

AARHUS UNIVERSITET

ANVENDTE MICROCONTROLLER SYSTEMER

6. SEMESTER

Color Sorting System

Gruppemedlemmer:

Daniel Tøttrup

Stinus Lykke Skovgaard

AUID

au544366

au520659



AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

29. maj 2018

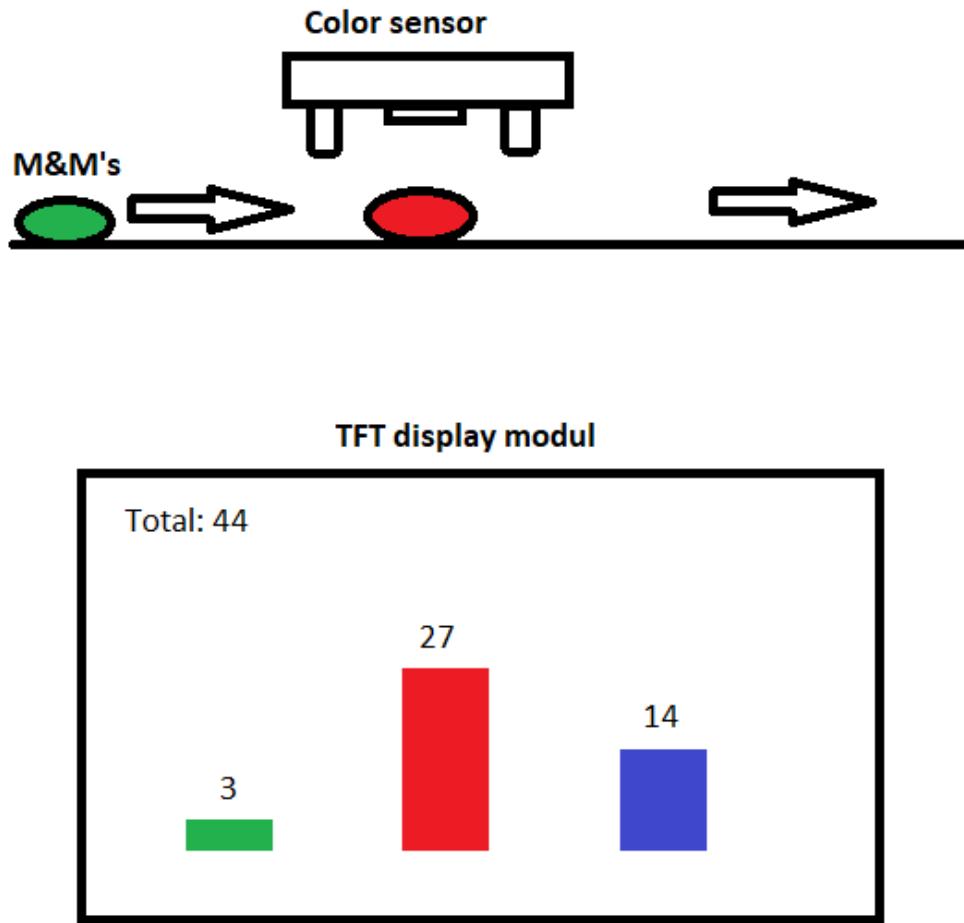
Indhold

1	Indledning	3
2	Krav	4
2.1	UC1 - Read Color	5
2.2	UC2 - Save Data	5
2.3	UC3 - Read Data	5
2.4	Afgrænsning	5
3	Systemarkitektur	6
3.1	Blokidentifikation	6
3.2	Blokinteraktion	6
4	Color Sensor Module	8
4.1	LC Technology TCS3200	8
4.2	Input Capture	9
4.3	Color Sensor Module Software	10
4.4	Test	11
5	TFT Display Module	14
5.1	Hardware	14
5.2	Software	14
5.3	Test	19
6	I2C kommunikation	21
6.1	I2C master	21
6.2	I2C slave	22
6.3	Test	23
7	Alternative Løsninger	25
7.1	Valg af Kommunikation	25
7.2	Valg af color sensor	25
7.3	Valg af skærm	25
8	Konklusion	26

Figurer

1	Konceptbillede for CSS	3
2	Usecase diagram for CSS	4
3	Color Sorting System BDD	6
4	Color Sorting System IBD	7
5	Styring af photodioder	8
6	Output frekvens ved forskellige farver	8
7	Output frekvens skalering	9
8	Input Capture illustration	9
9	Data flow diagram	11
10	Terminal output ved rød, grøn og blå test	12
11	Test opstilling af Color Sensor Modulet	13
12	MCU-Interface Mode	14
13	Tidsforsinkelser	15
14	Flowchart TFT Display software	19
15	TFT Display Module enhedstest	20
16	I2C timing diagram	21
17	Logic analyzer for I2C	23
18	Test opstilling af I2C	24

1 Indledning



Figur 1: Konceptbillede for CSS

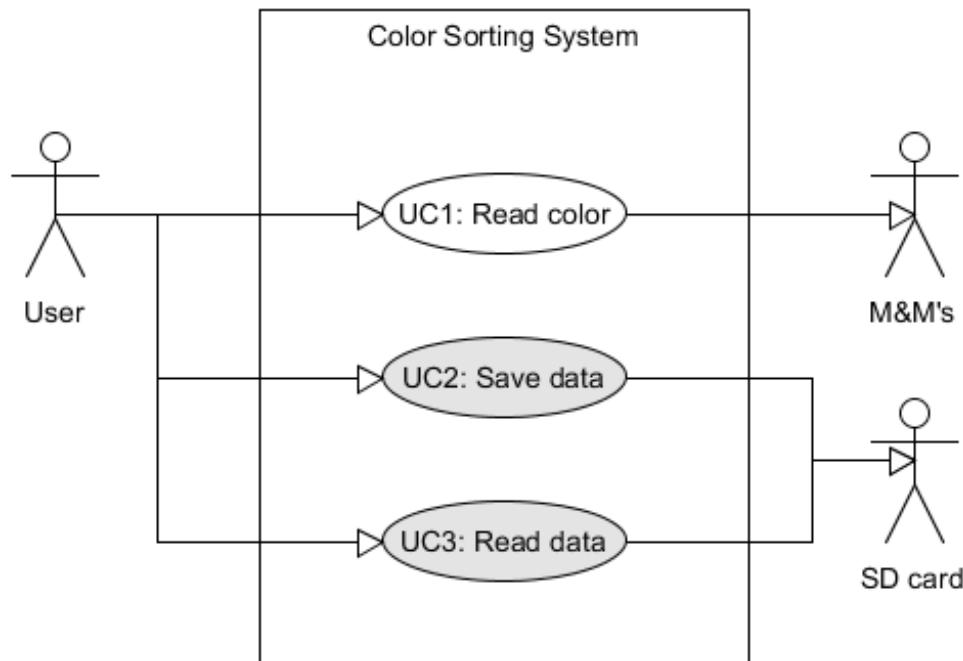
Color Sorting System(CSS) gør det nemt at sortere diverse emner baseret på deres farve. Disse emner kan være alt fra fødevare til maskin-komponenter, så længe de kan identificeres på farve. CSS fungerer ved at lade emnet passere under en farve sensor, som kan identificere farven på emnet og videresende hvilken farve emnet har til et TFT displaymodul. TFT modulet kan så ved hjælp af søjle diagrammer, fortælle hvor mange forskellige farvet emner der har passeret sensoreren. Resultaterne kan gemmes på et SD kort, hvis man senere skulle bruge dataen. Dog er implementeringen af SD kortet ikke fuldendt i denne prototype, og vil i en viderudvikling af projektet have stor prioritet. Desuden er der kun lagt vægt på color sensor og TFT display modularne, hvilket vil sige at selve sorteringsmekanikken ikke er implementeret i den nuværende prototype.

