

# Bilag P: Display til iPot

Anette Lihn

1. juni 2023



# Indhold

<b>1</b>	<b>Indledning</b>	<b>1</b>
<b>2</b>	<b>Design</b>	<b>1</b>
2.1	Valg af display . . . . .	1
2.1.1	Farve og sort/hvid . . . . .	1
2.1.2	Skærmstørrelse . . . . .	1
2.1.3	Kommunikationsprotokol . . . . .	2
2.1.4	Support . . . . .	2
2.2	Hardware . . . . .	2
2.2.1	Kredsløb . . . . .	2
2.3	Software . . . . .	3
2.3.1	Device Driver . . . . .	4
2.3.2	SSD1331 klasse . . . . .	4
2.3.2.1	SSD1331 konstanter . . . . .	6
2.3.3	Images klasse . . . . .	7
2.3.3.1	Bitmaps . . . . .	7
2.3.3.2	RGB farver . . . . .	7
2.3.3.3	Skærbilleder . . . . .	8
2.3.3.4	Tekstfunktionalitet . . . . .	9
2.3.4	Display klasse . . . . .	10
<b>3</b>	<b>Implementering</b>	<b>11</b>
3.1	Hardware . . . . .	11
3.2	Software . . . . .	11
3.2.1	Device Driver . . . . .	11
3.2.2	SSD1331 klasse . . . . .	12
3.2.3	Images klasse . . . . .	16
3.2.4	Display klasse . . . . .	22
<b>4</b>	<b>Modultest</b>	<b>23</b>
4.1	Device Driver og SSD1331 klasse . . . . .	24
4.2	Images klasse og Display klasse . . . . .	26
	<b>Litteraturliste</b>	<b>29</b>

# 1 Indledning

I dette dokument er alt udviklingsarbejdet for brugergrænsefladens display dokumenteret. Som det første beskrives valget af display ved at redegøre for de muligheder, der har været overvejet, samt fordelen ved at vælge et display frem for et andet. Dernæst redegøres der for de designvalg, der har lagt til grund for implementationen, hvorefter implementationen beskrives. Til sidst vises, hvordan de enkelte moduler er blevet testet, og af disse tests fremgår det ligeledes, i hvilken rækkefølge de enkelte moduler er blevet integreret i displayens samlede softwarepakke.

# 2 Design

## 2.1 Valg af display

Som display til brugergrænsefladen er OLED Board 16-bit Color 0.96" fra adafruit valgt. I dette afsnit listes først de tilgængelige teknologier, og derefter beskrives overvejelserne, som ligger til grund for valget af førnævnte display. Som ultimative krav til displayet er, at det skal kunne anvendes med Raspberry Pi, da denne er valgt som microcontroller til brugergrænsefladen, samt at displayet er tilgængeligt i embedded stock. Ud fra disse to ultimative krav samt overvejelserne i teknologianalysen, er følgende fire displays en mulighed:

- OLED Breakout Board - 16-bit Color 0.96" w/microSD holder (fra adafruit)
- Monochrome 0.96" 128x64 OLED Graphic Display (fra adafruit)
- 2.2" 18-bit color TFT LCD display with microSD card breakout (fra adafruit)
- Alphanumeric LCD Display - 2 x 24, I2C

### 2.1.1 Farve og sort/hvid

Fordelen ved at vælge et display med farve, er, at det giver mulighed for at skabe nogle mere tydelige billeder, som kan appellere mere til brugeren. Fordelen ved et display med sort/hvid er, at teknologien er mere simpel at programmere, og der findes bedre muligheder for at generere hurtige bitmaps. Et display med farve blev valgt, da tydelighed for brugeren vægtes højt og det trods alt er en begrænset mængde bitmaps, der skal produceres.

### 2.1.2 Skærmstørrelse

Fordelen ved et større display er, at der kan præsenteres mere indhold, mens fordelelens ved et mindre display er, at kodningen af diverse billeder tager mindre tid. Det vurderes, at der kan præsenteres det nødvendige indhold på et 96\*64 pixel display. Skulle ønsket om en større skærm opstå, kan der i det senere arbejde med brugergrænsefladen evt. opskaleres til 2.2" LCD displayet, da de to farvedisplays anvender samme kommunikationsprotokol.

### 2.1.3 Kommunikationsprotokol

Der er ikke en umiddelbar fordel ved at vælge SPI frem for I2C eller omvendt. Begge disse protokoller er inden for teknisk kunnen, og det er derfor ikke denne parameter, der er blevet afgørende for valg af display.

### 2.1.4 Support

Muligheden for support har også været en overvejelse i valget af display. Her er adafruit displays et godt valg, da adafruit anvender drivere med tilhørende datablad af tilfredsstillende kvalitet samt selv producerer meget supportmateriale til deres produkter. En ulempe ved adafruit er dog, at de overvejende laver support til Python, og derfor skal der kodes en del mere fra bunden, når der anvendes C++.

## 2.2 Hardware

Displayet anvender en SSD1331 driver, som netop virker godt til displays op til 96x64 pixels. SSD1331 har indbygget GDDRAM (Graphic Display Data RAM) og understøtter SPI interface. Displayet har 10 pins, som kan anvendes til at forbinde displayet til microcontrolleren. Der er VCC og ground til at forsyne displayet med strøm, MISO, MOSI, clock og slave select til SPI kommunikationen, reset til at nulstille displayet, D/C til at vælge mellem data eller kommandooverførsel samt SDCS pin og SD pin til anvendelse med SD kort.[2] MISO anvendes ikke, da der kun sendes fra Master til Slave. SDCS og SD pin anvendes ikke, da der ikke anvendes SD kort til brugergrænsefladen. SD kortet er fravalgt, da en del af skærmbillederne løbende skal opdateres med aktuelle værdier, og SD-kort funktionaliteten får strømforbruget til at stige markant.

### 2.2.1 Kredsløb

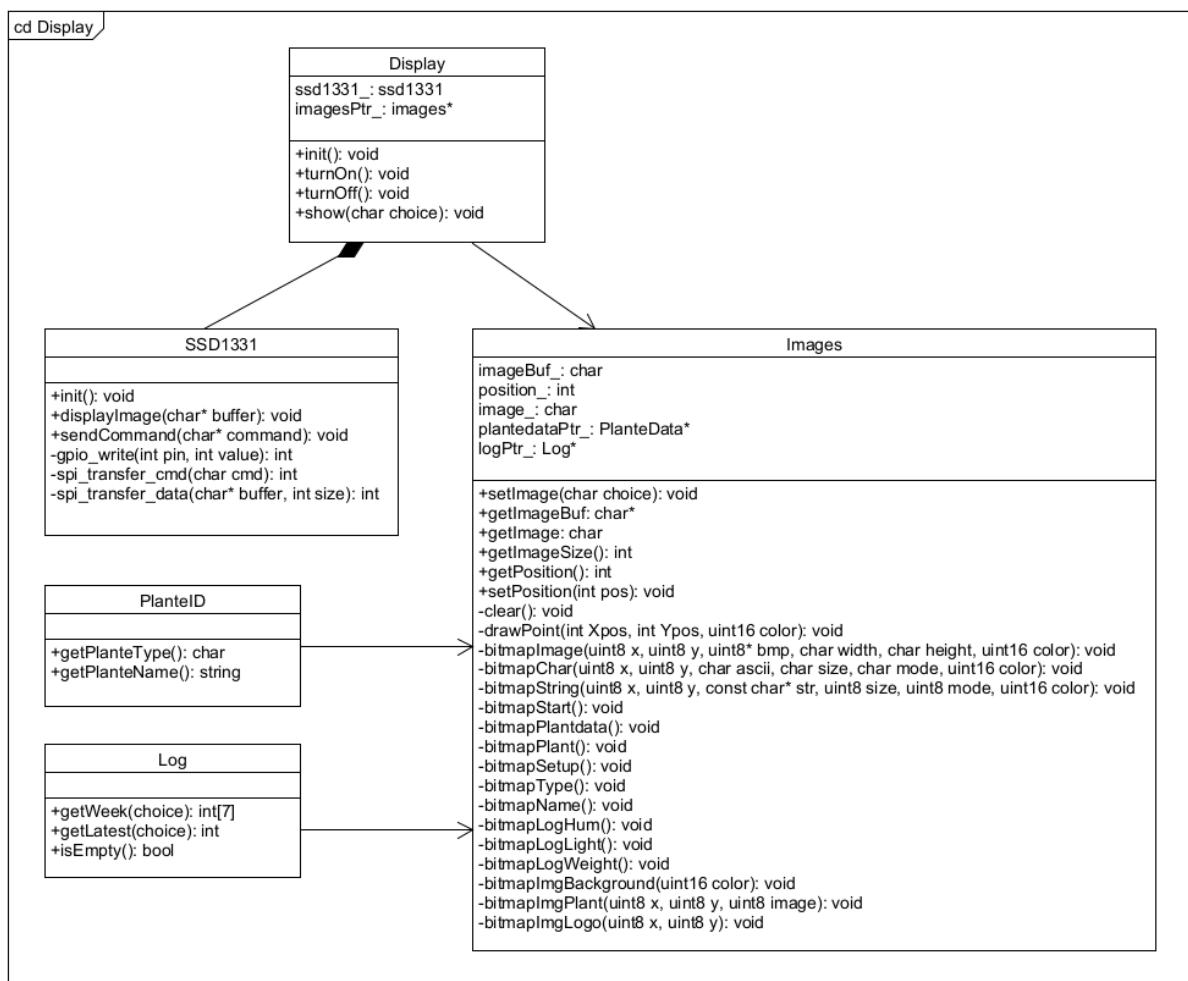
Displayet kan forbindes direkte til Raspberry Pi og disse forbindelser er vist i tabel 1. Da displayet kan være følsomt over for støj fra microcontrollernes 5V forsyninger, forbindes VCC på displayet til 3,3 V forsyningen på Raspberry Pi i og med denne er mindre påvirket af støj. Til SPI kommunikation anvendes SPI0, som har MOSI på GPIO10, SCK på GPIO11 og CE0 (select pin) på GPIO8.[3] Reset og D/C forbindes til hhv. GPIO 19 og GPIO 13, men disse pins kan ændres, hvis der er behov for det.

