UNIVERSITATEA “TITU MAIORESCU” DIN BUCURESTI

FACULTATEA DE INFORMATICA

LUCRARE DE LICENTA

COORDONATOR STIINTIFIC:

**Prof. Dan Laurentiu Grecu**

ABSOLVENT:

**Mihai Sarbulescu**

SESIUNEA IULIE

2015

UNIVERSITATEA “TITU MAIORESCU” DIN BUCURESTI

FACULTATEA DE INFORMATICA

LUCRARE DE LICENTA

**Sistem hard/soft de prelucrare si analiză unde cerebrale – achiziția, digitizarea și prelucrarea undelor cerebrale**

COORDONATOR STIINTIFIC:

**Prof. Dan Laurențiu Grecu**

ABSOLVENT:

**Mihai Sârbulescu**

SESIUNEA IULIE

2015

**UNIVERSITATEA TITU MAIORESCU**

**FACULTATEA DE INFORMATICĂ**

**DEPARTAMENTUL DE INFORMATICĂ**

**REFERAT**

***DE APRECIERE A LUCRĂRII DE LICENȚĂ/DISERTAȚIE***

***TITLU:***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***ABSOLVENT/ MASTERAND:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***PROFESOR COORDONATOR:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Referitor la conținutul lucrării, fac următoarele aprecieri:

|  |  |  |
| --- | --- | --- |
| A. | Conținutul științific al lucrării | 1 2 3 4 5 6 7 8 9 10 |
| B. | Documentarea din literatura de specialitate | 1 2 3 4 5 6 7 8 9 10 |
| C. | Contribuția proprie | 1 2 3 4 5 6 7 8 9 10 |
| D. | Calitatea exprimării scrise și a redactării lucrării | 1 2 3 4 5 6 7 8 9 10 |
| E. | Conlucrarea cu coordonatorul științific | 1 2 3 4 5 6 7 8 9 10 |
| F. | Realizarea aplicației practice | 1 2 3 4 5 6 7 8 9 10 |
|  | **Punctaj total** = (A+B+C+D+E+2F)/7 |  |

În concluzie, consider că lucrarea de licență/disertație întrunește/ nu întrunește condițiile pentru a fi susținută în fața comisiei pentru examenul de licență/disertație din sesiunea \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ și o apreciez cu nota\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

***CONDUCĂTOR ȘTIINȚIFIC,***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Conținut

[1. Introducere 5](#_Toc423023160)

[2. Noțiuni teoretice 6](#_Toc423023161)

[2.1. Brain-Computer Interface (BCI) 6](#_Toc423023162)

[2.2. Electroencefalografie (EEG) 6](#_Toc423023163)

[2.3. Self-Organizing Map (SOM) 9](#_Toc423023164)

[2.4. Transformări Fourier 12](#_Toc423023165)

[3. Prezentare generala a sistemului 14](#_Toc423023166)

[4. Descriere Hardware 15](#_Toc423023167)

[4.1. Ubuntu OS Laptop 15](#_Toc423023168)

[4.2. EEG TGAM Chip (NeuroSky) 15](#_Toc423023169)

[4.3. RF-USB Dongle (NeuroSky) 18](#_Toc423023170)

[5. Descriere Software 19](#_Toc423023171)

[5.1. EEG Connector - Linux (Ubuntu) shared library 19](#_Toc423023172)

[5.2. NodeJS Web Server backend (cu JXCore si Node-FFI) 35](#_Toc423023173)

[5.3. Librăria D3.js si AJAX cu JQuery.js pentru JavaScript frontend 39](#_Toc423023174)

[5.4. Self-Organizing Map (SOM) application 42](#_Toc423023175)

[6. Modul de funcționare a sistemului si rezultate obținute 45](#_Toc423023176)

[6.1. Preluarea si procesarea undelor cerebrale 45](#_Toc423023177)

[6.2. Afișarea undelor cerebrale 50](#_Toc423023178)

[6.3. Clasificarea undelor cerebrale 56](#_Toc423023179)

[6.4. Rezultate obținute 59](#_Toc423023180)

[7. Concluzii 60](#_Toc423023181)

[8. Bibliografie 61](#_Toc423023182)

# Introducere

Sistemele Brain-Computer Interface (BCI) au potențialul de a oferi oamenilor un mod inedit si inovativ, non-muscular, de a comunica cu dispozitive externe direct cu creierul. Aceste sisteme se bazează pe citirea si interpretarea comenzilor electrice codate ale creierului în semnale neuro-psihice, fără a se folosi de interfața musculara a sistemului nervos central.

Electroencefalografia (EEG) este una dintre metodele non-invazive de a citi activitatea electrică cerebrală. Citirea semnalelor EEG pentru a comunica cu un calculator a devenit relativ ușoara cu ajutorul ultimelor tehnologii.

În general, EEG este mai răspândit si folosit în cercetarea medicală. Din acest motiv majoritatea persoanelor nu sunt la curent cu progresele cercetării in acest domeniu sau îl considera încă un domeniu al filmelor SF .

Cu toate ca au început sa apară tot mai multe aplicații si pentru domeniul comercial, non-medical, acestea sunt încă foarte rudimentare, deși s-ar putea s-ar putea obține performanțe mult mai semnificative și de mai mare complexitate în acest domeniu. Cred că motivul pentru care nu se investește la același nivel de performanță și complexitate în soluțiile existente non-medicale (ex. entertainment) este faptul că se presupune ca nu exista piața pentru acest domeniu și, din acest motiv, oamenii de știința specializați în acest domeniu, nu lucrează decât în instituții medicale.

Dintr-un anumit punct de vedere este firesc, ar trebui să ne concentrăm pe lucrurile cu adevărat importante, și anume, să încercam să ajutăm oamenii cu probleme medicale care ar trebui să beneficieze cel mai mult de acest tip de cercetare. Cred însă că, dacă am încerca sa facem acest domeniu mai bine cunoscut, s-ar stimula investițiile în cercetare și mai mulți elevi si studenți ar fi interesați să lucreze în acest domeniu.

Unul din scopurile acestei lucrări constă în încercarea de a face marketing acestui domeniu, demonstrând cât de ușor este, de fapt, să construim un sistem BCI bazat pe EEG pentru a mapa diferite acțiuni ale unui calculator pe diferite modele detectate din activitatea undelor cerebrale. Dacă am reuși să demonstrăm ca acest domeniu este „cool”, atunci cu siguranță am avea mult mai mulți oameni interesați să studieze și să lucreze, atât în scopuri de divertisment, cât și în scopuri medicale.

La fel de interesant și de viitor este și domeniul inteligenței artificiale (AI). Combinând aceste doua științe se pot crea lucruri, pentru moment, inimaginabile. Acestea stau la baza următorului progres major al științei și al omenirii.

Sistemul prezentat în această lucrare este un sistem generalizat care citește undele cerebrale prin tehnologia EEG, le interpretează și creează modele pe baza rețelelor neurale din inteligența artificială, numite Self-Organizing Maps (SOM).

Aplicația practică a acestui sistem va fi mișcarea mouse-ului unui calculator cu ajutorul acestor modele detectate. Fiind un sistem generalizat, se poate extinde foarte simplu adăugând-se acțiuni noi la care un calculator poate reacționa în orice fel, definind modele noi de clasificare.

Obiectivul acestei lucrări de licența este de a demonstra cat de ușor este crearea unui simplu Brain-Computer Interface generic. Acesta poate fi adaptat pentru a produce rezultate mult mai complexe.

# Noțiuni teoretice

## Brain-Computer Interface (BCI)

Brain-Computer Interface (BCI) este un sistem de comunicare directă între creier și un dispozitiv extern. Domeniul de cercetare și dezvoltare în BCI a început în anii 1970 și se bazează în mare parte pe aplicații neuro-prostetice.

Istoria BCI începe cu descoperirea de către Hans Berger a activității electrice a creierului uman și a dezvoltării electroencefalografiei (EEG). In 1924, Berger a înregistrat activitatea unui creier uman folosindu-se de EEG și a reușit să identifice activitate oscilară in creier analizând înregistrările. Una dintre undele pe care le-a identificat este unda alpha (8-13 Hz), supranumită și unda lui Berger. Pentru a înregistra această activitate a inserat fire de argint sub scalpul pacienților investigați. Firele de argint au fost mai târziu înlocuite de folii de argint aplicate pe capul pacienților cu ajutorul unor bandaje de cauciuc.

Există trei categorii de BCI, cele invazive, cele parțial-invazive si cele non-invazive:

* **Invaziv**: Senzorii se implementează direct în creierul pacientului în timpul unei operații neuro-chirurgicale. Pentru că sunt implantate direct în creier produc cel mai de calitate semnal dar datorită reacției organismului la obiecte străine semnalul slăbește în timp sau este oprit din cauza producerii de țesut în zona de inserție.
* **Parțial-invaziv**: Senzorii se implementează sub craniu, sub duramater, deasupra i, într-un înveliș subțire de plastic. Prin această metodă se detectează un semnal mai slab decât cea invazivă, dar mai puternic decât prin metoda non-invazivă.
* **Non-invaziv**: Senzorii se implementează pe craniu atașați în diferite moduri. Deși încă se pot detecta frecvențele undelor cerebrale, este mult mai greu de sesizat acțiunea neuronilor individual sau din ce zonă creierului provine. Cea mai studiată formă a acestei categorii este EEG.

## Electroencefalografie (EEG)

Electroencefalografia (EEG) măsoară fluctuațiile voltajelor care rezultă din curentul ionic din neuroni și este foarte des folosită în diagnosticarea epilepsiei.

În general, înregistrările undelor cerebrale se obțin prin plasarea de electrozi pe scalp cu ajutorul unui gel sau a unei paste conductive. Se pot folosi însă și electrozi care sunt menținuți pe scalp cu ajutorul unei benzi elastice sau prin alte mijloace. Acest tip de electrozi se numesc electrozi uscați, iar cei atașați cu gel sau pastă conductivă se numesc electrozi umezi. Deși intensitatea semnalului poate fi mai slabă, se poate folosi orice metal conductiv plasat pe cap.

În citirea EEG complexă / aplicată, se folosesc sisteme cu un număr mare de electrozi. Aceștia formează o rețea de citire cu electrozi plasați în puncte/locuri specifice, cu nume specifice, care au fost stabilite în urma unor cercetări și definite în sistemul internațional 10-20.

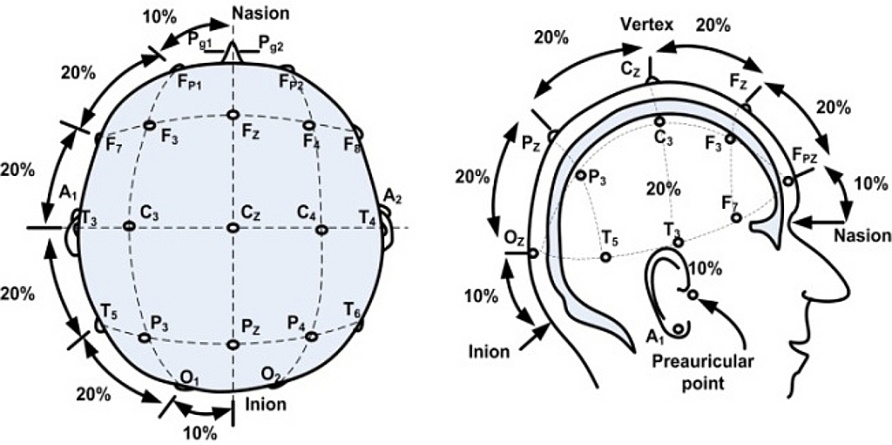
Sistemul 10-20 este o metodă internațional recunoscută pentru a descrie și aplica electrozi pe scalp în cadrul unei citiri EEG. Acest sistem a fost dezvoltat pentru a asigura reproductibilitatea standardizată a înregistrărilor, așa încât să poată fi comparate rezultatele obținute pe un pacient în diferite perioade de timp dar și cu ale altui pacient.

Sistemul este bazat pe relația dintre locul de aplicare a unui electrod și baza zonei cortexului cerebral. Numerele „10” si „20” din numele sistemului se referă la faptul că distanța dintre electrozii adiacenți este 10% sau 20% din distanța totală față-spate sau dreapta-stânga a craniului.

Fiecărei zone i se atribuie o anumită literă care identifică lobul și un anumit număr care identifică emisfera. Literele sunt F (frontal), T (temporal), C (central), P (parietal) si O (occipital). Deși nu exista un lob central, litera „C” este folosită doar pentru a identifica zona centrală. Litera „z” se referă la un electrod plasat pe linia de mijloc. Numere pare (2, 4, 6, 8) indică pozițiile electrozilor emisferei drepte, iar numerele impare (1, 3, 5, 7) pe cele a emisferei stângi.

Pentru a defini poziționarea sistemului sunt folosite două repere anatomice:

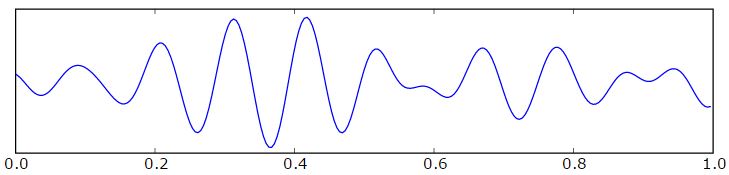
* **NASION:** zona din mijlocul ochilor, chiar deasupra nasului
* **INION:** zona cea mai de jos din spatele craniului



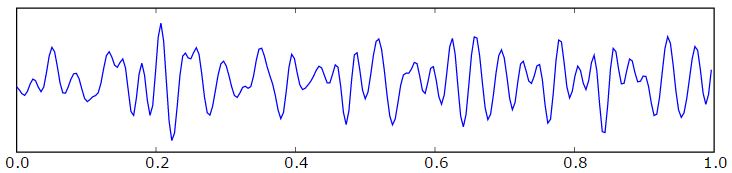
**[12]**

Activitatea EEG este descrisă ca o activitate ritmată. Aceasta activitate este împărțită în benzi de frecvență. Majoritatea semnalelor observate sunt între 1 și 20 de Hz, iar undele cerebrale principale sunt împărțite în lungimi de bandă numite: Alpha, Beta, Theta, Delta, Gamma, Miu.

**Alpha** are frecvențe între 7 și 14 Hz și se afla în zonele posterioare ale craniului, pe ambele parți, cu amplitudine ridicată pe partea emisferei dominante. Are activitate ridicată la închiderea ochilor și relaxare și scade la deschiderea ochilor și la efort mintal. Rezultate anormale ale acestei unde se regăsesc la persoanele care sunt în stare de comă.

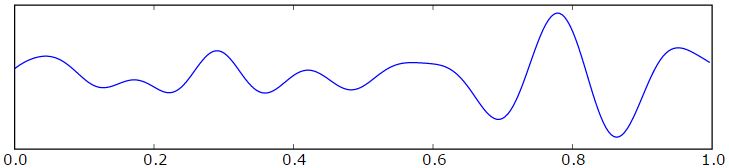
**[12]**

**Beta** are frecvențe intre 15 și 30 Hz și este distribuită simetric pe ambele părți ale capului, dar cel mai evident frontal. Activitatea din această zonă este legată de sistemul locomotor și scade, în general, în timpul mișcării. Amplitudinea scăzută a acestei unde cu frecvențe multiple și variate este asociată cu stări de gândire: activă, intensivă, stresată, obsesivă, concentrată. Este ritmul dominant al persoanelor alerte sau neliniștite.

****

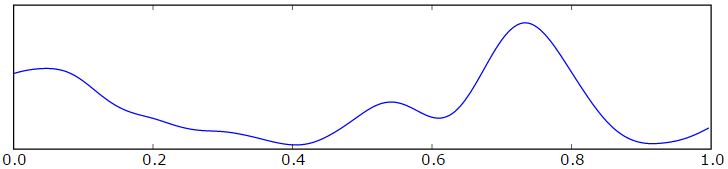
**[12]**

**Theta** are frecvențe intre 4 si 7 Hz si de obicei poate are activitate in zone care nu au legătura cu sarcina/acțiunea curenta. Are o activitate ridicata la copii sau in situații de excitare la adolescenți si adulți. In unele situații apare si in timpul meditației. In general, o activitate ridicata prelungita la oameni in vârsta, poate însemna o tulburare psihica.



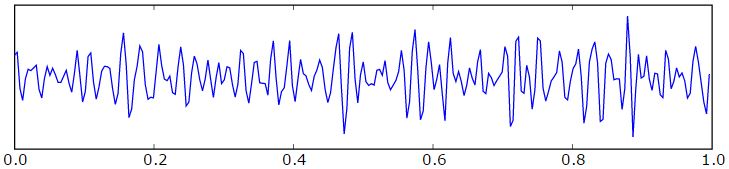
**[12]**

**Delta** are frecvențe sub 4 Hz și tinde să aibă cea mai mare amplitudine și cele mai încete unde. Are de obicei activitate ridicată la copii sau la adulți în timpul somnului, în faza *Non-rapid eye movement sleep* (NREM). Apare în zona frontală la adulți și în zona occipitală la copii. Activitatea anormală poate fi detectată în cazul diferitelor forme de leziuni cerebrale.



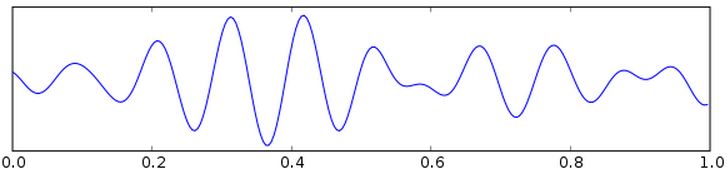
**[12]**

**Gamma** are frecvențe intre 30 si 100 Hz și are activitate ridicată în momentele în care se fac legături între diferite zone de neuroni pentru a forma o rețea complexă, având ca scop desfășurarea unei anumite funcții cognitive sau motorii, de exemplu pentru a trage cu arcul. De asemenea are activitate ridicată în procesul de amintire când se acceseaza memoria de scurtă durată. Deși nu a fost încă dovedit, activitatea scăzută a undelor Gamma poate însemna declin cognitiv.



