**CUPRINS**

|  |  |
| --- | --- |
| **INTRODUCERE**........................................................................................................ | 9 |
| **1. METODE DE PRELUCRARE NUMERICĂ A SEMNALELOR BIOMEDICALE ÎN DOMENIUL DE FRECVENŢE**......................................... | 11 |
| **1.1. Transformata Fourier Rapidă**.................................................................... | 11 |
| **1.2. Procesarea electrocardiogramei – detectarea vârfurilor R**..................... | 19 |
| **1.3. Prelucrarea numerică a semnalelor**........................................................... | 22 |
| **1.4. Medii de dezvoltare software**...................................................................... | 28 |
| **2. PROIECTAREA, IMPLEMENTAREA ŞI DESCRIEREA SISTEMULUI**......................................................................................................... | 31 |
| **2.1. Arhitectura generală a softului**................................................................... | 31 |
| **2.2. Realizarea algoritmului transformatei Fourier rapidă**............................ | 34 |
| **3. TESTAREA FUNCŢIONALITĂŢII SISTEMULUI PROIECTAT**................. | 39 |
| **4. ARGUMENTAREA ECONOMICĂ A SISTEMULUI**....................................... | 46 |
| **4.1. Descrierea proiectului**................................................................................. | 46 |
| **4.2. Analiza SWOT**............................................................................................. | 47 |
| **4.3. Planul calendaristic**..................................................................................... | 48 |
| **4.4. Argumentarea economică**........................................................................... | 50 |
| **4.5. Partea organizatorică: Selecţia personalului: noţiune şi etapele**  **de bază**......................................................................................................... | 55 |
| **CONCLUZII**............................................................................................................... | 57 |
| **BIBLIOGRAFIE**......................................................................................................... | 59 |
| **ANEXĂ**........................................................................................................................ | 60 |

**INTRODUCERE**

Putem defini un semnal ca fiind suportul fizic al informaţiei transferate între sisteme. Orice semnal este din punct de vedere matematic o funcţie de timp, dar în definirea semnalului pot interveni şi alte variabile reprezentate de mărimi fizice precum spaţiul, temperatura, etc.

Un compartiment foarte important al medicinii în tratamentul bolilor este diagnosticarea cu succes a pacienţilor şi totodată şi monitorizarea postoperatorie. Datorită progresului tehnico-ştiinţific în ramura biomedicinii s-a ajuns la implementarea dispozitivelor de monitorizare a stării pacientului cu automatizarea, achiziţia şi asigurarea interpretării computerizata a datelor medicale şi a semnalelor fiziologice.

Prelucrarea numerică a semnalelor (PNS) este un domeniu al ştiinţei care s-a dezvoltat foarte rapid în ultimii 30 de ani, ca urmare a progresului înregistrat de tehnologia calculatoarelor şi fabricarea circuitelor integrate. Prelucrarea numerică a semnalelor are aplicaţii în orice domeniu în care informaţia poate fi prezentată sub formă numerică. Dintre acestea se amintesc:

a) Procesarea de imagini: facsimil, harta vremii prin satelit, animaţie etc.;

b) Instrumentaţie/control: analiză spectrală, controlul poziţiei şi al vitezei, compresie de date;

c) Vorbire/audio: recunoaşterea vocii, sinteza vorbirii, egalizare etc.;

d) Telecomunicaţii: anulare ecou, egalizare adaptivă, conferinţe video, comunicaţii de date etc.;

e) Biomedical: scanare computer-tomografie, electroencefalografie, electrocardiografie etc.;

Modelele matematice de descriere a metodelor de extragere a parametrilor cer realizarea de funcţii complexe, care implică un aparat hardware puternic şi modern. Cele mai noi tehnici de prelucrare a semnalelor cer procesoare specializate, cu frecvenţe mari de lucru, memorii voluminoase sau chiar şi sisteme de operare specifice. Cu cât este mai variat sistemul şi poate realiza mai multe funcţii cu atât creşte şi complexitatea parametrilor ce vor trebui prelucraţi şi monitorizaţi.

Scopul acestei lucrării constă în proiectarea unui sistem informaţional pentru analiza de frecvenţă a semnalelor medicale, determinarea spectrului de frecvenţă a acestora, analizarea principalelor metode de procesare numerică a semnalelor biomedicale. Scopul a fost de a obţine un software ce poate fi instalat pe un calculator cu sistem de operare şi să poată reda la ecranul monitorului evoluţia în timp real a semnalelor biomedicale şi determinarea unor parametri a acestora.

Analiza în frecvenţă şi prelucrarea numerică a semnalelor este un domeniu cu perspective pe viitor. Dezvoltarea acestuia impune şi formarea noilor specialişti. Această ramură a digitalizării deschide noi drumuri, apar noi cerinţe. Deja nu este îndeajuns să se monitorizeze o activitate sau să se înregistreze nişte semnale achiziţionate cu ajutorul electrozilor. Se dezvoltă rapid noi algoritmi de procesare digitală a datelor, apar noi modalităţi de a interpreta rezultatele obţinute, se modelează fără mari dificultăţi suprafeţe 3D, se simulează interacţiunea diferitor medii, se construiesc grafice şi histograme în timp real. O înregistrare de semnal deja nu este aşa importantă fără posibilitatea prelucrării digitale a acestuia şi respectiv afişarea sa integrală sau pe porţiuni.

Transformarea Fourier discretă (Discrete Fourier Transform – DFT) este una dintre cele mai utilizate şi puternice proceduri din domeniul prelucrării numerice a semnalelor. DFT este o procedură matematică folosită pentru determinarea conţinutului în frecvenţă (armonic) al semnalelor. Ea provine din transformarea Fourier continuă.

Aceste reprezentări în esenţă implică descompunerea semnalelor în termene ca componente sinusoidale (sau exponenţiale complexe). Cu aşa o descompunere, semnalul se spune că este reprezentat în domeniul de frecvenţă. Analiza de frecvenţă a unui semnal implică descompunerea semnalului în componentele lui de frecvenţă (sinusoidale). Ca şi în fizică, termenul de spectru se întrebuinţează când se referă la conţinutul de frecvenţă a unui semnal. Procesul de obţinere a spectrului dintr-un semnal dat folosind uneltele de bază matematice este numit analiza de frecvenţă sau spectrală.

Aceste noi posibilităţi cu o accentuare mai largă pe viitor m-au motivat să aleg această temă şi să încerc să dezvolt un proiect care ar cuprinde posibilităţi de analiză în frecvenţă. Este o temă destul de interesantă, dar care necesită o pregătire matematică destul de bună. Cunoaşterea celor mai des întâlniţi algoritmi şi a tehnicilor de programare este o altă cerinţă de bază pentru a putea studia eficient prelucrarea numerică a semnalelor.

Pentru realizarea obiectivelor proiectului s-a ales algoritmul Cooley-Tukey pentru descrierea transformatei Fourier rapidă. Acesta este un algoritm optimizat ce poate rula cu uşurinţă pe un circuit DSP. Rezultatul este un vector complex, din care se afişează magnitudinea spectrului obţinut. Spectrul este reprezentat în diferite moduri: complet, o semi perioadă (doar frecvenţele pozitive) sau oglindit faţă de frecevenţa zero (frecvenţele mici sunt centrate în spectru). Aceste idei sunt descrise în cele patru capitole ale lucrării.

Capitolul 1 descrie aspecte teoretice despre transformata Fourier discretă şi unele proprietăţi de bază ale acesteia, ce servesc ca suport şi pentru realizarea algoritmului accelerat al acesteia – transformata Fourier Rapidă. Este dată şi o metodă de determinare a vârfurilor R.

Al doilea capitol conţine modul de realizare practic a algoritmului FFT, optimizarea acestuia. Este descrisă şi structura generală a sistemului, posibilităţile (caracteristicile) sale funcţionale. Sunt explicaţi algoritmii de bază utilizaţi în proiect.

Capitolul 3 conţine exemple de testare a funcţionalităţii softului, rezultate obţinute, analiza eficienţei algoritmilor în funcţie de datele de intrare. În cadrul proiectului ne-am limitat la prelucrarea semnalelor înregistrate şi salvate, fără posibilitatea analizei în timp real.

Ultimul capitol, al patrulea, se referă la argumentarea economică a proiectului, eficienţa de comercializare a acestuia, cheltuieli de producere, puncte forte şi slabe.

**1. METODE DE PRELUCRARE NUMERICĂ A SEMNALELOR BIOMEDICALE ÎN DOMENIUL DE FRECVENŢE**

**1.1. Transformata Fourier Rapidă**

Transformata Fourier rapidă (Fast Fourier Transform – FFT) este o tehnică cu viteză înaltă pentru calcularea transformatei Fourier discrete, care la rândul ei este versiunea discretă (DFT) a transformatei Fourier continue – originea tuturor celorlalte derivaţii ale sale. Astfel, în pas istoric, mai întâi a fost descoperită forma continuă a transformatei, apoi a fost creată forma discretă pentru semnale eşantionate şi în final a fost inventat algoritmul pentru calcularea rapidă a versiunii discrete.

J. W. Cooley şi J. W. Tukey sunt consideraţi ca primii care au făcut ca FFT să fie cunoscută în lume, prin lucrarea lor „Un algoritm pentru maşina de calcul a seriei Fourier complexe”, în Mathematics Computation, vol. 19, 1965, p. 297-301. În retrospectivă, alţii au descoperit această tehnică cu mulţi ani înainte. De exemplu, marele matematician german Karl Friedrich Gauss (1777-1855) a utilizat această metodă mai mult de un secol în urmă. Această muncă timpurie a fost în mare parte uitată, deoarece îi lipsea obiectul pentru a o face practică – computerul digital. Cooley şi Tukey sunt onoraţi pentru că ei au descoperit FFT la timpul potrivit, la începutul revoluţiei calculatorului.

Pentru a înţelege transformata Fourier rapidă trebuie să studiem transformata Fourier, care este o integrală de forma următoare [3]:

Fourier transform (1.1)

Transformata operează în domeniul valorilor complexe. Reamintim că exponenta imaginară ar putea fi scrisă ca:

Imaginary exponent (1.2)

Pentru funcţii eşantionate transformata continuă (1.1) trece în una discretă:

Discrete Fourier transform (DFT) (1.3)

În relaţia (1.3) {f0, f1, ... , fN-1} este funcţia discretă de intrare şi {F0, F1, ... , FN-1} este rezultatul transformatei Fourier. Se poate uşor observa că pentru programarea DFT este de ajuns de scris o buclă dublă şi de calculat suma produselor eşantioanelor (probelor) de intrare şi exponentelor imaginare. Numărul de operaţii necesare este evident de ordinul O(N2). Dar datorită proprietăţilor de transformare este posibil de redus numărul operaţiilor la ordinul O(Nlog2N).

Pentru accelerarea calculelor se pot utiliza unele şiretlicuri. De exemplu, pentru N=8 se obţine:

Discrete Fourier transform for N=8 (1.4)

Este uşor observabil că această sumă se poate despărţi în alte două sume similare separând termenii pari şi impari şi extrăgând un termen comun din ultima sumă:

Transform separation (1.5)

Acum se pot separa sumele din paranteze din nou:

Transform separation (1.6)

Astfel se primesc trei nivele de sumare de ordinul log28: cel mai adânc nivel din parantezele rotunde, cel din mijloc din parantezele pătrate şi cel din exterior. Pentru fiecare nivel exponenta ca factor al multiplicării pentru al doilea termen este aceeaşi. Astfel relaţia (1.6) devine:

Transform separation (1.7)

Şi acum cea mai importantă observaţie care se poate face pentru accelerarea procesului de calcul este periodicitatea exponentei ca factor al multiplicării.

Imaginary exponent periodicity (1.8)

Pentru exponenta din paranteze perioada este n=2, ceea ce înseamnă că sumele din paranteze sunt aceleaşi pentru n=0, 2, 4, 6 şi pentru n=1, 3,5, 7. Asta înseamnă că la nivelul cel mai adânc din paranteze sunt necesare 4\*2=8 sume periodice (sume în total). Întrucât n=1, 3, 5, 7 corespunde cu jumătatea perioadei π, exponenta este aceeaşi ca şi pentru n=0, 2, 4, 6, dar de semn opus.

Imaginary exponent periodicity (1.9)

Pentru exponenta din parantezele pătrate perioada este n=4, ceea ce înseamnă că sunt aceleaşi sume pentru perechile n=0, 4; n=1, 5; n=2, 6 şi n=3, 7. Înseamnă că la nivelul de mijloc din parantezele pătrate sunt 2\*4=8 sume şi a doua jumătate din ele se pot obţine din nou prin schimbarea semnului din prima lor jumătate – datorită faptului dat distanţa dintre n şi n+2 este π. Astfel, pentru n=0, 4 factorul este 1 şi pentru n=2, 6 este -1; pentru n=1, 5 este egal cu –i şi pentru n=3, 7 este i.

La nivelul exterior este doar o sumă pentru fiecare componentă a transformării, şi perioada exponentei este egală cu 8. Asta oferă 1\*8=8 sume şi a doua jumătate din acestea se pot obţine prin schimbarea semnului din prima jumătate.

Aşadar, la fiecare nivel al calculării se obţin 8 sume. În termenii lui N se obţin log2N nivele şi N sume la fiecare nivel, ceea ce oferă un număr de operaţii de ordinul O(Nlog2N). Pe de altă parte, având un număr constant de sume la fiecare etapă înseamnă că datele se pot procesa pe loc.

Pentru a trece din domeniul de frecvenţă în domeniul de timp se utilizează transformata Fourier inversă, descrisă de următoarea relaţie:

Inverse Fourier transform (1.10)

Forma discretă a acesteia este:

Inverse discrete Fourier transform (1.11)

Astfel diferenţa dintre transformata directă (1.3) şi cea inversă (1.11) este doar în semn şi nu numaidecât necesar factorul de scalare – nu este nevoie de acesta dacă suntem interesaţi de rata dintre componente, dar nu de valorile absolute. Asta înseamnă că rutinele pentru calcularea transformatei directe pot fi aplicate şi pentru cea indirectă cu uşoare modificări de cod.

1.1.1. DFT reală utilizând DFT complexă. Transformata Fourier rapidă se bazează pe cea discretă complexă, o versiune mai sofisticată a DFT reale. Aceste transformări sunt numite în funcţie de datele reprezentate, adică utilizând numere complexe sau numere reale. Deoarece FFT este un algoritm pentru calcularea DFT complexă, este important de înţeles cum se transferă datele din DFT reală în forma DFT complexă şi din ea înapoi. În figura 1.1 este comparat cum DFT reală şi complexă salvează datele.

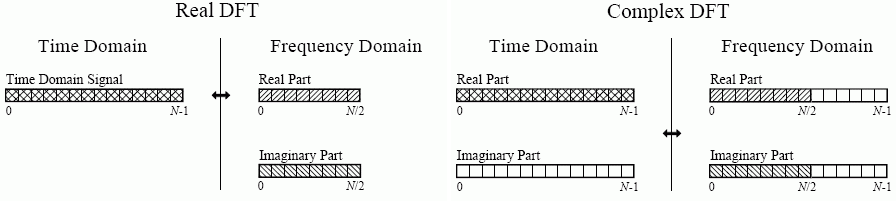


Fig. 1.1 Compararea DFT reală şi complexă [1]

Cea reală transformă un semnal de N puncte în domeniul timp în două semnale de N/2+1 puncte în domeniul de frecvenţă. Cele 2 semnale obţinute sunt numite partea reală şi partea imaginară, păstrând amplitudinile undelor sinus şi cosinus respectiv. În comparaţie, DFT complexă transformă două semnale de N puncte în domeniul timp în două semnale de N puncte în domeniul de frecvenţă. Toate valorile din aceste tablouri (şiruri) sunt numere obişnuite, nu complexe (lipseşte j).

De exemplu este dat un semnal de N puncte şi se cere calcularea DFT reală utilizând DFT complexă. Mai întâi se copie semnalul în partea reală a DFT complexă din domeniul de timp, şi apoi se setează cu zero toate probele din partea imaginară. Rezultatul calculării DFT complexă este într-un semnal real şi unul imaginar în domeniul de frecvenţă, fiecare compus din N puncte. Eşantioanele de la 0 la N/2 din aceste semnale corespund spectrului DFT reale.

Domeniul de frecvenţă al transformatei Fourier discrete este periodic atunci când frecvenţele negative sunt incluse (figura 1.2). Alegerea unei singure perioade este arbitrară, aşa că poate fi între -1 şi 0, -0.5 şi 0.5, 0 şi 1, sau orice alt interval unitate referit la rata de eşantionare. Spectrul de frecvenţe a DFT complexă include frecvenţele negative în aranjamentul dintre 0 şi 1. În alte cuvinte, o perioadă completă se întinde de la proba 0 până la proba N-1, corespunzând cu 0 până la 1 ori rata de eşantionare. Frecvenţele pozitive sunt situate între eşantionul 0 şi N/2, corespunzând cu 0 - 0.5. Celelalte probe, dintre N/2+1 şi N-1, conţin valorile frecvenţelor negative (care de obicei se ignoră).

Calcularea DFT reală inversă utilizând DFT complexă inversă este un pic mai greu. Aceasta este din cauza că trebuie să se asigure că frecvenţele negative sunt încărcate în forma corectă. Punctele de la 0 până la N/2 din DFT complexă sunt aceleaşi ca şi în DFT reală, pentru ambele părţi reală şi imaginară. Pentru partea reală, punctul N/2+1 este acelaşi ca şi N/2-1, punctul N/2+2 coincide cu N/2-2, etc. Aceasta continuă până la punctul N-1, care este acelaşi ca şi punctul 1. Acelaşi model se utilizează şi pentru partea imaginară, cu excepţia semnului, care este inversat. Astfel, punctul N/2+1 este negativul punctului N/2-1, punctul N/2+2 este negativul punctului N/2-2, etc. Se notează că eşantionul 0 şi N/2 nu au punct identic în această schemă de duplicare.

1.1.2. Proprietatea de periodicitate a transformatei Fourier discretă. Faţă de alte forme ale transformatei Fourier, DFT vede ambele domenii de timp şi de frecvenţă ca periodice. În figura 1.2 este dat un exemplu de periodicitate în domeniul de frecvenţe.

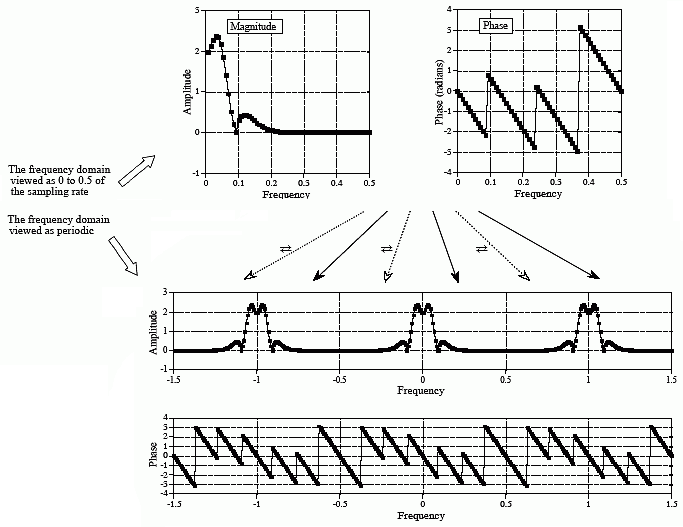


Fig. 1.2 Periodicitatea DFT în domeniul de frecvenţe [1]

În figurile din partea de sus este arătată magnitudinea şi faza spectrului, fiind vizualizate ca compuse din N/2+1 probe repartizate între 0 şi 0.5 din frecvenţa de eşantionare. Acesta este cel mai simplu mod de a vizualiza spectrul de frecvenţe, dar nu explică prea multe despre proprietăţile DFT.

