restul neuronilor de intrare nu va emite impuls pentru această valoare.

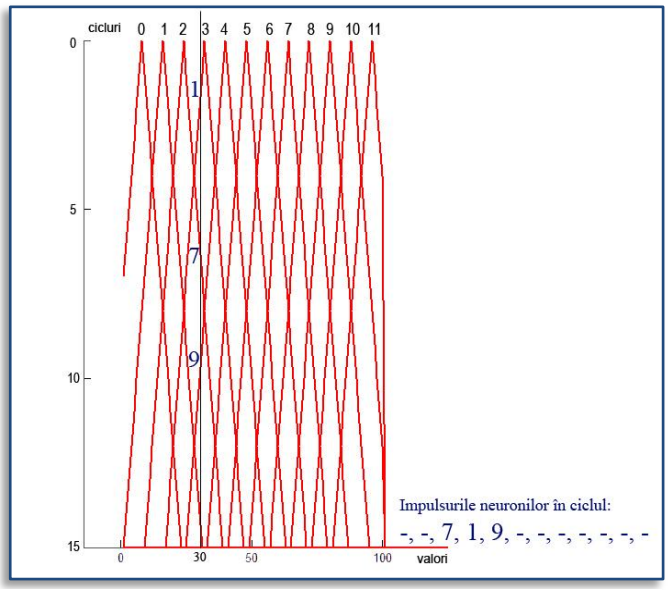


Figura 5.84 Funcţiile triunghiulare ale domeniilor receptive utilizate în cazul RNP fără strat ascuns

Deoarece RNP ce implementează această aplicaţie de clasificare are nouă intrări, avem nevoi de tot atâtea module BRAM pentru a stoca aceste valori ale momentelor de activare ale neuronilor de intrare. Ca şi în cazul aplicaţiilor prezentate anterior, aceste module sunt de tip dual-port pentru a putea accesa mai multe valori simultan şi a accelera procesarea reţelei neuronale.

Figura 5.85 prezintă o variantă îmbunătăţită a acestei codificări, utilizând 16 neuroni pentru o intrare, utilizată în a doua variantă a RNP pentru această aplicaţie, care implementează şi un strat ascuns de neuroni. Această alegere a fost luată pentru a creşte precizia codificări deci şi a sensibilităţii clasificării. A rezultat o codificare în care pentru orice valoare de intrare se vor activa patru neuroni în loc de trei (de exemplu în Figura 5.85, valoarea 35 va activa neuronii nr. 6, 7, 8 şi 9), activând procesul de învăţare în mai multe sinapse.

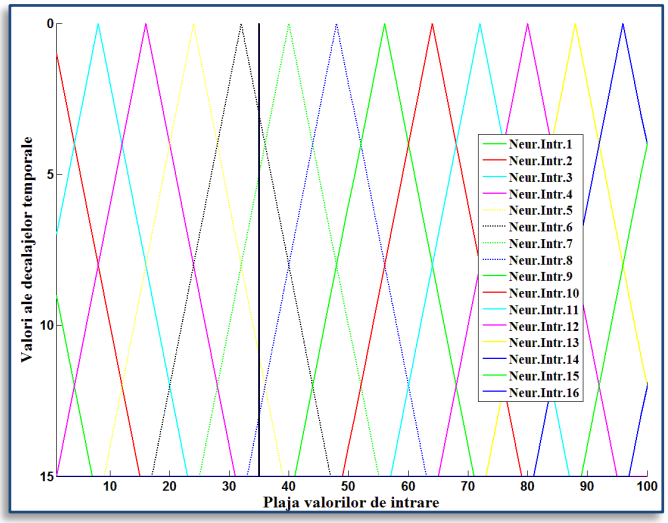


Figura 5.85 Funcţiile triunghiulare ale domeniilor receptive utilizate în cazul RNP cu strat ascuns

5.6.3.5. Structura reţelelor neuronale pulsative implementate pentru rezolvarea clasificării setului de date Wisconsin

Am experimentat realizarea practică a mai multor variante a RNP cu procesor încorporat MicroBlaze pentru aplicaţia de clasificare a bazei de date WBCD pentru a găsi o soluţie care să ofere o rezolvare bună, să funcţioneze în timp real şi în acelaşi timp să nu depăşească necesarul de resurse hardware oferit de sistemul de dezvoltare FPGA utilizat (Nexys2-1200). După cum s-a sugerat şi în subcapitolele anterioare se va prezenta în continuare structura şi funcţionarea a două dintre aceste RNP.

Versiunea 1 – RNP cu două straturi. Arhitectura acestei reţele neuronale artificiale pulsative implementate hardware este prezentată în Figura 5.86. După cum se vede şi în figură, există două straturi de neuroni, cel de intrare, în care găsim nouă blocuri de neuroni de intrare – corespunzătoare celor nouă proprietăţi a fiecărei mostre din WBCD, care formează valorile de intrare ale RNP – cu câte 12 neuroni ce codifică valorile de intrare în impulsuri decalate temporal. Fiecărui neuron din acest strat îi corespunde câte o sinapsă ale celor doi neuroni pulsativi din stratul de ieşire. Rezultă, că RNP conţine în total 9*12*2=216 sinapse, care sunt implementate de funcţii software rulate de procesorul MicroBlaze, ponderile asociate fiind stocate în memoria de date a acestuia. Tot aici este implementat şi algoritmul somei, funcţionând ca două funcţii scrise în limbaj *C*, realizând procesarea celor doi neuroni de ieşire, care se vor activa atunci când mostra de la intrare aparţine uneia dintre cele două clase (malign sau benign) asociate acestuia. În contrast cu aceste componente se află neuronii de intrare care sunt implementaţi utilizând limbajul de descriere hardware VHDL, devenind astfel un periferic separat – conectat la magistrala PLB al MicroBlaze – care funcţionează în paralel cu procesorul, fiind construiţi din elemente reconfigurabile distincte faţă de acesta.

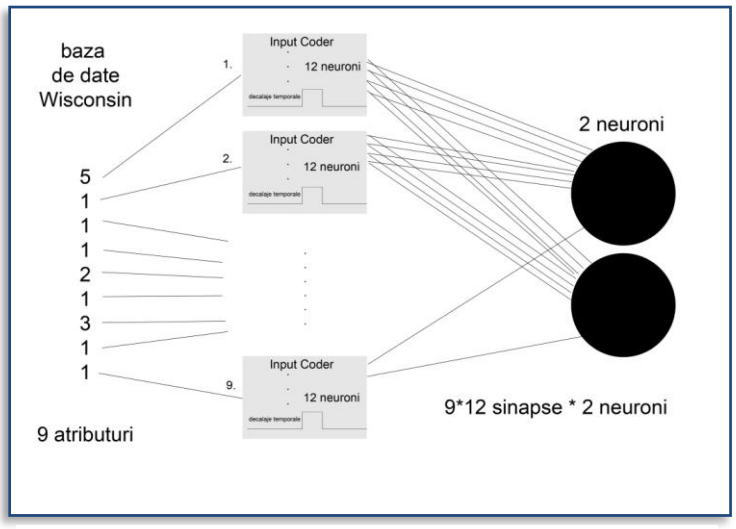


Figura 5.86 Prima versiune a RNP implementate pentru aplicaţia benchmark de clasificare a setului de date WDBC

Neuronii de ieşire sunt implementaţi conform aceluiaşi model de bază, *leaky integrate-and-fire*, adaptat pentru implementare în procesor încorporat soft-core. Calcularea parametrilor acestui model s-a efectuat după cum urmează. La alegerea valorii de prag a potenţialului de membrană s-a luat în considerare că în decursul celor 15 cicluri ale cadrului temporal prin cele 108 intrări post-sinaptice ale unui neuron de ieşire poate sosi un număr total de 27 impulsuri, deoarece din cele 9 blocuri de neuroni de intrare fiecare va avea câte trei neuroni activi în acel cadru. Valorile ponderilor au fost mărginite între 0 şi 1200 (reprezentabile pe min. 11 biţi).

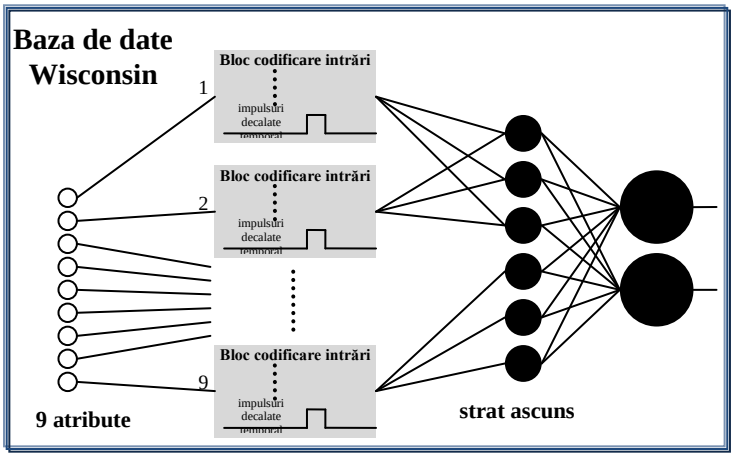


Figura 5.87 Versiunea cu strat ascuns a RNP implementate pentru aplicaţia benchmark de clasificare a setului de date WDBC

Rezultă, că valoarea THS=27*600=16200, unde 600 este valoarea medie a ponderilor sinaptice. Valoarea de repaus a PM s-a ales a fi 20% din THS, cea de hiperpolarizare 1% iar valoarea de scurgere 3%.

Versiunea 2 – RNP cu trei straturi. Diferenţa majoră faţă versiunea anterioară constă în introducerea unui strat ascuns de neuroni pulsativi pentru a îmbunătăţi performanţele RNP. Acest strat conţine 6 neuroni, întreaga reţea fiind complet conectată (Figura 5.87). De asemenea s-a schimbat şi acoperirea plajei valorilor de intrare, utilizându-se 16 neuroni de intrare/bloc, dintre care vor fi activi câte patru neuroni într-un cadru de timp de 15 cicluri de tact. Aceste schimbări de parametrii au dus la modificări radicale în structura perifericului de codificare a valorilor de intrare, crescând complexitatea şi necesarul de resurse reconfigurabile a acesteia. Un salt important a intervenit şi în numărul de sinapse ale reţelei, acest număr devenind (de la 216 în cazul anterior) 9*16*6=864 între stratul de intrare şi cel ascuns, respectiv 6*2=12 între cel ascuns şi neuronii de ieşire, în total 876 sinapse. Acest lucru înseamnă

totodată *o creştere de 4 ori a numărului de variabile pentru stocarea ponderilor* sinaptice, ceea ce a dus la situaţia în care modulele memorie BRAM încorporate în circuitul FPGA nu au mai fost suficiente pentru a stoca programul scris pe procesorul MicroBlaze, segmentul de date crescând în dimensiuni. Singura rezolvare a acestei probleme a constat în plasarea acestui segment în modulul de memorie DDRAM extern, prezent pe placa de dezvoltare Nexys2 utilizată. Această modificare, însă implică şi introducerea în proiectul dezvoltat în mediul Xilinx EDK (descris la subpunctul 5.6.4), a unui modul (predefinit în biblioteca de circuite EDK) *memory manager* care implementează protocolul de acces la acest tip de memorie, modul care utilizează la rându-i un cuantum important de resurse reconfigurabile. Am ajuns astfel, la o utilizare de peste 95% a capacităţii circuitului FPGA XC3S1200E.

Neuronii din stratul ascuns au fost implementaţi pentru a îndeplini rolul de captare a unor proprietăţi ale mostrelor de intrare, ale căror caracteristici (nouă la număr) definesc un spaţiu de 9 dimensiuni. Reducerea acestui spaţiu la 6 dimensiuni a contribuit la capacitatea stratului de ieşire de a învăţa segmentarea acestuia în două hiperplanuri. Acest artificiu a dus la o creştere semnificativă a performanţelor sistemului, după cum se prezenta în subcapitolul 5.6.3.1.

5.6.3.6. Algoritmul de învăţare implementat

Neuronii stratului ascuns sunt identici din punct de vedere constructiv cu cei de ieşire în consecinţă funcţionează în mod similar. Aceşti şase neuroni (Figura 5.87) primesc valori post-sinaptice – de la neuronii de intrare – decalate temporal. Am creat două grupe de câte trei neuroni în acest strat şi am alternat valoarea de activare prescrisă acestora conform clasificării date de baza de date WBCD. În mod evident, s-a creat un set de antrenare, utilizând un număr de mostre egal cu 65% din numărul total al elementelor din baza de date alese în mod aleatoriu şi restul elementelor au creat setul de test al RNP. Neuronii din stratul de ieşire au fost antrenaţi în Varianta 2 a RNP, accentuând efectul unei sinapse asupra PM atunci, când cel puţin trei dintre cele şase intrări ale acestor neuroni au fost activate. Valoarea cu care se modifică ponderile sinaptice variază în funcţie de pasul temporal la care soma recepţionează valoarea post-sinaptică corespunzătoare. Astfel, dacă această valorare soseşte chiar în primul ciclu de tact al cadrului temporal, însemnă că o funcţie de activare a acestuia este centrată exact pe valorarea de intrare actuală, deci ponderea trebuie mărită drastic (pas de învăţare iniţial, cu valoarea experimentată a fi de 60, la valori ale ponderilor cuprinse între 0-1200). În cazul în care sinapsa trimite valoarea post-sinaptică într-un ciclu ulterior, valoarea cu care creşte ponderea lui – în cazul în care şi valoarea axonală prescrisă este de activare – cu o valoare dată de formula de mai jos (ponderile se iniţializează cu $\beta_{j\ \max}^{ik} * 0.3$):

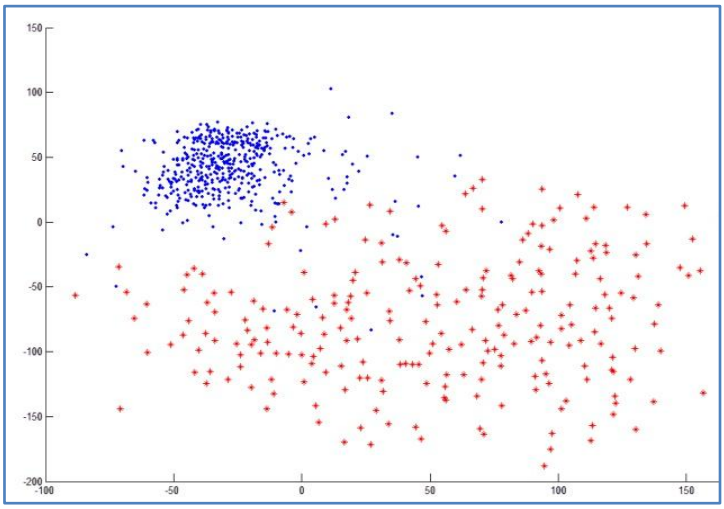


Figura 5.88 Reprezentarea grafică prin proiecţie Sammon 9D→2D a clasificării prescrise a bazei de date WBCD cu 699 mostre

$$\Delta\beta_j^{ik} = \beta_{j\ \max}^{ik} / m \qquad \textit{Ec. 79,}$$

unde $m = [1 \div 14]$ *este număul pasului temporal*

În cazul în care un neuron a emis impuls axonal atunci când valoarea prescrisă nu cerut acest lucru, sinapsa respectivă este slăbită, conform algoritmului prezentat în *Tabelul 12*.

5.6.3.7. Utilizarea proiecţiei lui Sammon pentru vizualizarea mulţimilor multidimensionale

Proiecţia lui Sammon (Sammon Jr., 1969) este o metodă de proiecţie neliniară folosită la cartografierea spaţiilor de mari dimensiuni pe spaţii de dimensiuni mai reduse. Algoritmul cartografiază spaţiul original pe spaţiul proiectat într-un mod în care distanţele dintre obiectele din spaţiul original

sunt conservate în spaţiul proiectat, la fel ca în tehnicile Self-Organizing Map (SOM). Totuşi, experimentele arată că algoritmul lui Sammon are de obicei performanţe mai bune decât orice alt algoritm de cartografiere (Biswas, Jain, & Dubes, 1981).

Vom nota cu d_{ij}^* distanţa dintre un obiect *i* şi un obiect *j* din spaţiul original, şi cu d_{ij} distanţa dintre aceleaşi obiecte în spaţiul proiectat. Algoritmul lui Sammon încearcă să minimizeze următoarea funcţie de eroare, adică deformarea proiecţiei:

$$E = \frac{1}{\sum\sum_{i<j} d_{ij}^*} \sum\sum_{i<j} \frac{\left(d_{ij}^* - d_{ij}\right)^2}{d_{ij}^*} \qquad \textit{Ec. 80}$$

Această ecuaţie poate fi minimizată folosind de exemplu o metodă de gradient cum propune şi Sammon (Sammon Jr., 1969). Deşi proiecţia lui Sammon nu este în principiu orientată spre reţele neuronale, ea poate fi modelată folosind o reţea neuronală, cum se arată în (Mao & Jain, 1995).

