

Clase și Obiecte radio

Clasele sînt o parte majoră a programării orientată-pe-obiecte și “inima” limbajului C++, implementînd diverse tipuri de date și operațiile destinate manipulării acestora, integral sau prin moștenire de la alte *clase* mai generale.

Obiectul este o bucată de cod care efectuează o sarcină specifică de programare, prezentînd utilizatorului său (programatorului care îl folosește) numai atît cît este necesar pentru o utilizare simplă. Toate mecanismele interne pe care utilizatorul nu este nevoie să le cunoască îi sînt ascunse. *Obiectele* reprezentînd același concept, fiind de același tip, pot fi grupate în *clase* și devin la momentul utilizării instanțe ale respectivei *clase*. *Clasa* conține atît structurile de date necesare descrierii obiectelor (date membre) cît și setul de metode (funcții membre) care pot fi aplicate *obiectelor*.

Dezvoltarea aplicațiilor vizînd echipamente radio definite prin program și mai ales cele virtuale este mult ușurată de abordarea orientată pe *obiecte*, întrucît permite simplificarea scrierii programelor, întreținerea ușoară și refolosirea modulelor de program. Funcționalitatea acestor echipamente pune ușor în evidență blocurile din structura lor care se pot constitui în *obiecte*.

De exemplu, unul dintre modulele funcționale aproape nelipsit din structura unui receptor virtual este cel destinat extragerii componentelor modulate în cuadratură ale unui semnal radio de interes (eng. DDC – **D**igital **D**own **C**onverter). În plus, ținînd seama că în receptoarele cu intrare de radiofrecvență de bandă largă se folosesc chiar mai multe asemenea module lucrînd în paralel pentru recuperarea informației din semnalele de bandă îngustă existente simultan în banda de intrare a receptorului, se justifică crearea unei clase, arbitrar denumită **DDC**, care să acopere în cît mai mare măsură formele particulare ale acestui modul avînd ca punct de plecare schema bloc din figura 1.

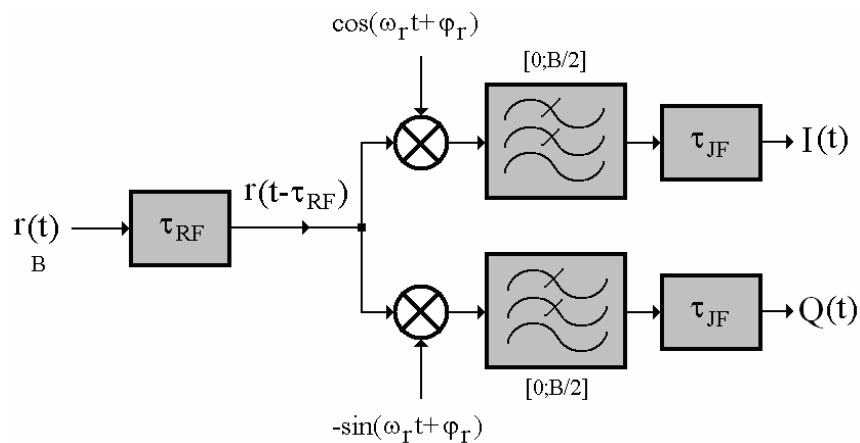


Figura 1

Clasa **DDC** poate avea în vedere următoarele procese generice:

- Generarea unor oscilații locale în cuadratură, cu frecvența f_r normată la frecvența de eșantionare și faza inițială φ_r exprimată în radiani precizate;
- Întîrzierea semnalului de intrare $r(t)$ cu o durată τ_{RF} stabilită ca multiplu a perioade de eșantionare;

- Mixarea semnalului de intrare întârziat cu oscilațiile locale în cuadratură;
- Extragerea componentelor modulatorie în cuadratură ale semnalului de intrare prin filtrarea trece-jos a produselor de mixare (filtrare de canal), frecvența de tăiere (lărgimea de bandă) $B/2$ a filtrelor fiind precizată și normată la frecvența de eșantionare;
- Întârzierea suplimentară (față de cea introdusă de filtrarea post-mixare) a componentelor din banda de bază cu o durată τ_{JF} stabilită ca multiplu a perioadei de eșantionare.