**[12]**

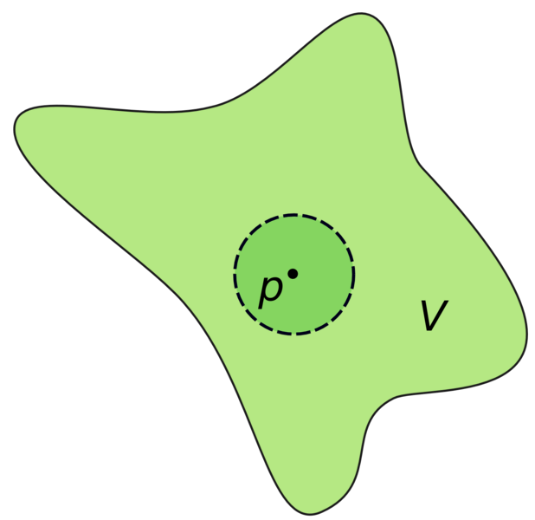
**Miu** are frecvențe între 8 si 13 Hz și se suprapune parțial cu alte unde, mai ales Alpha. Reflectă descărcările electrice ale neuronilor motorii în stadiul de relaxare. Deși se suprapune de obicei cu unda Alpha, apare în zone diferite ale creierului. Suprimarea activității este arată faptul că neuronii oglindă funcționează corect. Deficitul suprimării activității Miu, deci a neuronilor oglindă, ar putea fi un semn de autism.



**[12]**

## Self-Organizing Map (SOM)

Self-Organizing Map (SOM) este un tip de rețea neurală artificială (ANN) care este specializată în procesul de învățare nesupravegheată pentru a crea o reprezentare a unui spațiu de intrare a seturilor de antrenare, formând un tablou multi-dimensional. Se diferențiază de alte rețele neurale artificiale prin faptul ca folosesc o funcție de vecinătate.

In studiul spatiilor topologice, vecinătatea unui punct este o mulțime in care punctul poate fi mișcat într-o oarecare măsura, fără a părăsi acea mulțime. Daca X este un spațiu topologic si p este un punct al X, o vecinătate a lui p este o mulțime V, care include cel puțin o mulțime deschisa U care conține p.

\displaystyle  p \in U \subseteq V

**[12]**

La fel ca majoritatea rețelelor neurale artificiale, SOM funcționează în două moduri: învățare și mapare. Învățarea creează tabloul cu ajutorul intrărilor de instruire (training sets), iar maparea clasifică automat un nou vector de intrare. SOM este alcătuită din neuroni. Fiecare neuron are un vector de greutăți (weight vector) de aceeași dimensiune cu vectorul de intrare și o poziție în tablou spațial.

Aranjarea obișnuita a neuronilor este un spațiu bi-dimensional într-o grilă hexagonală sau rectangulară. SOM descrie o mapare dintr-un plan de intrare cu un număr mare de dimensiuni, într-un plan de mapare cu un număr mic de dimensiuni. Procedura pentru a plasa un vector din plan în tabloul de mapare, este de a găsi neuronul cu cea mai apropiată greutate de cea a vectorului din plan.

Scopul învățării unei SOM, este de a face ca diferite părți ale rețelei să răspundă similar la de diferite modele de intrare. Greutățile inițiale vor fi ori valori aleatorii mici, ori eșantionate egal din sub-spațiul planului din cei mai mari *eigenvectori* de componente principale. Cu această a doua metodă, învățarea este mult mai rapidă pentru ca greutățile inițiale au deja o aproximare bună a greutății SOM. Trebuie date multe *training sets* pentru toate variantele dorite de vectori pentru care se fac mapări și dintre acestea, multe iterații ale aceluiași tip.

Modul de învățare folosește învățarea competitivă. Când un *training set* este dat rețelei ca input, este calculata distanța sa Euclidiană către toți vectorii de greutate. Vectorul de greutate care este cel mai asemanător cu cel de intrare, este numit *best matching unit* (BMU). Greutățile BMU și neuronii apropiați de acestea în rețea sunt ajustați către vectorul de intrare. Magnitudinea ajustărilor în rețea scade cu timpul și cu distanta în rețea.

Formula de actualizare a unui neuron **v** cu vectorul de greutate **Wv**(s) este:

**Wv**(s + 1) = **Wv**(s) + Θ(u, v, s) α(s)(**D**(t) - **Wv**(s))

Aici, s este pasul curent, t este un index din *training set*, u este indexul BMU pentru **D**(t), α(s) este un coeficient de învățare de scădere monotona, **D**(t) este vectorul de intrare, Θ(u, v, s) este funcția de vecinătate care dă distanța dintre neuronul u și neuronul v din pasul s. Depinzând de implementare, t poate scana *training set*-ul sistematic (t este 0, 1, 2 … T-1, repetă, T unde este mărimea *training set*-ului), sau ales la întâmplare din *training set* sau poate avea un altfel de implementare de funcție de învățare.

Funcția de vecinătate Θ(u, v, s) depinde de distanta rețelei între BMU, (neuronul u) și neuronul v. În forma simplă este 1 pentru toți neuronii destul de apropiați de BMU si 0 pentru ceilalți, dar și o funcție Gaussiana este o alegere comună. Pentru orice tip de funcție însă, funcția de vecinătate scade în timp. La început, când vecinătatea este vastă, auto-organizarea are loc la scara globală în plan.

În timpul mapării va exista un singur câștigător (*winner takes all*): neuronul al cărui vector de greutate este cel mai apropiat de vectorul de intrare. Putem afla care va fi calculând distanta Euclidiana dintre vectorul de greutate și vectorul de intrare.

Variabilele folosite în ecuație sunt:

* este iterația curentă



* \lambda este limita iterațiilor
* t este indexul datelor vectorului de intrare în setul de date \mathbf{D}
* \mathbf{D(t)} este vectorul de intrare țintă
* v este indexul neuronului din rețea
* \mathbf{W_v} este vectorul de greutate curent al neuronului *v*
* este indexul BMU în rețea



* \Theta (u, v, s) este o restrângere cauzată de distanța de la BMU, numită funcția de vecinătate
* \alpha (s) este restrângerea de învățare cauzata de procesul de iterație

Algoritmul folosit este:

1. Generarea aleatorie a vectorilor de greutate a neuronilor
2. Preluăm un vector de intrare \mathbf{D(t)}
3. Traversăm fiecare neuron din rețea și:
   1. Folosim funcția de distanță Euclidiană pentru a găsi similaritatea (asemănarea) dintre vectorul de intrare și vectorul de greutate a neuronului.
   2. Înregistrăm neuronul care este la cea mai mica distanta (acesta va fi BMU).
4. Actualizăm neuronii din vecinătatea BMU (incluzând neuronul BMU) aducând-le mai aproape de vectorul de intrare: **Wv**(s + 1) = **Wv**(s) + Θ(u, v, s) α(s)(**D**(t) - **Wv**(s))
5. Incrementăm s si repetăm de la pasul 2. cat timp s < \lambda

## Transformări Fourier

Transformările Fourier (FT) descompun o funcție de timp (ex. un semnal) după frecvența și produc un spectru al acesteia. Transformarea Fourier a unei funcții de timp este o funcție cu valori complexe de frecvență, a cărei valori absolute reprezintă cantitatea frecvenței care se află în funcția originală, și al cărei argument complex este defazarea sinusoidei de baza din acea frecvență. FT se mai numește **reprezentarea domeniului de frecventa a semnalului original.**

Operațiile lineare pentru un domeniu (timp sau frecventa) au operații corespunzătoare în celălalt domeniu, care sunt uneori mai ușor de analizat. Operațiunea de diferențiere în domeniul timp corespunde cu multiplicarea cu frecvență, așa că anumite ecuații diferențiale sunt mai ușor de analizat în domeniul frecvență.

Putem numi FT prin \hat{f} a unei funcții integrale f : \mathbb R \rightarrow \mathbb C :

\hat{f}(\xi) = \int_{-\infty}^\infty f(x)\ e^{- 2\pi i x \xi}\,dx, pentru orice număr real *ξ*

Când variabila independentă *x* reprezintă timp, variabila de transformare *ξ* reprezintă frecvența. In condiții propice, f este determinat de \hat f cu ajutorul transformării inverse:

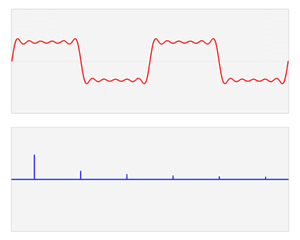
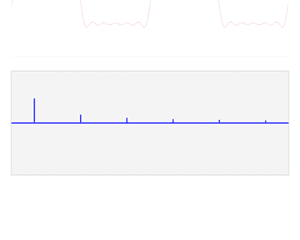
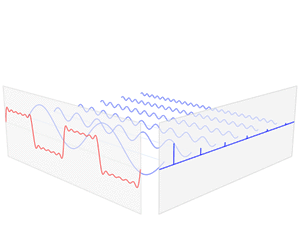
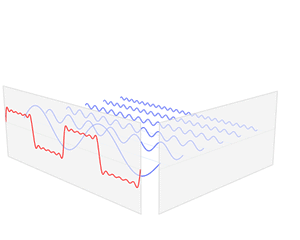
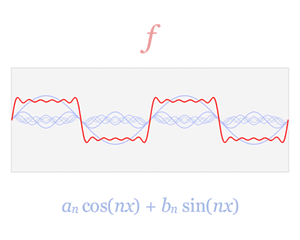
f(x) = \int_{-\infty}^\infty \hat f(\xi)\ e^{2 \pi i \xi x}\,d\xi, pentru orice număr real *x*

Afirmația că f poate fi reconstruit din \hat f este cunoscută drept teorema de inversie Fourier. Funcțiile f și \hat{f} se mai numesc și perechea de transformare Fourier.

Nevoia de FT vine din studiul seriilor Fourier, în care funcții complicate dar periodice sunt scrise ca sumă a unor funcții simple reprezentate matematic de *sinus* si *cosinus*. FT este o prelungire a seriilor Fourier care rezultă când perioada reprezentării funcției este lungită și lăsată să tindă spre infinit.

Datorită proprietăților lui *sinus si cosinus*, este posibil să recuperăm amplitudinea fiecărei unde a unei serii Fourier folosind o integrală. În majoritatea cazurilor este de preferat să se folosească formula *Euler* pentru a scrie serii Fourier în termenii undelor de baza *e*2*πiθ*, care afirmă că *e*2*πiθ* = cos(2*πθ*) + *i* sin(2*πθ*).

Avantajul este că simplifică multe din formulele pe care le vom folosi. Rescrierea *sinus*-urilor și *cosinus*-urilor în forma unor exponenți complecși creează nevoia de a scrie coeficienții Fourier sub forma unui număr complex. Acest număr complex reprezintă *amplitudinea* (sau mărimea) undei prezente în funcție de *faza* (sau unghiul inițial) acesteia. Exponențialele complexe pot conține „frecvențe” negative. Daca *θ* este măsurat în secunde, atunci ambele unde *e*2*πiθ* și *e*−2*πiθ* completează un ciclu pe secundă dar reprezintă frecvențe diferite.



**[12]**

# Prezentare generala a sistemului

Sistemul are doua părți separate:

1. Citirea și procesarea undelor cerebrale cu ajutorul unui driver care se conectează serial la USB RF Dongle, care la rândul lui este conectat prin radio la chip-ul EEG TGAM și afișarea live a acestor date într-o pagina web într-o diagramă dinamică.
   1. Driver-ul de citire și procesare este scris în C pentru Linux (Ubuntu) și compilat ca shared library.
   2. O aplicație web server-side care rulează pe Linux (Ubuntu) în NodeJS, va folosi această librărie (driver) pentru a primi seturi de date reprezentând frecvențele undelor cerebrale și va asculta cereri de la clienți pe un anumit port.
   3. O aplicație front-end, mare parte JavaScript (folosind D3.js si JQuery.js) va cere date în fiecare secundă de la server-side și le va afișa într-o diagramă dinamică.
2. Citirea și procesarea undelor cerebrale cu ajutorul unui driver care se conectează serial la USB RF Dongle, care la rândul lui este conectat prin radio la chip-ul EEG TGAM și clasificarea acestora folosind SOM (*Self-Organizing Map*) pentru a acționa (mișcarea mouse-ului de exemplu) bazat pe clasificările stărilor diferite ale undelor cerebrale.
   * Driver-ul de citire și procesare este scris în C pentru Linux (Ubuntu) și compilat ca shared library.
   * O librărie scrisă în C++, tot de tip shared library pentru Linux (Ubuntu) va putea primi aceste date și va face clasificări pentru a recunoaște modele folosind SOM (Self-Organizing Map).

O aplicație client va folosi librăria (driver) pentru a primi seturi de date reprezentând frecvențele undelor cerebrale citite, după care se va folosi de librăria SOM pentru a primi înapoi rezultatul unui anumit model (pattern) și pe baza acestei clasificări va face diferite acțiuni – de exemplu mișcarea mouse-ului stânga sau dreapta în funcție de frecvențele undelor cerebrale.

# Descriere Hardware

## Ubuntu OS Laptop

Deși specificațiile calculatorului folosit nu sunt relevante în acest scenariu, *software-*ul a fost scris pentru sistemul de operare Ubuntu.

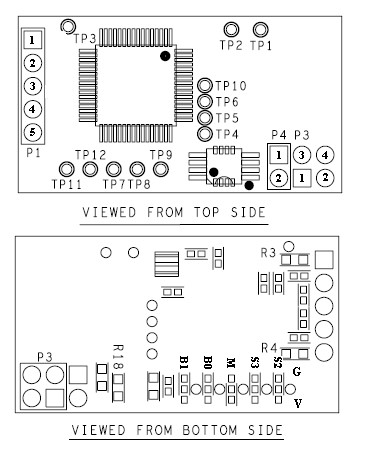
## EEG TGAM Chip (NeuroSky)

Specificațiile tehnice ale Chip-ului EEG TGAM:

|  |  |  |
| --- | --- | --- |
| Clasificare | Specificatii | Note |
| Product Family | ThinkGear-AM | A = ASIC, M = Module |
| Model Number | TGAM1 |  |
| Revision Number | 2.3 |  |
| Module Dimension (Max) | 27.9mm x 15.2mm x 2.5mm | 1.10in x 0.60in x  0.10in  (L x W x H) |
| Module Weight (Max) | 130mg | 0.0045 ounces |
| Operating Voltage | 2.97V ~ 3.63V | Stuff Option  SP6200 3.0~6.0V  MAX1595 1.8~5.5V |
| Max Input Voltage Noise | 10mV Peak to Peak |  |
| Max Power Consumption | 15mA @ 3.3V |  |
| ESD Protection | 4kV Contact Discharge  8kV Air Discharge | Tested at EEG, REF,  GND |
| Output Interface Standard | UART(Serial) | TX, RX, VCC(+), GND(-) |
| Output Baud Rate | 1200, 9600, 57600 | Default set with stuff option |
| #EEG Channels | 1 | 3 contacts (EEG, REF,  GND) |

**[2]**

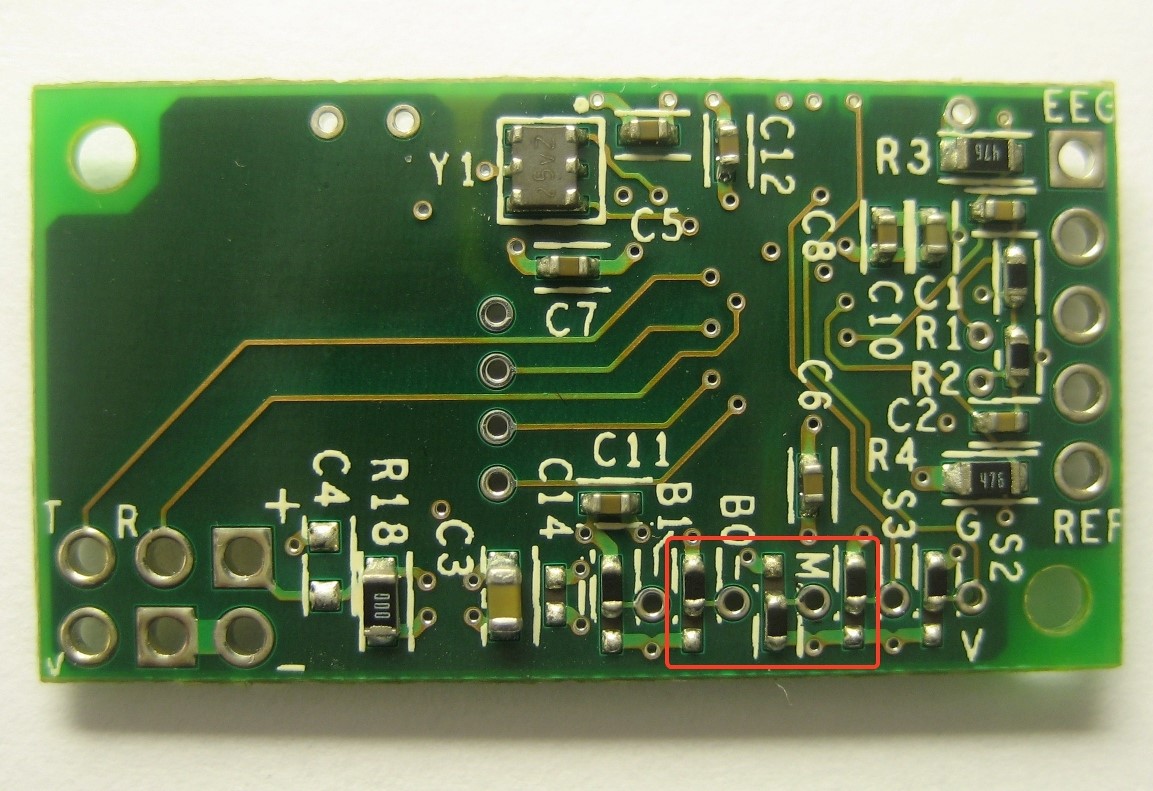
Pinii I/O ale Chip-ului EEG TGAM:



**[2]**

* **Header P1** (Electrode)
* **Pin1**: EEG Electrode "EEG"
* **Pin2**: EEG Shield
* **Pin3**: Ground Electrode
* **Pin4**: Reference Shield
* **Pin5**: Reference Electrode "REF"
* **Header P4** (Power)
* **Pin1**: VCC "+"
* **Pin2**: GND "-"
* **Header P3** (UART/Serial)
* **Pin1**: GND "-"
* **Pin2**: VCC "+"
* **Pin3**: RXD "R"
* **Pin4**: TXD "T"

TGAM are *pad*-uri de configurare care pot fi folosite pentru a schimba doua setări *default* care sunt aplicate la alimentarea si pornirea *chip*-ului. Acestea se pot găsi pe spatele *chip*-ului așa cum indica pătratul roșu din imaginea de mai jos. *Pad*-urile **B0** si **B1** configurează *baud rate* *output* si și conținutul datelor, iar pad-ul **M** configurează filtrul *notch* de frecvență.



