Danmarks Tekniske Universitet

02131 Embedded Systems - Assignment 1

Udarbejdet af:

Afleveret via CampusNet

s081828 - Ibragimov, Arthur

Afleveret via CampusNet

s103473 - Olsen, Anders

DTU Informatik
Institut for Informatik og Matematisk Modellering

02131 Embedded Systems

Vejleder: Paul Pop
Afleveringsdato: 24. oktober 2011
Rapporten indeholder 47 sider inkl. forside

Indhold

Indledning 1

I Assignment 1 ligger fokus på softwaredelen af et digital kamera. Her skal analyseres, designes og implementeres en applikation for et digital kamera, som kan udføre billedemanipulationer.

Man skal designe og implementere algoritmer for billedefiltrering og komprimering / dekomprimering. De skal testes og optimeres med hensyn til at, de skal kunne bruges på et indlejret system som har nogle hukommelses- og hastighedsbegrænsninger.

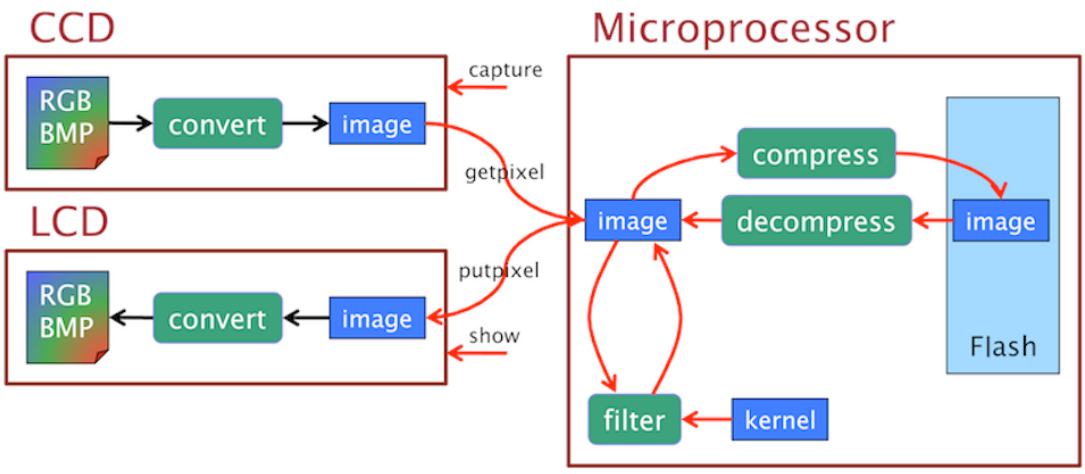
Funktionelle krav for applikationen:

- Aflæsning af 8bit og 24bit BMP billeder
- Lagring af 8bit grayscale billede
- Aflæsning og lagring af 8bit billede ved brug af RLE8-komprimering og dekomprimering.
- Filtrering af billede ved hjælp af faste 3×3-matricer.

Ikke-funktionelle krav:

- Programmeringssproget er C
- Korrekte og optimerede algoritmer

Der er blevet givet skelet for applikationen i form af .h og .c filer, som følger arkitekturen som er vist på figur 1



Figur 1: Software Arkitektur

Denne rapport vil primært omhandle hvordan vi fik implementeret vores funktionelle krav. Vi vil derudover have en smule analyse af vores program, samt kørsel af vores program på en simuleret ARM-processor.

Metoden for udvikling af applikationen er følgende:

1. Billedemanipulation algoritme designes og implementeres.

2. Komprimering/dekomprimerings algoritme designes og implementeres.

3. Applikationen analyseres med hensyn til at finde flaskehalse i softwaren og herfra udtænke en fordeling af funktioner mellem software og hardware delene.

4. Sammenligning af hastigheden på ARM og Intel processorer.

Grunden til at vi har et digital kamera som emne er, at det hele kan simuleres og er meget nutidigt.

Et digitalkamera har, rent basalt, én opgave: Tag et billede og gem det i hukommelsen. Dette er ikke så forskelligt fra hvad et almindeligt computerprogram kan gøre, og det gør at vi kan implementere hele vores digitalkamera som et program, ved at substituere CCD/CMOS-lyssensorerne som genererer elektriske signaler, med en BMP-fil som har data for de individuelle pixels liggende.

Når man har dette, skal man bare behandle pixel-data i BMP-filen som inputdata fra en CCD, og så har man et software baseret digitalkamera.

Idet C-sproget er nyt for os, og vi indtil videre aldrig har arbejdet på bit- og bytebasis i en fil, bliver det i starten svært at få de resultater som vi gerne vil have. Det sværeste bliver formentlig at køre et billedefilter som er et multidimensionel array over på vores pixeldata som er et almindeligt array.

Vi regner med at vores program opfylder de funktionelle krav, og har en lille menu til brugeren.

Analyse 2

Når man skal simulere et digitalkamera, er der følgende problemer som man skal overveje hvordan man vil takle:

1. Hvordan henter man pixeldata ind, når man ikke har en fysisk CCD/CMOS-sensor at hente data fra ?

2. Hvilket format skal man gemme sine billeder som; skal man gå efter kvalitet, filstørrelse eller hastighed/simplicitet?

3. Hvilke billedefiltre skal man inkludere ?

2.1 Problem 1: At læse pixeldata

Når vi skal læse vores pixels, så er der et par ting vi skal kende til, så som:

1. Er vores pixels i greyscale, dvs er een værdi lig med en pixel ?

2. Er vores pixels RGB-indexed så der går 3 værdier per pixel?

Der findes forskellige løsninger til til dette problem.
Vi kan læse vores data ind via en tegnsepareret fil, så som en `.csv` fil.
Vi kan læse vores pixels ind fra et eksisterende billede i `bmp`, `jpeg`, `gif` eller `png`-format.

2.2 Problem 2: Hvordan skal billedet gemmes

Til at gemme et billede, er det oplagt at bruge et almindeligt billedeformat som alle styresystemer har et program til at åbne, dvs. enten `bmp-`, `jpeg-`, `gif-` eller `png`-format.

Som der står i kravsspecificeringen under Indledning | af Anders Olsen (50%) og Arthur Ibragimov (50%), skal `bmp` formatet bruges i denne opgave.

2.3 Problem 3: Hvilke billedfiltre skal inkluderes

Der findes tusindvis af forskellige billedefiltre, som alle baserer sig på en $n \times n$ matrix.

Når man skal vælge matricer, så skal man overveje flere ting:

1. Hvilke filtre vil en almindelig forbruger normalt have brug for ?

2. Hvilken størrelse skal filtermatricen være?

De mest gængse billedefiltre vil være filtre så som: sløring (blur), bevægelsessløring(motion blur), kant-detektion (edge detection), gøre billedet skarpere (sharpening)

Jo større en matrix man bruger, desto mere glat effekt får de forskellige filtre, men det tager tilgengæld meget længere tid, jo større matrixen er.

Design 3

I forrige afsnit, så vi nogle af de problemer som man skulle overveje i forbindelse med designet.

Da vi ikke har en fysisk CCD/CMOS-sensor at læse vores pixels fra, så har vi som opgave at bruge en BMP[1]-fil til at holde de pixels som vi vil læse ind i vores program og behandle.

Grunden til at BMP er oplagt, er den simple struktur som filen har. Filen består af:

- En Bitmap File Header som beskriver filtypen, filstørrelsen og afstanden fra start til at pixeldata.
- En Bitmap Info Header som beskriver dimensionerne af billedet, antal bits pr pixel samt komprimering
  - Hvis antal bits pr pixel er < 24, så er der en farvepalette.
    Farvepaletten indeholder 3 arrays med 256 heltal i hver, som beskriver farven for en given værdi. For at få et gråtonet billede, skal de tre rækker af heltal være ens, dvs 1-1-1, 2-2-2, 3-3-3, osv.
  - Hvis antal bits pr. pixel er = 24, så starter pixeldata direkte efter infoheaderen
- Pixeldata, som enten er 3 byte pr. pixel eller 1 byte pr. pixel.
  - Hvis bits pr. pixel er = 24, så er der tre bytes pr. pixel, en værdi for den røde, den grønne og den blå komposant. Dette gør at man kan have en masse nuancer af farver, helt præcist 16.777.216 forskellige farver.
  - Hvis bits pr. pixel er = 8, så er der en byte pr. pixel, hvor værdien så går ind og ser på de tre positioner i paletten. Da paletten kun har plads til 256 forskellige farver, så giver det kun 256 mulige farver.

Algoritmer bliver designet efter de formler og råd som er givet i opgavebeskrivelsen og kravspecificeringen.

[1] Bitmap Image File

Vi valgte at ikke implementere en "workaround"med filtrering af kanter. Det krævede ekstra tid som vi ikke kunne finde.

Algoritmen på Listing 1 kunne være løsningen, men vi har ikke testet den grundigt, derfor har ikke inkluderet i selve implementeringen.

Listing 1: Første design af filtrerings algoritmen

```
fCenter = f / 2;     // filter matrix size/2

for(row = 1; row < w; ++row)                // rows
{
    for(column = 1; column < h; ++column)           // columns
    {
        for(i=0; i < fRows; ++i)      // filter rows
        {
            mm = fRows - 1 - m;

            for(j=0; j < fCols; ++j) // filter columns
            {
                nn = fCols - 1 - j;

                // used for checking boundary
                ii = row + (i - fCenter);
                jj = column + (j - fCenter);

                // ignore inputs which are out of bound
                if( ii >= 0 && ii < w && jj >= 0 && jj < h )
                out[row][column] += in[ii][jj] * filter[mm][nn];
            }
        }
    }
}
```

3.1 Valg af fastsatte parametre

Da det er en meget beregningstung opgave, så vil vi kunne optimere så meget som muligt, og har derfor valgt ikke at lade brugeren bestemme nogle værdier. Dette gør at vores compiler har en chance for at optimere, idet den kender de faste værdier.

På et almindeligt kamera har brugeren også kun nogle faste menupunkter at vælge, og alle brugerdefinerede filtre vil som regel blive brugt i forbindelse med et foto-redigeringsprogram på en computer, så som Adobe Photoshop eller GIMP[2]. Grundet den lille skærm på et digitalkamera, er det altid svært at se detaljer, og man vil derfor ikke bruge særlig meget tid på filtre, da det kan være svært at se om det giver det ønskede resultat.

