Elektrotehnički fakultet
Univerzitet u Banjoj Luci

IZVJEŠTAJ PROJEKTNOG ZADATKA

iz predmeta

SISTEMI ZA DIGITALNA OBRADA SIGNALA

Student:
Savičić Anđela, 11131/18

Mentori:
prof. dr Mladen Knežić
prof. dr Mitar Simić
ma Vedran Jovanović
dipl. inž. Dimitrije Obradović

Februar 2026. godine

SADRŽAJ

1. Opis projektnog zadatka

U sklopu projektnog zadatka „ Kompresija slike korišćenjem diskretne kosinusne transformacije na ADSP-21489 razvojnoj platformi “ potrebno je realizovati sisitem za kompresiju *grayscale* slika na razvojnom okruženju ADSP-21489 korišćenjem direktne kosinusne transformacije (DCT).

Kompresija se bazira na JPEG standardu za kompresiju, ali radi jednostavnosti, realizovani su samo određeni koraci, pa tako modifikovana metoda realizuje sledeće slučajeve:

- RGB sliku upisati na razvojnu platformu i izvršiti njenu konverziju u *grayscale* sliku.
- Centrirati piksele oko nule, segmentisati sliku, te primijeniti DCT kvantizaciju. Matricu kvantizacije odabrati proizvoljno ali voditi računa da će ovaj izbor direktno uticati na stepen kompresije. Takođe, ista matrica kvantizacije neophodna je da se izvrši dekompresija slike.
- Primjeniti cik-cak transformaciju te izvršiti kodovanje DC i AC koeficijenata.
- Serijalizovati u proizvoljnom formatu dobijene rezultate
- Analizirati dobijene rezultate u Python ili MATLAB programskom jeziku korišćenjem odgovarajućih metrika. Izvršiti dekompresiju slike u Python ili MATLAB i prikazeti MSE u odnosu na originalnu sliku. Voditi računa o zauzeću memoriskih resursa.

Pored funkcijonalne isparavnosti, cilj je optimizacija svih koraka kroz koje sistem prolazi (brzina izvršavanja i zauzeće memorijskih resursa). Takođe prilikom optimizacije potrebno je obratiti posebnu pažnju na „ *bottle neck* “ tj. koji dio koda čitavog sisitema zahtijeva najviše procesorskih ciklusa. Kvalitet rekonstrukcije se provjerava na računaru tako što se vrši dekompresija u Python-u ili MATLAB-u programskom jeziku i račina MES u odnosu na originalnu sliku.

Upitsvo za pokretanje, detalji izrade, slike, kao i rezultati rada sistema se nalaze na repozitorijumu https://github.com/AndjelaS997/image-compression-ADSP-21489.

2. Izrada projektnog zadatka

Učitavanje slike na ADSP-21489 razvojnu platformu je realizovano u okviru *Cross Core Embedded Studio* razvojnog okruženja. Izabrani format slike koja se obrađuje je *.bmp* format bez kompresije (24-bitni ili 32-bitni). Potrebno je iz slike tog formata izvući informacije o boji piksela i tako učitanu sliku pretvoriti u *grayscale* sliku. Podaci o učitanoj slici se nalaze u zaglavlju slike i čuvaju se u strukturi *bmp_header_t* u b*mp_io.h* fajlu.

```
12 #pragma pack(push, 1)
13 typedef struct{
14     u16 type;
15     u32 size;
16     u16 reserved1;
17     u16 reserved2;
18     u32 offset;
19
20     u32 header_size;
21     s32 width;
22     s32 height;          // m
23     u16 planes;
24     u16 bits_per_pixel;
25     u32 compression;
26     u32 image_size;
27     s32 x_pixel_per_meter;
28     s32 y_pixel_per_meter;
29     u32 colors_used;
30     u32 colors_important;
31 } bmp_header_t;
32 #pragma pack(pop)
```

Slika 2.1 – *Prikaz strukture bmp_header_t*

U fajlu bmp_io.c implementirano je čitanje zaglavlja BMP fajla i učitavanje piksela u niz 32-bitnih vrijednosti formata 0xRRGGBB. Podržano je i „*bottom-up*“ i „*top-down*” raspored redova (na osnovu znaka visine u header-u), kao i padding na kraju svake vrste na 4 bajta. Za verifikaciju, implementiran je i upis sive slike u .bmp format nakon konverzije.