**[2]**

|  |  |  |
| --- | --- | --- |
| BR1 | BR0 | Function |
| GND | GND | 9600 Baud with Normal\* Output Mode |
| GND | VCC | 1200 Baud with Normal\* Output Mode |
| VCC | GND | 57.6k Baud with Normal\* + Raw Output Mode |
| VCC | VCC | N/A |

\* output-ul normal include: *calitatea semnalului*, *valorile EEG*, *atenție*, *meditație* **[2]**

Rata *baud* poate fi configurată după pornirea *chip*-ului, prin trimiterea de comenzi prin interfața UART. Aceste comenzi sunt descrise în tabelul de mai jos. Când *chip*-ul este repornit, aceste valori revin la cele *default*, date de către B0 si B1.

|  |  |
| --- | --- |
| Command | Function |
| 0x00 | 9600 Baud with Normal\* Output Mode |
| 0x01 | 1200 Baud with Normal\* Output Mode |
| 0x02 | 57.6k Baud with Normal\* + Raw Output Mode |

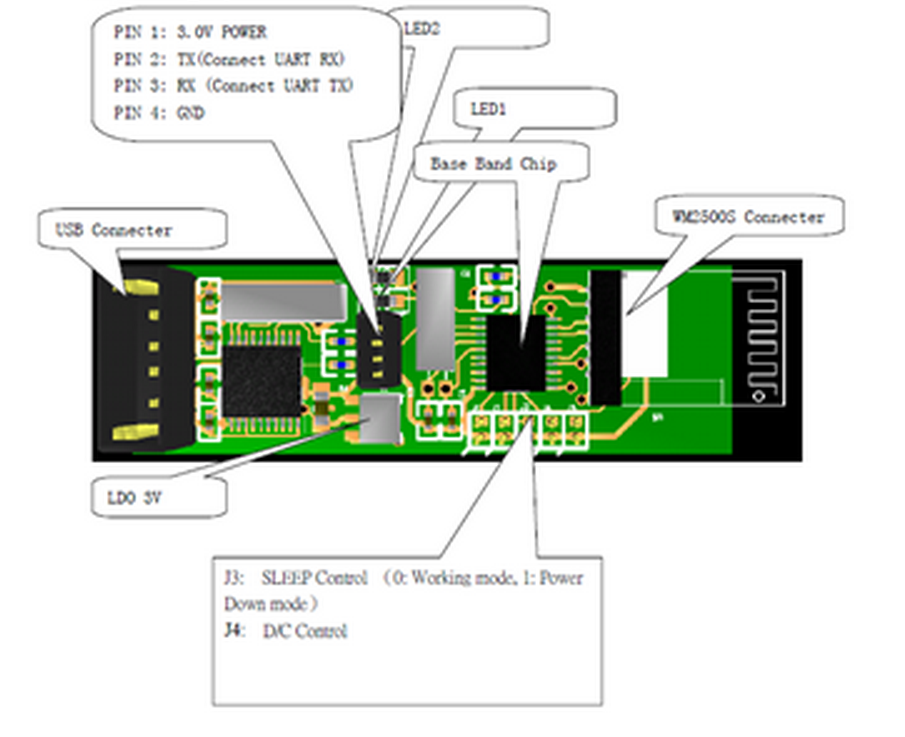
\* output-ul normal include: *calitatea semnalului*, *valorile EEG*, *atenție*, *meditație* **[2]**

## 4.3. RF-USB Dongle (NeuroSky)

Chip-ul TGAM este pus pe un shield care conține un modul bluetooth care comunica cu un (USB) RF Dongle.

Microcontroller-ul de pe shield-ul pe care este pus chip-ul TGAM, comunică prin UART (*Universal Asynchronous Reciver/Transmitter*) cu chip-ul TGAM și trimite semnalul mai departe prin chip-ul (radio) bluetooth.

Chip-ul (radio) bluetooth de pe RF Dongle primește semnalul și îl trimite către microcontroller care îl scoate pe conectorul USB.



**[2]**

# Descriere Software

## EEG Connector - Linux (Ubuntu) shared library

Pentru comunicarea cu modulul EEG TGAM de pe Ubuntu prin RF Dongle prin USB este nevoie de un Connector. Pentru a putea folosi acest Connector pentru orice aplicație care ar trebui să primească informațiile de la TGAM va fi implementat ca shared library.

Shared Libraries sunt librării care pot fi legate (*linked*) de orice program la pornire (*run-time*) și prin urmare permit o cale prin care putem folosi cod care poate fi încărcat oriunde în memorie. Odată încărcată, o librărie de acest tip poate fi folosită de mai multe programe în același timp, așa încât memoria nu este folosită pentru a ține același cod de mai multe ori pentru fiecare program ci este prezentă o singură dată în memorie.

Shared Libraries furnizează modularitate mediului de dezvoltare, pentru că, dacă este nevoie, se poate modifica și extinde codul librăriei, fără a trebui să se facă modificări și recompilări programelor care se folosesc de aceasta.

Aceste librării pot fi accesate folosind nume diferite:

* Numele folosit de către linker – prefixul „lib” urmat de numele librăriei, urmat de extensia „.so”
* Numele calificat întreg, numit si *soname* – prefixul „lib” urmat de numele librăriei, urmat de extensia „.so”, urmat de „.” și un număr de versiune
* Numele „real” (*Real Name*) – prefixul „lib” urmat de numele librăriei, urmat de extensia „.so”, urmat de „.” și un număr al versiunii minore, urmat de „.” și un număr al versiunii finale (*release version*)

Este nevoie de denumiri care să conțină și versiunea, pentru că putem avea diferite versiuni coexistente) ale unei librării. În cazul în care un program are nevoie de o versiune specifică să o poată găsi și folosi pe aceasta. Altfel programele care nu caută o versiune specifica o vor folosi întotdeauna pe ultima care există în sistem.

Librăriile au anumite locuri unde se pot găsi/copia în sistemul de fișiere, după standardele ierarhiei sistemelor de fișiere (FHS):

* */lib* – toate librăriile care vor fi folosite la pornirea sistemului de operare și care vor fi folosite de sistemul de fișiere root (*root filesystem*)
* */usr/lib* – librăriile folosite de sistem intern care nu sunt făcute cu scopul de a fi folosite de utilizatori sau script-uri
* */usr/local/lib* – dacă există acest director va conține toate librăriile folosite de distribuția respectivă a sistemului de operare (acestea sunt librăriile non-standard care se pot descarca și care ar putea avea eventuale bug-uri)

După ce creăm o Shared Library, o copiem în directorul de librarii pe care îl dorim (ex. */usr/local/lib* sau */usr/lib*), după care rulăm comanda **ldconfig** în acest director. Acest lucru este necesar, așa încât loader-ul (*/lib/ld-linux.so.X* – unde „X” este versiunea) să știe de unde să găsească librăria când trebuie sa o încarce automat pentru un anumit program care o va folosi.

Toate directoarele care pot fi căutate de către loader pentru a găsi librăriile cerute de către programe sunt definite în */etc/ld.so.conf*. Căutarea în toate aceste directoare poate lua mult timp, așa că de fiecare dată când rulam **ldconfig**, creează link-uri simbolice și un cache cu acestea în fișierul */etc/ld.so.cache* unde va scrie toate informațiile de care este nevoie pentru programe așa încât să poate încarca librăriile cerute.

Pentru a crea un Shared Library, trebuie sa compilăm într-un anumit fel – vom lua exemplul de mai jos pentru a explica.

Considerăm că vom crea librăria *libshared.so* pornind de la *shared.h* și *shared.c* care ar putea să conțină următorul cod:

* *shared.h*

#include <stdio.h>

extern unsigned int add(unsigned int a, unsigned int b);

* *shared.c*

#include "shared.h"

unsigned int add(unsigned int a, unsigned int b)

{

printf("\n Inside add()\n");

return (a + b);

}

Acum trebuie sa compilăm pentru o Shared Library:

* **gcc -c -Wall -Werror -fPIC shared.c**
* **gcc -c -shared -o libshared.so shared.o**

Prima comandă compilează codul C în cod binar, după care a doua comandă creează din fișierul binar „shared.o” librăria „libshared.so”. Pentru a folosi într-un program vom lua acest exemplu – *main.c*:

#include <stdio.h>

#include "shared.h"

int main(void)

{

unsigned int a = 1;

unsigned int b = 2;

unsigned int result = 0;

result = add(a, b);

printf("\n The result is [%u]\n", result);

return 0;

}

Acum trebuie compilat acest program, așa încât să îi spunem linker-ului că va avea nevoie de aceasta librărie. Dacă nu vrem să copiem librăria în unul dintre directoarele standard și să rulăm **ldconfig**, atunci putem să îi spunem liker-ului unde poate găsi această librărie la compilare folosind parametrul „-L” si apoi să compilăm în felul următor (presupunem că librăria se află în */usr/myuser/exercitiu*):

* **gcc -L/usr/myuser/exercitiu -Wall main.c -o main -lshared**

Acum, dacă rulîm programul „main”, vom vedea următorul rezultat în terminal:

Inside add()

The result is [3]

Putem vedea deci că librăria „libshared.so” a fost încărcată și programul „main” a rulat funcția *add(…)* din aceasta.

Pentru a crea conector-ul de Linux (Ubuntu) ca Shared Library care să știe să se conecteze și să comunice cu EEG TGAM Chip prin RF USB Dongle, ne folosim de documentația tehnică a Chip-ului (*ThinkGear Serial Stream Guide*).

Statusuri pentru RF USB Dongle:

* **Idle/Disconnected** – în acest status, RF Dongle-ul este în modul standby și LED-ul său luminează roșu. Acesta este status-ul în care va reveni și după ce îl deconectăm de la EEG TGAM Chip dacă vrem să oprim citirea.
* **Searching** – în acest status, RF Dongle-ul este în căutarea unui EEG TGAM Chip din apropiere, fie după un ID specific, fie primul pe care îl poate contacta (în cazul în care ar fi mai multe în jurul său).
* **Connected** – în acest status, RF Dongle-ul va fi conectat la chip-ul EEG TGAM și va trimite date de la acesta. LED-ul său va sta aprins în culoarea albastru în acest status.

Tipuri de comenzi pe care le putem trimite la RF USB Dongle:

* **Connect** – când va primi această comandă, Dongle-ul va încerca să se conecteze la un EEG TGAM Chip cu un ID specific care este pasat în comanda de conectare **[4]**:

Packet Format: (Headset Connect Command Byte) (2-byte Global Headset ID)

Example Command Bytes: 0xC0 0x00 0x01

The command above will request connection to a global headset ID of 0x0001.

* **Disconnect** – când va primi aceasta comanda, Dongle-ul se va deconecta de la EEG TGAM Chip-ul la care este conectat actual **[4]**:

Packet Format: (Headset Disconnect Command Byte)

Example Command Bytes: 0xC1

The command above will request disconnection from any/all headsets that is connected to the dongle.

* **Auto-connect** – când va primi această comandă, Dongle-ul va încerca să se conecteze la orice EEG TGAM va fi în apropiere, primul venit, primul servit, neținând cont de vreun ID specific **[4]**:

Packet Format: (Headset Auto-Connect Command Byte)

Command Byte: 0xC2

The command above will connect to any headsets it can find within 10 seconds.

Status-uri de conectare **[4]**:

* **Found and Connected** – acest status este trimis ca răspuns comenzii **Connect** sau **Auto-connect**, pentru a lăsa aplicația să știe că a avut loc conectarea cu succes:

DataRow Format: (Headset Connect Success Code) (Data Length) (2 byte Global Headset ID)

Example Packet: 0xAA 0xAA 0x04 0xD0 0x02 0x05 0x05 0x23 means that

the dongle found a headset with Global Headset ID 0x0505 and connected to it.

* **Not Found** – acest status este trimis ca răspuns atunci când EEG TGAM Chip-ul la care încercăm să ne conectăm folosind ID-ul specific, nu a fost găsit sau dacă nu a fost găsit nici un chip în modul de conectare automată (**Auto-connect**) **[4]**:

DataRow Format: (Headset Not Found Code) (Data Length) (2 byte Global Headset ID)

Example Packet: 0xAA 0xAA 0x04 0xD1 0x02 0x05 0x05 0xF2 means that

Headset ID 0x0505 could not be found.

* **Diconnected** – acest status este trimis ca răspuns dacă RF Dongle-ul s-a deconectat de la EEG TGAM Chip **[4]**:

DataRow Format: (Headset Disconnected Code) (Data Length) (2 byte Global Headset ID)

Example Packet: 0xAA 0xAA 0x04 0xD2 0x02 0x05 0x05 0x21 means that

the dongle disconnected from headset with Global

Headset ID 0x0505.

* **Request Denied** – acest status este trimis ca răspuns dacă RF Dongle-ul nu acceptă ultimul byte din comanda de **Connect**/**Auto-connect** trimis de către aplicație **[4]**:

DataRow Format: (Request Denied Code) (Data Length 0)

Example Packet: 0xAA 0xAA 0x02 0xD3 0x00 0x2C means that the last

command request was denied.

* **Dongle in Standby Mode** – acest status este trimis de către RF Dongle o dată pe secundă când Dongle-ul este în modul Standby **[4]**:

DataRow Format: (Standby/Scan Mode Code) (Data Length 1) (0x00)

Example Packet: 0xAA 0xAA 0x03 0xD4 0x01 0x00 0x2A means that the headset is in standby mode

awaiting for a command.

* **Dongle is trying to find an EEG TGAM set** – acest status este trimis o dată pe secundă cât timp Dongle-ul încearcă să se conecteze și caută un EEG TGAM set **[4]**:

DataRow Format: (Standby/Scan Mode Code) (Data Length 1) (0x01)

Example Packet: 0xAA 0xAA 0x03 0xD4 0x01 0x01 0x29 means that the headset is trying to establish a

connection with a headset.

Lungimea acestor statusuri în bytes este **[4]**:

|  |  |  |
| --- | --- | --- |
| Status (cod) | lungime (bytes) | definitie |
| 0xD0 | **3** | Connect Success |
| 0xD1 | **2** | Not Found |
| 0xD2 | **3** | Disconnected |
| 0xD3 | **0** | Request Denied |
| 0xD4 | **1** | Standby/Scan Mode |

Valorile unui pachet care conține date procesabile într-un pachet de la EEG TGAM Chip pot fi de mai multe tipuri:

* **POOR\_SIGNAL Quality:**

Acesta este un byte de tip *unsigned integer* care descrie cât de bun este semnalul datelor EEG citite de către chip. Are valori de la 0 la 255 și orice valoare diferită de zero arată existența unui fel de zgomot în conexiune. Cu cât numărul este mai mare, cu atât este mai mult zgomot și prin urmare calitatea semnalului este mai slabă. Valoarea exactă 200 are o însemnătate specială, aratând că electrozii nu fac contact cu pielea.

Această informație este redată o dată pe secundă și arată cât de slab este semnalul. Acesta poate fi slăbit datorită mai multor factori:

* + Electrozii ***Sensor***, ***Ground*** și/sau ***Reference*** au un contact slab cu pielea sau nu ating deloc pielea.
  + Mișcarea electrozilor pe piele, dacă de exemplu purtătorul se mișca brusc o perioadă îndelungată.
  + Electricitate electrostatică de nivel ridicat în zona chip-ului.

Un anumit nivel de zgomot este de neevitat în folosirea normală a chip-ului, dar pentru îmbunătățirea semnalului sunt implementate pe chip doua tehnologii, respectiv algoritmi de corectare: *NeuroSky Filter* si *eSense*.

Această informație este activată default și redată o dată pe secundă.

* **eSense™ Meters:**

eSense™ redă anumite informații deja procesate de un algoritm intern al chip-ului cum ar fi: nivelul de ***meditație***, nivelul de ***atenție*** și ***detectarea și puterea unei clipiri***. Acestea sunt redate pe o scală relativă de 1 la 100. Pe această scală o valoare între 40 și 60 în orice moment este considerată „neutră” și este similară cu linia de bază stabilită în convențiile tehnicilor de măsurare EEG. O valoare între 60 și 80 este considerată ușor crescută și poate fi interpretată ca nivel mai mare decât normal. Valori între 80 și 100 sunt considerate crescute și reprezintă nivele ridicate de eSense™.

Similar, pe partea cealaltă a scalei, o valoare între 20 și 40 reprezintă nivele scăzute de eSense™ și poate fi un indicator de atenție distrasă, agitație sau anormalitate.

O valoare care reprezintă 0 este specială și arată faptul ca eSense™ nu poate calcula un nivel destul de înalt de siguranță a informației. Acest lucru poate fi cauzat (și de obicei este) de un nivel înalt de zgomot și ar putea fi verificat din informația POOR\_SIGNAL Quality.

Motivul pentru exitența acestei game oarecum largă de valori este faptul că unele parți ale algoritmului eSense™ au învățare dinamică și uneori pot folosi algoritmi „slow-adaptive” pentru a ajusta fluctuațiile și trendurile naturale ale fiecărui utilizator. Acestea iau în calcul și compensează pentru faptul că semnalul EEG este supus in creierul uman unor variații si fluctuații naturale. Acesta este motivul pentru care senzorii și chip-ul EEG TGAM pot citi de la o scara largă de persoane și în medii înconjurătoare diferite păstrând un nivel înalt de acuratețe și siguranță.

* **ATTENTION eSense™:**

Un byte de tip *unsigned integer* raportează valoarea curent interpretată a atenției unui utilizator care indică nivelul mintal de „concentrare” sau „atenție” care este atins în stări de concentrare și activitate mintală direcționată și stabilă. Valorile sunt între 0 si 100. Atenție distrasă, gânduri fugitive, lipsă de concentrare sau stări de anxietate, pot cauza scăderea nivelului acestei valori.

Această informație este activată default și redată o dată pe secundă.

* **MEDITATION eSense™:**

Un byte de tip *unsigned integer* raportează valoarea curent interpretată de meditație a unui utilizator care indică nivelul mintal de „calm” sau „relaxare”. Valorile sunt între 0 si 100, iar acestea indică nivelul mintal de relaxare, nu pe cel fizic.

Meditația presupune reducerea activității proceselor mintale active. A fost dovedit că, de exemplu, închiderea ochilor cauzează oprirea procesării de imagine în creier, așa că închiderea ochilor este o metodă eficientă de a crește nivelul stării de meditație.

Această informație este activată default și redată o dată pe secundă.

* **8BIT\_RAW Wave Value:**

Valoarea acestui *unsinged integer* de 1 Byte este echivalentul valorii lui **RAW Wave Value (16-bit)** de tip *signed integer* descrisă mai jos, cu excepția faptului că este scalată să fie *unsigned*, și numai cei mai importanți 8 bits sunt redați. Acest lucru ne dă posibilitatea de a avea ca output valori raw ale undelor, având în vedere restricțiile comunicării seriale la baud rate 9600, cu prețul de a nu avea în output și bits-ii de precizie. Pentru majoritatea aplicațiilor (mai ales de ex. afișarea undelor în graph), redarea a 8 bits de precizie este suficientă, pentru că ochiul uman nu poate distinge, în general, destul de rapid, pixel-ii care corespund bits-ilor de precizie specializați (least-significat bits).