5.6.3.1. Rezultate experimentale

În Figura 5.89 este ilustrată clasificarea originală a elementelor conform valorilor prescrise din baza de date. După 100 de cicluri de antrenare a celor 683 de elemente, reţeaua neuronală pulsativă (Varianta 1 – fără strat ascuns) a clasificat elementele după cum se poate vedea în Figura 5.90.

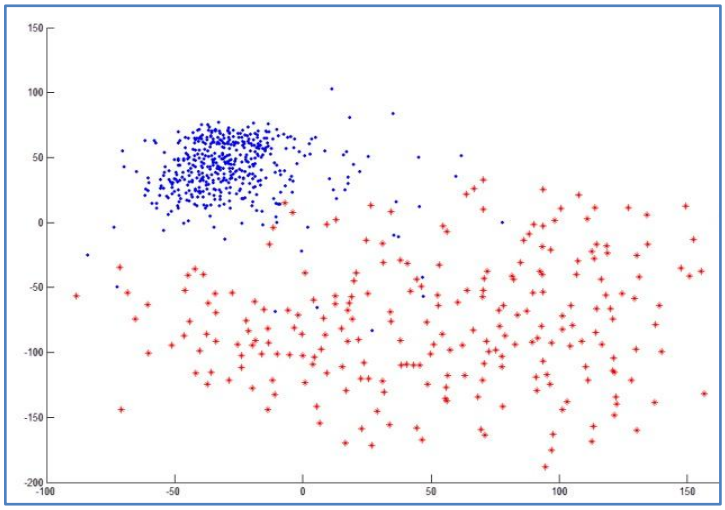


Figura 5.89 Reprezentarea grafică prin proiecţie Sammon 9D → 2D a clasificării prescrise a bazei de date WBCD cu 683 mostre

Punctele colorate cu albastru şi roşu din Figura 5.90 – care afişează rezultatul clasificării WBCD cu RNP Varianta 1 (fără strat ascuns) aparţin celor două clase. Punctele notate cu plus (negru) nu au fost clasificate în nici un grup. Elementele bazei de date pot fi încadrate în 9 dimensiuni dar pentru a putea vizualiza grafic am realizat o proiecţie din 9D în 2D cu ajutorul algoritmului Sammon.

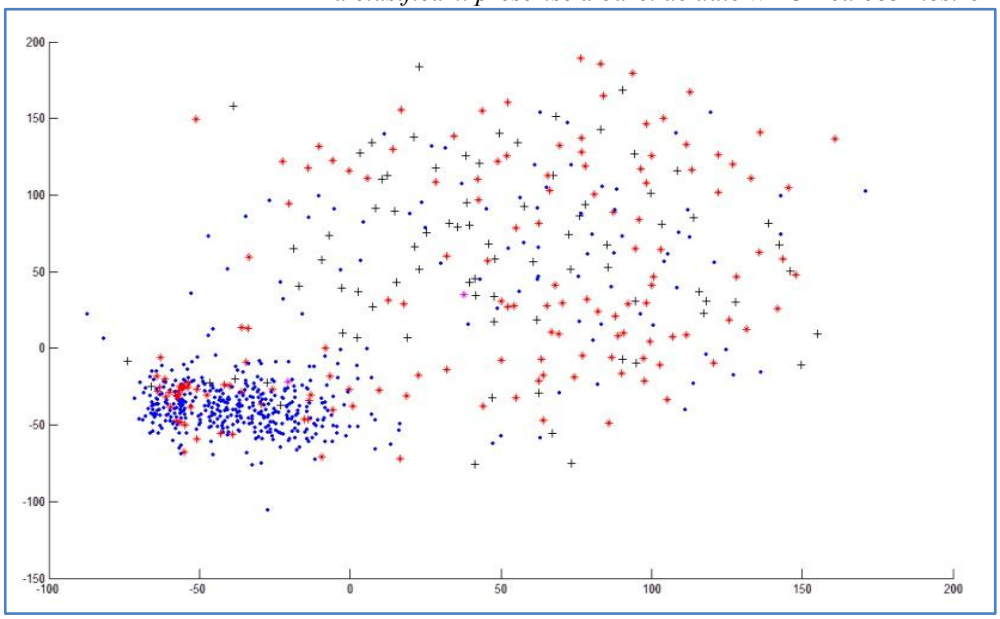


Figura 5.90 Rezultatul clasificării bazei de date WBCD cu RNP fără strat ascuns, implementată hardware, reprezentat grafic prin proiecţie Sammon 9D → 2D

În concluzie mărimea reţelei neuronale în varianta fără strat ascuns este insuficientă pentru a putea învăţa separarea tuturor elementelor din baza de date, lăsând aproximativ 20% dintre elemente neclasificate. Totuşi este important de remarcat, că chiar şi o reţea neuronală pulsativă atât de simplă atinge acest scor apreciabil, lucru greu de imaginat dacă am fi implementat o reţea neuronală clasică de dimensiuni identice, bazată pe modelul perceptron.

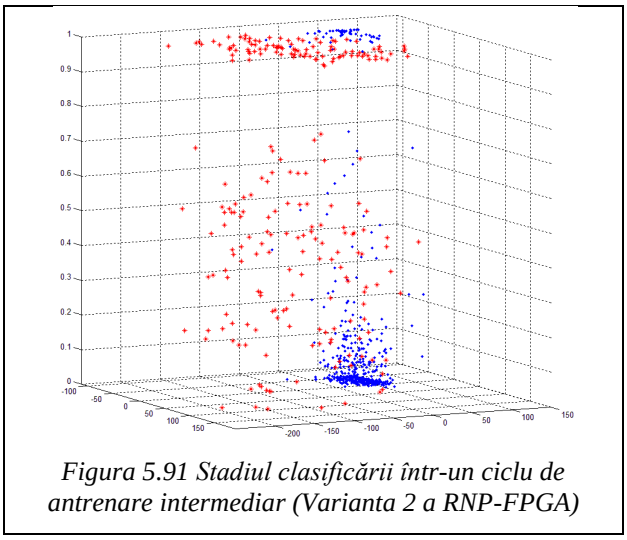


Figura 5.91 Stadiul clasificării într-un ciclu de antrenare intermediar (Varianta 2 a RNP-FPGA)

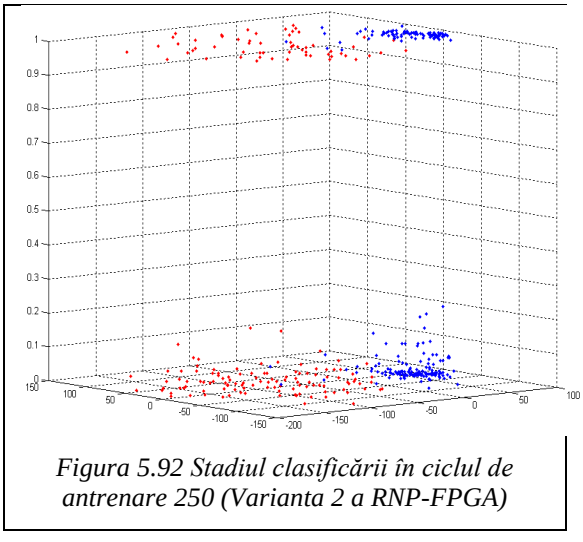
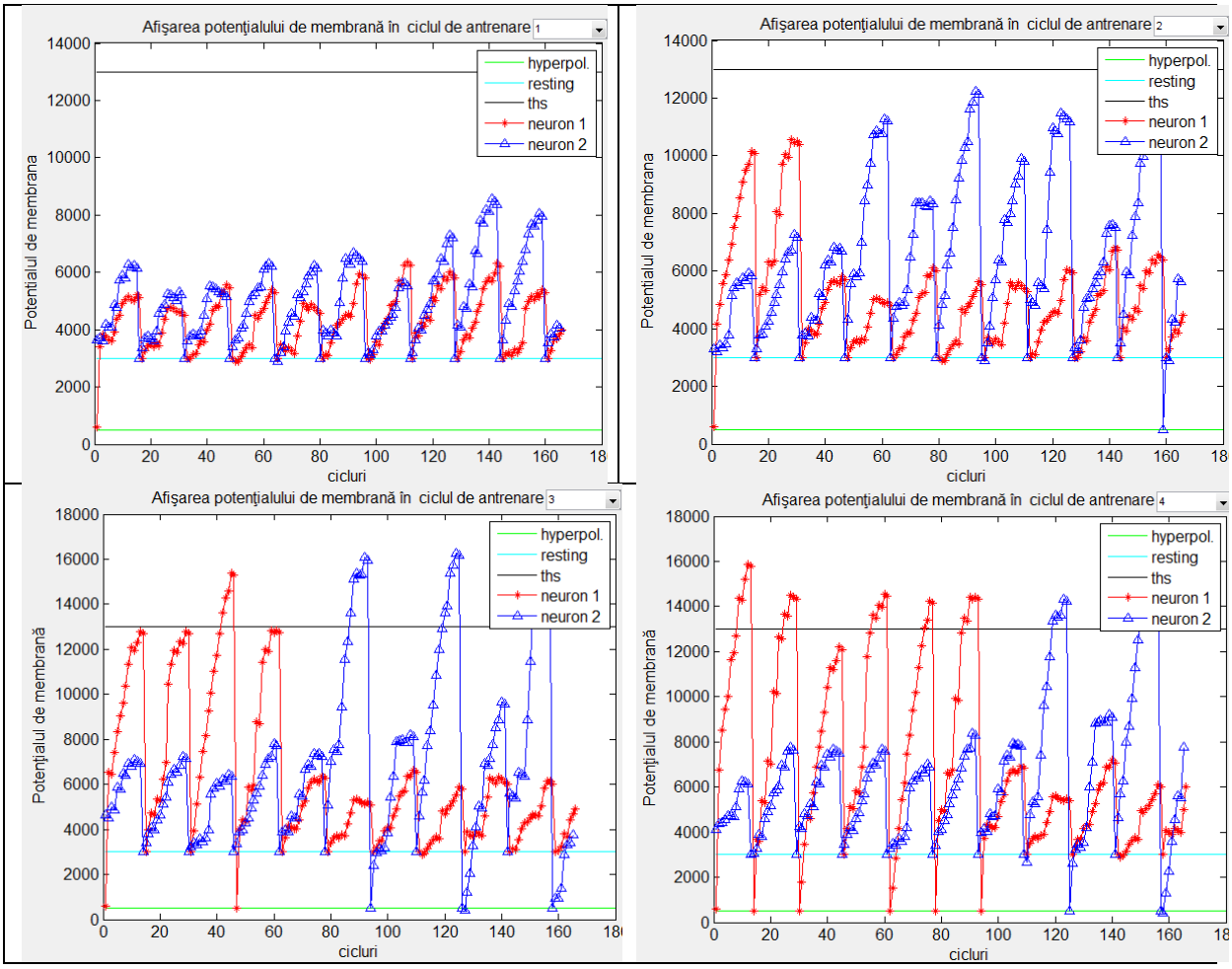
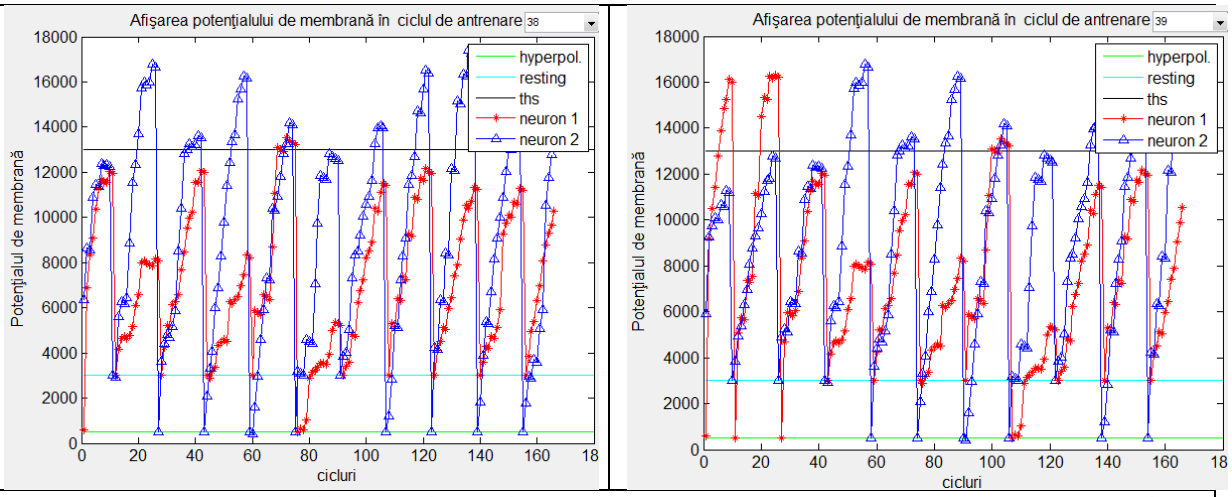


Figura 5.92 Stadiul clasificării în ciclul de antrenare 250 (Varianta 2 a RNP-FPGA)

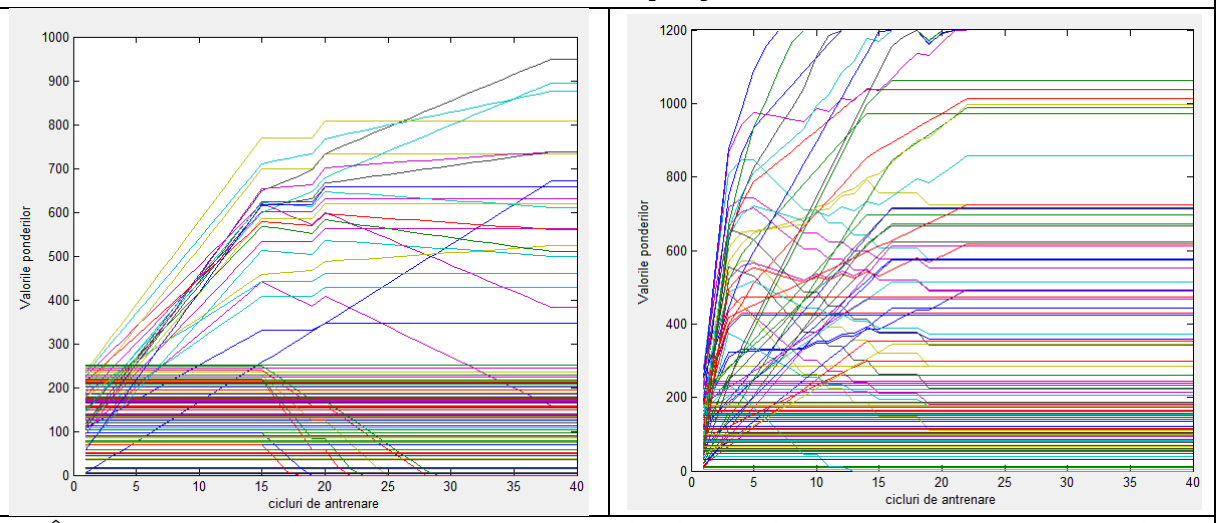
Varianta 2 a RNP, cu strat ascuns, atinge un scor mult bun, clasificând toate elementele date, greşind clasa în proporţie de ~10%. Graficele de mai jos arată – prin proiecţie Sammon 9D → 3D – două măsurători în timpul antrenării, în pasul 100 (Figura 5.91) respectiv 250 (Figura 5.92) dintre cele 300 utilizate. Pa axele X, Z găsim valorile scalate generate de algoritmul Sammon (între -150...150 respectiv -200...10), iar pe axa Y a fost reprezentată procentele de eroare cu care o anumită mostră a fost clasificată malign (roşu) sau benign (albastru).

Tabelul 17 Măsurători în timpul procesului de învăţare asupra variaţiei potenţialelor de membrană respectiv a ponderilor sinaptice





A se nota, că în primele cicluri de învăţare PM nu depăşeşte încă THS, iar în ciclurile ulterioare numai unul dintre cei doi neuroni de ieşire se va active. Se poate observa, de asemenea intrarea neuronilor în starea de hiperpolarizare dup fiecare impuls axonal emis la trecerea PM peste nivelul de prag.



În aceste figuri s-a reprezentat variaţia unui număr redus de ponderi sinaptice (reţeaua neuronală pulsativă implementată hardware conţine mai multe sute de sinapse), observându-se că unele sinapse se vor stabiliza după un număr destul de redus de cicluri de antrenare.

Este important de menţionat, că rezultatele prezentate mai sus sunt obţinute de RNP implementată în hardware în timp real. Procesul de învăţare se desfăşoară mai lent, încetinit şi de funcţiile de comunicare cu calculatorul prin portul serial RS232 al plăcii de dezvoltare FPGA (prin care se trimit datele ce arată funcţionarea şi comportamentul reţelei). După încheierea procesului de învăţare, însă, sistemul neuronal implementat – fapt valabil la toate aplicaţiile prezentate în această teză – este capabil să dea un rezultat într-un interval de timp măsurabil în milisecunde (chiar microsecunde la sistemele implementate complet paralel). Este deci o contribuţie importantă, realizarea unor astfel de sisteme neuronale hardware cu procesare în timp real.

