

AARHUS UNIVERSITET

ANVENDTE MICROCONTROLLER SYSTEMER

6. SEMESTER

---

# Color Sorting System

---

*Gruppemedlemmer:*

Daniel Tøttrup

Stinus Lykke Skovgaard

*AUID*

au544366

au520659



AARHUS  
UNIVERSITY

SCHOOL OF ENGINEERING

26. maj 2018

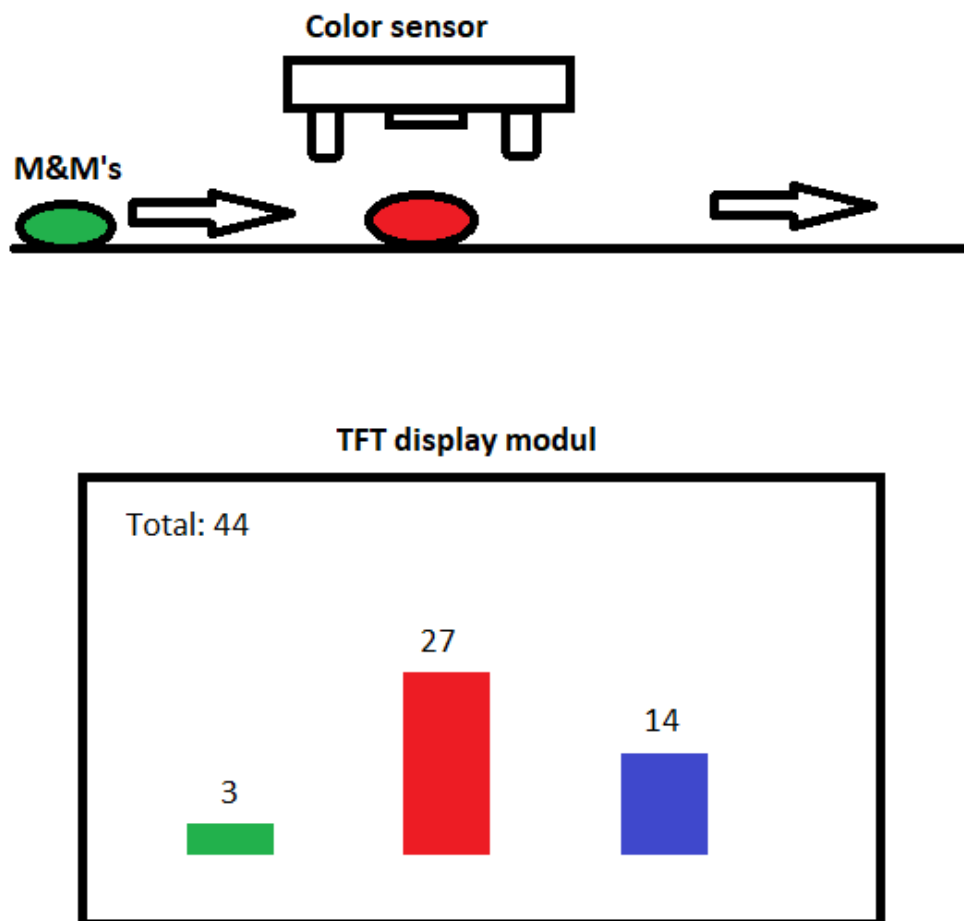
# Indhold

|          |                                 |           |
|----------|---------------------------------|-----------|
| <b>1</b> | <b>Indledning</b>               | <b>3</b>  |
| <b>2</b> | <b>Krav</b>                     | <b>4</b>  |
| 2.1      | UC1 - Read Color . . . . .      | 5         |
| 2.2      | UC2 - Save Data . . . . .       | 5         |
| 2.3      | UC3 - Read Data . . . . .       | 5         |
| 2.4      | Afgrænsning . . . . .           | 5         |
| <b>3</b> | <b>Color Sensor Module</b>      | <b>6</b>  |
| 3.1      | LC Technology TCS3200 . . . . . | 6         |
| 3.2      | Input Capture . . . . .         | 7         |
| <b>4</b> | <b>TFT Display Module</b>       | <b>9</b>  |
| 4.1      | Hardware . . . . .              | 9         |
| 4.2      | Software . . . . .              | 9         |
| <b>5</b> | <b>Test</b>                     | <b>12</b> |
| <b>6</b> | <b>Diskussion</b>               | <b>13</b> |
| <b>7</b> | <b>Konklusion</b>               | <b>14</b> |