Vi har dog valgt at brugeren selv kan skrive navnet på inputfilen, således at han har nogen kontrol, således at han ikke behandler det samme billede hver gang. Oftest vil dette være fastlåst i et digitalkamera, således at billederne får navnet IMG*XXX* hvor *XXX* er værdien for den interne billedetæller.

[2]GIMP is the GNU Image Manipulation Program. It is a freely distributed piece of software for such tasks as photo retouching, image composition and image authoring. It works on many operating systems, in many languages. http://www.gimp.org/

3.2 Størrelse på filterne

Vi har valgt et standard 3×3 matrix som standard for vores filter. Det er for at få den bedste hastighed mulig.

Hvis man regner lidt på det, så har vi et filter hvor vi skal regne på $3^2 = 9$ pixels.

Hvis vi udvider vores filter til en 5×5 matrix, så skal vi pludselig regne på $5^2 = 25$ pixels. Det er 277% så mange pixels i forhold til vores 3×3.

Hvis vi så udvider atter engang til en 7×7, så har vi lige pludselig $7^2 = 49$ pixels, et forhold på 544%.

Når man har en kraftig hastighedsbegrænsning, så kan det komme til at tage virkelig lang tid, jo større vores filter er. Hvis det for eksempel tager 10 minutter at åbne et billede, lægge et filter på og gemme det igen, med en 3×3 matrix, så vil det lige pludselig tage ≈ 54 minutter med en 7×7 matrix.

3.3 Komprimering

Der findes flere forskellige metoder for komprimering af data.

Her bruger vi en af de nemmeste datakomprimeringsalgoritmer – "Run-length encoding", som er beskrevet i opgavebeskrivelsen og i sektion 4 af Microsoft Windows Bitmap Format.

Metoden er ikke den mest effektive (i worst case bliver filen større), men nem at implementere.

I denne opgave vores primære fokus ligge på softwaredelen, derfor implementerer vi komprimeringsmetoden i C, men i virkeligheden ville det være mere effektivt at implementere komprimeringen i hardware delen, dvs. når man gemmer eller aflæser filen foregår komprimering/dekomprimering direkte.

Implementering 4

4.1 Løsning 1: At læse pixeldata

Alle dele skal læses sekventielt fra filen, altså fra start til slut og det letter selve processen med at hente billedet ind:

Selve processen kan beskrives således:

Listing 2: Process til at læse BMP billede ind

```
Read the Bitmap File Header
Read the Bitmap Info Header
if (bitcount == 24) {
  Read the pixels, 3 byte pr pixel
}
else if (bitcount == 8 && compression == 0){
  Read the color palette
  Read the pixels, 1 byte pr pixel
}
else if (bitcount == 8 && compression == 1){
  Read the color palette
  while (There are still more pixeldata){
    Read the number of continues pixels
    Read the pixels value
    from (1 \to number of sequent pixels){
      Save the current pixel value into memory
    }
  }
}
```

4.2 Løsning 2: Hvordan billedet skal gemmes

Ligesom at vi læser fra en `bmp` fil, så gemmer vi også som en `bmp` fil, med den ændring at:

Hvis billedet har 24 bit pr pixel, så:

- laver vi en palette med gråtoner, dvs de tre rækker i arrayet har samme værdi.
- henter vi værdierne for Rød, Grøn og Blå, og laver en beregning der siger: Ny pixelværdi $= 0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B$
  Dette resulterer i en grå farve når vi ser på denne positions farve-værdi i vores palette.
- sætter BiBitCount = 8 således at billedet er et gråtonet billede.

Hvis billedet har 8 bit pr pixel, så:

- læser vi farvepaletten ind, og beholder den til det nye billede.

- læser vi hver pixel ind, uden at gøre noget ved dem.

Når vi så er færdige med at læse billedet, så skriver vi så alle delene tilbage til filen, i samme rækkefølge som vi læste dem:

- Skriv File Header til den nye outputfil.
- Skriv Info Header til den nye outputfil.
- Skriv greyscale farvepaletten.
- Skriv så hver pixel som en enkelt byte

4.3 At tilgå pixel arrayet og anvende filteret

Vi ved at et billede altid består af $x \times y$ pixels, men når vi læser vores pixels ind, så får vi et enkelt-dimensionelt aray som er $x \times y$ bredt.

Dette giver problemer når vi skal lægge filteret på, idet filteret bliver ganget på de pixels der ligger omkring. Derfor er vi nødt til at lave beregninger for hvor de pixels som ligger i rækken over den nuværende pixel ligger i det flade array.

For at få fat på den originale værdi som vi lægger filter på, bruger vi følgende linje:

Listing 3: At finde en bestemt pixel fra vores pixelarray

```
currentPixel = pixels[row*w+column];
```

Når vi så skal lægge filteren på, bruger vi formlen fra opgaven som siger:

$$\sum_{i=-\frac{m-1}{2}}^{\frac{m-1}{2}} \sum_{j=-\frac{m-1}{2}}^{\frac{m-1}{2}} F[i,j]I[r+i,c+j]$$

Vi har oprettet vores filter som et multidimensionelt array, således:

Listing 4: Eksempel på et af vores filtre

```
1 //Blur-filter
2 int filter1[3][3] = {
3     {0,1,0},
4     {1,1,1},
5     {0,1,0}
6 };
```

Når dette skal implementeres, bruger vi to for-loops til at udføre summationsfunktionerne:

Listing 5: At finde en bestemt pixel fra vores pixelarray

```
for (x = -(m-1)/2; x <= (m-1)/2; x++){
        for(y = -(m-1)/2; y <= (m-1)/2; y++){
          tempPixel += filter[x+1][y+1] * pixels[((row+x)*w) + column+y];
        }
      }
```

Da vores filtermatrix er 0-indexed, er vi nødt til at lægge 1 til, for ikke at få en `OutOfBounds` fejl.

Til sidst er vi nødt til at multiplicere vores nye pixel-værdi med en normaliseringsværdi for at korrigere kontrast. Normaliseringsværdien er lig summen af vores filtermatrix, men hvad gør vi, hvis filtersummen er lig 0 ?

Vi har valgt at sætte normaliseringsværdien til 1, i tilfælde af at filtersummen er 0.

Dette er gjort med et simpelt `if-else` check:

Listing 6: Beregning af normaliseringsværdi

```
filterSum = 0;
for (x = 0; x < m; x++){
  for(y = 0; y < m; y++){
    filterSum += filter[x][y];
  }
}
double normalize;
if (filterSum != 0){
  normalize = 1.0/filterSum;
}
else {
  normalize = 1.0;
}
```

Vi skal så overveje hvad vi skal gøre, i tilfælde af at vi får en ny pixelværdi som ligger udenfor vores farvepalette. Vi har så valgt at sætte værdien = 0, således at pixellen bliver komplet sort, da det er mindre åbenlyst end en hvid farve.

Vi har valgt at gøre dette således:

Listing 7: Tjek om den nye pixelsværdi ligger indenfor paletten

```
newPixel = normalize * tempPixel;
if (newPixel > 0 && newPixel < 255){
  filteredPixels[row*512+column] = newPixel;
}
else if (newPixel < 0) {
  filteredPixels[row*512+column] = 0;
}
else {
  filteredPixels[row*512+column] = 0;
}
```

4.4 Komprimering af billede

Vores simple "Run-length encoding"komprimering virker således:

Istedet for at skrive "00 00 00 00 00"så skrives der istedet "05 00". Når filen så skal dekomprimeres, så forstås de to tal som `number` og `value`

Programmet læser det herefter som at der er `number` pixels i træk, alle med pixelværdien `value`

Dette har vi implementeret således:

Listing 8: Dekomprimeringsloop

```
1  BOOL running = 1;
2      printf("Reading the pixels from the compressed image:\n");
3      while(running){
4        fread(&B1, sizeof(BYTE), 1, fp);
5        fread(&B2, sizeof(BYTE), 1, fp);
6        printf("B1: %d B2: %d\n",B1,B2);
7
8        if (B1 != 00){
9          for (i = 0; i < B1; i++){
10           image->Pixels[x] = B2;
11           x++;
12         }
13       }
14       else if(B1 == 0 && B2 == 1){
15         printf("Done running");
16         running = 0;
17       }
```

Hvor `B1` og `B2` er deklareret som to `BYTE` objekter.

Når vi så skal komprimere vores billede, så gælder der følgende ting:

- Hvis der er mere end 255 ens pixels i træk, så skal vi stoppe ved 255, skrive `255 <pixelværdi>` og så starte tælleren forfra og så går videre med at tælle
- Vi tæller fra første pixel og hver eneste pixel fremad. Når vi har talt til det antal pixel som billedet er bredt, så skal der sættes et specielt linjeskift ind, og så nulstilles pixel-tælleren.

Vi har implementeret dette således:

Listing 9: Implementering af komprimering

```
1  ... if (bmih.BiCompression == 1){
2      for(i = 0; i < image->Height * image->Width; i++){
3        if (B2 == image->Pixels[i]){
4          B1++;
5          if(B1 == 255){
6            fwrite(&B1, sizeof(B1),1,fp);
7            fwrite(&B2, sizeof(B2),1,fp);
8            numberOfBytes = numberOfBytes + 2;
9            B1 = B1-255;
10         }
11       }
```

```
12        else {
13          //Only do it if B1 is not 0
14          if(B1 != 0){
15            fwrite(&B1, sizeof(B1), 1, fp);
16            fwrite(&B2, sizeof(B2), 1, fp);
17            numberOfBytes = numberOfBytes  +2;
18          }
19          B1 = 1;
20          B2 = image->Pixels[i];
21        }
22
23        pixelCounter++;
24
25        if (pixelCounter == 512){
26          fwrite(&zeroIndicator, sizeof(zeroIndicator),1,fp);
27          fwrite(&zeroIndicator, sizeof(zeroIndicator),1,fp);
28          numberOfBytes = numberOfBytes + 2;
29          pixelCounter = 0;
30        }
31      }
32
33      //At the end of the pixels, write the EOF:
34      B1 = 0; B2 = 1;
35      fwrite(&B1, sizeof(B1), 1, fp);
36      fwrite(&B2, sizeof(B2), 1, fp);
37      numberOfBytes = numberOfBytes +2;
38
39      //Change the SizeImage value and re-write the infoheader:
40      bmih.BiSize = sizeof(bmih);
41      bmih.BiSizeImage = numberOfBytes;
42      fseek(fp, sizeof(bmfh), SEEK_SET);
43      fwrite(&bmih, sizeof(bmih),1,fp);
44    }
```

Hele koden kan man se i `Desktop - bmp.c` under Appendix.