5.6.4. Mediul de dezvoltare utilizat pentru sisteme încorporate

Embedded Development Kit este un pachet software dezvoltat de Xilinx pentru proiectarea sistemelor programabile încorporate. Pachetul include toate instrumentele, documentaţia, şi module de circuite predefinite de care utilizatorul are nevoie pentru a proiecta sisteme care încorporează nuclee de procesor tip hard-core IBM PowerPC™, şi/sau nuclee de procesor tip soft-core Xilinx MicroBlaze™.

Deşi pare complex la o vedere în ansamblu, fluxul de proiectare a sistemului combină pur şi simplu fluxul hardware standard folosit pentru a crea date de configurare a circuitelor FPGA şi fluxul software standard folosit pentru a crea fişiere executabile de procesor ELF. De fapt, exceptând cazul în care resursele de memorie încorporate în circuitul de reconfigurare sunt folosite pentru a stoca imaginea softului, Embedded Development Kit poate fi considerat ca fiind nimic mai mult decât o extensie a instrumentului de generare nuclee CoreGen de la Xilinx.

Primul pas (Figura 5.93) constă în crearea unui „System Netlist" folosind Embedded Development Kit şi instanţierea acestui „netlist" în codul HDL al proiectării. Proiectarea hardware este deci sintetizată, unită şi implementată folosind exact acelaşi flux de proiectare ca cel folosit pentru orice alt nucleu „black box". Deşi se obişnuieşte să se includă o porţiune deja creată a imaginii software în circuitul FPGA folosind blocul RAM, fişierul „Compiled BIT" creat în timpul acestei faze de dezvoltare conţine doar descrierea hardware a sistemului.

Al doilea pas constă în crearea pachetului „Board Support Package" (BSP) folosind Embedded Development Kit (EDK) şi includerea driverelor necesare în codul sursă C al sistemului. Codul sursă este apoi compilat şi legat cu diversele funcţii disponibile în BSP, el fiind identic cu al oricărui alt sistem procesor.

Deoarece sistemul încorporat este construit folosind resursele FPGA, BSP-ul este personalizat pentru acest set particular de periferice incluse în „System Netlist". Spre deosebire de un procesor cu destinaţie generală larg răspândit, sistemele încorporate cu Virtex-II Pro pot include orice combinaţie de periferice produse de Xilinx sau create de utilizatori. Asta înseamnă că fiecare BSP are potenţialul de a fi unic şi astfel EDK are sarcina de a personaliza un set generic de drivere după nevoie pentru a asigura funcţionarea optimă a „sistemului procesor arbitrar". Odată ce setul final de periferice şi structuri de magistrală au fost bine stabilite, fluxurile de hardware şi software pot rula în mod independent. Chiar dacă o parte sau întreaga imagine software este stocată folosind blocul de memorie RAM încorporat în circuitul de reconfigurare, fluxul software nu necesită repornirea fluxului hardware de la zero atunci când se fac schimbări în software. Doar dacă se fac schimbări în „System Netlist"-ul instanţiat atunci hardware-ul trebuie implementat din nou înainte ca o imagine software nouă care se bazează pe schimbările arhitecturale operate să poată fi încărcată şi rulată.

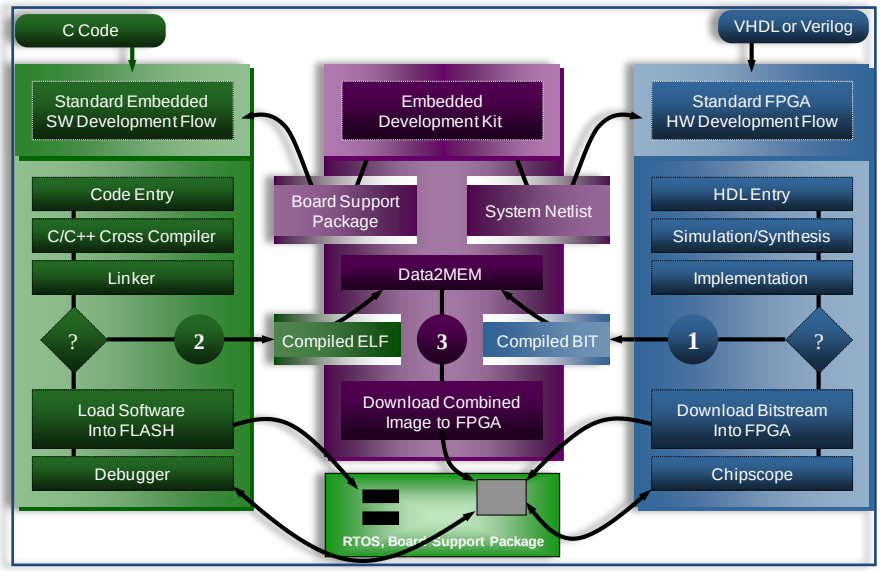


Figura 5.93 Paşii proiectării sistemelor incorporate în mediul Xilinx EDK

Dacă imaginea software este stocată complet extern, configurarea circuitului FPGA şi încărcarea dispozitivului de stocare extern se efectuează la fel ca în cazul unei soluţii tipice bazată pe două chip-uri. Dacă o parte sau întreaga imagine software este stocată folosind blocul de memorie RAM încorporat în circuitul reconfigurabil şi este astfel inclusă în datele de configurare a circuitului FPGA, este nevoie de un pas în plus înainte ca circuitul FPGA să poată fi configurat. EDK pune la dispoziţie un instrument denumit Data2MEM care uneşte secţiunile fişierului „Compiled ELF" (Figura

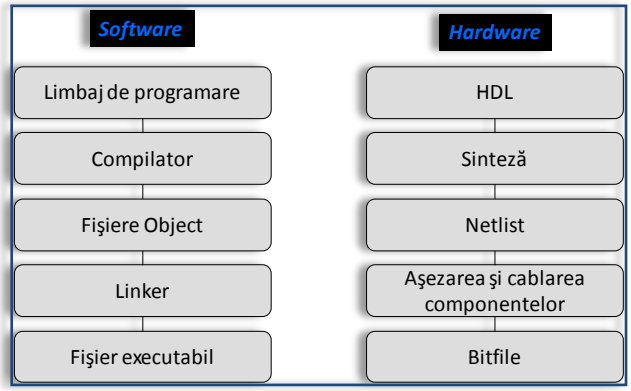


Figura 5.94 Fazele dezvoltării comune hardware-software

5.94) cu secţiunile potrivite din fişierul „Compiled BIT”. Fişierul BIT rezultant este creat de obicei în câteva secunde şi poate fi apoi folosit pentru a configura circuitul FPGA. Când întreaga imagine software este stocată în interiorul circuitului FPGA, doar fişierul BIT este necesar pentru a configura sistemul şi pentru a încărca imaginea software. Dacă doar o porţiune a imaginii software, cum ar fi segmentul de instrucţiuni, este stocată în interiorul circuitului FPGA, atunci se rulează Data2MEM pentru a crea fişierul BIT combinat şi sistemul este gata pentru configurare / încărcare la fel ca orice altă soluţie bazată pe două chip-uri, folosind secţiunile care nu au fost unite din fişierul ELF şi secţiunile din fişierul BIT combinat.

Depanarea softului care rulează pe sistem se realizează la fel ca în cazul oricărui procesor cu destinaţie generală. Se foloseşte GDB sau alt instrument de depanare pentru conectarea la platformă, asigurarea controlului de execuţie al platformei şi chiar pentru încărcarea unor noi imagini când se doreşte acest lucru. Spre deosebire de procesoarele cu destinaţie generală, sistemele fizice pot fi testate folosind module ChipScope. Acest lucru asigură un nivel de transparenţă ridicat al modului de operare al sistemului, care nu poate fi egalat de procesoarele externe.

5.7. Concluzii

Acest capitol a prezentat cele mai recente rezultate ale cercetărilor autorului în domeniul implementărilor hardware de reţele neuronale neuromorfe. Pornind de la rezultatele expuse în capitolul anterior, s-a ajuns la concluzia, că implementarea cu succes a unor aplicaţii complexe bazate pe reţele neuronale pulsative nu este fezabilă în mod complet paralel din cauza necesarului excesiv de resurse reconfigurabile, care nu sunt disponibile în majoritatea circuitelor FPGA de azi. De aceea a fost nevoie de schimbare de direcţie uşoară în continuarea cercetărilor, schimbare ce s-a materializat prin abordarea temei de serializare parţială a acestor implementări hardware. Conceptul de serializare paralelă în acest caz însemnă introducerea unor elemente de execuţie a unor algoritmi pentru a înlocui modulele care ar necesita un număr mare de resurse reconfigurabile, adică utilizarea procesoarelor încorporate de tip soft-core. S-au utilizat două astfel de procesoare (care sunt de fapt un set de programe VHDL sau Verilog ce implementează funcţionalitatea acestora şi pot fi încorporate în orice proiect) şi anume Xilinx PicoBlaze şi Xilinx MicroBlaze. PicoBlaze, cunoscut şi ca KCPSM3, este un microcontroler RISC pe 8 biţi, cu o amprentă de siliciu extrem de redusă, de numai 96 de slice-uri dintr-un FPGA din familia Xilinx Spartan3 (optimizat pentru această familie) ceea ce reprezintă aproximativ 1,25% din totalul de 7680 de astfel de structuri logice disponibile de exemplu în circuitul XC3S1000. Acest necesar scăzut de resurse face ca PicoBlaze să fie candidatul ideal al implementărilor multi-core, fapt ce a fost exploatat în două dintre implementările din acest capitol. Pe de altă parte, nucleul MicroBlaze (pe 32 de biţi) implementează o arhitectură de tip Harvard, cu interfeţe de magistrală separate pentru acces de date şi de instrucţiuni, mult mai complexă, fiind astfel corespunzător în cazul unor aplicaţii cu necesar de putere de calcul mai mare.

Realizările practice prezentate în acest capitol sunt de două feluri, aplicative şi de test reper (benchmark) şi au ca scop determinarea şi ilustrarea capacităţii acestor sisteme inteligente inovative implementate hardware. Toate aplicaţiile din acest capitol pot fi incluse în clasa generală a clasificărilor supervizate, cele aplicative fiind detectarea componentelor de frecvenţă a unui semnal analogic zgomotos (implementat utilizând multiple procesoare încorporate PicoBlaze) respectiv o aplicaţie de recunoaştere de caractere (realizată ca un sistem încorporat având ca procesor central un nucleu MicroBlaze). S-au implementat două teste benchmark clasice şi anume testul de clasificare a setului de date Fisher Iris (multi-core PicoBlaze) şi testul de clasificare a setului de date Wisconsin Breast Cancer Database (MicroBlaze).

În aceste aplicaţii s-a utilizat o metodă originală de codare a intrărilor numerice a reţelelor neuronale implementate prin codificarea acestora în impulsuri decalate temporal, codificare realizată printr-un model neuronal pseudo-RBF. Aceste module de codificare au fost implementate în fiecare aplicaţie utilizând resurse hardware reconfigurabile dedicate, calculele inerente fiind executate astfel în paralel, ajungându-se la un timp de codificare extrem de redus (~15 cicluri de tact la ~100MHz, adică ~150ns).

Pentru a putea evalua performanţele atinse de sistemele implementate, am studiat literatura de specialitate selectând rezultate de reper cu care se pot efectua comparaţii relevante. Există o serie de metode de clasificare des utilizate, dintre care se vor prezenta în continuarea câteva – cu scop ilustrativ – pentru a înlesni analiza rezultatelor expuse ulterior.

Algoritmul K-means clustering: K-medii este unul din cei mai simplii algoritmi de învăţare nesupravegheaţi care rezolvă bine cunoscuta problemă de clustering. Procedeul urmează o cale simplă şi uşoară pentru a clasifica setul de date de intrare într-un număr de K grupe (clusters) (K fixat a priori). ideea de bază este de a defini K centre de greutate, câte unul pentru fiecare grupă. Aceste centre de greutate trebuie fixate raţional, deoarece locaţii diferite conduc la rezultate diferite. Alegerea cea mai bună este să le fixăm cât este posibil mai depărtate unele de altele. Următorul pas este să luăm fiecare element al setului de date şi să-l asociem celui mai apropiat centru de greutate. Prima etapă a grupării se termină când nu mai există elemente negrupate. În acest moment trebuie să calculăm K noi centre ale grupelor rezultate din pasul anterior. Procedeul continuă până în momentul în care poziţiile noilor centre nu se mai modifică semnificativ.

Algoritmul K-medoids: K-medoids este un algoritm de clustering înrudit cu algoritmul *k*-means şi cu algoritmul medoidshift. Ambii algoritmi *k*-means şi *k*-medoids sunt algoritmi de partiţionare (împărţind setul de date în grupuri) şi ambii încearcă să minimizeze eroarea pătratică, distanţa dintre puncte marcate ca făcând parte dintr-un grup (cluster) şi un punct desemnat ca fiind centru al acelui grup. Spre deosebire de algoritmul *k*-means, algorithm *k*-medoids alege elemente ale setului de date ca centre (medoids sau exemplare). *k*-medoid este o tehnică de partiţionare clasică care grupează setul de date de *n* obiecte în *k* grupuri fixate *a priori*. Un instrument folositor pentru determinarea lui *k* este metoda *silhouette*. Este mai robust în condiţii de zgomot şi valori extreme în comparaţie cu *k*-means. Un medoid poate fi definit ca un obiect al unui grup, a cărui diferenţă medie faţă de toate obiectele acelui grup este minimă, adică este punctul situat cel mai central din setul de date respectiv.

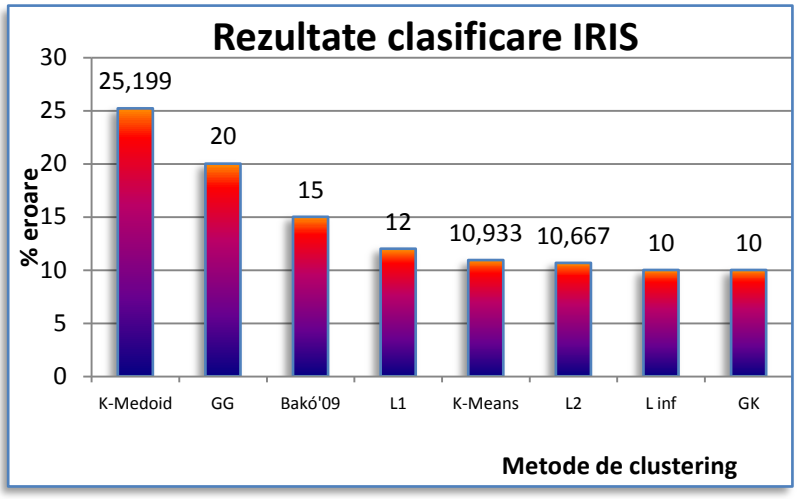


Figura 5.95 Comparaţie a rezultatelor clasificării setului de date Fisher IRIS cu diferite metode similare

Algoritmul L1 şi adâncimea relativă a datelor: În (Jörnsten, Vardi, & Zhang, 2002) s-a introdus adâncimea relativă a datelor, ReD, ca un instrument de validare a grupurilor folosit împreună cu un algoritm K-median exact. Adâncimea datelor L1 a unei observaţii relative la un grup de date indică cât de reprezentativă este această observaţie pentru grupul respectiv. Astfel, pentru date cu etichetă de clasă ne aşteptăm ca adâncimea datelor L1 să fie maximizată faţă de grupul de date corespunzător etichetei de clasă corecte. O regulă simplă de clasificare este să clasificăm observaţiile cu etichete necunoscute după maximul adâncimii

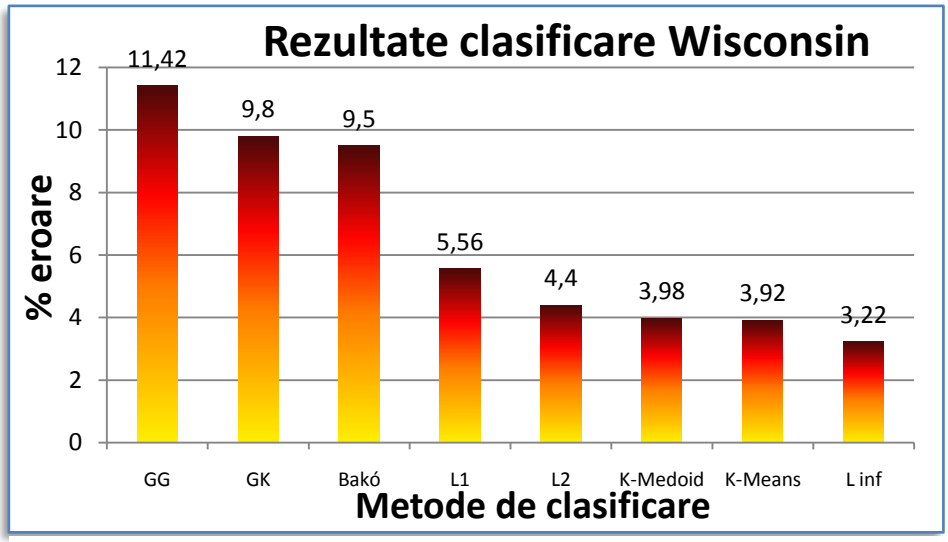


Figura 5.96 Comparaţie a rezultatelor clasificării setului de date WBCD cu diferite metode similare

datelor faţă de datele etichetate. Ne referim la datele etichetate ca fiind setul de antrenare, iar datele neetichetate ca fiind setul de testare.