Dacă este nevoie de o precizie mai mare, se va folosi output-ul standard/normal **RAW Wave Value (16-bit)** dar este nevoie de un baud rate mai mare de 9600.

Deși numai cei mai semnificanți 8 bits sunt afișați, atunci când modul 8BIT\_RAW este activ, toate calculele sunt încă făcute în chip-ul EEG TGAM, așa că intern este încă folosită precizia maximă. Numai output-ul va fi de precizie mai mică (8 bits).

Default, output-ul acestui tip de RAW Value (**8BIT\_RAW**) este dezactivat. La fel ca în modul normal **16-bit RAW Wave Value**, valorile **8BIT\_RAW** sunt redate de 128 de ori pe secundă, deci aproximativ o dată la fiecare 7.8 milisecunde.

* **RAW\_MARKER Section Start:**

Acesta nu este cu adevărat un tip de date și este folosit pentru debugging-ul foarte precis al timpilor și sincronizării datelor RAW, sau pentru cercetare. Momentan, valoarea va fi întotdeauna 0x00.

Aceasta informație este dezactivata default și redată o dată pe secundă.

* **RAW Wave Value (16-bit):**

Acest tip de date este format din 2 bytes și reprezintă un set de valori raw al undelor cerebrale redate. Valoarea acestui tip de date este un *16-bit signed integer* de la *-32768* până la *32767*. Primii 4 bytes reprezintă bits-ii de ordin înalt, în timp ce al doilea byte, reprezintă bits-ii de ordin scăzut.

Pentru a reconstrui valoarea raw întreagă, putem face un 8 bit left shift cu primul byte și un bitwise-OR cu al doilea byte:

short raw = (Value[0] << 8) | Value[1];

unde Value[0] este byte-ul high-order, iar Value[1] este byte-ul low-order.

Această informație este dezactivată default și redată de 128 de ori pe secundă, deci aproximativ o dată la fiecare 7.8 milisecunde. Pentru modelele ASIC însă, vor fi redate de 512 ori pe secundă, adică aproximativ o dată la fiecare 2 milisecunde.

Din cauza baud rate-ului mare la care este redat acest tip de output și a numărului mare de bytes redat, este posibil modul **16-bit RAW Wave Value** doar la un baud rate de 57600 sau mai mare la comunicarea serial.

* **EEG\_POWER:**

Acest tip de date reprezintă magnitudinea curentă a celor 8 tipuri de frecvențe ale undelor cerebrale redate. Această valoare este formată din 8 numere de tip *floating point*, de 4 bytes în ordinea următoare:

* + **Delta** (0.5 – 2.75 Hz)
  + **Theta** (3.5 – 6.75 Hz)
  + **Low-Alpha** (7.5 – 9.25 Hz)
  + **High-Alpha** (10 – 11.75 Hz)
  + **Low-Beta** (13 – 16.75 Hz)
  + **High-Beta** (19 – 29.75 Hz)
  + **Low-Gamma** (31 – 39.75 Hz)
  + **Mid-Gamma** (41 – 49.75 Hz)

Aceste valori nu au unități, așa că au sens doar atunci când sunt comparate una cu cealaltă sau cu ele însele, pentru a calcula cantitatea relativă și fluctuațiile temporale. Formatul *floating point* este standard *big-endian* IEEE 754, așa că cei 32 bytes ai valorii pot fi direct transformati (cast) ca *float\** in C (în sisteme *big-endian*) și pot fi apoi folosite ca un vector de *float*-uri.

Această informație este dezactivată default și redată o dată pe secundă.

Acest tip de date nu există în chip-urile ASIC așa că pentru acestea se va folosi tipul de date **ASIC\_EEG\_POWER\_INT**.

* **ASIC\_EEG\_POWER\_INT:**

Acest tip de date reprezintă magnitudinea curentă a celor 8 tipuri de frecvențe ale undelor cerebrale redate și este echivalentul ASIC al tipului de date **EEG\_POWER**, cu diferența majoră că valorile sunt un output al unei serii de 8 *unsigned intergers* de 3 bytes în format *big-endian* în loc de 4 byte *floating points*.

Frecvența celor 8 unde cerebrale redate este în ordinea următoare **[4]**:

* + **Delta** (0.5 – 2.75 Hz)
  + **Theta** (3.5 – 6.75 Hz)
  + **Low-Alpha** (7.5 – 9.25 Hz)
  + **High-Alpha** (10 – 11.75 Hz)
  + **Low-Beta** (13 – 16.75 Hz)
  + **High-Beta** (19 – 29.75 Hz)
  + **Low-Gamma** (31 – 39.75 Hz)
  + **Mid-Gamma** (41 – 49.75 Hz)

Aceste valori nu au unități așa că au sens doar când sunt comparate una cu cealaltă sau cu ele însele, pentru a calcula cantitatea relativă si fluctuațiile temporale.

Această informație este activată default și redată o dată pe secundă.

* **DATA\_BLINK\_STRENGTH:**

Acest tip de date de un byte ca *unsigned integer*, redă intensitatea relativă a clipirii. Valorile acesteia sunt între 1 si 255 și va raporta de fiecare data când utilizatorul va clipi.

* **Mind-wandering Level:**

Acest tip de date de un byte ca *unsigned integer*, redă nivelul de „rătăcire” a minții. Are valori de la 0 la 10.

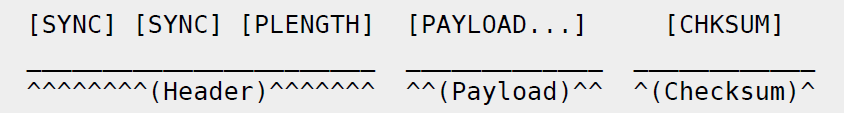
Un pachet de date (ASIC) primit de la EEG TGAM constă dintr-un stream de bytes care trebuie procesat și interpretat pentru a ajunge la valorile de care suntem interesați, în funcție de tipul de date pe care îl citim.

Un pachet este format din 3 parți:

* **Packet Header**
* **Packet Payload**
* **Payload Checksum**

Formatul unui pachet este făcut să fie robust și flexibil. Combinate, Header-ul și Checksum-ul asigură sincronizarea și integritatea datelor, iar Payload-ul asigură că pot fi scoase sau adăugate noi tipuri de date pe viitor, fără a strica funcționalitatea aplicațiilor de procesare existente.

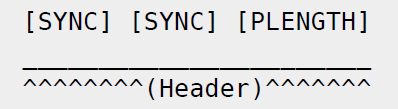
Fiecare pachet începe cu un Header, urmat de Payload și se termină în Checksum **[4]**:



Secțiunea **[PAYLOAD…]** poate avea maxim 169 bytes, în timp ce fiecare **[SYNC]**, **[PLENGTH]** si **[CHKSUM]** au un singur byte. Asta înseamnă că un pachet valid complet, trebuie să aibă minim 4 bytes (există posibilitatea să nu avem [**PAYLOAD…]**, respectiv [**PAYLOAD…]**-ul sa aibă 0 bytes) și lungimea maximă a unui pachet poate fi de 173 bytes (pentru ca [**PAYLOAD…]**-ul poate avea maximum 169 bytes).

* **Packet Header:**

Header-ul unui pachet are 3 bytes: 2 bytes de sincronizare **[SYNC]** (0xAA, 0xAA), urmați de un byte **[PLENGTH]** (lungimea Payload-ului) **[4]**:



Cei 2 bytes **[SYNC]** sunt folosiți pentru a semnala începutul unui nou pachet și au valoarea **0xAA** (170 decimal). Sincronizarea are deci 2 bytes în loc de unul pentru a reduce șansa ca bytes de tip **[SYNC]** să apară în Payload (0xAA = 170 decimal) și să fie folosiți din greșeală în procesul de sincronizare. Deși tot este posibil ca doi bytes cu valoarea 0xAA (de **[SYNC]**) să apară unul după altul în Payload, [PLENGTH] și [CHECKSUM] combinați, garantează că nu vom fi out-of-sync în procesarea unui pachet.

Byte-ul **[PLENGTH]** indicp lungimea în bytes a secțiunii Payload a pachetului curent (**[PAYLOAD…]**) și poate avea orice valoare între 0 si 169. O valoare mai mare de 169, indică o eroare în datele primite sau procesate. Valoarea totală a unui pachet va fi deci întotdeauna **[PLENGTH] + 4**.

* **Data Payload:**

Payload-ul unui pachet este pur și simplu o serie de bytes. Numărul de bytes al Payload-ului curent este dat de către byte-ul **[PLENGTH]** din Header. Interpretarea Payload-ului depinde de tipul de date pe care îl primim.

Este important de notat faptul ca nu ar trebui sa înceapă interpretarea pachet-ului Payload înainte de a verifica integritatea și validitatea pachetului folosindu-ne de ultimul byte din pachet, respectiv **[CHKSUM]**.

* **Payload Checksum:**

Byte-ul [CHECKSUM] este ultimul byte al unui pachet și trebuie folosit pentru a valida integritatea pachetului. Checksum-ul este definit astfel:

1. Suma tuturor bytes din **[PAYLOAD]**
2. Luăm cei mai de jos 8 bits din aceasta sumă
3. Facem bit-inverse (*NOT*) pe acest 8 bits

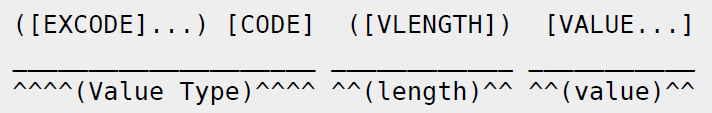
O aplicație care primește pachete trebuie să folosească acești 3 pași pentru a valida integritatea pachetului cu ajutorul valorii byte-ului **[CHKSUM]**. Dacă valoare calculată nu este egală cu valoarea din **[CHKSUM]**, atunci pachetul ar trebui lăsat și ar trebui trecut înapoi la sincronizare. Dacă acestea sunt egale și deci pachetul este valid, atunci se trece la procesarea Payload-ului.

* **Data Payload Structure:**

O dată ce a fost verificată integritatea pachetului, putem trece la procesarea Payload-ului. Acesta este alcătuit dintr-o serie de Date de un anumit tip (descrise mai sus), fiecare conținând o serie de bytes numita un **DataRow**.

Un **DataRow** conține informație despre ce tip de date am primit, lungimea acestor date și (bytes) datele efective. Pentru a procesa un Payload, trebuie deci să procesam fiecare **DataRow** în parte pânî când toți bytes din Payload au fost procesați.

**DataRow**-ul are următorul format **[4]**:

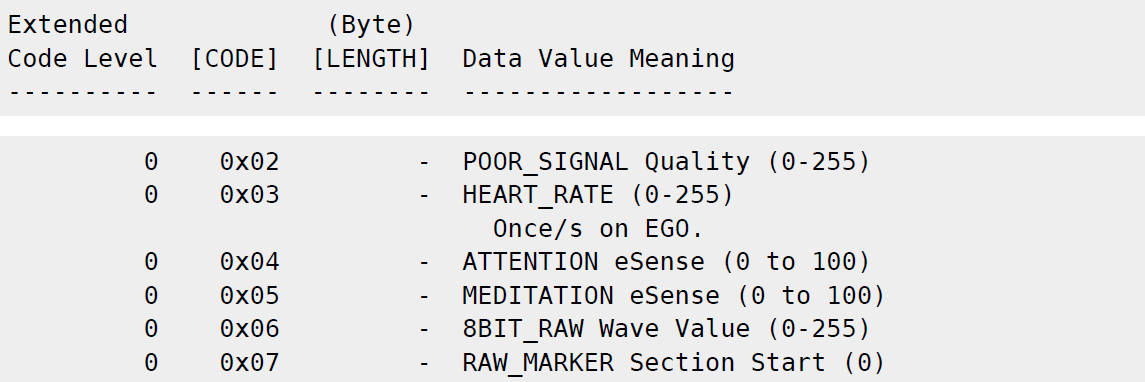


Valorile trecute în paranteză „**(**[*VAL*]**)**” sunt condiționali, ceea ce înseamnă că pot apărea în anumite **DataRows**, dar nu neapărat și în altele.

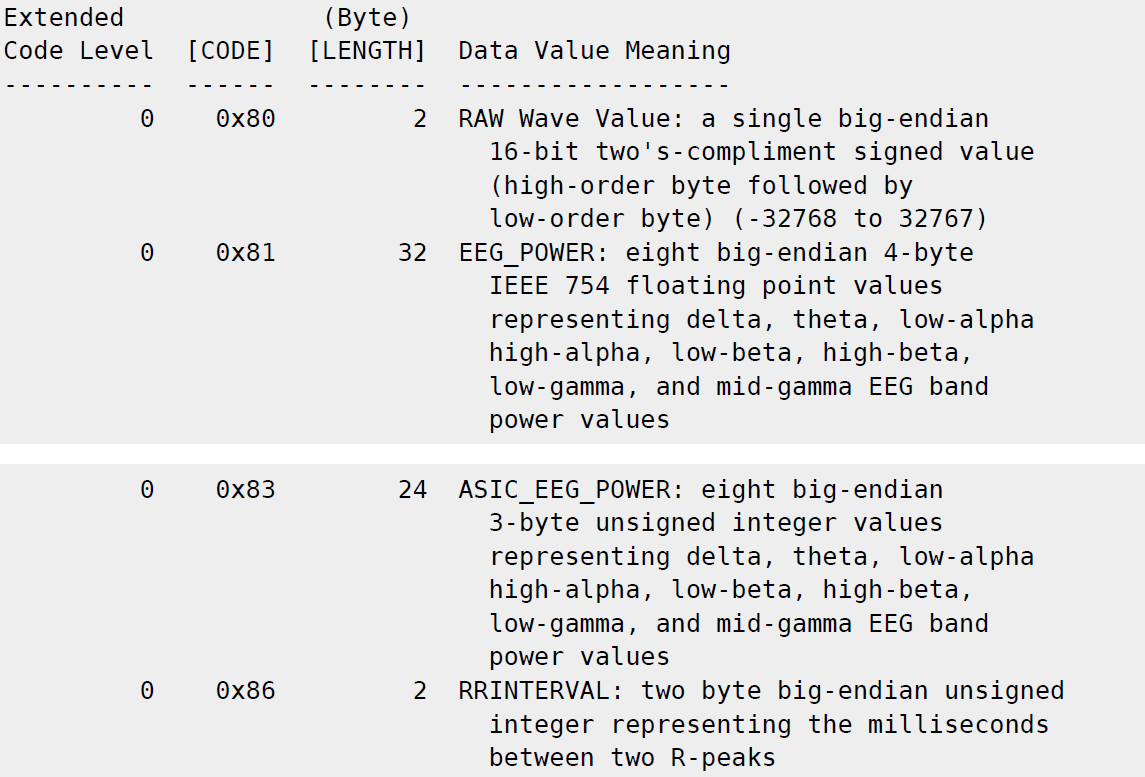
Un DataRow poate începe cu zero sau mai mulți bytes de tip **[EXCODE]** (***Ext***ended ***Code***) care sunt bytes cu valoarea **0x55**. Numărul de bytes **[EXCODE]** (**0x55**) indică nivelul codului extins (*Extended Code Level*). Acesta este folosit cu byte-ul **[CODE]** pentru a determina ce tip de date este conținut în acest **DataRow**. De exemplu, dacă avem *Extended Code Level* = **0** si **[CODE]** = **0x04**, înseamnă că acest **DataRow** conține tipul de date **eSense™ Attention**. Lista întreagă de interpretare poate fi regăsită mai jos în ***CODE Definition Table***. A se nota că valoarea **0x55,** care reprezintă **[EXCODE]** nu va fi niciodată folosită ca valoare pentru **[CODE]**. Același lucru este valabil și pentru valoarea **0xAA** (**[SYNC]**). Dacă **[CODE]** are valori de bytes între **0x00** si **0x7F**, atunci lungimea valorii [VALUE…] va avea un byte (*Single-Byte Value*). Dacă însă **[CODE]** are o valoare mai mare de **0x7F**, atunci urmează imediat după, un byte de tip **[VLENGTH]** (*Value Length*) care reprezintă lungimea în bytes a valorii efective **[VALUE…]** (*Multi-Byte Value*). Aceste coduri de tip **[CODE]** de valori mari (**> 0x7F**) sunt folosite pentru a transmite vectori de valori sau valori care nu au loc într-un singur byte.