Resultat 5

Vi brugte primært følgende billede til vores program:

Figur 2: 24bit BMP billede med kvinde som motiv

Fælles for alle vores output-billeder er, at de er blevet greyscale. Herudover har vi brugt tre forskellige filtre, nemlig sløring (blur), bevægelsessløring (motion blur) og kant detektion (edge detection).

På næste side kan I se resultaterne[3].

[3] Billederne er blevet krympet for at kunne passe på siden. Se de vedhæftede filer for de fulde billeder.

(a) Almindeligt greyscale

(b) Blur-effekt

(c) Motion blur-effekt

(d) Edge detection-effekt

Figur 3: Vores 4 output-billeder

Vi har lavet forskellige analyser af vores program, heriblandt en profiling ved hjælp af gprof[4]. Når et program er blevet kompileret med profiling flaget `-pg` vil, hver gang programmet bliver kørt, genere en fil med navnet `gmon.out` hvori alle data om kørselstider og funktionskald bliver gemt.

Det vil dog kun være de funktioner som bliver brugt, der bliver registreret og derfor har vi valgt at lave to profiler af vores program; en profil uden noget filter, og en profil med filter.

Her er vores resultater[5]

```
C:\Users\Anders Olsen\Dropbox\DTU\Indlejrede Systemer\Workspace\Assignment 1>gprof
 -p "Debug\Assignment 1.exe" gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
100.00      0.01     0.01        1    10.00    10.00  bmp_open
  0.00      0.01     0.00   262144     0.00     0.00  ccd_get_pixel
  0.00      0.01     0.00   262144     0.00     0.00  lcd_set_pixel
  0.00      0.01     0.00        1     0.00     0.00  bmp_readFileHeader
  0.00      0.01     0.00        1     0.00     0.00  bmp_readInfoHeader
  0.00      0.01     0.00        1     0.00     0.00  bmp_save
  0.00      0.01     0.00        1     0.00    10.00  ccd_capture_image_custom
  0.00      0.01     0.00        1     0.00     0.00  ccd_get_height
  0.00      0.01     0.00        1     0.00     0.00  ccd_get_width
  0.00      0.01     0.00        1     0.00     0.00  ccd_reset_pointer
  0.00      0.01     0.00        1     0.00     0.00  lcd_reset_pointer
  0.00      0.01     0.00        1     0.00     0.00  lcd_set_height
  0.00      0.01     0.00        1     0.00     0.00  lcd_set_width
  0.00      0.01     0.00        1     0.00     0.00  lcd_show_image_custom
```

Figur 4: Profil data for kørsel uden filter

```
C:\Users\Anders Olsen\Dropbox\DTU\Indlejrede Systemer\Workspace\Assignment 1>gprof
 -p "Debug\Assignment 1.exe" gmon.out

Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 75.00      0.03     0.03        1    30.00    30.00  applyFilter
 25.00      0.04     0.01        1    10.00    10.00  bmp_open
  0.00      0.04     0.00   262144     0.00     0.00  ccd_get_pixel
  0.00      0.04     0.00   262144     0.00     0.00  lcd_set_pixel
  0.00      0.04     0.00        1     0.00     0.00  bmp_readFileHeader
  0.00      0.04     0.00        1     0.00     0.00  bmp_readInfoHeader
  0.00      0.04     0.00        1     0.00     0.00  bmp_save
  0.00      0.04     0.00        1     0.00    10.00  ccd_capture_image_custom
  0.00      0.04     0.00        1     0.00     0.00  ccd_get_height
  0.00      0.04     0.00        1     0.00     0.00  ccd_get_width
  0.00      0.04     0.00        1     0.00     0.00  ccd_reset_pointer
  0.00      0.04     0.00        1     0.00     0.00  lcd_reset_pointer
  0.00      0.04     0.00        1     0.00     0.00  lcd_set_height
  0.00      0.04     0.00        1     0.00     0.00  lcd_set_width
  0.00      0.04     0.00        1     0.00     0.00  lcd_show_image_custom
```

Figur 5: Profil data for kørsel med filter

Som man kan se på figur 4 så tager det omkring 10 ms at åbne billedet og beregne et nyt greyscale billede ud fra pixeldata med `bmp_open`. Derudover så går det så hurtigt med at gemme, at `bmp_save` slet ikke får målt nogen tid.

[4]The GNU Profiler

[5]Begge profiler er blevet dannet under kørsel på Windows 7 64-bit med en AMD Phenom II X4 945 (3,0 GHz)

Når man ser på figur 5 så ligner det til forveksling det øverste billede, men med den forskel at `applyFilter` nu er kommet i brug.

`applyFilter` tager 3 gange så lang tid at køre igennem som `bmp_open`, men det giver jo også meget god mening, idet der skal laves mange flere beregninger i `applyFilter`.

Vi har også lavet tidsmålinger på vores ARM-simuleringer. Alle tidsmålinger er påbegyndt efter at brugeren har givet sit input, således at brugerens reaktionshastighed ikke har nogen indflydelse på tiden.

Effekt:	Start-clock [s]	Stop-clock	Time used	Total Cycles	Time per cycle [ns]
No filter	860000	160500000	159	159640000	995,99
Blur	210000	283570000	283	283360000	998,73
Motionblur	920000	329270000	328	328350000	998,93
Edge	930000	312260000	311	311330000	998,94

Tabel 1: ARM Simuleringer

Dette giver os en gennemsnits-clock på $\frac{1}{998{,}14ns} = 1001854,782$ Hz$= 1001$ MHz.

Diskussion 6

I dette afsnit vil vi kigge på vores resultater, vores oplevelser gennem projektet og vores generelle overvejelser for et digital kamera.

6.1 Fordeling af tidsforbrug

Under vores resultater, ser man at `applyFilter` tager 75% af kørselstiden.

På et almindeligt digitalkamera er det yderst vigtigt at billedet bliver taget lynhurtigt, således at kameraet hurtigt er klar til næste billede. Der er intet værre end kun at kunne nå at tage et enkelt billede når noget vigtig eller spændende lige pludselig sker.

En ideel metode til at kunne tage billeder hurtigt, ville være at have en lynhurtig cache som alle billeder bliver sendt til, uden nogen form for databehandling. Når så fotografen er færdig med at tage billeder (dvs de små tidsrum hvor kameraet ikke ikke er igang med at fokusere og tage billeder) så vil processoren så arbejde på at gemme billeder i den langsommere flash-hukommelse efter de valgte indstillinger.

Alternativt kunne man have to processorer, eller en multi-kerne processor, hvor den ene processor eller kerne så står for kun at tage billeder og gemme dem i en midlertidig cache og så vil den anden processor/kerne så arbejde med filter og komprimering, oftest JPEG-komprimering hvor der bl.a. bruges 8×8 matricer hvilket tager relativt lang tid at arbejde med.

6.2 Valg af processor

Til dette projekt, fik vi at vide at vi skulle køre vores program på en simuleret ARM-processor. Dette virker også logisk, da det ikke er realistisk at have en CPU fra en bærbar, installeret i et digital kamera.

ARM-processorerne er blevet meget mere udbredt de sidste 5-6 år, idet Apples iPhone[6] samt mange Android-baserede smartphones[7] bruger ARM-processorer.

Dette er også fint, fordi en smartphones bedste egenskab er, at den kan køre programmer som en almindelig pc. Af samme grund, giver det ikke nogen mening at bruge en RISC-processor til et digitalkamera.

Et digitalkamera har én og kun én opgave fra det bliver solgt, til at det går i stykker. Det er at tage billeder, gøre noget bestemt med disse billeder og derpå gemme dem. Et digitalkamera skal ikke lige pludselig kunne modtage smser, spille musik eller gå

[6] http://en.wikipedia.org/wiki/Apple_A4
[7] http://en.wikipedia.org/wiki/HTC_Desire#Hardware

på nettet. Derfor ville det logiske være, at bruge en ASIC struktur; Application Specific Integrated Circuit. Et integreret kredsløb som er beregnet til en bestemt anvendelse.

Det bedste valg til et kamera ville være at bruge en Digital Signal Processing processor, da vores billedsensor ikke har nogen instruktioner eller data men kun leverer digitale signaler som så skal behandles og gemmes.

6.3 Energieffektivitet

Man skal også huske at et digitalkamera har en begrænset energikapacitet. Algoritmer med lange køretider og kraftige processorer øger energiforbruget og forkorter derved tiden som kameraet kan være tændt i. En digitalkamera er et mobilt apparat som man tage med, og som skal være i stand til at tage billeder i lang tid og derfor duer det ikke at det løber tør for strøm i løbet af ingen tid.

Energi effektivitet er en anden grund til at vælge specifikke `Digital Signal Processing` processorer og implementere komprimeringsfunktionen i hardwaredelen.

6.4 2D Convolution

Opgavebeskrivelsen siger at vi skal bruge de konventionelle convolution metoder som er beskrevet i sektion 2.1 af opgaveformuleringen.

Vi har foretaget en lille research og fundet ud at man kunne bruge mere avanceret metoder for 2D convolution. Metoden hedder "Fast Fourier Transform Convolution". Ved hjælp af FFT algoritmen kan man repræsentere filtermatricen og pixeldata i form af signaler (funktioner), og så få en tredje funktion som resultat af de to funktionernes overlap. Bagefter konverterer man signalet igen til pixeldata.

FFT er meget hurtigere end de konventionelle convolution metoder, især hvis man bruger større filtermatricer.

Desuden findes der flere gratis C biblioteker som implementerer FFT, f. eks. FFTW [8]

Selv om det kunne være smartere at bruge FFT algoritmen for billedmanipulation og komprimering, så valgte vi at implementere convolution med de konventionelle metoder pga. begrænset tid og vores manglende erfaring med C sproget, .

En anden problem som kunne være løst vha. FFT, er et problem med filtrering af kanter, fordi vi bliver nødt til at holde øje med matricens grænser når vi bruger konventionelle metoder .