Graficul comparativ din Figura 5.95 prezintă performanţele implementării benchmark a clasificării setului de date Fisher IRIS, iar Figura 5.96 rezultatele obţinute cu circuitul neuronal pulsativ ce implementează clasificarea setului de date Wisconsin Breast Cancer Database. Deşi rezultatele afişate nu excelează faţă de celelalte metode în mod special, trebuie luat în considerare faptul extrem de important, că toate sistemele hardware realizate şi prezentate în această teză funcţionează cu învăţare on-chip şi procesează calculele necesare în timp real.

Tabelul 18 Comparaţie a mai multor algoritmi de clasificare rulate în software cu algoritmul implementat hardware pentru clasificare seturilor de date benchmark

Setul de date Fisher IRIS						
					precizie	
algoritm	**intrări**	**ascuns**	**ieşiri**	**iteraţii**	**set antrenare**	**set de test**
SpikeProp	50	10	3	1000	96,4%	95,3%
Matlab BP	50	10	3	$2.6*10^6$	97,9%	95,8%
Matlab LM	50	10	3	3750	98,8%	95,9%
$\boldsymbol{RNP_{Bakó}}$	4	0	3	300	85%	83,4%
Setul de date Wisconsin Breast Cancer						
					precizie	
algoritm	**intrări**	**ascuns**	**ieşiri**	**iteraţii**	**set antrenare**	**set de test**
SpikeProp	64	15	2	1500	97.6%	97,3%
Matlab BP	64	15	2	$9.2*10^6$	97.8%	96,2%
Matlab LM	64	15	2	3500	98.1%	96.5%
$\boldsymbol{RNP_{Bakó}}$	9	6	2	250	90,2%	89,5%

Rezultatele prezentate în Tabelul 18 au fost obţinute prin măsurători proprii (Matlab, RNP) respectiv din literatura de specialitate (Bohte, Kok, & La Poutré, Spike-prop: errorbackpropagation in multi-layer networks of spiking neurons, 2000). Ambele seturi de date au fost împărţite în set de antrenare şi set de test. Algoritmul Matlab LM a fost rulat pe 50 de epoci cu 1500 de cicluri fiecare. Simularea software prin algoritmul de propagare înapoi a erorii a fost implementată în mediul Matlab, utilizând rutinele „TRAINLM" (LM) respectiv „TRAINGD" (BP).

Implementările descrise în acest capitol au fost publicate într-un număr de lucrări ştiinţifice (Bakó L. , MACRo 2009, 2009), (Bakó L. , 2009), (Bakó & Brassai, 2009), (Bakó & Székely, Challenges for implementations of delay-coded neuromorphic neural networks on embedded digital hardware, 2009), (Bakó & Székely, 2009), (Bakó L. , Brassai, Székely, & Baczó, 2008), ale autorului, una din ele aducând şi un premiu la o conferinţă internaţională (***"The Best Presenter in Information Technology"*** la Conferinţa The 4th International PhD, DLA Symposium, organizat de University of Pécs, Pollack Mihály Faculty of Engineering, Pécs, Ungaria, 20-21 Octombrie 2008.).

6. CONCLUZII FINALE ŞI CONTRIBUŢII ORIGINALE

6.1. Concluzii finale

Rezultatele cercetărilor neurobiologice au condus la apariţia celei de-a treia generaţii de reţele neuronale artificiale, care se străduiesc să construiască modele cât mai fidele ale reţelelor neuronale naturale. Dintre posibilităţile oferite de aceste modele, am ales cele ce se bazează pe proprietatea pulsativă a neuronilor naturali. Studierea acestor neuroni şi realizarea de modele teoretice şi modele practice cât mai fidele ale acestora, care sunt în acelaşi timp implementabile în mod eficient pe platforme digitale reconfigurabile a reprezentat scopul cercetării efectuate de-a lungul desfăşurării programului de doctorat. Teza de faţă încearcă să surprindă cele mai importante şi relevante teorii şi aplicaţii existente în acest domeniu de cercetare – care stă de fapt la confluenţa ştiinţelor sistemelor de calcul, electronicii digitale şi a biologiei neuronale – descrise de literatura de specialitate. Deşi bazele teoretice au fost puse şi în cazul acestui domeniu cu mulţi ani în urmă (mai ales în ceea ce priveşte modelarea matematică a proceselor neurobiologice), după cum reiese şi din sintezele prezentate în teză, lucrările aplicative cu implementări software şi/sau hardware bazate pe aceste modele sunt foarte recente.

Se poate concluziona, deci, că studiul reţelelor neuronale neuromorfe, şi implementarea de sisteme hardware cu ajutorul acestora este un domeniu de cercetare foarte proaspăt, cu puţine rezultate răsunătoare şi cu multe direcţii de evoluare abordate de diferitele grupuri de oameni de ştiinţă, rezultând posibilităţi foarte importante de a realiza contribuţii originale semnificative. De fapt, prin studierea constantă în ultimii şase ani a lucrărilor apărute în acest sens pe plan mondial, am găsit, că există numai câteva centre de cercetare sunt specializate pe asemenea sisteme, cum ar fi École Polytechnique Fédérale de Lausanne din Elveţia, sau alte grupuri de cercetători din Mexic, Argentina, SUA sau Irlanda.

În acest context autorul acestei teze a încercat o abordare originală în acest domeniu, prin dezvoltarea de modele neuronale proprii, cât mai compatibile cu implementare pe circuite FPGA şi demonstrarea prin aplicaţii proprii – aplicative şi de tip benchmark – capacitatea de calcul şi de asimilare de informaţie extraordinară a acestor sisteme, evidenţiind totodată viteza excepţional de rapidă de care sunt capabile acestea.

Pentru a avea o privire de ansamblu asupra conţinutului tezei de doctorat se prezintă în continuare cele mai importante aspecte abordate de fiecare capitol a lucrării în parte.

Primul capitol *introductiv* prezintă aspecte generale ale domeniului mai larg de cercetare în care se încadrează teza, cea a inteligenţei artificiale şi a maşinilor cu autoinstruire. Fundamentele biologice ale cercetării sunt detaliate de asemenea în acest capitol, urmat de o descriere sumară a uneia din cele mai avansate domenii de aplicare a inteligenţei artificiale, şi anume sistemele de control inteligent, domeniu în care autorul tezei a publicat o serie de lucrări ştiinţifice (Brassai & Bakó, Visual trajectory control of a mobile robot using FPGA implemented neural network, 2008), (Brassai & Bakó, 2008), (Brassai S. , Bakó, Székely, & Dan, 2008), (Brassai, Márton, Dávid, & Bakó, 2008), (Brassai, Gidró, Bakó, & Csernáth, 2008), (Brassai & Bakó, 2007), (Brassai, Bakó, & Dan, 2007), (Brassai, Dávid, & Bakó, 2004).

În prima parte a **Capitolului 2**, intitulat „*Modelarea şi simularea reţelelor neuronale neuromorfe*” s-a discutat modelarea reţelelor neuronale neuromorfe, expunând comportamentul intrinsec temporal al acestora, utilizarea impulsurilor în comunicarea datelor în cadrul acestora şi evidenţiind capacitatea de calcul mai ridicată decât în cazul reţelelor neuronale bazate pe modele clasice.

Studiul neuronilor biologici a rezultat în apariţia mai multor ipoteze pentru codarea valorilor reale în trenuri de impulsuri. Acestea pot apărea ca o codare a ratei impulsurilor, care ia în considerare numai valoarea medie a timpului de apariţie a unui impuls sau ca codarea exactă a momentelor temporale de apariţie a acestor impulsuri.

Când vorbim despre RNP există de fapt o supraabundenţă de diferite modele, de la cele simple de tipul integrează şi activează prin neuroni cu diferite modele sinaptice până la modele comportamentale complete descrise prin ecuaţii diferenţiale. Există câteva studii teoretice (Maass, 1996)

care arată că unele dintre aceste modele sunt mai eficiente decât celelalte, dar aceste studii se limitează doar la un set redus de modele şi scheme de codare. O explicaţie generală a capacităţii de calcul a diferitelor modele neuronale pulsative şi a schemelor de codare conexe reprezintă şi astăzi o temă deschisă cercetărilor actuale.

În orice studiu legat de dinamica unei reţele neuronale pulsative, sunt două probleme specifice, şi anume ce model descrie dinamica pulsaţiei fiecărui neuron respectiv cum sunt conectaţi neuronii. O alegere greşită a modelului sau o conexiune greşită, poate duce la rezultate complet eronate.

În cadrul acestui capitol s-a elaborat prima problemă, comparând neuronii pulsatori. Se vor prezenta diferite modele de neuroni pulsatori precum şi o clasificare a acestora.

În continuare am prezentat cele mai des utilizate metode de antrenare (ne-supervizate şi supervizate) utilizate în învăţarea reţelelor neuronale neuromorfe. Astfel, în această parte a capitolului se regăsesc discuţii asupra realizării învăţării Hebbiene competitive precum şi asupra modalităţilor specifice de aplicare a regulii back-propagation în reţele neuronale pulsative codificate temporal (algoritmul SpikeProp).

După acest studiu amănunţit al acestui domeniu de inteligenţă artificială se prezintă modelul neuronal teoretic ales pentru implementările hardware ce urmează a fi prezentate în capitolele următoare.

A doua contribuţie importantă a acestui capitol este prezentarea generală a strategiilor de simulare software a RNP împreună cu descrierea celor mai importante astfel de medii existente la ora actuală, dar şi a simulatorului neuronal implementare proprie, cu care am testat funcţionarea modelului dezvoltat. Se detaliază încorporarea algoritmilor de învăţare în procesul de simulare software şi se expun rezultatele experimentale obţinute pe această cale, care au stat la baza primelor publicaţii ale autorului (Bakó, Székely, & Brassai, Development of Advanced Neural Models. Software And Hardware Implementation, 2004) (Bakó, Analiza şi simularea sistemelor cu inteligenţă artificială neuromorfă, 2005) (Bakó, Székely, Dávid, & Brassai, 2004) în acest domeniu de cercetare.

Capitolul 3, cu titlul „*Dispozitive FPGA şi implementarea reţelelor neuronale artificiale*” este o sinteză cuprinzătoare a evoluţiei dispozitivelor reconfigurabile, realizând clasificări şi evaluări de performanţe ale acestora, dar descriind şi funcţionarea celor mai importante elemente constructive ale arhitecturilor de circuite FPGA.

Creşterea continuă în densitate (număr porţi/suprafaţă) a sistemelor FPGA a făcut posibilă realizarea unor proiecte de sisteme on-chip cu o complexitate mai mare de un milion de porţi şi RAM intern. De aceea, sistemele FPGA s-au dovedit în prezent a fi o platformă hardware atractivă pentru algoritmii RNA care necesită mult spaţiu. Un alt avantaj al acestui dispozitiv este capacitatea de a combina programabilitatea cu viteza crescută a operaţiilor asociate cu soluţiile hardware paralele.

Arhitecturile bazate pe circuite FPGA pot fi exploatate bine pentru realizarea de reţele neuronale artificiale, deoarece utilizând capacitatea lor de reconfigurare rapidă şi concurentă se poate ajunge la performanţe bune în adaptabilitatea ponderală şi chiar topologică. Cu toate acestea, luând în calcul că prima astfel de aplicaţie (Cox & Blanz, 1992) a apărut numai cu aproximativ un deceniu în urmă, realizarea de reţele neuronale de dimensiuni mari, cu mulţi neuroni în circuite FPGA este o mare provocare chiar şi în zilele noastre. Acest lucru se datorează faptului că majoritatea modelelor neuronale clasice necesită un număr mare de operaţii de multiplicare, ceea ce duce la realizări hardware foarte costisitoare (multe circuite multiplicatoare complexe). În pofida acestei probleme, există tehnologii care exploatează proprietatea de reconfigurabilitate a acestor circuite şi reuşesc să implementeze ieftin şi eficient RNA în acestea.

Orice realizare de reţea neuronală cu FPGA trebuie să se străduiască să utilizeze într-un anumit fel configurabilitatea acestor circuite, implicit optimizarea lor pentru problema propusă. Această proprietate care se poate aplica într-un timp scurt şi de nenumărate ori, este de folos în special în dezvoltarea de *prototipuri şi simulări*. Această flexibilitate permite realizarea rapidă de reţele neuronale cu diferite structuri şi strategii de învăţare, dar şi demonstrarea unor principii prin simulări primordiale.

Procedurile de *creştere a densităţii* realizează o mai bună funcţionalitate raportată la unitatea de arie FPGA, utilizând reconfigurarea. Acest lucru se poate valorifica dacă dispunem de un circuit FPGA capabil de reconfigurare – cel puţin parţială – în timpul funcţionării (run-time/partial reconfigurability).

Primul tip de astfel de procedură este *multiplexarea în timp,* ceea ce înseamnă că simularea RNA este descompusă în mai multe faze secvenţiale, care corespund fiecare unei configurări diferite ale aceluiaşi circuit FPGA. Astfel se poate realiza un sistem în care circuitul este întotdeauna optimizat pentru faza care este în execuţie.

Al doilea tip de procedură pentru creşterea densităţii de implementare se bazează pe execuţia de operaţii cu multiplicatori constanţi şi se numeşte *schimbare dinamică de constante*. Deoarece aceste două proceduri au avut ca scop primordial creşterea densităţii de implementare, în ceea ce priveşte o performanţă superioară, nu putem avea pretenţii faţă de aceste sisteme, numai în cazul în care timpul de reconfigurare a circuitului este neglijabil în comparaţie cu timpul utilizat pentru efectuarea calculelor.

În circuitele FPGA reconfigurabile dinamic, există posibilitatea de implementare a reţelelor neuronale cu *adaptare topologică*. Altfel spus, putem construi RNA care suferă modificări arhitecturale succesive. În faza de învăţare, pe lângă topologia reţelei avem posibilitatea de a acorda şi precizia calculelor, conform funcţiei de criteriu utilizate de algoritmul de învăţare.

Neurohardware-ul digital diferă de asemenea în reprezentarea numerică a operaţiilor aritmetice. Implementările în virgulă flotantă oferă o precizie mai bună, dar mai complexă şi cu necesităţi de suprafaţă mai mare. Aritmetica în virgulă fixă este mai puţin complexă, cu necesităţi de suprafeţe mai reduse, dar oferă o precizie scăzută faţă de implementarea în virgulă flotantă. Aritmetica în virgulă fixă poate fi utilă în cazul în care un algoritm RNA nu necesită precizie ridicată pentru implementarea unor aplicaţii simple.

Al patrulea Capitol al tezei de doctorat, „*Posibilităţi de implementare total paralelă a modelelor neuronale pulsative cu circuite FPGA*" începe prin a prezenta caracteristici generale ale sistemelor hardware analogice şi digitale în ceea ce priveşte implementarea reţelelor neuronale artificiale, aducând motivaţii pro şi contra în fiecare caz. De asemenea sunt prezentate cele mai importante implementări hardware de reţele neuronale din literatura de specialitatea actuală cu accent asupra realizărilor pe diferite platforme digitale. Se trece apoi la domeniul specific temei acestei lucrări, prezentând implementări software şi hardware ale RNP realizate pe calculatoare paralele sau chiar în circuite FPGA.

După această parte de sinteză se prezintă contribuţiile proprii în ceea ce priveşte implementarea complet paralelă de RNP în circuite FPGA. Iniţial se expune implementarea modelului teoretic adaptat realizării hardware, validat prin simulare software în capitolele anterioare. Aceste subcapitole conţin descrieri amănunţite ale proiectării diferitelor subansamble ale acestor reţele neuronale, cum ar fi sinapsele şi modulele de somă sau corp celular. Acestea sunt testate prin simularea funcţionării circuitelor proiectate pentru a implementa funcţionalitatea specifică acestora.

O aplicaţie de test a acestor circuite neuronale realizate reprezintă continuarea logică a acestui capitol, aplicaţia fiind cea clasică de diferenţiere a două forme simple cum ar fi literele ***T*** şi ***H*** sau semnele **+** şi ***X***. Se descrie reţeaua neuronală care implementează în circuit FPGA prin resurse hardware dedicate fiecărui subansamblu al acestuia (implementare complet paralelă), specificând algoritmul de învăţare utilizat şi enumerând rezultatele măsurătorilor experimentale. În cadrul unei analize a performanţelor (*32μs* - timpul de învăţare a reţelei, *3,2μs timp de recunoaştere a formelor de intrare*) acestei RNP se prezintă şi o comparaţie cu o rezolvare pur software (în mediul Matlab) a problemei propuse, rezultând performanţe net superioare ale versiunii hardware, care funcţionează în timp real.