2 Krav

I dette afsnit beskrives kravene til CSS og hvilken funktionalitet systemet skal have. Der er stillet nogle enkelte krav fra undervisers side, hvor nogle af disse krav skal indgå i projektet:

- Use In Circuit debug tools
- Implement Drivers, dealing with time critical parameters.
- Implement Boot Loaders for updating microcontroller firmware
- Use USB to interface a microcontroller
- Use operating systems for microcontrollers
- Use microcontroller knowledge in a final mini project

Flere af disse krav er implementeret i CSS. Der er brugt tidskritiske drivers til color sensoren og til implementeringen af I2C mellem de to microcontrollers.



Figur 2: Usecase diagram for CSS

På [Figure 2](#) ses usecase diagrammet som beskriver sammenhængen mellem aktørerne og de forskellige funktionaliteter der findes for systemet. Usecase 2 og 3 er grå, hvilket betyder at de ikke er implementeret i prototypen.

2.1 UC1 - Read Color

Denne usecase danner ramme for hvordan CSS måler en farve. Brugeren placerer emne der ønsket aflæst. Brugeren trykker på aflæs-knappen og den aflæste farve kan nu ses talt op på TFT display modulet.

2.2 UC2 - Save Data

Denne usecase danner ramme for, at gemme data på SD kort. Brugeren trykker på ”Save Data”knappen på skærmen. Data bliver derefter gemt på SD kort.

2.3 UC3 - Read Data

Denne usecase danner ramme for, at hente data fra SD kortet. Brugeren trykker på ”Load Data”knappen på skærmen. Gemt data bliver hentet fra SD kortet og vist på TFT skærmen.

2.4 Afgrænsning

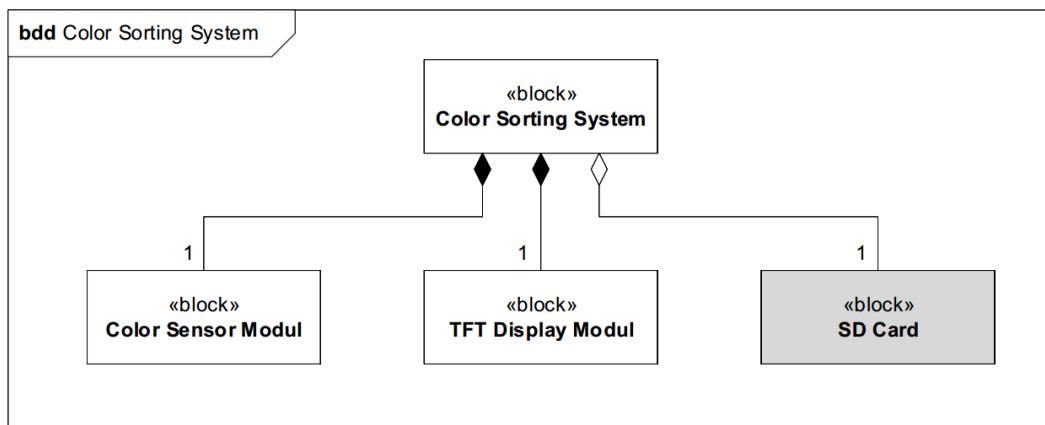
Det skal sige at usecase 1 og 2 ikke er implementeret i denne prototype. Der er gjort forsøg på at få dem implementeret, men grundet tidspres fik de Derudover var det også tænkt fra start, at der ikke skulle være behov for brugerinput for at kunne aflæse farve, men på grund af tidsmangel fik gruppen ikke implementeret et system der kunne fodre CSS med emner. Dette gøres istedet for manuelt og vil i fremtiden skulle noget automatisering af CSS implementeres.

3 Systemarkitektur

Vores systemarkitektur fungerer som den overordnede ramme for hvordan vi senere har implementeret vores system. Dette afsnit vil give et overblik over vores systems arkitektur, for at give et overskueligt overblik over systemet. Det er her at den beskrevne funktionalitet deles ud i mindre moduler.

3.1 Blokidentifikation

På figur [Figure 3](#) ses det overordnede BDD, som beskriver de enkelte moduler som systemet indeholder. Hver blok beskriver en funktionalitet, som systemet håndterer. I det følgende vil de enkelte moduler og funktionalitet kort beskrives.



Figur 3: Color Sorting System BDD

TFT Display Module:

Har til opgave at modtage data fra Color Sensor Module og vise brugeren på baggrund af det modtaget data, antallet af hver enkelt farve målt, samt det totale antal farver. Dette modul skulle også havde haft ansvaret for at sende data til SD-kortet, hvis den del var blevet implementeret.

Color Sensor Module:

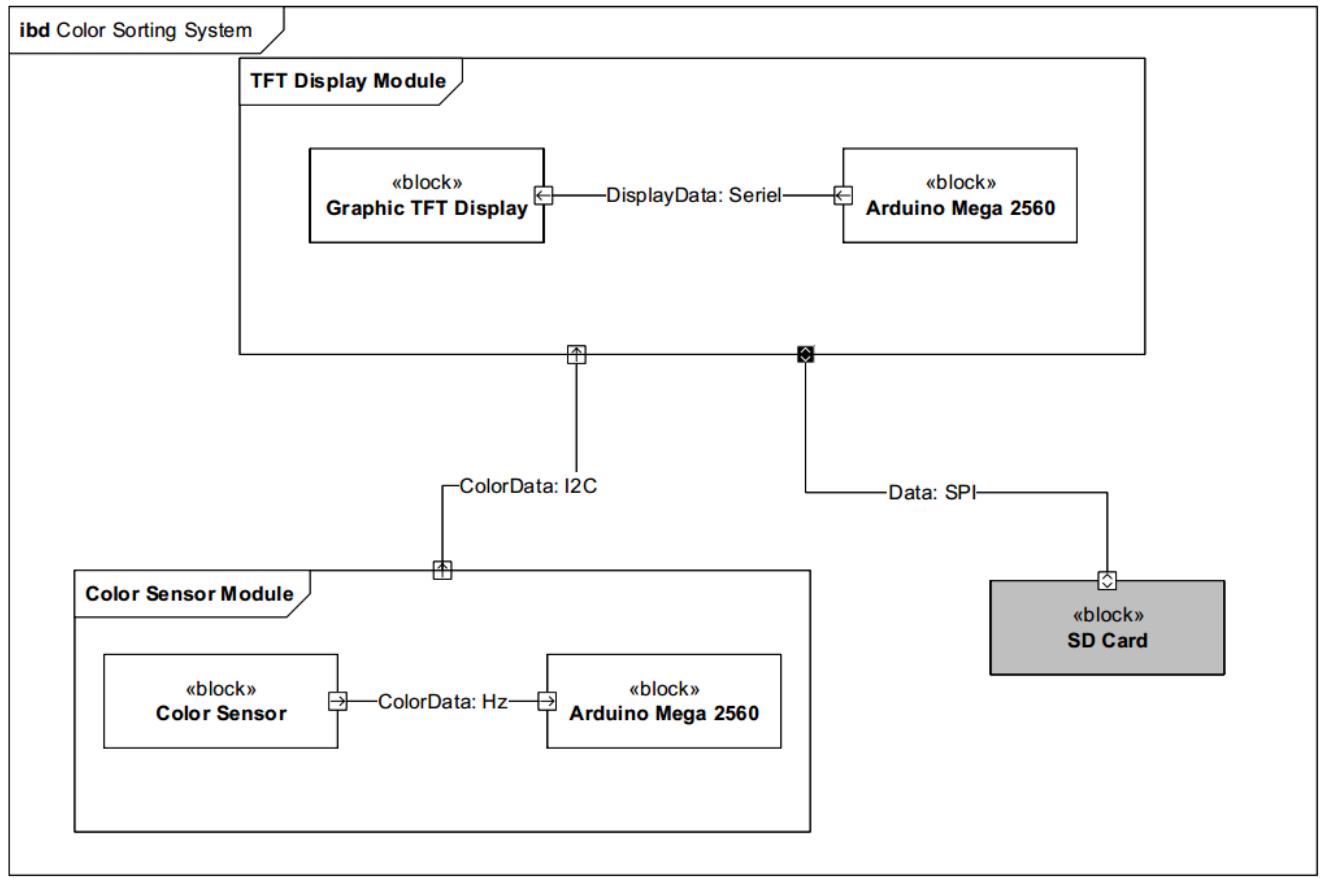
Har til opgave at måle farven placeret under Color Sensoren, samt sende informationen videre til TFT Display Module.

SD-Kort:

Et SD kort som var tiltænkt at kunne opbevare data som efter systemet slukkes. Dette blev dog taget ud af systemet. Hvilket visualiseres ved den grå farve i BDD'et.

3.2 Blokinteraktion

På [Figure 4](#) nedenfor ses det overordnede IBD for systemet. IBD'et viser de forskellige hardwareblokke i systemet og deres interaktion mellem hinanden. Interaktionen mellem blokkene bliver beskrevet mere detaljeret under dokumentationen for de enkelte blokke. I forhold til det overordnede BDD på [Figure 3](#) har vi valgt at vise hvilke hardware blokke de enkelte moduler består af, for at give et indblik i interaktionen internt i modulerne.



Figur 4: Color Sorting System IBD

4 Color Sensor Module

Color sensor modulet indeholder både en microcontroller og en color sensor. Til dette projekt har gruppen valgt at bruge en LC Technology TCS3200. Sensoren blev valgt fordi den var på lager i Embedded Stock, og at den ville passe godt til dette projekt. Til at styre denne sensor bruges en Arduino mega 2560.

4.1 LC Technology TCS3200

LC Technology TCS3200, virker ved at have 8x8 array af fotodioder. 16 fotodioder med et grønt filter, 16 fotodioder med et blåt filter, 16 fotodioder med et rødt filter og 16 fotodioder uden noget filter. De kan alle styres ved at sætte to pins(S2 og S3) høj eller lav. [3] Se [Figure 5](#)

S2	S3	PHOTODIODE TYPE
L	L	Red
L	H	Blue
H	L	Clear (no filter)
H	H	Green

Figur 5: Styring af photodioder

For at aflæse farveintensiteten, bliver man nødt til at måle på sensorens output pin. Signalet der kommer ud er et firkantssignal og farveintensiteten er bestemt alt efter hvor høj frekvensen er. For at se hvilken farve emnet har, bliver man nødt til at aktivere de forskellige fotodioder hver for sig, og tage en måling på output pin'en hver gang man har skiftet fotodiode. Derefter kan man så sammenligne de tre målinger (Clear bliver ikke målt) og se hvilket er størst. Hvis der skal måles andre farver end rød, grøn og blå, som fx gul, bliver man nødt til at kigge både på grøn og på rød. Hvis begge er lige høje må det være gul. Dog reagerer de forskellige fotodioder forskelligt på hvilken farve man ser på, så der skal der tages højde for. På [Figure 6](#) kan man se hvordan de forskellige fotodioder reagerer ved forskellige bølgelængder(farver)[3].

PARAMETER	TEST CONDITIONS	CLEAR PHOTODIODE S2 = H, S3 = L			BLUE PHOTODIODE S2 = L, S3 = H			GREEN PHOTODIODE S2 = H, S3 = H			RED PHOTODIODE S2 = L, S3 = L			UNIT
		MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
f_o Output frequency (Note 9)	$E_e = 47.2 \mu\text{W}/\text{cm}^2$, $\lambda_p = 470 \text{ nm}$	12.5 (4.7)	15.6 (5.85)	18.7 (7)	61%	84%	22%	43%	0%	6%	kHz			
	$E_e = 40.4 \mu\text{W}/\text{cm}^2$, $\lambda_p = 524 \text{ nm}$	12.5 (4.7)	15.6 (5.85)	18.7 (7)	8%	28%	57%	80%	9%	27%				
	$E_e = 34.6 \mu\text{W}/\text{cm}^2$, $\lambda_p = 640 \text{ nm}$	13.1 (4.9)	16.4 (6.15)	19.7 (7.4)	5%	21%	0%	12%	84%	105%				

Figur 6: Output frekvens ved forskellige farver

En sidste ting der er værd er at vide om TCS3200, er dens indbygget frequency scaler. Den kan bruges til at styre skaleringen af output signalets frekvens. Skaleringen kan styres ved