* **CODE Definitions Table:**

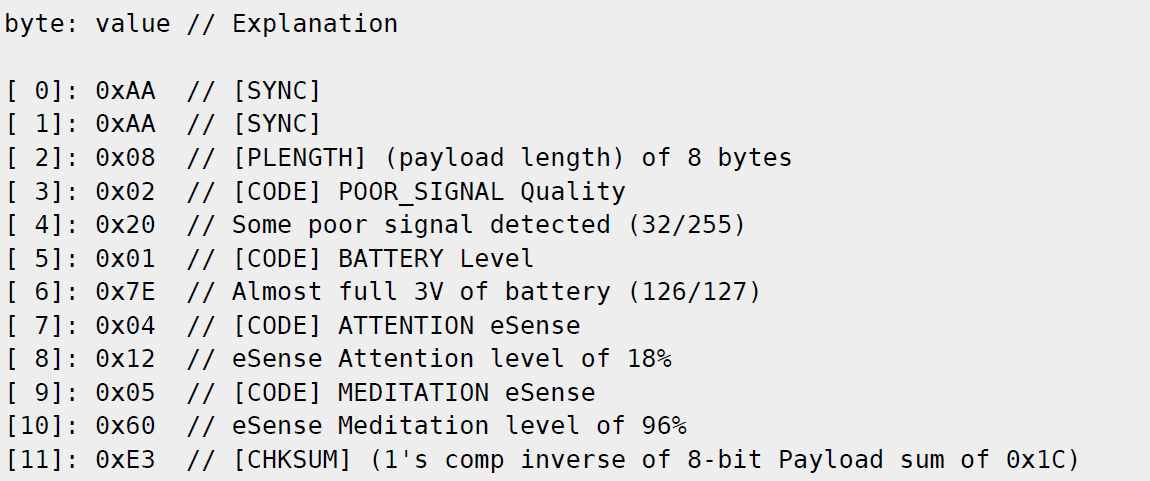
1. **Single-Byte CODEs [4]:**



1. **Multi-Byte CODEs [4]:**



Un exemplu de pachet ar putea arăta ca mai jos. În afara de bytes-ii **[SYNC]**, **[PLENGTH]** și **[CHKSUM]**, toți ceilalți bytes, ([3] pâna la [10]) fac parte din **[PAYLOAD…]**. A se nota că **DataRows** din Payload, nu vor apărea garantat în fiecare Payload **[4]**:



* **Procesarea unui pachet:**

1. Se citesc bytes până când un byte de tip **[SYNC]** (**0xAA**) este citit
2. Se citește următorul byte și se asigură că este tot de tip **[SYNC]** (**0xAA**)
   1. Dacă nu este tot de tip **[SYNC]** ne întoarcem la pasul **1.**
   2. Daca este, atunci continuăm cu pasul **3.**
3. Se citește următorul byte și se interpretează ca fiind **[PLENGTH]** (Payload Length)
   1. Dacă **[PLENGTH]** este 170 repetăm pasul **3.** (170 = 0xAA => **[SYNC]**)
   2. Dacă **[PLENGTH]** este mai mare de 170, atunci sărim înapoi la pasul 1. pentru că înseamnă că pachetul nu este valid (PLENGTH too large error).
   3. Altfel, continuăm cu pasul **4.**
4. Se citește un număr de bytes egal cu **[PLENGTH]** din următorii bytes care vor fi **[PAYLOAD…]-**ul salvându-i în memorie (de ex. intr-un vector unsigned char payload[168]) și sumăm valoarea fiecărui byte (de ex. checksum += byte).
5. Se iau cei mai de jos (least-significant bits) 8 bits din această sumă (checksum) și îi inversăm (*NOT* => checksum = ~checksum & 0xFF).
6. Se citește următorul (ultimul) byte care reprezintă **[CHKSUM]** și se compară cu valoarea calculată la pasul anterior (checksum).
   1. Dacă valorile nu sunt egale, sărim la pasul **1.** (CHKSUM failed error).
   2. Altfel, vom începe procesarea Payload-ului, împărțit în DataRows pentru a obține datele efective.
7. După procesarea Payload-ului, se repetă procesul de la pasul **1.**

* **Procesarea unui DataRow:**

Se vor executa următorii pași atâta timp cât toți bytes (indicat de către **[PLENGTH]**) din Payload au fost procesați:

1. Se citește fiecare byte pe rând atâta timp cât acesta este de tip **[EXCODE]** (**0x55**) și se numără de câte ori a apărut pentru a determina nivelul codului de extensie (*Extended Code Level*).
2. Se citește următorul byte care va fi de tip **[CODE]**.
3. Daca **[CODE] >= 0x80**, atunci se citește următorul byte si îl interpretăm ca **[VLENGTH]** (*Value Length*).
4. Se citesc și se procesează nu număr egal cu **[VLENGTH]** de bytes care vor reprezenta **[VALUE…]**, ținând cont de nivelul **[EXCODE]** și valoarea **[CODE]**.
5. Dacă nu au fost interpretați toți bytes din vectorul payload[168], atunci se sare la pasul **1.** și se continuă procesarea.

* **EXEMPLU cod de procesare in C [4]:**

#include <stdio.h>

#define SYNC 0xAA

#define EXCODE 0x55

int parsePayload(unsigned char \*payload, unsigned char pLength) {

unsigned char bytesParsed = 0;

unsigned char code;

unsigned char length;

unsigned char extendedCodeLevel;

int i;

/\* Loop until all bytes are parsed from the payload[] array... \*/

while(bytesParsed < pLength) {

/\* Parse the extendedCodeLevel, code, and length \*/

extendedCodeLevel = 0;

while(payload[bytesParsed] == EXCODE) {

extendedCodeLevel++;

bytesParsed++;

}

code = payload[bytesParsed++];

if(code & 0x80) length = payload[bytesParsed++];

else length = 1;

// TODO: Handle the next "length" bytes according to [CODE] Definitions Table

printf("EXCODE level: %d CODE: 0x%02X length: %d\n",

extendedCodeLevel, code, length);

printf("Data value(s):");

for(i = 0; i<length; i++) {

printf(" %02X", payload[bytesParsed + i] & 0xFF);

}

printf("\n");

/\* Increment the bytesParsed by the length of the Data Value \*/

bytesParsed += length;

}

return(0);

}

int main(int argc, char \*\*argv) {

int checksum;

unsigned char payload[256];

unsigned char pLength;

unsigned char c;

unsigned char i;

/\* TODO: Initialize 'stream' here to read from a serial data

\* stream, or whatever stream source is appropriate for your

\* application. See documentation for "Serial I/O" for your

\* platform for details.

\*/

FILE \*stream = 0;

stream = fopen("COM4", "r");

/\* Loop forever, parsing one Packet per loop... \*/

while(1) {

/\* Synchronize on [SYNC] bytes \*/

fread(&c, 1, 1, stream);

if(c != SYNC) continue;

fread(&c, 1, 1, stream);

if(c != SYNC) continue;

/\* Parse [PLENGTH] byte \*/

while(true) {

fread(&pLength, 1, 1, stream);

if(pLength ~= 170) break;

}

if(pLength > 169) continue;

/\* Collect [PAYLOAD...] bytes \*/

fread(payload, 1, pLength, stream);

/\* Compute [PAYLOAD...] chksum \*/

checksum = 0;

for(i = 0; i<pLength; i++) checksum += payload[i];

checksum &= 0xFF;

checksum = ~checksum & 0xFF;

/\* Parse [CKSUM] byte \*/

fread(&c, 1, 1, stream);

/\* Verify [PAYLOAD...] chksum against [CKSUM] \*/

if(c != checksum) continue;

/\* Since [CKSUM] is OK, parse the Data Payload \*/

parsePayload(payload, pLength);

}

return(0);

}

* **Command Bytes:**

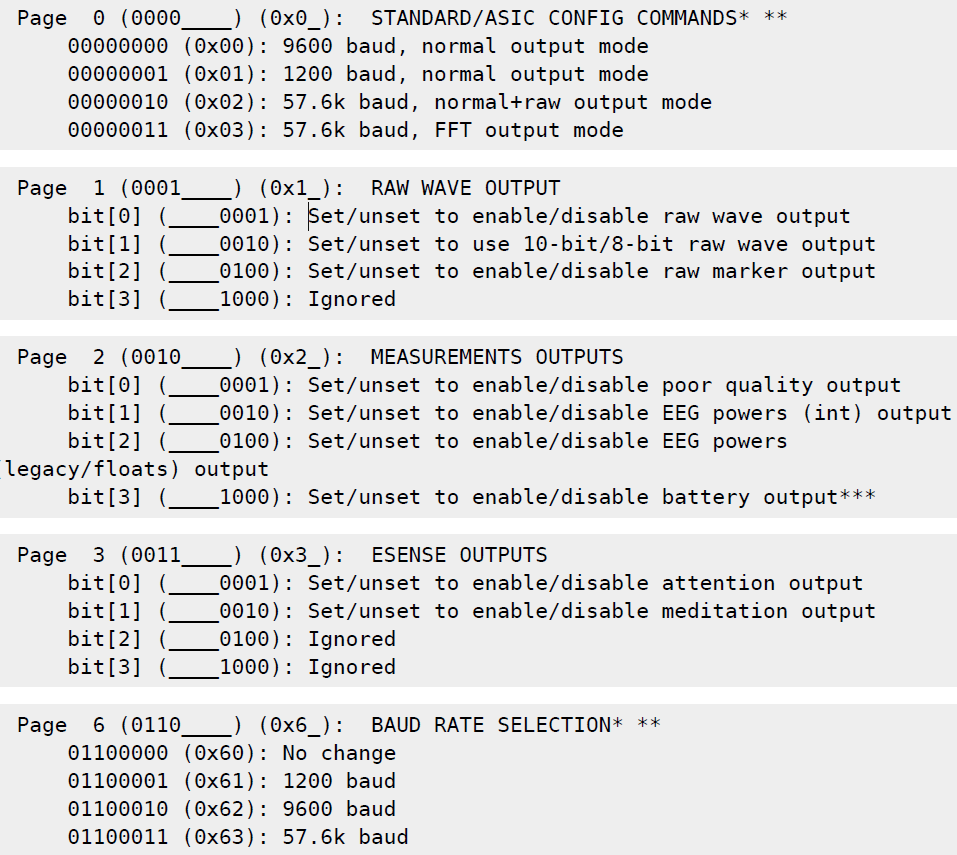
La pornire, chip-ul EEG TGAM ASIC va porni mereu cu setările default. Putem trimite însă anumiți bytes de configurare, fiecare fiind de un byte, unde fiecare dintre cei 8 bits reprezintă o anumită setare care poate fi activată (1) sau dezactivată (0) în acest fel. O dată retratat, setările revin la cele default.

Din cei 8 bits, primii 4 bits de jos (least-significant bits) reprezintă setări pentru a controla baud rate, output-ul pentru eSense™ Attention activat sau dezactivat, etc., în timp ce, cei 4 bits de sus (most-significant bits) reprezinta pagina de comanda (*Command Page*) si definesc sensul celor 4 bits de jos (least-significant bits).

De exemplu un byte de comandă cu valoarea **0x0E** are valoarea **0000** **1110** în bits. În cazul acesta bits-ii de sus (most-significant bits) sunt **0000** și se referă deci la pagina **0** de comandă.

Dacă ne uităm în tabelul paginilor de comandă (mai jos la **Command Byte Table**), putem vedea că bits-ii de jos (least-significant bits) care aici au valoarea **1110**, sunt folosiți pentru a controla modul de output al chip-ului.

* **Command Byte Table [4]:**



**\*** Daca schimbam baud rate, atunci trebuie sa închidem conexiunea din aplicație si sa o redeschidem la baud rate-ul nou configurat.

## NodeJS Web Server backend (cu JXCore si Node-FFI)

NodeJS este un mediu de rulare cross-platform, open-source, pentru aplicații server. Limbajul de programare este JavaScript si aplicațiile scrise pot rula sub NodeJS pe Windows, Linux, MAC OS X, FreeBSD, NonStop si IBM i.

Ceea ce este special la NodeJS, este faptul ca are o arhitectura bazata pe evenimente care sunt non-blocking. Acest lucru face aplicațiile sa fie mai optime si scalabile. Aceste tehnologii sunt folosite pentru aplicații web real-time.

NodeJS folosește Google V8 ca motor de execuție JavaScript si o mare parte din modulele de baza sunt scrise in JavaScript. Pentru ca are si librarii care pot acționa ca Web Server, nu mai este nevoie de o platforma de Web Server in plus cum ar fi IIS, Apache, HTTP Server, etc.

Cu NodeJS se pot face servere web si aplicații de rețea folosind JavaScript si o serie de module care au diferite funcționalități de baza. Câteva dintre aceste sunt:

* File system I/O
* Networking (HTTP, TCP, UDP, DNS, TLS/SSL)
* Binary data (buffers)
* Criptograpy
* Data streams

Mai sunt si alte module existente si se pot adaugă si altele noi daca este nevoie. Acestea au in API simplu si elegant, așa încât timpul si complexitatea de a scrie aplicații sunt mult reduse.

NodeJS este folosit principal pentru servere web si este similar cu PHP sau Python. Spre diferența de PHP însă, limbajul este non-blocking, deci metodele sunt executate in paralel si folosesc funcții de callback pentru a semnala completarea execuției. Tipul de programare este bazat pe evenimente, așa încât se pot crea aplicații scalabile fără a fi nevoie de threading si a fost creat pentru ca concurenta (concurrency) este mai dificil de programat in majoritatea limbajelor de programare server-side (web) si de multe ori influențează performanta de executare negativ.

NodeJS este construit pe motorul Google V8 JavaScript, pentru ca:

* V8 este open-source sub licența BSD
* V8 este foarte rapid
* V8 este gândit pentru web, așa încât este specializat pe fundamente de retelistica
* JavaScript este un limbaj de programare bine cunoscut

Detalii tehnice:

* **Threading:**

NodeJS rulează sub un singur thread, folosind apeluri non-blocking, ceea ce permite zeci de mii de conexiuni concurente fără nevoia si costurile de thread context-switching. Designul de a împarți același thread intre toate cererile, înseamnă ca poate fi folosit pentru a crea aplicații de concurenta înalta. Orice funcție care va face I/O trebuie sa implementeze un callback.

Dezavantajul poate apărea din cauza faptului ca nu se scalează după numărul de CPU-uri pe care îl are un sistem, si nu se poate nici configura pentru disponibilitate mare, de exemplu prin folosirea unui cluster.

* **V8:**

V8 este motorul de execuție JavaScript creat pentru Google Chrome si făcut open-source de către Google. Este scris in C++ si compilează cod JavaScript direct in cod nativ in loc sa îl interpreteze real-time.

NodeJS folosește **libuv** pentru a trata evenimente asincrone si este un strat de abstractizare pentru retelistica si sisteme de fișiere pentru atât pentru Windows, cat si pentru sisteme bazate pe POSIX ca Linux, Mac OS X si Unix.

Funcționalitățile de baza sunt scrise in JavaScript, iar legăturile dintre acestea sunt scrise in C++ pentru a conecta diferitele module si pentru a interacționa cu sistemul de operare.

* **Package management:**

**NPM** este managerul de pachete pre-instalat cu NodeJS si este folosit pentru a instala module pentru NodeJS din registrul sau. Prin organizarea instalării si administrarea pachetelor 3rd party, ajuta programatorii sa compileze rapid.

* **Unified API:**

NodeJS combinat cu un browser, o baza de date (ca MongoDB sau CouchDB) si JSON, oferă o stiva de dezvoltare unificată. Concentrându-se pe framework-uri client-side si adoptând modele de programare de tip server-side, ca MVC, MVP sau MVVM, NodeJS oferă posibilitatea de a refolosi același model si interfața de serviciu intre client-side si server-side.

* **Event loop:**

NodeJS se înregistrează cu sistemul de operare, așa încât este notificat când o conexiune este creata. In aceasta situație sistemul de operare va apela un callback. In NodeJS, fiecare conexiune este o mica alocare de memorie din heap. In mod tradițional, procese si/sau thread-uri relativ de complexitate mai mare se ocupa de fiecare conexiune, dar NodeJS folosește un **Event loop** in loc de procese sau thread-uri, pentru a scala milioane de conexiuni in același timp. Contradictoriu cu alte aplicații condusă de evenimente, Event loop-ul nu trebuie apelat explicit, ci exista callback-uri definite si server-ul intra automat in acest loop la sfârșitul definiției callback-ului si iese din acest loop când nu mai sunt alte callback-uri care trebuie executate.

Pentru a dezvolta partea server-side a aplicației care primește datele de la driver-ul care citește si procesează undele cerebrale de la chip-ul EEG TGAM, a fost nevoie de doua module noi pentru NodeJS:

* **Node-FFI:**

Node-FFI este un modul pentru NodeJS, iar numele sau vine de la *Foreign Function Interface*. Acesta poate fi folosit pentru a încarcă librarii scrise in C sau C++ si pentru a apela funcții din aceste librarii.

Deși se putea folosi si NodeJS pentru a citi date serial de pe USB, nu avea rost sa rescriem funcționalitatea driver-ului in JavaScript pentru NodeJS. Cea mai buna soluție deci, era sa apelam funcția din driver care returnează informația undelor cerebrale in tip JSON, direct din JavaScript in NodeJS.

Pentru a încărca o librarie cu Node-FFI, putem face ca in acest exemplu:

var ffi = require('ffi');

var lib = ffi.Library('libNAME', {

'myFunction': ['string', ['double']]

});

In acest exemplu, presupunem ca libraria se chiama “libNAME” si functia pe care vrem sa o importam si folosim se chiama “myFunction”, primeste un parametru de tip **double** si are ca return type **string**. Desi in C, tipul este **char\***, echivalentul sau in JavaScript este **string**, asa ca asta trebuie definit in acest caz.

Pentru a apela functia, vom face ca in acest exemplu:

var myStringValFromC = lib.myFunction(3.1415);

Acest apel, va stoca sirul de caractere returnat de functia *myFunction* in variabila *myStringValFromC*.

* **JXcore:**

JXcore este o extensie a NodeJS-ului care permite mai multe thread-uri pentru NodeJS. Funcționează practic prin crearea a mai multor procese având ca baza NodeJS, care pot avea o memorie pe care o împart si care pot comunica intre ele.

A fost nevoie de acest lucru, pentru ca funcția care este apelata pentru a primi datele de la driver folosește un loop pentru a citi datele. Fiind o librarie, aceasta trebuie sa stea mereu încărcata si sa execute codul din loop care citește undele cerebrale si le procesează.

Ca sa fie blocata execuția de apelul la funcția de citire din driver prin Node-FFI (pentru ca aceasta va rula un loop) si pentru ca avem nevoie ca librăria sa rămână mereu încărcata si sa adune date, a fost nevoie de un thread separat. Din acest motiv, am folosit JXcore pentru a atinge acest scop.

Pentru ca vom porni thread-uri separate, fiecare trebuie sa fie definit ca o funcție JavaScript care sa aibă propriile variabile pentru ca variabilele locale nu pot fi împărțite/accesate intre thread-urile (respectiv, procesele) separate.

Acesta ar fi un exemplu pentru a porni 2 thread-uri separate (vor fi de fapt procese):

var myFirstFunction() {

console.log("Hi, I'm the first function running in my own process/thread!");

}

var mySecondFunction() {

console.log("Hi, I'm the second function running in my own process/thread!");

}

jxcore.tasks.runOnThread(0, myFirstFunction);

jxcore.tasks.runOnThread(1, mySecondFunction);

Daca este însă (si de obicei este) nevoie de date care sa fie accesibile de către thread-uri diferite, atunci JXcore are si metode speciale de a crea si acest fel de variabile care sunt stocate in memorie împărțita de către thread-uri (respectiv, procese).

JScore are un Memory Store, care implementează obiecte **jxcore.store** si **jxcore.store.shared**. Acestea pot fi folosite pentru a putea avea on loc central de memorie care poate fi împărțit intre diferitele procese si thread-uri. **jxcore.store.shared** se va folosi pentru a avea variabile care pot fi accesate de procese/thread-uri diferite si are metode simple pentru a stoca si sau prelua valori din aceste variabile: *set(…)*, *get()*, *exists(…)*, etc.

Datele sunt însă ținute ca string așa încât, daca este nevoie sa stocam structuri de date complexe, trebuie sa le serializam la scriere si sa le deserializam la citire. Un exemplu simplu pentru a stoca si citi un string pe care îl vom pune într-o variabila numita „**myVar**” ar fi așa:

var myString = "This is a test string for this demo.";

jxcore.store.shared.set("myVar", myString); // stocam valoarea

jxcore.store.shared.get("myVar"); // citim valoarea

## Librăria D3.js si AJAX cu JQuery.js pentru JavaScript frontend

Pentru partea client-side / web front end, respectiv JavaScript-ul care rulează in browser într-o pagina HTML, am folosit librăria **D3.js** pentru ca aceasta permite generarea simpla de date vizuale interactive.

