Laborator 4 TPI 2022-2023

Aplicații Blackfin BF533 pentru compresie cu pierderi și fără pierderi a informației

1. Introducere. Obiectul lucrării

Tehnicile de compresie a informației urmăresc reducerea cantității de informație pentru stocare sau transmitere. Conceptual, algoritmii de compresie se clasifică în:

- algoritmi de compresie fără pierderi
- algoritmi de compresie cu pierderi

Algoritmii de compresie fără pierderi (lossless compression) permit refacerea completă a informației la recepție (sau la decompresie în cazul sistemelor de stocare) și se bazează pe tehnici diverse cum ar fi: codare simbol cu simbol pe baza conceptelor teoretice inspirate din Teorema I a lui Shannon, tehnici de compresie aritmetică, înlocuirea unui șir repetitiv de simboluri informaționale identice cu un un cuvânt de cod mai scurt, tehnici de codare pe șiruri pe baza unor dicționar construit dinamic atât la codare cât și la decodare .

Spre deosebire de tehnicile mai sus menționate, algoritmii de compresie cu pierderi (lossy compression) sunt construiți în jurul ideii de înlăturare deliberată a unei anumite cantități de informație, în condițiile unor modificări minimale aduse conținutului informațional al imaginii/ semnalului de comprimat. Spre deosebire față de algoritmii de compresie fără pierderi, tehnicile de tip lossy au un domeniu de aplicabilitate mai restrâns, folosirea acestora fiind posibilă doar în situațiile în care pierderea de informație este acceptabilă.

Obiectul lucrării de laborator constă în studiul şi implementarea unor algoritmi de compresie cu și fără pierderi pe procesoare digitale din familia Blackfin, cu aplicații directe în compresia datelor și compresia imaginilor. Tehnicile de compresie ce vor fi implementate sunt algoritmul de compresie LZW și codarea cu pierderi prin transformate DCT.

2. Implementarea algoritmului de compresie LZW pe procesorul BF533

Proiectul ***LZW_Date*** plasat pe pagina de web a cursului ilustrează o posibila implementare pentru algoritmului de compresie LZW. Funcțiile folosite în proiect sunt incluse în anexa 1 și sunt explicate mai jos. Marea majoritate a funcțiilor C operează pe o structură de date esențială în funcționarea algoritmului LZW – dicționarul de codare . Definirea unei structuri de date se face standard în C folosind cuvântul cheie **struct** iar în cadrul proiectului definirea constă în următoarea secvență de declarații în limbajul de programare C:

```
struct DICTIONAR{
        int cod;
         int subsir;
}dictionar[MARIME_DICTIONAR] ;
```

Declarațiile anterioare definesc un tip global de date (vizibil în orice funcție din interiorul proiectului) – **dictionar** – cu 1024 de poziții, fiecare poziție având la rândul ei 2 variabile: **cod** (etichetă alocată dinamic) unui **subșir** format din caractere individuale ce apare în mod explicit în secvența ce ce urmează a fi codată

Constantele predefinite în program sunt :

MARIME_CARACTERE - definește tipul datelor ce vor fi procesate (8- pentru coduri ASCII)
NUMAR_CARACTERE – numărul de caractere distincte din secvența ce va fi codată (setată initial in proiect la valoarea 4)

NUMAR_ MAX_ CODURI –evită inserarea de noi siruri în dicționar la umplerea acestuia. **MARIME_DICTIONAR** - constantă predefinită în proiect (1024).