Rezultatele şi contribuţiile personale prezentate în acest capitol au fost publicate într-un număr de lucrări ştiinţifice ale autorului tezei, apărute în publicaţii ale unor conferinţe de specialitate (Bakó & Brassai, Hardware spiking neural networks: parallel implementations using FPGAs, 2006), (Bakó & Brassai, 2005), (Brassai & Bakó, 2007), în reviste de specialitate (Bakó & Brassai, Spiking neural networks built into FPGAs: Fully parallel implementations, 2006) (Bakó, Székely, & Brassai, Development of Advanced Neural Models. Software And Hardware Implementation, 2004), precum şi în rapoartele unor granturi de cercetare în care autorul a fost membru.

Capitolul 5 intitulat „*Implementări parţial paralele ale RNP utilizând microcontrolere încorporate în circuite FPGA*” a prezentat cele mai recente rezultate ale cercetărilor autorului în domeniul implementărilor hardware de reţele neuronale neuromorfe. Pornind de la rezultatele expuse în capitolul anterior, s-a ajuns la concluzia, că implementarea cu succes a unor aplicaţii complexe bazate pe reţele neuronale pulsative nu este fezabilă în mod complet paralel din cauza necesarului excesiv de resurse reconfigurabile, care nu sunt disponibile în majoritatea circuitelor FPGA de azi. De aceea a fost nevoie de schimbare de direcţie uşoară în continuarea cercetărilor, schimbare ce s-a materializat prin abordarea temei de serializare parţială a acestor implementări hardware. Conceptul de serializare paralelă în acest caz însemnă introducerea unor elemente de execuţie a unor algoritmi care ar necesita un număr mare de resurse reconfigurabile, adică utilizarea procesoarelor încorporate de tip soft-core. S-au utilizat două astfel de procesoare (care sunt de fapt un set de programe VHDL sau Verilog ce implementează funcţionalitatea acestora şi pot fi încorporate în orice proiect) şi anume Xilinx PicoBlaze şi Xilinx MicroBlaze. PicoBlaze, cunoscut şi ca KCPSM3, este un microcontroler RISC pe 8 biţi, cu o amprentă de siliciu extrem de redusă, de numai 96 de slice-uri dintr-un FPGA din familia Xilinx Spartan3 (optimizat pentru această familie) ceea ce reprezintă aproximativ 1,25% din totalul de 7680 de astfel de structuri logice disponibile de exemplu în circuitul XC3S1000. Acest necesar scăzut de resurse face ca PicoBlaze să fie candidatul ideal al implementărilor multi-core, fapt ce a fost exploatat în două dintre implementările din acest capitol. Pe de altă parte, nucleul MicroBlaze (pe 32 de biţi) implementează o arhitectură de tip Harvard, cu interfeţe de magistrală separate pentru acces de date şi de instrucţiuni, mult mai complexă, fiind astfel corespunzător în cazul unor aplicaţii cu necesar de putere de calcul mai mare.

Realizările practice prezentate în acest capitol sunt de două feluri, aplicative şi de test reper (benchmark) şi au ca scop determinarea şi ilustrarea capacităţii acestor sisteme inteligente inovative implementate hardware. Toate aplicaţiile din acest capitol pot fi incluse în clasa generală a clasificărilor supervizate, cele aplicative fiind detectarea componentelor de frecvenţă a unui semnal analogic zgomotos (implementat utilizând multiple procesoare încorporate PicoBlaze) respectiv o aplicaţie de recunoaştere de caractere (realizată ca un sistem încorporat având ca procesor central un nucleu MicroBlaze). S-au implementat două teste benchmark clasice şi anume testul de clasificare a setului de date Fisher Iris (multi-core PicoBlaze) şi testul de clasificare a setului de date Wisconsin Breast Cancer Database (MicroBlaze).

În fiecare dintre aceste aplicaţii s-a utilizat o metodă originală de codificare a intrărilor numerice a reţelelor neuronale implementate prin transformarea acestora în impulsuri decalate temporal, codificare realizată printr-un model neuronal pseudo-RBF. Aceste module de codificare au fost implementate în fiecare aplicaţie utilizând resurse hardware reconfigurabile dedicate, calculele inerente fiind executate astfel în paralel, ajungându-se la un timp de codificare extrem de redus (~15 cicluri de tact la ~100MHz, adică ~150ns).

Graficele comparative din finalul acestui capitol prezintă performanţele implementării benchmark a clasificării setului de date Fisher IRIS respectiv a circuitului neuronal pulsativ ce implementează clasificarea setului de date Wisconsin Breast Cancer Database. Deşi rezultatele afişate nu excelează faţă de celelalte metode în mod special, trebuie luat în considerare faptul extrem de important, că toate sistemele hardware realizate şi prezentate în această teză funcţionează cu învăţare on-chip şi procesează calculele necesare în timp real.

Rezultatele prezentate în Tabelul 18 au fost obţinute prin măsurători proprii (Matlab, RNP) respectiv din literatura de specialitate (Bohte, Kok, & La Poutré, Spike-prop: errorbackpropagation in multi-layer networks of spiking neurons, 2000). Ambele seturi de date au fost împărţite în set de antrenare şi set de test. Algoritmul Matlab LM a fost rulat pe 50 de epoci cu 1500 de cicluri fiecare. Simularea software prin algoritmul de propagare înapoi a erorii a fost implementată în mediul Matlab, utilizând rutinele „TRAINLM” (LM) respectiv „TRAINGD” (BP).

Implementările descrise în acest capitol au fost publicate într-un număr de lucrări ştiinţifice (Bakó L. , MACRo 2009, 2009), (Bakó L. , 2009), (Bakó & Brassai, 2009), (Bakó & Székely, Challenges for implementations of delay-coded neuromorphic neural networks on embedded digital hardware, 2009), (Bakó & Székely, 2009), (Bakó L. , Brassai, Székely, & Baczó, 2008), ale autorului.

6.2. Contribuţii originale

În acest ultim subcapitol al acestei lucrări se vor trece în revistă contribuţiile originale ale tezei de doctorat.

- Am realizat un studiu de sinteză privind aspecte generale ale domeniului mai larg de cercetare în care se încadrează teza, cea a inteligenţei artificiale şi a maşinilor cu autoinstruire. Fundamentele biologice ale cercetării sunt detaliate şi se descrie sumar una din cele mai avansate domenii de aplicare a inteligenţei artificiale, şi anume sistemele de control inteligent.
- Am realizat un studiu cu privire la alegerea optimă a modelelor pentru reţele neuronale artificiale neuromorfe.
- Un model neuronal pulsativ propriu a fost dezvoltat care se pretează în mod intrinsec implementărilor hardware digitale.
- S-a efectuat o evaluare a celor mai importante reguli de învăţare nesupervizate aplicabile în implementări hardware de reţele neuronale precum şi a singurei metode supervizate descrise în literatura de specialitate, SpikeProp, care s-a considerat a nu fi fezabilă din acelaşi punct de vedere.
- Sinteza realizată asupra simulatoarelor neuronale ale reţelelor bazate pe modele pulsative prezintă strategiile ce trebuie abordate în astfel de sisteme, metode pentru simulări sinaptice rapide precum şi motivele pro şi contra în alegerea tipului de algoritm ales (sincron – bazat pe tact sau asincron – bazat pe eveniment). Această sinteză înglobează şi câte o prezentare generală a celor mai semnificative medii de simulare software de reţele neuronale pulsative.
- Am realizat un simulator software propriu bazat pe un pachet de funcţii open-source (rulate pe sistem de operare Linux) pentru validarea modelului neuronal dezvoltat. S-au prezentat ideile de bază ale acestui program, algoritmul simulării, metoda de învăţare bazată pe valori de prag utilizată dar şi rezultatele experimentale obţinute cu acest sistem.
- S-a realizat un studiu cu privire la implementarea reţelelor neuronale artificiale pe sisteme FPGA:
  - s-au prezentat diferite moduri de abordare de cuplare şi configurări pentru FPGA
  - s-a prezentat maparea diferiţilor algoritmi de reţele neuronale pe sisteme FPGA
  - s-au sintetizat modurile de implementare a funcţiilor de activare,
  - s-au prezentat diferite aritmetici de reprezentare a datelor.
- Într-un subcapitol s-au discutat avantajele şi dezavantajele diferitelor platforme de implementare hardware ale reţelelor neuronale, prezentând o serie de astfel de implementări existente.
- S-au expus separat şi exemple de implementări hardware şi software a reţelelor neuronale pulsative.
- Implementarea cu circuit FPGA a modelului neuronal propriu dezvoltat, validat prin simulare software este de asemenea o contribuţie importantă a tezei. Se descriu în detaliu procesele de proiectare premergătoare şi cele de implementare pe circuit FPGA. Fiecare componentă a fost testată în simulator hardware, urmat de asamblarea primului circuit digital propriu ce realizează un neuron pulsativ.
- Pe baza acestui concept dezvoltat de realizare a unui neuron neuromorf digital s-a prezentat o primă aplicaţie de test (diferenţeierea unor modele de intrare simple, de ex. ***T*** şi ***H***) realizată cu o RNP simplă, implementată complet paralel în FPGA.
- Utilizând rezultatele obţinute cu RNP implementată complet paralel s-a realizat o comparaţie cu o soluţie pur software (rulat pe PC) a aceleiaşi probleme.
- Am dezvoltat un concept de realizare a unor circuite digitale pentru codificarea valorilor de intrare a RNP în impulsuri decalate temporal, pentru a putea fi interpretate şi prelucrate de aceste reţele neuronale speciale. Procedeul este foarte flexibil, putând fi aplicat la o gamă largă de aplicaţii cu spaţii de intrare de diferite dimensiuni (spaţii definite de plaja valorilor de intrare).
- Am realizat un studiu asupra fezabilităţii implementărilor parţial paralele în hardware digital a reţelelor neuronale pulsative prin introducerea utilizării procesoarelor încorporate de tip soft-core.

- În cadrul realizărilor practice parţial paralele am implementat o RNP pentru detectarea componentelor de frecvenţă a unui semnal analogic perturbat de zgomot pe baza spectrului acestui semnal prezentat la intrările acestei reţele neuronale, realizare cu some neuronale implementate prin procesoare încorporate PicoBlaze.
- Prima aplicaţie benchmark implementată a fost cea de clasificare a setului de date Fisher IRIS. Ca şi în cazul aplicaţiei de detectare a frecvenţelor, sistemul neuronal rezultant este unul de tip multi-core, deoarece conţine patru instanţe ale procesorului PicoBlaze.
- Trecând la procesoare încorporate soft-core cu putere de calcul mai ridicată (MicroBlaze) am implementat o aplicaţie de recunoaştere de caractere, prezentând rezultatele experimentale obţinute.
- A doua aplicaţie benchmark este clasificarea setului de date Wisconsin Breast Cancer, care a fost realizată fuzionând circuitul de codificare a valorilor de intrare în impulsuri decalate temporal cu procesorul MicroBlaze, rezultând un sistem foarte rapid şi cu performanţe în clasificare comparabile cu cele ce se regăsesc în literatura de specialitate.
- Am realizat o comparaţie între performanţele măsurate prin experimente ale sistemelor neuronale pulsative realizate cu utilizarea procesoarelor încorporate şi a performanţelor unor metode cunoscute.

BIBLIOGRAFIE

1. Abbott, L. F., & Nelson, S. B. (2000). Synaptic plasticity: taming the beast. *Nature Neuroscience , 3* (Suppl), 1178-1283.
2. Abramson, D., Smith, K., Logothetis, P., & Duke, D. (1998). FPGA based implementation of a Hopfield neural network for solving constraint satisfaction problem. *Proceedings of 24th euromicro workshop on computational intelligence*, (pg. 688-693). Sweden.
3. Agis, R., Díaz, J., Ros, E., Carrillo, R., & Ortigosa, E. (2006). Event-Driven Simulation Engine for Spiking Neural Networks on a Chip. În *Book Chapter of Book Series Lecture Notes in Computer Science* (Vol. 3985). SpringerLink.
4. Alippi, C., & Storti-Gajani, G. (1991). Simple Approximation of sigmoidal functions: Realistic designs of digital neural networks capable of learning. *IEEE International Symposium on Circuits and Systems*, (pg. 1505-1508). Singapore.
5. Amdahl, G. (1967). Validity of single-processor approach to achieving large-scale computing capability. *Proceedings of AFIPS Conference*, (pg. 483-485). Reston, VA.
6. Asanovic, e. a. (2006, Dec 18). The Landscape of Parallel Computing Research: A View from Berkeley. *Tech. Report No. UCB/EECS-2006-183* .
7. Ashenden, P. (1990). *The VHDL Cookbook (First Edition).* Adelaide, South Australia: Dept. Computer Science, University of Adelaide.
8. Badoual, M., Rudolph, M., Piwkowska, Z., Destexhe, A., & Bal, T. (2005). High discharge variability in neurons driven by current noise. *Neurocomputing , 65*, 493-498.
9. **Bakó, L. (2005)**. *Analiza şi simularea sistemelor cu inteligenţă artificială neuromorfă.* Brasov, Ro: Referat Nr. 1, din cadrul programului de doctorat, Conducător ştiinţific: Prof.dr.ing. Iuliu Székely.
10. **Bakó, L. (2009)**. Partially Serialized Computation in Networks of Pulse-based Artificial Neurons. *1st International Conference on Recent Achievements in Mechatronics, Automation, Computer Science and Robotics, MACRo 2009* (p. 19). Tîrgu Mures, Romania: Sapientia University, Department Of Electrical Engineering, Department Of Mechanical Engineering.
11. **Bakó, L. (2009)**. Real-time clustering of datasets with hardware embedded neuromorphic neural networks. *HiBi 2009 (High performance computational systems Biology) Workshop.* Trento, Italy: COSBi (Microsoft Research - University of Trento Centre for Computational and Systems Biology), In press.
12. **Bakó, L.**, & Brassai, S. **(2006)**. Spiking neural networks built into FPGAs: Fully parallel implementations. *WSEAS Transactions on Circuits and Systems , 5* (3), 346-353.
13. **Bakó, L.**, & Brassai, S. T. **(2008)**. Embedded neural controllers based on spiking neuron models. În M. Iványi (Ed.), *Fourth International PhD, DLA Symposium, University of Pécs, Hungary, Pollack Mihály Faculty of Engineering.* Komló, Hungary: Rotari Press.
14. **Bakó, L.**, & Brassai, S. T. **(2009)**. Embedded neural controllers based on spiking neuron models. *Pollack Periodica, An International Journal for Engineering and Information Sciences , 4* (3), xx-xx.
15. **Bakó, L.**, & Brassai, S. T. **(2004)**. Fejlett neuronmodellek szimulációja és megvalósítása. *Számokt 2004 – Cluj-Napoca* (pg. 98-107). Cluj-Napoca: EMT.
16. **Bakó, L.**, & Brassai, S. T. **(2006)**. Hardware spiking neural networks: parallel implementations using FPGAs. *Proceedings of the 8th WSEAS Int. Conference on Automatic Control, Modeling and Simulation, Prague, Czech Republic*, (pg. 261-266). Prague.
17. **Bakó, L.**, & Brassai, S. T. **(2005)**. Természetazonos felépítésű mesterséges neurális hálózatok hardware megvalósítása. *Számokt 2005 Kolozsvár* (pg. 219-230). Cluj-Napoca: EMT.
18. **Bakó, L.**, & Székely, I. **(2009)**. Challenges for implementations of delay-coded neuromorphic neural networks on embedded digital hardware. *2nd INCF Congress of Neuroinformatics.* Pilsen, Czech Republic: In press.
19. **Bakó, L.**, & Székely, I. **(2009)**. Challenges for implementations of delay-coded neuromorphic neural networks on embedded digital hardware. *Frontiers in Neuroinformatics. Conference Abstract: 2nd INCF Congress of Neuroinformatics* .
20. **Bakó, L.**, Brassai, S., & Székely, I. **(2006)**. Fully Parallel Implementation of Spiking Neural Networks on FPGA. *Proceedings of the 10th International Conference on Optimisation of Electrical and Electronic Equipment (OPTIM '06). III*, pg. 135-142. Braşov (Moeciu): Transilvania University Press.
21. **Bakó, L.,** Brassai, S., Székely, I., & Baczó, M. **(2008)**. Hardware Implementation of Delay-coded Spiking-RBF Neural Network for Unsupervised Clustering. *Proceedings of the 11th International Conference on Optimisation of Electrical and Electronic Equipment (OPTIM '08)* (pg. 51-56). Braşov: Transilvania University of Braşov.
22. **Bakó, L.**, Székely, I., & Brassai, S. **(2004)**. Development of Advanced Neural Models. Software And Hardware Implementation. *Transaction on Electronics and communication, Scientific buletin of the „Politehnica" University of Timişoara* , 214-219.
23. **Bakó, L.**, Székely, I., Dávid, L., & Brassai, S. T. **(2004)**. Simulation of Spiking Neural Networks. *Proceedings of the 9th International Conference on Optimisation of Electrical and Electronic Equipment (OPTIM '04)* (pg. 179-184). Braşov: Trasilvania University Press.
24. Banitt, Y., Martin, K. A., & Segev, I. (2005). Depressed responses of facilitatory synapses. *Journal of Neurophysiology , 94*, 865-870.
25. Basterretxea, K., & Tarela, J. (2004). Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons. *IEEE proceedings on Circuits, devices and Systems.* Athens.
26. Beeman, D. (2005). GENESIS Modeling Tutorial. *Brains, Minds, and Media* .
27. Beiu, V., Peperstraete, J., & Vandewalle, J. (1994). Close approximation of sigmoid functions by sum of steps for VLSI implementation of Neural networks. *The Scientific Annals, section: Informatics , 40* (1).
28. Bernard, C., Ge, Y. C., Stockley, E., Willis, J. B., & Wheal, H. V. (1994). Synaptic integration of NMDA and non-NMDA receptors in large neuronal network models solved by means of differential equations. *Biological Cybernetics , 70* (3), 267-73.