## Figurer

|    |  |    |
|----|--|----|
| 1  | Konceptbillede for CSS . . . . .                 | 3  |
| 2  | Usecase diagram for CSS . . . . .                | 4  |
| 3  | Styring af photodioder . . . . .                 | 6  |
| 4  | Output frekvens ved forskellige farver . . . . . | 6  |
| 5  | Output frekvens skalering . . . . .              | 7  |
| 6  | Input Capture illustration . . . . .             | 7  |
| 7  | MCU-Interface Mode . . . . .                     | 9  |
| 8  | WriteCommand kode . . . . .                      | 10 |
| 9  | Tidsforsinkelser . . . . .                       | 10 |
| 10 | WriteData kode . . . . .                         | 11 |

# 1 Indledning



Figur 1: Konceptbillede for CSS

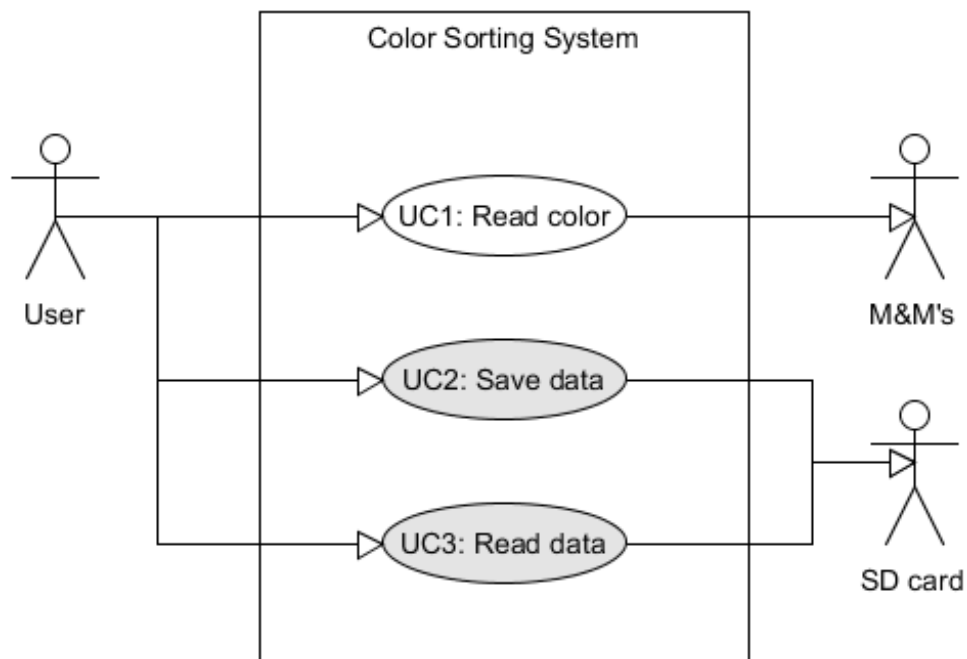
Color Sorting System(CSS) gør det nemt at sortere diverse emner baseret på deres farve. Disse emner kan være alt fra fødevarer til maskin-komponenter, så længe de kan identificeres på farve. CSS fungerer ved at lade emnet passere under en farve sensor, som kan identificere farven på emnet og videresende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

#### 3.1 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 3](#)

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

Figur 3: 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 4](#) kan man se hvordan de forskellige fotodioder reagerer ved forskellige bølgelængder(farver).