Listele 1 și 2 de mai jos prezintă, în formalismul mediului integrat Borland Builder ([1]) pentru dezvoltarea aplicațiilor C++, fișierul .h conținând definițiile datelor membre, funcțiilor membre, instanțelor claselor membre (pentru filtrarea și întârzierea post-mixare), și respectiv fișierul .cpp descriind relațiile operaționale între acestea.

```
#ifndef DDCH
#define DDCH

#include "FIR.h"
#include "Delay.h"
/*
    Clasă derivată, destinată unui model generic pentru partea de prelucrări la
    care este supus un semnal de bandă îngustă într-un receptor.
*/
class DDC
{
public:
    DDC(double,
        double,
        double,
        unsigned int,
        unsigned int,
        double,
        unsigned int,
        unsigned int);    //Constructorul clasei.
    ~DDC(void);           //Destructorul clasei.
    void Update(double,double*);

private:
    double ReceiverTuningFrequency;    //Frecvența oscilației locale normată
                                        // la frecvența de eșantionare.
    double LocalOscillatorInitialPhase; //Faza inițială a oscilației locale exprimată în radiani.
    double LocalOscillatorPhase;        //Faza oscilației locale.
    double ChannelFiltersBandwidth;     //Lărgimea de bandă (~frecvența limită superioară)
                                        // a filtrelor trece-jos post-mixare, normată la
                                        // frecvența de eșantionare. Este egală cu
                                        // jumătate din banda canalului de radiofrecvență.
    unsigned int ChannelFiltersTapsNumber; //Numărul de prize ale unui filtru post-mixare.
    unsigned int ChannelFiltersWindowType; //Constantă numerică definită în clasa Window.
    double ChannelFiltersWindowParameter; //Parametru al ferestrei aplicate coeficienților
                                        // filtrelor post-mixare (dacă este cazul).
    unsigned int RF_Delay;               //Numărul de eșantioane cu care se întârzie eșantioanele
                                        // de semnal aplicate la intrarea DDC.
```

```

unsigned int JF_Delay;           //Numărul de eşantioane cu care se întârzie eşantioanele
                                   // de la ieşirile filtrelor post-mixare.
FIR* ChannelFilter_I;           //Filtrul post-mixare pe calea I.
FIR* ChannelFilter_Q;           //Filtrul post-mixare pe calea Q.
Delay* RF_DelayLine;           //Linia de întârziere pe calea de RF.
Delay* JF_DelayLine_I;          //Linia de întârziere JF pe calea I.
Delay* JF_DelayLine_Q;          //Linia de întârziere JF pe calea Q.

protected:
};
#endif

```

Lista 1 – DDC.h

```

#include "DDC.h"
#include <math.h>
//-----
DDC::DDC(double _ReceiverTuningFrequency=0.25,
        double _LocalOscillatorInitialPhase=0.0,
        double _ChannelFiltersBandwidth=0.125,
        unsigned int _ChannelFiltersTapsNumber=33,
        unsigned int _ChannelFiltersWindowType=RECTANGLE,
        double _ChannelFiltersWindowParameter=0.0,
        unsigned int _RF_Delay=5,
        unsigned int _JF_Delay=5):
    ReceiverTuningFrequency(_ReceiverTuningFrequency),
    LocalOscillatorInitialPhase(_LocalOscillatorInitialPhase),
    ChannelFiltersBandwidth(_ChannelFiltersBandwidth),
    ChannelFiltersTapsNumber(_ChannelFiltersTapsNumber),
    ChannelFiltersWindowType(_ChannelFiltersWindowType),
    ChannelFiltersWindowParameter(_ChannelFiltersWindowParameter),
    RF_Delay(_RF_Delay),
    JF_Delay(_JF_Delay)
{
    //Se crează instanțe ale clasei FIR pentru filtrele post-mixare.
    ChannelFilter_I=new FIR(LPF,ChannelFiltersTapsNumber,2*ChannelFiltersBandwidth,0.0,
                           ChannelFiltersWindowType,ChannelFiltersWindowParameter);
    ChannelFilter_Q=new FIR(LPF,ChannelFiltersTapsNumber,2*ChannelFiltersBandwidth,0.0,
                           ChannelFiltersWindowType,ChannelFiltersWindowParameter);
    //Se crează instanțe ale clasei Delay pentru toate întârzierile.
    RF_DelayLine=new Delay(RF_Delay+1);
    JF_DelayLine_I=new Delay(JF_Delay+1);
    JF_DelayLine_Q=new Delay(JF_Delay+1);
    //Se inițializează unele variabile.
    LocalOscillatorPhase=LocalOscillatorInitialPhase;
}
//-----
DDC::~DDC(void)
{
    //Destructorul eliberează memoria ocupată de clasă.
    delete ChannelFilter_I;
    delete ChannelFilter_Q;
    delete RF_DelayLine;
    delete JF_DelayLine_I;
    delete JF_DelayLine_Q;
}