Konverzija slike RGB u *grayscale* se vrši u rgb_gray.c koristeći cijelobrojnu aproksimaciju:

$$Y = \frac{77 * R + 150 * G + 29 * B + 128}{256} \qquad (2.1)$$

računa se bez floating-point operacija i daje opseg 0-255.

```
10 void rgb_to_gray(u32* pixels, int width, int height)
11 {
12     //printf("Pocinje konvertovanje iz rgb u gray.\n");
13 
14     int n = width * height;
15     for(int i=0; i < n; i++)
16     {
17         int R = (pixels[i] >> 16)& 0xFF;
18         int G = (pixels[i] >> 8)& 0xFF;
19         int B = (pixels[i])& 0xFF;
20 
21         int Y = (77*R + 150*G + 29*B + 128) >> 8;  // 0..255
22         pixels[i] = ((u32)Y << 16) | ((u32)Y << 8) | (u32)Y;
23     }
```

Slika 2.2 – *Funkcija kojom se implementira konverzija RGB u grayscale*

Slika se obrađuje blokovski, pri čemu se slika dijeli na blokove dimenzija 8x8. Svaki blok se transformiše nezavisno, a to omogućava lokalnu obradu i kasniju kompresiju koeficijenata. Funkcija *extract_block_centered()* formira blok od 8x8 uz ponavljanje rubnih ivica kada blok prelazi granice, tj. za rubne blokove, indeksi se ograničavaju na validan opseg, na taj način je omogućena obrada slike čije dimenzije nisu dijeljive sa 8. Iz svakog piksela uzima se vrijednost sivog tona, a zatim se radi centriranje oduzimanjem 128 od izračunate gray vrijednosti:

$$Y_c = Y - 128 \tag{2.2}$$

ovo odgovara dijelu koda napisanom u funkciji *extract_block_centered()*:

```
for(int x=0; x <8; x++)
{
    int sx= restrict_(x0 + x, 0, width-1);

    /*uzmi Y iz grayscale*/
    u8 Y = (u8)(pixels[sy * width + sx] & 0xFF);

    /*centriraj oko 0*/
    out[y * 8 + x] = (s16)Y - 128;
}
```

Slika 2.3 – *Dio koda funkcije extract_block_centered*

Za svaki blok se zatim računa 2D DCT. DCT koeficijenti predstavljaju raspodijelu energije bloka po prostornim frekvencijama:

- Element F(0,0) je DC komponenta (prosječna vrijednost bloka)
- Ostali koeficijenti su AC komponente (promjene intenziteta- detalji)

DCT matrica C dimenzije N×N se inicijalizuje jednom u funkciji dct8x8_init() koristeći ortonormalnu definiciju:

$$C_{(u,x)} = \alpha(\mathrm{u})\cos(\frac{\pi}{N}(x+\frac{1}{2})u) \tag{2.3}$$

,za N=8 gdje je

$$\alpha(0) = \sqrt{\frac{1}{8}}, \text{ a } \alpha(\mathrm{u}) = \sqrt{\frac{2}{8}} \quad \text{za u} > 0 \tag{2.4}$$

U matričnom obliku koristi se relacija :

$$F = CfC^T \tag{2.5}$$

Gdje je C ortonormalna DCT matrica. U kodu se C unaprijed izračuna u *dct8x8_init()* i čuva u statičkom nizu C[8][8].

```
10 static float C[8][8];
11 static int inited = 0;
12
13 void dct8x8_init(void)
14 {
15     const float N = 8.0;
16     for(int u = 0; u < 8; u++)
17     {
18         float a = (u == 0) ? sqrtf(1.0/N) : sqrtf(2.0/N);
19         for(int x = 0; x < 8; x++)
20         {
21             C[u][x] = a * cosf((PI/N) * ((float)x + 0.5) * (float)u);
22         }
23     }
24     inited = 1;
25 }
```

Slika 2.4 – *Funkcija dct8x8_init()*

Sama transformacija *dct8x8()* se realizuje separabilno (u dvije faze) preko privremene matrice tmp:

prvo se vrši transformacija po jednoj dimenziji (sumiranje po x), a zatim po drugoj (sumiranje po y).