Celelalte două figurii de jos arată cum DFT vede acest spectru ca fiind periodic. „Cheia” constă în aceea că spectrul dintre 0 şi 0.5 pare să aibă o oglindire a frecvenţelor dintre 0 şi -0.5. Această oglindire a frecvenţelor negative este puţin diferită pentru magnitudine şi pentru fază. În magnitudine, semnalul este oglindit de la stânga la dreapta şi schimbat în semn. Astfel semnalul magnitudinii este un semnal par, în timp ce faza este unul impar. Dacă spectrul de frecvenţă este convertit în părţile reală şi imaginară, cea reală va fi mereu pară, iar cea imaginară – impară.

Considerând aceste frecvenţe negative, DFT vede domeniul de frecvenţe ca periodic, cu o perioadă de 1 timpi din frecvenţa de eşantionare, aşa ca -0.5 până la 0.5, sau de la 0 la 1. Din punctul de vedere a numărului de eşantioane, acesta face ca lungimea perioadei în domeniul de frecvenţe să fie egală cu N, la fel ca şi în domeniul de timp.

Periodicitatea domeniului de frecvenţă devine susceptibilă către efectul de aliasing, care este complet analog cu cel din domeniul de timp. Dacă semnalul în domeniul de timp este modificat, este evident că şi spectrul de frecvenţe se va schimba. Dacă spectrul modificat nu poate încăpea în spaţiul prevăzut, se va împinge în perioadele adiacente. Acest efect de aliasing cauzează două probleme: frecvenţele nu sunt acolo unde ar trebui să fie şi frecvenţele suprapuse din diferite perioade se adună, distrugând informaţia.

1.1.3. Mecanismul de funcţionarea al FFT. FFT operează prin descompunerea unui semnal de N puncte din domeniul timpului în N semnale în acelaşi domeniu, fiecare fiind compus dintr-un singur punct. Deci pasul următor este calcularea a N spectre de frecvenţă ce corespund acestor N semnale din domeniul timpului. La final, aceste N spectre sunt sintetizate într-un singur spectru.

Schematic acest proces este reprezentat şi în figura următoare, în care un semnal de 16 puncte este descompus prin intermediul a 4 etape separate:

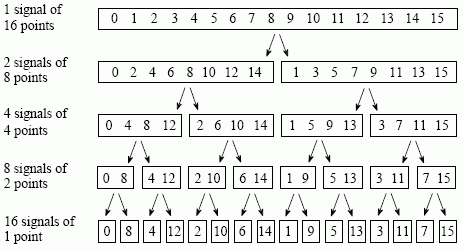


Fig. 1.3 Descompunerea FFT [1]

Prima etapă desparte semnalul în două semnale a câte 8 puncte. A doua – în patru a câte 4 puncte. Acest model continuă până când se obţin N semnale de câte un punct. De fiecare dată când semnalul este despărţit în două, o descompunere întreţesută este utilizată, astfel semnalul este separat în probele sale numerotate ca pare şi impare. În această descompunere sunt necesare log2N etape.

Decompoziţia nu este altceva decât reordonarea probelor în semnal. Figura 1.4 arată modelul de rearanjare necesar. În partea stângă, numerele probei semnalului original sunt enumerate împreună cu echivalenţii lor binari. În partea dreaptă, sunt deja numerele probei deja aranjate şi de asemenea cu echivalenţii lor binari. Ideea importantă este că numerele binare sunt inversul reciproc a lor. De exemplu, proba numărul 3 (0011) este schimbată cu proba 12 (1100). De asemenea, proba 14 (1110) este schimbată cu proba numărul 7 (0111), şi aşa în continuare. Decompoziţia FFT în domeniul de timp este de obicei efectuată de un algoritm de sortare inversă pe biţi. Acest lucru implică rearanjarea ordinului a N probe din domeniul de timp prin numărarea în binar cu biţii oglindiţi de la stânga la dreapta (ca în coloana din dreapta a figurii 1.4).

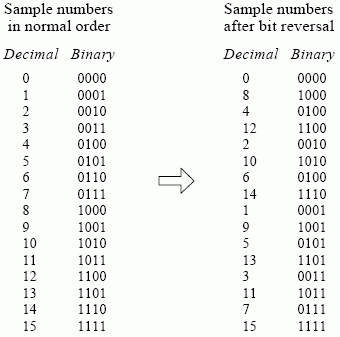


Fig. 1.4 Sortarea inversă pe biţi pentru FFT [1]

Următorul pas în algoritmul FFT este găsirea spectrului de frecvenţe a semnalelor de un punct din domeniul de timp. Acest lucru este simplu, deoarece spectrul unui punct este egal cu el însuşi. Asta înseamnă că nu se cere nimic altceva de făcut la această etapă. Deşi nu se cere nimic de făcut, nu trebuie să se uite că deja fiecare din aceste puncte este un spectru şi nu un semnal în timp.

Ultima etapă în FFT este să se combine cele N spectre de frecvenţă în ordine inversă exact cum a avut loc descompunerea din domeniul de timp. Deja algoritmul de inversare pe biţi nu poate fi aplicabil şi trebuie să se reîntoarcă câte un nivel de fiecare dată. În prima etapă spectrul din 16 frecvenţe este sintetizat în 8 spectre (câte 2 puncte fiecare). În stadia următoare, cele 8 spectre sunt sintetizate în 4 spectre (de câte 4 puncte fiecare) şi procesul continuă. Ultima etapă returnează rezultatul FFT, un spectru de frecvenţe de 16 puncte.

Figura 1.5 arată cum 2 spectre de frecvenţă, fiecare compus din 4 puncte, sunt combinate într-un singur spectru din 8 puncte. Această sinteză trebuie să anuleze decompoziţia întreţesută făcută în domeniul de timp. Prin alte cuvinte, operaţiunea din domeniul de frecvenţe trebuie să corespundă cu procedura în domeniul timpului de combinarea a două semnale de 4 puncte prin întreţesere. Diluarea domeniului de timp cu zerouri corespunde cu duplicarea spectrului. Prin urmare, spectrul de frecvenţe este combinat în FFT prin duplicarea lor şi apoi sumând spectrele dublate împreună.

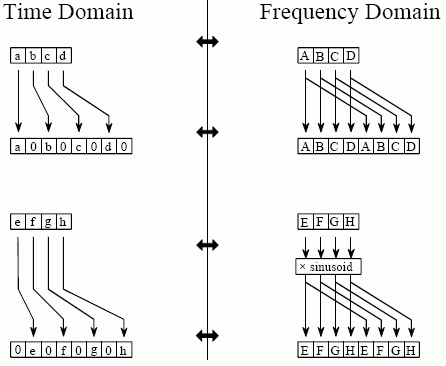


Fig. 1.5 Sinteza FFT [1]

Pentru a corespunde când sunt sumate, cele două semnale în domeniul de timp sunt amestecate cu zerouri într-un mod puţin mai diferit. Într-un semnal, punctele impare sunt zero, în timp ce în celălalt semnal, punctele pare sunt zero. Cu alte cuvinte, unul din semnalele în domeniul de timp este deplasat spre dreapta cu o probă. Această deplasare în domeniul de timp corespunde cu multiplicarea spectrului cu o sinusoidă

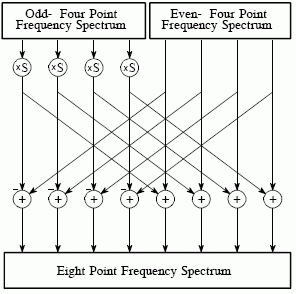


Fig. 1.6 Diagrama de stare a sintezei FFT [1]

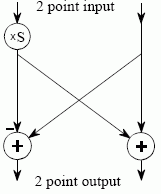


Fig. 1.7 Celula fluture FFT [1]

În figura 1.6 este arătată diagrama de decurgere a procesului pentru combinarea a două spectre de 4 puncte într-un spectru de 8 puncte. Pentru a reduce situaţia şi mai mult, se notează că figura 1.6 este formată dintr-un model de bază, iar în figura 1.7 se repetă mereu aceeaşi schemă.

Diagrama din figura 1.7 se numeşte „fluture” datorită aspectului său „înaripat”. Acest fluture este elementul de calcul de bază a FFT, transformând 2 puncte complexe în altele două tot complexe.

Figura 1.8 arată structura întregului algoritm FFT. Decompoziţia domeniului de timp este realizată cu un algoritm de sortare inversă a biţilor. Transformarea datelor descompuse în domeniul de frecvenţe nu implică nici o acţiune şi de aceea nu apare în figură.

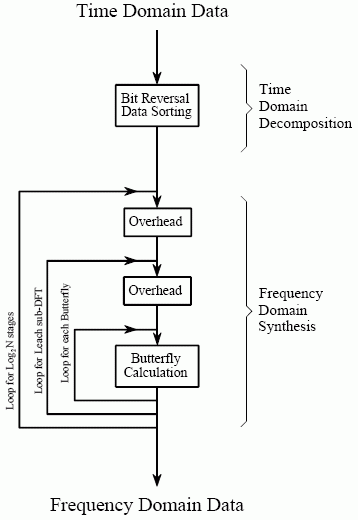


Fig. 1.8 Diagrama de calculare a FFT [1]

Sinteza în domeniul de frecvenţă necesită 3 bucle. Cea mai exterioară parcurge log2N etape. Cea din mijloc parcurge fiecare spectru în parte la etapa în care este prelucrat. Cel mai interior ciclu foloseşte celula fluture pentru a calcula punctele din fiecare spectru.

**1.2. Procesarea electrocardiogramei – detectarea vârfurilor R**

1.2.1. Prelucrarea semnalelor electrocardiografice. Monitorizarea activităţii cardiace implică achiziţia şi prelucrarea cvasipermanentă a ECG, realizate în timp real (rapid, prin circuite specializate) sau „off-line” (pe ECG memorate). Tehnicile semiautomate de prelucrare a ECG uşurează analiza acesteia şi înlătură subiectivismul interpretării.

Frecvenţa cardiacă instantanee sau perioada instantanee a ciclului cardiac se poate măsura pe ECG prin detectarea undelor R. În monitorizare, afişarea ritmului cardiac instantaneu are loc prin comanda bazei de timp de către undele R. Impulsul astfel format trebuie întârziat cu 0.2 sau cu 0.25 sec. pentru a se vizualiza toate semnalele ce apar într-un ciclu cardiac. Pe ecran declanşarea spotului are aceeaşi poziţie indiferent de ciclul cardiac, dar spotul se termină la o distanţă proporţională cu durata ciclului. Astfel, ecranul poate fi etalonat în unităţi de timp sau/şi în bătăi/minut.

Histograma intervalelor şi histograma frecvenţelor – variaţia într-un timp de observare t a ritmului cardiac (sau a ciclurilor cardiace, intervalul R-R) poate fi determinată prin numărarea ciclurilor care au avut o anumită valoare de timp (figura 1.9a). Similar se poate calcula histograma frecvenţelor cardiace (b).

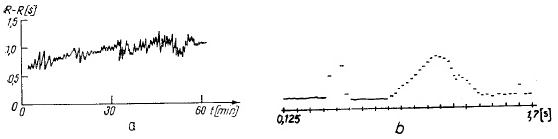


Fig. 1.9 Variaţia duratelor ciclurilor cardiace (a) şi histograma acestor durate (b) [1]

Prelucrări automate ale ECG – în multe cazuri clinice analiza vizuală a unei ECG nu poate stabili exact diagnosticul pacientului, datorită numărului mare de parametri (cca. 20 pentru o singură derivaţie) care trebuie analizaţi şi corelaţi între ei. În plus, diagnosticarea precisă implică informaţii corelate de la toate cele 12 derivaţii.

Mărimea ce caracterizează un parametru se împarte în anumite domenii, limitele acestor domenii fiind impuse de natura anomaliilor în funcţionarea cordului. De exemplu, pentru aritmii: dacă frecvenţa este în domeniul 15-40 băt/min există bradicardie sinusală pronunţată; între 40-60 băt/min – brahicardie sinusală; 60-110 băt/min – ritm sinusal normal; 110-130 băt/min – tahicardie sinusală. Valoarea parametrului trebuie analizată în toate ECG aferente celor 12 derivaţii.

Alte metode moderne de filtrare a semnalului ECG folosesc filtrarea multibandă, tehnici „wavelet” (ce permit analiza în domeniul timp-frecvenţă), operatori morfologici (eroziune, expandare, închidere, deschidere).

Fazele prelucrării semnalului ECG:

a) Detecţia perioadei:

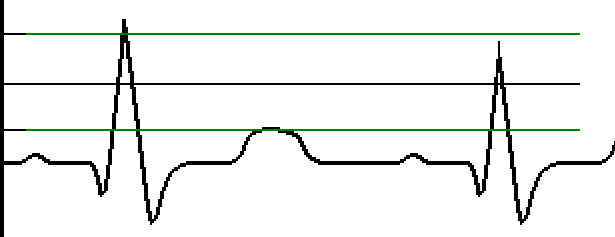


Fig. 1.10 Detecţia perioadei – metoda intersecţiei de nivel [6]

b) Eliminarea artefactelor:

* linia de zero;
* netezirea (0.25 + 0.5 + 0.25).

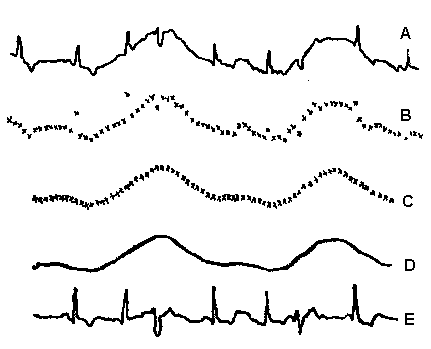


Fig. 1.11 Eliminarea artefactelor [6]

c) Tipificarea QRS:

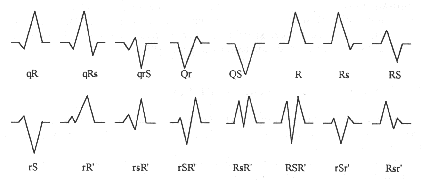


Fig. 1.12 Tipificarea QRS – recunoaşterea formelor de complexe QRS [6]

d) Tipificarea ST – T:

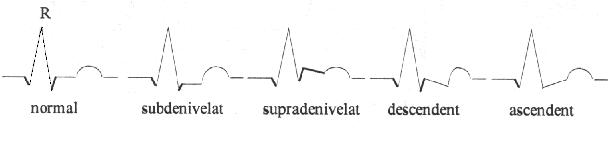


Fig. 1.13 Tipificarea ST – T (boli coronariene) [6]

e) Detecţia undelor P;

f) Selecţia ciclului tipic pentru calcule.

1.2.2. Algoritm de detecţie a vârfurilor R. Sarcina de bază la prelucrarea electrocardiogramei este detectarea vârfurilor R. Aici însă pot să apară unele dificultăţi: distanţa iregulară dintre vârfuri, forma iregulară a acestuia, prezenţa componentei de frecvenţă joasă în semnal datorată respiraţiei pacientului, etc. Pentru rezolvarea acestei sarcini procesarea trebuie să conţină în particular etape pentru reducerea influenţei factorilor.

Pentru descrierea funcţionării metodei fie un semnal ECG digital care constituie datele de intrare:

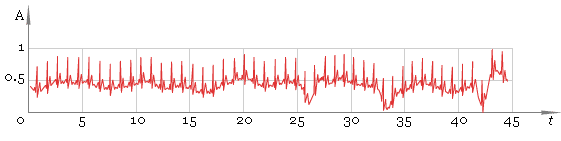


Fig. 1.14 Semnalul original [4]

După cum se observă acest semnal este neuniform. Astfel primul pas necesar este „netezirea” acestui semnal. Adică în limbaj matematic asta înseamnă că trebuie înlăturată componenta de frecvenţă joasă. Ideea principală este aplicarea transformatei Fourier rapidă (FFT), înlăturarea frecvenţelor joase şi restabilirea semnalului cu ajutorul FFT inverse. Rezultatul procesării semnalului utilizând transformata Fourier rapidă este în figura următoare:

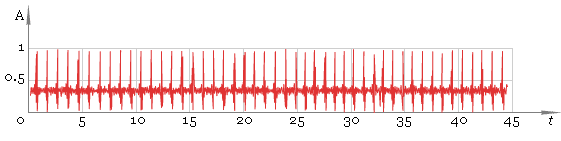


Fig. 1.15 Semnalul ECG filtrat cu ajutorul FFT [4]

Următorul pas este găsirea punctele de maxim local. Pentru aceasta se utilizează un filtru tip fereastră care „vede” doar maximul din încadrarea sa şi ignoră toate celelalte valori. Iniţial se utilizează o fereastră de dimensiuni predefinite şi se obţine:

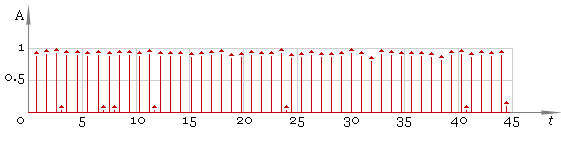


Fig. 1.16 Semnalul ECG filtrat – primul pas [4]

Acum trebuiesc înlăturate valorile mici şi păstrate cele semnificative. Aici se utilizează un filtru de nivel (threshold filter):

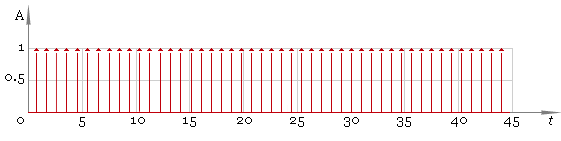


Fig. 1.17 Vârfurile R detectate [4]

În cazul dat s-a obţinut un rezultat bun, dar în general nu putem fi siguri că s-au detectat toate vârfurile. Deci pasul următor este ajustarea lăţimii filtrului tip fereastră şi repetarea filtrării. Comparând rezultatul obţinut (figura 1.18) cu cel din figura 1.13 se observă că calitatea filtrării este mult mai bună.