[8] `http://www.fftw.org/`

6.5 Hastighed på simulering

Under vores resultater kom vi frem til, at vores simulerede enhed havde en clockfrekvens på ≈ 1101 MHz. Hvordan kan det så være at, at det tager ≈ 6 minutter at loade et billede ind, lave det om til gråtoner, lægge et filter på og gemme det igen, når præcis samme process tager < 1 sekund på en 3000 MHz processor ?

Det må simpelthen være fordi at vores simulerede processor skal bruge flere clocks til at udføre den samme opgave. Grunden til at det tager flere clock-cycles, må så være en dårlig opbyging af den simulerede enhed, at enheden simpelthen bruger for mange clocks for at gøre noget simpelt.

6.6 Problemer med simulering

Da vi startede på projektet, fik vi udleveret et skelet til at arbejde ud fra. Vi fik vores program til at køre, uden problemer, men da vi så skulle over og køre det på vores simulerede ARM, så fik vi lige pludselig problemer.

For det første havde vi lavet vores `applyFilter` funktion i `bmp.c`. Det var jo logisk for os at placere en funktion som har med et bmp-billede at gøre, inde i selve filen som har alle funktionerne til at arbejde med bmp-filer.

Derudover havde vi lavet vores program med vores egne `bmp_open` og `bmp_save` funktioner, og vores `applyFilter` funktion havde direkte adgang til IMAGE->pixels[] arrayet.

Alt dette var lige pludselig ubrugeligt da vi skulle over på ARM-simuleringen, for der var der ikke mulighed for at interagere med selve pixel-arrayet og vi skulle lige pludselig heller ikke bruge de funktioner til at åbne og gemme bmp-filer, som vi havde siddet og arbejdet med de forrige uger.

Det "eneste"vi egentlig skulle, var at:

1. Bryde udtrykket i `main.c` hvor vi havde `lcd_set_pixel(ccd_get_pixel())` således at vi fik gemt alle pixels fra `ccd_get_pixel()` i et integer-array.

2. Køre `applyFilter` på vores nye midlertidige pixel-array

3. Køre et loop, hvor vi sender alle pixels fra vores behandlede array til LCDen med `lcd_set_pixel()`

men dette var meget svært at forstå ud fra det opgaveoplæg vi havde.

Konklusion 7

Både funktionel og ikke-funktionel kravene er opfyldt.

Alt i alt kan vi drage flere konklusioner:

Simulering af ARM processor:
I vores tilfælde med ARM-simuleringen, tror vi simpelthen ikke at vi vil få det samme resultat hvis vi havde den fysiske ARM-platform, istedet for en simuleret ARM-processor, i et styresystem som kører på en simuleret computer som kører på et styresystem som så kører på en fysisk processor.

Der er simpelthen for mange forskellige faktorer der spiller ind og kan ødelægge ens resultater, så det eneste vi ville bruge simuleringen til, var at se om koden kører som ønsket og giver de ønskede resultater, og så ville vi så bruge en fysisk prototype til at måle hastigheder på.

Valg af filtyper:
Valget af BMP-filtypen er formentlig valgt pga. den meget simple struktur, hvilket gjorde at selve arbejdet med at åbne og gemme filer var hurtigt at implementere, men når først man er oppe i billedestørrelser som moderne kameraer bruger, 10 megapixel og over, så bliver billederne simpelthen så store at man ikke har plads til særlig mange i forhold til JPG-komprimerede filer.

Inkludering af filter på et digitalkamera:
Personligt syntes vi at det er ulogisk at skulle lægge disse filtre på billederne med et digitalkamera. Den energi og tid som kameraet bruger på dette, ville være langt bedre udnyttet til fx `jpeg`-komprimering således at vi har plads til flere billeder på et hukommelseskort.

Desuden er skærmen på et almindeligt digitalkamera også kun i størrelsesordenen 2"til 4"og billederne er oftest i størrelser over 10 megapixels og man kan derfor ikke se om filteret har haft præcis den effekt som man ønskede, og man har ikke nogen backup af det originale billede til at gøre det om.

Derfor er det komplet ulogisk at implementere filtre på et digitalkamera, når man ofrer andre funktioner så som `jpeg`-komprimering.

Overall oplevelse:
Generelt har det været meget lærerigt, specielt for os som ikke har arbejdet med billedebehandling før. Vi indså dog rimelig hurtigt, at dette projekt udelukkende er til at give os en smagsprøve på hvordan indlejrede systemer virker, og ikke er et reelt eksempel på hvorledes en udviklingsafdeling hos fx Canon eller Nikon arbejder med et nyt produkt.

Litteratur

[1] Eric Huss. The c library reference guide. http://www.acm.uiuc.edu/webmonkeys/book/c_guide/.

[2] Victor Podlozhnyuk. *FFT based 2D Convolution*. NVIDIA, 2007. http://developer.download.nvidia.com/compute/cuda/2_2/sdk/website/projects/convolutionFFT2D/doc/convolutionFFT2D.pdf.

[3] Stephen Prata. *C Primer Plus*. SAMS, 2001.

[4] Wikipedia. Bmp file format. http://en.wikipedia.org/wiki/BMP_file_format.

[5] Wikipedia. Convolution. http://en.wikipedia.org/wiki/Convolution.

[6] Wikipedia. Fast fourier transform. http://en.wikipedia.org/wiki/Fast_Fourier_transform.

[7] Wikipedia. Image compression. http://en.wikipedia.org/wiki/Image_compression.

Danmarks Tekniske Universitet

02131 Embedded Systems - Assignment 1

Appendiks

Udarbejdet af:

Afleveret via CampusNet

s081828 - Ibragimov, Arthur

Afleveret via CampusNet

s103473 - Olsen, Anders

DTU Informatik
Institut for Informatik og Matematisk Modellering

02131 Embedded Systems
Vejleder: Paul Pop

Indholdsfortegnelse

Listings

Figurer

Exercise 1 - Kode A

Listing 10: Exercise 1 - HelloWorld.c

```c
#include <stdio.h>

int main(){
  printf("Hello world!");
  return 0;
}
```

Listing 11: Exercise 1 - GCD.c

```c
#include <stdio.h>

int GCD(a, b){
  while (a != b){
    if (a < b){
      b = b-a;
    }
    else{
      a = a-b;
    }
  }
  return a;
}
void GetNumbers(char name[], int *a, int *b){
  FILE *fp;
  if( (fp = fopen(name, "r")) != NULL){
    fscanf(fp, "%d %d", a ,b);
  }
  else {
    printf("Cannot open file.\n");
  }
  fclose(fp);
}
int main(void){
  int userinput;
  int a;
  int b;
  int gcd;

  /*Get a user input to decide if GCD should load from user or from file*/
  puts("Greatest Common Divisor.\n1) Input numbers yourself\n2) Read numbers from file
      gcd_numbers.txt");

  fflush(NULL);
  scanf("%d",&userinput);

  switch (userinput) {
    case 1:
      printf("Write the first number: ");

      fflush(NULL);
      scanf("%d",&a);

      printf("\nWrite the second number: ");

      fflush(NULL);
      scanf("%d",&b);

      break;
    case 2:
      printf("Loading numbers from gcd_numbers.txt...\n");
      GetNumbers("gcd_numbers.txt", &a, &b);
      break;
    default:
```

```
54        break;
55    }
56
57    if (a != 0 && b != 0 ) {
58      gcd = GCD(a,b);
59      printf("The greatest common divisor for %d and %d is %d", a, b, gcd);
60    }
61    return 0;
62 }
```

Listing 12: Indholdet af gcd_numbers.txt

```
1 36 48
```

Exercise 2 - Kode B

Listing 13: Exercise 2 - main.c

```
1  /*
2   * main.c
3   *
4   *  Created on: 07/09/2011
5   *      Author: Anders Olsen
6   */
7  #include <stdio.h>
8  #include "bmp.h"
9
10 int main(){
11   IMAGE bmpimage;
12   char* filename = "example24.bmp";
13   char* savefile = "example24gs.bmp";
14   bmp_open(filename, &bmpimage);
15   bmp_save(savefile, &bmpimage);
16   return 0;
17 }
```

Listing 14: Exercise 2 - bmp.h

```
1
2  /**********************************************************************************
3   * BMP Header
4   **********************************************************************************/
5
6  #ifndef __BMP_H
7  #define __BMP_H
8
9  #include "types.h"
10
11 #define MAX_WIDTH 512
12 #define MAX_HEIGHT  512
13
14 /* data structure for the grayscale image, 1 BYTE / pixel */
15
16 typedef struct {
17   WORD  Height;
18   WORD  Width;
19   BYTE  Pixels[MAX_WIDTH * MAX_HEIGHT];
20 } IMAGE;
21
22 /* open and read a BMP file into image */
23 BOOL bmp_open(char* file, IMAGE* image);
24
25 /* store image to BMP file */
26 BOOL bmp_save(char* file, IMAGE* image);
```

```
27
28 #endif /* __BMP_H */
```

Listing 15: Exercise 2 - bmp.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "bmp.h"
5
6  /* see lecture notes for more information on pragma pack directive */
7  #pragma pack(push, 1)
8
9  /*
10  *   BITMAP FILE: See http://en.wikipedia.org/wiki/BMP_file_format
11  *    ___________________________________________________________
12  *  |           |           |                                   |
13  *  |   file    |   info    |  Palette   |        Pixel data    |
14  *  |  header   |  header   | (optional) |                      |
15  *  |___________|___________|____________|______________________|
16  *  start of file                    end of file
17  *
18  *  - Lines must be word-aligned!
19  *
20  *
21  */
22
23 /**********************************************************************
24  * Bitmap File Header
25  *********************************************************************/
26 typedef struct {
27   HALFWORD  BfType;          /* Must be 0x4D42           */
28   WORD      BfSize;        /* Size of the file in bytes    */
29   HALFWORD  BfReserved1;     /* Should be 0             */
30   HALFWORD  BfReserved2;     /* Should be 0             */
31   WORD      BfOffBits;       /* Offset of image data in file   */
32 } BITMAPFILEHEADER;
33
34 /**********************************************************************
35  * Bitmap Information Header
36  *********************************************************************/
37 typedef struct {
38   WORD      BiSize;        /* Size of this structure       */
39   WORD      BiWidth;       /* Width of the image in bytes    */
40   WORD      BiHeight;      /* Height of the image in bytes   */
41   HALFWORD  BiPlanes;      /* Should be 1             */
42   HALFWORD  BiBitCount;      /* Bit count (..)           */
43   WORD      BiCompression;     /* Compression used         */
44   WORD      BiSizeImage;     /* Size of the image in bytes     */
45   WORD      BiXPelsPerMeter;   /* Pixels per meter, X        */
46   WORD      BiYPelsPerMeter;   /* Pixels per meter, Y        */
47   WORD      BiClrUsed;       /* number of colors used      */
48   WORD      BiClrImportant;    /* number of important colors     */
49 } BITMAPINFOHEADER;
50
51 typedef struct {
52   BYTE rgbBlue;
53   BYTE rgbGreen;
54   BYTE rgbRed;
55   BYTE rgbReserved;
56 }RGBQUAD;
57
58 #pragma pack(pop)
59
60 BITMAPFILEHEADER bmfh;
61 BITMAPINFOHEADER bmih;
62 RGBQUAD aColors[256];
63 BYTE R, G, B;
64
65 BOOL bmp_open(char* file, IMAGE* image) {
```