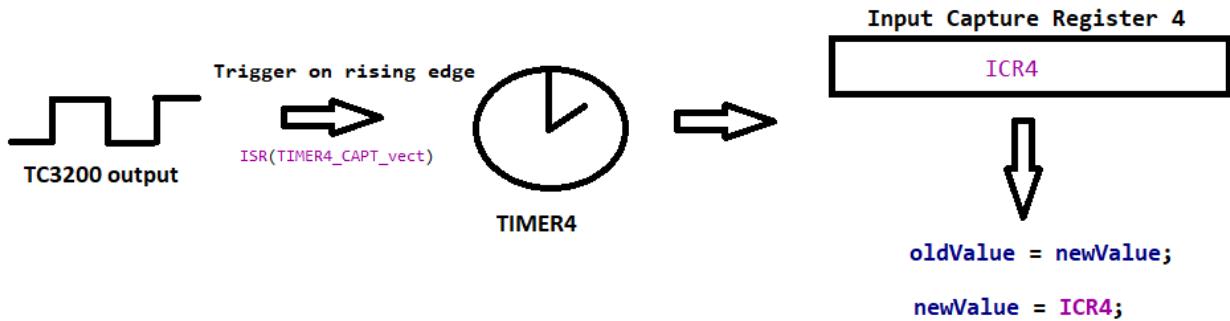
at sætte S1 og S2 høj eller lav. Til CSS bruges 2% skalering, da 100% skalering, ville betyde at output frekvensen kunne komme op over 50kHz. En lavere frekvens vil være fordelagtig i forhold til at få en præcis måling. [3] Se [Figure 7](#)

S0	S1	OUTPUT FREQUENCY SCALING (f_o)
L	L	Power down
L	H	2%
H	L	20%
H	H	100%

Figur 7: Output frekvens skalering

4.2 Input Capture

Input Capture er en metode der ofte bruges i embedded systemer, til at måle på diverse signaler. Til at måle outputsignalet fra TC3200 color sensor bruges denne metode.



Figur 8: Input Capture illustration

Input Capture [1] fungerer ved at have et interrupt der trigger på rising edge. Når dette interrupt bliver kaldt, tages et øjebliksbillede af TIMER4s værdi, som gemmes i Input Capture Register 4. Denne værdi gemmes i en variabel, som bagefter bruges til at beregne en frekvens.

Derudover skal der også tages højde for overflow, ellers kan man risikerer at en måling ikke vil give en korrekt frekvens. For at udbedre dette problem bruges en anden interrupt, ISR(TIMER4_OVF_vect). Den trigger hver gang der kommer overflow, og 65535 lægges til newValue. Koden til hvordan frekvensen udregnes kan ses nedenunder.

```

1 ISR(TIMER4_CAPT_vect)
2 {
3     oldValue = newValue;
4     newValue = ICR4;
5
6     if(newValue < oldValue)
7     {
8         period = oldValue-newValue;
9     }
10    else
11    {
12        newValue + overflow;
13        period = oldValue - newValue;
14    }
15    freq = F_CPU/period;
16    FREQFLAG = 1;
17 }
```

Det kan også ses i koden, hvordan overflow værdien lægges til newValue, hvis newValue er mindre end oldValue. FREQFLAG bruges så man altid er sikker på at freq har en ny værdi. FREQFLAG skal selvfølgelig sættes til 0, når man har brugt freq.

4.3 Color Sensor Module Software

For nemt at kunne forstå softwaren brugt til TC3200, er der blevet udarbejdet et data flow diagram, der giver et overblik over dataflowet i softwaren. Dette diagram kan ses på [Figure 9](#). Som det ses i diagrammet, bliver der taget en frekvensmåling for hver farve. Dette gøres, som beskrevet tidligere, ved at sætte S2 og S3 enten høj eller lav. Når en måling er taget, sammenlignes de tre målinger for at se hvilken farve er mest repræsenteret. Derefter sendes en char som enten er 'R', 'G' eller 'B'. Kommunikationen sker via I2C. I2C vil blive uddybt mere i kommunikationsafsnittet. Derefter starter koden forfra igen, og tager en ny frekvensmåling. Source koden kan også ses under bilag.



Figur 9: Data flow diagram

4.4 Test

En enhedstest er blevet lavet da modulet var færdigt. Resultatet af denne test vil bliver fremvist her. Først er input capture softwaren testet ved at bruge en funktions generator til at sende et firkantssignal ind på input capture pin'en. For at se om softwaren kunne aflæse den rigtige frekvens, er der gjort brug af UART kommunikation til en tilsluttet PC. Fra en terminal på PC'en har man kunne aflæse frekvensen fra arduino'en og derfra konstatere at programmet har virket.

```

528 Green
601 Blue
THE MEASURED COLOR IS RED!
1839 Red
528 Green
606 Blue
THE MEASURED COLOR IS RED!
1837 Red
775 Green
605 Blue
THE MEASURED COLOR IS RED!
1833 Red
530 Green
604 Blue
THE MEASURED COLOR IS RED!
1832 Red
498 Green
604 Blue
THE MEASURED COLOR IS RED!
688 Green
557 Blue
THE MEASURED COLOR IS GREEN!
494 Red
690 Green
551 Blue
THE MEASURED COLOR IS GREEN!
493 Red
686 Green
573 Blue
THE MEASURED COLOR IS GREEN!
513 Red
685 Green
550 Blue
THE MEASURED COLOR IS GREEN!
494 Red
684 Green
551 Blue
THE MEASURED COLOR IS GREEN!
506 Red
684 Green
550 Blue
THE MEASURED COLOR IS GREEN!
501 Green
1133 Blue
THE MEASURED COLOR IS BLUE!
517 Red
506 Green
1154 Blue
THE MEASURED COLOR IS BLUE!
667 Red
540 Green
1165 Blue
THE MEASURED COLOR IS BLUE!
557 Red
541 Green
1167 Blue
THE MEASURED COLOR IS BLUE!
555 Red
544 Green
1174 Blue
THE MEASURED COLOR IS BLUE!
555 Red
545 Green
1178 Blue
THE MEASURED COLOR IS BLUE!

```

Figur 10: Terminal output ved rød, grøn og blå test

Til test af color sensoren, er der gjort brug af Input Capture programmet sammen med UART kommunikation til en PC. Sensorens output pin blev koblet til input capture pin'en, og tre målinger blev taget, én for hver farve(RGB). Farven med den højeste frekvens blev desuden sendt med som et bogstav, enten R B eller G. Som farvet forsøgsemne, blev tre forskelligt farvet stykker papir brugt. For at få så godt et resultat som muligt, skulle papiret holdes max 1cm fra sensoren, dog var det grønne papir stadig svært at opfange for sensoren. På [Figure 10](#) kan man se terminal outputtet fra sensoren. Tallene til venstre for "Red", "Green" og "blue" er frekvensen målt ved den farve. Man kan se hvordan sensoren har svært ved at opfange den grønne farve. Dette skyldes højest sandsynligt at farven ikke var den samme som databladet brugte til at teste med($\lambda = 524nm$)[\[3\]](#). Udover den grønne farve ikke passede helt, kan sensoren stadig skelne imellem de tre forskellige farver.