29. Beuchat, J., Haenni, J., & Sanhez, E. (1998). Hardware Reconfigurable Neural Networks. *Proc. of the 5th Reconfigurable Architectures Workshop*. Orlando, Florida, USA.
30. Bhalla, U. S. (2004). Signaling in small subcellular volumes: II. Stochastic and diffusion effects on synaptic network properties. *Biophysical Journal , 87*, 745-753.
31. Bhalla, U. S., & Iyengar, R. (1999). Emergent properties of networks of biological signaling pathways. *Science , 283*, 381-387.
32. Bishop, C. (1995). *Neural networks for pattern recognition.* Oxford: Clarendon Press.
33. Biswas, G., Jain, A. K., & Dubes, R. C. (1981). Evaluation of projection algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence , 3* (6), 701-708.
34. Blum, K., & Abbott, L. (1996). A model of spatial map formation in the hippocampus of the rat. *Neural Computation* (8), 85–93.
35. Bohte, S., Kok, J., & La Poutr, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* (48), 17–37.
36. Bohte, S., Kok, J., & La Poutré, H. (2000). Spike-prop: errorbackpropagation in multi-layer networks of spiking neurons. În V. M. (Ed.), *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*, (pg. 419-425).
37. Bohte, S., Kok, J., & La Poutre, H. (2000). Unsupervised classification in a layered network of spiking neurons. *Proceedings of IJCNN'2000, IV*, pg. 279–285.
38. Bohte, S., Kok, J., & La Poutré, H. (2002). Unsupervised classification in a layered RBF network of spiking neurons. *IEEE Trans. Neural Networks* , 426-435.
39. Bondalpati, K. (2001, August). Modelling and mapping for dynamically reconfigurable hybrid architecture. *PhD thesis* . Computer Engineering Department, University of South California.
40. Booij, O. (2004). Temporal pattern classification using spiking neural networks. *Master's thesis* . University of Amsterdam.
41. Booij, O., & Nguyen, H. (2005). A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters , 95* (6), 552-558.
42. Bower, J. M. (1995). Reverse engineering the nervous system:An in vivo, in vitro, and in computo approach to understanding the mammalian olfactory system. În S. F. Zornetzer, J. L. Davis, & C. Lau (Ed.), *An introduction to neural and electronic networks, second edn* (pg. 3–28). New York: Academic Press.
43. Bower, J. M., & Beeman, D. (1998). *The book of GENESIS: Exploring realistic neural models with the General Neural Simulation System, second edn*. New York: Springer.
44. Bower, J., & Beeman, D. (1998). *The Book of GENESIS: Exploring Realistic Neural Models with the General SImulation System, Second ed*. New York: Springer.
45. Brassai, S. T., & **Bakó, L. (2008)**. Mobilis robot mesterséges idegsejt hálóval való szabályzása pályakövetési feladatokra. *Enelko-SzámOkt 2008* (pg. 116-121). Sumuleu-Ciuc: EMT Cluj-Napoca.
46. Brassai, S. T., & **Bakó, L. (2008)**. Visual trajectory control of a mobile robot using FPGA implemented neural network. În M. Iványi (Ed.), *Fourth International PhD, DLA Symposium, University of Pécs, Hungary, Pollack Mihály Faculty of Engineering*. Komló, Hungary: Rotari Press.
47. Brassai, S. T., **Bakó, L.**, & Dan, S. **(2007).** FPGA Parallel Implementation of CMAC Type Neural Network with on Chip Learning. *SACI 2007* (pg. 111-115). Budapest: Budapest Tech, Hungary.
48. Brassai, S. T., Dávid, L., & **Bakó, L. (2004)**. Hardware Implementation of CMAC based artificial network with process control application. *Transaction on Electronics and communication, Scientific buletin of the „Politehnica" University of Timişoara* , 209-213.
49. Brassai, S. T., Gidró, L., **Bakó, L.**, & Csernáth, G. **(2008)**. Practical Implementation of an Embedded Intelligent Control System. *Proceedings of the International Symposium for Design and Technology of Electronic Packages, Faculty Of Electrical Engineering And Computer Science*. Predeal: Department Of Electronics And Computers, "Transilvania" University Of Brasov and Center For Technological Electronics And Interconnection Techniques "Politehnica" University Bucharest.
50. Brassai, S. T., Márton, L., Dávid, L., & **Bakó, L. (2008)**. Hardware implemented neural network based mobile robot control. *Proceedings of the International Symposium for Design and Technology of Electronic Packages, Faculty Of Electrical Engineering And Computer Science*. Predeal: Department Of Electronics And Computers, "Transilvania" University Of Brasov and Center For Technological Electronics And Interconnection Techniques "Politehnica" University Bucharest.
51. Brassai, S., & **Bakó, L. (2007)**. Hardware Implementation of CMAC Type Neural Network on FPGA for Command Surface Approximation. *Acta Polytechnica Hungarica , 4* (3), 5-16.
52. Brassai, S., **Bakó, L.**, Székely, I., & Dan, S. **(2008)**. Neural Control Based on RBF Network implemented on FPGA. *Proceedings of the 11th International Conference on Optimisation of Electrical and Electronic Equipment (OPTIM '08)* (pg. 41-46). Braşov: Transilvania University of Braşov.
53. Brown, D., Francis, R., Rose, J., & Vranesic, Z. (1993). *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, USA.
54. Brown, R. (1988). Calendar queues: A fast 0(1) priority queue implementation for the simulation event set problem. *Journal of Communication ACM , 31* (10), 1220-1227.
55. Buonomano, D., & Merzenich, M. (1999). A neural network model of temporal code generation and position-invariant pattern recognition. *Neural Computation , 1* (11), 103–116.
56. Carnevale, N. T., & Hines, M. L. (2006). *The NEURON book*. Cambridge: Cambridge University Press.
57. Chakrabarti, S., Roy, S., & Soundalgekar, M. (2001). Fast and Accurate Text Classification via Multiple Linear Discriminant Projections. *Proceedings of International Conference on Very Large Data Bases* (pg. 658-669). Hong Kong: Morgan Kaufmann.
58. Cîrstea, M., Dinu, A., & Nicula, D. (2001). *A practical guide to VHDL design.* Bucharest: Editura Technică.
59. Claverol, E., Brown, A., & Chad, J. (2002). Discrete simulation of large aggregates of neurons. *Neurocomputing , 47*, 277-297.

60. Cloutier, J., Pigeon, S., & Boyer, F. (1996). VIP:An FPGA-based Processor for image processing and neural networks. *5th international conference on Microelectronics for neural networks and fuzzy systems*, (pg. 330-336). Switzerland.
61. Compton, K., & Hauck, S. (2000). An Introduction to Reconfigurable Computing. *Technical Report* . Evanston, IL, USA: Northwestern University, Department of Electrical and Computer Engineering.
62. Connollly, C., Marian, I., & Reilly, R. (2003). Approaches to efficient simulation with spiking neural networks. În *WSPC*.
63. Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2001). *Introduction to algorithms, second edn.* Cambridge: MIT Press.
64. Cox, C., & Blanz, E. (1992). GangLion – a fast field-programmable gate array implementation of a connectionist classifier. *IEEE Journal of Solid-State Circuits , 23* (1), 288-299.
65. Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems* (2), 303-314.
66. Day, M., Carr, D. B., Ulrich, S., Ilijic, E., Tkatch, T., & Surmeier, D. J. (2005). Dendritic excitability of mouse frontal cortex pyramidal neurons is shaped by the interaction among HCN, Kir2, and k(leak) channels. *Journal of Neuroscience , 25*, 8776-8787.
67. Dayan, P., & Abbott, L. (2001). *Theoretical Neuroscience: Computational and Methematical Modeling of Neural Systems.* Cambridge, MA: MIT Press.
68. de Garis, H., & Korkin, M. (2002). The cam-brain machine (cbm) an FPGA based hardware tool which evolves a 1000 neuron net circuit module in seconds and updates a 75 million neuron articial brain for real time robot control. *Neurocomputing , 42* (1-4).
69. de Garis, H., Gers, F., & Korkin, M. (1997). A simplied cellular automatabased neuron model. *Artificial Evolution Conference (AE97).* Nimes, France.
70. De Schutter, E., & Bower, J. M. (1994). An active membrane model of the cerebellar Purkinje cell. I. Simulation of current clamps in slice. *Journal of Neurophysiology , 71*, 375-400.
71. Dehon, A. (2000). The Density Advantage of Configurable Computing. *IEEE Computer , 5* (33), 41-49.
72. Dehon, A. (2000). The Density Advantage of Configurable Computing. *IEEE Computer , 33* (5), 41-49.
73. Delorme, A., & Thorpe, S. (2001). Face identification using one spike per neuron: resistance to image degradations. *Neural Networks* , 795–804.
74. Delorme, A., & Thorpe, S. (2003). SpikeNET:an event-driven simulation package for modelling large networks of spiking neurons. *Network:Comput. Neural Systems , 4* (14), 613–628.
75. Deneve, S. (2005). Bayesian inference in spiking neurons. (S. K. Lawrence, Y. Weiss, & L. Bottou, Ed.) *Advances in Neural Information Processing Systems* (17), 353–360.
76. Destexhe, A., & Sejnowski, T. J. (2001). *Thalamocortical assemblies.* New York: Oxford University Press.
77. Destexhe, A., Mainen, Z., & Sejnowski, T. (1994). An efficient method for computing synaptic conductances based on a kinetic model of receptor binding. *Neural Computation , 6*, 14-18.
78. Destexhe, A., Mainen, Z., & Sejnowski, T. (1994). Synthesis of models for excitable membranes, synaptic transmission and neuromodulation using a common kinetic formalism. *Journal of Computational Neuroscience , 1*, 195–230.
79. Diesmann, M., & Gewaltig, M. -O. (2002). NEST: An environment for neural systems simulations. În T. Plesser, & V. Macho (Ed.), *Forschung und wisschenschaftliches Rechnen, Beitrage zum Heinz-Billing-Preis 2001, Volume 58 of GWDG-Bericht* (pg. 43–70). Gottingen: Ges. fur Wiss. Datenverarbeitung.
80. Duda, R. O., & Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* John Wiley & Sons.
81. Eldredge, J. (1994). FPGA Density Enhancement of a Neural Network Through Run-Time Reconfiguration. *Master's thesis* . Department of Electrical and Computer Engineering, Brigham Young University.
82. Eldredge, J., & Hutchings, B. (1994). Density enhancement of a neural network using FPGAs and run-time reconfiguration. *Proceedings of IEEE Workshop on Field-Programmable Custom Computing Machines*, (pg. 180-188).
83. Eldredge, J., & Hutchings, B. (1994). RRANN: A Hardware Implementation of the Backpropagation Algorithm Using Reconfigurable FPGAs. *IEEE World Conference on Computational Intelligence.* Orlando, FL.
84. Ermentrout, B. (2004). *Simulating, analyzing, and animating dynamical systems: A guide to XPPAUT for researchers and students.* Philadelphia: SIAM.
85. Ferrucci, A. (1994). ACME: A Field-Programmable Gate Array Implementation of a Self-Adapting and Scalable Connectionist Network. University of California, Santa Cruz.
86. Ferscha, A. (1996). Parallel and distributed simulation of discrete event systems. În A. Y. Zomaya (Ed.), *Parallel and Distributed Computing Handbook* (pg. 1003-1041). New York: McGraw-Hill.
87. Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annual Eugenics , 7*, 179-188.
88. Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence , 1*.
89. Földiák, P. (1990). Forming sparse representations by local anti-hebbian learning. *Biological Cybernetics , 64*, 165-170.
90. Frank, G., & Hartmann, G. (1995). An Artificial Neural Network Accelerator for Puls-Coded Model-Neurons. *Proceedings of ICNN'95.* Perth, Australia.
91. Fujimoto, R. M. (2000). *Parallel and distributed simulation systems.* New York: Wiley.
92. Gerstner, W. (2001). Spiking neurons. În W. Maass, & C. Bishop (Ed.), *Pulsed Neural Networks.* Cambridge, MA: MIT Press.
93. Gerstner, W., & Kistler, W. M. (2002). Mathematical formulations of hebbian learning. *Biological Cybernetics , 87*, 404–415.
94. Gerstner, W., & Kistler, W. M. (2002). *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge: Cambridge University Press.
95. Gerstner, W., Kempter, R., van Hemmen, J., & Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature* (383), 76–78.
96. Ghosh-Dastidar, S., & Adeli, H. (2007). Improved spiking neural networks for EEG classification and epilepsy and seizure detection. *Integrated Computer-Aided Engineering* (14), 1–26.
97. Gibson, J., Belerlein, M., & Connors, B. (1999). Two networks of electrically coupled inhibitory neurons in neocortex. *Nature , 402*, 75–79.

98. Girau, B., & Tisserand, A. (1996). OnLine Arithmetic-Based Reprogrammable Hardware Implementation of Multilayer Perceptron Back-Propagation. *Proceedings of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems* (pg. 168-175). IEEE Computer Society Press.
99. Giugliano, M. (2000). Synthesis of generalized algorithms for the fast computation of synaptic conductances with markov kinetic models in large network simulations. *Neural Computation , 12*, 903–931.
100. Giugliano, M., Bove, M., & Grattarola, M. (1999). Fast calculation of short-term depressing synaptic conductances. *Neural Computation , 11*, 1413–1426.
101. Glackin, B., McGinnity, T., Maguire, L., Wu, Q., & Belatreche, A. (2005). A Novel Approach for the Implementation of Large Scale Spiking Neural Networks on FPGA Hardware. In *Computational Intelligence and Bioinspired Systems* (Vol. 3512). SpringerLink.
102. Grill, W. M., Simmons, A. M., Cooper, S. E., Miocinovic, S., Montgomery, E. B., Baker, K. B., și alții. (2005). Temporal excitation properties of parenthesias evoked by thalamic microstimulation. *Clinical Neurophysiology , 116*, 1227–1234.
103. Gustafson, J. (1988). Reevaluating Amdahl's Law. *CACM , 5* (31), 532-533.
104. Gütig, R., & Sompolinsky, H. (2006). The tempotron: A neuron that learns spike timing-based decisions. *Nature Neuroscience , 9*, 420–428.
105. Hammarlund, P., & Ekeberg, Ö. (1998). Large neural network simulations on multiple hardware platforms. *Journal of Computational Neuroscience , 5*, 443-459.
106. Hauck, S. (1995). Multi-FPGA Systems. *PhD Thesis* . University of Washington.
107. Hereld, M., Stevens, R. L., Teller, J., & van Drongelen, W. (2005). Large neural simulations on large parallel computers. *International Journal of Bioelectromagnetism , 7*, 44-46.
108. Hikawa, H. (1999). Frequency-based multiplayer neural network with on-chip learning and enhanced neuron characteristics. *IEEE Trans. Neural Netw. , 10*, 545–553.
109. Hines, M. (1984). Efficient computation of branched nerve equations. *International Journal of Bio-Medical Computing , 15*, 69-76.
110. Hines, M., & Carnevale, N. T. (1997). The neuron simulation environment. *Neural Computation , 9*, 1179-1209.
111. Hines, M., & Carnevale, N. (1997). The NEURON simulation environment. *Neural Computation* (9), 1179-1209.
112. Hirsch, M., & Smale, S. (1974). *Differential equations, dynamical systems, and linear algebra. Pure and applied mathematics.* New York: Academic Press.
113. Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology , 117* (4), 500-544.
114. Hodgkin, A., & Huxley, A. (1952). A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve. *Journal of Physiology* (117), 500-544.
115. Hopfield, J. (1995). Pattern recognition computation using action potencial timing for stimulus representation. *Nature* (376), 33-36.
116. Houweling, A. R., Bazhenov, M., Timofeev, I., Steriade, M., & Sejnowski, T. J. (2005). Homeostatic synaptic plasticity can explain post-traumatic epileptogenesis in chronically isolated neocortex. *Cerebral Cortex , 15*, 834-845.
117. Hutchings, B., & Writhlin, M. (1995). Implementation approaches for reconfigurable logic applications. *5th international workshop on FPGA.* Oxford, England.
118. Iannella, N., & Kindermann, L. (2005). Finding iterative roots with a spiking neural network. *Information Processing Letters , 6* (95), 545–551.
119. Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks , 14*, 1569-1572.
120. Izhikevich, E. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks , 14*, 1569–1572.
121. Izhikevich, E. (2004). Which model to use for cortical spiking neurons? *IEEE Transactions on neural networks , 15* (5), 1063-1070.
122. Izhikevich, E., Desai, N., Walcott, E., & Hoppensteadt, F. (2003). Bursts as a unit of neural information: Selective communication via resonance. *Trends in Neuroscience , 26*, 161–167.
123. Jahnke, A., Roth, U., & Klar, H. (1995). Towards Efficient Hardware for Spike-Processing Neural Networks. *Proc. World Congress on Neural Networks*, (pg. 460-463).
124. Jahnke, A., Roth, U., & Schoenauer, T. (1998). Digital simulation of spiking neural networks. În W. Maass, & C. M. Bishop (Ed.), *Pulsed neural networks.* Cambridge: MIT Press.
125. James-Roxby, P., & Blodget, B. (2000). Adapting constant multipliers in a neural network implementation. *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, (pg. 335-336).
126. Kanold, P. O., & Manis, P. B. (2005). Encoding the timing of inhibitory inputs. *Journal of Neurophysiology , 93*, 2887-2897.
127. Knoblauch, A. (2005). Neural associative memory for brain modeling and information retrieval. *Information Processing Letters , 95* (6), 537-544.
128. Koch, C. (1999). *Biophysics of Computation: Information Processing in Single Neurons.* Oxford, UK: Oxford University Press.
129. Kohn, J., & Wörgötter, F. (1998). Employing the Z-transform to optimize the calculation of the synaptic conductance of NMDA and other synaptic channels in network simulations. *Neural Computation , 10*, 1639–1651.
130. Lazarro, J., & Wawrzynek, J. (1993). Silicon Auditory Processors as Computer Peripherals. *NIPS* (5), 820-827.
131. Lee, G., & Farhat, N. H. (2001). The double queue method: A numerical method for integrate-and-fire neuron networks. *Neural Networks , 14*, 921-932.
132. Lysaght, J., Stockwood, J., Law, & Girma, D. (1994). Artificial Neural Network Implementation on a Fine-Grained FPGA. În R. Hartenstein, & M. Servit (Ed.), *Field-Programmable Logic: Architectures, Synthesis and Applications. 4th International Workshop on Field-Programmable Logic and Applications* (pg. 421-431). Prague, Czech Republic: Springer-Verlag.
133. Lysaght, P., Stockwood, J., Law, J., & Girma, D. (1994). Artificial Neural Network Implementation on a Fine-Grained FPGA. În R. Hartenstein, & M. Servit (Ed.), *Field-Programmable Logic: Architectures, Synthesis and*