**D3.js** (*Data-Driven Documents*) este o librărie JavaScript care ajuta la crearea de date vizuale dinamice si interactive. Folosește tehnologii standard HTML5, CSS si SVG.

Intr-o pagina HTML, librăria D3.js foloseste functii JavaScript prefabricate pentru a selecta elemente, a crea obiecte SVG (*Scalable Vector Graphics*), a le stiliza si/sau a le adăuga tranziții si/sau efecte acestora. Seturi mari de date pot fi legate cu ușurința in obiecte SVG folosind funcții simple D3.js pentru a crea diagrame interactive.

* **Selectors:**

Design-ul central al D3.js este făcut așa încât sa dea posibilitatea programatorilor de a selecta un anumit element DOM (*Document Object Model*) prin metode de tip CSS-selector, si a manipula aceste obiecte o data obținute.

De exemplu, pentru a selecta toate elementele ***<p>…</p>*** dintr-o pagina, si pentru a le schimba culoarea textului, adăuga o noua clasa CSS si a seta poziția orizontala, putem face așa **[12]**:

d3.selectAll("p") // select all <p> elements

.style("color", "lavender") // set style "color" to value "lavender"

.attr("class", "squares") // set attribute "class" to value "squares"

.attr("x", 50); // set attribute "x" (horizontal position)

* **Transitions:**

Prin declararea unei tranziții, valorile atributelor si a stilurilor pot for interpolate pe o anumita perioada de timp. Următorul exemplu va face toate elementele ***<p>…</p>*** dintr-o pagina sa își schimbe gradual culoarea din culoare curenta, in verde **[12]**:

d3.selectAll("p") // select all <p> elements

.transition("trans\_1") // transition with name "trans\_1"

.delay(0) // transition starting 0ms after trigger

.duration(500) // transitioning during 500ms

.ease("linear") // transition easing progression is linear...

.style("color", "green"); // ... to color:green

* **Data-binding:**

Putem încărca un set de date cu D3.js si pentru fiecare element, sa creăm un obiect SVG cu proprietăți diferite (forme, culori, valori) si comportamente diferite (tranziții, evenimente).

De exemplu o diagrama in forma de cerc care arata informații statistice despre 3 tari diferite (setul de date) **[12]**:

// Data

var data = [

{ name: "Ireland", income: 53000, life: 78, pop: 6378, color: "green" },

{ name: "Norway", income: 73000, life: 87, pop: 5084, color: "blue" },

{ name: "Tanzania", income: 27000, life: 50, pop: 3407, color: "grey" }

];

// Create SVG container

var svg = d3.select("#hook").append("svg")

.attr("width", 120)

.attr("height", 120)

.style("fill", "#D0D0D0");

// Create SVG elements from data

svg.selectAll("circle") // create virtual circle template

.data(data) // bind data

.enter() // for each row in data...

.append("circle") // bind circle & data row such that...

// set the circle's id according to the country name

.attr("id", function (d) { return d.name })

// set the circle's horizontal position according to income

.attr("cx", function (d) { return d.income / 1000 })

// set the circle's vertical position according to life expectancy

.attr("cy", function (d) { return d.life })

// set the circle's radius according to country's population

.attr("r", function (d) { return d.pop / 1000 \* 2 })

// set the circle's color according to country's color

.attr("fill", function (d) { return d.color });

API-ul D3.js conține sute de funcții predefinite aparținând următoarelor categorii:

* **Selections**
* **Transitions**
* **Arrays**
* **Math**
* **Color**
* **Scales**
* **SVG**
* **Time**
* **Layouts**
* **Geography**
* **Geometry**
* **Behaviors**

Pentru a face apel o data la un anumit interval la aplicația server-side pentru a returna un set nou de date, am folosit AJAX (*Asynchronous JavaScript and XML*). AJAX este o tehnologie client-side care poate fi folosita pentru a trimite si primi date asincron de la aplicația server-side fără a reîncărca pagina curenta in browser.

Deși cum spune si numele, AJAX se poate folosi pentru a trimite si primi date in format XML, permite si formatul JSON, care este mai nou si mai folosit. Se prefera formatul JSON pentru ca este mai optim pentru ca structura datelor este mai mica, așa încât avem mai puțin trafic si o procesare mai rapida.

**JQuery.js** este o librărie JavaScript foarte cunoscuta si folosita in dezvoltarea aplicațiilor front-end. O mare parte din funcționalitățile sale, seamănă cu D3.js, respectiv *Selections* si *Transitions*.

Dar unde D3.js folosește la crearea de date vizuale dinamice si interactive, JQuery.js folosește mai mult pentru crearea de interfețe avansate, dinamice, cum ar fi front-end MVC si are multe elemente care pot fi folosite pentru a ușura sau a face mai atractiva interacțiunea vizuala a paginii cu utilizatorul.

Exemple ar fi elemente de tip calendar pentru a selecta o data si/sau ora într-un fel plăcut si simplu pentru un input, dar in același timp, si pentru a respecta un anumit format așteptat de către aplicație.

Printre altele, are funcții prestabilite si pentru tehnologia AJAX. De exemplu putem face un apel AJAX la aplicația server-side (in acest exemplu paginii „example.php”) pentru a trimite anumite variabile si a primi înapoi alte date **[12]**:

$.ajax({

type: "POST",

url: "example.php",

data: "name=Mihai&location=Bogdan"

}).done(function (msg) {

alert("Data Saved: " + msg);

}).fail(function (xmlHttpRequest, statusText, errorThrown) {

alert(

"Your form submission failed.\n\n"

+ "XML Http Request: " + JSON.stringify(xmlHttpRequest)

+ ",\nStatus Text: " + statusText

+ ",\nError Thrown: " + errorThrown);

});

In funcția de callback *done()* din exemplul de mai sus, variabila „**msg**” conține un simplu sir de caractere, dar ar putea la fel de bine sa conțină un XML sau un JSON care ar trebui procesat pentru a extrage datele din acesta. Pentru acest lucru, putem folosi funcția *$ajax.parseJSON(…)* din JQuery.js ca in acest exemplu **[12]**:

var json = '{ "name": "Mihai" }';

var obj = jQuery.parseJSON(json);

// will alert TRUE

alert(obj.name === "Mihai");

## Self-Organizing Map (SOM) application

Aplicația de bază care va face diferite acțiuni în funcție de starea undelor cerebrale va folosi o librărie (shared library) care va primi seturi de date conținând frecvențele undelor cerebrale de la driver library și va clasifica diferitele stări folosind SOM (Self-Organizing Map).

Implementarea SOM este făcută în C++ și compilată tot ca shared library pentru Linux (Ubuntu).

Pentru a clasifica modelele frecvențelor undelor cerebrale voi folosi un SOM bidimensional (2D). Fiecare neuron din rețea este interconectat cu input-ul și are o poziție topologica specifica (x, y în plan) și un vector de greutăți de aceeași dimensiune cu vectorul de input, care va avea 8 dimensiuni (pentru cele 8 unde cerebrale pe care le citim).

SOM nu are nevoie de un rezultat așteptat pentru antrenare pentru că, spre diferență de alte tipuri de rețele neurale, învățarea va fi nesupervizată.

Acolo unde vectorul de greutăți al unui neuron se va potrivi cu vectorul de input zona din rețea este optimizată selectiv pentru a semăna mai mult cu datele clasei din care face parte vectorul de intrare.

De la o distribuire aleatorie de greutăți de început, de-a lungul multelor iterații, SOM se va stabiliza într-un plan cu zone stabile; fiecare zonă devenind un clasificator.

Învățarea implementată decurge prin pașii următori de-a lungul multor iterații:

1. Se inițializează vectorul de greutăți ale fiecărui neuron.
2. Se alege un vector de input din setul de învățare.
3. Fiecare neuron este examinat pentru a calcula care dintre aceștia are vectorul de greutăți cel mai apropiat de vectorul de input. Neuronul câștigător va deveni BMU (*Best Matching Unit*).
4. Se calculează radiusul vecinătății neuronului BMU. Aceasta valoare va fi inițial mai mare (egala cu radiusul rețelei) dar va scădea în timp cu fiecare vector de input din setul de învățare.
5. Se ajustează vectorul de greutăți al fiecărui neuron din vecinătatea neuronului BMU, pentru a-l face sa fie mai asemanator cu vectorul de input. Cu cât un neuron este mai apropiat de neuronul BMU, cu atât mai mult este afectat.
6. Se repetă pașii începând cu pasul 2.

class CNode

{

private:

vector<double> m\_dWeights;

double m\_dPosX;

double m\_dPosY;

public:

CNode(double posX, double posY, int numWeights):

m\_dPosX(posX),

m\_dPosY(posY)

{

for(int w = 0; w < numWeights; w++)

{

m\_dWeights.push\_back(RandFloat());

}

}

Putem a determina neuronul câștigător (BMU) vom itera prin toți neuronii și vom calcula distanța Euclidiana dintre vectorul de greutate al neuronului și vectorul de input. Neuronul cel mai apropiat va fi BMU:

double CNode::GetEucDistance(const vector<double> &vecInput)

{

double distance = 0;

for(int i = 0; i < m\_dWeights.size(); i++)

{

distance += (vecInput[i] - m\_dWeights[i]) \*

(vecInput[i] - m\_dWeights[i]);

}

return sqrt(distance);

}

Vom determina vecinătatea neuronului BMU calculând cât ar trebui să fie radiusul vecinătății în momentul acestei iterații apoă aplicînd teorema lui Pythagoras. Pentru că este necesar să micșorăm aria vecinătății la fiecare iterație va trebui să micșoram radiusul folosind o funcție de scădere exponențială:

m\_dMapRadius = max(constWindowWidth, constWindowHeight) / 2;

m\_dTimeConstant = m\_iNumIterations / log(m\_dMapRadius);

m\_dNeighbourhoodRadius = m\_dMapRadius \*

exp(-(double)m\_iIterationCount /

m\_dTimeConstant);

Următorul pas este sa ajustăm vectorul de greutăți al fiecărui neuron din vecinătatea neuronului BMU (inclusiv BMU) folosind tot o funcție de scădere exponențială, dar pentru că vectorul de greutăți trebuie ajustat mai mult, cu cât se apropie mai mult de BMU și ajustat mai puțin cu cât se îndepărtează de BMU, variabila de învățare va fi ajustată folosind o funcție de scădere Gaussiană:

bool CSom::Learn(const vector<vector<double>> &pVecData)

{

if(vecData[0].size() != constSizeOfInputVector) return false;

if(m\_bDone) return true;

if(--m\_iNumIterations > 0)

{

int vecCurrent = RandInt(0, data.size() - 1);

m\_pWinningNeuron = FindBestMatchingNeuron(pVecData[vecCurrent]);

m\_dNeighbourhoodRadius = m\_dMapRadius \*

exp(-(double)m\_iIterationCount / m\_dTimeConstant);

for(int n = 0; n < m\_SOM.size(); n++)

{

double DistToNeuronSq = (m\_pWinningNeuron->GetPosX() - m\_SOM[n].GetPosX()) \*

(m\_pWinningNeuron->GetPosX() - m\_SOM[n].GetPosX()) +

(m\_pWinningNeuron->GetPosY() - m\_SOM[n].GetPosY()) \*

(m\_pWinningNeuron->GetPosY() - m\_SOM[n].GetPosY());

double WidthSq = m\_dNeighbourhoodRadius \* m\_dNeighbourhoodRadius;

if(DistToNeuronSq < (m\_dNeighbourhoodRadius \* m\_dNeighbourhoodRadius))

{

m\_dInfluence = exp(-(DistToNeuronSq) / (2 \* WidthSq));

m\_SOM[n].AdjustWeights(pVecData[vecCurrent],

m\_dLearningRate,

m\_dInfluence);

}

}

m\_dLearningRate = constStartLearningRate \*

exp(-(double)m\_iIterationCount / m\_iNumIterations);

m\_iIterationCount++;

}

else

{

m\_bDone = true;

}

return true;

}

# Modul de funcționare a sistemului si rezultate obținute

## Preluarea si procesarea undelor cerebrale

Pentru citirea semnalului de la RF Dongle, am creat o aplicație in C, care se conectează la USB serial si citește pachetele de date. Conexiunea se face in funcția **initialize(…)** care este apelata la încărcarea librăriei automat, de către funcția **libmain(…)** care va porni un thread separat.

void \_\_attribute\_\_ ((constructor)) libmain(void);

A fost nevoie sa pornesc un thread separat, pentru ca as vrea ca funcția de callback sa nu blocheze thread-ul de citire si in plus, fiind ca apelez funcția care ne returnează date din alt proces, aceea trebuie sa poată citi într-un thread separat.

void libmain() {

pthread\_t tid = NULL;

int err = pthread\_create(&tid, NULL, &initialize, NULL);

if(err != 0) printf("\ncan't create thread :[%s]", strerror(err));

}

Funcția de inițializare deschide o conexiune serial si citește fiecare byte trimis într-un *while loop*. La fiecare iteratie este apelata functia de procesare **ThnkrEegDecoderParse(…)**.

void\* initialize(void\* args) {

eegDataQueue = createQueue();

char conBuf;

dev = open(PORT\_NAME, O\_RDWR | O\_NOCTTY | O\_SYNC);

setInterfaceAttributes(dev, BAUD\_RATE, 0);

conBuf = (char)THNKR\_CODE\_DISCONNECT;

write(dev, (char\*)&conBuf, 1);

sleep(5);

conBuf = (char)THNKR\_CODE\_AUTOCONNECT;

write(dev, (char\*)&conBuf, 1);

sleep(5);

ThnkrEegDecoder parser;

ThnkrEegDecoderInit(&parser, THNKR\_TYPE\_PACKETS, handleDataValueFunc, NULL);

size\_t bufSize = 1 \* sizeof(char);

unsigned char buf = 0;

while(1) {

read(dev, &buf, bufSize);

ThnkrEegDecoderParse(&parser, buf);

}

}

Parsarea pachetului se face cu ajutorul unei structuri ajutătoare care tine cont de statusul primit, starea si momentul de procesare in care suntem actual. După ce am terminat de citit si interpretat un payload întreg, vom apela o funcție de callback, care va stoca datele într-un queue.

typedef struct ThnkrEegDecoder {

unsigned char type;

unsigned char state;

unsigned char lastByte;

unsigned char payloadLength;

unsigned char payloadBytesReceived;

unsigned char payload[256];

unsigned char payloadSum;

unsigned char chksum;

void (\*handleDataValue) (

unsigned char extendedCodeLevel,

unsigned char code,

unsigned char numBytes,

const unsigned char\* value,

void\* customData

);

void\* customData;

} ThnkrEegDecoder;

Folosind aceasta structura, funcția de procesare va citi si interpreta bytes-ii primiți, conform documentației chip-ului. Implementarea este:

int ThnkrEegDecoderParse(

ThnkrEegDecoder\* pParser,

unsigned char byte

) {

int returnValue = 0;

if(!pParser) return -1;

// Pick handling according to current state...

switch(pParser->state) {

// Waiting for SyncByte

case THNKR\_STATE\_SYNC:

if( byte == THNKR\_SYNC\_BYTE ) {

pParser->state = THNKR\_STATE\_SYNC\_CHECK;

}

break;

// Waiting for second SyncByte

case THNKR\_STATE\_SYNC\_CHECK:

if(byte == THNKR\_SYNC\_BYTE) {

pParser->state = THNKR\_STATE\_PAYLOAD\_LENGTH;

} else {

pParser->state = THNKR\_STATE\_SYNC;

}

break;

// Waiting for Data[] length

case THNKR\_STATE\_PAYLOAD\_LENGTH:

pParser->payloadLength = byte;

if(pParser->payloadLength == MAX\_PAYLOAD\_SIZE) {

pParser->state = THNKR\_STATE\_SYNC;

returnValue = -2;

} else if(pParser->payloadLength > MAX\_PAYLOAD\_SIZE) {

pParser->state = THNKR\_STATE\_SYNC;

returnValue = -3;

} else {

pParser->payloadBytesReceived = 0;

pParser->payloadSum = 0;

pParser->state = THNKR\_STATE\_PAYLOAD;

}

break;

// Waiting for Payload[] bytes

case THNKR\_STATE\_PAYLOAD:

pParser->payload[pParser->payloadBytesReceived++] = byte;

pParser->payloadSum = (unsigned char)(pParser->payloadSum + byte);

if(pParser->payloadBytesReceived >= pParser->payloadLength) {

pParser->state = THNKR\_STATE\_CHKSUM;

}

break;

// Waiting for CKSUM byte

case THNKR\_STATE\_CHKSUM:

pParser->chksum = byte;

pParser->state = THNKR\_STATE\_SYNC;

if(pParser->chksum != ((~pParser->payloadSum) & 0xFF)) {

returnValue = -2;

} else {

returnValue = 1;

parsePacketPayload(pParser);

}

break;

// Waiting for high byte of 2-byte raw value

case THNKR\_STATE\_WAIT\_HIGH:

// Check if current byte is a high byte

if((byte & THNKR\_CODE\_CONNECT) == THNKR\_CODE\_RAW\_SIGNAL) {

// High byte recognized, will be saved as parser->lastByte

pParser->state = THNKR\_STATE\_WAIT\_LOW;

}

break;

// Waiting for low byte of 2-byte raw value

case THNKR\_STATE\_WAIT\_LOW:

// Check if current byte is a valid low byte

if((byte & THNKR\_CODE\_CONNECT) == THNKR\_CODE\_LOW\_VALUE) {

// Stuff the high and low part of the raw value into an array

pParser->payload[0] = pParser->lastByte;

pParser->payload[1] = byte;

// Notify the handler function of received raw value

if(pParser->handleDataValue) {

pParser->handleDataValue(

0,

THNKR\_CODE\_RAW\_SIGNAL,

2,

pParser->payload,

pParser->customData

);

}

returnValue = 1;

}

// Return to start state waiting for high

pParser->state = THNKR\_STATE\_WAIT\_HIGH;

break;

// unrecognized state

default:

pParser->state = THNKR\_STATE\_SYNC;

returnValue = -5;

break;

}

// Save current byte

pParser->lastByte = byte;

return returnValue;

}

La momentul potrivit, după ce payload-ul a fost citit in întregime, vom interpreta structura de bytes care conține frecventele undelor cerebrale:

case THNKR\_CODE\_ASIC\_EEG\_POWER\_INT:

/\*\*

\* This Data Value represents the current magnitude of 8 commonly-recognized types of EEG (brainwaves).