Test opstillingen kan ses på figuren nedenunder:



Figur 11: Test opstilling af Color Sensor Modulet

5 TFT Display Module

TFT Display modulets primære opgave er både at fungere som visuel grænse flade til brugeren, samt at fungere som et slags kontrol modul for hele systemet. Kontrol modul forstået på den måde, at dette modul står for at hente data fra Color Sensor Modulet, og optælle den hentede information. Dette modul afsnit vil beskrive TFT Display modulets funktionelit, samt overvejelser omkring analyse og design både hardware, softwaremæssigt. I dette afsnit vil kommunikationen mellem TFT Display modulet og resten af systemet også blive beskrevet.

5.1 Hardware

I arkitekturfasen gik den første overvejelse på hvilket display vi skulle benytte som grænsefladen til brugeren. Vi ønskede et displayet som kunne vise farver og havde en nogenlunde opløsning for give en god visuel oplevelse for brugeren. Da vi først havde opsat kravene for vores display, var selve valget ikke særligt svært. I undervisningen har vi arbejdet med "Graphic TFT Display", dette display opfyldte vores ønskede krav angående opløsning samt muligheden for at vise farver. Derfor faldt valgt ret hurtigt på dette display, da det også spillede sammen med vores Arduino 2560. For at kunne påmontere "Graphic TFT Display" på Arduino Mega 2560, har vi benyttet os af "ITDB02 Arduino MEGA shield 2.0". Databladet for dette shield er vedhæftet XX. I dette datablad er det også markeret hvilke porte ITDB02 shieldet der hører til bestemte indgange på displayet.

5.2 Software

I og med at dette moduls primære opgave er at være visuel grænseflade for brugeren, har langt det meste arbejde med dette modul lagt i softwaren. Hvis man kigger på Graphic TFT Display, kan den fungere i fire forskellige MCU-Interface modes, hvilket simpelt betyder, hvor stor en bus-interface man ønsker at arbejde med. Vi har valgt at arbejde med 16-bit bus-interface.

IM3	IM2	IM1	IM0	MCU-Interface Mode	CSX	WRX	RDX	D/CX	Function
0	0	0	0	8080 MCU 8-bit bus interface I	"L"	↓	"H"	"L"	Write command code.
					"L"	"H"	↓	"H"	Read internal status.
					"L"	↓	"H"	"H"	Write parameter or display data.
					"L"	"H"	↓	"H"	Reads parameter or display data.
0	0	0	1	8080 MCU 16-bit bus interface I	"L"	↓	"H"	"L"	Write command code.
					"L"	"H"	↓	"H"	Read internal status.
					"L"	↓	"H"	"H"	Write parameter or display data.
					"L"	"H"	↓	"H"	Reads parameter or display data.

Figur 12: MCU-Interface Mode

I og med at vi udelukkende ønsker at skrive til vores display, vælger vi at ignorere læse kommandoerne og udelukkende fokusere på at skrive kommandoerne. Derfor er RDX altid sat høj. Først implementerede vi WriteCommand i vores kode som ses lige nedenfor.

```

1 void WriteCommand(unsigned int command)
2 {
3     DATA_PORT_LOW = command;
4
5     DC_PORT &= ~(1<<DC_BIT);
6     CS_PORT &= ~(1<<CS_BIT);
7     WR_PORT &= ~(1<<WR_BIT);
8
9     _NOP();
10    WR_PORT |= (1<<WR_BIT);
11    _NOP();
12 }

```

Som det ses på figur [Figure 12](#) fra databadet[2] skal DCX og CSX sættes lavt, samt trigger kommandoen på WRX stigende flanke, derfor sættes WRX lav til at starte med. På figur [Figure 13](#) nedenfor ses et skema over de tidsforsinkelser der opstår, ved forskellige operationer. Her ses det at når WRX sættes lav, opstår der en forsinkelse på min 15ns. Derfor er der indsat en NOP() funktion i koden, hvilket står for "No Operation" som vil sige at programmet laver ingenting i en cyklus. Med en MCPU frekvens på 16Mhz svare det til 62,5ns. Herefter sættes WRX høj igen for at trigger kommandoen efterfulgt af endnu en NOP() funktion da der opstår samme forsinkelse når WRX sættes høj.

Signal	Symbol	Parameter	min	max	Unit	Description
DCX	tast	Address setup time	0	-	ns	
	taht	Address hold time (Write/Read)	0	-	ns	
CSX	tchw	CSX "H" pulse width	0	-	ns	
	tcs	Chip Select setup time (Write)	15	-	ns	
	trcs	Chip Select setup time (Read ID)	45	-	ns	
	trcsm	Chip Select setup time (Read FM)	355	-	ns	
	tcsf	Chip Select Wait time (Write/Read)	10	-	ns	
WRX	twc	Write cycle	66	-	ns	
	twrh	Write Control pulse H duration	15	-	ns	
	twrl	Write Control pulse L duration	15	-	ns	
RDX (FM)	trcfm	Read Cycle (FM)	450	-	ns	
	trdhfm	Read Control H duration (FM)	90	-	ns	
	trdlfm	Read Control L duration (FM)	355	-	ns	
RDX (ID)	trc	Read cycle (ID)	160	-	ns	
	trdh	Read Control pulse H duration	90	-	ns	
	trdl	Read Control pulse L duration	45	-	ns	
D[17:0], D[15:0], D[8:0], D[7:0]	tdst	Write data setup time	10	-	ns	For maximum CL=30pF For minimum CL=8pF
	tdht	Write data hold time	10	-	ns	
	trat	Read access time	-	40	ns	
	tratfm	Read access time	-	340	ns	
	trod	Read output disable time	20	80	ns	

Figur 13: Tidsforsinkelser

Dernæst implementerede vi WriteData, som tilnærmedesvis ligner WriteCommand bortset fra at DCX skal sættes høj i stedet for lav. Koden for WriteData ses lige nedenfor. På samme måde som WriteCommand trigger WriteData på en voksende flanke på WRX, derfor sættes WRX først lav og dernæst høj, med indsat NOP() funktioner for at tage højde for tidsforsinkelser.

```

1 void WriteData(unsigned int data)
2 {
3     DATA_PORT_HIGH = data >> 8;
4     DATA_PORT_LOW = data;
5
6     DC_PORT |= (1<<DC_BIT);
7     CS_PORT &= ~(1<<CS_BIT);
8     WR_PORT &= ~(1<<WR_BIT);
9
10    _NOP();
11    WR_PORT |= (1<<WR_BIT);
12    _NOP();
13 }