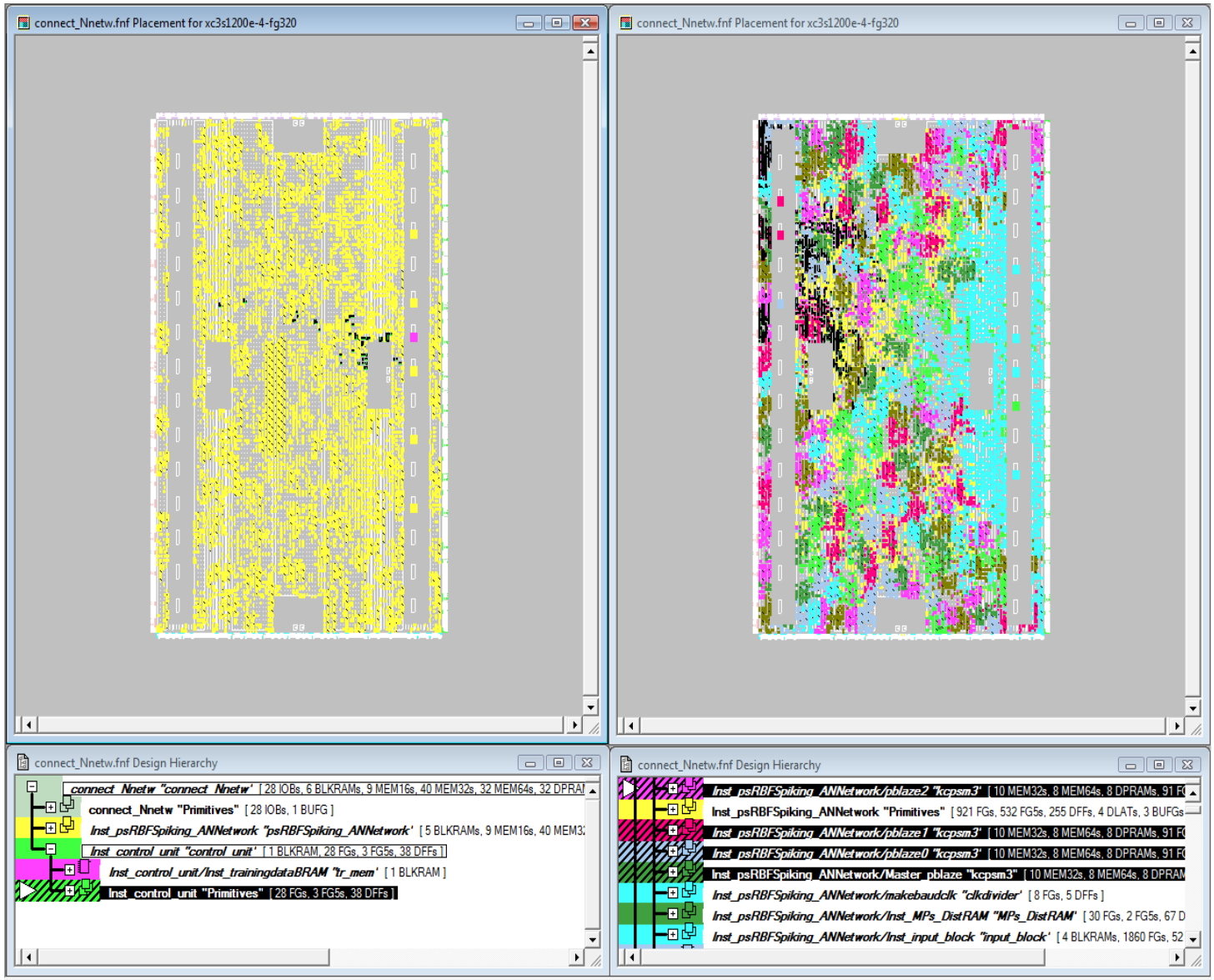
Applications. 4th International Workshop on Field-Programmable Logic and Applications (pg. 421-431). Prague, Czech Republic: Springer-Verlag.

134. Lytton, W. W. (1996). Optimizing synaptic conductance calculation for network simulations. *Neural Computation , 8*, 501-509.
135. Maas, W. (1998). A simple model for neural computation with firing rates and firing correlations. *Network: Computation in Neural Systems* (9), 1-17.
136. Maas, W. (1997). Fast sigmoidal networks via spiking neurons. *Neural Computation , 2* (9), 279–304.
137. Maas, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks , 9* (10), 1659-1671.
138. Maas, W. (1997). Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. În M. Mozer, M. Jordan, & T. Petsche, *Advances in Neural Information Processing Systems* (Vol. 9, pg. 211–217). Cambridge: MIT Press.
139. Maas, W., & Bishop, C. M. (1999). *Pulsed Neural networks.* Cambridge, MA: MIT Press.
140. Maas, W., & Natschläger, T. (1997). Networks of spiking neurons can emulate arbitrary Hopfield nets in temporal coding. *Network: Computation in Neural Systems , 4* (8), 355-372.
141. Maass, W. (1996). Lower bounds for the computational power of networks of spiking neurons. *Neural Computation , 1* (8), 1–40.
142. Maass, W., Natschlager, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation , 14* (11), 2531–2560.
143. Makino, T. (2003). A Discrete-Event Neural Network Simulator for General Neuron Models. *Neural Comput. & Applic.* (11), 210-223.
144. Mangasarian, O., & Wolberg, W. (1990). Cancer diagnosis via linear programming. *SIAM News , 23* (5), 1-18.
145. Mantas, C., Puche, J., & Mantas, J. (2006). Extraction of similarity based fuzzy rules from artificial neural networks. *International Journal of Approximate Reasoning* (43), 202–221.
146. Mao, J., & Jain, A. K. (1995). Artificial neural networks for feature extraction and multivariate data projection. *IEEE Transactions on Neural Networks , 6* (2), 296-317.
147. Marcelo, H., & Martin, A. (1994). Reconfigurable Hardware Accelerator for Back-Propagation Connectionist Classifiers. University of California, Santa Cruz.
148. Marchesi, M., Orlandi, G., Piazza, F., & Uncini, A. (1993). Fast neural networks without multipliers. *IEEE Trans. Neural Netw. , 4* (1), 53–62.
149. Markaki, M., Orphanoudakis, S., & Poirazi, P. (2005). Modelling reduced excitability in aged CA1 neurons as a calcium-dependent process. *Neurocomputing , 65*, 305-314.
150. Mattia, M., & Del Giudice, P. (2000). Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses. *Neural Computation , 10* (12), 2305–2329.
151. Mattia, M., & Del Giudice, P. (2000). Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses. *Neural Computation , 12*, 2305-2329.
152. Mayrhofer, R., Affenzeller, M., Prähofer, H., Hfer, G., & Fried, A. (2002). Devs simulation of spiking neural networks. În *Proceedings of Cybernetics and Systems (EMCSR)* (Vol. 2, pg. 573-578). Austrian Society for Cybernetic Studies.
153. McCartor, H. (1991). Back Propagation Implementation on the Adaptive Solutions CNAPS Neurocomputer Chip. În R. L. al. (Ed.), *Proc. of NIPS-3, "Advances in Neural Information Processing Systems 3"* (pg. 1028-1031). Morgan Kaufmann Pub.
154. McCulloch, W., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* (5), 115-133.
155. Mehrtash, N., Jung, D., Hellmich, H., Schoenauer, T., Lu, V., & Klar, H. (2003). Synaptic Plasticity in Spiking Neural Networks (SP2INN): A System Approach. *IEEE Transactions on Neural Networks , 14* (5).
156. Migliore, M., Hines, M. L., & Shepherd, G. M. (2005). The role of distal dendritic gap junctions in synchronization of mitral cell axonal output. *Journal of Computational Neuroscience , 18*, 151-161.
157. Moffitt, M. A., & McIntyre, C. C. (2005). Model-based analysis of cortical recording with silicon microelectrodes. *Clinical Neurophysiology , 116*, 2240-2250.
158. Molz, R., Molz, P., Moraes, F., Torres, L., & Robert, M. (2000). Codesign of fully parallel neural network for a classification problem. *5th world multi-conference on systematics, cybernatics and informatics.* Orlando, Florida.
159. Morrison, A., Mehring, C., Geisel, T., Aertsen, A., & Diesmann, M. (2005). Advancing the boundaries of high connectivity network simulation with distributed computing. *Neural Computation , 17*, 1776-1801.
160. Motomura, M., Aimoto, Y., Shibayama, A., Yabe, Y., & Yamashina, M. (1998). An Embedded DRAM-FPGA Chip with Instantaneous Logic Reconfiguration. *Proc. of IEEE Symp. on FPGAs for Custom Computing Machines* (pg. 264-266). IEEE Computer Society.
161. Muller, J. (1997). *Elementary Functions: Algorithms and Implementation.* Boston: Birkhauser.
162. Murtagh, P., & Tsoi, A. (1992). Implementation issues of sigmoid function and its derivative for VLSI neural networks.
163. Myers, D., & Hutchison, R. (1989). Efficient implementation of piecewise linear activation function for digital VLSI neural networks. *Electronic Letters* , 1662-1663.
164. Nagami, K., Oguri, K., Shiozawa, T., Ito, H., & Konishi, R. (1998). Plastic Cell Architecture: Towards Reconfigurable Computing for General Purpose. *Proc. of IEEE Symp. on FPGAs for Custom Computing Machines* (pg. 68-77). Los Alamitos, CA: IEEE Computer Society.
165. National Science Foundation. (2007). *UC Irvine Machine Learning Repository*. Preluat de pe http://archive.ics.uci.edu/ml/
166. Natschläger, T., & Ruf, B. (1998). Spatial and temporal pattern analysis via spiking neurons. *Network: Computational Neural Systems , 3* (9), 319–332.

167. Nenadic, Z., Ghosh, B. K., & Ulinski, P. (2003). Propagating waves in visual cortex: A large scale model of turtle visual cortex. *Journal of Computational Neuroscience , 14*, 161-184.
168. Nichols, K. (2003, December). A Reconfigurable Computing Architecture For Implementing Artificial Neural Networks On FPGA. *A Thesis Presented to The Faculty of Graduate Studies of The University of Guelph* .
169. Nordstrom, T. (1995, March). Highly Parallel Computers for Articial Neural Networks. *Ph.d.thesis (1995:162 f)* . Sweden: Division of Computer Science and Engineering, Lulea University of Technology.
170. O'Dwyer, C., & Richardson, D. (2005). Spiking neural nets with symbolic internal state. *Information Processing Letters - Special issue on applications of spiking neural networks , 95* (6), 529 - 536.
171. Oldfield, J., & Dorf, R. (1995). *Field Programmable Arrays: Reconfigurable Logic for Rapid Prototyping and Implementation of Digital Systems.* John Wiley and Sons.
172. Olshausen, B. A., & Field, D. J. (2005). How close are we to understanding V1? *Neural Computation , 17*, 1665-1699.
173. Omondi, A. (1994). *Computer Arithmetic Systems: Algorithms, Architecture, and Implementations.* Prentice-Hall, UK.
174. *Open source data visualization and analysis for novice and experts*. (2005). Preluat de pe http://www.ailab.si/orange/doc/datasets/breast-cancer-wisconsin-cont.tab
175. Pandya, V. (2005, August). A Handel-C Implementation Of The Back-Propagation Algorithm On Field Programmable Gate Arrays. *A Thesis Presented to The Faculty of Graduate Studies Of The University of Guelph* .
176. Pearson, M., Gilhespy, I., Gurney, K., Melhuish, C., Mitchinson, B., Nibouche, M., și alții. (2005). A Real-Time, FPGA Based, Biologically Plausible Neural Network Processor. În *Book Chapter of Book Series Lecture Notes in Computer Science, Category: Issues in Hardware Implementation* (Vol. 3697). SpringerLink.
177. Pérez-Uribe, A. (1999, Oct.). Structure-Adaptable Digital Neural Networks. *Ph.d. thesis* . Logic Systems Laboratory, Computer Science Department, Swiss Federal Institute of Technology-Lausanne.
178. Pérez-Uribe, A., & Sanchez, E. (1996). FPGA Implementation of an Adaptable Size Neural Network. *VI international conference on ANN ICANN' 96*, (pg. 383-1388). Bochum, Germany.
179. Pérez-Uribe, A., & Sanchez, E. (1996). FPGA implementation of an Adaptible-Size Neural Network. În C. Von der Malsburg, W. Von Seelen, J. Vorbrüggen, & B. Sendhoff (Ed.), *Proceedings of the International Confernce on Neural Networks (ICANN96). 1112 of Lecture Notes in Computer Science*, pg. 383-388. Heidelberg: Springer-Verlag.
180. Pérez-Uribe, A., & Sanchez, E. (1997). Speeding-up adaptive heuristic critic learning with fpga-based unsupervised clustering. *Proceedings of the IEEE International Conference on Evolutionary Computation ICEC'97*, (pg. 685-689). Indianapolis.
181. Poormann, M., Witkowski, U., Kalte, H., & Ruckert, U. (2002). Implementation of ANN on a reconfigurable hardware accelerator. *Euromicro workshop on parallel, distributed and network based processing*, (pg. 243-250). Spain.
182. Prescott, S. A., & De Koninck, Y. (2005). Integration time in a subset of spinal lamina I neurons is lengthened by sodium and calcium currents acting synergistically to prolong subthreshold depolarization. *Journal of Neuroscience , 25*, 4743-4754.
183. Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T. (1993). *Numerical recipes in C: The art of scientific computing.* Cambridge: Cambridge University Press.
184. Rao, R. P. (2005). Hierarchical bayesian inference in networks of spiking neurons. (S. K. Lawrence, Y. Weiss, & L. Bottou, Ed.) *Advances in Neural Information Processing Systems* (17), 1113–1120.
185. Reutimann, J., Guigliano, M., & Fusi, S. (2003). Event-driven simulation of spiking neurons with stochastic dynamics. *Neural Computation* (15), 811-830.
186. Rochel, O., & Martinez, D. (2003). An event-driven framework for the simulation of networks of spiking neurons. În *Proceedings of the 11th European Symposium on Artificial Neural Networks - ESANN'2003* (pg. 295-300). Bruges.
187. Ros, E., Carrillo, R., Ortigosa, E., Barbour, B., & Agis, R. (2005). Event-driven Simulation Scheme for Spiking Neural Models based on Characterization Look-up Tables. *Neural Computation* .
188. Roth, U., Jahnke, A., & Klar, H. (1995). Hardware Requirements for Spike-Processing Neural Networks. *Proc. IWANN95*, (pg. 720-727).
189. Rotter, S., & Diesmann, M. (1999). Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. *Biological Cybernetics , 81*, 381-402.
190. Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature* (325), 533-536.
191. Saghatelyan, A., Roux, P., Migliore, M., Rochefort, C., Desmaisons, D., Charneau, P., și alții. (2005). Activity-dependent adjustments of the inhibitory network in the olfactory bulb following early postnatal deprivation. *Neuron , 46*, 103-116.
192. Sahin, I., Gloster, C., & Doss, C. (2000). Feasibility of floating point arithmetic in reconfigurable computing systems. *MAPLD International Conference.* DC, USA.
193. Sammon Jr., J. W. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers , C-18*, 401-409.
194. Sammut, K., & Jones, S. (1991). Implementing non-linear activation functions in neural network emulators. *Electronic Letters , 27* (12).
195. Sanchez-Montanez, M. (2001). Strategies for the optimization of large scale networks of integrate and fire neurons. În J. Mira, & A. Prieto (Ed.), *IWANN, Volume 2084/2001 of Lecture Notes in Computer Science.* NewYork: Springer-Verlag.
196. Schoenauer, T., Atasoy, S., Mehrtash, N., & Klar, H. (2002). NeuroPipe-Chip: A Digital Neuro-Processor for Spiking Neural Networks. *IEEE Trans. Neural Networks , 13* (1), 205-213.
197. Schrauwen, B., & Van Campenhout, J. (2006). Parallel hardware implementation of a broad class of spiking neurons using serial arithmetic. *ESANN'2006 proceedings - European Symposium on Artificial Neural Networks.* Bruges, Belgium: ISBN 2-930307-06-4.
198. Simard, P., & Graf, H. (1993). Backpropagation without Multiplication. *Advances in Neural Information Processing Systems* , 232-239.