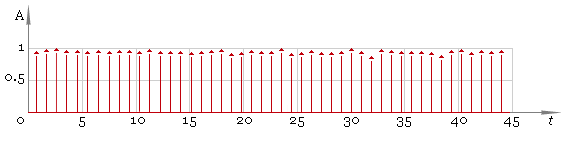


Fig. 1.18 Semnalul ECG filtrat – al doilea pas [4]

Repetând operaţiile necesare se obţine rezultatul final al detectării vârfurilor R:

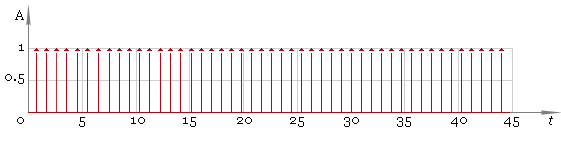


Fig. 1.19 Vârfurile R – rezultat final [4]

**1.3. Prelucrarea numerică a semnalelor**

Prelucrarea numerică a semnalelor are aplicaţii în orice domeniu în care informaţia poate fi prezentată sub formă numerică. Câteva dintre avantajele acestui mod de prelucrare a semnalelor sunt:

a) Acurateţe garantată – determinată de numărul de biţi folosiţi în reprezentarea semnalului;

b) Reproductibilitate perfectă – se obţin performanţe identice de la unitate la unitate, dacă nu variază toleranţele componentelor, de exemplu o înregistrare numerică poate fi copiată sau reprodusă fără vreo degradare a calităţii semnalului;

c) Nu are abateri cu temperatura sau vechimea;

d) Sistemele de PNS pot fi realizate sub formă de circuite integratecare prezintă siguranţă crescută, gabarit redus, putere mică, cost mic;

e) Flexibilitate crescută – sistemele de PNS pot fi programate şi reprogramate pentru a realiza o varietate de funcţii, fără modificarea hardului;

f) Performanţe superioare – sistemele de PNS pot realiza funcţii inaccesibile prelucrării analogice, de exemplu obţinerea unui răspuns de fază liniară, implementarea de algoritmi pentru filtrarea adaptivă.

Evident, există şi dezavantaje ale PNS:

a) Viteză şi cost – sistemele de PNS pot fi scumpe când sunt implicate semnale de bandă largă. În prezent, convertoarele analog/numerice şi numeric/analogice sunt costisitoare sau nu au suficientă rezoluţie pentru aplicaţii PNS de bandă largă. Timpul necesar conversiei limitează viteza de lucru. Obişnuit, numai circuitele integrate specializate pot procesa semnale în domeniul MHz şi sunt scumpe. Semnale de bandă mai mare de 100 MHz se prelucrează numai analogic;

b) Timpul de proiectare – uneori proiectarea unui circuit poate consuma nejustificat de mult timp;

c) Problema lungimii finite a cuvintelor – în situaţiile de prelucrare în timp real, consideraţii economice impun ca algoritmii PNS să fie implementaţi pe un număr limitat de biţi. Dacă acesta nu este suficient pentru a reprezenta variabilele, apar degradări serioase ale performanţelor circuitului. Sistemele numerice sunt afectate de zgomotul de cuantizare al convertoarelor analog/numerice, care este cu atât mai mare cu cat numărul de biţi folosit în reprezentarea eşantioanelor semnalului de intrare este mai mic. Mai mult, în timpul prelucrării, datorită operaţiei de rotunjire, apare un zgomot care, prin acumulare, poate conduce la instabilitate pentru sistemele de ordin superior.

Prelucrarea numerică a semnalelor implică reprezentarea, transmisia şi prelucrarea semnalelor folosind tehnici numerice şi procesoare numerice, deci, se poate spune că PNS se ocupă cu reprezentarea numerică a semnalelor şi utilizarea procesoarelor numerice pentru a analiza, modifica sau extrage informaţii din semnale.

Domeniul prelucrării numerice a semnalelor este foarte dinamic, ajungându-se, în funcţie de aplicaţie la dezvoltarea unor algoritmi şi metode de analiză foarte sofisticate. Semnalele şi sistemele se analizează în domeniul timp şi frecvenţă pentru a le determina caracteristicile. În domeniul timp un filtru numeric este caracterizat de răspunsul la impuls{h[n]}. Suma de convoluţie permite determinarea ieşirii {y[n]}, cunoscute fiind secvenţa de intrare {x[n]} şi răspunsul la impuls. Cunoaşterea răspunsului la impuls permite determinarea stabilităţii filtrului. Ecuaţiile cu diferenţe constituie o descriere alternativă a filtrelor în domeniul timp, utilă în implementarea lor.

De obicei, specificaţiile filtrelor se dau în domeniul frecvenţă, motiv pentru care va fi folosită transformata Fourierpentru examinarea proprietăţilor semnalelor şi sistemelor în acest domeniu. Transformata Fourier a răspunsului la impuls {h[n]} determină funcţia de transfer H(ω)a filtrului şi reprezintă câştigul filtrului la diferite frecvenţe. Transformata Fourier a unei secvenţe {x[n]} defineşte spectrul X(ω)al acesteia. Transformata Fourier discretăeste folosită pentru analiza spectrală cu ajutorul calculatorului numeric, folosind algoritmi rapizi de calcul. Tot pentru analiza semnalelor şi sistemelor discrete se foloseşte o tehnică mai generală oferită de transformata Z, cu ajutorul căreia se obţine o interpretare facilă a răspunsului în frecvenţă al filtrului. Funcţia de sistem H(z) este transformata Z a răspunsului la impuls. Metodele de sinteză implică aflarea coeficienţilor pentru satisfacerea specificaţiilor dorite ale filtrelor. De asemenea, sunt prezentate câteva metode simple de obţinere a unor filtre numerice selective de frecvenţă.

Procesarea analogică versus procesarea numerică – marea majoritate a sistemelor de natură precum şi din unele procese tehnologice sunt de tip continuu (analogic). Prelucrarea semnalelor analogice, se face de către echipamente analogice, care din punct de vedere teoretic pot fi privite ca sisteme analogice. Prin toate aceste echipamente, semnalul analogic se propagă de la intrare la ieşire, suferind anumite condiţionări, prelucrări, pe care le putem denumi generic procesări analogice.

Tehnicile şi tehnologiile moderne obligă tot mai adesea la utilizarea calculatorului în prelucrarea semnalelor. Relaţia dintre procesarea numerică de semnal şi semnalul analogic din care provine semnalul prelucrat este sintetizată în figura 1.20.

Faţă de această schemă bloc generală pot exista şi alternative. Astfel dacă procesarea se limitează doar al analiza unui semnal, atunci datele numerice nu mai sunt reconvertite în semnale analogice, ci sunt destinate exclusiv analizei şi stocării. În schimb dacă procesarea numerică serveşte doar la generarea sau simularea semnalelor analogice, atunci lipseşte CAN.

Procesarea Numerică a Semnalelor (Digital Signal Processing) nu prezintă altceva decât prelucrarea cu ajutorul calculatorului, prin operaţii matematice (adunări, înmulţiri, operaţii logice etc.), a semnalelor reprezentate numeric, având ca scop atingerea unor obiective dinainte propuse, specifice fiecărui domeniu de activitate.

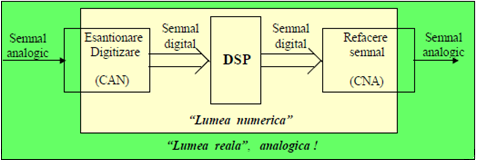


Fig. 1.20 Schema bloc a unei procesări numerice de semnal

Iată doar câteva din domenii care au beneficiat esenţial de avantajele procesării numerice a semnalelor:

* Comunicaţii: codarea / decodarea digitală a sunetului în telefonia digitală cu multiplexarea a mai multor convorbiri pe acelaşi fir, Fax-ul, Internet-ul etc.;
* Medicină: analiza semnalelor biomedicale (ECG, EEG, computer – tomografia etc.), diagnosticarea automată, monitorizarea diverselor funcţii vitale, proteze complexe etc.;
* Conducerea automată a proceselor: pilotarea automată a navelor, avioanelor, şi rachetelor, servomecanisme, roboţi, controlul proceselor industriale complexe sau periculoase;
* Radioul şi televiziunea digitală;
* Aplicaţii legate de vorbire: filtrare, recunoaşterea vorbirii, sinteza vorbirii;
* Multimedia: captarea, generarea, procesarea, transmiterea şi stocarea sunetului şi imaginilor; compact-discul.

Avantajele utilizării de procesare numerică de semnal faţă de sistemele analogice sunt următoarele:

* Flexibilitatea. Un sistem de prelucrare este în esenţă un algoritm de calcul, algoritm pe care îl efectuează un sistem de calcul (sistem cu microprocesor, calculator specializat, sau chiar un PC). Algoritmul poate fi uşor schimbat, prin reprogramare, ceea ce face ca sistemul să poată fi schimbat cu eforturi materiale nule. Deci, prin schimbarea algoritmului, sistemul îşi modifică comportarea, fără nici o modificare fizică a sistemului de calcul.
* Eficienţa economică. Procesarea numerică are avantaje economice deosebite. Dacă unui sistem analogic i se impune schimbarea caracteristicilor, comportării, el trebuie modificat fizic, ceea ce implică cheltuieli materiale, experimente şi noi teste de omologare. În cazul unui amplificator numeric pentru schimbarea comportării sale, i se va schimba prin programare doar o mică parte din algoritmul de calcul fără nici o modificare fizică a sistemului.
* Fiabilitatea. Un algoritm de calcul „nu rugineşte”. Bineînţeles că problema fiabilităţii unui sistem digital rămâne în seamă, dar ea depinde esenţial de fiabilitatea părţii hard a acestuia. Tehnologiile moderne de realizare a circuitelor numerice au ajuns la performanţe atât de înalte încât, şi din punct de vedere al fiabilităţii, partea hard a sistemelor digitale este comparabilă şi adesea superioară sistemelor analogice.
* Diagnoza. Legat de fiabilitate se pune şi problema întreţinerii, depanări sistemelor. Starea unui sistem trebuie să poată fi uşor controlată atât în procesul de punere în funcţiune cât şi mai ales în situaţii de avarie. Tot aşa de util este ca testarea să se facă şi în timpul, funcţionării normale, în scop profilactic. Din acest punct de vedere, sistemele digitale se comportă mai bine decât cele analogice la folosirea unor algoritmi care să testeze şi să furnizeze informaţii uşor de interpretat despre starea sistemului, ba chiar sa decidă folosirea unor resurse hardware de rezervă.
* Integrarea. Sisteme digitale pot realizate adesea, într-o singură capsulă de circuit integrat. Consecinţă a tehnologiilor moderne, integrarea are implicaţii pozitive asupra fiabilităţii şi costurilor.
* Adaptabilitatea. Odată realizat un algoritm de procesare numerică destinat unui anume sistem, este simplu ca el să poată fi folosit şi în alte aplicaţii prin simpla adaptare, ajustare a unor parametri. Mai mult chiar, în cadrul aceluiaşi proces algoritmul de calcul poate fi schimbat dinamic, adaptat la schimbările intervenite în proces.
* Stocarea şi transmisia performantă a datelor. Pentru stocarea datelor analogice suportul clasic îl reprezintă banda magnetică. Pentru datele numerice există soluţii net avantajoase de a stoca date mult mai rapid şi cu o densitate mult mai mare pe unitatea fizică de volum. Marele avantaj al folosirii semnalului numeric faţă de cel digital este însă imunitatea redusă la zgomot atât în procesul de stocare, cât şi în cel de transmisie.
* Performanţe superioare. Nu în ultimul rând, trebuie menţionat că performanţele sistemelor numerice sunt cel mai adesea superioare sistemelor analogice. Mai mult chiar, există numeroase tipuri de procesări care nici nu pot fi realizate în sistemele analogice, ca de exemplu filtre de ordin mare sau filtre având impuse anumite caracteristici de frecvenţă.

Analiza frecvenţială:

a) Reprezentarea semnalelor:

- temporal: Ampl = f (timp);

- frecvenţial (spectru): Ampl = f (frecv).

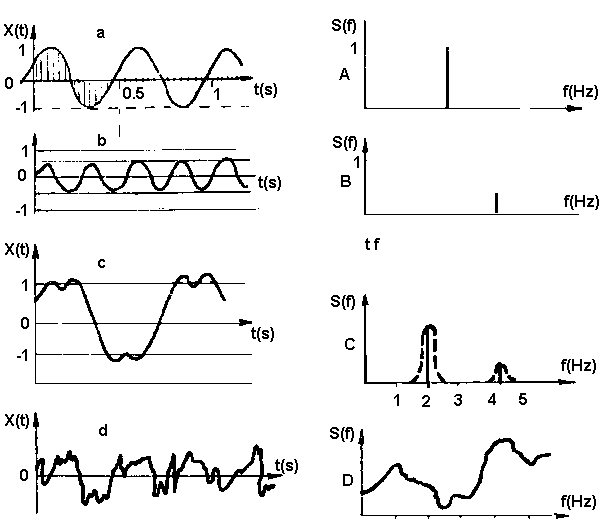


Fig. 1.21 Spectrele diferitelor semnale [6]

b) Analiza prin filtre:

- filtre trece-bandă (δ, θ, α, β);

- proporţia undelor – mingografe.

c) Analiza Fourier:

- domeniu: 0 - 30 Hz;

- tipuri de spectre:

* amplitudine;
* putere (proporţională cu A2).

d) Rezoluţia spectrală (~1/ΔT);

e) Constanta de timp;

f) Detecţia fenomenelor tranzitorii:

* contribuţia SPIKE-urilor în spectru;
* metoda ferestrei mobile (BERG).

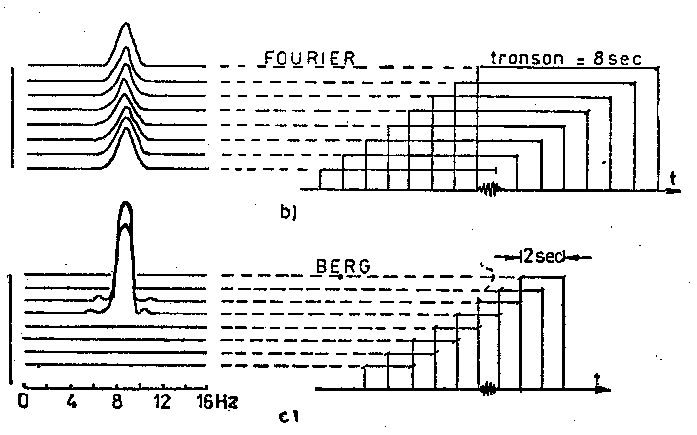


Fig. 1.22 Metoda ferestrei mobile Fourier şi BERG [6]

g) Reprezentări grafice:

* curbe, histograme;
* reprezentarea Bickford (evoluţia în timp a spectrelor).

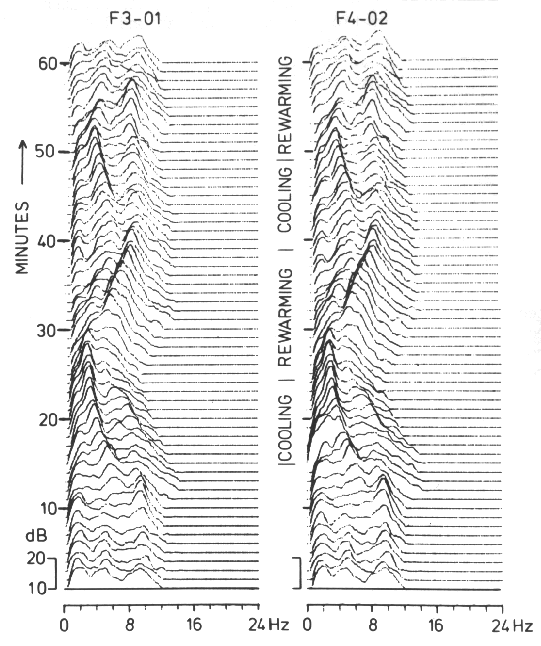


Fig. 1.23 Reprezentarea grafică Bickford [6]

h) Alte analize frecvenţiale:

* Walsh, Haar, zero crossing;
* filtrarea recursivă Kalman;
* funcţia de convoluţie.

**1.4. Medii de dezvoltare software**

Odată cu acceptarea beneficiilor aduse de interfaţa grafică a MS Windows (GUI) programarea pentru acest mediu a devenit foarte căutată. Programarea pentru Windows este diferită de cea orientată pe tranzacţii (old-style batch). O diferenţă esenţială între cele două este că un program rulat sub Windows procesează informaţiile utilizatorului cu ajutorul mesajelor de la sistemul de operare, pe când un program MS-DOS foloseşte sistemul de operare pentru a furniza informaţiile. Visual C#, produs de firma Microsoft, este un limbaj textual ce foloseşte un program pentru crearea de interfeţe grafice pentru a uşura munca programatorului.

Cu zece ani în urmă Visual Basic 1.0 a revoluţionat dezvoltarea aplicaţiilor pentru milioane de programatori. După mai mulţi ani de inactivitate pe piaţa compilatoarelor, Microsoft apare cu o tehnologie nouă, .Net(dot net), încercând să reducă cât de cât popularitatea crescândă a platformei Java pentru soluţiile Enterprise. Astăzi, Visual Studio.NET preia podiumul pentru următoarea decadă. Microsoft Visual Studio.NET este cel mai cuprinzător mediu de programare pentru crearea şi integrarea rapidă a serviciilor web (Web Services) şi al aplicaţiilor, mărind productivitatea dezvoltatorilor.

Spre deosebire de versiunea Visual Studio 6 când fiecare software inclus (C++, VB, ASP) avea câte un mediu de dezvoltare de sine stătător acum toate au un mediu de dezvoltare comun, deci, în acelaşi mediu de dezvoltare, se pot dezvolta C++, VB.NET, C# cu condiţia ca acestea să fie instalate în sistem. Din punct de vedere al tehnologiei, este de spus că pachetul Visual Studio .NET vine cu o tehnologie nouă şi anume .NET. Această tehnologie este implementată pe un nou framework, şi anume framework-ul .NET.

Proiectat cu o integrare adâncă a standardelor şi protocoalelor folosite în Internet, ca de exemplu XML (Extensible Markup Language) şi SOAP (Simple Object Access Protocol), Visual Studio.NET scurtează dramatic ciclul de proiectare al aplicaţiilor.

O dată cu lansarea .NET-ului, se pun bazele unei platforme care să fie atât de nivel jos, pentru a fi performantă şi pentru a uşura portabilitatea, cât şi de nivel înalt pentru a permite utilizarea tuturor conceptelor avansate de programare (obiecte, servicii, securitate, etc.).

Ultimele două decenii au aparţinut limbajelor C şi C++. Ambele furnizează programatorului un control fin, însă acest lucru are un preţ în productivitate. În comparaţie cu alte limbaje cum ar fi Microsoft Visual Basic, aceleaşi aplicaţii dezvoltate în C şi C++ necesitau mai mult timp pentru a fi realizate. Datorită complexităţii şi timpului îndelungat asociat acestor limbaje, mulţi programatori au căutat limbaje care să le ofere un mai bun echilibru între putere şi productivitate.

În ziua de azi există limbaje de programare cu productivitate ridicată, însă cu sacrificarea flexibilităţii cu care erau obişnuiţi programatorii de C şi C++. Asemenea soluţii reprezintă constrângeri pe care dezvoltatorii nu le acceptă uşor.

Soluţia ideală pentru programatori ar fi dezvoltarea rapidă, combinată cu puterea de a accesa toate funcţionalităţile platformei de bază. Aceştia vor un mediu sincronizat complet cu standardele Web care apar, şi care asigură integrare facilă cu aplicaţiile deja existente.

.NET este un cadru (Framework) de dezvoltare software unitară care permite realizarea, distribuirea şi rularea aplicaţiilor desktop Windows şi aplicaţiilor WEB.

Tehnologia .NET pune laolaltă mai multe tehnologii (ASP, XML, POO) şi limbaje de programare (VB, C++, C#, J#) asigurând totodată atât portabilitatea codului compilat între diferite calculatoare cu sistem Windows, cât şi reutilizarea codului în programe, indiferent de limbajul de programare utilizat.