Display	Raspberry Pi
VCC (V+)	gpio (3.3 V)
Ground (G)	gpio (GND)
SCK (CK)	gpio11 (SPI_SCLK)
MOSI (SI)	gpio10 (SPI_MOSI)
OCS (CS)	gpio8 (SPI_CE0)
RST (R)	gpio19
D/C (DC)	gpio13

Tabel 1: Forbindelser mellem Display og Raspberry Pi

## 2.3 Software

I designet af softwaren til displayet er der taget udgangspunkt i den overordnede softwarearkitektur. Designet udvides ved at tilføje flere klasser og metoder for at lave en tydeligere opdeling af funktionalitet. Klassen SSD1331, som håndterer kommunikationen til SSD1331 driveren, implementeres med komposition, da displayet *har* en SSD1331 driver. Der designes en device driver, som skal fungere i samspil med SSD1331 klassen. Display klassen *bruger* de skærmbilleder, som Images klassen laver, og derfor implementeres dette med association. Images klassen *anvender* både data fra Log og PlanteID klasserne, og disse implementeres ligeledes med association. Det udvidede design er vist på figur 1. Der vises kun de metoder for Log og PlanteID, som anvendes af Images klassen, da der redegøres for designet af begge klasser i anden dokumentation.



Figur 1: Klassediagram for Display

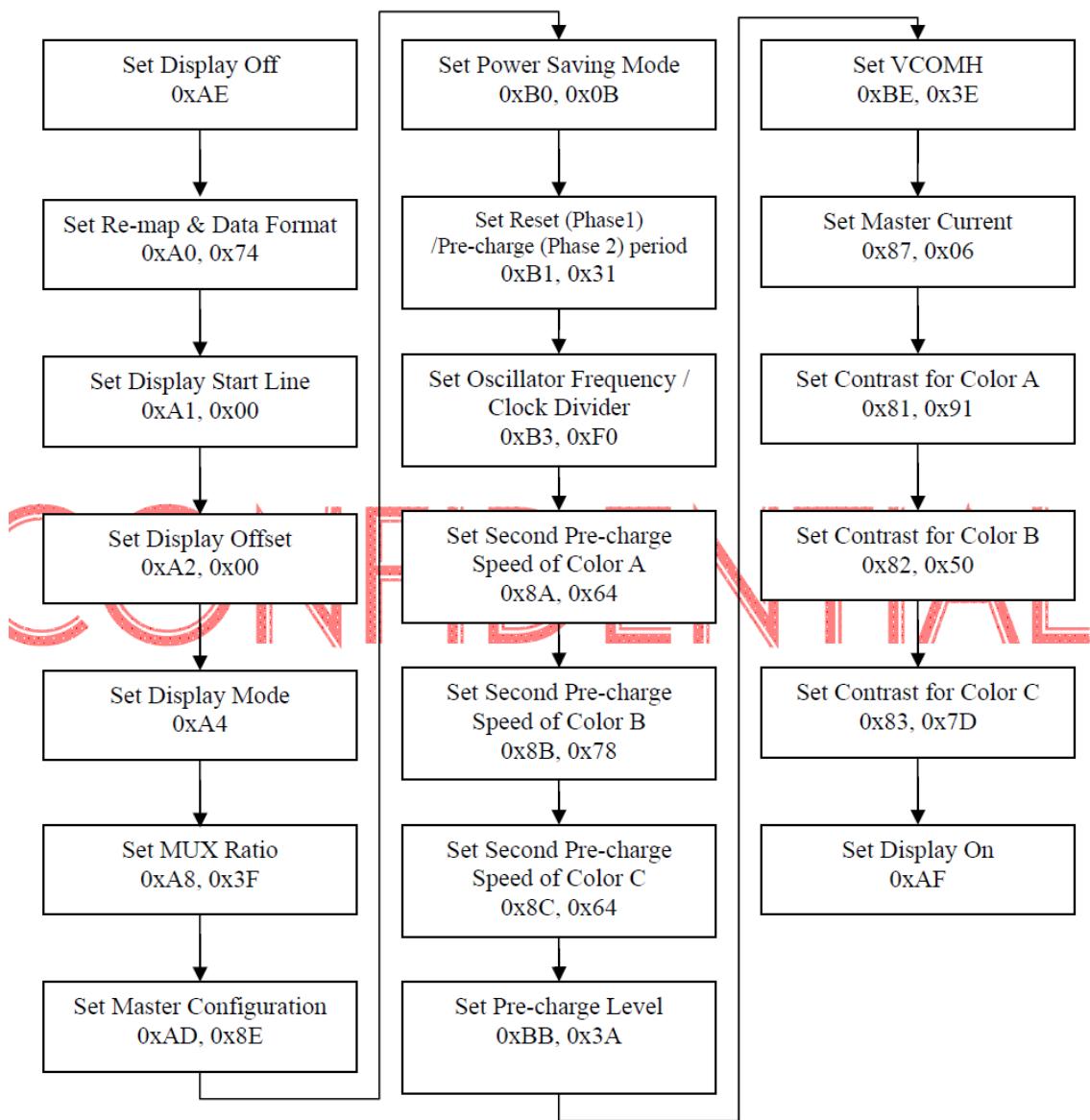
### **2.3.1 Device Driver**

For at kunne kommunikere med displayet skal der først implementeres en device driver. En alternativ mulighed er at anvende et bibliotek, men i og med det vil betyde mindre kontrol over implementationen, fravælges denne løsning. Device driveren skal oprette tre devices, SPI, D/C OG RST, og der skal implementeres en write funktion, der håndterer kommunikationen med hver af disse tre devices. Write implementationen skal tage højde for, at der skal implementeres SPI write, der både kan håndtere at sende en kommando (8 bit) og et data array (12.288 bit).