```

I vores DisplayInit() som vi kalder en gang i koden til at Initialisere displayet, starter vi med at sætte vores Control Pins som outputs, samt sætte dem høje. Dernæst bliver RST sat lav i 300ms, det skyldes at der i databladet på side 230 står minimum 120ms, så det er sat til 300ms for at være på den sikre side. Efter vi igen sætter RST igen bliver sat høj, skal vi igen vente 120ms før vi må kalde SleepOut Command, derfor indsætter et delay på 130ms. Herefter vækkes displayet med Sleepout() kommandoen efterfulgt af Displayon(). Begge disse kommandoer er at finde i databladet på side 83. Den kommando der bliver sendt til MemoryAccessControl() sørger for at sætte rækkefølgen til BGR i stedet for RGB. Til sidst bliver en kommando sendt til InterfacePixelFormat(), denne kommando fortæller displayet at vi ønsker at køre med 16bit pr. pixel se side 134 i databladet.

```

1 DisplayInit()
2 {
3     DDRG |= 0b00000111;
4     DDRD |= 0b10000000;
5     DDRA = 0xFF;
6     DDRC = 0xFF;
7     PORTG |= 0b00000111;
8     PORTD |= 0b10000000;
9
10    RST_PORT &= ~(1<<RST_BIT);
11    _delay_ms(300);
12
13    RST_PORT |= (1<<RST_BIT);
14    _delay_ms(130);
15
16    SleepOut();
17
18    DisplayOn();
19
20    MemoryAccessControl(0b00001000);
21
22    InterfacePixelFormat(0b00000101);
23 }

```

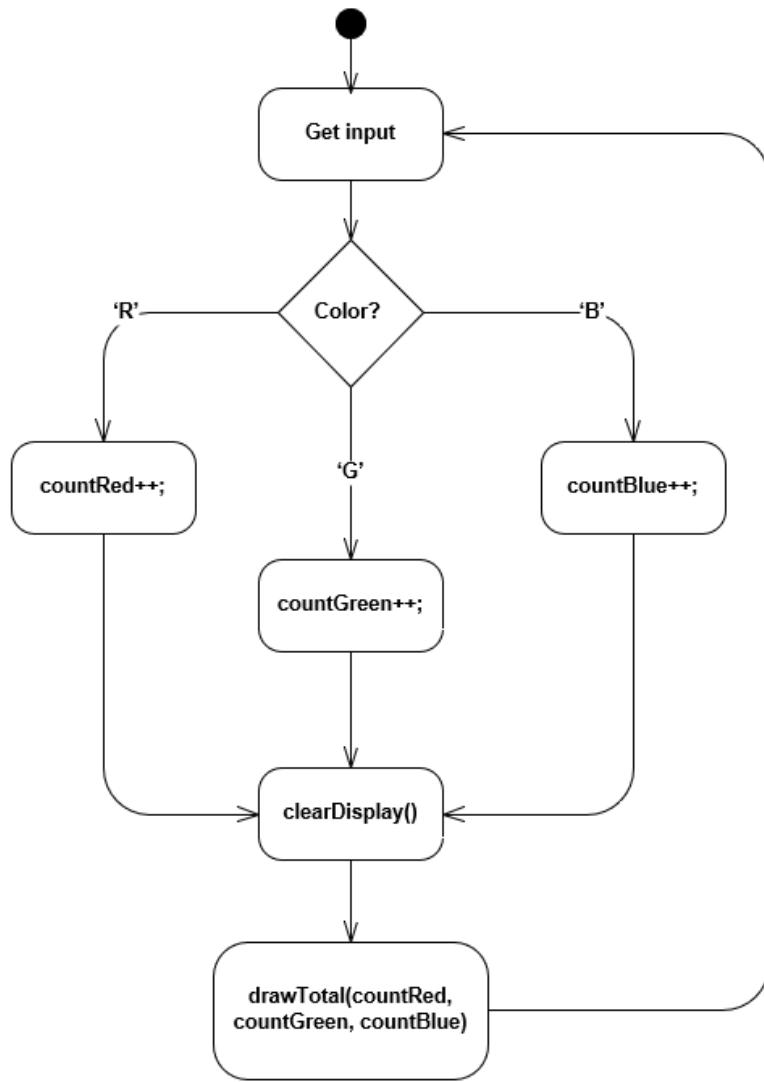
Efter at have initialiseret og opsat displayet efter egen ønske. Var den næste opgave at kunne skrive tal og bogstaver ud på vores display. Til dette benyttede vi os af et program ved navn 'TheDotFactory'. Vi fik programmet til at udskrive et kæmpe array, som indeholder alle

symboler, tal samt bogstaver vi kunne havde brug for. Sammen med et tilhørende array, der fortæller længden af hvert symbol samt dens offset. Disse to arrays genereret af programmet 'TheDotFactory' har vi lagt ind i en h-fil ved navn "DotFactory.h". På den efterfølgende [Table 1](#) vil funktionerne vi har benyttet blive overordnet blive beskrevet, i vores vedlagte kode vil en mere detaljeret gennemgang af koden kunne ses, i form af kommentar til hver linje kode i den vedlagte kode.

Navn	Beskrivelse
WritePixel()	Formålet er at bestemme farven for en pixel. Tager imod 3 parameter. Rød 0-31, grøn 0-63, blå 0-31 smider dem ind i write data.
SetColumnAddress()	Formålet er at kunne skrive til en hel linje lodret på en gang. Tager imod to parameter start og stop.
SetPageAddress()	Formålet er at kunne skrive til en hel linje vandret på en gang. Tager imod to parameter start og stop.
FillRectangle()	Meningen er at fyldet et rektangel med én farve. Tager imod syv parametre: start-x, start-y, bredde, højde, rød, blå og grøn.
getSymbolParameters()	Den tager tre parametre, start x og start y samt længden på det givne symbol. Formålet for denne funktion er at opdele alle de nødvendige informationer for et symbol. Så som længen af symbolet i byte, offset i forhold til arrayet fra TheDotFactory, offset i forhold til displayet. Denne information bliver så viderefivet til drawSymbol().
drawSymbol()	Tager information viderefivet fra getSymbolParameters(). Finder det givne symbol i arrayet fra TheDotFactory ved hjælp af informationen. Herefter gennemgås symbolet bit for bit, og displayet en sort pixel eller hvid pixel alt efter om det er et '1' eller '0' på den givne plads.
writeString()	Tager imod en string. Hvorefter den gennemgår denne string symbol for symbol og kalder getSymbolParameters() som videre kalder drawSymbol() på hvert symbol. Med mindre at symbolet er et mellemrum, så plusses x-positionen med seks.
writeInt()	Tager imod en long int. Hvorefter den int bliver lagt over i et array. Og på hver plads i dette array bliver getSymbolParameters kaldt, efter det første tal er detekteret.
DrawRed(), DrawGreen(), DrawBlue()	Meningen med denne funktion er at vise en søjle i enten farven rød, grøn eller blå alt efter hvilken af disse tre funktioner der bliver kaldt. Over denne søjle skal så skrives et tal, som følger søjlens højde. Funktionerne modtager to int, den første int angiver tallet som skal stå over søjlen, den anden int angiver højden på søjlen. Tallet bliver vist ved hjælp af funktionen writeInt(), og søjlen bliver vist ved hjælp af funktionen FillRectangle(). Den eneste forskel på disse tre funktioner er farven af søjlen.
drawTotal()	Meningen med denne funktion er at den skal tage tre parametre som antallet af hver farve detekteret. Disse tre parametre bliver så lagt sammen i en totalCount variable, og udregnet hvor stor en procentdel de hver udgør af det samlet antal optællinger. Den procentdel udgør så hvor høj hver enkel søjle skal være. Så vil DrawRed(), DrawGreen() og DrawBlue() blive kaldt hvor de tre parametre vil være tallene over søjlerne, højden vil være den udregnede procentdel og totalCount vil blive vist i øverste højre hjørne ved hjælp af writeInt().