.NET Framework este o componentă livrată împreună cu sistemul de operare Windows. De fapt, .NET 2.0 vine cu Windows Server 2003, se poate instala pe versiunile anterioare, până la Windows 98 inclusiv; .NET 3.0 vine instalat pe Windows Vista şi poate fi instalat pe versiunile Windows XP cu SP2 şi Windows Server 2003 cu minimum SP1.

Pentru a dezvolta aplicaţii pe platforma .NET este bine să avem trei componente esenţiale:

* un set de limbaje (C#, Visual Basic .NET, J#, Managed C++, Perl, Fortran, Cobol, Lisp, Pascal, etc.);
* un set de medii de dezvoltare (Visual Studio .NET, Visio);
* şi o bibliotecă de clase pentru crearea serviciilor Web, aplicaţiilor Web aplicaţiilor desktop Windows.

.NET Framework este ultima interfaţă între aplicaţiile .NET şi sistemul de operare şi actualmente conţine:

* Limbajele C#, VB.NET, C++ şi J#. Pentru a fi integrate în platforma .NET toate aceste limbaje respectă specificaţii POO numite Common Type System (CTS). Ele au ca elemente de bază: clase, interfeţe, delegări, tipuri valoare şi referinţă, iar ca mecanisme: moştenire, polimorfism şi tratarea excepţiilor;
* Platformă comună de executare a programelor numită Common Language Runtime (CLR), utilizată de toate cele 4 limbaje. CTS face parte din CLR;
* Ansamblu de biblioteci necesare în realizarea aplicaţiilor desktop sau Web numit Framework Class Library (FCL).

Microsoft Visual Studio este un mediu de dezvoltare integrat (IDE) de la Microsoft. Acesta poate fi utilizat pentru a dezvolta în consolă şi interfaţă grafică aplicaţii Windows Forms, site-uri web, aplicaţii şi servicii web, atât în cod nativ cât şi în cod administrat, pentru toate platformele suportate de către Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework şi Microsoft Silverlight.

Visual Studio include un code editor ce suportă aşa numitele IntelliSense şi code refactoring. Debugger-ul lucrează atât ca debegger la nivel de sursă cât şi ca debugger la nivel de cod maşină.

Alte instrumente integrate ar fi designer-ul de ferestre pentru aplicaţiile Graphical User Interface, designer web şi designer-ul database schema.

Limbajele de programare incluse de Visual Studio sunt C/C++, VB.NET, C#. Suportul pentru alte limbaje, cum ar fi Chrome, F#, Python şi Ruby, este pus la dispoziţie prin intermediul unor servici ce sunt instalate separat.

Limbajul C# fost dezvoltat de o echipă restrânsă de ingineri de la Microsoft, echipă din care s-a evidenţiat Anders Hejlsberg (autorul limbajului Turbo Pascal şi membru al echipei care a proiectat Borland Delphi). C# este un limbaj simplu, cu circa 80 de cuvinte cheie, şi 12 tipuri de date predefinite. El permite programarea structurată, modulară şi orientată obiectual, conform perceptelor moderne ale programării profesioniste. Principiile de bază ale programării pe obiecte (încapsulare, moştenire, polimorfism) sunt elemente fundamentale ale programării C#.

**2. PROIECTAREA, IMPLEMENTAREA ŞI DESCRIEREA SISTEMULUI**

**2.1. Arhitectura generală a softului**

Acest soft este descris în limbajul C#. Acesta este un limbaj cu cod gestionat, adică obiectele neutilizabile sunt înscrise într-o zonă aparte de către modulul de colectarea a gunoiului (garbage collection). Astfel nu este cazul să se monitorizeze strict resursele din memoria heap a sistemului. Se pot crea obiecte noi fără să se elibereze zona de memorie a aceluiaşi obiect.

Sistemul este proiectat pe baza bibliotecilor .NET Framework 4, utilizând tehnologia Windows Forms. S-a ales ultima versiune (v4.0.3) rulabilă în mediul de dezvoltare Visual Studio 2010. Acest mediu include şi comenzi LINQ (Language-Integrated Query), începând cu versiunea 2008, care extind sintaxa limbajului C#, pentru a extrage informaţiile necesare dintru set de date (lucru foarte necesar în manipularea cu bazele de date sau alte colecţii de date neomogene). O altă variantă ce ar înlocui Windows Forms este Windows Presentation Fundation, care se axează mult pe interfaţa sau designul aplicaţiei.

Limbajul C# prevede ca orice funcţie să fie definită şi descrisă în interiorul unei clase. Faţă de C/C++ nu se poate scrie cod ce este în afara domeniului de vizibilitate al clasei. O altă restricţie este că clasele trebuie să fie grupate în nume de spaţiu (namespace). În afara lor nu se permite să se afle nici o declaraţie. Astfel softul dat este compus din 2 nume de spaţiu, fiecare reprezentând câte un proiect (project). Primul este biosignals – proiectul de bază la rulare. Al doilea, GraphDisplayLib, este compilat ca bibliotecă dinamică (.dll) şi incorporat ca referinţă în primul proiect.

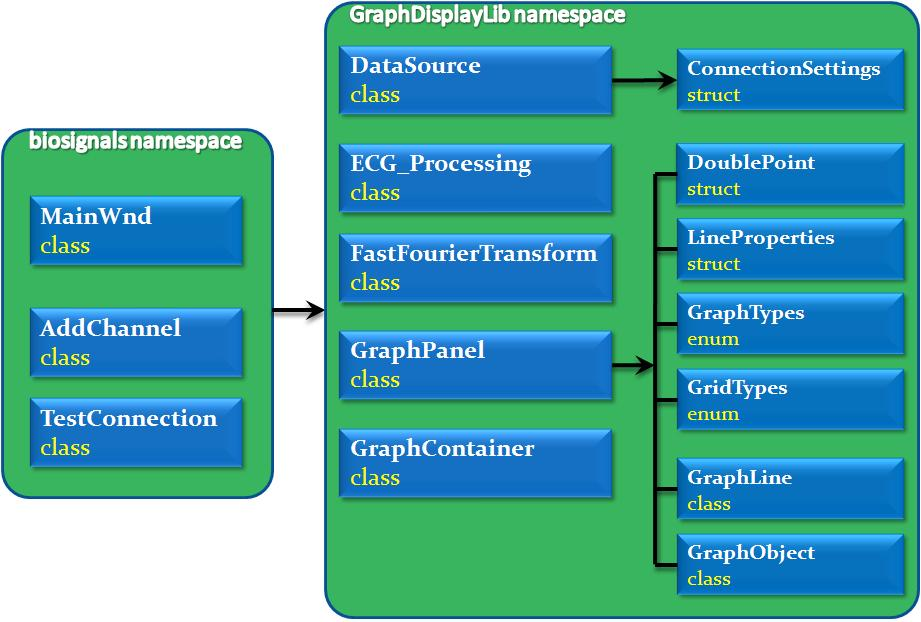


Fig. 2.1 Arhitectura generală a softului

GraphDisplayLib conţine în general clasele pentru afişarea unui semnal, clasele ce conţin funcţiile de procesare a semnalelor, clasa ce descrie protocolul de comunicare cu dispozitivul de achiziţie. La general dependenţa numelor de spaţii şi arhitectura sistemului este arătată în figura 2.1.

Unele clase sunt de tip formular şi participă la crearea interfeţei, altele sunt fără interfaţă grafică şi conţin funcţiile ce descriu funcţionalitatea sistemului. Aceste clase sunt:

* MainWnd – este clasa de bază pentru execuţia şi crearea interfeţei cu utilizatorul. Gestionează acţiunile provenite din exterior, execută comenzi de apelare a funcţiilor descrise.
* AddChannel – este o fereastră de dialog pentru selectarea canalului pentru înregistrare şi achiziţie, cât şi alegerea filtrelor prin care va trece acest semnal. Ea conţine 2 constructori supraîncărcaţi, iar pentru a întoarce valorile selectate sunt definite 3 proprietăţi: ChannelValue, HPFValue, LPFValue. Tot prin intermediul acestei ferestre se poate înlătura un canal din listă.
* TestConnection – este un formular pentru verificarea conexiunii şi a transferului de date între porturile seriale. Se pot testa porturile existente în calculator, cât şi cele virtuale create automat de sistemul de operare la conexiunea unei interfeţe USB-COM.
* DataSource – descriere protocolul de comunicare cu dispozitivul, asigură conexiunea cu acesta şi achiziţia datelor, transmite comenzi şi interpretează pachetele obţinute. Mai conţine structura ConnectionSettings pentru setarea parametrilor conexiunii la portul serial necesar.

Pentru a asigura că toate comenzile ajung la dispozitiv, în caz că acesta nu reuşeşte să le prelucreze, este implementată o coadă de mesaje, fiecare element al căreia este o structură ce codifică comanda. Această coadă este citită periodic de un timer. O comandă se va expedia numai dacă s-a obţinut răspunsul de confirmarea a execuţiei precedentei comenzi.

* ECG\_Processing – aici sunt până ce două funcţii: pentru detectarea vârfurilor R şi pentru descrierea funcţionalităţii filtrului de tip fereastră, ce participă la selectarea poziţiei acestor vârfuri.
* FastFourierTransform – permite execuţia FFT asupra unui semnal de intrare. Conţine diferite forme a datelor de intrare (complex sau real) cerând separat vectorii de intrare-ieşire sau un singur vector unde se va întoarce şi rezultatul. Este descrisă desigur şi metoda transformatei Fourier rapidă inversă, cât şi metoda de deplasare sau centrare faţă de frecvenţa zero a spectrului obţinut.
* GraphContainer – este un obiect de tip UserControl, o derivaţie a clasei de bază Control, specifică tuturor obiectelor din mediul Windows Forms. Acest obiect este creat cu scopul grupării mai multor grafice şi automatizarea aranjării sau poziţionării lor în cadrul acestui container. Are o definiţie mai specială faţă de celelalte clase şi anume prin utilizarea atributului Designer, în care se indică că obiectul dat moşteneşte clasa ScrollableControlDesigner. Astfel să poate de defilat în cadrul acestui obiect chiar din mediul de redactare sau design al formularului.
* GraphPanel – este cea mai voluminoasă clasă şi conţine metodele pentru reprezentarea grafică a datelor în diferite moduri liniar, bare, histograme, stem, setarea unor proprietăţi de afişare ca grile, culori şi stiluri a liniilor, axe, fundal. Conţine unele tipuri de date locale pentru realizarea acestor funcţionalităţi. De exemplu fiecare linie este memorată ca un obiect aparte într-o listă dinamică. O proprietate specifică (Hold) este reprezentarea mai multor grafice pe aceeaşi suprafaţă, cu scalarea automată a acestora pentru a se reprezenta toate graficele necesare.

În anexă este reprezentat mai detaliat conţinutul acestor clase.

Deoarece se simte un oarecare nivel de abstractizare între semnalele ce vor fi înregistrare şi momentul în care acestea sunt achiziţionate, este cam dificilă crearea unei interfeţe simple şi comode, automatizarea proceselor de înregistrare în fişiere şi afişarea semnalelor în timp real. Aceste complicaţii au dus la aceea că sistemul este la curent limitat la afişarea semnalelor aplicate unei prelucrări. Cele înregistrate sunt vizibile doar după salvarea lor în fişiere. Astfel la rulare, aplicaţia arată cam sumbru, având următoarea interfaţă:

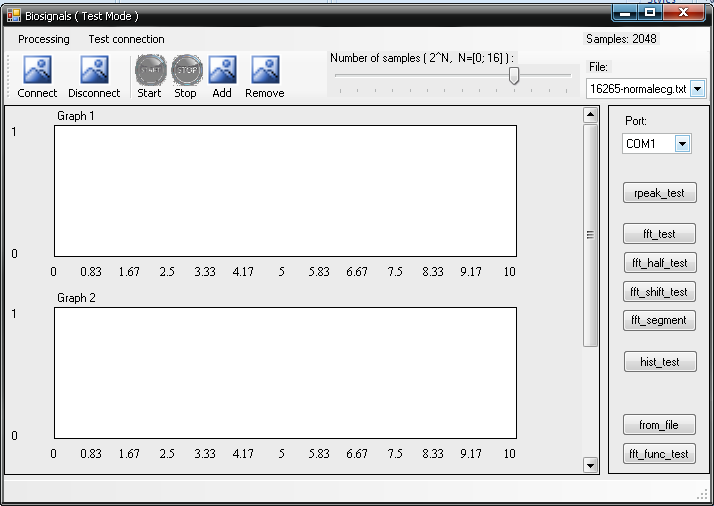


Fig. 2.2 Interfaţa sistemului în momentul execuţiei sale

Se observă cu uşurinţă deficienţele acestei interfeţe. Nici organizarea obiectelor de tip control nu este chiar lizibilă. Toate operaţiile ar trebuie să fie disponibile din meniul principal, dar în regim de testare cu fişiere de alt format, faţă de cele pe care le salvează aplicaţia, s-a introdus panoul din dreapta. Pe acesta este şi o casetă derulantă pentru selectarea portului de conexiune serială, deoarece încă nu este implementat un algoritm de conectare şi detectare automată a dispozitivului de achiziţie.

Deoarece nu este creat nici un obiect de tip lupă, să adăugat un glisor în partea se sus din dreapta pentru a alege câte probe să fie desenate, pentru o vizibilitate mai clară.

**2.2. Realizarea algoritmului transformatei Fourier rapidă**

Există diferite metode pentru calcularea transformatei Fourier discrete (DFT), precum rezolvarea simultană a ecuaţiilor liniare sau utilizând metoda corelaţiei [1]. Transformata Fourier rapidă (FFT) este o altă metodă de calcul a DFT. În timp ce produce acelaşi rezultat ca şi alte abordări, este cu mult mai eficientă, deseori reducând timpul de calcul de sute de ori. Aceasta este aceeaşi îmbunătăţire ca şi zborul într-un avion cu reacţie faţă de mersul pe jos. În timp ce FFT necesită doar câteva zeci de linii de cod, aceasta este una dintre cei mai complicaţi algoritmi din procesarea digitală a semnalelor.

Deci algoritmul în pseudocod s-ar putea scrie ca [3]:

1) pregătirea datelor de intrare pentru sumare – plasarea lor într-o ordine convenabilă;

2) pentru fiecare nivel de sumare:

A) pentru fiecare exponentă ca factor a semiperioadei:

a) calculează acest factor;

b) pentru fiecare sumă a acestui factor:

i) calculează produsul dintre acest factor şi al doilea termen al sumei;

ii) calculează suma;

iii) calculează diferenţa.

FFT operează prin descompunerea unui semnal de N puncte din domeniul timpului în N semnale în acelaşi domeniu, fiecare fiind compus dintr-un singur punct. Deci pasul următor este calcularea a N spectre de frecvenţă ce corespund acestor N semnale din domeniul timpului. La final, aceste N spectre sunt sintetizate într-un singur spectru de frecvenţă.

Primul pas al reordonării este plasarea datelor de intrare în ordinea lor naturală ca în (1.7). De exemplu pentru N=8 avem {f0, f1, f2, f3, f4, f5, f6, f7} → {f0, f4, f2, f6, f1, f5, f3, f7}. Deoarece această nouă ordine s-a obţinut ca rezultat a separării succesive a termenilor în pari şi impari, în domeniul binar acesta arată ca ordonarea biţilor începând de la cel mai puţin semnificativ bit (figura 2.3).

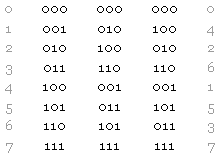


Fig. 2.3 Rearanjarea în domeniul binar [3]

Asta duce la o aritmetică oglindită – adică coloana binară rezultantă oglindită coincide cu coloana iniţială. Astfel, pentru a aranja elementele de intrare este de ajuns să numărăm într-o ordine oglindită. Deoarece oglindirea dublă întoarce acelaşi număr, rearanjarea se reduce la interschimbări.

Nivelele de sumare includ parantezele rotunde, pătrate şi nivelul din afară. În general asta duce la iteraţii pe perechi, cvadrupleţi, octade ş.a. Mai departe se iterează se componentele semiperioadei, iar a doua jumătate se obţine ca rezultat a luării diferenţei în loc de sumele pentru prima perioadă. Astfel pentru cel mai adânc nivel din paranteze perioada este 2 şi semiperioada este 1, ceea ce înseamnă că acest ciclu se va executa numai o dată. Pentru al doilea nivel perioada este 4 şi semiperioada este 2 şi cilul va fi executat de 2 ori. În caz general se obţine succesiunea 1, 2, 4, 8, etc.

Calcularea factorilor impune calcularea exponentei imaginare. Pentru micşorarea numărului de apeluri la funcţiile trigonometrice (1.2) se foloseşte recurenţa:

Trigonometric recurrence (2.1)

care este echivalentă cu expresia:

Trigonometric recurrence (2.2)

scrisă într-un mod mai şmecher pentru a nu pierde semnificaţia pentru δ mici – pentru acest caz cosδ≈1 şi sinδ≈δ, ceea ce indică că pentru cosδ memoria va fi umplută cu .99(9), dar pentru sinδ se va obţine o informaţie mult mai utilă. Astfel, (2.1) este doar un mod de a elimina cosδ din calcule. Dacă să ne reîntoarcem la (1.7) se va vedea că pentru N=8 δ=π/2, π/4 – numere nu chiar mici. Dar pentru transformarea la un N mult mai mare δ=π/2, π/4, π/8, ... , 2π/N şi la sigur pot fi foarte mici.

Cel mai adânc ciclu caută sumele, unde este prezentă exponenta imaginară calculată, determină produsul şi ia suma şi diferenţa, care este suma pentru a doua jumătate a perioadei la o distanţă egală cu π, unde exponenta îşi schimbă semnul, dar nu şi valoarea absolută în conformitate cu (1.9). Pentru efectuarea procesului de înlocuire se utilizează următoarea schemă:

Fig. 1. Butterfly.

Fig. 2.4 Schema fluture [3]

Această operaţie este una de bază la calcularea FFT prin înlocuire şi de obicei se numeşte „fluture”. Termenul de jos este multiplicat cu exponenta imaginară şi apoi suma termenilor este salvată în locul termenului de sus, şi diferenţa este salvată în locul termenului de jos. Imaginea generală a fluturilor este:

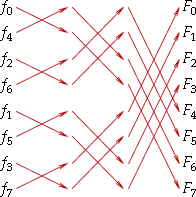


Fig. 2.5 Totalitatea fluturilor pentru N=8 [3]

Pentru implementarea acestei scheme în cod s-a declarat clasa FastFourierTransform:

public class FastFourierTransform

{

public bool FFT(Complex[] Input, Complex[] Output, uint N) {...}

public bool FFT(double[] Input, Complex[] Output, uint N) {...}

public bool FFT(Complex[] Data, uint N) {...}

public bool IFFT(Complex[] Input, Complex[] Output, uint N, bool scale = true) {...}

public bool IFFT(double[] Input, Complex[] Output, uint N, bool scale = true) {...}

public bool IFFT(Complex[] Data, uint N, bool scale = true) {...}

private void Rearrange(Complex[] Input, Complex[] Output, uint N) {...}

private void Rearrange(double[] Input, Complex[] Output, uint N) {...}

private void Rearrange(Complex[] Data, uint N) {...}

private void Perform(Complex[] Data, uint N, bool Inverse = false) {...}

private void Scale(Complex[] Data, uint N) {...}

public bool FFTShift(Complex[] Input, Complex[] Output, uint N) {...}

}

Clasa conţine metode publice pentru realizarea transformatei Fourier rapidă directă şi inversă, cu diferiţi parametri de intrare şi ieşire (funcţii supraîncărcate). Aceste metode sunt descrise în 2 moduri diferite: unul care cere separat vectorii datelor de intrare şi ieşire şi altul care cere un singur vector de intrate unde se salvează şi rezultatul transformării.

