

AARHUS UNIVERSITET

ANVENDTE MICROCONTROLLER SYSTEMER

6. SEMESTER

Color Sorting System

Gruppemedlemmer:

Daniel Tøttrup

Stinus Lykke Skovgaard

AUID

au544366

au520659



AARHUS
UNIVERSITY

SCHOOL OF ENGINEERING

27. maj 2018

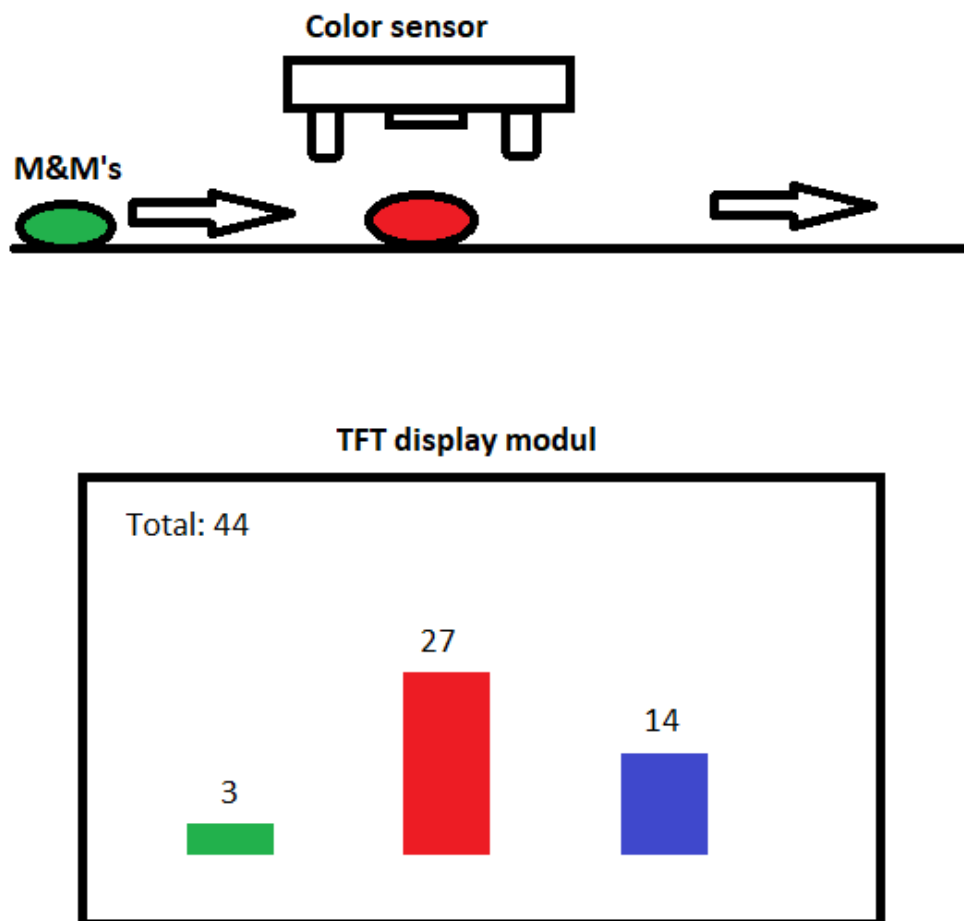
Indhold

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	12
5.1	Hardware	12
5.2	Software	12
6	I2C kommunikation	16
6.1	I2C master	16
6.2	I2C slave	17
6.3	Test	18
7	Konklusion	19

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	MCU-Interface Mode	12
11	WriteCommand kode	13
12	Tidsforsinkelser	13
13	WriteData kode	14
14	DisplayInit kode	15
15	I2C timing diagram	16