Funcțiile folosite în program sunt :

void initializare_dictionar(void)	-funcția este apelată la lansarea în execuție a aplicației și are rolul de a inițializa elementele **cod** ale dicționarului cu o valoare predefinită. Funcția NU initializează tabelul de codare cu simboluri individuale.
void Compresie_LZW(void);	- funcția este apelată în corpul funcției **main** și definește succesiunea de declarații și apeluri de funcții necesare pentru realizarea compresiei de tip LZW.
int citire_cod(int, int);	- funcția are rolul de citire a dicționarului și returnează poziția din dicționar asociată unui șir (dacă acesta există deja) sau o valoare prestabilită (dacă șirul curent nu există în dicționar).
int citire_dictionar(int, int) ;	- funcție de tip wrapper ce folosește rezultatul funcției precedente
void inserare_tabel(int , int);	- funcție cu rol de inserare în tabel a unui șubșir nou.
int output(int, int);	- returnează codul de transmis pentru o anumită poziție în cadrul șirului de codat

3 Exerciţii

1. Folosind explicațiile de la curs:
    a) Codați manual șirul **ababcbababaadaa**1; 1 este un caracter folosit pentru simularea caracterului EOF și nu face parte din mesaj
    b) Modificați corpul funcției Compresie_LZW pentru inițializarea dicționarului cu simbolurile distincte din mesajul anterior. (Indicație: un apel de tipul ***inserare_tabel (poz,0x00000061)***; inserează în dicționar pe poziția **poz** codul ASCII al literei A (0x00000061- în notație hexazecimală)
    c) Lansați în execuție programul modificat și verificați modul de codare de la punctul a).
    d) Alegeți un alt șir format din 5 caractere distincte de lungime 20 de caractere. Ce modificări trebuie aduse programului pentru a permite codarea acestui șir?
    e) Explicați modul de funcționare al programului prin rulare în mod Debug. După construirea proiectului accesați **View-> Debug Windows->Locals** și alegeți variabila **_dicționar** din memoria Blackfin. Variabila poate fi vizualizată la fiecare rulare a programului prin comanda Halt. Deplasarea în cod se poate face folosind comanda **Run to cursor (CTRL+F10)**.
2. Modificați proiectul astfel încât acesta să permită codarea primilor 1000 de pixeli ai imaginii **Lena.bmp** plasată în directorul curent. Vizualizați rezultatele obținute și estimați factorul de compresie obținut în urma compresiei. Cum se modifică acesta dacă lungimea dicționarului este limitată la 300 poziții?
3. Conectați placa EZ_LITE și comprimați întreaga imagine. Care este factorul de compresie obținut?
4. Creați un nou proiect și propuneți o implementare pentru codarea de tip RLC.

4. Implementarea unui algoritm de compresie de tip DCT

Proiectul DSP ***Compresie_DCT*** plasat pe pagina de web propune o posibilã implementare pentru un algoritm de compresie de tip DCT conform explicațiilor din curs Funcțiile asociate programului au prototipurile listate în fișierul DCT.h.

5 Exerciţii

1. Pe baza explicațiilor teoretice de la curs, analizați modul de implementare al funcțiilor din anexa 2 și explicați rolul fiecărei funcții inserând comentarii în cod.
2. Lansați în execuție aplicația pe placa EZ_KIT_LITE, analizați rezultatul obținut și notați observațiile și explicațiile proprii.
3. Modificați programul asfel încât algoritmul să implementeze o tehnică de compresie cu pierderi minime.
4. Modificați aplicația astfel încât factorul de compresie obținut să fie mai mare. Ce modificări sunt observabile asupra calității imaginii procesate?
5. Ce modificări trebuie aduse programului astfel încât acesta să implementeze standardul de compresie JPEG? Schițați în pseudocod aceste modificări.