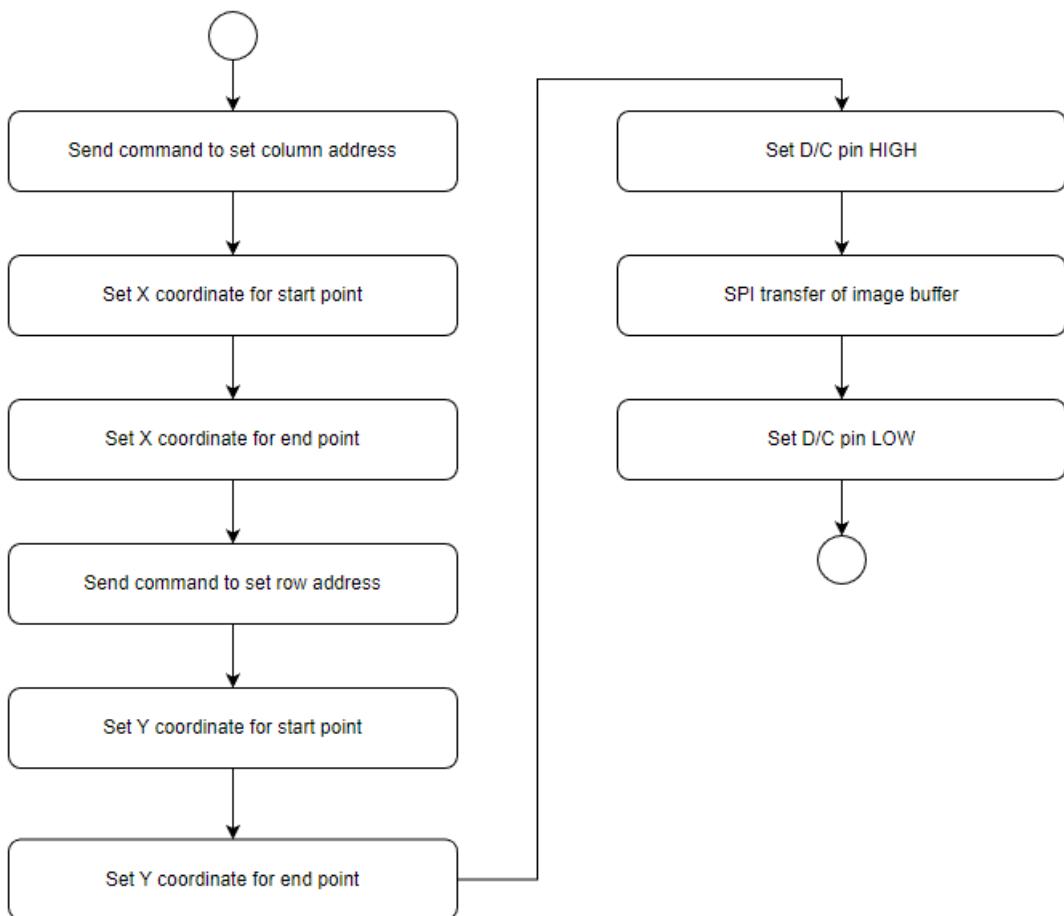
### **2.3.2 SSD1331 klasse**

SSD1331 klassen håndterer som bekendt kommunikationen til SSD1331 driveren. I databladet for displayet kan vi se, at D/C pin er en kontrolpin, hvor D står for data og C for command. Når den sættes høj, vil den overførte data blive skrevet til Graphic Display Data RAM (GDDRAM), og når den sættes lav, vil den overførte data blive læst som en kommando og skrevet til det tilsvarende kommandoregister. Displayets Reset pin sættes lav, når displayets chip skal initialiseres, og sættes høj ved almindeligt brug af skærmen. Al data overføres med SPI.[2]

SSD1331 klassen indeholder de metoder, der er vist på figur 1. Public metoderne, som består af init(), displayImage() og sendCommand(), anvendes af Display klassen. Private metoderne, der skal implementeres således, at de anvender device driveren, består af gpio\_write(), spi\_transfer\_cmd() og spi\_transfer\_data(). Disse metoder anvendes af public metoderne til at lave hhv. SPI og gpio output. De implementeres ved at skrive direkte til device filerne. Metoden init() skal initialisere displayet, så det er indstillet korrekt. Dette gøres ved først at lave automatisk initialisering ved at sætte Reset pin lav for derefter at lave en brugerdefineret initialisering af diverse indstillinger ud fra det flowchart, der er vist på figur 2. Metoden displayImage() skal printe hele skærbilledet. Flowcharten på figur 3 viser, hvordan vi først skal sende en række kommandoer, der angiver det område der skal tegnes inden for. I vores tilfælde definerer vi dette område som hele skærmen, og vi starter i koordinaterne (0,0) og slutter i koordinaterne (95,63). Derefter sættes D/C høj, da der skal sendes data, hvorefter den pågældende data overføres. Til sidst sættes D/C lav for at sikre, at eventuel støj ikke efterfølgende medfører pixel fejl på skærmen. Metoden sendCommand() implementeres ved at sætte D/C lav og derefter sende den angivne 8-bit kommando.



Figur 2: Flowchart for initialisering af display[5]



Figur 3: Flowchart for displayImage()

#### 2.3.2.1 SSD1331 konstanter

I databladet fremgår det, at der er værdier, som repræsenterer bestemte kommandoer. I designet oprettes derfor en fil, ssd1331\_constants.h, som indeholder de konstanter, der er defineret i databladet samt eventuelle andre konstanter som fx skærmstørrelse, valgte pins mm.

### 2.3.3 Images klasse

Images klassens formål er som nævnt tidligere at håndtere de skærbilleder, som skal vises på displayet. Ved at anvende public metoderne setImage(), getImageBuf(), getImage() og getImageSize() kan brugeren af klassen lave og hente skærbillederne. Metoden setImage() står for at lave skærbilledet ved at kalde den hjælpemetode, der tilsvarer det ønskede skærbillede. Metoden getImage() skal returnere en char værdi, der indikerer hvilket skærbillede der er gemt i bufferen. Metoden getImageBuf() skal returnere en pointer til bufferen, og metoden getImageSize() skal returnere størrelsen på bufferen. Derudover laves der yderligere to public metoder, getPosition() og setPosition(), som er knyttet til attributen position\_, der i det samlede system skal bruges i samspil med knapperne til at holde styr på, hvor brugeren befinner sig på skærmen. Attributen position\_ har en varierende maksimal værdi afhængigt af det aktuelle skærbillede, så der skal laves valideringen af værdierne for position\_ i setPosition().

#### 2.3.3.1 Bitmaps

Til at lave skærbillederne anvendes bitmaps. Et bitmap er et array, der definerer tilstanden for hvert enkelt pixel i et billede. Da displayet kan vise farve, vil hvert pixel skulle kodes med en farvekode. Dette display anvender en farvekode på 16 bit, og bufferen skal derfor have størrelsen  $(2 * 96 * 64) = 12.288$ . Images inkluderer filen "bitmaps.h", hvis funktion er at henvise til diverse h-filer med bitmaps. Hvert bitmap kan enten være en fuld skærmfigur eller mindre figurer, som i det endelige bitmap kan placeres på et valgfrit sted. Da vi gerne vil lave polykromatiske bitmaps, skal der oprettes et separat array til at håndtere hvert enkel farve, som efterfølgende kan samles. Disse array består af et bestemt antal 8-bit tal, hvor hvert bit svarer til en pixel. 1 betyder at der tegnes en prik i den pågældende farve, mens 0 betyder, at baggrundsfarven bliver. Som default baggrundsfarve på displayet anvendes sort, da denne farve bruger mindst energi, fordi et OLED display ikke bruger et generelt baggrundslys, men derimod lyser hvert enkelt pixel op.[4] Metoden drawPoint() bruges til at gemme 16-bit RGB værdien i bufferen for et enkelt punkt.

#### 2.3.3.2 RGB farver

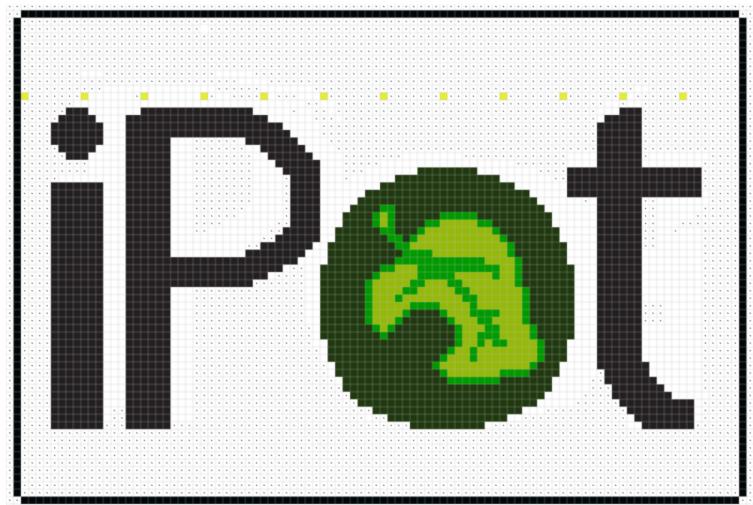
Farverne programmeres som 16-bit RGB, og som man kan se på figur 4, anvendes der 5 bit til rød, 5 bit til blå og 6 bit til grøn. De RGB farvekoder, som kan findes på nettet, vil typisk være 24-bit RGB koder, og derfor vil disse ikke kunne bruges helt ukritisk til displayet, da der mistes 3 bit for rød, 2 bit for grøn og 3 bit for blå. Farverne gemmes som konstanter i h-filen "images\_constants.h".

