Lab4 IPT 2022-2023

Blackfin B533-based implementations for lossy and lossless compression

1. Introduction

Information compression techniques are aiming to reduce the amount of information for storage or transmission purposes. Conceptually, the compression algorithms are classified as:

- lossless compression algorithms
- lossy compression algorithms

Lossless compression algorithms are allowing complete recovery of information at the reception (or decompression for storage systems) and rely on various techniques such as symbol-by-symbol coding, arithmetic compression, run length coding techniques or dynamic dictionary techniques built both by the encoding and decoding piece of equipment. Unlike the above described techniques, lossy compression algorithms are built around the idea of deliberate removal of a certain amount of information, with minimal changes to the information content of the image or the signal to be compressed. In contrast to lossless compression algorithms, such lossy techniques can be applied only in situations where loss of information is acceptable.

The purpose of the laboratory is to study and implement algorithms and lossless compression on Blackfin digital processors, with direct applications in data and image compression. Compression techniques to be implemented are the LZW compression algorithm and the DCT based lossy image compression algorithm

2. LZW on the BF533 DSP

The LZW_Date project that can be downloaded from the course's website illustrates the implementation of the LZW compression algorithm. The functions used in the project are included in Appendix 1 and are explained below. Most of the presented C functions are operating on a data structure essential to LZW – a dictionary comprising symbols and labels. Such a data structure is done using the standard C keyword **struct**. The project employs the following data structure :

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

The previous statements define a global type of data (visible for all the code) - **dictionar** – as a table with 1024 positions. For each index in the table two variables are stored: - dynamically asigned labels

- individual characters and substring formed using individual characters uniquely associated to a given label

Some predefined constants are also used in the code:

MARIME_CARACTERE - size in bits of each individual character (8- for ASCII codes or for graylevel images)

NUMAR_CARACTERE – the number of distinct characters within the data that will have to be compressed (with an implicit value set to 4)

NUMAR_MAX_CODURI – used for forbidding insertion of new substrings into the dictionary whenever the last one is filled.

MARIME_DICTIONAR - the size of the dictionary (1024 by default).

The functions used in the application are :

void initializare_dictionar(void)	- function is called at application startup and for initialising the dictionary with a predefined value
void Compresie_LZW(void);	- used for performing the actual encoding of an input string
int citire_cod(int, int);	- reads the dictionary and returns the current position of a substring in the dictionary or a predefined value if the later has not been yet included
int citire_dictionar(int, int) ;	- wrapper function that uses the result of the previous function
void inserare_tabel(int , int);	- inserts new substrings into the dictionary
int output(int, int);	- returns the label associated to a substring in the dictionary

3 Exercises

1. Using the explanation from the course:
    a) Encode manually the string **ababcbababaadaa**1; 1 is used for simulating EOF (End of File) functionality and it is not an actual symbol.
    b) Modify the *Compresie_LZW* function in order to allow initialization of the dictionary with all the distinct characters included in the message. (Indication: a call of the type ***inserare_tabel (poz,0x00000061)*** inserts at index **poz** the ASCII code of letter a (0x00000061- in hexadecimal notation)
    c) Run the program and verify the correct encoding of the message indicated at a).
    d) Choose another message having a length of 20 characters, composed of 5 distinct characters. What modifications should be brought to the program in this case?
    e) Explain how actually encoding is done. For doing this the **Debug** mode must be selected and the **dictionar** variable's content can be visualized from **View-> Debug Windows->Locals**. At each step modifications brought to this memory locations can be visualized by moving through the code using the **Run to cursor** command **(CTRL+F10)**.
2. Modify the project in order to allow compression of the first 1000 pixels of the Lena.bmp image. Estimate the compression ratio and indicate how this measure is affected in case of a limit of the dictionary size set to 300.
3. Connect the EZ_LITE board and compress the whole image. What's the value of the compression ratio?
4. Create a new project and write all the neccesary code for implementing the RLC compression algorithm.

4. Lossy compression using the DCT transform

The project ***Compresie_DCT*** placed on the course's website proposes an implementation of a DCT lossy compression algorithm following the explanations from the relevant lecture. The prototypes of the functions are included in the DCT.h file.

5 Exercises

1. Explain how the DCT/the inverse DCT transforms are implemented by inserting comments into the code. Explain how compression is achieved.
2. Connect the EZ_KIT_LITE board and run the application using the Lena.bmp image. Which are the effects of the compression algorithm on the quality of the processed image?
3. Modify the algorithm in order in order to obtain a better quality of the processed image.
4. Modify the algorithm in order in order to obtain a better compression ratio.
5. Which are the modifications needed in the code for allowing JPEG compression to take place?

Appendix 1. Functions used for the LZW algorithm

```
#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();
}
```

Appendix 2. Functions used for DCT based lossy compression

```
#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);

}
```