1 Indledning



Figur 1: Konceptbillede for CSS

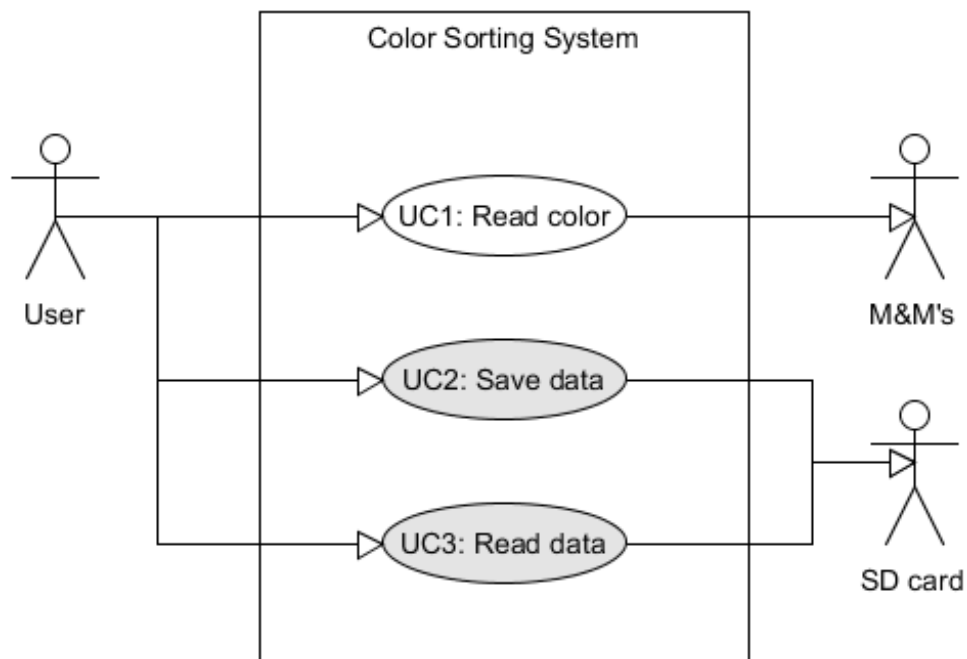
Color Sorting System(CSS) gør det nemt at sortere diverse emner baseret på deres farve. Disse emner kan være alt fra fødevarer til maskin-komponenter, så længe de kan identificeres på farve. CSS fungerer ved at lade emnet passere under en farve sensor, som kan identificere farven på emnet og videresende 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 sensoren. 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 modulet, 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. Uscase 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

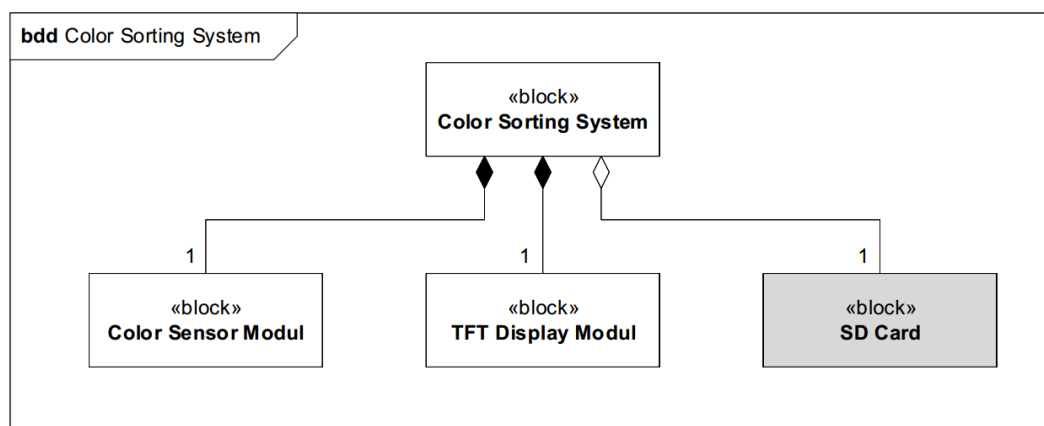
Det skal sige at usecase 1 og 2 ikke er implementeret i denne prototype. Der er gjort forsøg på at få dem implementeret, men grundet tidspres fik de Derudover var det også tænkt fra start, at der ikke skulle være behov for brugerinput for at kunne aflæse farve, men på grund af tidsmangel fik gruppen 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 modtagne data, antallet af hver enkelt farve målt, samt det totale antal farver. Dette modul skulle også have haft ansvaret for at sende data til SD-kortet, hvis den del var blevet implementeret.