```
66   int i;
67
68   /* note: "rb" means open for binary read */
69   FILE* fp = fopen(file, "rb");
70   if(fp == NULL) {
71     /* failed to open file, return failure */
72     perror("Could not open file");
73     return FALSE;
74   }
75
76   /*
77    * READ FILE HEADER
78    */
79   fread(&bmfh.BfType, sizeof(bmfh.BfType), 1, fp);
80   printf("Filetype: %#x",bmfh.BfType);
81
82   fread(&bmfh.BfSize, sizeof(bmfh.BfSize), 1, fp);
83   printf("\nFilesize: %d", bmfh.BfSize);
84
85   fread(&bmfh.BfReserved1, sizeof(bmfh.BfReserved1), 1, fp);
86   printf("\nReserved1: %d",bmfh.BfReserved1);
87
88   fread(&bmfh.BfReserved2, sizeof(bmfh.BfReserved2), 1, fp);
89   printf("\nReserved2: %d", bmfh.BfReserved2);
90
91   fread(&bmfh.BfOffBits, sizeof(bmfh.BfOffBits), 1, fp);
92   printf("\nOffbits: %d", bmfh.BfOffBits);
93
94   /*
95    * READ INFO HEADER
96    */
97   fread(&bmih.BiSize, sizeof(bmih.BiSize), 1, fp);
98   printf("\nHeader Size: %d",bmih.BiSize);
99
100  fread(&bmih.BiWidth, sizeof(bmih.BiWidth), 1, fp);
101  printf("\nWidth: %d", bmih.BiWidth);
102
103  fread(&bmih.BiHeight, sizeof(bmih.BiHeight), 1 , fp);
104  printf("\nHeight: %d", bmih.BiHeight);
105
106  fread(&bmih.BiPlanes, sizeof(bmih.BiPlanes), 1, fp);
107  printf("\nPlanes: %d", bmih.BiPlanes);
108
109  fread(&bmih.BiBitCount, sizeof(bmih.BiBitCount), 1, fp);
110  printf("\nBitcount: %d", bmih.BiBitCount);
111
112  fread(&bmih.BiCompression, sizeof(bmih.BiCompression), 1, fp);
113  printf("\nCompression: %d", bmih.BiCompression);
114
115  fread(&bmih.BiSizeImage, sizeof(bmih.BiSizeImage), 1, fp);
116  printf("\nSizeImage: %d", bmih.BiSizeImage);
117
118  fread(&bmih.BiXPelsPerMeter, sizeof(bmih.BiXPelsPerMeter), 1, fp);
119  printf("\nXPelsPerMeter: %d", bmih.BiXPelsPerMeter);
120
121  fread(&bmih.BiYPelsPerMeter, sizeof(bmih.BiYPelsPerMeter), 1, fp);
122  printf("\nYPelsPerMeter: %d", bmih.BiYPelsPerMeter);
123
124  fread(&bmih.BiClrUsed, sizeof(bmih.BiClrUsed), 1, fp);
125  printf("\nClrUsed: %d", bmih.BiClrUsed);
126
127  fread(&bmih.BiClrImportant, sizeof(bmih.BiClrImportant), 1, fp);
128  printf("\nClrImportant: %d", bmih.BiClrImportant);
129
130
131  /*
132   * READ PIXELS:
133   */
134
135  for(i = 0; i < 262144; i++){
```

```
136     fread(&R, 1,1, fp);
137     fread(&G, 1,1, fp);
138     fread(&B, 1,1, fp);
139     image->Pixels[i] = (0.3*R + 0.59*G + 0.11*B);
140 //USED FOR DEBUGGING: printf("R:%d G:%d B:%d\n",R,B,G);
141   }
142
143   image->Height = bmih.BiHeight;
144   image->Width = bmih.BiWidth;
145   /* success */
146   fclose(fp);
147   return TRUE;
148 }
149
150 BOOL bmp_save(char* file, IMAGE* image) {
151   int i;
152
153   /* note: "wb" means open for binary write */
154   FILE* fp = fopen(file, "wb");
155
156   if(fp == NULL) {
157     /* failed to open file, return failure */
158     perror("Could not open file");
159     return FALSE;
160   }
161   //Change bitcount to 8 for greyscale
162   bmih.BiBitCount = 8;
163
164   //Create the grayscale:
165   for (i = 0; i <= 255; i++){
166     aColors[i].rgbBlue = i;
167     aColors[i].rgbGreen = i;
168     aColors[i].rgbRed = i;
169     aColors[i].rgbReserved = 0;
170   }
171
172   //Set the offset so that the palette is not read as a part of the image
173   bmfh.BfOffBits =sizeof(bmfh) + sizeof(bmih) + sizeof(aColors);
174
175   //Write file header:
176   fwrite(&bmfh, sizeof(bmfh), 1, fp);
177
178   //Write info header:
179   fwrite(&bmih, sizeof(bmih), 1, fp);
180
181   //Write color pallette:
182   fwrite(&aColors, sizeof(aColors), 1, fp);
183
184   //Write pixels
185   fwrite(&image->Pixels, sizeof(image->Pixels),1,fp);
186
187   fclose(fp);
188   return TRUE;
189 }
```

Listing 16: Exercise 2 - types.h

```
1  #ifndef __TYPES_H
2  #define __TYPES_H
3
4  #define TRUE    1
5  #define FALSE   0
6
7  typedef unsigned int   BOOL;
8  typedef unsigned char  BYTE;
9  typedef unsigned short HALFWORD;
10 typedef unsigned int   WORD;
11
12 #endif /* __TYPES_H */
```

ARM cosimulation C

Den eneste fil der er ændret ved i forhold til det skelet som vi fik udleveret, er vores `main.c` fil.

Listing 17: ARM - main.c

```
1
2  /*************************************************************************
3   * Test (main)
4   *************************************************************************/
5
6  #include "config.h"
7  #include "ccd.h"
8  #include "lcd.h"
9  #include "driver.h"
10 #include "types.h"
11 #include <stdio.h>
12 #include <time.h>
13
14
15
16 void applyFilter(WORD w, WORD h, int choice);
17 int main(int argc, char *argv[]) {
18   WORD i, w, h;
19    int start, end, answer = 0;
20    double cpu_time_used;
21
22
23
24    /* Print the menu */
25    fflush(stdout);
26    printf("----------\nHere's your choices:\n1) Normal picture, no filter\n2)Blur effect\
          n3)Motionblur effect\n4) Edge detection\nWrite your answer: ");
27    fflush(stdout);
28    /* Read the answer */
29    fflush(stdin);
30    fflush(stdout);
31      scanf("%d",&answer);
32    fflush(stdout);
33    fflush(stdin);
34
35
36    start = clock();
37
38    /* capture image */
39 #ifdef DEBUG_ARM
40    fflush(stdout);
41    printf("ccd_capture_image()\n");
42    fflush(stdout);
43 #endif /* DEBUG_ARM */
44    ccd_capture_image();
45
46    /* reset ccd and lcd pointers */
47 #ifdef DEBUG_ARM
48    fflush(stdout);
49    printf("ccd_reset_pointer()\n");
50    fflush(stdout);
51 #endif /* DEBUG_ARM */
52    ccd_reset_pointer();
53 #ifdef DEBUG_ARM
54    fflush(stdout);
55    printf("lcd_reset_pointer()\n");
56    fflush(stdout);
57 #endif /* DEBUG_ARM */
58    lcd_reset_pointer();
59
```

```
60   /* set width/height of lcd to match captured image */
61 #ifdef DEBUG_ARM
62   fflush(stdout);
63   printf("ccd_get_width()\n");
64   fflush(stdout);
65 #endif /* DEBUG_ARM */
66   w = ccd_get_width();
67 #ifdef DEBUG_ARM
68   fflush(stdout);
69   printf("ccd_get_height()\n");
70   fflush(stdout);
71 #endif /* DEBUG_ARM */
72   h = ccd_get_height();
73
74 #ifdef DEBUG_ARM
75   fflush(stdout);
76   printf("lcd_set_width(%d)\n",w);
77   fflush(stdout);
78 #endif /* DEBUG_ARM */
79   lcd_set_width(w);
80 #ifdef DEBUG_ARM
81   fflush(stdout);
82   printf("lcd_set_height(%d)\n",h);
83   fflush(stdout);
84 #endif /* DEBUG_ARM */
85   lcd_set_height(h);
86
87   /*Interpret the answer */
88   if (answer == 1){
89     fflush(stdout);
90     printf("Not applying a filter\n");
91     fflush(stdout);
92     /* transfer image from ccd to lcd a pixel at a time */
93     for(i = 0; i < w * h; i++) {
94     #ifdef DEBUG_ARM
95     if (i % w == 0) {
96       fflush(stdout);
97       printf("Transfer pixel line %d/%d\n", 1+i/w,h);
98       fflush(stdout);
99     }
100    #endif /* DEBUG_ARM */
101    lcd_set_pixel(ccd_get_pixel());
102    //lcd_set_pixels(ccd_get_pixels());
103    }
104  }
105  else {
106    fflush(stdout);
107    printf("Applying a filter\n");
108    fflush(stdout);
109    applyFilter(w, h, answer);
110  }
111
112
113
114  /* show image on lcd */
115 #ifdef DEBUG_ARM
116   fflush(stdout);
117   printf("lcd_show_image()\n");
118   fflush(stdout);
119 #endif /* DEBUG_ARM */
120   lcd_show_image();
121
122   put(putByte, 0xFF);
123   printf("result %X\n", get(getByte));
124
125 #ifdef DEBUG_ARM
126   fflush(stdout);
127   printf("EXIT ARM PROGRAM\n");
128   fflush(stdout);
129 #endif /* DEBUG_ARM */
```