Anexa 1. Funcții folosite pentru implementarea codării LZW a unui șir de date

```
#include <stdio.h>
#include <LZW_Date.h>
#include <stdlib.h>
#include <string.h>


static char sir_intrare[16] = "ababcbababaadaa1";


struct DICTIONAR{
       int cod;
    int sir;
}dictionar[MARIME_DICTIONAR];

int nr_caract;

int citire_dictionar(int sir, int cur){
    int b = citire_cod(sir, cur);

    if(dictionar[b].cod == INEXISTENT || dictionar[b].sir != sir)
        return INEXISTENT;
    return dictionar[b].sir;
}

int citire_cod(int sir, int lim){
    int j = 0;
    do{
        if( dictionar[j].sir == sir)
            return j;
        j = (j+1);
        }while(j!=lim);
     j=INEXISTENT;
    return j;
}

void Compresie_LZW(){
       int i, pozitieCurenta, c, prefix, temp, e ;
    int contor,cod_prefix, depasire;

    nr_caract=0;
    initializare_dictionar();
    pozitieCurenta = NUMAR_CARACTERE;
    contor=0;


       c=sir_intrare[contor++];

        if(c!=0x00000031){
        prefix = c;

        c=sir_intrare[contor++];

       while(c!=0x00000031){
              temp = (prefix<<MARIME_CARACTERE)+c;
              depasire=prefix & MASCA;
              if (depasire==0){
              e= citire_dictionar(temp, pozitieCurenta);
              if(e==INEXISTENT){    //nu exista in dictionar
                     cod_prefix=output(prefix, pozitieCurenta);
```

```
                        fprintf(stdout,"%d ", cod_prefix);
                        if(pozitieCurenta<NUMAR_MAX_CODURI)
                        inserare_tabel( pozitieCurenta++,temp);
                        prefix = c;
                               }
                else
                        prefix = e;
                }
                else{
                        e= citire_dictionar(temp, pozitieCurenta);
                        cod_prefix=output(prefix, pozitieCurenta);
                        fprintf(stdout,"%d ", cod_prefix);
                        prefix=c;
                }

          c=sir_intrare[contor++];

         }
        fprintf(stdout,"%d ", output(prefix, pozitieCurenta));

    }
    fprintf(stdout,"\n%d ", nr_caract);
}

void initializare_dictionar(){
    int i;
    for(i=0; i<MARIME_DICTIONAR; i++)
        dictionar[i].cod = INEXISTENT;
}

int output(int prefix, int poz){
       int j=0,d,c;
       do{
        if( dictionar[j].sir == prefix)
            break;
        j = (j+1);
        }while(j!=poz);
        nr_caract++;
     return j;
}


void inserare_tabel(int poz,int cod ){

 dictionar[poz].sir = cod;
 dictionar[poz].cod = poz;

}

void main(){
        Compresie_LZW();
}
```