Celelalte funcţii private sunt pentru procesarea datelor – plasarea lor într-o ordine convenabilă, funcţia de bază pentru efectuarea transformării şi o funcţie auxiliară pentru scalarea rezultatului transformate inverse.

Aceste metode publice sunt aproape similare. De exemplu funcţia FFT directă este:

public bool FFT(Complex[] Input, Complex[] Output, uint N)

{

// Check input parameters

if (Input == null || Output == null || N < 1 || (N & (N - 1)) != 0) return false;

// Initialize data

Rearrange(Input, Output, N);

// Call FFT implementation

Perform(Output, N);

// Succeeded

return true;

}

Aici, mai întâi se verifică parametrii de intrare, apoi datele sunt pregătite şi rearanjate, după care urmează transformarea propriu-zisă. Funcţia pentru ordonarea vectorului de intrare foloseşte metoda matematică de oglindire descrisă mai sus pentru a defini o poziţie nouă pentru fiecare element şi a le schimba (vezi în anexă). În funcţia Rearrange() ultima buclă while incrementează cu unu într-o manieră de oglindire pentru a obţine poziţia de destinaţie (salvare) a elementului.

Dacă se analizează codul funcţiei Perform() se observă că este reflecţia exactă a algoritmului FFT descris şi a schemei fluture din figura 2.5. Iniţializările din interiorul primului ciclu (cel mai exterior) sunt doar pregătiri pentru calcularea succesivă a factorilor prin recurenţă trigonometrică. Execuţiile din cel mai interior ciclu efectuează operaţiile de fluture. Recurenţa trigonometrică de pe ultima linie este exact expresia (2.1).

Funcţia pentru realizarea FFT inversă este descrisă în acelaşi mod ca şi cea directă. Diferenţa dintre aceste două transformări este reflectată în prima linie a metodei Perform() unde se setează semnul corect pentru argumentul exponentei.

public bool IFFT(Complex[] Input, Complex[] Output, uint N, bool scale = true)

{

// Check input parameters

if (Input == null || Output == null || N < 1 || (N & (N - 1)) != 0) return false;

// Initialize data

Rearrange(Input, Output, N);

// Call FFT implementation

Perform(Output, N, true);

// Scale if necessary

if (scale) Scale(Output, N);

// Succeeded

return true;

}

Singura diferenţă este scalarea condiţionată a rezultatului după procesare. În mod implicit scalarea se face în acord cu relaţia (1.11), dar dacă cineva are necesitate doar de valori relative se poate de sărit peste această operaţie prin apelarea funcţiei cu parametrul scale = false.

Deci, algoritmul FFT implementat aici se numeşte Cooley-Tukey. Mai există şi algoritmul Sand-Tukey care rearanjează datele după efectuarea operaţiilor fluture şi în cazul ei aceste operaţii arată ca cele din figura 2.5, dar oglindite spre dreapta astfel încât „fluturii cei mari” apar primii, iar cei mici se îndeplinesc ultimii.

Din toate consideraţii le făcute rezultă că lungimea vectorului de intrare pentru algoritmul implementat trebuie să fie puterea lui 2. În caz contrar, se poate extinde mărimea acestuia spre cea mai apropiată putere a lui 2 şi completa spaţiile suplimentare cu zero sau cu un fragment din datele de intrare într-o manieră de periodicitate – doar se copie o parte necesară de la începutul semnalului. Completarea cu zerouri de obicei produce efectul scontat.

Dacă se trage atenţie se poate observa că operaţiile fluture din parantezele rotunde şi cele pătrate din (1.7) nu au într-adevăr nevoie de multiplicări, dar doar adunări şi scăderi. Astfel, optimizând cele două nivele mai profunde a fluturilor se pot îmbunătăţi performanţele pentru FFT.

**3. TESTAREA FUNCŢIONALITĂŢII SISTEMULUI PROIECTAT**

Pentru testarea softului s-au utilizat câteva semnale ECG salvate în fişiere textuale. Aceste semnale sunt doar nişte modele şi nu pot asigura pe deplin funcţionarea corectă a funcţiilor pentru alte cazuri reale. Rezultatele obţinute sunt doar informative. Pentru o testare completă se cere generalizarea modului de obţinere a datelor de intrare şi stocarea acestora.

Cel mai important modul este realizarea algoritmului FFT şi afişarea răspunsului acestuia. În figura 3.1 este reprezentat spectrul unui semnal ECG normal, iar în figura 3.2 este un semnal cu fibrilaţie atrială.

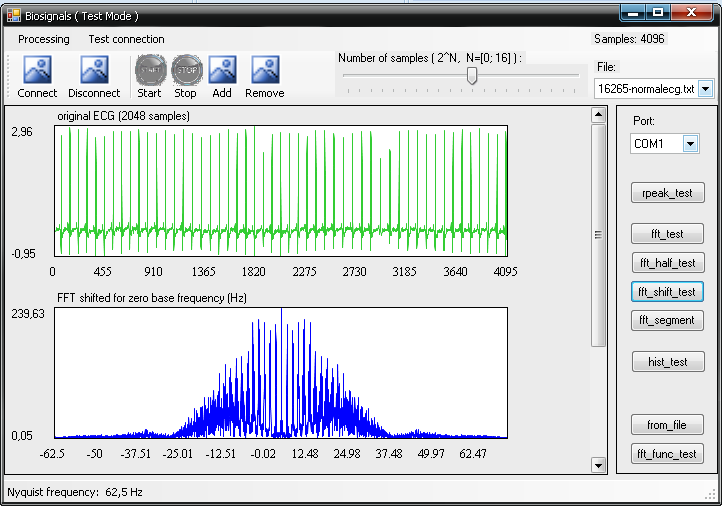


Fig. 3.1 Spectrul de frecvenţe a unui semnal ECG normal

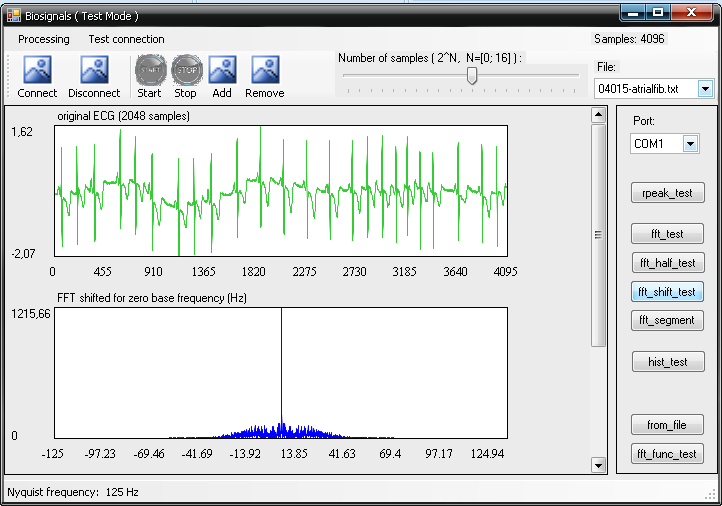


Fig. 3.2 Spectrul de frecvenţe a unui semnal ECG cu fibrilaţie atrială

Deoarece răspunsul pentru FFT este periodic şi se repetă, se reprezintă frecvenţele simetric faţă de frecvenţa zero din spectru. Adică se schimbă cu locurile prima şi a doua jumătate a vectorului răspuns. A doua jumătate conţine frecvenţele negative. Normal răspunsul transformatei Fourier este:

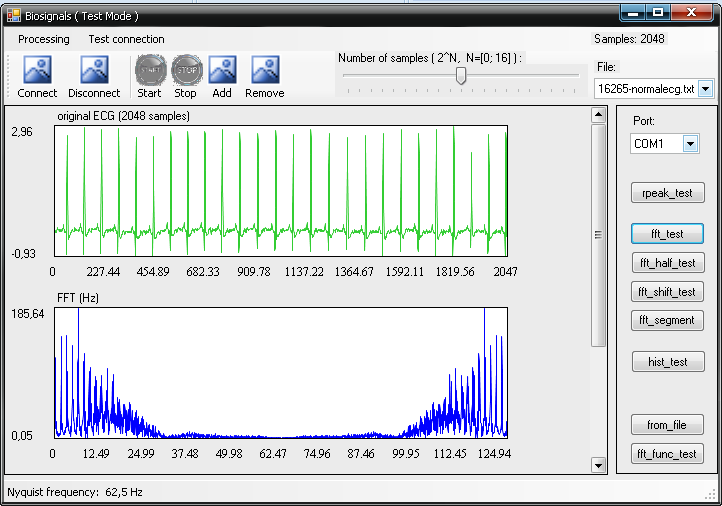


Fig. 3.3 Spectrul de frecvenţe obţinut de algoritmul FFT

Dacă se compară figura 3.1 cu 3.2 se observă că semnalul electrocardiografic normal este periodic, lucru bine evidenţiat şi în figura 3.3. În cazul unei maladii această proprietate nu este bine pronunţată şi în figura 3.2 se obţine un „impuls” de frecvenţe aproape de zero.

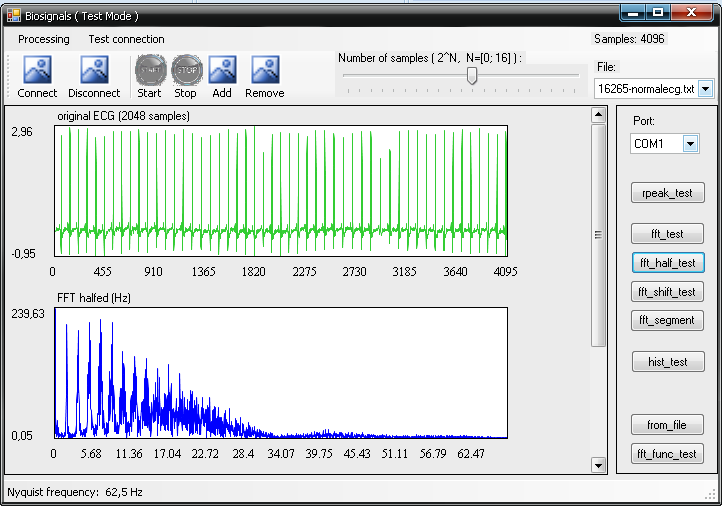


Fig. 3.4 Eliminarea periodicităţii din spectrul de frecvenţe

O altă modalitate de a reprezenta transformata Fourier este afişarea doar a primei jumătăţi din vectorul de frecvenţe (figura 3.4), cele pozitive, eliminând a doua jumătate care se diferă doar prin semn. Pe bara de stare a aplicaţiei este afişată şi frecvenţa Nyquist, care este jumătate din frecvenţa de eşantionare. Corectitudinea spectrului depinde şi de numărul de probe luate din semnal. Cu cât avem mai multe, cu atât şi spectrul este mai corect.

În program este prevăzută şi posibilitatea determinării FFT pe fiecare ciclu cardiac în parte:

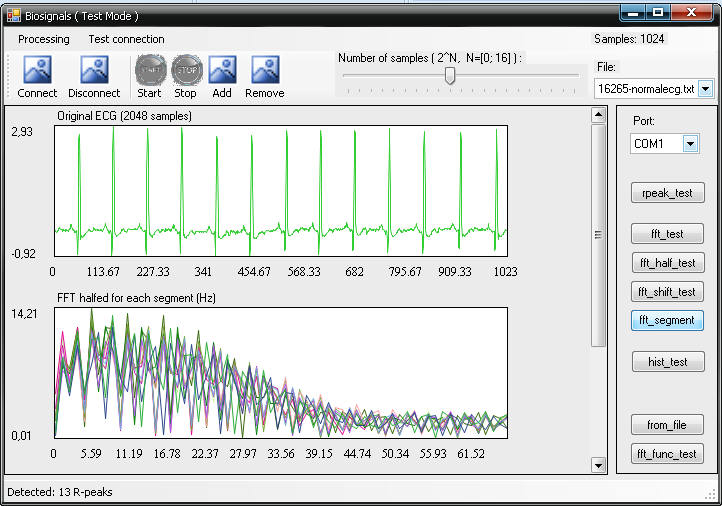


Fig. 3.5 Aplicarea FFT pe fiecare segment R-R determinat

Informaţia este cam dificilă de interpretat în 2D, dar modelarea unei suprafeţe necesită nişte algoritmi mai complecși care nu sunt prevăzuţi aici. O soluţie ar fi exportarea datelor şi importarea lor într-un mediul ca MatLab sau LabView, unde se poate încerca de a reprezenta în spaţiul variabilitatea acestor perioade R-R. Cu cât spaţiul este mai neted, cu atât şi ciclul cardiac îşi menţine un ritm normal şi exclude parţial posibilitatea unei eventuale aritmii cardiace.

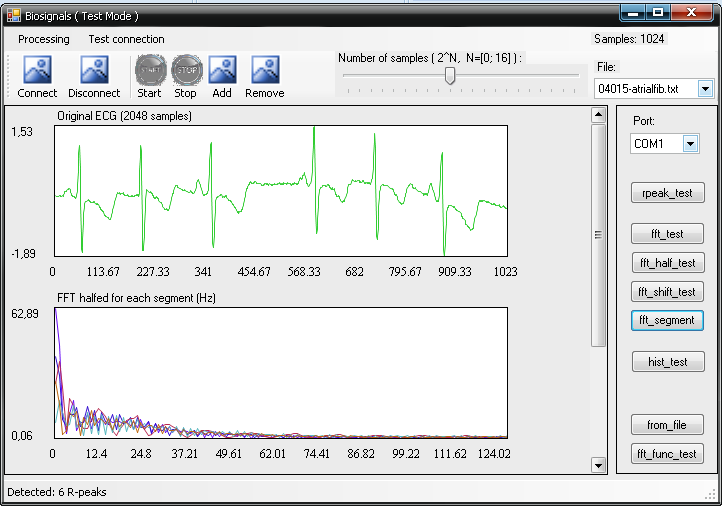


Fig. 3.6 Aplicarea FFT pe fiecare segment R-R determinat (semnal cu fibrilaţie atrială)

Pentru un semnal ECG în derivaţie precordială se obţine următorul spectru:

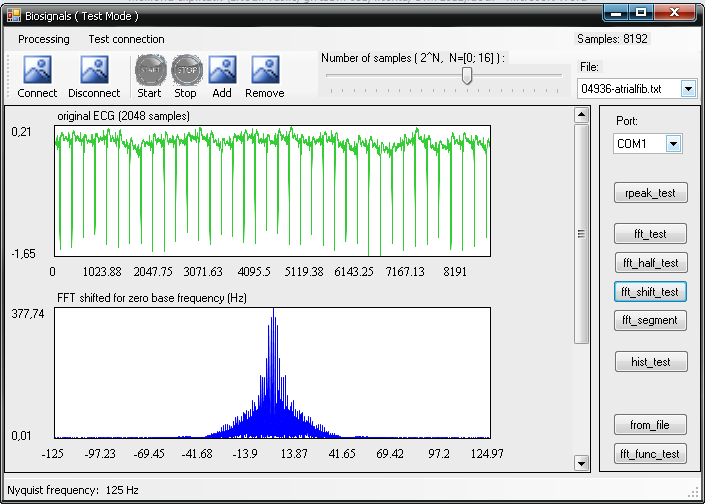


Fig. 3.7 Spectrul de frecvenţe a unui semnal ECG în derivaţie precordială

Pe baza algoritmului FFT şi a transformatei inverse IFFT se realizează şi detectarea vârfurilor R. În principiu se întoarce poziţia acestora în cadrul vectorului cu probele (semnalul) de intrare. Dacă se cunoaşte poziţia se poate uşor determina frecvenţa ritmului cardiac şi media acesteia. Pe imaginea următoare se pot vedea aceste date, cât şi numărul de vârfuri detectate:

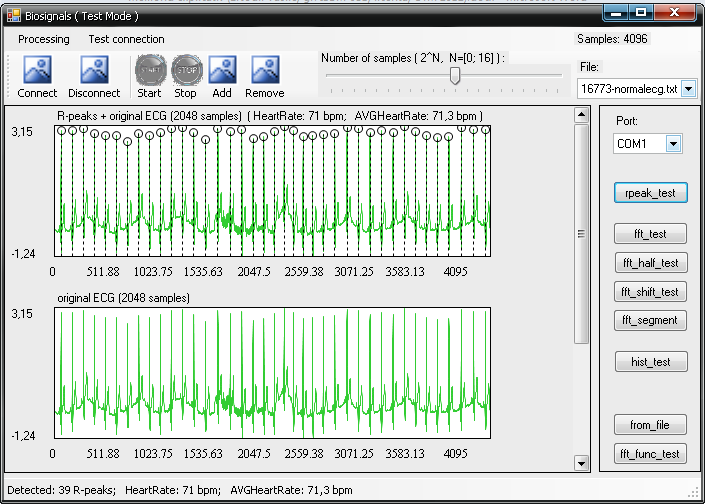


Fig. 3.8 Detectarea vârfurilor R pe o electrocardiogramă normală

Acest algoritm depinde şi de frecvenţa de eşantionare. Pentru o frecvenţă prea mică sau mai mare decât numărul de probe transmise se pot genera erori de tipul depăşirii indexului maxim al vectorului cu datele de intrare. Deoarece se foloseşte un filtru fereastră ajustabil algoritmul permite detectarea vârfurilor R şi în condiţii de variaţie a perioadei semnalului.