Ovakav pristup je efikasan i odgovara standardnoj implementaciji 2D DCT.

```
28 void dct8x8(const s16 in[64], float out[64])
29 {
30     if(!inited) dct8x8_init();
31
32     float tmp[8][8];  // tmp[u][y] = sum_x C[u][x] * f[y][x]
33
34     for(int u = 0; u < 8; u++){
35         for(int y = 0; y < 8; y++){
36             float s = 0.0;
37             for(int x = 0; x < 8; x++){
38                 float fx = (float)in[y*8 + x];
39                 s += C[u][x] * fx;
40             }
41             tmp[u][y] = s;
42         }
43     }
44
45     for(int u = 0; u < 8; u++){
46         for(int v = 0; v < 8; v++){
47             float s = 0.0;
48             for(int y = 0; y < 8; y++){
49                 s += tmp[u][y] * C[v][y];
50             }
51             out[u*8 + v] = s;
52         }
53     }
54 }
```

Slika 2.4 – *Funkcija dct8x8()*

Nakon transformacije, dobijeni koeficijneti se kvantizuju matricom Q, koja jače kvantizuje visoke frekvencije (koje su manje vizuelno bitne), a slabije niske (vizelno bitnije). Kvantizacija je :

$$F\hat{}(u,v) = round(\frac{F(u,v)}{Q(u,v)}) \tag{2.6}$$

što u kodu *quantize8x8()* predstavljeno uz korektno zaokruživanje i za pozitivne i negativne vrijednosti.

```
6
7  /* Kvantizacija: q[i] = round(F[i] / Q[i]) */
8  void quantize8x8(const float F[64], const u8 Q[64], s16 Fq[64])
9  {
0      for(int i = 0; i < 64; i++)
1      {
2          Fq[i] = round_to_s16(F[i] / (float)Q[i]);
3      }
4  }
5
```

Slika 2.5 – *Funkcija quantize8x8()*

Kvantizovani koeficijenti se preuređuju metodom zig-zag, kako bi se niskofrekventni koeficijenti smijestili na početak, a niz visokofrekvencijskih (često nula) na kraj. Time se povećava efikasnost izvršavanja kodovanja.

U *write_block_zigzag_rle()* se radi zig-zag redoslijed, DC komponenta se koduju diferencijalno (DC diff u odnosu na prethodni blok), AC koeficijenti se koduju RLE parovima (*run, value*) i kraj bloka se označava EOB markerom (*0,0*).

```
void write_block_zigzag_rle(FILE* f, const s16 q[64], s16* prev_dc)
{
    s16 zz[64];

    // zigzag
    for(int i=0; i<64; i++)
    {
       zz[i] = q[ZZ[i]];
    }
    // DC diff
    s16 dc = zz[0];
    s16 dc_diff = (s16)(dc - *prev_dc);
    *prev_dc = dc;
    fprintf(f, "%d ", (int)dc_diff);
    //write_s16_le(f, dc_diff);
    // AC rle
    int run=0;
    for(int i=1; i<64; i++)
    {
        s16 val=zz[i];

        if(val==0)
        {
            run++;
            continue;
        }

        //write_u8(f, (u8)run);
        //write_s16_le(f, val);
        fprintf(f, "%u %d ", (unsigned)run, (int)val);
        run = 0;
    }
    fprintf(f, "0 0\n");
    /* EOB-end of block marker*/
     //write_u8(f, 0);
     //write_s16_le(f, 0);
}
```

Slika 2.5 – *Funkcija write_block_zigzag_rle()*

Kako bi se provjerila ispravnost implementirane kompresije i kvantitativno ocijenijo kvalitet rekonstruisane slike, izvrsena je dekompresija u Python-u i izračunata metrika MSE (*Mean Squared Error*) u odnosu na originalnu sliku. Kompresovani rezultat se nalazi u fajlu imgX.dct (tekstualni foormat), koji sadrži zaglavlje sa dimenzijama slike, brojem blokova i kvantizacionom tabelom, te sekvencu kodovanih blokova.