Anexa 2. Funcții folosite pentru implementarea compresiei DCT a unei imagini

```
#include <dct.h>
#include <math.h>

unsigned char imagineIn[16384];
unsigned char imagineOut[16384];
int transDCT[16384] ;
int transDCTQ[16384] ;
float imTemp[16384];


void initializare_dct()
{
 int i, j;
 double s;

 for (i=0;i<64;i++)
       Tabel_Quant50[i]=Tabel_Quant50[i];
 for (i=0; i<MARIME_BLOC; i++)
       {
       s = (i==0) ? sqrt(1.0f/MARIME_BLOC) : sqrt(2.0/MARIME_BLOC) ;
         for (j=0; j<MARIME_BLOC; j++)
              coeficienti[i][j] = s * cos((PI/(float)MARIME_BLOC)*i*(j+0.5));
       }
}


void DCT_SEP(int * blocDest,unsigned char * blocSursa, int lat)
{
 int i, j, k;
 double s;
 double tmp[MARIME_BLOC * MARIME_BLOC];

 for(i = 0; i < MARIME_BLOC; i++)
       {
       for(j = 0; j < MARIME_BLOC; j++)
              {
              s = 0.0;
              for(k=0; k<MARIME_BLOC; k++)
                     s += coeficienti[j][k] * (blocSursa[lat * i + k]-128);
                 tmp[MARIME_BLOC * i + j] = s;
              }
       }

       for(j = 0; j < MARIME_BLOC; j++)
              {
              for(i = 0; i < MARIME_BLOC; i++)
              {
              s = 0.0;
              for(k=0; k<MARIME_BLOC; k++)
                     s += coeficienti[i][k] * tmp[MARIME_BLOC * k + j];
              blocDest[lat * i + j] = (int)s;
              }
       }
}

void DCT_SEP_INV(float * blocDest, int *blocSursa, int lat)
{
 int i, j, k, v;
 double produs_partial;
```

```
 double tmp[MARIME_BLOC * MARIME_BLOC];

 for (i=0; i<MARIME_BLOC; i++)
       {
       for (j=0; j<MARIME_BLOC; j++)
              {
              produs_partial = 0.0;
              for (k=0; k<MARIME_BLOC; k++)
                     produs_partial+= coeficienti[k][j]*blocSursa[lat*i+k];
          tmp[MARIME_BLOC*i+j] = produs_partial;
              }
       }
        for (j=0; j<MARIME_BLOC; j++)
              {
               for (i=0; i<MARIME_BLOC; i++)
                     {
                     produs_partial = 0.0;
                     for (k=0; k<MARIME_BLOC; k++)
                            produs_partial+= coeficienti[k][i]*tmp[MARIME_BLOC*k+j];

                     v = (int) floor(produs_partial+0.5);
                     v=v;
                     blocDest[lat*i+j] =v;
                     }
              }
}


void DCT_DIR(int* result,  unsigned char * intrare, int latime, int inaltime, int block_size)
{
 int i, j;
 int image_latime;
 int image_inaltime;

 image_latime = latime;
 image_inaltime = inaltime;

 for(j = 0; j < image_inaltime/block_size; j++)
 for(i = 0; i < image_latime/block_size; i++)
       DCT_SEP(&result[(j*image_latime + i) * block_size] ,&intrare[(j*image_latime + i) *
block_size], image_latime);
}

void DCT_INV(float * result, int* intrare, int latime, int inaltime, int block_size)
{
 int i, j;
 int image_latime;
 int image_inaltime;

 image_latime = latime;
 image_inaltime = inaltime;

 for(j = 0; j < inaltime/block_size; j++)
 for(i = 0; i < latime/block_size; i++)
   DCT_SEP_INV(&result[(j*image_latime + i) * block_size] ,&intrare[(j*image_latime + i) *
block_size], image_latime);
}

void Cuantizare (int * In, int * Out, int latime, int inaltime, int marime)
{
```

```
	int i,j;
	int u,v;
	for(i = 0; i < latime; i=i+marime)
	for(j = 0; j < inaltime; j=j+marime)
		for (u=0;u<marime;u++)
		for (v=0;v<marime;v++)
			Out[i+u+(v+j)*latime]=In[i+u+(v+j)*latime]/Tabel_Quant50[u+v*marime];

}

void DeCuantizare (int * In, int * Out, int latime, int inaltime, int marime)
{
	int i,j;
	int u,v;
	for(i = 0; i < latime; i=i+marime)
	for(j = 0; j < inaltime; j=j+marime)
		for (u=0;u<marime;u++)
		for (v=0;v<marime;v++)
			Out[i+u+(v+j)*latime]=In[i+u+(v+j)*latime]*Tabel_Quant50[u+v*marime];
}

void Renormalizare (float * iI, unsigned char * iO, int latime, int inaltime)
{
	int i,j,u;
	for(i = 0; i < latime; i=i+1)
	for(j = 0; j < inaltime; j=j+1){
		iI[i+j*latime]=iI[i+j*latime]+128;
		iI[i+j*latime]=LIMITARE(iI[i+j*latime]);
		iO[i+j*latime]=iI[i+j*latime];
	}
}

void main ()
{
 initializare_dct();
 DCT_DIR(transDCT, imagineIn, marime_x , marime_y, MARIME_BLOC);
 Cuantizare (transDCT, transDCTQ, marime_x , marime_y, MARIME_BLOC);
 DeCuantizare (transDCTQ, transDCT, marime_x , marime_y, MARIME_BLOC);
 DCT_INV(imTemp, transDCT, marime_x , marime_y, MARIME_BLOC);
 Renormalizare (imTemp, imagineOut,  marime_x , marime_y);

}
```