Color Sensor Module:

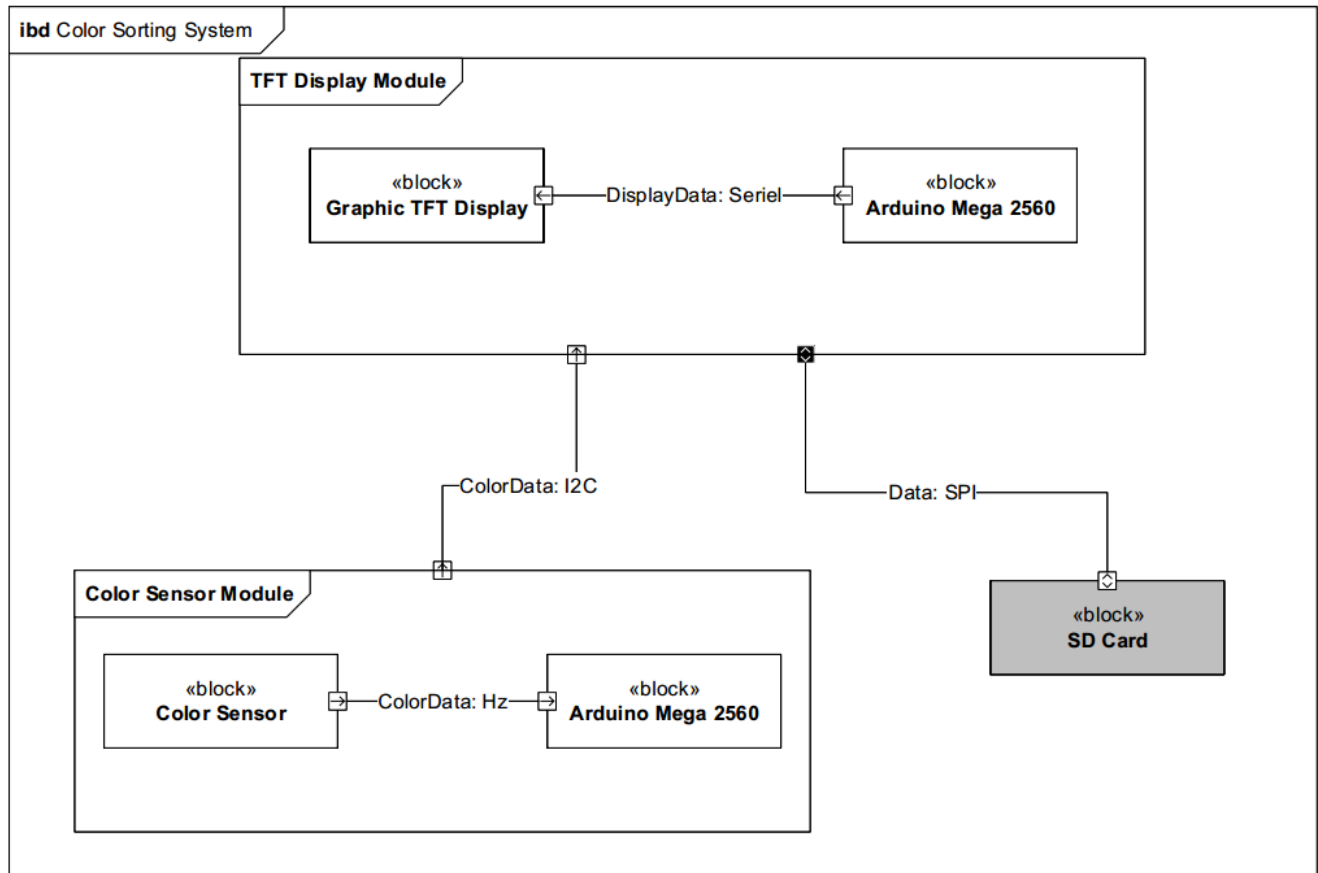
Har til opgave at måle farven placeret under Color Sensoren, samt sende informationen videre til TFT Display Module.