Rød					Grøn					Blå				
14				11	10				5	4				0

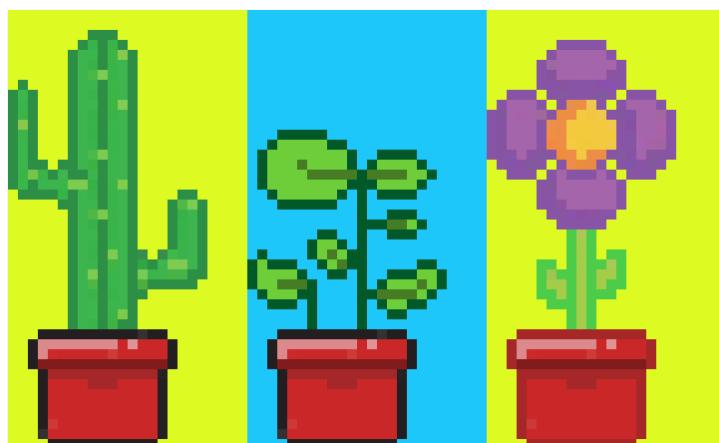
Figur 4: Design af 3x5 font

### 2.3.3.3 Skærbilleder

For at skabe et mere personligt forhold for brugeren til planten designes der plantebilleder til displayet, og disse kan ses på figur 6. Der designes også et logo, der vises hver gang skærmen tændes, og dette kan ses på figur 5. Metoden `bitmapImage()` bruges til at tegne billederne af skærmen.



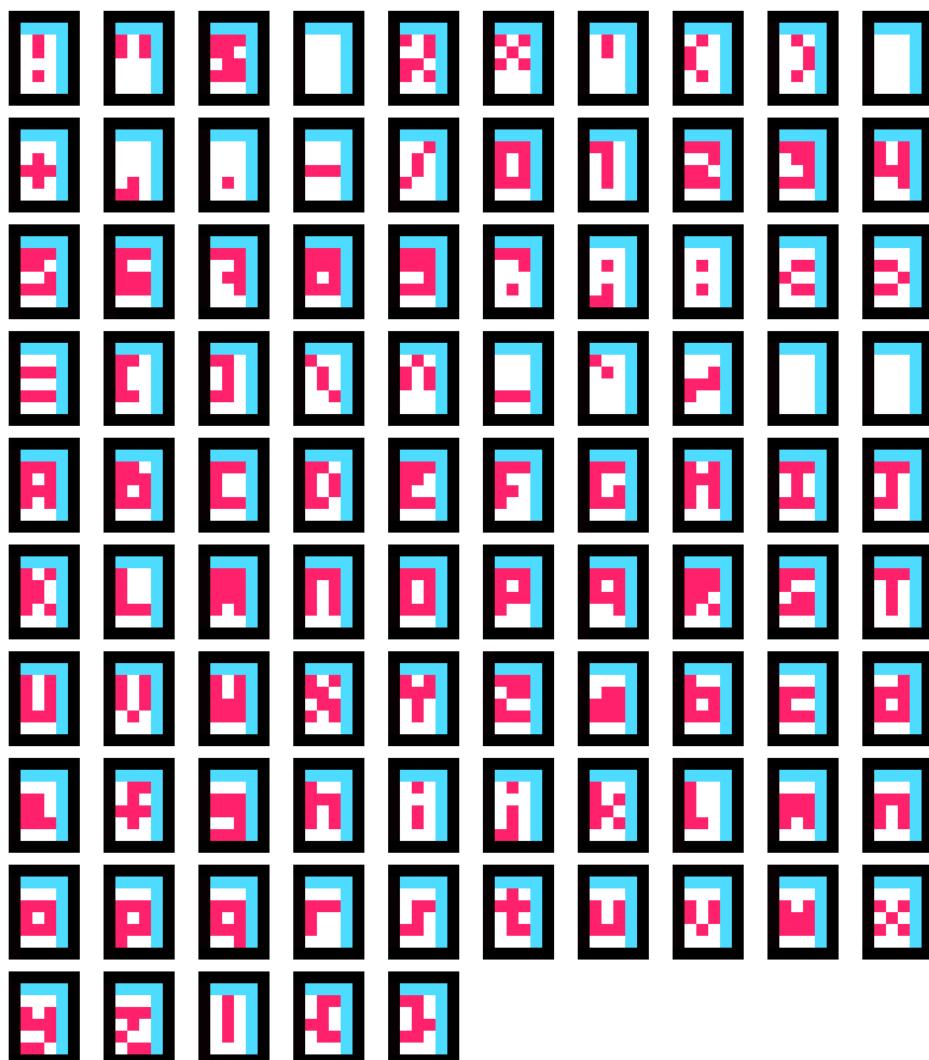
Figur 5: Design af iPot logo



Figur 6: Design af plantebilleder

#### 2.3.3.4 Tekstfunktionalitet

Der skal skrives en del tekst til displayet, og denne tekst ændrer sig, efterhånden som værdierne for lys, vægt, fugtighed og vandstand ændrer sig. Der designes derfor to metoder, bitmapString() og bitmapChar(), som printer henholdsvis tekststrenge og enkelte characters til displayet. Metoden bitmapString() skal kunne tage en hel streng som parameter og printe denne ved at bruge bitmapChar(), som tager en enkelt character som parameter og printer denne. Metoden bitmapChar() skal implementeres med og uden invertering, da der ved invertering vil blive vist tekst som sort i en farvet bjælke. Dette kan bruges til at vise, når brugeren står på en valgmulighed. Metoderne designes ydermere til at kunne skrive med to fontstørrelser på henholdsvis 3x5 pixel og 5x7 pixel. Designet af disse er vist på figur 7 og figur 8. 3x5 pixel har den fordel, at der kan vises mere tekst på displayet, som for eksempel er et behov ved visning af log, mens 5x7 har den fordel, at den fremgår mere tydelig og kan vise nogle af de mere særprægede tegn.



Figur 7: Design af 3x5 font



Figur 8: Design af 5x7

#### 2.3.4 Display klasse

Display klassen designes meget simpelt og har derfor ingen private metoder, da funktionaliteten for displayet i forvejen er delt op ved at anvende de andre klasser. Dermed vil man kunne lave ændringer i de andre klasser uden det har betydning for, hvordan displayet implementeres i et program. Display klassen har metoderne init(), turnOn(), turnOff() og show(). Metoden init() initialiserer skærmens, turnOn() og turnOff() henholdvis tænder og slukker for skærmens og show() viser det valgte billede på skærmens.

## 3 Implementering

### 3.1 Hardware

Hardwaren implementeres som beskrevet i designafsnittet. Ledningsforbindelserne farvekodes som angivet i tabel 2, så det er nemmere at forbinde de to enheder korrekt. Den fysiske forbindelse mellem de to enheder er tilsvarende til den, der er brugt ved modultest.

Display	Raspberry Pi	Ledningsfarve
VCC (V+)	gpio (3.3 V)	Rød
Ground (G)	gpio (GND)	Sort
SCK (CK)	gpio11 (SPI_SCLK)	Grøn
MOSI (SI)	gpio10 (SPI_MOSI)	Gul
OCS (CS)	gpio8 (SPI_CE0)	Lilla
RST (R)	gpio19	Orange
D/C (DC)	gpio13	Hvid

Tabel 2: Farvekoder for forbindelser mellem Display og Raspberry Pi

### 3.2 Software

I dette afsnit vises og begrundes dele af implementationen for softwaren. Der anvendes udsnit af sourcekoden, hvor kodekommentarerne er fjernet, da det således får en skriftstørrelse, der er nemmere at læse. For den fulde udgave af sourcekoden med tilhørende kommentarer, henvises der til fil-bilagene.

#### 3.2.1 Device Driver

Til implementationen af device driveren er der taget udgangspunkt i en SPI device driver, som er blevet udviklet som gruppearbejde til øvelse 7 i faget HAL.[1] SPI device driveren blev som en del af HAL undervisningen udviklet til det punkt, hvor det lykkedes at sende en 8-bit char kommando til kernel space, som kunne tænde skærmen.