```
130
131   end = clock();
132
133   fflush(stdout);
134   printf("Start-timestamp: %d\n", start);
135   fflush(stdout);
136
137   fflush(stdout);
138   printf("End-timestamp: %d\n", end);
139   fflush(stdout);
140
141   fflush(stdout);
142   cpu_time_used = (end-start)/ CLOCKS_PER_SEC;
143   printf("Time used: %f\n", cpu_time_used);
144   fflush(stdout);
145
146
147
148   return 0;
149 }
150
151 void applyFilter(WORD w, WORD h, int choice){
152   BYTE pixels[w*h];
153   int row, column, x, y, currentPixel, filterSum, tempPixel, newPixel, i;
154   int m = 3;
155   int (* filter)[3];
156
157   //Blur-filter
158   int filter1[3][3] = {
159       {0,1,0},
160       {1,1,1},
161       {0,1,0}
162   };
163
164   //Motion-Blur-filter
165   int filter2[3][3] = {
166         {1,0,0},
167         {0,1,0},
168         {0,0,1}
169           };
170
171   //Edge-detection-filter
172   int filter3[3][3] = {
173         {-1,-1,-1},
174         {-1,8,-1},
175         {-1,-1,-1}
176           };
177   //Original image
178   int filter4[3][3]={
179         {0,0,0},
180          {0,1,0},
181          {0,0,0}
182                       };
183
184   switch (choice){
185     case 2:
186     filter = filter1;
187     break;
188
189     case 3:
190     filter = filter2;
191     break;
192
193     case 4:
194     filter = filter3;
195     break;
196
197     default:
198     filter = filter4;
199     break;
```

```
200   }
201
202   /* load pixels into BYTE-array */
203   for (i = 0; i < w * h; i++){
204     #ifdef DEBUG_ARM
205     if (i % w == 0) {
206       fflush(stdout);
207       printf("Transfer pixel line %d/%d\n", 1+i/w,h);
208       fflush(stdout);
209     }
210     #endif /* DEBUG_ARM */
211     pixels[i] = ccd_get_pixel();
212   }
213
214
215   printf("\n\n");
216
217   /*int width =w;
218   int height = h;*/
219   printf("Width: %d\n",w);
220   printf("Height: %d\n", h);
221
222   BYTE filteredPixels[w*h];
223
224   filterSum = 0;
225   for (x = 0; x < m; x++){
226     for(y = 0; y < m; y++){
227       filterSum += filter[x][y];
228     }
229   }
230   printf("Filtersum: %d\n", filterSum);
231
232   double normalize;
233   if (filterSum != 0){
234     normalize = 1.0/filterSum;
235   }
236   else {
237     normalize = 1.0;
238   }
239
240   for (row = 1; row < w; row++){ //Going through each row
241     for (column = 1; column < h; column++){ //Going through the column
242       //Apply filter to image->Pixels[r*512+c]
243       currentPixel = pixels[row*w+column];
244       //       printf("\nCurrentPixel: %d \n", currentPixel);
245       tempPixel = 0;
246       for (x = -(m-1)/2; x <= (m-1)/2; x++){
247         for(y = -(m-1)/2; y <= (m-1)/2; y++){
248           //Multiply the filter with the original pixels and summarize:
249           tempPixel += filter[x+1][y+1] * pixels[((row+x)*w) + column+y];
250           //printf("TempPixel: %d ", tempPixel);
251         }
252       }
253       //Normalize the calculation and store the pixel:
254       newPixel = normalize * tempPixel;
255       //       printf("Temporary pixel: %d\n", tempPixel );
256       //       printf("New pixel: %d\n", newPixel );
257       if (newPixel > 0 && newPixel < 255){
258         filteredPixels[row*512+column] = newPixel;
259       }
260       else if (newPixel < 0) {
261         filteredPixels[row*512+column] = 0;
262       }
263       else {
264         filteredPixels[row*512+column] = 0;
265       }
266     }
267   }
268
269   for (i = 0; i < w*h; i++){
```

```
270     pixels[i] = filteredPixels[i];
271   }
272   printf("Normalize value: %f\n", normalize);
273
274   /* send pixels out to LCD */
275   for (i = 0; i < w * h; i++){
276     #ifdef DEBUG_ARM
277     if (i % w == 0) {
278       fflush(stdout);
279       printf("Transfer pixel line %d/%d\n", 1+i/w,h);
280       fflush(stdout);
281     }
282     #endif /* DEBUG_ARM */
283   lcd_set_pixel(pixels[i]);
284   }
285 }
```

Desktop kildekode D

Dette er vores kildekode til kørsel på en almindelig x86 processor. Vi har kun valgt at inkludere de filer som vi har ændret i, i forhold til skelet-filerne på Campusnet, i vores appendix.

Listing 18: Desktop - main.c

```
1
2 /*****************************************************************************
3  * Test (main)
4  *****************************************************************************/
5
6 #include "ccd.h"
7 #include "lcd.h"
8 #include <time.h>
9 #include <stdio.h>
10 #include "types.h"
11 #include <string.h>
12
13 void applyFilter(WORD w, WORD h, int choice);
14 int main(int argc, char *argv[]) {
15   WORD i, w, h;
16   char input[100];
17   char output[100];
18   int start, end, answer = 0;
19   double cpu_time_used;
20
21   fflush(stdout);
22   printf("Please enter the path to read the image from: ");
23   fflush(stdout);
24
25   /* Read the answer */
26   fflush(stdin);
27   fflush(stdout);
28   gets(input);
29   fflush(stdout);
30   fflush(stdin);
31
32
33   fflush(stdout);
34   printf("\nPlease enter the path to save the image to: ");
35   fflush(stdout);
36
37   /* Read the answer */
38   fflush(stdin);
```

```
39    fflush(stdout);
40    gets(output);
41    fflush(stdout);
42    fflush(stdin);
43
44    /* Print the menu */
45      fflush(stdout);
46      printf("\n----------\nHere's your choices:\n1)Normal picture, no filter\n2)Blur
          effect\n3)Motionblur effect\n4)Edge detection\nWrite your answer: ");
47      fflush(stdout);
48
49      /* Read the answer */
50      fflush(stdin);
51      fflush(stdout);
52      scanf("%d",&answer);
53      fflush(stdout);
54      fflush(stdin);
55
56    start = clock();
57
58    /* capture image */
59    if(strlen(input) == 0){
60    ccd_capture_image();
61    } else {
62      ccd_capture_image_custom(input);
63    }
64    /* reset ccd and lcd pointers */
65    ccd_reset_pointer();
66    lcd_reset_pointer();
67
68    /* set width/height of lcd to match captured image */
69    w = ccd_get_width();
70    h = ccd_get_height();
71
72    lcd_set_width(w);
73    lcd_set_height(h);
74
75    //  /* transfer image from ccd to lcd a pixel at a time */
76    //  for(i = 0; i < w * h; i++)
77    //    lcd_set_pixel(ccd_get_pixel());
78
79    /*Interpret the answer */
80    if (answer == 1){
81      printf("Not applying a filter\n");
82      /* transfer image from ccd to lcd a pixel at a time */
83      for(i = 0; i < w * h; i++) {
84        lcd_set_pixel(ccd_get_pixel());
85      }
86    }
87    else {
88      printf("Applying a filter\n");
89      applyFilter(w, h, answer);
90    }
91
92    /* show image on lcd */
93    if(strlen(output) == 0){
94    lcd_show_image();
95    } else{
96      lcd_show_image_custom(output);
97    }
98
99    end = clock();
100
101   printf("Start-clocks: %d\n", start);
102   printf("End-clocks: %d\n", end);
103   cpu_time_used = (end-start)/ CLOCKS_PER_SEC;
104   printf("Time used: %f\n", cpu_time_used);
105
106   return 0;
107 }
```

```
108
109 void applyFilter(WORD w, WORD h, int choice){
110   BYTE pixels[w*h];
111   int row, column, x, y, currentPixel, filterSum, tempPixel, newPixel, i;
112   int m = 3;
113   int (* filter)[3];
114
115   //Blur-filter
116   int filter1[3][3] = {
117       {0,1,0},
118       {1,1,1},
119       {0,1,0}
120   };
121
122   //Motion-Blur-filter
123   int filter2[3][3] = {
124       {1,0,0},
125       {0,1,0},
126       {0,0,1}
127   };
128
129   //Edge-detection-filter
130   int filter3[3][3] = {
131       {-1,-1,-1},
132       {-1,8,-1},
133       {-1,-1,-1}
134   };
135   //Original image
136   int filter4[3][3]={
137       {0,0,0},
138       {0,1,0},
139       {0,0,0}
140   };
141
142   switch (choice){
143   case 2:
144     filter = filter1;
145     break;
146
147   case 3:
148     filter = filter2;
149     break;
150
151   case 4:
152     filter = filter3;
153     break;
154
155   default:
156     filter = filter4;
157     break;
158   }
159
160   /* load pixels into BYTE-array */
161   for (i = 0; i < w * h; i++){
162     pixels[i] = ccd_get_pixel();
163   }
164
165
166   printf("\n\n");
167
168   /*int width =w;
169   int height = h;*/
170   printf("Width: %d\n",w);
171   printf("Height: %d\n", h);
172
173   BYTE filteredPixels[w*h];
174
175   filterSum = 0;
176   for (x = 0; x < m; x++){
177     for(y = 0; y < m; y++){
```