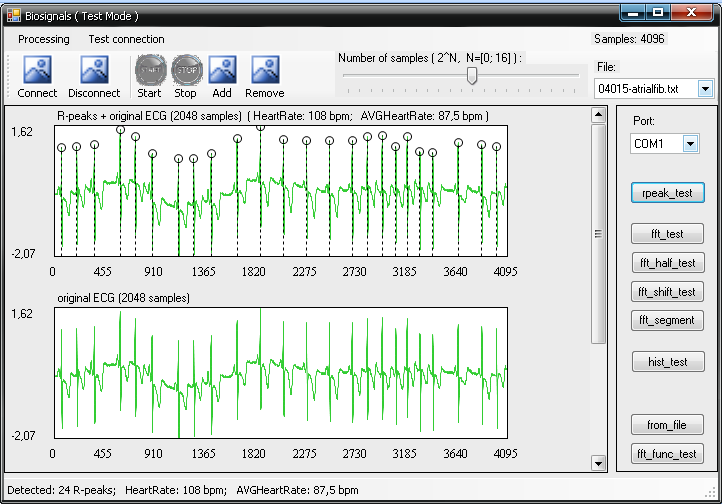


Fig. 3.9 Detectarea vârfurilor R pe o electrocardiogramă cu disfuncţie atrială

Pentru o analiză mai bună a variabilităţii ritmului cardiac sunt create histograme:

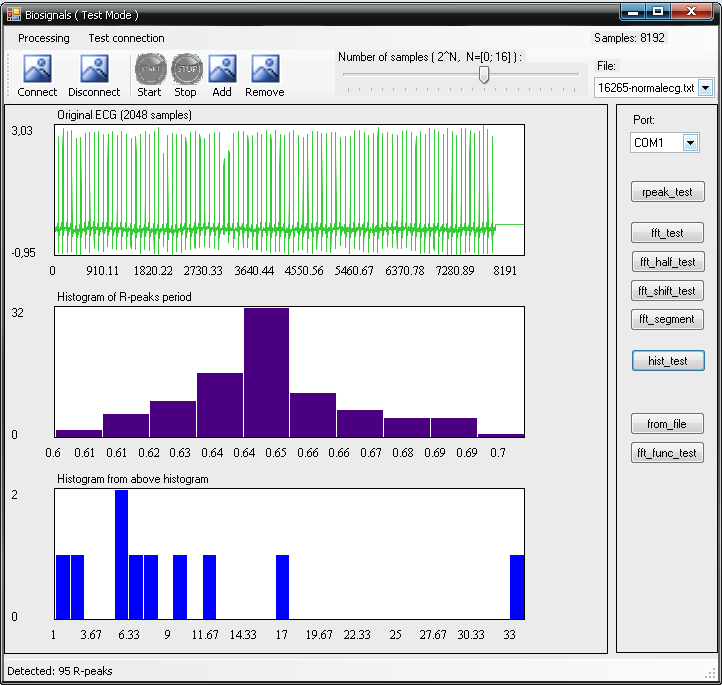


Fig. 3.10 Variaţia duratei segmentelor R-R pentru un semnal ECG normal

Pentru a se putea construi aceste histograme, este necesar mai întâi de determinat vârfurile R şi segmentele dintre acestea. Se calculează durata fiecărui interval şi aceste valori le reprezentăm pe prima histogramă. A doua rezultă din prima prin enumerarea grupurilor cu aceeaşi valoare a duratei (perioadei). În funcţie de anumite regiuni în care se pot separa ultima histogramă, se pot determina unele disfuncţii. Din aceste histograme se observă că pentru un semnal ECG normal această distribuţie a perioadelor este de tip Gauss având un spectru continuu centrat. Pentru un semnal cu aritmie acest lucru deja nu se respectă, obţinându-se o împrăştiere a valorilor. Informaţia este mai corectă dacă numărul de eşantioane este mai mare.

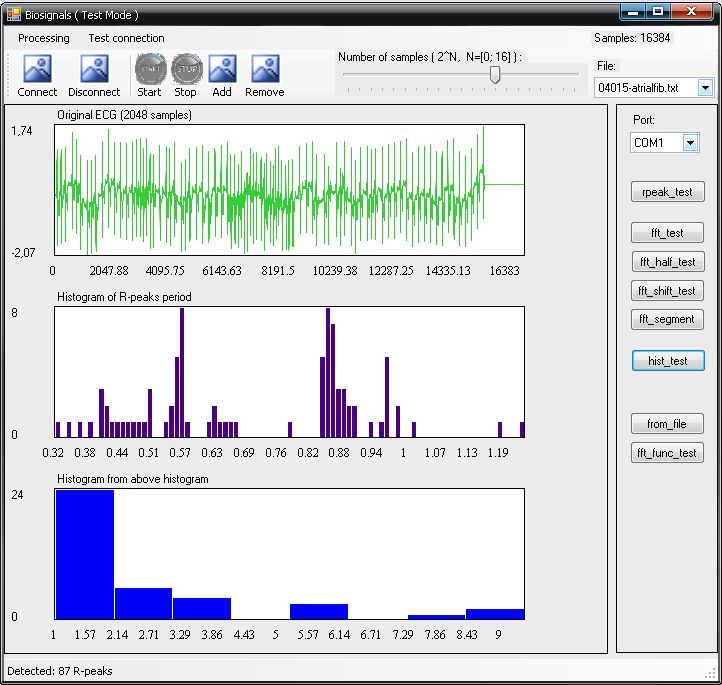


Fig. 3.11 Variaţia duratei segmentelor R-R pentru un semnal ECG cu fibrilaţie atrială

Pe lângă aceste funcţionalităţi softul are descris şi un protocol de comunicare cu un dispozitiv. Pe bara de instrumente de sub meniul principal sunt plasate butoane pentru acţionarea unor comenzi. Pentru adăugarea sau înlăturarea unui canal sursă pentru achiziţionare se folosesc butoanele Add şi Remove. La acţionarea lor se deschide o fereastră de tipul dat:

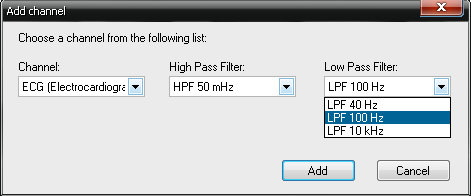
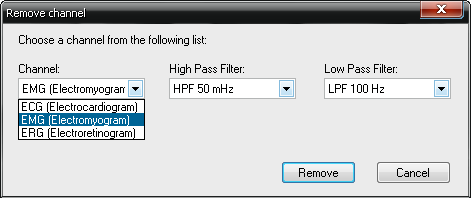
 

Fig. 3.12 Ferestrele pentru controlul semnalelor înregistrate

În figura 3.13 este redat un exemplu de simulare a comunicării cu dispozitivul. Dispozitivul (reprezentat de terminal) transmite o probă măsurată din semnalul ECG, care se arată în fereastra de testare a porturilor COM. Programul îi transmite comanda să adauge un nou canal pentru achiziţie (semnal ECG), vizibilă în secţiunea Receive a ferestrei terminalului. Lungimea pachetului transmis de controler este de 22 baiţi, iar cel expediat de comp de 7 octeţi.

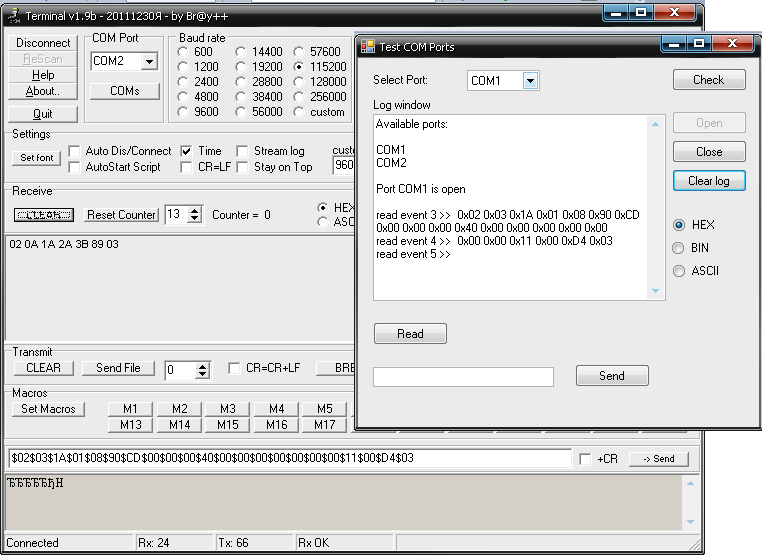


Fig. 3.13 Testarea protocolului de comunicare

Se poate menţiona că unele deficienţe se întâlnesc la detectarea vârfurilor R. Algoritmul pentru FFT este stabil şi nu creează probleme în utilizare. Se poate reprezenta cu uşurinţă spectrul pentru diferite lungimi a vectorului de intrare (numărul de probe). S-a testat acest algoritm şi pentru alte fişiere, de exemplu cele sonore. S-a observat o diminuate a vitezei de calcul pentru un număr foarte mare de eşantioane. Pentru reprezentarea spectrului pe fiecare segment R-R aparte, se cere un timp mai mare de procesare dacă numărul probelor este mai mare de 4096, deoarece acestea duc direct şi la creşterea numărului de eventuale segmente detectate.

**4. ARGUMENTAREA ECONOMICĂ A SISTEMULUI**

Orice sistem informaţional necesită anumite cheltuieli pentru proiectarea, organizarea şi funcţionarea lui. Ca urmare se pune întrebarea dacă aceste cheltuieli sunt justificate din punct de vedere economic. Totodată, reieşind din faptul că pentru diferite domenii pot fi elaborate mai multe variante de sisteme informatice, se cere selectarea soluţiei optime, adică a sistemului care necesită minim de cheltuieli în favoarea unei eficienţe maxime.

Argumentarea economică a oricărui proiect este necesară pentru demonstrarea eficienţei şi viabilităţii lui în condiţiile curente. Pentru aceasta se calculează bugetul proiectului, se prevede volumul vânzărilor posibile pentru cele comerciale, se determină rentabilitatea proiectului, etc. Se cere ca valorile calculate şi indicatorii financiari să fie actuali. Evaluarea eficienţei este chemată să justifice fundamentarea tehnico-economică a unui sistem informatic.

Pe lângă determinarea cheltuielilor de realizare şi a venitului (profitului), este necesară şi implementarea unei strategii de marketing, promovarea produsului prin diferenţierea sa faţă de alte produse similare şi aducerea unor argumente solide prin comparaţie, cu scopul convingerii clienţilor sau posibililor utilizatori că acest sistem este mai bun, mai optim pentru cerinţele lor. Urmărirea dezvoltării produselor concurenţilor şi luarea unor măsuri la timp ne poate salva de la un eşec. O descriere detaliată, dar clară, a sistemului poate atrage viitorii clienţi.

**4.1. Descrierea proiectului**

Proiectul dat este un sistem informaţional, iniţial conceput pentru un dispozitiv ca aplicaţie de interfaţă pe calculator. Acesta are rolul de a înregistra anumite semnale medicale, iar sistemul trebuie să le selecteze, înregistreze şi să aplice anumite analize asupra lor. Pentru aceasta se folosesc metodele matematice de analiză în frecvenţă (transformata Fourier).

Pentru a opera sistemul în general necesită o sursă de semnal, care l-a moment se poate citi din fişiere binare. Pentru a obţine aceste date este descris un protocol de comunicare cu dispozitivul şi toate operaţiile acestuia sunt dirijate din soft. Astfel putem transmite comenzi la dispozitiv pentru a alege ce semnal dorim să înregistrăm.

Semnalele sunt salvate în fişiere, dar trebuie de perfecţionat modul de înregistrare automatizat, independent de sursa de semnal ales. Pentru semnalul ECG este implementat modul de detectare a vârfurilor R-R şi perioadei acestora. Sunt create diferite moduri de reprezentare a transformatei Fourier sau aplicarea ei pe segmente. Sunt prevăzute şi unele histograme pentru analiza variabilităţii ritmului cardiac.

Din lipsă de experienţă este dificilă interpretarea rezultatelor analizei Fourier şi descrierea anumitor filtre digitale pentru detecţia vârfurilor undelor diferitor tipuri de semnale, în special când acestea nu au o periodicitate constantă. Pentru aceasta este implementat un filtru tip fereastră, care îşi auto-ajustează lăţimea sa pentru a detecta aceste amplitudini locale. Nu se exclude nici posibilitatea apariţiei unor erori când numărul de probe din semnalul iniţial este prea mic.

Ca soft de analiză este dificilă generalizarea unor module, dar se prevede implementarea unei mici baze de date a anomaliilor cardiace pentru stabilirea unui diagnostic automatizat, cât şi diferenţierea datelor pe pacienţi şi perioada de înregistrare.

**4.2. Analiza SWOT**

Pentru a identifica raţionalitatea proiectului trebuie să efectuăm analiza mediului intern şi extern al acestuia. Pentru aceasta se poate utiliza o metodă generală ca analiza SWOT. Această abreviere vine din limba engleză de la iniţialele cuvintelor Strenghts (puncte forte), Weaknesses (puncte slabe), Opportunities (oportunităţi) şi Threats (ameninţări). Analiza SWOT este de fapt o tehnică prin care se pot identifica punctele forte şi slabe şi se pot examina oportunităţile şi ameninţările unui proiect, ale unei acţiuni sau ale unei persoane şi poate fi utilizată ca element de realizare a bilanţului.

În general, există două moduri în care poate fi utilizată analiza SWOT: în scopuri profesionale sau personale. În context profesional, analiza SWOT poate fi utilizată pentru a măsura profitabilitatea unei afaceri sau a unui proiect.

Tabelul 4.1

Analiza SWOT

|  |  |
| --- | --- |
| Puncte forte | Puncte slabe |
| 1) Biblioteci separate, flexibile, pot fi utilizate şi în alte aplicaţii;  2) Reprezentarea FFT în diferite moduri;  3) Protocol de comunicare cu dispozitivul pentru selectarea diferitor filtre;  4) Control de adăugare/înlăturare asupra canalelor (semnalului) înregistrate;  5) Utilizarea celor mai noi biblioteci .NET Framework;  6) Viteză mare de procesare a semnalului digital. | 1) Modulul de comunicare depinde de configuraţia hardware a dispozitivului;  2) Sunt necesare unele cunoştinţe despre transformata Fourier şi spectre de frecvenţă;  3) Nu este implementat modulul de vizualizare în timp real a semnalelor înregistrate;  4) Nu este implementată baza de date a maladiilor şi disfuncţiilor cardiace, pentru automatizarea rezultatelor analizei. |

Continuarea tabelului 4.1

|  |  |
| --- | --- |
| Oportunităţi | Riscuri |
| 1) Pot fi adăugate noi funcţionalităţi cu mici modificări a arhitecturii softului;  2) Aplicaţia poate fi uşor extinsă şi pentru alte domenii sau semnale;  3) Sunt puţine aplicaţii sau biblioteci gratuite pe piaţă care permit analiza de frecvenţă a semnalelor biomedicale. | 1) Variabilitatea semnalelor şi a perioadei lor fundamentale poate produce unele excepţii (erori);  2) Conectarea altor derivaţii poate produce erori în timpul analizei; |

Pentru o eficienţă mai bună, este recomandabil să efectuăm periodic această analiză. Înlăturarea punctelor slabe şi a posibilelor riscuri poate să modernizeze sistemul, să ducă la apariţia un proiect nou, să convingă utilizatorii că proiectul continuă şi se dezvoltă. Proiectul poate deveni unul de succes dacă sunt identificate punte forte superioare celor de la alte produse similare a concurenţilor.

De exemplu în cazul proiectului dat, se pot micşora numărul punctelor slabe prin separarea modului de achiziţie de date faţă de cele de prelucrare şi afişare. Acest lucru va permite crearea unei interfeţe mai generale, sistemul informatic putând fi uşor extins şi pentru alte domenii. Implementarea corectă a unui modul de simulare sau reprezentare a semnalelor, rezultatelor în timp real, cu o automatizare coeficienţilor de scalare produce deja o impresie benefică a proiectului către utilizatori. Este mult mai comod să analizezi nişte date în format grafic, decât text.

**4.3. Planul calendaristic**

Acest plan reprezintă o descriere a acţiunilor ce trebuiesc întreprinse şi timpul necesar pentru efectuarea lor. Pentru a începe un proiect trebuie să stabilim obiectivele acestuia, să-l despărţim pe etape sau nivele şi să stabilim ce trebuie să se obţină la sfârşitul fiecărei perioade. Cel mai simplu mod de reprezentare a unui proiect în plan temporal este modelul în cascadă. În cadrul acestui model se poate continua cu următoarea etapă dacă cea precedentă a fost îndeplinită şi validată. Un dezavantaj al acestei metode însă este faptul că dacă am făcut o greşeală la proiectare, aceasta s-ar putea manifesta mai târziu la dezvoltarea proiectului. În acest caz devine dificilă întoarcerea la un punct complet funcţional şi ar putea fi necesar să facem unele reproiectări mai puţin plăcute.

În cadrul proiectului diplomantul (D) îndeplineşte funcţia de programator. Ca coordonator (supraveghetor) şi consultant se stabileşte o persoană ca conducător de proiect (C). Acesta ajută la luarea deciziilor majore de implementare a metodelor de analiză utilizate la descrierea softului, verifică îndeplinirea sarcinilor propuse. Pe parcursul desfăşurării activităţilor aceste două persoane conlucrează între ele pentru o bună funcţionarea a produsului final.

Tabelul 4.2

Planul calendaristic

|  |  |  |  |
| --- | --- | --- | --- |
| Denumirea acţiunii | Executanţi | Durata  (perioada) | Descrierea acţiunii |
| 1) Stabilirea temei proiectului | C  D | 5 zile  (06.02 – 10.02) | Se alege tema, sarcinile şi condiţiile iniţiale. Se stabilesc obiectivele şi scopul final al softului. |
| 2) Colectarea şi analizarea informaţiilor despre semnale. Caracteristicile lor | C  D | 15 zile  (13.02 – 02.03) | Stabilirea semnalelor ce vor fi înregistrate, banda lor de amplitudine şi frecvenţă. Determinarea principalilor parametri ce pot fi măsuraţi. |
| 3) Determinarea metodelor de prelucrare a biosemnalelor | C | 4 zile  (05.03 – 09.03) | Specificarea metodelor de analiză în frecvenţă, alegerea tipurilor de filtre digitale. |
| 4) Proiectarea generală pe module. Alegerea mediului de dezvoltare software | D | 5 zile  (12.03 – 16.03) | Descrierea arhitecturii sistemului, a relaţiilor dintre module (clase, biblioteci). Alegerea mediului de dezvoltare software C# şi a platformei .net Framework 4. |
| 5) Elaborarea interfeţei principale | D | 5 zile  (19.03 – 23.03) | Proiectarea interfeţei principale a programului şi alegerea componentelor vizibile. |
| 6) Înregistrarea, stocarea şi afişarea biosemnalelor | D | 18 zile  (26.03 – 20.04) | Crearea obiectelor pentru protocolul de comunicare şi de afişare a semnalelor, rezultatelor analizei. |
| 7) Elaborarea modulelor secundare | C  D | 12 zile  (24.04 – 11.05) | Descrierea unei componente de testare a conexiunii cu dispozitivul, codificarea canalelor şi asigurarea transmiterii comenzilor la dispozitiv. |
| 8) Testarea softului. Analiza şi corectarea rezultatelor | C  D | 5 zile  (14.05 – 18.05) | Verificarea funcţiilor claselor create prin transmiterea unor semnale de testare din fişiere. |
| 9) Validarea programului. Testarea în condiţii reale | C  D | 5 zile  (21.05 – 25.05) | Conexiunea dispozitivului la pacienţi şi efectuare analizei semnalului. |

Continuarea tabelului 4.2

|  |  |  |  |
| --- | --- | --- | --- |
| 10) Argumentarea economică | C  D | 7 zile  (28.05 – 05.06) | Analiza economică a proiectului. |
| 11) Conformarea lucrării | D | 3 zile  (06.06 – 08.06) | Finalizarea lucrării, structurarea memoriului explicativ. |

Durata desfăşurării proiectului este de 126 de zile sau 18 săptămâni (06.02 – 08.06). Din acest număr de zile au fost scoase zilele de odihnă sau week-end (36 zile) şi cele de sărbătoare desemnate de către Guvernul Republicii Moldova (6 zile): 8 martie, 16 şi 17 aprilie, 23 aprilie, 1 mai şi 9 mai. Deci conform calculelor avem în total 84 de zile lucrătoare în cadrul celor 4 luni de proiectare.

**4.4. Argumentarea economică**

Acest subpunct este compus din mai multe părţi care prevăd calcularea cheltuielilor pentru materialele utilizate, retribuirea muncii, cheltuieli indirecte. Acestea alcătuiesc preţul de cost necesar elaborării unei copii a sistemului informatic. În calcule se folosesc şi indicatori financiari care trebuie să fie actuali. Deci cu cât mai corect facem aceste calcule a bugetului necesar, cu atât şi preţul de realizare şi profitul dorit vor fi mai bine definite. În concluzie putem uşor determina care va fi rentabilitatea proiectului.