SPI device driveren er derfor blevet udvidet således at den kan håndtere et 12.288 bit char array. Det viste sig, at det ikke er muligt at allokere statisk hukommelse til et array på 12.288 bit i kernel space, da det maksimale, man kunne oprette på stacken, var 1024 bit. Løsningen blev derfor at allokere et array dynamisk ved hvert kald til `ssd1331_drv_write()`, så det istedet blev oprettet på heapen. Ulempen ved dynamisk allokering er, at det er dyrt i forhold til den tid, det tager, når der sammenlignes med statisk allokering. Derfor implementeres driveren således, at der anvendes en statisk allokert char buffer, når der sendes en 8-bit kommando, og der allokeres dynamisk, når der sendes data.

```

if(minor == 0){
    len = count;
    if(len > MAX_LEN_BUF){
        char* kbufData = kmalloc(sizeof(char) * len),
        GFP_KERNEL);
        if(copy_from_user(kbufData, ubuf, len))
            return -EFAULT;

        spi_write_message(ssd1331_dev.spi, kbufData, len);

        kfree(kbuf);
    }
    else{
        if(copy_from_user(kbuf, ubuf, len))
            return -EFAULT;

        spi_write_message(ssd1331_dev.spi, kbuf, len);
    }
}

```

### 3.2.2 SSD1331 klasse

Metoden init() implementeres for SSD1331 klassen ved først at anvende gpio\_write() til at skrive til Reset. Der indsættes et delay for at sikre, at SSD1331 driveren når at registrere Reset som hhv. HIGH og LOW. Dernæst gennemføres de trin, der er angivet på figur 2. Der er ændret i nogle af værdierne, så de passer med den ønskede implementation. Remap og color depth indstillingerne er ændret således at tredje bit sættes lav, hvilket betyder, at der mappes fra venstre mod højre, da dette føles mere intuitivt. Power Save Mode sættes til, da vi ønsker, at systemet bruger så lidt strøm så muligt, så det er velegnet til at kunne køre på batteri. Precharge Speed sættes til samme værdi som kontrastværdien for farve A, B og C, og da disse ændres til maksimal værdi, ændres Precharge Speed sig tilsvarende. Grunden til at kontrastværdien sættes op er, at vi ønsker at farverne skal stå meget stærkt, så det forstærker gamification udtrykket og dermed fanger brugerens blik. Implementationen kommer således til at se sådan ud:

```
void ssd1331::init() {
    gpio_write(RST, HIGH);
    usleep(100);
    gpio_write(RST, LOW);
    usleep(100);
    gpio_write(RST, HIGH);

    sendCommand(DISPLAY_OFF);
    sendCommand(SET_REMAP_AND_COLOR_DEPTH);
    sendCommand(0x72);
    sendCommand(SET_DISPLAY_START_LINE);
    sendCommand(0x0);
    sendCommand(SET_DISPLAY_OFFSET);
    sendCommand(0x0);
    sendCommand(MODE_NORMAL);
    sendCommand(SET_MULTIPLEX_RATIO);
    sendCommand(0x3F);
    sendCommand(SET_MASTER_CONFIG);
    sendCommand(0x8E);
    sendCommand(POWER_SAVE_MODE);
    sendCommand(0x1A);
    sendCommand(PHASE_PERIOD_ADJUSTMENT);
    sendCommand(0x74);
    sendCommand(DISPLAY_CLOCK_DIV);
    sendCommand(0xF0);
    sendCommand(SET_PRECHARGE_SPEED_A);
    sendCommand(0xFF);
    sendCommand(SET_PRECHARGE_SPEED_B);
    sendCommand(0xFF);
    sendCommand(SET_PRECHARGE_SPEED_C);
    sendCommand(0xFF);
    sendCommand(SET_PRECHARGE_LEVEL);
    sendCommand(0x3A);
    sendCommand(SET_V_VOLTAGE);
    sendCommand(0x3E);
    sendCommand(MASTER_CURRENT_CONTROL);
    sendCommand(0x06);
    sendCommand(SET_CONTRAST_A);
    sendCommand(0xFF);
    sendCommand(SET_CONTRAST_B);
    sendCommand(0xFF);
    sendCommand(SET_CONTRAST_C);
    sendCommand(0xFF);
}
```

Metoderne displayImage() og sendCommand() er implementeret fuldstændig som beskrevet under design ved at anvende de andre metoder i klassen til at udføre de anførte trin.

Metoden gpio\_write() tager to parametre: En pin og en værdi. Først undersøges der for, om pin er angivet til at være Reset eller D/C, og dernæst oprettes en filedescriptor med skriverettigheder til enten /dev/gpio\_rst eller /dev/gpio\_dc. Endeligt skrives værdien til filen, hvilket medfører, at write funktionen i driveren kaldes.

```
int ssd1331::gpio_write(int pin, int value)
{
    size_t fd;
    int err;
    char buf[GPIO_BUF_SIZE];

    if(pin == RST)
    {
        if((fd = open(FILE_PATH_RST, O_WRONLY)) < 0) {
            err = errno;
            printf("ERROR: GPIO device can not open\n");
            return errno;
        }

        sprintf(buf, "%d", value);
        err = write(fd, &buf, strlen(buf));
        close(fd);
    }

    if(pin == DC)
    {
        if((fd = open(FILE_PATH_DC, O_WRONLY)) < 0) {
            err = errno;
            printf("ERROR: GPIO device can not open\n");
            return errno;
        }

        sprintf(buf, "%d", value);
        err = write(fd, &buf, strlen(buf));
        close(fd);
    }

    return err;
}
```

Metoderne spi\_transfer\_cmd() og spi\_transfer\_data() opretter begge en filedescriptor med skriverettigheder til /dev/gpio.spi\_drv0. Forskellen mellem de to metoder ligger primært i de parametre, som de tager. spi\_transfer\_cmd() tager en char som parameter og spi\_transfer\_data() tager en pointer til et char array som parameter. Derudover tager sidstnævnte også størrelsen på arrayet som parameter. Dette er nødvendigt, da vi ellers ikke har mulighed for at vide, hvor stort et array, pointeren peger på. Implementationen af de to metoder ligner hinanden meget, så derfor vises der kun kode for en af metoderne.

```
int ssd1331::spi_transfer_data(char* buffer, int size)
{
    int status = 0;
    int fd, num_wr, value;

    fd = open(FILE_PATH_SPI0, O_WRONLY);
    if(fd == -1)
    {
        printf("Failed to open with err: %s", strerror(errno));
        return -1;
    }

    num_wr = write(fd, buffer, size);
    if(num_wr == -1)
    {
        printf("Failed to write with err: %s", strerror(errno));
        return num_wr;
    }

    status = close(fd);
    if(status == -1)
    {
        printf("Failed to close with err: %s", strerror(errno));
        return status;
    }

    return 0;
}
```

### 3.2.3 Images klasse

Metoden drawPoint er implementeret ved først at tjekke for, om x og y værdierne er gyldige. Dernæst gemmes farvekoden i bufferen ved først at gemme de 8 mest betydende bits og dernæst de 8 mindst betydende bits fra RGB farvekoden. Figur 9 viser et eksempel på, hvordan bufferen for en skærm på 2x3 pixels kunne se ud. For at gemme på den rigtige plads i arrayet skal vi for mest betydende bits multiplicere både x- og y-værdien med 2, da hvert plads i arrayet er 8 bit og hvert koordinatsæt er repræsenteret af 2 x 8 bit. Derudover skal y-værdien også multipliceres med bredden på skærmen, da vi hver gang, vi stiger med en y-værdi, flytter os en med hel række x-værdier. Endeligt summeres de nye x- og y-værdier. Et eksempel på en udregning kunne være for koordinaterne (1,2), som giver beregningen  $1*2+2*2=10$ , hvilket stemmer overens med det forventede for figur 9. For mindst betydende bit lægges 1 til i beregningen, da der således skrives i næste plads i arrayet.

```
void images::drawPoint(int x, int y, uint16_t color)
{
    if(x >= OLED_WIDTH || y >= OLED_HEIGHT)
        return;

    imageBuf_[x * 2 + y * OLED_WIDTH * 2] = color >> 8;
    imageBuf_[x * 2 + y * OLED_WIDTH * 2 + 1] = color;
}
```

0	1	2	3	4	5	6	7	8	9	10	11
MSB (0,0)	LSB (0,0)	MSB (1,0)	LSB (1,0)	MSB (0,1)	LSB (0,1)	MSB (1,1)	LSB (1,1)	MSB (0,2)	LSB (0,2)	MSB (1,2)	LSB (1,2)