Fajl .dct započinje oznakom „magic“ DCT8_TXT, nakon čega slijedi širina w, visina v, broj blokova po x i y (BxN, byN), kvantizacijona tabela od 64 elementa, kodovni podaci po blokovima dc_diff, a zatim parovi RLE - *Run-Length Encoding* (run, val) sve do oznake 0 0. U Python skripti se fajl parsira kao sekvenca tokena (brojeva i oznaka), pri čemu se za svaki blok :

- rekonstruiše DC komponenta iz diferencijalnog zapisa *dc=prev_dc+dc_diff,*
- RLE parovi se „raspakuju“ u zig-zag niz koeficijenata
- Vrši se inverzni zig-zag raster redoslijed 8x8

Nakon dobjenih kvantizovanih koeficijenata $F\hat{}(u,v)$, radi se dekvantizacija:

$$F(u,v) = F\hat{}(u,v)Q(u,v) \tag{2.7}$$

čime se vraćaju aproksimirane DCT vrijednosti prije kvantizacije. Zatim se primjenjuje IDCT korišćenjem iste ortogonalne matrice C kao u C implementaciji:

$$f = C^T F\ C \tag{2.8}$$

Kako je prije DCT blok centriran oduzimanjem 128, nakon IDCT u Pythonu se radi vraćanje opsega:

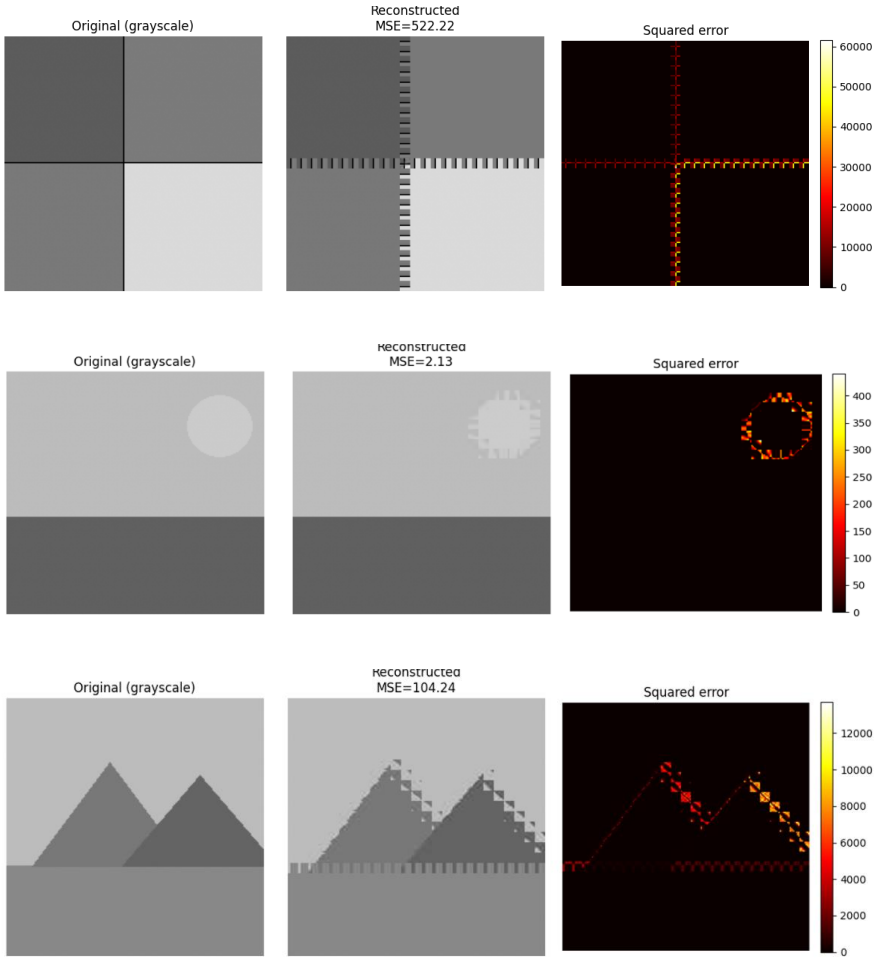
$$Y(x,y) = f(x,y) + 128 \tag{2.9}$$

i vrijednosti se ograničavaju na [0, 255]. Radi korektnog poređenja u Python-u se takođe slika pretvara u sivu sliku koristeći isti izraz kao u DSP kodu, na taj način se izbjegava greška koja bi nastala kada bi se originalna slika pretvarala drugim grayscale algoritmom.

$$MSE = \frac{1}{WH}\sum_{x=0}^{W-1}\sum_{y=0}^{H-1}(I(x,y) - I\hat{}(x,y))^2 \tag{2.10}$$

Gdje je *I* originalna grayscale slika, a *I^* rekonstrujisana slika dobijena dekompresijom.