199. Skrbek, M. (1999). Fast Neural Network Implementation. *Neural Network World , 9* (5), 375-391.
200. Sloot, A., Kaandorp, J. A., Hoekstra, G., & Overeinder, B. J. (1999). Distributed simulation with cellular automata: Architecture and applications. În J. Pavelka, G. Tel, & M. Bartosek (Ed.), *SOFSEM'99, LNCS* (pg. 203-248). Berlin: Springer-Verlag.
201. Snippe, H. (1996). Parameter extraction from population codes: A critical assessment. *Neural Computation* , 511-529.
202. Sobieski, J., & Storaasli, O. (2004, Oct). Computing at the Speed of Thought. *Aerospace America* , 35-38.
203. Song, S., & Abbott, L. (2001). Column and Map Development and Cortical Re-Mapping Through Spike-Timing Dependent Plasticity. *Neuron* (32), 339-350.
204. Song, S., Miller, K. D., & Abbott, L. F. (2000). Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience , 3*, 919–926.
205. Song, S., Miller, K., & Abbott, L. (2000). Competitive Hebbian learning though spike-timing dependent synaptic plasticity. *Nature Neuroscince* (3), 919-926.
206. Strenski, D. (2007, Jan 12). FPGA Floating Point Performance. *HPCWire* .
207. Stricanne, B., & Bower, J. M. (1998). A network model of the somatosensory system cerebellum, exploring recovery from peripheral lesions at various developmental stages in rats (abstract). *Society of Neuroscience Abstracts , 24*, 669.
208. Tang, K., & Kak, S. (2002). Fast Classification Networks for Signal Processing. *Circuits Systems Signal Processing , 2* (21), 207-224.
209. Trappenberg, T. P. (2002). *Fundamentals of Computational Neuroscience.* Oxford: Oxford University Press.
210. Triesch, J. (2007). Synergies between intrinsic and synaptic plasticity mechanisms. *Neural Computation* (19), 885–909.
211. Upegui, A., Peñ a-Reyes, C., & Sánchez, E. (2004). A Hardware Implementation of a Network of Functional Spiking Neurons with Hebbian Learning. În *Book Chapter of Book Series Lecture Notes in Computer Science* (Vol. 3141). SpringerLink.
212. Upegui, A., Peña-Reyes, C., & Sánchez, E. (2003). A methodology for evolving spiking neuralnetwork topologies on line using partial dynamic reconfiguration. *Proceedings of II- International Congress on Computational Intelligence (CIIC'03).* Medellín,Colombia.
213. van Emde Boas, P., Kaas, R., & Zijlstra, E. (1976). Design and implementation of an efficient priority queue. *Theory of Computing Systems , 10*, 99-127.
214. Verstraeten, D., Schrauwen, B., Stroobandt, D., & Van Campenhout, J. (2005). Isolated word recognition with the Liquid State Machine: a case study. *Information Processing Letters* (95), 521-528.
215. Vitko, I., Chen, Y. C., Arias, J. M., Shen, Y., Wu, X. R., & Perez-Reyes, E. (2005). Functional characterization and neuronal modeling of the effects of childhood absence epilepsy variants of CACNA1H, a T-type calcium channel. *Journal of Neuroscience , 25*, 4844-4855.
216. Watts, L. (1994). Event-driven simulation of networks of spiking neurons. În *Advances in neural information processing systems* (pg. 927-934).
217. Wilson, M. A., & Bower, J. M. (1989). The simulation of largescale neural networks. În C. Koch, & I. Segev (Ed.), *Methods in neuronal modeling: From synapses to networks* (pg. 291-333). Cambridge: MIT Press.
218. Wolf, J. A., Moyer, J. T., Lazarewicz, M. T., Contreras, D., Benoit-Marand, M., O'Donnell, P., şi alţii. (2005). NMDA/AMPA ratio impacts state transitions and entrainment to oscillations. *Journal of Neuroscience , 25*, 9080-9095.
219. Xilinx, DS099. (2006, April 26). *Spartan-3 FPGA Family:Complete Data Sheet* .
220. Xilinx, XAPP463. (2005, March 1). Using Block RAM in Spartan-3 Generation FPGAs. *XAPP463 (v2.0)* .
221. Xilinx, XAPP464. (2005, March 1). Using Look-Up Tables as Distributed RAM in Spartan-3 Generation FPGAs. *XAPP464 (v2.0)* .
222. Xin, J., & Embrechts, M. (2001). Supervised learning with spiking neural networks. *Proceedings of International Joint Conference on Neural Networks (IJCNN)*, (pg. 1772-1777).
223. Zeigler, B. P., & Vahie, S. (1993). DEVS formalism and methodology: Unity of conception/diversity of application. *Proceedings of the 1993 Winter Simulation Conference*, (pg. 573-579). Los Angeles, December 12–15.
224. Zeigler, B., Praehofer, H., & Kim, T. (2000). *Theory of modeling and simulation, second edn. Integrating discrete event and continuous complex dynamic systems.* New York: Academic Press.
225. Zemel, R. S., Huys, Q. J., Natarajan, R., & Dayan, P. (2005). Probabilistic computation in spiking populations. (S. K. Lawrence, Y. Weiss, & L. Bottou, Ed.) *Advances in Neural Information Processing Systems* (17), 1609–1616.
226. Zhang, M., Vassilliadis, S., & Fris, J. (1996). Sigmoid generators for neural computing using piece-wise approximation. *IEEE transactions on Computers , 45* (9).
227. Zhu, J., & Gunther, B. (1999). Towards an FPGA based reconfigurable computing environment for neural network implementations. *roceedings of Ninth International Conference on Artificial Neural Networks, 2*, pg. 661-666.

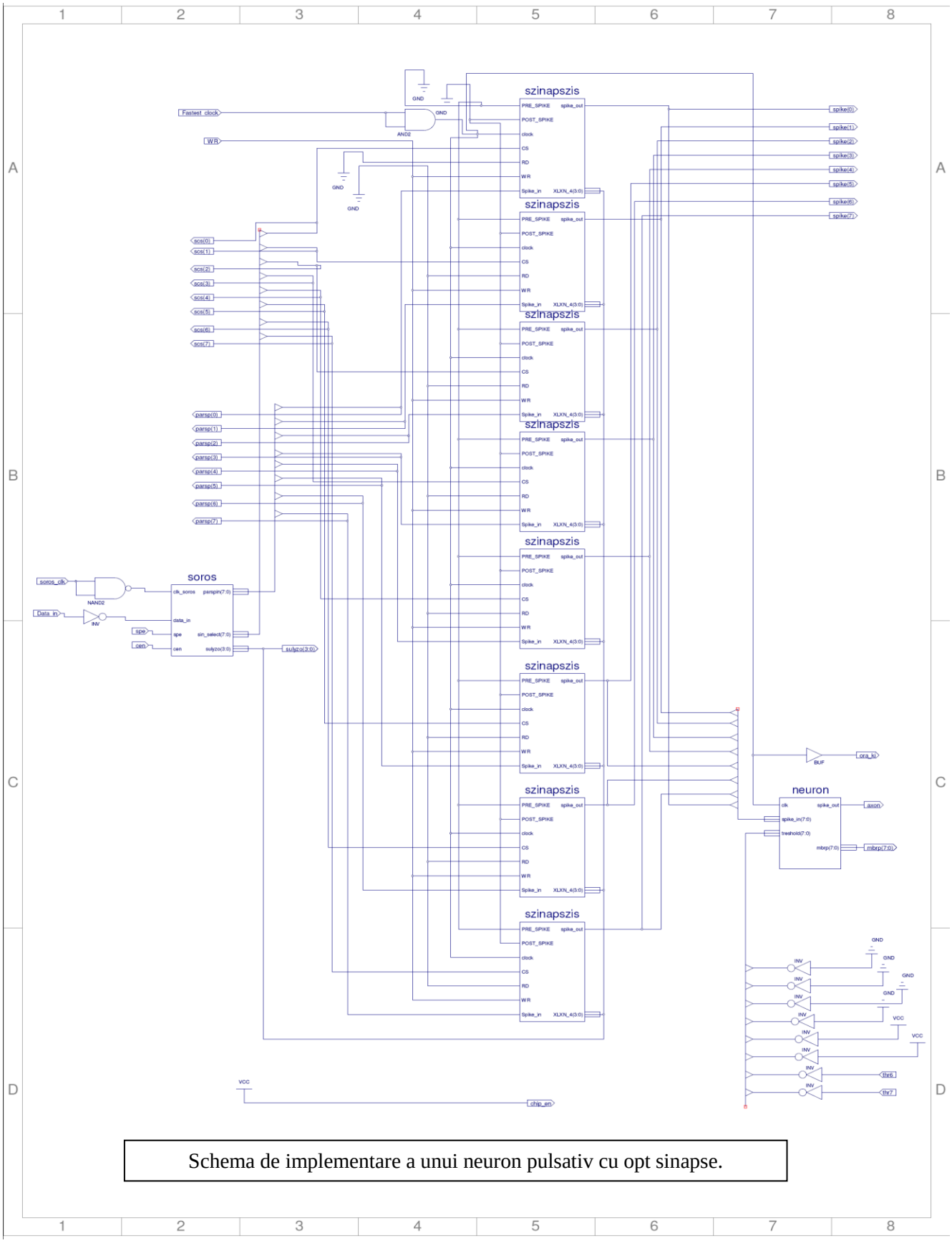
ANEXA 1 – DISTRIBUŢIA COMPONENTELOR UTILIZATE PE CIRCUITELE FPGA (DETECTOR FRECVENŢE VS. CLASIFICARE IRIS DATASET)



În partea stângă a figurii se poate vedea utilizarea resurselor FPGA de către circuitul ce implementează aplicaţia de detectarea a componentelor de frecvenţe, prezentată în subcapitolul 5.5.2 şi pe care se poate distinge, aria de circuit utilizată de modulul de control şi antrenare (control unit) cu culoare neagră respectiv RNP reprezentată cu galben.

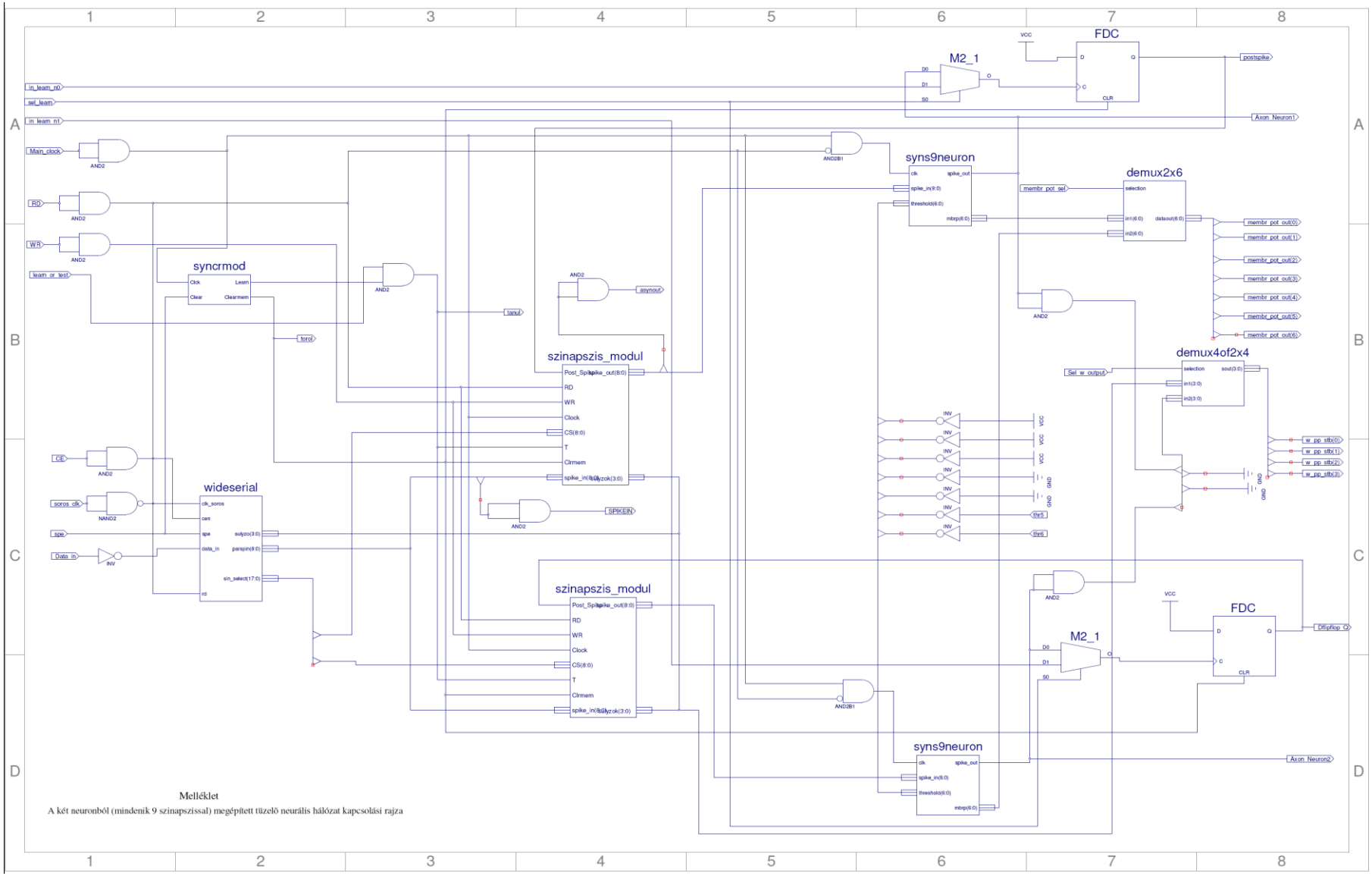
Imaginea din dreapta figurii prezintă utilizarea de resurse reconfigurabile de către proiectul ce implementează aplicaţia benchmark de clasificare a setului de date Fischer IRIS, expusă la subpunctul 5.5.3. A se nota faptul, că cele patru microcontrolere PicoBlaze din acest proiect (marcate cu negru pe figură) utilizează o suprafaţă relativ mică. Suprafaţa în albastru deschis (dreapta-centru) este mult mai mare, e reprezentând blocul neuronilor de intrare ce codifică valorile de intrare.

ANEXA 2- SCHEMA DE IMPLEMENTARE A UNUI NEURON PULSATIV



Schema de implementare a unui neuron pulsativ cu opt sinapse.

ANEXA 3- SCHEMA DE IMPLEMENTARE A UNEI REŢELE NEURONALE PULSATIVE



Melléklet

A két neuronból (mindenik 9 szinapszissal) megépített tüzelő neurális hálózat kapcsolási rajza

ANEXA 4- SCHEMA DE IMPLEMENTARE A REŢELE NEURONALE PULSATIVE PENTRU CLASIFICARE SETULUI DE DATE IRIS