Figur 9: Illustration af et buffer array

Implementeringen af bitmapImage() løber alle bits i et helt bitmap igennem med 2 for-løkker, og hvis det enkelte bit er sat, vil der blive tegnet et punkt i den farve, der er angivet som parameter.

```

void images::bitmapImage(uint8_t x, uint8_t y, const uint8_t *bmp,
char width, char height, uint16_t color)
{
    uint8_t i, j, byteWidth = (width + 7) / 8;
    for(j = 0; j < height; j++)
    {
        for(i = 0; i < width; i++) {
            if(*(bmp + j * byteWidth + i / 8) & (128 >> (i & 7))) {
                drawPoint(x + i, y + j, color);
            }
        }
    }
}

```

Første skridt i implementationen af tekstfunktionaliteten er at lave to bitmaps, som indeholder de ønskede characters for begge skriftstørrelser. Der tages udgangspunkt i ascii og startes ved ascii værdi nummer 21, mellemrum, og laves bitmap til og med ascii værdi nummer 126, tilte. For 3x5 gemmes hvert tegn ved at aflæse hvert enkelt bit i vandret rækkefølge for de enkelte tegn på figur 7. For 5x7 gemmes hvert tegn ved at aflæse hvert enkelt bit i lodret rækkefølge for de enkelte tegn på figur 8. Grunden til at det gøres forskelligt er ene og alene, fordi det er nemmest at aflæse HEX-værdierne for henholdsvis 4 og 8 bit uden linjespring.

```

static const uint8_t font3x5[95][3] = {
{0x00, 0x00, 0x00}, /* " ", 0 */
{0x04, 0x40, 0x40}, /* "!", 1 */

...
{0x00, 0xAE, 0x2C}, /* "y", 89 */
{0x00, 0xE4, 0x86}, /* "z", 90 */
{0x06, 0x4C, 0x60}, /* {" , 91 */
{0x04, 0x44, 0x40}, /* |/ , 92 */
{0x0C, 0x46, 0xC0}, /* }" , 93 */
{0x00, 0x2E, 0x80}, /* ~" , 94 */
};


```

I metoden bitmapChar gemmes værdien for det tegn, der skal printes, ved at tage ascii værdien og fratrække ”mellemrum”, dvs. værdien 32. Dette gøres, da ascii værdien 32 er lig med array plads 0. Dernæst tjekkes for den ønskede font, 3 for 3x5 eller 6 for 5x7 samt hvorvidt mode er sat til invertering (0) eller normal (1). Endeligt løbes arrayet igennem med to for-løkker i henholdsvis vertikal eller horisontal rækkefølge, og hvert enkelt punkt farves enten i sort eller den farve, som er givet som parameter.

```

void images::bitmapChar(uint8_t x, uint8_t y, char acsii,
char size, char mode, uint16_t hwColor)
{
    ...
    uint8_t ch = acsii - ' ';
    for(i = 0;i<size;i++) {
        if(size == 6)
        {
            if(mode)temp=font5x7[ch][i];
            else temp = ~font5x7[ch][i];
            for(j =0;j<8;j++)
            {
                if(temp & 0x80)
                    drawPoint(x, y, hwColor);
                else
                    drawPoint(x, y, 0);
                temp <<=1;
                y++;
                if((y-y0)==8)
                {
                    y = y0;
                    x++;
                }
            }
        }
        if(size == 3)
        {
            ...
        }
    }
}

```

Metoden bitmapString() tjekker først om den plads i char-arrayet, der peges på, indeholder nultermineringen. Er dette ikke tilfældet kaldes funktionen bitmapChar(), x-værdien flyttes de pladser til højre, som svarer til bredden på fontstørrelsen og pointeren flyttes til næste plads i char-arrayet.

```
void images::bitmapString
(uint8_t x, uint8_t y, const char *pString,
uint8_t size, uint8_t mode, uint16_t hwColor)
{
    if(size == 6)
    {
        while (*pString != '\0') {
            bitmapChar(x, y, *pString, size, mode, hwColor);
            x += size;
            pString++;
        }
    }

    if(size == 3)
    {
        ...
    }
}
```

RGB farverne laves ved definere RGB som 16 bit, der består af de 5 mest betydende bit fra rød, 6 mest betydende bit fra grøn og 5 mest betydende bit fra blå. Dette opnås ved bitshiftning af 8-bit værdier for hver af de tre farver. Farverne gemmes som enum.

```
#define RGB(R,G,B) (((R >> 3) << 11) | ((G >> 2) << 5) | (B >> 3))
enum Color {
    BLACK      = RGB( 0, 0, 0), // black
    ...
    CHOCOLATE = RGB(136, 68, 16), // chocolate
};
```

For at printe sensorværdierne på skærmen hentes værdierne fra Log klassen ved at bruge metoderne getLatest() og getWeek(). Der laves en char buffer og sprintf bruges til at kopiere en hel streng ind i bufferen, som så kan printes til skærmen.

```
char light[14];
sprintf( light, sizeof(light), "LIGHT: %d",
logPtr_->getLatest(LIGHT) );
```

For at printe plantens type og navn hentes værdierne fra planteID klassen ved at bruge metoderne getPlanteType() og getPlanteName(). Plantetypen bruges til at bestemme, hvilket billede der skal printes på skærmen. Der laves en char buffer og strcpy bruges til at kopiere strengen ind i bufferen. Ydermere anvendes .str() til at omdanne den C++ streng, som modtages fra planteID, til et C char array.

```
switch(planteIDPtr_->getPlanteType())
{
    case TYPE_FLOWER:
        bitmapImgPlant(76,0, FLOWER);
        break;

    case TYPE_LEAF:
        bitmapImgPlant(76,0, LEAF);
        break;

    case TYPE_CACTUS:
        bitmapImgPlant(76,0, CACTUS);
        break;

    default:
        break;
}

char name[6];
strcpy(&name[0], (planteIDPtr_->getPlanteName()).c_str());
```

Valideringen i metoden setPosition() tager udgangspunkt i, hvor mange valgmuligheder brugeren har på de forskellige skærbilleder. I plantedata har vi for eksempel fire valgmuligheder, når der er data i loggen, nemlig 0=lys, 1=vægt, 2=fugtighed og 3=ny plante, samt en enkelt valgmulighed, 0=ny plante, når der ikke er data i loggen.

```

oid images::setPosition(int pos)
{
    position_ = pos >= 0 ? pos : 0;
    switch (getImage()) {
        case PLANT_DATA:
            if(logPtr_->isEmpty())
            {
                if (pos > 0)
                    position_ = 0;
            }
            else
            {
                if (pos > 3)
                    position_ = 3;
            }
            break;

        case SETUP:
        case PLANT:
            if (pos > 1)
                position_ = 1;
            break;

        case SET_TYPE:
            if (pos > 2)
                position_ = 2;
            break;

        case SET_NAME:
            if (pos > 7)
                position_ = 7;
            break;

        default:
            break;
    }

}

```

### 3.2.4 Display klasse

I metoden show() laves først et kald til setImage() fra Image klassen, så der laves det korrekte skærbilledet i bufferen. Dernæst laves et kald til displayImage() fra SSD1331 klassen, hvor der gives en pointer til image bufferen ved at kalde getImageBuf() samt størrelsen på bufferen.