SD-Kort:

Et SD kort som var tiltænkt at kunne opbevare data som efter systemet slukkes. Dette blev dog taget ud af system. 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 blå 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. Se [Figure 5](#)

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

Figur 5: Styling 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).

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 μW/cm ² , λ _p = 470 nm	12.5	15.6	18.7	61%		84%	22%		43%	0%		6%	kHz
		(4.7)	(5.85)	(7)										
	E _e = 40.4 μW/cm ² , λ _p = 524 nm	12.5	15.6	18.7	8%		28%	57%		80%	9%		27%	
		(4.7)	(5.85)	(7)										
	E _e = 34.6 μW/cm ² , λ _p = 640 nm	13.1	16.4	19.7	5%		21%	0%		12%	84%		105%	
		(4.9)	(6.15)	(7.4)										

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

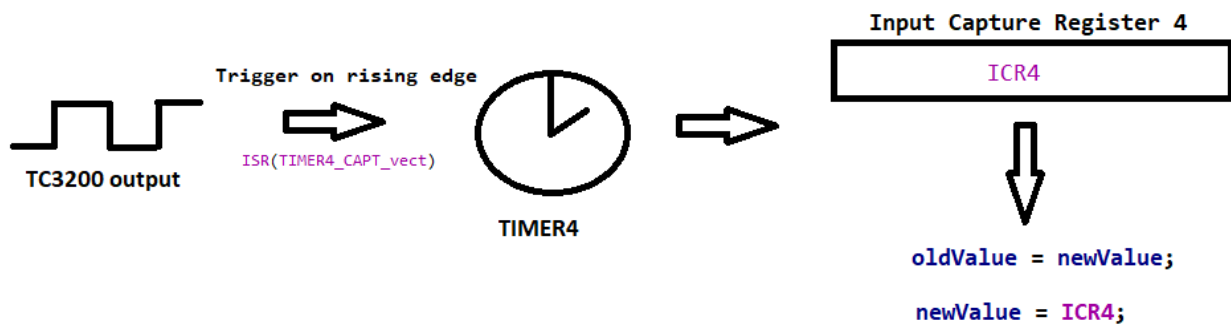
at sætte S1 og S2 høj eller lav. Til CSS bruges 2% skalering, da 100% skalering, ville betyde at output frekvensen kunne komme op over 50kHz. En lavere frekvens vil være fordelagtig i forhold til at få en præcis måling. Se [Figure 7](#)