Tabel 1: Funktioner benyttet til TFT Display

På [Figure 14](#) ses et lille flow chart over main koden til TFT Display Modul. Den får sit input igennem en I2C forbindelse, som vil blive mere uddybet i det efterfølgende afsnit. Det input bliver så undersøgt, og alt efter hvilket char 'R', 'G' eller 'B' der bliver modtaget optælles den tilhørende conut til den farve, hvorefter displayet bliver clearet og det nye resultat bliver vist. Så er modulet klar til et nyt input.



Figur 14: Flowchart TFT Display software

5.3 Test

Efter modulet stod færdig blev en enhedstest af modulet lavet resultatet af dette ses på [Figure 15](#). I testen blev countBlue og countGreen hver pludset op 1 og countRed blev pludset op med 2 en gang i sekundet. Som det kan ses passer vores procentvise fordeling af søjlerne. Blå og grøn søje har samme høje og den røde søje er dobbelt så høj. Det er kun i enhedstesten af vi pludser count variablerne op en gang i sekundet. I det færdige system vil denne data komme fra I2C linjen.



Figur 15: TFT Display Module enhedstest

6 I2C kommunikation

Til at starte med havde gruppen ikke planer om at inkorporere I2C i NSS, da den simple løsning var at have både TFT skærmen og Color Sensoren sat til samme Arduino. Dette viste sig ikke at kunne lade sig gøre, da skærmen og sensoren delte nogle pins, hvilket gjorde at systemet ikke fungerede. Desværre kunne vi ikke bruge andre pins til sensoren, da de to pins der er at vælge imellem begge sad i vejen for skærmen. Derfor valgte gruppen at splitte systemet op og anvende I2C. På grund af denne ekstra arbejsbyrde, blev SD kort implementationen nedprioriteret.

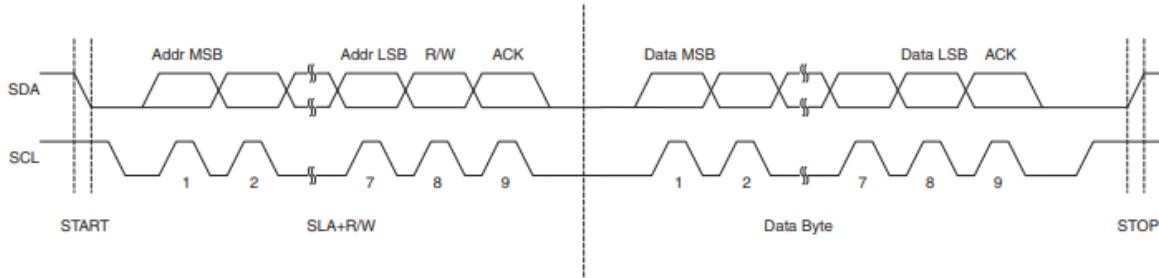
6.1 I2C master

Masteren blev valgt implementeret på TFT display modulet, da det ikke ville give mening at lade sensor modulet sende kommandoer til display modulet. Istedet for sendes der en char fra slaven, som repræsenterer en farve.

Før masteren kan bruges skal den først initiereres. Dette sker ved at sætte bestemte dataregistre op rigtigt. Først bestemmer man hvilken clock SCL skal have. SCL kan udregnes på følgende måde:

$$SCL = \frac{CPU}{16 + 2(TWBR) * 4^{TWPS}} \quad (1)$$

Gruppen har valgt at en clock på 500KHz. De eneste krav til clokken er at den er 16 gange lavere end slavens cpu clock, ifølge megea 2560 datablad. Da slaven kører med 16MHz, opfylder den det krav.



Figur 16: I2C timing diagram

Til udviklingen af masteren har et timing diagram over I2C protokollen, se [Figure 16](#), været utrolig nyttig. Ved hjælp af dette diagram har gruppen kunne sende korrekte beskeder over I2C.

For at have en så overskuelig kode som overhovedet muligt, er der blevet lavet funktioner der kan generere ACK, NACK, WAIT, START og STOP. Disse funktioner er så brugt i vores i2c_master_receive funktion. Et Kodeudsnit af denne funktion kan findes nedenunder:

```

1 i2c_master_start();
2 while(transfer && i < 250)
3 {
4     //uart_transmit(twi_master_status());
5     switch (i2c_master_status())
6     {
7         // Start condition has been transmitted
8         case 0x08:
9             // Contact slave and enter master transmitter mode
10            TWDR = address_r;
11            i2c_master_ack();
12            i2c_master_wait();
13            break;
14
15        // SLA+R has been transmitted and acked
16        case 0x40:
17            i2c_master_nack();
18            i2c_master_wait();
19            break;
20
21        // Data byte has been received and acked
22        case 0x50:
23            //SendChar('A');
24            data = TWDR;
25            transfer = 0;
26            break;
27
28        // Data byte has been received and nacked
29        case 0x58:
30            //SendChar('B');
31            data = TWDR;
32            transfer = 0;
33            break;
34    }
35    i++;
36 }
37 i2c_master_stop();

```

Som det ses i koden bliver i2c_master_status() hele tiden tjekket på, for at finde ud af om slaven reagerer på nogle af kommandoerne fra masteren. Status funktion tjekker TWSR registeret og AND'er det med 0xF8, for at sætte de tre sidste bits til 0. I mega 2560 datablad kan der findes en tabel over hvad de forskellige status koder står for. I kode udsnittet står de som kommentarer over hver case.

6.2 I2C slave

Ligesom masteren skal slave også initialiseres, ved at sætte nogle bestemte dataregistre op. TWAR registeret bestemmer slavens adresse, i vores tilfælde er den sat til 40. TWCR registeret bestemmer hvilken slave mode den skal sættes i, i vores tilfælde er det slave transmitter mode. Det vil sige at slaven kun skal kunne sende data til masteren, og ikke omvendt.