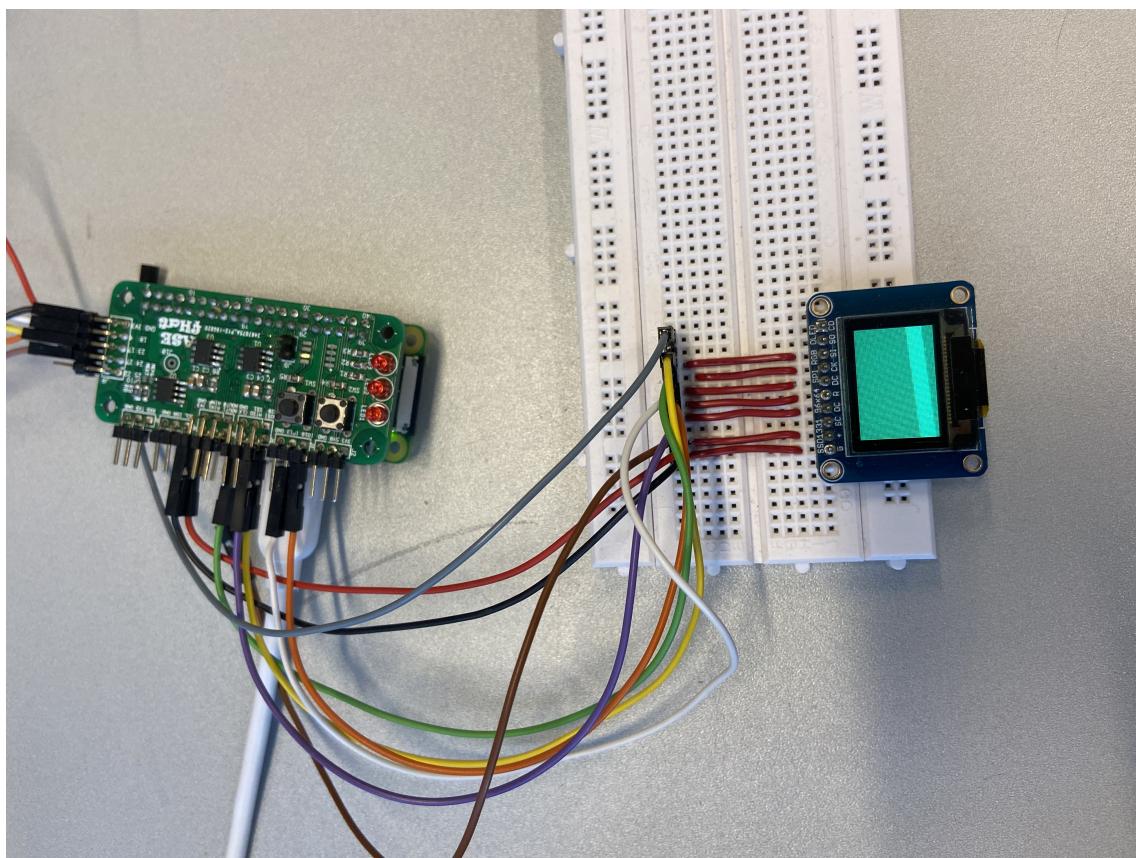
```
void display::show(char choice) {  
  
    imagesPtr_->setImage(choice);  
    ssd1331_.displayImage(imagesPtr_->getImageBuf(),  
                          imagesPtr_->getImageSize());  
}
```

I metoderne turnOn() og turnOff() laves der kald til sendCommand() fra SSD1331 klassen. Ydermere cleares skærbilledet ved turnOff, så vi ikke risikerer at brugeren tænder skærmen med et tidligere billede.

```
void display::turnOn() {  
    ssd1331_.sendCommand(DISPLAY_ON_NORMAL);  
}  
  
void display::turnOff() {  
    ssd1331_.sendCommand(DISPLAY_OFF);  
    show(CLEAR);      // Ensures black screen at startup  
}
```

## 4 Modultest

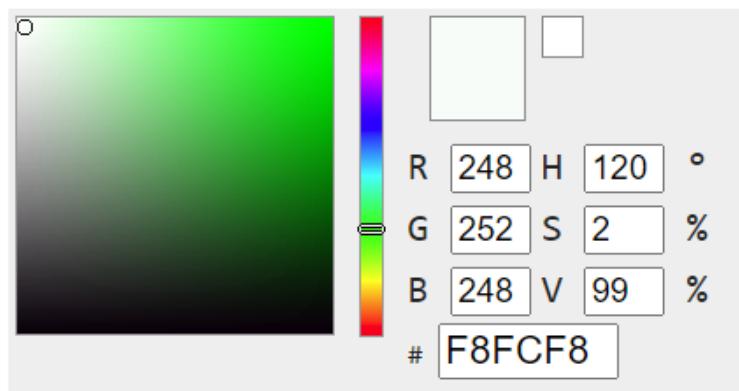
Der modultestes ved først at teste Device driveren og SSD1331 klassen sammen. Den primitive udgave af device driveren, som blev udviklet i HAL undervisningen, er allerede testet, og her ved vi, at SPI kommunikationen og output til GPIO virker, så det er netop rettelser, der baserer sig på et samspil med SSD1331 klassen, der skal testes for denne. Derfor giver det mening at teste dem begge sammen. Efterfølgende modultestes Display-klassen og images klassen sammen, da Display klassen anvender SSD1331 klassen og images klassen, men det er svært at teste images klassen uden at anvende Display klassen til at vise, om vi får de forventede images. Til at lave alle test omkring displayet anvendes testopsætningen vist på figur 10, hvor forbindelserne er lavet i overensstemmelse med tabel 2.



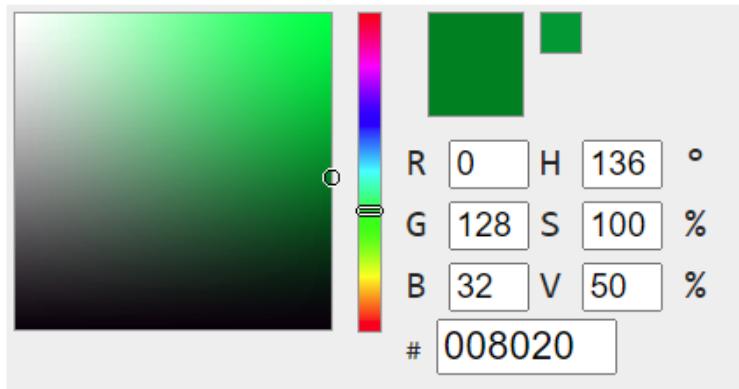
Figur 10: Testopsætning for test af display

## 4.1 Device Driver og SSD1331 klasse

I testen af SSD1331 klassen testes alle metoder, og hvis alle metoder virker som forventet, betyder det, at SSD1331 device driveren virker som forventet. Der laves et testprogram, som kalder de tre public metoder fra SSD1331 klassen. I de tre public metoder laves der kald til alle tre private metoder, og derfor vil disse også automatisk blive testet. Som billede til displayet laves en buffer på 12.288 bit, og denne skal indeholde det data, som skal vises i testen. For at gøre det simpelt fyldes bufferen med en ensartet værdi, hvilket vil svare til, at hele skærmen farves i én farve. Der vælges to tilfældige værdier, som ikke tilsvarer sort, til at fyldе bufferen, hhv. 0xFF og 0x04. Disse værdier svarer til de to 24-bit RGB farver, som er vist på figur 11 og figur 12.



Figur 11: Testfarve 1 - Hvid



Figur 12: Testfarve 2 - Grøn

Testprogrammet er vist nedenfor og her fremgår det, at der først laves et kald til init(). Efterfølgende kaldes sendCommand(DISPLAY\_ON\_NORMAL), som tænder skærmen. Endeligt kaldes displayImage(), først med den hvide testfarve og dernæst med den grønne testfarve.

```

#include "ssd1331.h"

int main()
{
    char imageBuf_[OLED_WIDTH * OLED_HEIGHT * 2];

    for(int i= 0; i < sizeof(imageBuf_); i++)
    {
        imageBuf_[i] = 0xFF;
    }

    ssd1331 ssd1331_;
    ssd1331_.init();

    ssd1331_.sendCommand(DISPLAY_ON_NORMAL);
    ssd1331_.displayImage(&imageBuf_[0], sizeof(imageBuf_));

    sleep(1);

    for(int i= 0; i < sizeof(imageBuf_); i++)
    {
        imageBuf_[i] = 0x04;
    }

    ssd1331_.displayImage(&imageBuf_[0], sizeof(imageBuf_));

    return 0;
}

```

Testen gennemføres, og der konkluderes at skærmen viser det forventede. Updateingen af skærmens billede sker uden at man med det blotte øje kan se, at hvert enkelt pixel tegnes. Hermed virker SSD1331 klassen og SSD1331 device driveren som ønsket.