\* It is the ASIC equivalent of EEG\_POWER, with the main difference being that this Data Value is output as a series of

\* eight 3-byte unsigned integers instead of 4-byte floating point numbers.

\* These 3-byte unsigned integers are in big-endian format.

\*\*/

eegItem.delta = (value[0] << 16) | (value[1] << 8) | value[2];

eegItem.theta = (value[3] << 16) | (value[4] << 8) | value[5];

eegItem.lAlpha = (value[6] << 16) | (value[7] << 8) | value[8];

eegItem.hAlpha = (value[9] << 16) | (value[10] << 8) | value[11];

eegItem.lBeta = (value[12] << 16) | (value[13] << 8) | value[14];

eegItem.hBeta = (value[15] << 16) | (value[16] << 8) | value[17];

eegItem.lGamma = (value[18] << 16) | (value[19] << 8) | value[20];

eegItem.mGamma = (value[21] << 16) | (value[22] << 8) | value[23];

eegDataQueue.push(&eegDataQueue, eegItem);

Funcția pe care o apelam dintr-o aplicație pentru a citi datele, va citi aceste date din queue si le va trimite in format JSON:

char\* getThnkrDataJSON() {

if(eegDataQueue.size == 0) return "";

EegData eegItem = eegDataQueue.pop(&eegDataQueue);

/\* {"attention":"","meditation":"","delta":"","theta":"","low\_alpha":"","high\_alpha":"","low\_beta":"","high\_beta":"","low\_gamma":"","mid\_gamma":""} \*/

char num[25];

char\* buf = (char\*)malloc(255 \* sizeof(char));

strncpy(buf, "{\"attention\":\"", 14);

snprintf(num, 25, "%d", eegItem.attention);

strncat(buf, num, strlen(num));

strncat(buf, "\",\"meditation\":\"", 16);

snprintf(num, 25, "%d", eegItem.attention);

strncat(buf, num, strlen(num));

strncat(buf, "\",\"delta\":\"", 11);

snprintf(num, 25, "%f", eegItem.delta);

strncat(buf, num, strlen(num));

strncat(buf, "\",\"theta\":\"", 11);

snprintf(num, 25, "%f", eegItem.theta);

strncat(buf, num, strlen(num));

strncat(buf, "\",\"low\_alpha\":\"", 15);

snprintf(num, 25, "%f", eegItem.lAlpha);

strncat(buf, num, strlen(num));

strncat(buf, "\",\"high\_alpha\":\"", 16);

snprintf(num, 25, "%f", eegItem.hAlpha);

strncat(buf, num, strlen(num));

strncat(buf, "\",\"low\_beta\":\"", 14);

snprintf(num, 25, "%f", eegItem.lBeta);

strncat(buf, num, strlen(num));

strncat(buf, "\",\"high\_beta\":\"", 15);

snprintf(num, 25, "%f", eegItem.hBeta);

strncat(buf, num, strlen(num));

strncat(buf, "\",\"low\_gamma\":\"", 15);

snprintf(num, 25, "%f", eegItem.lGamma);

strncat(buf, num, strlen(num));

strncat(buf, "\",\"mid\_gamma\":\"", 15);

snprintf(num, 25, "%f", eegItem.mGamma);

strncat(buf, num, strlen(num));

strncat(buf, "\"}\0", 3);

return buf;

}

Rezultatul reieșit din teste unde am afișat acest JSON ca output într-o aplicație de test in terminal, este ca funcționează si reda datele corecte înapoi. De aici am trecut mai departe la aplicația vizuala (web) pentru a afișa datele.

## Afișarea undelor cerebrale

Aplicația server bazata pe NodeJS care citește datele, va avea doua procese separate, cu ajutorul JXCore. Unul dintre procese va încarcă conectorul si va începe sa citească date, iar al doilea proces va asculta pe un port request-uri de la clienți care vor sa citească aceste date.

Procesul de citire folosește Node-FFI pentru a încarcă librăria si pentru a apela cod C:

function getData() {

var ffi = require("node-ffi");

var lib = ffi.Library("libThnkrEegDecoder", { "getThnkrDataJSON": [ 'string', [ ] ] });

var json = "";

while(1) {

setTimeout(function(){}, 700);

json = lib.getThnkrDataJSON();

if(json != "") {

jxcore.store.shared.set("jsonData", json);

json = "";

}

}

}

La HTTP Request, procesul asculta si trimite datele citite in format JSON:

function runServer() {

var http = require('http');

jxcore.store.shared.set("jsonData", "{\"attention\":\"0\",\"meditation\":\"0\",\"delta\":\"0\",\"theta\":\"0\",\"low\_alpha\":\"0\",\"high\_alpha\":\"0\",\"low\_beta\":\"0\",\"high\_beta\":\"0\",\"low\_gamma\":\"0\",\"mid\_gamma\":\"0\"}");

http.createServer(function(req, res) {

res.writeHead(200, {

'Content-Type': 'text/plain',

'Access-Control-Allow-Origin': '\*'

});

res.end(jxcore.store.shared.get("jsonData"));

}).listen(8080, '192.168.0.18');

console.log('NodeJS server is running @ http://192.168.0.18:8080/');

}

Testele aici au fost făcute prin a scoate ca output in terminal direct din NodeJS aceste date folosind **console.log(…)**. Informațiile sunt corecte așa ca am putut trece la pasul următor: front-endul care cere aceste date si le afișează într-o diagrama dinamica.

Pentru diagrama, folosind libraria D3.js, am creat o clasa care poate fi folosita pentru a crea diagrama in diferite feluri:

var ThnkrChart = function (gWith, gHight) {

var \_self = this;

function s4() {

return Math.floor((1 + Math.random()) \* 0x10000).toString(16).substring(1);

}

function guid() {

return s4() + s4() + '-' + s4() + '-' + s4() + '-' + s4() + '-' + s4() + s4() + s4();

}

\_self.guid = guid();

\_self.DataSeries = [];

\_self.Ticks = 20;

\_self.TickDuration = 1000;

\_self.MaxValue = 100;

\_self.w = gWith;

\_self.h = gHight;

\_self.margin = { top: 50, right: 120, bottom: 60, left: 300 };

\_self.width = \_self.w - \_self.margin.left - \_self.margin.right;

\_self.height = \_self.h - \_self.margin.top - \_self.margin.bottom;

\_self.xText = '';

\_self.yText = '';

\_self.titleText = '';

\_self.chartSeries = {};

\_self.Init = function() {

d3.select('#chart-' + \_self.guid).remove();

\_self.svg = d3.select("body").append("svg")

.attr("id", 'chart-' + \_self.guid)

.attr("width", \_self.w)

.attr("height", \_self.h)

.append("g")

.attr("transform", "translate(" + \_self.margin.left + "," + \_self.margin.top + ")");

// Use Clipping to hide chart mechanics

\_self.svg.append("defs").append("clipPath")

.attr("id", "clip-" + \_self.guid)

.append("rect")

.attr("width", \_self.width)

.attr("height", \_self.height);

// Generate colors from DataSeries Names

\_self.color = d3.scale.category10();

\_self.color.domain(\_self.DataSeries.map(function (d) { return d.Name; }));

// X,Y Scale

\_self.xscale = d3.scale.linear().domain([0, \_self.Ticks]).range([0, \_self.width]);

\_self.yscale = d3.scale.linear().domain([0, \_self.MaxValue]).range([\_self.height, 0]);

// X,Y Axis

\_self.xAxis = d3.svg.axis()

.scale(d3.scale.linear()

.domain([0, \_self.Ticks])

.range([\_self.width, 0]))

.orient("bottom");

\_self.yAxis = d3.svg.axis()

.scale(\_self.yscale)

.orient("left");

// Line/Area Chart

\_self.line = d3.svg.line()

.interpolate("basis")

.x(function (d, i) { return \_self.xscale(i - 1); })

.y(function (d) { return \_self.yscale(d.Value); });

\_self.area = d3.svg.area()

.interpolate("basis")

.x(function (d, i) { return \_self.xscale(i - 1); })

.y0(\_self.height)

.y1(function (d) { return \_self.yscale(d.Value); });

// Title

\_self.Title = \_self.svg.append("text")

.attr("id", "title-" + \_self.guid)

.style("text-anchor", "middle")

.text(\_self.titleText)

.attr("transform", function (d, i) { return "translate(" + \_self.width / 2 + "," + -10 + ")"; });

// X axis text

\_self.svg.append("g")

.attr("class", "x axis")

.attr("transform", "translate(0," + \_self.yscale(0) + ")")

.call(\_self.xAxis)

.append("text")

.attr("id", "xName-" + \_self.guid)

.attr("x", \_self.width / 2)

.attr("dy", "3em")

.style("text-anchor", "middle")

.text(\_self.xText);

// Y axis text

\_self.svg.append("g")

.attr("class", "y axis")

.call(\_self.yAxis)

.append("text")

.attr("id", "yName-" + \_self.guid)

.attr("transform", "rotate(-90)")

.attr("y", 0)

.attr("x", -\_self.height / 2)

.attr("dy", "-3em")

.style("text-anchor", "middle")

.text(\_self.yText);

// Vertical grid lines

\_self.svg.selectAll(".vline").data(d3.range(\_self.Ticks)).enter()

.append("line")

.attr("x1", function (d) { return d \* (\_self.width / \_self.Ticks); })

.attr("x2", function (d) { return d \* (\_self.width / \_self.Ticks); })

.attr("y1", function (d) { return 0; })

.attr("y2", function (d) { return \_self.height; })

.style("stroke", "#eee")

.style("opacity", .5)

.attr("clip-path", "url(#clip-" + \_self.guid + ")")

.attr("transform", "translate(" + (\_self.width / \_self.Ticks) + "," + 0 + ")");

// Horizontal grid lines

\_self.svg.selectAll(".hline").data(d3.range(\_self.Ticks)).enter()

.append("line")

.attr("x1", function (d) { return 0; })

.attr("x2", function (d) { return \_self.width; })

.attr("y1", function (d) { return d \* (\_self.height / (\_self.MaxValue / 10)); })

.attr("y2", function (d) { return d \* (\_self.height / (\_self.MaxValue / 10)); })

.style("stroke", "#eee")

.style("opacity", .5)

.attr("clip-path", "url(#clip-" + \_self.guid + ")")

.attr("transform", "translate(" + 0 + "," + 0 + ")");

// Bind DataSeries to chart

\_self.Series = \_self.svg.selectAll(".Series")

.data(\_self.DataSeries)

.enter().append("g")

.attr("clip-path", "url(#clip-" + \_self.guid + ")")

.attr("class", "Series");

// Draw path from Series Data Points

\_self.path = \_self.Series.append("path")

.attr("class", "area")

.attr("d", function (d) { return \_self.area(d.Data); })

.style("fill", function (d) { return \_self.color(d.Name); })

.style("fill-opacity", .25)

.style("stroke", function (d) { return \_self.color(d.Name); });

// Legend

\_self.Legend = \_self.svg.selectAll(".Legend")

.data(\_self.DataSeries)

.enter().append("g")

.attr("class", "Legend");

\_self.Legend.append("circle")

.attr("r", 4)

.style("fill", function (d) { return \_self.color(d.Name); })

.style("fill-opacity", .5)

.style("stroke", function (d) { return \_self.color(d.Name); })

.attr("transform", function (d, i) { return "translate(" + (\_self.width + 6) + "," + (10 + (i \* 20)) + ")"; });

\_self.Legend.append("text")

.text(function (d) { return d.Name; })

.attr("dx", "0.5em")

.attr("dy", "0.25em")

.style("text-anchor", "start")

.attr("transform", function (d, i) { return "translate(" + (\_self.width + 6) + "," + (10 + (i \* 20)) + ")"; });

\_self.tick = function (id) {

\_self.thisTick = new Date();

var elapsed = parseInt(\_self.thisTick - \_self.lastTick);

var elapsedTotal = parseInt(\_self.lastTick - \_self.firstTick);

if (elapsed < 900 && elapsedTotal > 0) {

\_self.lastTick = \_self.thisTick;

return;

}

if (id < \_self.DataSeries.length - 1 && elapsedTotal > 0) {

return;

}

\_self.lastTick = \_self.thisTick;

// Add new values

for (i in \_self.DataSeries) {

\_self.DataSeries[i].Data.push({ Value: \_self.chartSeries[\_self.DataSeries[i].Name] });

// Backfill missing values

while (\_self.DataSeries[i].Data.length - 1 < \_self.Ticks + 3) {

\_self.DataSeries[i].Data.unshift({ Value: 0 })

}

}

d3.select("#yName-" + \_self.guid).text(\_self.yText);

d3.select("#xName-" + \_self.guid).text(\_self.xText);

d3.select("#title-" + \_self.guid).text(\_self.titleText);

\_self.path

.attr("d", function (d) { return \_self.area(d.Data); })

.attr("transform", null)

.transition()

.duration(\_self.TickDuration)

.ease("linear")

.attr("transform", "translate(" + \_self.xscale(-1) + ",0)")

.each("end", function (d, i) { \_self.tick(i); });

// Remove oldest values

for (i in \_self.DataSeries) {

\_self.DataSeries[i].Data.shift();

}

}

\_self.firstTick = new Date();

\_self.lastTick = new Date();

\_self.start = function () {

\_self.firstTick = new Date();

\_self.lastTick = new Date();

\_self.tick(0);

}

\_self.start();

}

\_self.addSeries = function (SeriesName) {

\_self.chartSeries[SeriesName] = 0;

\_self.DataSeries.push({ Name: SeriesName, Data: [{ Value: 0 }] });

\_self.Init();

}

}

Am creat după aceea, pagina **index.html** care folosind aceasta clasa, creează diagrama si începe sa citească date de la server cu ajutorul AJAX:

<!DOCTYPE html>

<html>

<meta charset="utf-8">

<head>

<title>This is the NEDU live EEG display POC</title>

<link rel="stylesheet" type="text/css" href="style.css">

<script src="jquery.min.js" charset="utf-8"></script>

<script src="d3.min.js" charset="utf-8"></script>

<script src="Thnkrchart.js" charset="utf-8"></script>

<script type="text/javascript">

function init() {

var server = "http://192.168.0.18:8080/";

var chart = new ThnkrChart(1800, 900);

chart.xText = "Seconds";

chart.yText = "Frequency (Hz)";

chart.titleText = "NEDU Project EEG Series TEST";

chart.Ticks = 5;

chart.TickDuration = 1000;

chart.MaxValue = 120;

function update() {

$.getJSON(server, function(data) {

var items = [];

$.each(data, function(key, val) {

chart.chartSeries[key] = val;

});

}).done(function() {

$('#status').html("SUCCESS: connected to server ...");

}).fail(function() {

$('#status').html("FAILED: to connect to server ...");

});

setTimeout(update, 500);

}

$.getJSON(server, function(data) {

var items = [];

$.each(data, function(key, val) {

chart.addSeries(key);

});

}).done(function() {

$('#status').html("SUCCESS: connected to server ...");

}).fail(function() {

$('#status').html("FAILED: to connect to server ...");

});

setTimeout(update, 500);

}

</script>

</head>

<body onLoad="init()">

<footer id="status">LOADING ...</footer>

</body>

</html>

## Clasificarea undelor cerebrale

Implementarea SOM pentru acest scop este făcută dintr-o matrice de 50x50. Implementarea este din doua clase, cea a unui neuron si cea a SOM-ului in sine.

class CNeuron

{

private:

vector<double> m\_dWeights;

double m\_dPosX;

double m\_dPosY;

public:

CNeuron(double posX, double posY, int numWeights):

m\_dPosX(posX),

m\_dPosY(posY)

{

/\*

\* the weights are initialized to small random values

\*/

for(int w = 0; w < numWeights; w++)

{

m\_dWeights.push\_back(RandFloat());

}

}

/\*

\* returns the euclidean distance

\* between the neuron's weights and the input vector

\*/

inline double GetEucDistance(

const vector<double> &vecInput

);

/\*

\* having a learning rate and target vector,

\* the function adjusts the neuron's weights

\*/

inline void AdjustWeights(

const vector<double> &vecTarget,

const double learningRate,

const double influence

);

double getPosX() const { return m\_dPosX; }

double getPosY() const { return m\_dPosY; }

};

Clasa SOM este definita in următorul fel:

class CSom

{

private:

vector<CNeuron> m\_SOM;

CNeuron\* m\_pWinningNeuron;

double m\_dMapRadius;

double m\_dTimeConstant;

int m\_iNumIterations;

int m\_iIterationCount;

double m\_dNeighbourhoodRadius;

double m\_dInfluence;

double m\_dLearningRate;

bool m\_bDone;

/\*

\* represents an input vector to each neuron in the network

\* and calculates the Euclidian distance between the vectors

\* of each neuron. it returns a pointer to the BMU

\*/

CNeuron\* FindBestMatchingUnit(const vector<double> &vecInput);

/\*

\* gets the gaussian distance between the BMU and

\* another neuron from the neighborhood

\*/

inline double GetGaussianDistance(const double dist, const double sigma);

public:

CSom():

m\_pWinningNeuron(NULL),

m\_iIterationCount(1),

m\_iNumIterations(0),

m\_dTimeConstant(0),

m\_dMapRadius(0),

m\_dNeighbourhoodRadius(0),

m\_dInfluence(0),

m\_dLearningRate(constStartLearningRate),

m\_bDone(false)

{}

/\*

\* creates and initializes the network

\*/

void Create(

int cxClient,

int cyClient,

int CellsUp,

int CellsAcross,

int NumIterations

);

/\*

\* taking a vector of input vectors, choses one at random

\* and runs the network through one training set

\*/

bool Learn(const vector<vector<double>> &data);

};

In timpul testelor, am descoperit că durează foarte mult să antrenăm SOM pentru „gânduri” sau stări mintale specifice și acestea depind, bine înțeles, de la persoană la persoană, așa încât ar fi nevoie de seturi de antrenare pentru fiecare persoană care ar dori să folosească acest sistem.

Fiecare ar putea decide câte clasificări specifice ar vrea sa aibă. După acest episod trebuie pornit sistemul iar acesta va înregistra și va salva setul de date într-un fișier. În timpul înregistrării ar trebui menținut același gând / aceeași stare timp de 20-30 de minute, după care trebuie trecut la starea următoare și așa mai departe, până reușim să definim toate clasificările dorite.

Acest set de antrenare este dat aplicației care va antrena SOM. După ce se parcurge setul de antrenare de câteva ori putem considera SOM ca fiind antrenată „destul”, se salvează vectorii de greutate pentru fiecare neuron într-un fișier specific pentru acea persoană, așa încât dacă pornim sistemul în modul de operare (nu antrenare) acesta va reacționa în funcție de clasificări (după ce definim mapările între clasificări și acțiuni dorite).