```

```

}
//-----
void DDC::Update(double _CurrentInputSample, double* _DDC_OutputSamples)
{
    //Se introduce în DDC eșantionul curent al semnalului de intrare _CurrentInputSample,
    // și se obțin eșantioanele de la cele două ieșiri din banda de bază, _OutputSample_I
    // și _OutputSample_Q.
    double DelayedInputSample; //Eșantionul curent de la ieșirea liniei de întârziere de RF.
    double Temp_I, Temp_Q; //Produse de mixare filtrate.

    //Se întârzie semnalul de intrare.
    if(RF_Delay>0) DelayedInputSample=RF_DelayLine->Update(_CurrentInputSample);
    else DelayedInputSample=_CurrentInputSample;
    LocalOscillatorPhase+=2*M_PI*ReceiverTuningFrequency;
    //Se reduce faza curentă a oscilației locale la [0;2*M_PI).
    if(LocalOscillatorPhase>2*M_PI) LocalOscillatorPhase-=2*M_PI;
    else if(LocalOscillatorPhase<-2*M_PI) LocalOscillatorPhase+=2*M_PI;
    //Se mixează semnalul întârziat și se filtrează produsele.
    Temp_I=ChannelFilter_I->Update(DelayedInputSample*cos(LocalOscillatorPhase));
    Temp_Q=ChannelFilter_Q->Update(-DelayedInputSample*sin(LocalOscillatorPhase));
    //Se întârzie componentele din banda de bază.
    if(JF_Delay>0)
    {
        _DDC_OutputSamples[0]=JF_DelayLine_I->Update(Temp_I);
        _DDC_OutputSamples[1]=JF_DelayLine_Q->Update(Temp_Q);
    }
    else
    {
        _DDC_OutputSamples[0]=Temp_I;
        _DDC_OutputSamples[1]=Temp_Q;
    }
    return;
}

```

Lista 2 – DDC.cpp

Lista 3 arată cum se pot crea *obiectele* din clasa **DDC** într-o aplicație radio în care trebuie recepționate simultan două emisiuni de bandă îngustă ocupînd fiecare cîte o bandă de 8 kHz în jurul frecvențelor purtătoare de 10 kHz și respectiv 25 kHz. Semnalele radio sînt aduse prin mixări analogice multiple în banda 0 Hz – 32 kHz și sînt eșantionate cu 64 ksps. Filtrele trece-jos post-mixare sînt de tip FIR cu 555 de prize și coeficienții ponderați cu fereastra Blackman. Fluxul de eșantioane preluat de la convertorul analog-numeric și de la ieșirile fitrelor post-mixare sînt întârziate cu cîte o perioadă de eșantionare.

```

...
DDC* DDC_1;
DDC* DDC_2;
double Rx1_IQSamples[2]; //Tabelul perechii de eșantioane I și Q produse la ieșirea DDC_1.
double Rx2_IQSamples[2]; //Tabelul perechii de eșantioane I și Q produse la ieșirea DDC_2.
...
DDC_1=new DDC(10000.0/64000.0,0.0,4000.0/64000.0,555,BLACKMAN,0.0,1,1);
DDC_2=new DDC(25000.0/64000.0,0.0,4000.0/64000.0,555,BLACKMAN,0.0,1,1);
...
DDC_1->Update(InputSamples[i], Rx1_IQSamples);
DDC_2->Update(InputSamples[i], Rx2_IQSamples);

```

```
...  
//Se eliberează memoria ocupată de instanțele clasei DDC.  
delete DDC_1;  
delete DDC_2;
```

Lista 3

BIBLIOGRAFIE

[1] - <http://www.embarcadero.com/>