4.4.1. Cheltuieli materiale şi nemateriale. Pentru realizarea proiectului sunt necesare câteva active materiale şi mai multe nemateriale. Un avantaj al dezvoltării softurilor pe calculator este acela de a nu cheltui sume imense pe echipamente hardware. În cod este posibil să faci unele modificări fără implicarea unor cheltuieli adiţionale. Însă şi aici apar deseori probleme şi anume cele legate de licenţe. Deoarece sunt necesare câteva aplicaţii de dezvoltare software, biblioteci şi alte module, deseori acestea nu sunt gratuite (freeware) şi licenţele costă destul de mult. Dar în dependenţă de scopul şi complexitatea finală a proiectului se pot alege unele alternative open-source (aplicaţii cu acces public la cod) sau freeware.

Tabelul 4.3

Active materiale pe termen lung

|  |  |  |  |
| --- | --- | --- | --- |
| Denumirea obiectului | Preţul unităţii, lei | Cantitate | Suma, lei |
| Calculator | 6600 | 1 | 6600 |

Tabelul 4.4

Active nemateriale pe termen lung

|  |  |  |  |
| --- | --- | --- | --- |
| Denumirea obiectului | Preţul unităţii, lei | Cantitate | Suma, lei |
| Windows XP (SP3) | 2400 | 1 | 2400 |
| Sistemul MATLAB 6.5 | 1440 | 1 | 1440 |
| C# Edition 2010 | free | 1 | free |
| Bray++ Terminal | free | 1 | free |
| .net Framework 4 | free | 1 | free |

Tabelul 4.5

Consumuri materiale directe

|  |  |  |  |
| --- | --- | --- | --- |
| Denumirea obiectului | Preţul unităţii, lei | Cantitate | Suma, lei |
| Birotica (stilou, hârtie) | 10 | 1 | 10 |
| Cablu COM sau USB | 20 | 1 | 20 |

Deci după cum observăm componentul material de bază este calculatorul. Din resursele soft sau utilizat cele cu distribuire gratuită, pentru că satisfac pe deplin cerinţele necesare dezvoltării proiectului. Licenţa pentru acestea din categoria „Profesional” este destul de scumpă ajungând de exemplu pentru mediul Visual Studio 2010 Ultimate (pachet din care face parte şi C#) la 219566 lei. Biblioteca .net Framework, propusă în mai multe variante, este baza întregului sistem. Fără aceasta nici nu se poate discuta despre aplicaţii cu cod gestionat, din care face parte şi acest proiect.

4.4.2. Retribuirea muncii. La realizarea acestui proiect rolul principal îl ocupă programatorul (diplomantul). Pentru calcularea cheltuielilor de retribuire a muncii este stabilit un salariu lunar. Din acesta calculăm după volumul de lucru fondul de retribuire a muncii.

Programatorul – este persoana care cunoaşte la un nivel mediu limbajele de programare C# şi derivaţiile acestuia. Acesta se ocupă cu proiectarea şi descrierea sistemului în limbaj de programare, descrierea interfeţei cu utilizatorul. Sunt necesare unele cunoştinţe de lucru cu pachetul MatLab pentru efectuarea unor testări sau simulări.

Tabelul 4.6

Cheltuielile privind retribuirea muncii

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Funcţia îndeplinită | Cantitate, luni | Salariu, lei/lună | Suma (Frm), lei | Fondul social (FS), lei | Asigurarea medicală (AM), lei | Total, lei |
| Programator | 4 | 1500 | 6000 | 1380 | 210 | 7590 |

Pentru completarea acestui tabel am calculat valorile următorilor indicatori [7]:

1) Frm – Fondul de Retribuire a Muncii, in baza căruia se calculează suma contribuţiilor în Fondul Social (FS) şi valoarea primei de asigurare medicală obligatorie (AM).

Exemplu: Frm = 1500 lei/lună \* 4 luni = 6000 lei.

2) FS = Frm \* Cfs (%) (4.1)

unde Cfs – cota contribuţiilor de asigurări sociale de stat obligatorii, care se aprobă anual prin „Legea bugetului asigurărilor sociale de stat”.

Conform „Legii bugetului asigurărilor sociale de stat” pentru anul 2012, nr. 270 din 23.12.2011 (Monitorul Oficial nr. 15 din 17.01.2012), intrată în vigoare la 01.01.2012, cuantumul contribuţiilor de asigurări sociale rămâne neschimbat, aceasta fiind de 6% pentru angajat şi 23% pentru angajator calculat din fondul de salarizare lunar şi alte recompense.

Exemplu: FS = 6000 lei \* 0.23 = 1380 lei.

3) AM = Frm \* Cam (%) (4.2)

unde Cam – cota primei de asigurare obligatorie de asistenţă medicală, ce se aprobă în fiecare an prin Legea Republicii Moldova „Privind fondurile asigurării obligatorii de asistenţă medicală”.

Conform „Legii privind fondurile asigurării obligatorii de asistenţă medicală” pentru anul 2012, nr. 271 din 23.12.2011 (Monitorul Oficial nr. 15 din 17.01.2012), intrată în vigoare la 01.01.2012, cota primei de asigurare obligatorie de asistenţă medicală suportată de angajator constituie 3.5% din fondul de salarizare.

Exemplu: AM = 6000 lei \* 0.035 = 210 lei.

Suma totală a cheltuielilor privind retribuirea muncii este:

Frm + FS + AM = 6000 + 1380 + 210 = 7590 (lei).

4.4.3. Cheltuieli indirecte. În aceste cheltuieli intră cele care nu au fost introduse în categoriile anterioare: cheltuieli de regie (energie electrică, internet, apă) şi suma defalcărilor de amortizare pentru activele materiale şi cele nemateriale, care constituie partea importantă a cheltuielilor indirecte.

Fondul de amortizare (FA) se calculează uniform pe toată perioada decurgerii proiectului conform relaţiei următoare:

(4.3)

unde Costul – valoarea activului care trebuie sa fie amortizată;

T – termenul de funcţionarea utilă a activului;

T1 – durata de realizare a proiectului (pentru acest proiect este de aproximativ 4 luni).

Tabelul 4.7

Calculul fondului de amortizare a activelor materiale

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Denumirea activului | Cantitatea | Costul, lei | Termen de utilizare, luni | Suma, lei |
| Calculator | 1 | 6600 | 36 | 733.33 |

Tabelul 4.8

Calculul fondului de amortizare a activelor nemateriale

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Denumirea activului | Cantitatea | Costul, lei | Termen de utilizare, luni | Suma, lei |
| Windows XP (SP3) | 1 | 2400 | 24 | 400 |
| Sistemul MATLAB 6.5 | 1 | 1440 | 24 | 240 |

Exemplu de calcul: FAcalculator = 6600 / 36 \* 4 = 733.33 (lei);

AmortizareWinXP = 2400 / 24 \* 4 = 400 (lei).

Pentru calcularea energiei electrice consumate putem considera că s-au utilizat 2 lămpi cu incandescenţă cu o putere de consum de 60 W/h şi un calculator de 550 W/h. Dacă considerăm că am utilizat becurile câte 3 ore/zi şi calculatorul câte 8 ore/zi putem calcula cantitatea de energie cheltuită:

Ebecuri = 2 \* 60 W/h \* 3 ore/zi \* 84 zile = 30240 W = 30.24 kW;

Ecalculator = 1 \* 550 W/h \* 8 ore/zi \* 84 zile = 369600 W = 369.6 kW;

Etotal = Ebecuri + Ecalculator = 30.24 + 369.6 = 399.84 (kW) ≈ 400 (kW).

Conform Hotărârii Consiliul de administraţie al Agenţiei Naţionale pentru Reglementare în Energetică nr.410 din 15.04.2011 (Monitorul Oficial nr.63-64/320 din 20.04.2011) „Privind tarifele la energia electrică” [8], este stabilit un tarif (fără TVA) de 1.48 lei/kWh la energia electrică livrată de către Î.C.S. „RED Union Fenosa” S.A. pentru consumatorii finali instalaţiile cărora sunt racordate la reţelele de tensiune până la 0.4 kV.

Tabelul 4.9

Cheltuieli pentru regie

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Denumirea | Unitatea de măsură | Prețul, lei/unitate | Cantitatea | Suma, lei |
| Energie electrică | kW | 1.48 | 400 | 592 |
| Internet | luni | 150 | 4 | 600 |

Deci totalul cheltuielilor indirecte este:

FAactive materiale + FAactive nemateriale + Tregie = 733.33 + 640 + 1192 = 1373.33 + 1192 = 2565.33 (lei).

4.4.4. Preţul de realizare şi rentabilitatea proiectului. Din motiv că sistemul are un caracter comercial este nevoie de determinat preţul de cost al unei copii a programului elaborat. Preţul de cost se calculează pe o unitate – pe o copie de produs, pe un set, pe o licenţă pentru comoditate în determinarea preţului de realizare a unei unităţi către consumatorii finali.

Pentru determinarea preţului de realizare pe piaţă a programului elaborat utilizăm metoda „bottom-up”, care presupune calcularea următoarelor preţuri:

1) Preţ brut = Preţ de cost + Profit; (4.4)

2) Rentabilitate = Profit / Preţ de cost \* 100%; (4.5)

3) Preţ de realizare = Preţ brut + TVA. (4.6)

Preţul de cost include totalitatea cheltuielilor necesare pentru elaborarea proiectului. Acestea sunt reprezentate în tabelul următor:

Tabelul 4.10

Cheltuieli totale de elaborarea a proiectului

|  |  |
| --- | --- |
| Denumirea articolului | Suma, lei |
| Consumuri materiale directe | 30 |
| Cheltuieli directe privind retribuirea muncii, total | 7590 |
| Total FA pentru echipamente pe termen lung  Total regie | 1373.33  1192 |
| Total: | 10185.33 |

Profitul se calculează din preţul de cost. Deci pentru un profit de 17% şi TVA de 20% obţinem:

Profit = Preţ de cost \* 0.17 = 10185.33 \* 0.17 = 1731.51 (lei);

Preţ brut = 10185.33 + 1731.51 = 11916.84 (lei);

Rentabilitate = 1731.51 / 10185.33 \* 100% = 17%;

Preţ de realizare = 11916.84 + 11916.84 \* 0.2 = 14300.21 (lei).

În concluzie putem menţiona că preţul de cost al proiectului este unul relativ mic în comparaţie cu proiectele de o arie mai extinsă, care ar incorpora metodele utilizate aici. Cu o experienţă de lucru mai bună şi cu definirea concretă a obiectivelor finale a softului se poate reduce perioada de realizare la 2-2.5 luni. Desigur că se pot reduce şi cheltuielile de regie la organizarea corectă a timpului de desfăşurare a activităţilor.

Pentru a comercializa proiectul se cere analiza produselor similare existente şi implementarea unei funcţionalităţi diferite, care să se bazeze pe tehnici inteligente, pentru ca utilizatorul să înţeleagă cu uşurinţă modul de funcţionare şi să nu ceară de la acesta implicarea directă în orice acţiune.

**4.5. Partea organizatorică: Selecţia personalului: noţiune şi etapele de bază**

Problema selecţiei resurselor umane se pune ori de câte ori apare nevoia de a alege între două sau mai multe persoane pentru ocuparea unui post. Ea impune o analiză obiectivă a concordanţei dintre caracteristicile profesionale ale unui post şi posibilităţile fizice, psihice şi informaţionale pe care le prezintă solicitantul postului [9].

Selecţia resurselor umane trebuie abordată din mai multe puncte de vedere:

1) Abordarea din perspectiva economică – selecţia creează premisele pentru creşterea calităţii forţei de muncă, se reduc accidentele de muncă;

2) Abordarea de natură psihologică – interesele angajaţilor, abilităţile personale, motivaţia angajaţilor;

3) Abordarea sociologică – repartizarea corectă pe locuri de muncă şi relaţiile din cadrul grupului de muncă;

4) Abordarea medicală – contraindicaţiile pentru candidaţi de a ocupa unele posturi.

Într-un examen de selecţie se pleacă în primul rând de la cerinţele psihofiziologice, psihice şi sociale ale fiecărei profesii, cerinţe ce sunt sintetizate într-o profesiogramă.

Etapele selecţiei:

1) Analiza CV-urilor *–* cuprind informaţii cu privire la: numele şi adresa candidatului, vârsta, educaţie, calificare, experienţa, etc. Se mai foloseşte şi scrisoarea de motivare.

2) Interviu (convorbire) – este convorbirea directă dintre unul sau mai mulţi reprezentanţi ai organizaţiei şi candidat. Scopul interviului este de a permite organizaţiei să constate dacă este cazul să-şi dezvolte interesul preliminar faţă de candidat.

Există următoarele tipuri de interviuri:

a) Interviul structurat – întrebările sunt planificate în avans şi puse fiecărui candidat exact în aceeaşi ordine;

b) Interviul nestructurat – întrebările nu sunt planificate;

c) Interviul semistructurat – presupune o planificare flexibilă din partea intervievatorului, ceea ce permite acestuia adaptarea pe parcurs a întrebărilor.

În urma supunerii candidaţilor la interviu o parte din candidaţi vor fi reţinuţi fie pentru angajare imediată fie prin prezentare la testare, iar restul vor fi eliminaţi. Ca motive de eliminare pot fi:

* lipsa unei pregătiri şi a unor performanţe adecvate;
* slăbiciuni evidente care rezultă din aspectul şi comportamentul persoanei;
* incapacitatea de a îndeplini standardele minime;
* incapacitatea de a dispune de aptitudinile minime necesare;
* aspecte negative de personalitate;
* informaţii nefavorabile sau negative despre performanţele din trecut;
* lipsa unei capacităţi înnăscute a ambiţiei sau a altor trăsături necesare;
* inapt din punct de vedere fizic pentru post.

3) Testare – candidaţilor ajunşi în această etapă li se va întocmi în urma examenului de selecţie cu ajutorul testelor o psihogramă, care se vor corela cu datele din profesiogramă şi se vor reţine candidaţii cei mai potriviţi.

Orice test are anumite însuşiri diagnostice privind:

* Tipul de informaţie pe care-l solicită (teste de inteligenţă, teste de personalitate, teste de aptitudini, teste de cunoştinţe, etc.);
* Gradul de extindere şi profunzime a informaţiei pe care o solicită;
* Tipul de prelucrare a informaţiei solicitate (etalonarea – ansamblul operaţiilor şi procedeelor prin care se realizează clasificarea subiecţilor în funcţie de performanţele obţinute la diferite teste, mai apoi se va stabili analiza corelaţiilor şi regresiilor faţă de performanţa cerută într-un post).

4) Verificarea scrisorilor de referinţă;

5) Examenul medical;

6) Interviul final – se anunţă decizia de angajare. Se expediază scrisori atât la persoanele selectate, cât şi la cele refuzate. În general, participanţii sunt evaluaţi în funcţie de următoarele criterii:

* conducere, control;
* capacităţi de organizare şi de planificare, luarea deciziilor;
* aptitudini de comunicare orală şi scrisă, competenţa în privinţa relaţiilor umane;
* iniţiativă, energie, originalitate;
* capacităţi analitice, potenţial general;
* rezistenţă la stres;
* flexibilitate comportamentală.

Evaluarea recrutării şi selecţiei:

1) Rata recrutării – este raportul procentual între numărul de candidaţi potenţiali către numărul de candidaţi angajaţi;

2) Rata selecţiei – este indicatorul invers proporţional ratei recrutării.

Cu ajutorul acestor indicatori întreprinderea analizează atractivitatea sa pe piaţa muncii.

**CONCLUZII**

În cadrul acestui proiect s-a pus ca scop principal determinarea spectrului de frecvenţă a diferitor semnale medicale, adică trecerea acestora din domeniul de timp în cel de frecvenţă. Algoritmul FFT pentru determinarea spectrului unui semnal s-a implementat prin metoda Cooley-Tukey, o metodă destul de rapidă ce poate fi realizată şi pe un DSP (microcontrolere specializate în calcule matematice cu virgulă flotantă). Singura condiţie impusă de acest algoritm este ca lungimea vectorului de intrare să fie puterea lui 2. Acest lucru se poate ajusta prin „extinderea” vectorului. În acest proiect acesta este completat cu zerouri.

Rezultatele obţinute de la algoritmul FFT sunt reprezentate în diferite moduri. Prin intermediul acestei transformări este realizat şi algoritmul de determinare a vârfurilor R, în care FFT serveşte ca filtru a frecvenţelor joase. Prin intermediul FFT inversă se restabileşte semnalul în domeniul de timp. S-a implementat şi afişarea rezultatelor periodicităţii ciclului cardiac de pe o electrocardiogramă înregistrată ca histogramă. A rămas însă nesoluţionată problema determinării anumitor maladii atriale analizând histograma şi segmentarea ei în zone ce ar corespunde apartenenţei unei anumite disfuncţionalităţi. Se poate remarca că pentru determinarea vârfurilor R se foloseşte un filtru fereastră, care necesită să fie introdusă frecvenţa de eşantionare a semnalului. Asta impune unele piedici la utilizarea algoritmului atunci când se pot folosi diferite surse de semnale. S-a observat că pot apărea erori dacă frecvenţa de eşantionare este prea mare.

S-a aplicat transformata Fourier asupra fiecărui ciclu cardiac din cadrul unui semnal, dar nu este implementat un modul de reprezentare ca suprafaţă 3D a acestui răspuns de la fiecare segment. Reprezentarea 2D nu oferă în sine un răspuns clar, dar pentru modelarea unei suprafeţe se cere realizarea unor algoritmi complicaţi, care necesită destul de mult timp pentru cei fără experienţă.

O analiză în frecvenţă impune în sine şi reprezentarea grafică a rezultatelor. Pentru acesta este creată o clasă aparte pentru crearea unei suprafeţe de desenare, dar este una modestă ce nu permite utilizatorului să întreprindă careva măsuri în mod interactiv. Semnalele reprezentate sunt scalate ca să încapă în regiunea definită. Pentru o mai bună vizualizare s-ar cere crearea unui control de tip lupă, sau „deplasarea prin semnal”, stabilirea unor cursoare interactive de reprezentare a valorilor de pe grafic. Aceste module cer însă crearea unui obiect complex, care pot deveni prioritare în dezvoltarea aplicaţiei şi interacţiunea comodă a utilizatorului cu sistemul informatic.

Analiza spectrală a semnalelor medicale poate oferi unele informaţii relevante despre maladii sau disfuncţii „ascunse”, care uneori nu sunt vizibile utilizând metodele de înregistrare bazate pe potenţiale electrice. O filtrare corectă a semnalelor în domeniul de frecvenţă poate creşte probabilitatea determinării unor variaţii a periodicităţii acestora, fluctuaţiile undelor (impulsurilor) normale de activitate, reconstruirea corectă a semnalelor. Nerespectarea criteriului Nyquist de înregistrare şi eşantionare a semnalelor duce la determinarea greşită a spectrului semnalului.