U rezultaima se pored numeričke vrijednosti MSE, prikazuju : originalan grayscale slika, rekonstrujisana slika i mapa kvadratne gerške po pikselu, koja vizuelno pokazuje gdije se greške najviše javljaju:



Slika 2.5 – *Reprezentovanje dobijenih rezultata od slike img1.bmp, img2.bmp, img3.bmp*

Napomena: Greška (MSE) je očekivana zbog kvantizacije koja uvodi grešku i „ odbacuje ” dio visokofrekvencijskih detalja.Tipično što su veće vrijednosti u Q tabeli (agresivnija kvantizacija), veći je stepen kompresije i veći MSE.

Optimizacija resursa: Za praćenje broja ciklusa potrebnih da se izvrše sve faze obrade slike koristimo zaglavlje *cycle_count*. Ovo zaglavlje sadrzi dva makroa kojim se omogućava pristup sadržaju EMUCLK registara koji prikazuju broj izvršenih ciklusa : *START_CYCLE_COUNT(S)* i

STOP_CYCLE_COUNT(T, S), gdje su parametri T i S *cycle-t* tipa. Takođe je definisan i makro za ispis promijenljive tipa *cycle-t PRINT-CYCLES(String, T)*.

Za slučaj bez optimizacije broj potrebnih ciklusa za izvršavanje pojedinacnih fazaobrade izbosi:

- Ucitavanje slike: 8425631 ciklusa,
- rgb_to_gray: 1640041 ciklusa,
- Ispis sive: 10 000 759 ciklusa,
- extract_block_centered=3 070 000 ciklusa,
- dct =23 862 500 ciklusa,
- quant total=3 637 741 ciklusa,
- zig_zag_rle total=3 519 498 ciklusa,

Iz dobijenih rezultata se može primjetiti da najviše resursa zauzima algoritam za dct što je i očekivano jer algoritam primjenjuje 2D DCT na svaki blok.

Naredna optimizacija je optimizacija po brzini , poterno je da izaberemo *Enable optimization* ili da pozovemo *#pragma optimize_for_speed* i dobijamo rezultate:

- Ucitavanje slike: 7 060 113 ciklusa,
- rgb_to_gray: 440 037 ciklusa,
- Ispis sive: 9 034 174 ciklusa,
- extract_block_centered=1001875 ciklusa,
- dct =1 635 000 ciklusa,
- quant total=1 895 625 ciklusa,
- zig_zag_rle total=2 082 726 ciklusa,

Primjećuje se značajno smanjenje broja ciklusa pri obradi slike .

Ono što je ključ ovoga jeste ukoliko bi nastavili sa optimizovanjem, vrlo brzo bi primjetili da se nije desilo poboljšanje u odnosu na to kad nemmao optimizaciju.

Signalizacija napredka LED diodama:

Radi demonstracije rada ADSP-21489 razvojnoj platformi, implementirana je signalizacija LED dioda putem SRU jedinice (sru21489.h) i flag pinova.

3. Zaključak

Implementiran je funkcijonalan sistem kompresije slike yasnovan na blokovskoj DCT transformaciji, kvantiyaciji i jednostavnom RLE kodovanju. Sistem je verifikovan kroz dekompresiju u Python-u izračunavanjem MSE u odnosu na originalnu sliku, a performanse su analizirane mjerenjem ciklusa po bloku. Kao moguća poboljšanja izdvajaju se prelazak na binarni format zapisa, implementacija entropijskog kodvanja (npr. Huffman) , mogućnost obrade slike većih dimenzija .

4. Literatura

[1] https://www.analog.com/media/en/dsp-documentation/evaluation-kit-manuals/ADSP_21489_EZ_Board_Manual_Rev_1_0_April_2010.pdf

[2] https://en.wikipedia.org/wiki/BMP_file_format

[3] https://dev.to/marycheung021213/understanding-dct-and-quantization-in-jpeg-compression-1col

[4] Materijali sa predavanja i laboratorijskih vježbi