## 4.2 Images klasse og Display klasse

Den endelige test af Images kan først gennemføres efter PlanteID og Log klasserne er lavet. Da Display klassen anvender både SSD1331 klassen, skal den testes sidst, og derfor laves en samlet test for Images og Display klasserne. I det følgende vises et uddrag af testprogrammet:

```
int main()
{
    ...

    displayObj.turnOn();
    sleep(5);
    displayObj.show(START);
    sleep(5);

    planteIDObj.setPlanteType(TYPE_FLOWER);
    planteIDObj.setPlanteName("JOHN");
    displayObj.show(PLANT_DATA);
    sleep(5);

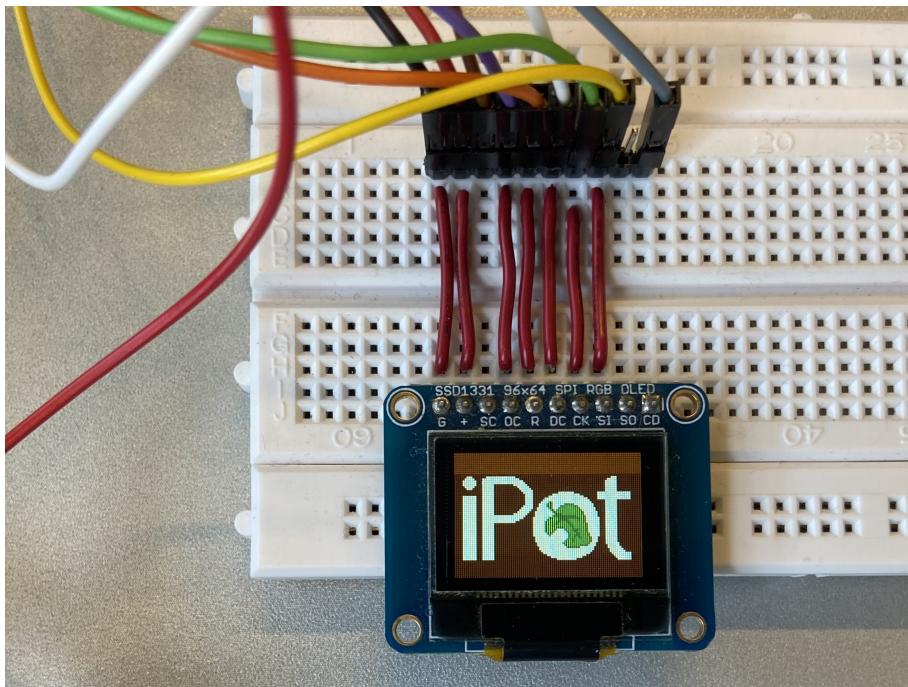
    int a, b, c, d;
    for (size_t i = 0; i < 12 * 24 * 7; i++){
        a = 10; b = 20; c = 3; d = 4;
        logObj.setLog({ a,b,c,d });
    }

    planteIDObj.setPlanteType(TYPE_LEAF);
    displayObj.show(PLANT_DATA);
    sleep(5);
    imagesObj.setPosition(1);
    planteIDObj.setPlanteType(TYPE_CACTUS);
    displayObj.show(PLANT_DATA);
    sleep(5);
    imagesObj.setPosition(2);
    planteIDObj.setPlanteType(TYPE_CACTUS);
    displayObj.show(PLANT_DATA);
    sleep(5);
    imagesObj.setPosition(3);
    planteIDObj.setPlanteType(TYPE_CACTUS);
    displayObj.show(PLANT_DATA);
    sleep(5);

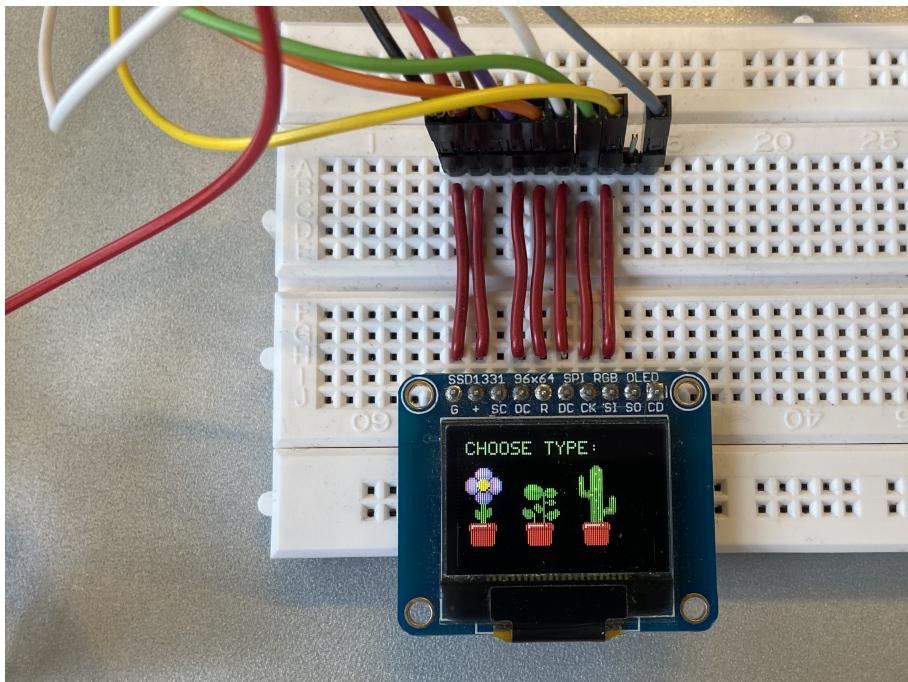
    ...

    return 0;
}
```

Resultaterne af testen vises i tabel 3, og der er vist et par eksempler fra gennemførelsen af testen på figur 13 og figur 14.



Figur 13: Test af iPot logo



Figur 14: Test af plantebilleder

Test	Forventet resultat	Resultat
turnOn() kaldes	Skærmen forbliver sort	OK
show(START) kaldes	Startskærmen med iPot logo vises	OK
Plantetype sættes til flower, plantenavn sættes til John, show(PLANT_DATA) kaldes	Skærmen viser en blomst. Den hedder John. Der vises "no data!"	OK
Plantetype sættes til leaf, log fyldes med data, show(PLANT_DATA) kaldes	Skærmen viser bladplanten. Der vises data for light, weight, humidity og waterlevel. Light teksten er inverteret.	OK
Position sættes til 1, show(PLANT_DATA) kaldes	Light er ikke inverteret. Weight inverteres.	OK
Position sættes til 2, show(PLANT_DATA) kaldes	Weight er ikke inverteret. Humidity inverteres.	OK
Position sættes til 3, show(PLANT_DATA) kaldes	Humidity er ikke inverteret. New plant inverteres.	OK
Position sættes til 0, show(PLANT) kaldes	Plante vises med billede og navn. OK er inverteret.	OK
Position sættes til 1, show(PLANT) kaldes	Plante vises med billede og navn. Cancel er inverteret.	OK
Position sættes til 0, show(SETUP) kaldes	Setup new plant vises. OK er inverteret.	OK
Position sættes til 1, show(SETUP) kaldes	Setup new plant vises. Cancel er inverteret.	OK
Position sættes til 0, show(SET_TYPE) kaldes	Plantetyper vises. 1. valgmulighed er inverteret.	OK
Position sættes til 1, show(SET_TYPE) kaldes	Plantetyper vises. 2. valgmulighed er inverteret.	OK
Position sættes til 2, show(SET_TYPE) kaldes	Plantetyper vises. 3. valgmulighed er inverteret.	OK
Position sættes til 0, show(SET_NAME) kaldes	Plantenavne vises. 1. valgmulighed er inverteret.	OK
Position sættes til 1, show(SET_NAME) kaldes	Plantenavne vises. 2. valgmulighed er inverteret.	OK
Position sættes til 2, show(SET_NAME) kaldes	Plantenavne vises. 3. valgmulighed er inverteret.	OK
Position sættes til 3, show(SET_NAME) kaldes	Plantenavne vises. 4. valgmulighed er inverteret.	OK
Position sættes til 4, show(SET_NAME) kaldes	Plantenavne vises. 5. valgmulighed er inverteret.	OK
Position sættes til 5, show(SET_NAME) kaldes	Plantenavne vises. 6. valgmulighed er inverteret.	OK
Position sættes til 6, show(SET_NAME) kaldes	Plantenavne vises. 7. valgmulighed er inverteret.	OK
Position sættes til 7, show(SET_NAME) kaldes	Plantenavne vises. 8. valgmulighed er inverteret.	OK
Position sættes til 0, show(LOG_LIGHT) kaldes	Lyslog vises.	OK
show(LOG_HUMIDITY) kaldes	fugtighedslog vises.	OK
show(LOG_WEIGHT) kaldes	vægtlog vises.	OK

Tabel 3: Modultest af Images og Display klasserne

## Litteratur

- [1] Michell Bonde Koretke Anette Lihn Szymon Patryk Palka. *HAL Øvelse 7 - Linux Device Model and Bus Interface*. 2023.
- [2] Solomon Systech Limited. *SSD1331*. 2007.
- [3] *Raspberry Pi Documentation*. <https://www.raspberrypi.com/documentation/computers/os.html>. [Online; besøgt 07-05-2023].
- [4] Argha Sengupta. “Does the Dark Mode on the Computer Actually Save Electricity?” I: *ScienceABC* (2022). URL: <https://www.scienceabc.com/innovation/does-the-dark-mode-on-the-computer-actually-save-electricity.html>.
- [5] Univision. *Product Specification UG-9664HDDAG01*. 2006.