Interpretarea rezultatelor FFT depinde şi de proprietăţile acesteia. Deoarece această transformare operează în domeniul complex, răspunsul este unul periodic şi de obicei se reprezintă doar jumătate din frecvenţele obţinute. În cealaltă jumătate acestea se repetă, constituind frecvenţele negative. De aceea spectrul devine cel mai vizibil la afişarea în simetrie faţă de frecvenţa zero. Aici frecvenţele mici tot timpul sunt situate mai spre centru, pe când cele mari sunt spre periferii. În dependenţă de ce bandă de frecvenţe ne interesează să vizualizăm, filtrăm acest spectru şi analizăm rezultatele obţinute. Încadrarea într-o bandă de frecvenţă sau în alta poate duce la concluzii preliminare despre starea normală sau abnormală a semnalului achiziţionat.

O problemă o constituie şi separarea completă a softului de sursa se semnale de intrare. Deoarece în program se utilizează fişiere binare, se cere ca semnalele achiziţionate să fie înscrise într-un format specific. Dacă acesta diferă pentru diferite surse, atunci ar trebui ca de fiecare dată o persoană competentă să creeze fişiere compatibile cu acest soft. Asta duce la demoralizarea utilizatorului, confuzii şi face ca sistemul să fie dependent de anumite cazuri concrete. Pentru universalitatea unui sistem informatic trebuie să se prevadă diferite situaţii şi să fie implementate anumite standarde. Modulele aplicaţiei trebuie să fie independente unul de altul, pentru a obţine o arhitectură flexibilă şi a permite modificarea softului fără mari cheltuieli. Acest lucru şi s-a urmărit în cadrul proiectului, care în mare măsură separă partea de analiză de cea de achiziţionare şi afişare. Însă pentru o arhitectură mai complexă şi sigură trebuie şi experienţă mai bogată în domeniul dezvoltării aplicaţiilor şi a modelelor de proiectare arhitecturală.

Din punct de vedere economic proiectul poate fi şi comercializabil, dar în primele sale versiuni poate fi mai mult ca aplicaţie de cercetare. Nu necesită multe resurse, cheltuieli şi pentru extindere sau adăugarea unor noi funcţionalităţi sunt necesare doar câteva persoane. Pentru hardware nu se cere nimic suplimentar, decât calculatorul în sine. Pentru resurse software necesită doar unele biblioteci ce pot fi descărcate gratuit de pe site-ul oficial al producătorului. Astfel cheltuielile sunt minime, dar şi complexitatea sistemului este redusă, în comparaţie cu unele proiecte naţionale pe platforme mobile şi de la distanţă, ca de exemplu E-Risc.

**BIBLIOGRAFIE**

1. Steven, W. S. The Scientist and Engineer's Guide to Digital Signal Processing. Hard cover edition, vol. 1, ch. 12, 1997.

http://www.dspguide.com/ch12.htm [Resursă electronică] accesat la 05.05.2012.

2. Cotfas, P. Signal Processing: Lecturer. Course 7.

http://ocw.cti.ac.at:8080/eduCommons/ccol/signal-processing/Signal\_Processing/content/ COURSE/Course7\_Discrete\_Fourier\_Transform.pdf [Resursă electronică] accesat la 05.05.2012.

3. Fast Fourier transform – FFT. Digital signal processing (DSP) software development.

http://www.librow.com/articles/article-10 accesat la 04.04.2012.

4. ECG processing – R-peaks detection. Medical digital signal processing software development.

http://www.librow.com/cases/case-2 accesat la 03.04.2012.

5. http://myweb.msoe.edu/~martynsc/signals/ecg/ecg.html accesat la 30.04.2012.

6. Ceangă, E., Munteanu, I., Bratcu, A. Semnale, circuite şi sisteme. Partea I: Analiza semnalelor. Editura Academica, Galaţi, 2001.

http://www.etc.ugal.ro/imunteanu/Csemnale\_bmk.pdf accesat la 12.04.2012 [Resursă electronică] accesat la 15.05.2012.

7. http://www.contabilsef.md/libview.php?l=ro&id=4872&idc=13 accesat la 04.06.2012.

8. http://www.anre.md/rate/index.php?vers=1&sm=9 accesat la 04.06.2012.

9. http://www.venividi.org/cretu/download/mru5.doc [Resursă electronică] accesat la 04.06.2012.

10. http://www.manager.ro/articole/resurse-umane/selectia-personalului-172.html accesat la 07.06.

11. Hamilton, P. S. Open Source ECG Analysis Software Documentation. 2002.

http://www.eplimited.com/osea13.pdf [Resursă electronică] accesat la 22.04.2012.

12. Afonso, V. X. ECG QRS Detection, ch. 12.

http://www.physik.uni-freiburg.de/~severin/ECG\_QRS\_Detection.pdf [Resursă electronică] accesat la 22.04.2012.

13. ECG review – ACLS Program Ohio State University Medical Center.

https://edr.medctr.ohio-state.edu/Downloads/ACLS/ECGRhythmsCharEx.pdf[Resursă electronică] accesat la 22.04.2012.

14. http://www.ecglibrary.com/ecghome.html [Resursă electronică] accesat la 03.05.2012.

15. http://en.ecgpedia.org/ [Resursă electronică] accesat la 03.05.2012.

**ANEXĂ**

**Descrierea claselor şi a unor metode (codul sursă parţial)**

using ...

namespace biosignals

{

public partial class **MainWnd** : Form

{

private DataSource data1;

public MainWnd() {...}

private void Form2\_Load(object sender, EventArgs e) {...}

private void startbt\_Click(object sender, EventArgs e) {...}

private void stopbt\_Click(object sender, EventArgs e) {...}

private void connectbt\_Click(object sender, EventArgs e) {...}

private void disconnectbt\_Click(object sender, EventArgs e) {...}

private void rpeaksDetectionToolStripMenuItem\_Click(object sender, EventArgs e) {}

private void fFTToolStripMenuItem\_Click(object sender, EventArgs e)

{

openFileDialog1.InitialDirectory = Application.StartupPath;

openFileDialog1.ShowDialog();

if (openFileDialog1.FileName.Length == 0) return;

BinaryReader r = new BinaryReader(File.OpenRead(openFileDialog1.FileName));

FileInfo fi = new FileInfo(openFileDialog1.FileName);

if (fi.Length == 0)

{

MessageBox.Show("This is an empty file !", "Read file error");

toolStripStatusLabel1.Text = "FFT operation canceled";

return;

}

int N = (int)Math.Ceiling(Math.Log(fi.Length, 2));

N = (int)Math.Pow(2.0, (double)N);

double[] Y = new double[N];

int i = 0;

try

{

for (i = 0; i < fi.Length; i++) Y[i] = (double)r.ReadByte();

}

catch (EndOfStreamException ex)

{

if ((long)i != fi.Length) toolStripStatusLabel1.Text="File is corupted !";

return;

}

Complex[] fftresult = new Complex[Y.Length];

FastFourierTransform ft = new FastFourierTransform();

bool a = ft.FFT(Y, fftresult, (uint) Y.Length);

if (!a)

{

MessageBox.Show("Can't execute FFT !","FFT error");

return;

}

graphPanel1.Title = "ECG ( " + openFileDialog1.SafeFileName + " )";

graphPanel1.Plot(Y, new LineProperties(Color.LimeGreen));

double[] X = new double[Y.Length];

for (i = 0; i < Y.Length; i++)

{

X[i] = (double) 6646 \* i / N;

Y[i] = fftresult[i].Magnitude;

}

graphPanel2.Plot(X, Y, new LineProperties(Color.Blue));

}

...

}

public partial class **AddChannel** : Form

{

private byte code;

public byte ChannelValue

{

get

{

switch(chValues.SelectedIndex)

{

case 0: code = GraphDisplayLib.DataSource.ECG; break;

case 1: code = GraphDisplayLib.DataSource.EMG; break;

case 2: code = GraphDisplayLib.DataSource.ERG; break;

}

return code;

}

}

public byte HPFValue {...}

public byte LPFValue {...}

public AddChannel(){...}

public AddChannel(string ctype) : this(){...}

private void AddChannel\_Load(object sender, EventArgs e) {...}

private void OKbt\_Click(object sender, EventArgs e) {...}

private void cancelbt\_Click(object sender, EventArgs e) {...}

}

public partial class **TestConnection** : Form

{

public TestConnection() {...}

private void Form1\_Load(object sender, EventArgs e) {...}

private void FormDesign() {...}

private void portslist\_SelectionChangeCommitted(object sender, EventArgs e) {...}

private void open\_Click(object sender, EventArgs e) {...}

private void close\_Click(object sender, EventArgs e) {...}

private void read\_Click(object sender, EventArgs e) {...}

private void send\_Click(object sender, EventArgs e) {...}

private void serialPort1\_DataReceived(object sender,SerialDataReceivedEventArgs e)

{

byte[] s = new byte[serialPort1.BytesToRead];

serialPort1.Read(s, 0, serialPort1.BytesToRead);

logwnd.Invoke(new MethodInvoker(delegate { datacomevent("\r\nread event " + (k++).ToString() + " >> ", s); }));

}

private void datacomevent(string header, byte[] data) {...}

...

}

}

namespace GraphDisplayLib

{

namespace Processing

{

public class **ECG\_Processing**

{

public double[] Detect\_RPeaks(double[] ecg, double samplingrate,int nFilter=1) //samplerate in Hz

{

Complex[] fresult = new Complex[ecg.Length];

FastFourierTransform ft = new FastFourierTransform();

// Remove lower frequencies

bool a = ft.FFT(ecg, fresult, (uint)ecg.Length);

if (!a)

{

MessageBox.Show("Can't execute FFT !", "FFT function Error !", MessageBoxButtons.OK, MessageBoxIcon.Error);

double[] err = {-1.0};

return err;

}

int v = (int)Math.Round(((double)fresult.Length \* 5) / samplingrate);

for (int i = 0; i < v; i++) fresult[i] = 0.0;

for (int i = fresult.Length - v; i < fresult.Length; i++) fresult[i]=0.0;

double[] corrected = new double[fresult.Length];

Complex[] corrected\_C = new Complex[fresult.Length];

a = ft.IFFT(fresult, corrected\_C, (uint)fresult.Length);

if (!a)

{

MessageBox.Show("Can't execute IFFT !", "IFFT function Error !", MessageBoxButtons.OK, MessageBoxIcon.Error);

double[] err = { -1.0 };

return err;

}

for (int i=0; i < fresult.Length; i++) corrected[i] = corrected\_C[i].Real;

// Filter - first pass

double WinSize = Math.Floor(samplingrate \* 571.0 / 1000.0);

if (Math.IEEERemainder(WinSize, 2.0) == 0.0) WinSize += 1;

double[] filtered1 = ECGWinMax(corrected, WinSize);

// Scale ecg

double[] peaks1 = new double[filtered1.Length];

double v1 = filtered1.Max() / 7.0;

for (int i = 0; i < peaks1.Length; i++) peaks1[i] = filtered1[i] / v1;

//Filter by threshold filter

for (int data=0; data<peaks1.Length; data++)

if (peaks1[data] < 4) peaks1[data] = 0;

else peaks1[data] = 1;

int[] positions = find(peaks1);

int distance = positions[1] - positions[0];

for (int data=0; data<positions.Length-1; data++)

if (positions[data+1]-positions[data]<distance) distance=positions[data+1]-positions[data];

// Optimize filter window size

double QRdistance = Math.Floor(0.04 \* samplingrate);

if (Math.IEEERemainder(QRdistance, 2.0) == 0.0) QRdistance += 1;

WinSize = 2.0 \* (double)distance - QRdistance;

// Filter - second pass

double[] filtered2 = ECGWinMax(corrected, WinSize);

double[] peaks2 = new double[filtered2.Length];

for (int i = 0; i < peaks2.Length; i++) peaks2[i] = filtered2[i];

for (int data=0; data<peaks2.Length; data++)

if (peaks2[data] < 4) peaks2[data] = 0;

else peaks2[data] = 1;

if (nFilter == 1) return peaks1;

else if (nFilter == 2) return peaks2;

else

{

MessageBox.Show("nFilter must be \"1\" or \"2\" !", "R-Peaks function Error !", MessageBoxButtons.OK, MessageBoxIcon.Error);

double[] err = { -1.0 };

return err;

}

}

private double[] ECGWinMax(double[] Original, double WinSize) {...}

private int[] find(double[] source) {...}

}

//Cooley–Tukey FFT algorithm

public class **FastFourierTransform**

{

public bool FFT(Complex[] Input, Complex[] Output, uint N) {...}

public bool FFT(double[] Input, Complex[] Output, uint N)

{

// Check input parameters

if(Input == null || Output == null || N<1 || (N & (N-1))!=0) return false;

// Initialize data

Rearrange(Input, Output, N);

// Call FFT implementation

Perform(Output, N);

// Succeeded

return true;

}

public bool IFFT(Complex[] Input, Complex[] Output, uint N, bool scale = true)

{

// Check input parameters

if(Input == null || Output == null || N<1 || (N & (N-1))!=0) return false;

// Initialize data

Rearrange(Input, Output, N);

// Call FFT implementation

Perform(Output, N, true);

// Scale if necessary

if (scale) Scale(Output, N);

// Succeeded

return true;

}

private void Rearrange(double[] Input, Complex[] Output, uint N)

{

// Data entry position

uint Target = 0;

Complex[] inp = new Complex[Input.Length];

for (uint i = 0; i < Input.Length; i++) inp[i]=new Complex(Input[i], 0.0);

// Process all positions of input signal

for (uint Position = 0; Position < N; ++Position)

{

// Set data entry

Output[Target] = inp[Position];

// Bit mask

uint Mask = N;

// While bit is set

while ((Target & (Mask >>= 1)) != 0)

// Drop bit

Target &= ~Mask;

// The current bit is 0 - set it

Target |= Mask;

}

}

private void Perform(Complex[] Data, uint N, bool Inverse = false)

{

double pi = (Inverse ? 3.14159265358979323846 : -3.14159265358979323846);

// Iteration through dyads, quadruples, octads and so on...

for (uint Step = 1; Step < N; Step <<= 1)

{

// Jump to the next entry of the same transform factor

uint Jump = Step << 1;

// Angle increment

double delta = pi / (double)Step;

// Auxiliary sin(delta / 2)

double Sine = Math.Sin(delta \* 0.5);

// Multiplier for trigonometric recurrence

Complex Multiplier = new Complex(-2.0 \* Sine \* Sine, Math.Sin(delta));

// Start value for transform factor, fi = 0

Complex Factor = new Complex(1.0, 0.0);

// Iteration through groups of different transform factor

for (uint Group = 0; Group < Step; ++Group)

{

// Iteration within group

for (uint Pair = Group; Pair < N; Pair += Jump)

{

// Match position

uint Match = Pair + Step;

// Second term of two-point transform

Complex Product = new Complex((Factor \* Data[Match]).Real, (Factor \* Data[Match]).Imaginary);

// Transform for fi + pi

Data[Match] = Data[Pair] - Product;

// Transform for fi

Data[Pair] += Product;

}

// Successive transform factor via trigonometric recurrence

Factor = Multiplier \* Factor + Factor;

}

}

}

private void Scale(Complex[] Data, uint N)

{

double Factor = 1.0 / (double)N;

// Scale all data entries

for (uint Position = 0; Position < N; ++Position) Data[Position]\*=Factor;

}

public bool FFTShift(Complex[] Input, Complex[] Output, uint N) {...}

...

}

public struct ConnectionSettings {...}

public class **DataSource**

{

// General receive package format

// [START | PACKID | CMDID | DATA0 | ... | DATA16 | CRC | END]

// General transmit package format

// [START | CMDID | DATA0 | DATA1 | DATA2 | CRC | END]

// General channel selector format [ CHANNEL | HPF | LPF ]

#region Constants for pack/unpack data

...

#endregion // Constants for pack/unpack datas

public DataSource() {...}

public void AddCommand(byte CMDID, byte PARAM0, byte PARAM1, byte PARAM2)

{

cmdFIFO.Enqueue(new cmd(CMDID, PARAM0, PARAM1, PARAM2));

}

public void SetConnectionSettings(ConnectionSettings sett) {...}

public bool OpenConnection() {...}

public bool CloseConnection() {...}

private void ExternPortSource\_DataReceived(object sender, SerialDataReceivedEventArgs e)

{

while (ExternPortSource.BytesToRead > 0)

{

for (int i = 1; i < pk\_Length; i++) Packet[i - 1] = Packet[i];

Packet[pk\_Length - 1] = (byte)ExternPortSource.ReadByte();

if (Packet[pk\_Length - 1] == ENDBYTE)

if (Packet[0] == STARTBYTE) TranslatePacket();

}

}

private void TranslatePacket() {...}

private void ToBinaryFile(string file) {...}

private bool SendPacket(cmd command) {...}

...

}

[Designer("System.Windows.Forms.Design.ScrollableControlDesigner, System.Design", typeof(IDesigner))]

public partial class **GraphContainer** : UserControl {...}

public partial class **GraphPanel** : UserControl

{

public GraphPanel() {...}

#region Properties

[Category("Properties")] //take this out and will have problems with serialization

[DefaultValue(typeof(string), "Graph 1")]

[Browsable(true)]

[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]

public string Title

{

get { return GraphTitle.Text; }

set

{

GraphTitle.Text = value;

this.Invalidate(true);

}

}

[...]

public Color GraphBkColor {...}

[...]

public GridTypes Grid {...}

public bool Hold {...}

#endregion // Properties

#region Draw methods

public void Plot(double[] X, double[] Y, LineProperties lineprop)

{

if (X.Length != Y.Length)

{

MessageBox.Show("X must have same length as Y !", "Plot Error !");

return;

}

if (!hold)

if (gobjects.Count != 0) gobjects.Clear();

gobjects.Add(new GraphObject(X, Y));

gobjects[gobjects.Count - 1].line.properties = lineprop;

this.Refresh();

}

public void Stem(double[] Y, LineProperties lineprop, bool StartFromZeroOffSet = true, bool RemoveZeroPoints = true)

{

double[] X = new double[Y.Length];

for (int i = 0; i < X.Length; i++) X[i] = i;

if (!hold)

if (gobjects.Count != 0) gobjects.Clear();

gobjects.Add(new GraphObject(X, Y, GraphTypes.STEM));

gobjects[gobjects.Count - 1].line.properties = lineprop;

gobjects[gobjects.Count - 1].StartFromZeroOffSet = StartFromZeroOffSet;

gobjects[gobjects.Count - 1].RemoveZeroPoints = RemoveZeroPoints;

this.Refresh();

}

public double[] Histogram(double[] Edges, double[] Y, LineProperties lineprop)

{

double[] counted = new double[Edges.Length-1];

for (int i = 0; i < Edges.Length - 1; i++) counted[i] = 0.0;

for (int i = 0; i < Edges.Length - 1; i++)

for (int j = 0; j < Y.Length; j++)

if (Y[j] >= Edges[i] && Y[j] < Edges[i+1]) counted[i]=counted[i]+1.0;

Bar(Edges, counted, lineprop);

return counted;

}

public void Bar(double[] Edges, double[] Y, LineProperties lineprop) {...}

private void SelectWndSize(out double minX, out double maxX, out double minY, out double maxY) {...}

private void DrawGraph() {...}

#endregion // Draw methods

...

} // End GraphPanel Class

#region Graphical objects

struct DoublePoint

public struct LineProperties

enum GraphTypes { PLOT, STEM, BAR };

public enum GridTypes { None, XGrid, YGrid, BothGrids};

class GraphLine

class GraphObject

#endregion Graphical objects

...

}