Når slaven er sat op i transmitter mode, bliver der gjort brug af et interrupt, der når i2c interfacet bliver brugt. Når et interrupt bliver triggered, køres vores interrupt rutine, som

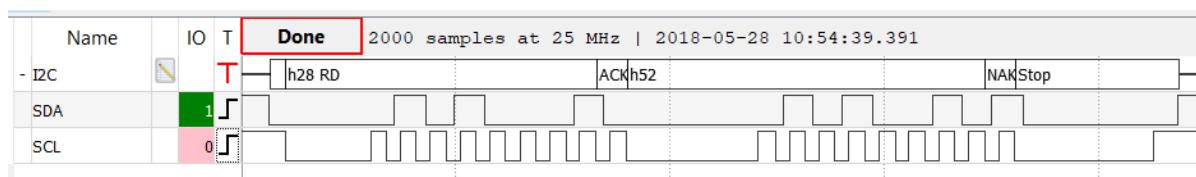
kan ses nedenunder:

```
1 if(i2c_slave_addressed())
2 {
3
4     switch(i2c_slave_status())
5     {
6
7         case 0x60:
8             i2c_slave_ack();
9             break;
10
11        case 0x80:
12            SendChar('A');
13            data = TWDR;
14            i2c_slave_ack();
15            transferring = 0;
16            break;
17
18        case 0xA8:
19            TWDR = dataToSend; //data send to master
20            i2c_slave_ack();
21            break;
22
23            // Last byte sent by master
24        case 0xC0:
25            transferring = 0;
26            i2c_slave_ack();
27            break;
28    }
29 }
```

Interrupt rutinen minder meget om masterens receive funktion. Den tjekker hele tiden på status registeret(TWSR), for at holde styr på hvor langt den er i I2C processen. dataToSend indeholder en char med information om hvilken farve color sensoren har opfanget.

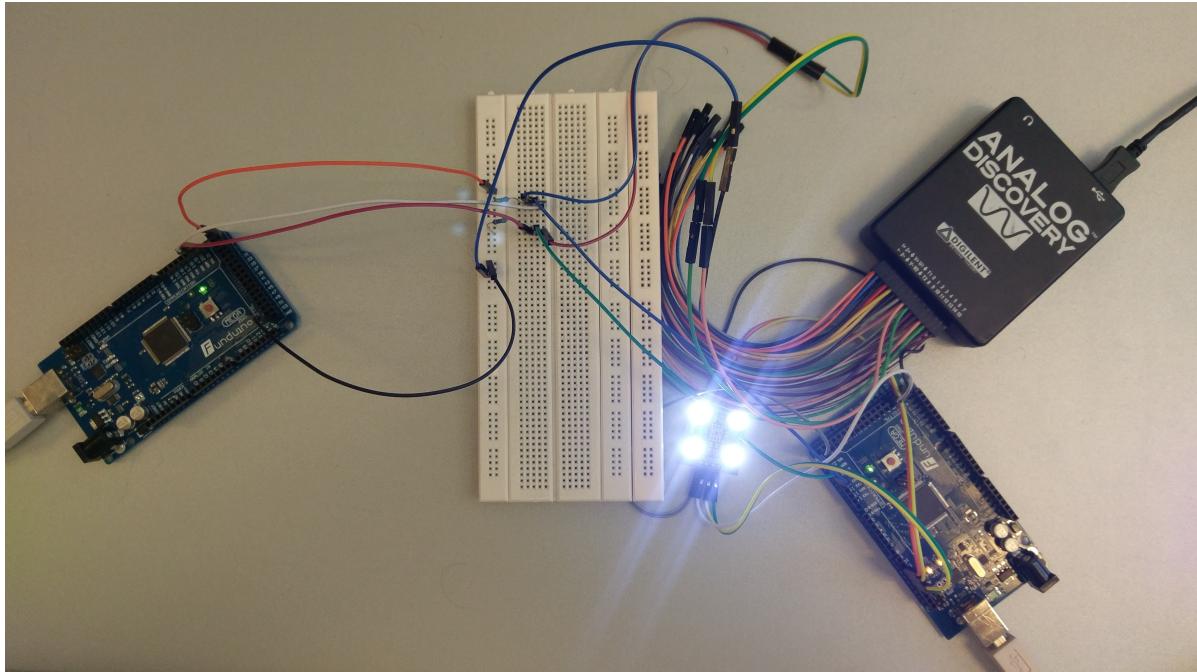
6.3 Test

Til test af I2C, har vi gjort brug af en logic analyzer, som bruges til at afkode beskeder der sende med forskellige protokoller. Det har været en stor hjælp til at finde fejl i vores software. Blandt andet fandt vi ud af at hvis UART og I2C blev brugt samtidig, skabte det en stor nok forsinkelse i I2C protokollen. Dette skyldes at der ikke blev sendt et ACK på det rigtige tidspunkt.



Figur 17: Logic analyzer for I₂C

På figuren ovenfor kan man se en perfekt I2C transmission. På figuren kan man se at et start condition, starter transmissionen, hvorefter adressen på slaven bliver sendt og at R/Wbit bliver sat højt, hvilket betyder READ. Derfter kommer dataen. I dette tilfælde bliver der sendt h52, hvilket svarer i ASCII svarer til bogstavet 'R'. Et NACK kommer bagefter for at signallere at der ikke er mere data, hvorefter et stop condition blever genereret. Testopstillingen kan ses nedenunder.



Figur 18: Test opstilling af I2C

7 Alternative Løsninger

I dette afsnit vil fordele og ulemper for forskellige alternative løsninger blive diskuteret.

7.1 Valg af Kommunikation

Istedet for I2C som kommunikationsprotokol var SPI oplagt. SPI har den fordel at være hurtigere, men kræver 4 ledninger. SPI er hurtigere fordi protokollen kan køre full-duplex, hvorimod I2C er half-duplex. Men da antallet af ledninger var mindre og at vores behov for overførelseshastighed var opfyldt med I2C, valgte vi denne protokol.

Det skal siges at det aldrig var meningen at indbringe to arduinoer, da det ville gøre system mere kompleks uden at give ekstra værdi, men da sensor og skærm ikke kunne komme til at fungere op samme arduino, endte valget på I2C som en løsning på det problem.

7.2 Valg af color sensor

Da projektet ikke vil kunne realiseres uden en color sensor, og at der ikke var andre end LC technology TC3200 sensoren på lager, var den det oplagte valg. Men efter at have arbejdet med den og testet den, fandt vi hurtigt ud af at den ikke var specielt præcis i sin målinger. Derfor vil en sensor af bedre kvalitet have være at foretrække, men som proof of concept fungerer den nuværende sensor fint.

7.3 Valg af skærm

Istedet for en farve-skærm, kunne vi have valgt et alphanumerisk display, men dette vil være en betydelig nedgradering fra den nuværende farve-skærm. Derudover vil det ikke være muligt at vise sjælediagrammer på et alphanumerisk display. Desuden er der indbygget touch i farve-skærmen, som i fremtiden vil kunne inkorporeres, hvis dette ønskes.

8 Konklussion

Litteratur

- [1] ATMEL. *ATmega 2560 datasheet kapitel 17*, 1.0 edition, Juli 2014.
- [2] ILITEX. *a-Si TFT LCD Single Chip Driver 240RGBx320 Resolution and 262K color*, Juni 2011.
- [3] TAOS. *Programmable color light-to-frequency converter datasheet*, 1.0 edition, Juli 2009.