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 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 risikere at en måling ikke vil give en korrekt frekvens. For at udbedre dette problem bruges en anden interrupt, `ISR(TIM4_OVF_vect)`. Den trigger hver gang der kommer overflow, og 65535 lægges til `newValue`. Koden til hvordan frekvensen udregnes kan ses nedenunder.

```

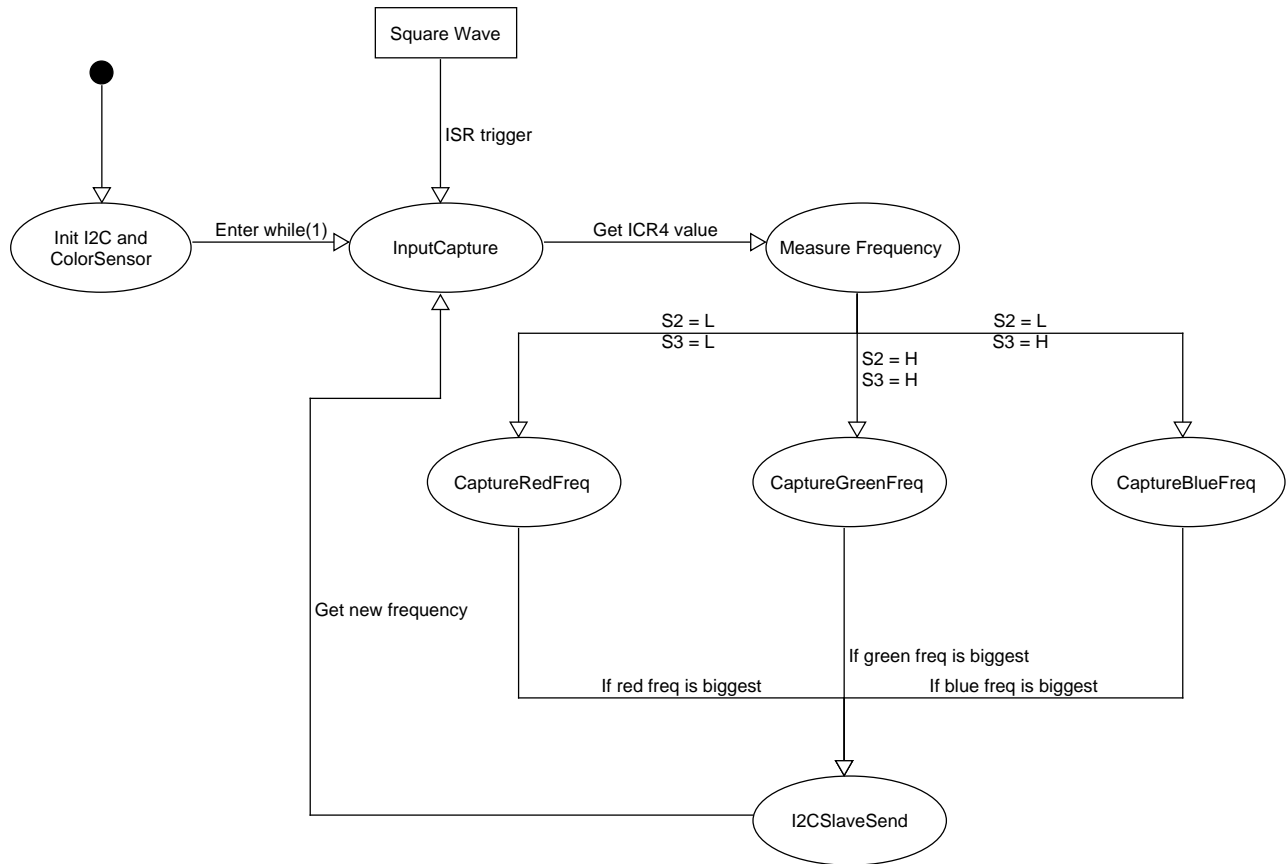
1  ISR(TIMER4_CAPT_vect)
2  {
3      oldValue = newValue;
4      newValue = ICR4;
5
6      if(newValue < oldValue)
7      {
8          period = oldValue-newValue;
9      }
10     else
11     {
12         newValue + overflow;
13         period = oldValue - newValue;
14     }
15     freq = F_CPU/period;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

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. Dette skyldes højst sandsynligt at farven ikke var den samme som databladet brugte til at teste med($\lambda = 524nm$). Udover den grønne farve ikke passede helt, kan sensoren stadig skelne imellem de forskellige farver.

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 funktionalitet, 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 10: 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 på figur [Figure 11](#) nedenfor

```
void WriteCommand(unsigned int command){
    // Data set up (commands only use lower byte of the data bus)
    DATA_PORT_LOW = command;
    // DCX low
    // CSX low
    // WRX low
    DC_PORT &= ~(1<<DC_BIT);
    CS_PORT &= ~(1<<CS_BIT);
    WR_PORT &= ~(1<<WR_BIT);
    // twrl > 15ns
    _NOP();
    //WRX high
    WR_PORT |= (1<<WR_BIT);
    // twrh > 15ns
    _NOP();
}
```

Figur 11: WriteCommand kode

Som det ses på figur [Figure 10](#) fra databadet skal DCX og CSX sættes lavt, samt trigger kommandoen på WRX stigende flanke, derfor sættes WRX lav til at starte med. På figur [Figure 12](#) 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 MCU 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	tchwh	CSX "H" pulse width	0	-	ns	
	tcs	Chip Select setup time (Write)	15	-	ns	
	trcs	Chip Select setup time (Read ID)	45	-	ns	
	trcsfm	Chip Select setup time (Read FM)	355	-	ns	
	tcsf	Chip Select Wait time (Write/Read)	10	-	ns	
	twc	Write cycle	66	-	ns	
WRX	twrh	Write Control pulse H duration	15	-	ns	
	twrl	Write Control pulse L duration	15	-	ns	
	trcfm	Read Cycle (FM)	450	-	ns	
RDX (FM)	trdhfm	Read Control H duration (FM)	90	-	ns	
	trdlfm	Read Control L duration (FM)	355	-	ns	
	trc	Read cycle (ID)	160	-	ns	
RDX (ID)	trdh	Read Control pulse H duration	90	-	ns	
	trdl	Read Control pulse L duration	45	-	ns	
	tdst	Write data setup time	10	-	ns	
D[17:0], D[15:0], D[8:0], D[7:0]	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	
						For maximum CL=30pF For minimum CL=8pF

Figur 12: Tidsforsinkelser

Dernæst implementerede vi WriteData, som tilnærmelsesvis ligner WriteCommand bortset fra at DCX skal sættes høj i stedet for lav. Koden for WriteData ses nedenfor på figur [Figure 13](#). 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.