```
178         filterSum += filter[x][y];
179       }
180     }
181     printf("Filtersum: %d\n", filterSum);
182
183     double normalize;
184     if (filterSum != 0){
185       normalize = 1.0/filterSum;
186     }
187     else {
188       normalize = 1.0;
189     }
190
191     for (row = 1; row < w; row++){ //Going through each row
192       for (column = 1; column < h; column++){ //Going through the column
193         //Apply filter to image->Pixels[r*512+c]
194         currentPixel = pixels[row*w+column];
195         //USED FOR DEBUGGING printf("\nCurrentPixel: %d \n", currentPixel);
196         tempPixel = 0;
197         for (x = -(m-1)/2; x <= (m-1)/2; x++){
198           for(y = -(m-1)/2; y <= (m-1)/2; y++){
199             //Multiply the filter with the original pixels and summarize:
200             tempPixel += filter[x+1][y+1] * pixels[((row+x)*w) + column+y];
201             //USED FOR DEBUGGING printf("TempPixel: %d ", tempPixel);
202           }
203         }
204         //Normalize the calculation and store the pixel:
205         newPixel = normalize * tempPixel;
206         //USED FOR DEBUGGING printf("Temporary pixel: %d\n", tempPixel );
207         //USED FOR DEBUGGING printf("New pixel: %d\n", newPixel );
208         if (newPixel > 0 && newPixel < 255){
209           filteredPixels[row*512+column] = newPixel;
210         }
211         else if (newPixel < 0) {
212           filteredPixels[row*512+column] = 0;
213         }
214         else {
215           filteredPixels[row*512+column] = 0;
216         }
217       }
218     }
219
220     for (i = 0; i < w*h; i++){
221       pixels[i] = filteredPixels[i];
222     }
223     printf("Normalize value: %f\n", normalize);
224
225     /* send pixels out to LCD */
226     for (i = 0; i < w * h; i++){
227       lcd_set_pixel(pixels[i]);
228     }
229 }
```

Listing 19: Desktop - bmp.h

```
1
2 /*************************************************************************
3  * BMP Header
4  *************************************************************************/
5
6 #ifndef __BMP_H
7 #define __BMP_H
8
9 #include "types.h"
10
11 #define MAX_WIDTH  512
12 #define MAX_HEIGHT  512
13
14 /* data structure for the grayscale image, 1 BYTE / pixel */
15
```

```
16 typedef struct {
17   WORD  Height;
18   WORD  Width;
19   BYTE  Pixels[MAX_WIDTH * MAX_HEIGHT];
20 } IMAGE;
21
22 /* open and read the fileheader of a BMP file */
23 void bmp_readFileHeader(FILE* file, IMAGE* image);
24 /* open and read the infoheader of a BMP file */
25 void bmp_readInfoHeader(FILE* file, IMAGE* image);
26
27 /* open and read a BMP file into image */
28 BOOL bmp_open(char* file, IMAGE* image);
29
30 /* store image to BMP file */
31 BOOL bmp_save(char* file, IMAGE* image);
32
33 /*Set image to be compressed upon saving */
34 void bmp_setCompressed();
35
36 #endif /* __BMP_H */
```

Listing 20: Desktop - bmp.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "bmp.h"
5
6  /* see lecture notes for more information on pragma pack directive */
7  #pragma pack(push, 1)
8
9  /*
10  *   BITMAP FILE: See http://en.wikipedia.org/wiki/BMP_file_format
11  *    ___________________________________________________________
12  *   |           |           |           |                       |
13  *   |   file    |   info    |  Palette  |       Pixel data      |
14  *   |  header   |  header   | (optional)|                       |
15  *   |___________|___________|___________|_______________________|
16  *  start of file                    end of file
17  *
18  *  - Lines must be word-aligned!
19  *
20  *
21  */
22
23 /**********************************************************************
24  * Bitmap File Header
25  **********************************************************************/
26 typedef struct {
27   HALFWORD  BfType;          /* Must be 0x4D42           */
28   WORD      BfSize;        /* Size of the file in bytes    */
29   HALFWORD  BfReserved1;     /* Should be 0            */
30   HALFWORD  BfReserved2;     /* Should be 0            */
31   WORD      BfOffBits;       /* Offset of image data in file   */
32 } BITMAPFILEHEADER;
33
34 /**********************************************************************
35  * Bitmap Information Header
36  **********************************************************************/
37 typedef struct {
38   WORD      BiSize;        /* Size of this structure       */
39   WORD      BiWidth;       /* Width of the image in bytes    */
40   WORD      BiHeight;      /* Height of the image in bytes   */
41   HALFWORD  BiPlanes;      /* Should be 1            */
42   HALFWORD  BiBitCount;      /* Bit count (..)           */
43   WORD      BiCompression;     /* Compression used         */
44   WORD      BiSizeImage;     /* Size of the image in bytes     */
45   WORD      BiXPelsPerMeter;   /* Pixels per meter, X        */
46   WORD      BiYPelsPerMeter;   /* Pixels per meter, Y        */
```

```
47   WORD      BiClrUsed;       /* number of colors used       */
48   WORD      BiClrImportant;    /* number of important colors      */
49 } BITMAPINFOHEADER;
50
51 /* add here other structs you want to pack */
52 typedef struct {
53   BYTE rgbBlue;
54   BYTE rgbGreen;
55   BYTE rgbRed;
56   BYTE rgbReserved;
57 }RGBQUAD;
58
59 typedef struct{
60   BITMAPINFOHEADER bmiHeader;
61   RGBQUAD bmiColors[1];
62 } BITMAPINFO;
63
64 #pragma pack(pop)
65
66 BITMAPFILEHEADER bmfh;
67 BITMAPINFOHEADER bmih;
68 RGBQUAD aColors[256];
69 BYTE R, G, B;
70
71 void bmp_readFileHeader(FILE* file , IMAGE* image){
72
73   /*
74    * READ FILE HEADER
75    */
76   fread(&bmfh.BfType, sizeof(bmfh.BfType), 1, file);
77   printf("Filetype: %#x",bmfh.BfType);
78
79   fread(&bmfh.BfSize, sizeof(bmfh.BfSize), 1, file);
80   printf("\nFilesize: %d", bmfh.BfSize);
81
82   fread(&bmfh.BfReserved1, sizeof(bmfh.BfReserved1), 1, file);
83   printf("\nReserved1: %d",bmfh.BfReserved1);
84
85   fread(&bmfh.BfReserved2, sizeof(bmfh.BfReserved2), 1, file);
86   printf("\nReserved2: %d", bmfh.BfReserved2);
87
88   fread(&bmfh.BfOffBits, sizeof(bmfh.BfOffBits), 1, file);
89   printf("\nOffbits: %d", bmfh.BfOffBits);
90
91 }
92
93 void bmp_readInfoHeader(FILE* file , IMAGE* image){
94   /*
95    * READ INFO HEADER
96    */
97   fread(&bmih.BiSize, sizeof(bmih.BiSize), 1, file);
98   printf("\nHeader Size: %d",bmih.BiSize);
99
100  fread(&bmih.BiWidth, sizeof(bmih.BiWidth), 1, file);
101  printf("\nWidth: %d", bmih.BiWidth);
102
103  fread(&bmih.BiHeight, sizeof(bmih.BiHeight), 1 , file);
104  printf("\nHeight: %d", bmih.BiHeight);
105
106  fread(&bmih.BiPlanes, sizeof(bmih.BiPlanes), 1, file);
107  printf("\nPlanes: %d", bmih.BiPlanes);
108
109  fread(&bmih.BiBitCount, sizeof(bmih.BiBitCount), 1, file);
110  printf("\nBitcount: %d", bmih.BiBitCount);
111
112  fread(&bmih.BiCompression, sizeof(bmih.BiCompression), 1, file);
113  printf("\nCompression: %d", bmih.BiCompression);
114
115  fread(&bmih.BiSizeImage, sizeof(bmih.BiSizeImage), 1, file);
116  printf("\nSizeImage: %d", bmih.BiSizeImage);
```

```
117
118   fread(&bmih.BiXPelsPerMeter, sizeof(bmih.BiXPelsPerMeter), 1, file);
119   printf("\nXPelsPerMeter: %d", bmih.BiXPelsPerMeter);
120
121   fread(&bmih.BiYPelsPerMeter, sizeof(bmih.BiYPelsPerMeter), 1, file);
122   printf("\nYPelsPerMeter: %d", bmih.BiYPelsPerMeter);
123
124   fread(&bmih.BiClrUsed, sizeof(bmih.BiClrUsed), 1, file);
125   printf("\nClrUsed: %d", bmih.BiClrUsed);
126
127   fread(&bmih.BiClrImportant, sizeof(bmih.BiClrImportant), 1, file);
128   printf("\nClrImportant: %d", bmih.BiClrImportant);
129
130 }
131
132 BOOL bmp_open(char* file, IMAGE* image) {
133   int i;
134   BYTE pixel;
135
136   /* note: "rb" means open for binary read */
137   FILE* fp = fopen(file, "rb");
138   if(fp == NULL) {
139     /* failed to open file, return failure */
140     perror("Could not open file");
141     return FALSE;
142   }
143
144   printf("***************\nReading Picture: %s\n", file);
145   bmp_readFileHeader(fp, image);
146   if(bmfh.BfType != 0x4D42)
147   {
148           perror("The picture is not bmp");
149           return FALSE;
150   }
151   bmp_readInfoHeader(fp, image);
152
153
154   /*
155    * READ PIXELS:
156    * a) Bitcount = 24, then read each pixel and store as greyscale
157    * b) Bitcount = 8, then read each pixel and store directly into pixels-array
158    * c) Bitcount = 8 and Compression = 1, then read two bytes at a time and write
           number of pixels to array.
159    */
160   //a)
161   if(bmih.BiBitCount == 24){
162     for(i = 0; i < 262144; i++){
163       fread(&R, 1,1, fp);
164       fread(&G, 1,1, fp);
165       fread(&B, 1,1, fp);
166       image->Pixels[i] = (0.3*R + 0.59*G + 0.11*B);
167       //printf("R:%d G:%d B:%d\n",R,B,G);
168     }
169   }
170   //b)
171   else if (bmih.BiBitCount == 8 && bmih.BiCompression == 0){
172     //Read palette:
173     fread(&aColors, sizeof(aColors),1,fp);
174     for(i = 0; i < image->Height * image->Width; i++){
175       fread(&pixel, sizeof(pixel), 1, fp);
176       image->Pixels[i] = pixel;
177     }
178   }
179   //c)
180   else if (bmih.BiBitCount == 8 && bmih.BiCompression == 1){
181     BYTE B1 = 0;
182     BYTE B2 = 0;
183     int x = 0;
184
185     bmih.BiCompression = 0;
```

```
186     bmih.BiSize = 0;
187
188     fread(&aColors, sizeof(aColors),1,fp);
189     //While-loop checks if the two values B1 and B2 are not 00 and 01
190     BOOL running = 1;
191     printf("Reading the pixels from the compressed image:\n");
192     while(running){
193       fread(&B1, sizeof(BYTE), 1, fp);
194       fread(&B2, sizeof(BYTE), 1, fp);
195       printf("B1: %d B2: %d\n",B1,B2);
196
197       if (B1 != 00){
198         for (i = 0; i < B1; i++){
199           image->Pixels[x] = B2;
200           x++;
201           //          printf("x = %d ", x);
202         }
203       }
204       else if(B1 == 0 && B2 == 1){
205         printf("Done running");
206         running = 0;
207       }
208     }
209   }
210   image->Height = bmih.BiHeight;
211   image->Width = bmih.BiWidth;
212   /* success */
213   printf("\n********************\n");
214   fclose(fp);
215   return TRUE;
216 }
217
218 void bmp_setCompressed(){
219   bmih.BiCompression = 1;
220 }
221 BOOL bmp_save(char* file, IMAGE* image) {
222   int i;
223
224   /* note: "wb" means open for binary write */
225   FILE* fp = fopen(file, "wb");
226
227   if(fp == NULL) {
228     /* failed to open file, return failure */
229     perror("Could not open file");
230     return FALSE;
231   }
232
233   /*
234    * CONVERT TO GREYSCALE:
235    * a) BitCount = 24, then convert to greyscale
236    * b) BitCount = 8, then just save the pixels
237    */
238   //a)
239   if (bmih.BiBitCount == 24){
240     //Change bitcount to 8 for greyscale
241     bmih.BiBitCount = 8;
242
243     //Create the grayscale:
244     for (i = 0; i <= 255; i++){
245       aColors[i].rgbBlue = i;
246       aColors[i].rgbGreen = i;
247       aColors[i].rgbRed = i;
248       aColors[i].rgbReserved = 0;
249     }
250     printf("Converting image to greyscale.\n");
251     bmfh.BfOffBits =sizeof(bmfh) + sizeof(bmih) + sizeof(aColors);
252   }
253
254
255   //Write file header:
```

```
256   printf("Writing the file header\n");
257   fwrite(&bmfh, sizeof(bmfh), 1, fp);
258
259   //Write info header:
260   printf("Writing info header\n");
261   fwrite(&bmih, sizeof(bmih), 1, fp);
262
263   //Write color pallette:
264   printf("Writing color palette\n");
265   fwrite(&aColors, sizeof(aColors), 1, fp);
266
267   /*
268    * //Write pixels
269    * a) BiCompression = 0, then just write the pixels
270    * b) BiCompression = 1, then loop and write RLE8-compressed pixel-data
271    */
272   if (bmih.BiCompression == 0){
273     printf("Writing the pixel-data without compression.\n");
274     fwrite(&image->Pixels, sizeof(image->Pixels),1,fp);
275   }
276   else if (bmih.BiCompression == 1){
277     printf("Writing the pixel-data with RLE8 compression");
278     BYTE B1 = 0, B2 = 0;
279     BYTE numberOfBytes = 0;
280     BYTE zeroIndicator = 0;
281     BYTE pixelCounter = 0;
282
283
284
285     /*
286      * LOOP THROUGH ALL PIXELS
287      */
288     for(i = 0; i < image->Height * image->Width; i++){
289
290       /*
291        * IF THE NEXT PIXEL IS EQUAL TO THE PREVIOUS ONE, INCREMENT B1 BY 1:
292        */
293       if (B2 == image->Pixels[i]){
294
295         B1++;
296
297         /*
298          * IF B1 HAS REACHED MAX VALUE THEN WE WRITE B1 AND B2 AND RESET B1:
299          */
300         if(B1 == 255){
301           fwrite(&B1, sizeof(B1),1,fp);
302           fwrite(&B2, sizeof(B2),1,fp);
303           numberOfBytes = numberOfBytes + 2;
304           B1 = B1-255;
305         }
306       }
307
308
309       /*
310        * IF THE NEXT PIXEL DOES NOT MATCH THE PREVIOUS THEN WE WRITE B1 AND B2, SAVE
              THE NEW PIXEL AS B2 AND RESET B1:
311        */
312       else {
313         /*
314          * ONLY WRITE IF B1 IS NOT 0
315          */
316         //Only do it if B1 is not 0
317         if(B1 != 0){
318           fwrite(&B1, sizeof(B1), 1, fp);
319           fwrite(&B2, sizeof(B2), 1, fp);
320           numberOfBytes = numberOfBytes  +2;
321         }
322         B1 = 1;
323         B2 = image->Pixels[i];
324       }
```

```
325
326         /*
327          * KEEP TRACK OF THE LINE OF PIXELS
328          */
329         pixelCounter++;
330
331         /*
332          * IF WE HAVE REACHED A NEW LINE, WE WRITE THE BREAKLINE-COMBO AND RESET THE
                 PIXEL COUNTER
333          */
334         if (pixelCounter == 512){
335           fwrite(&zeroIndicator, sizeof(zeroIndicator),1,fp);
336           fwrite(&zeroIndicator, sizeof(zeroIndicator),1,fp);
337           numberOfBytes = numberOfBytes + 2;
338           pixelCounter = 0;
339         }
340       }
341
342       //At the end of the pixels, write the EOF:
343       B1 = 0; B2 = 1;
344       fwrite(&B1, sizeof(B1), 1, fp);
345       fwrite(&B2, sizeof(B2), 1, fp);
346       numberOfBytes = numberOfBytes +2;
347
348       //Change the SizeImage value and re-write the infoheader:
349       bmih.BiSize = sizeof(bmih);
350       bmih.BiSizeImage = numberOfBytes;
351       fseek(fp, sizeof(bmfh), SEEK_SET);
352       fwrite(&bmih, sizeof(bmih),1,fp);
353     }
354     fclose(fp);
355     return TRUE;
356 }
```