| PARAMETER   | TEST CONDITIONS   | CLEAR<br>PHOTODIODE<br>S2 = H, S3 = L |                |               | BLUE<br>PHOTODIODE<br>S2 = L, S3 = H |     |     | GREEN<br>PHOTODIODE<br>S2 = H, S3 = H |     |     | RED<br>PHOTODIODE<br>S2 = L, S3 = L |     |      | UNIT |
|---|---|---------------------------------------|----------------|---------------|--------------------------------------|-----|-----|---------------------------------------|-----|-----|-------------------------------------|-----|------|------|
|   |   | MIN                                   | TYP            | MAX           | MIN                                  | TYP | MAX | MIN                                   | TYP | MAX | MIN                                 | TYP | MAX  |      |
| f <sub>O</sub><br>Output<br>frequency<br>(Note 9) | E <sub>e</sub> = 47.2 μW/cm <sup>2</sup> ,<br>λ <sub>p</sub> = 470 nm | 12.5<br>(4.7)                         | 15.6<br>(5.85) | 18.7<br>(7)   | 61%                                  |     | 84% | 22%                                   |     | 43% | 0%                                  |     | 6%   | kHz  |
|   | E <sub>e</sub> = 40.4 μW/cm <sup>2</sup> ,<br>λ <sub>p</sub> = 524 nm | 12.5<br>(4.7)                         | 15.6<br>(5.85) | 18.7<br>(7)   | 8%                                   |     | 28% | 57%                                   |     | 80% | 9%                                  |     | 27%  |      |
|   | E <sub>e</sub> = 34.6 μW/cm <sup>2</sup> ,<br>λ <sub>p</sub> = 640 nm | 13.1<br>(4.9)                         | 16.4<br>(6.15) | 19.7<br>(7.4) | 5%                                   |     | 21% | 0%                                    |     | 12% | 84%                                 |     | 105% |      |

Figur 4: 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

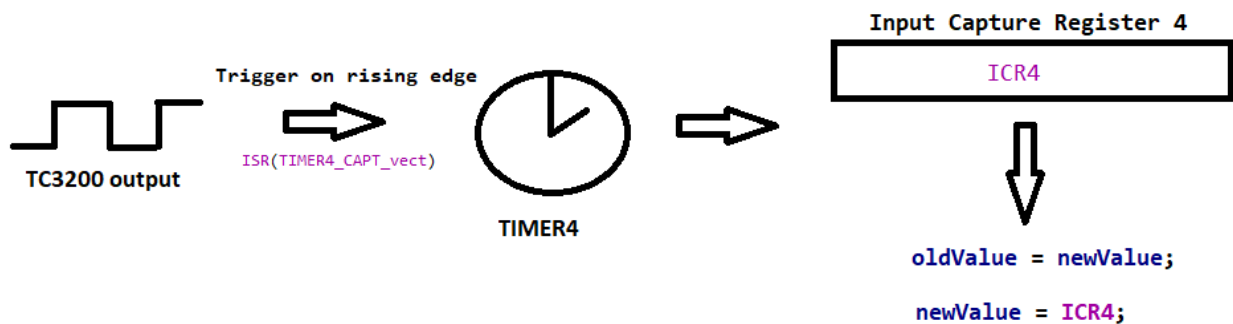
at sætte S1 og S2 høj eller lav. Til CSS bruges 2% skalering, da 100% skalering, ville betyde at output frekvensen kunne komme op over 50kHz. En lavere frekvens vil være fordelagtig i forhold til at få en præcis måling. Se [Figure 5](#)

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

Figur 5: Output frekvens skalering

## 3.2 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 6: 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 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.

### 4.1 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.

### 4.2 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 1  | "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 1 | "L"            |  | "H"  | "L"            | Write command code.              |
|     |     |     |     |                                 | <del>"L"</del> | <del>"H"</del>  | <del></del> | <del>"H"</del> | Read internal status.            |
|     |     |     |     |                                 | "L"            |  | "H"  | "H"            | Write parameter or display data. |
|     |     |     |     |                                 | <del>"L"</del> | <del>"H"</del>  | <del></del> | <del>"H"</del> | Reads parameter or display data. |

Figur 7: 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 8](#) 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 8: WriteCommand kode

Som det ses på figur [Figure 7](#) 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 9](#) 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],<br>D[15:0],<br>D[8:0],<br>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   |             |

Figur 9: 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 10. 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 10: WriteData kode

## 5 Test

## 6 Diskussion

## 7 Konklusion

**Litteratur**

**Litteratur**