```
void WriteData(unsigned int data){
    // Data set up (commands only use lower byte of the data bus)
    DATA_PORT_HIGH = data >> 8;
    DATA_PORT_LOW = data;
    // DCX high
    // CSX low
    // WRX low
    DC_PORT |= (1<<DC_BIT);
    CS_PORT &= ~(1<<CS_BIT);
    WR_PORT &= ~(1<<WR_BIT);
    // twrl > 15ns
    _NOP();
    //WR high
    WR_PORT |= (1<<WR_BIT);
    // twrh > 15ns
    _NOP();
}
```

Figur 13: WriteData kode

I vores DisplayInit() som vi kalder en gang i koden til at Initialisere displayet, starter vi med at sætte vores Control Pins som output 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.

```

DisplayInit(){
    //Sets the control pins as output and set them high
    //Sets data pins as output
    DDRC |= 0b00000111;
    DDRD |= 0b10000000;
    DDRA = 0xFF;
    DDRC = 0xFF;
    PORTG |= 0b00000111;
    PORTD |= 0b10000000;

    //Reset the display (RST low)
    RST_PORT &= ~(1<<RST_BIT);
    //Wait 300 ms
    _delay_ms(300);
    // RST high
    RST_PORT |= (1<<RST_BIT);
    _delay_ms(130);

    // Exit sleep mode.
    SleepOut();
    // Set display on
    DisplayOn();
    // set bit BGR (scanning direction)
    MemoryAccessControl(0b00001000);
    //16 bits (2bytes) per pixel
    InterfacePixelFormat(0b00000101);
}

```

Figur 14: DisplayInit kode

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 arbejdsbyrde, blev SD kort implementationen nedprioriteret.

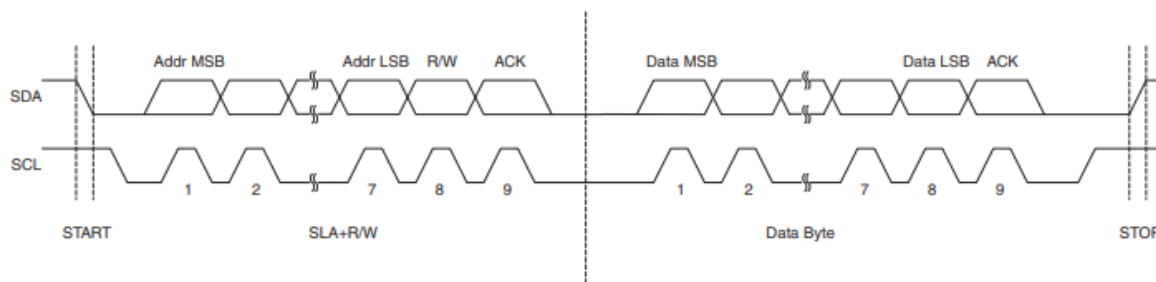
6.1 I2C master

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 initieres. 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 klokken 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 15: I2C timing diagram

Til udviklingen af masteren har et timing diagram over I2C protokollen, se [Figure 15](#), 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     //usart_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 triggeret, 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

Billede af logic analyzer! og forklarende tekst

7 Konklusion

Litteratur

Litteratur