Listing 21: Desktop - ccd.h

```
1
2 /*****************************************************************************
3  * CCD Header
4  *****************************************************************************/
5
6 #ifndef __CCD_H
7 #define __CCD_H
8
9 #include "types.h"
10
11 /* get the width of the next image to show */
12 WORD ccd_get_width();
13
14 /* get the width of the next image to show */
15 WORD ccd_get_height();
16
17 /* capture an image */
18 void ccd_capture_image();
19
20 /* capture an image (with a custom path) */
21 void ccd_capture_image_custom(char* path);
22
23 /* reset internal pixel pointer */
24 void ccd_reset_pointer();
25
26 /* get one pixel of the captured image */
27 BYTE ccd_get_pixel();
28
29 /* get four pixels of the captured image */
30 WORD ccd_get_pixels();
31
32 #endif /* __CCD_H */
```

Listing 22: Desktop - ccd.c

```
1
2  /*************************************************************************
3   * CCD
4   *************************************************************************/
5
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  #include "bmp.h"
10
11 static IMAGE current_image;
12
13 WORD ccd_pixel_pointer;
14
15 WORD ccd_get_height() {
16   return current_image.Height;
17 }
18
19 WORD ccd_get_width() {
20   return current_image.Width;
21 }
22
23 void ccd_capture_image() {
24   if(!bmp_open("example24.bmp", &current_image)) {
25     printf("ccd_capture_image(): failed to open file\n");
26     exit(0);
27   }
28 }
29
30 void ccd_capture_image_custom(char* path) {
31   if(!bmp_open(path, &current_image)) {
32     printf("ccd_capture_image(): failed to open file\n");
33     exit(0);
34   }
35 }
36
37 void ccd_reset_pointer() {
38   /* reset */
39   ccd_pixel_pointer = 0;
40 }
41
42 BYTE ccd_get_pixel() {
43   return current_image.Pixels[ccd_pixel_pointer++];
44 }
45
46 WORD ccd_get_pixels() {
47
48   /* update pixel pointer */
49   ccd_pixel_pointer += 4;
50
51   return 0;
52 }
```

Listing 23: Desktop - lcd.h

```
1
2  /*************************************************************************
3   * LCD Header
4   *************************************************************************/
5
6  #ifndef __LCD_H
7  #define __LCD_H
8
9  #include "types.h"
10
11 /* set the width of the next image to show */
12 void lcd_set_width(WORD width);
13
```

```
14 /* set the height of the next image to show */
15 void lcd_set_height(WORD height);
16
17 /* show the image */
18 void lcd_show_image();
19
20 /* show the image (and save it with a custom filename) */
21 void lcd_show_image_custom(char* path);
22 /* reset the internal pixel pointer */
23 void lcd_reset_pointer();
24
25 /* set the next four pixels of the image */
26 void lcd_set_pixel(BYTE pixel);
27
28 /* set the next four pixels of the image */
29 void lcd_set_pixels(WORD pixels);
30
31 #endif /* __LCD_H */
```

Listing 24: Desktop - lcd.c

```
1
2 /*****************************************************************************
3  * LCD
4  *****************************************************************************/
5
6 #include <stdio.h>
7 #include <stdlib.h>
8
9 #include "lcd.h"
10 #include "bmp.h"
11
12 /* image representation compabible with bmp library */
13 static IMAGE image;
14
15 /* next pixel to write */
16 static WORD lcd_pixel_pointer;
17
18 void lcd_set_width(WORD width) {
19   /* set desired image width */
20   image.Width = width;
21 }
22
23 void lcd_set_height(WORD height) {
24   /* set desired image height */
25   image.Height = height;
26 }
27
28 void lcd_show_image() {
29   if(!bmp_save("output.bmp", &image)) {
30     printf("lcd_show_image(): failed to save file\n");
31   }
32 }
33
34 void lcd_show_image_custom(char* path ) {
35   if(!bmp_save(path, &image)) {
36     printf("lcd_show_image(): failed to save file\n");
37   }
38 }
39
40 void lcd_reset_pointer() {
41
42   /* reset, ready for new image */
43   lcd_pixel_pointer = 0;
44 }
45
46 void lcd_set_pixel(BYTE pixel) {
47   image.Pixels[lcd_pixel_pointer++] = pixel;
48 }
49
```

```
50 void lcd_set_pixels(WORD pixels) {
51
52   /* todo: store (four) pixels */
53
54   /* update pixel pointer */
55   lcd_pixel_pointer += 4;
56 }
```