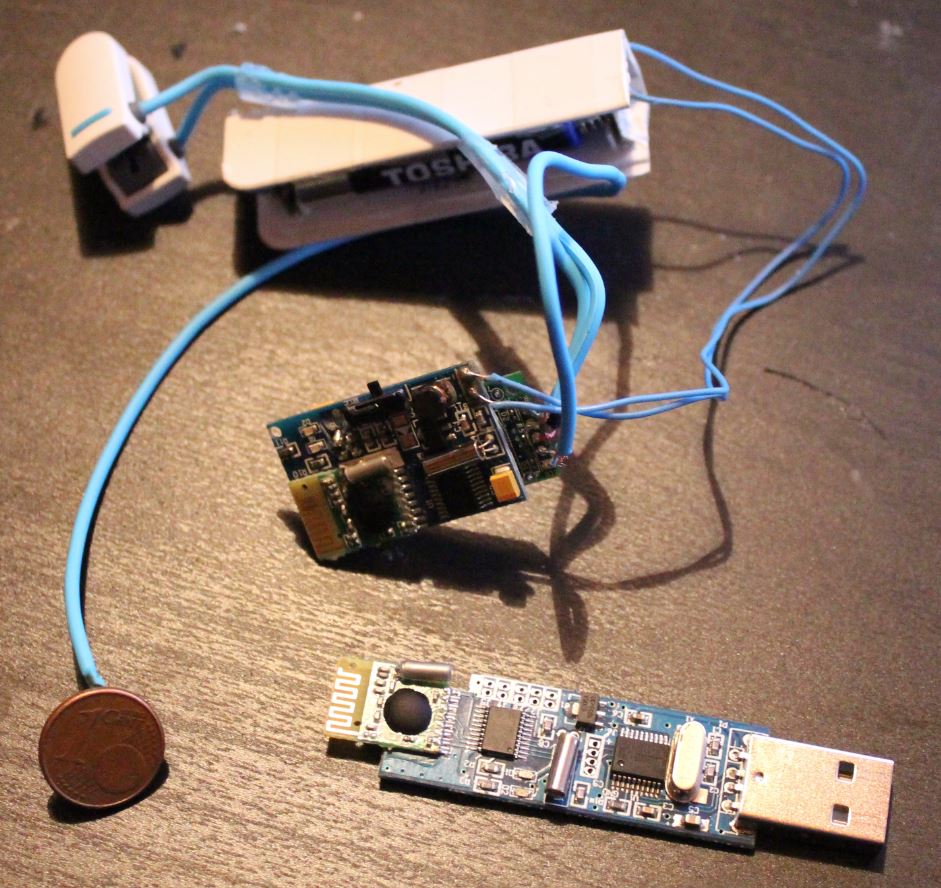
Pentru simplitatea acestui PoC (Proof of Concept) am ales doua clasificări după interpretarea undelor cerebrale prin SOM, pentru că cele mai simple și rapid de recunoscut și reprodus sunt starea de calm / meditație și starea de concentrare.

Din cauză că modulul EEG TGAM nu este un cititor complex, este dificil (deși nu imposibil) și durează mult timp să creăm seturi de antrenare specifice pentru a calibra SOM în așa fel încât să putem avea diferite alte acțiuni.

Pentru simplitate și pentru că aceste două stări se pot detecta în general pe (aproape) orice persoană, vom mișca cursorul mouse-ului la stânga atunci când ne calmăm, iar atunci când ne concentram, vom muta mouse-ul la dreapta.

## Rezultate obținute

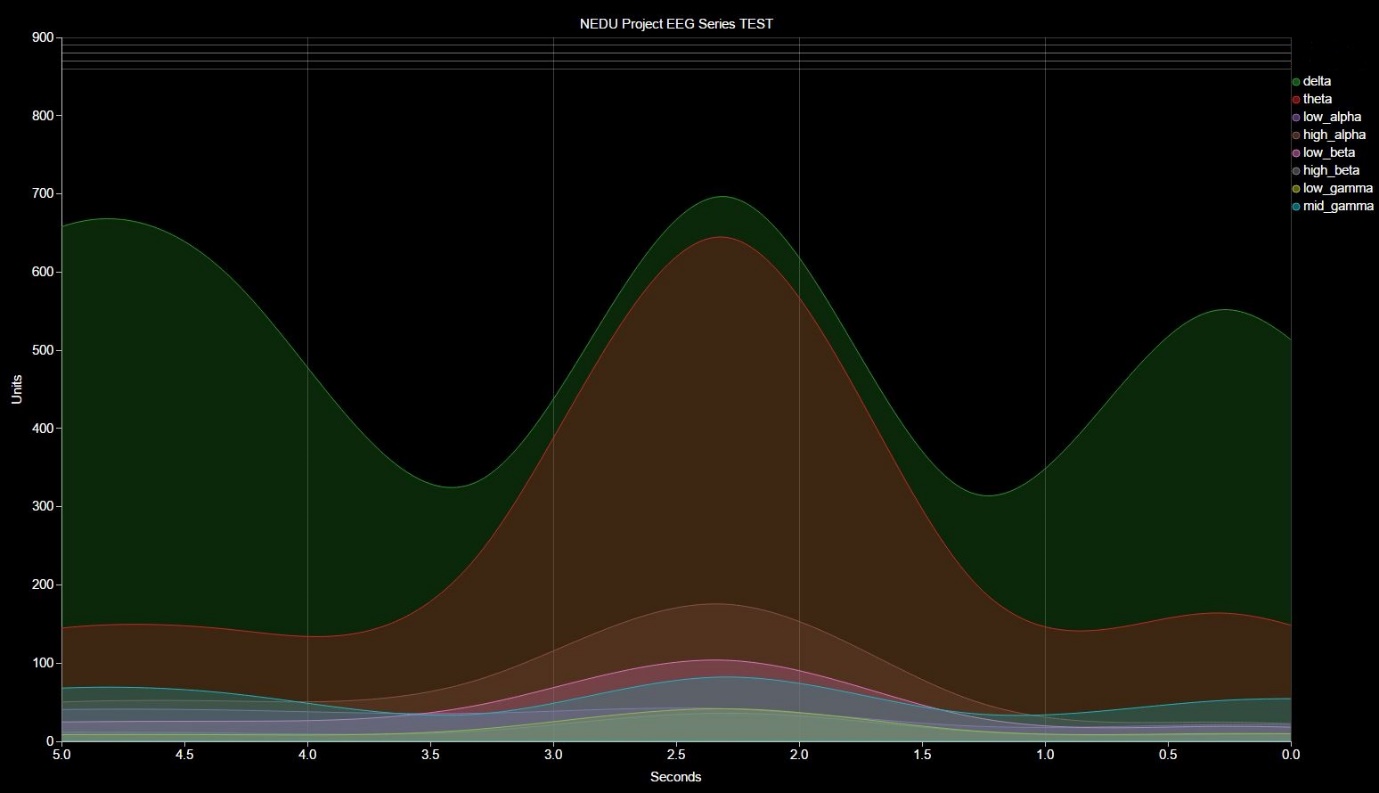
Partea hardware, respectiv USB RT Dongle-ul si TGAM EEG Chip-ul pe board-ul cu radio si senzorii atașați (+alimentare) arata așa:



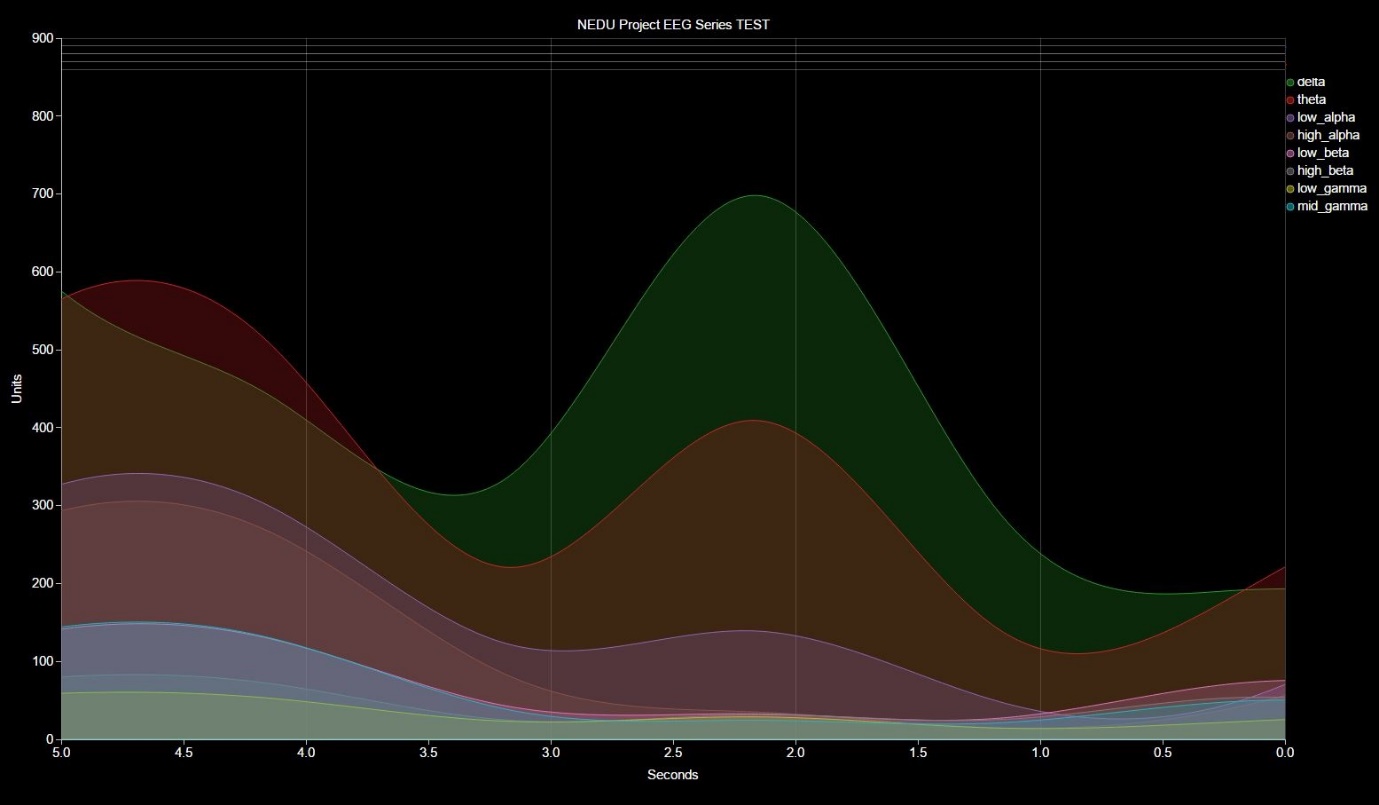
Un senzor se atașează de urechea stânga si celălalt se pune pe frunte in centru. Este amuzant de menționat ca senzorul care se pune pe frunte este o moneda de un Euro-Cent. Fiind din cupru si având o suprafața destul de mica, își servește scopul foarte bine.

SOM clasifica la citire cu acuratețe mare stările si rezultatul produs dintre cele doua stări propuse – meditație si concentrare – este consistent, așa ca cele doua acțiuni mapate funcționează ca bine. In acest exemplu, acțiunile fiind mișcarea mouse-ului in stânga (meditație) sau dreapta (concentrare).

Citirea si afișarea merg bine si din diferite teste, apar modele destul de clare. De exemplu starea mea de relaxare arata așa:



Mai jos este un exemplu foarte bun de trecere de la o stare de concentrare, la o stare de relaxare (meditație). Putem vedea o activitate crescuta in unda Theta, dar scăzută in Delta, după care acestea se inversează:



# Concluzii

Antrenând destul de mult rețeaua SOM si salvând greutățile rezultate, putem avea multe clasificări, care la rândul lor pot fi mapate pe diferite coduri. Fiecare cod poate fi folosit într-o aplicație care le primește si bazat pe fiecare cod, se pot face diferite acțiuni oricât de complexe.

Aceste coduri pot fi trimise la un micro-controller sau la mai multe in funcție de rezultat, care ar putea alcătui un sistem hardware pentru proteze, sau aparate de orice fel care sa ajute oameni cu un anumit handicap. Ar putea la fel de bine sa fie folosite pentru a controla o casa pentru a aprinde si stinge lumini, porni diferite aparate si așa mai departe.

Ceva foarte interesant observat in timpul diferitelor teste, este faptul ca am putea sa ne folosim de aceste înregistrări a activității undelor cerebrale pentru a compara diferite stări intre persoane diferite. Este interesant de comparat pe aceeași persoană activitatea undelor cerebrale in timpul viselor – de exemplu diferența activității dintre un vis plăcut si un coșmar de exemplu.

Gândindu-mă la transformările Fourier, am putea chiar sa transformam frecventele înregistrate in sunete, si acestea pot fi ascultate direct. Poate ca aceste sunete pot fi clasificate cumva si de propriul nostru creier auzind sunetele produse. Ar putea fi trimise efectiv si prin telefon. Reinterpretate si procesate la ieșire.

Deși acest sistem este momentan un simplu PoC (*Proof of Concept*) putem vedea cât de ușor este de fapt să începem să experimentăm și să dezvoltăm aplicații bazate pe citiri EEG. Având ca cititor EEG hardware un chip care nu are decât doi senzori, acțiunile pe care le putem defini pe baza numărului de clasificări este foarte mic.

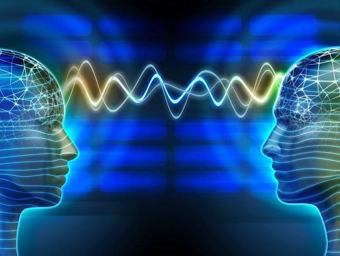
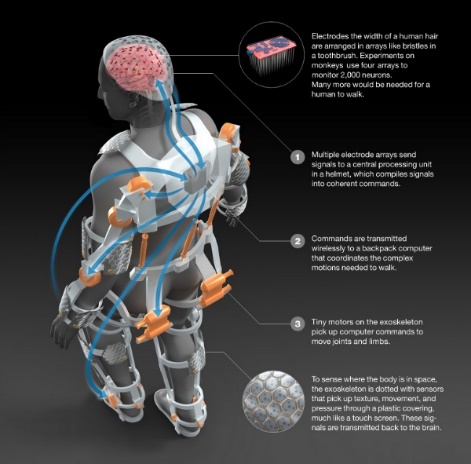
Deși chip-ul TGAM reușește foarte bine să citească undele cerebral doar cu acești doi senzori, folosindu-și algoritmii, pentru a dezvolta un software cu adevărat complex și util, ne-ar trebui un cititor EEG profesional, respectiv unul care să aibă cel puțin ~28 de senzori.

Mai mult decât atât, este de preferat ca senzorii să fie electrozi uzi pentru a face contact cât mai bun cu pielea și pentru a rămâne stabili. Ar fi ideal ca acești electrozi uzi să fie atașați de interiorul unei caști de cauciuc care se poate mula perfect pe forma craniului. Ca soluție de clasificare, am putea încerca diferite alte forme de rețele neurale și cel mai important, cu diferite variabile de început și numere diferite de neuroni.

Intenția mea este ca acest proiect sa fie doar începutul urmând să dezvolt în viitor sisteme hardware cât și software, mult mai complexe. Voi continua studiile în domeniul neuro-științei și inteligenței artificiale. Îmi doresc să lucrez într-un centru de cercetare în acest domeniu.

Chiar dacă pentru moment pare imposibil, prea ambițios sau prea departe în viitor, scopul final ar fi înțelegerea cât mai complexă a funcționării creierului uman și dezvoltarea unei adevărate/reale inteligențe artificiale.

Progresul hardware este constant și ultimele descoperiri în domeniul neuro-științei, al inteligenței artificiale și fizicii cuantice (aici – CPU cuantic de ex.) sunt foarte promițătoare. Sper că aceste ținte, aparent greu de atins, vor deveni realitate și generația noastră coautoare sau cel puțin contemporană, cu propriul vis împlinit.



**[14]**

Dacă în urmă cu 200-300 de ani am fi afirmat că omul va citi impulsurile electrice ale creierului (despre care atunci nu se știa că există) apoi le vom prelua și procesa pentru a interacționa putea interacționa cu un instrument (calculatorul) în felul în care o facem astăzi, reacția ar fi fost asemănătoare cu cea pe care o au contemporanii noștri atunci când vorbim despre inteligența artificială reală și autonomă.

Progresul uriaș, evident în toate domeniile științei, apariția unor noi discipline și cercetarea în domenii necunoscute până de curând sau în domenii de graniță ale științei, explorarea universului macro și micro, crearea de instrumente din ce în ce mai performante și de aplicații din ce în ce mai numeroase și diverse, pregătesc marele salt către reproducerea inteligenței vii care să servească unui om moral și responsabil.

# Bibliografie

1. **Erik Andreas Larsen, *Classification of EEG Signals in a Brain-Computer Interface System***,Norwegian University of Science and Technology, 2011
2. **NeuroSky, *TGAM1 Spec Sheet***,NeuroSky, 2010
3. **NeuroSky, MindWave *Dongle Communication Protocol***,NeuroSky, 2010
4. **NeuroSky, *ThinkGear Serial Stream Guide***,NeuroSky, 2010
5. **Chee-Keong Alfred Lim, Wai Chong Chia, *Analysis of Single-Electrode EEG Rhythms Using MATLAB to Elicit Correlation with Cognitive Stress***, International Journal of Computer Theory and Engineering, 2015
6. **Haiyang Liu, Abhishek Chandra and Jaideep Srivastava, *eSENSE: Energy Efficient Stochastic Sensing Framework for Wireless Sensor Platforms****,* University of Minnesota, 2006
7. **K. Amarasinghe, D.Wijayasekara, M.Manic, *EEG Based Brain Activity Monitoring using Artificial Neural Networks***, University of Idaho, 2014
8. **Peter Dayan, L. F. Abbott, *An Exploration of the Utilization of Electroencephalography to Create Thought-Based Brain-Computer Interfaces***, Theoretical Neuroscience, 2000
9. **Jzau-Sheng Lin, Cheng-Hung Shieh, *An Ssvep-Based Bci System and its Applications***, International Journal of Advanced Computer Science and Applications, 2014
10. **Scott Meyers, *Effective C++: 55 Specific Ways to Improve Your Programs and Designs (3rd Edition)***, Addison-Wesley Professional, 2005
11. **Scott Meyers, *Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library***, Addison-Wesley Professional, 2001
12. **Wikipedia.org**
13. **Linux.org**
14. **Google.com**