**GFV Forår 2019**

# Laboratorieøvelse # 2
# Kommunikationsbusser

**Dato: 13/03/2019**

**Gruppe 21**


| |
|---|
| ***#1***<br>Stud.nr.: 201404212 Navn: Rasmus Vesterheim |
| ***#2***<br>Stud.nr.: 201710712 Navn: Marcus Bech |
| ***#3***<br>Stud.nr.: 201710685 Navn: Martin Krøjmand |

# Indledning

Formålet med denne øvelse er at blive bedre bekendt med PSoC igennem praktiske deløvelser, hvor vi skal ind og kigge på kommunikationsprotokollerne I2C og SPI. Der startes først med at måle temperaturen af LM75 med I2C og skrive det ud på terminalen. Herefter kommunikeres der mellem 2 PSoC's via SPI med henholdsvis aktivering af LED og sende status af Switch.

# 1. I2C Experiment: PSoC Master and LM75 Slave

## Indledning

I dette eksperiment vil vi kommunikere ved brug af protokollen I2C, hvor vi vil bruge en PSoC som master og to temperatursensorer (LM75) som slaver. I2C slaverne skal have forskellige adresser, så man kan vælge en af dem, man vil have data fra, og på det udleverede print er de derfor sat sammen med tre switches til styring af de sidste 3 bits af adressen.
På figuren nedenunder ses hvordan en besked bliver sendt ved brug af I2C-protokollen. Vi gør i denne opgave brug af 7 bits til at bestemme adressen.

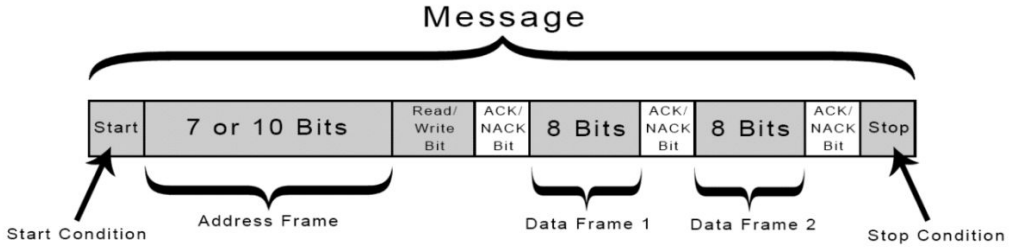



*Figure 1 Skabelon af I2C-besked, der starter med, at masteren trækker SDA lav, hvorefter vi bruger syv bits til at bestemme adressen på slave og slutter af med at sende '1', hvis slaven skal skrive, og '0', hvis slaven skal læse.*

# Design og implementering

På figuren nedenunder ses design af kredsløbet, hvor der er tilsluttet to LM75 slaver.

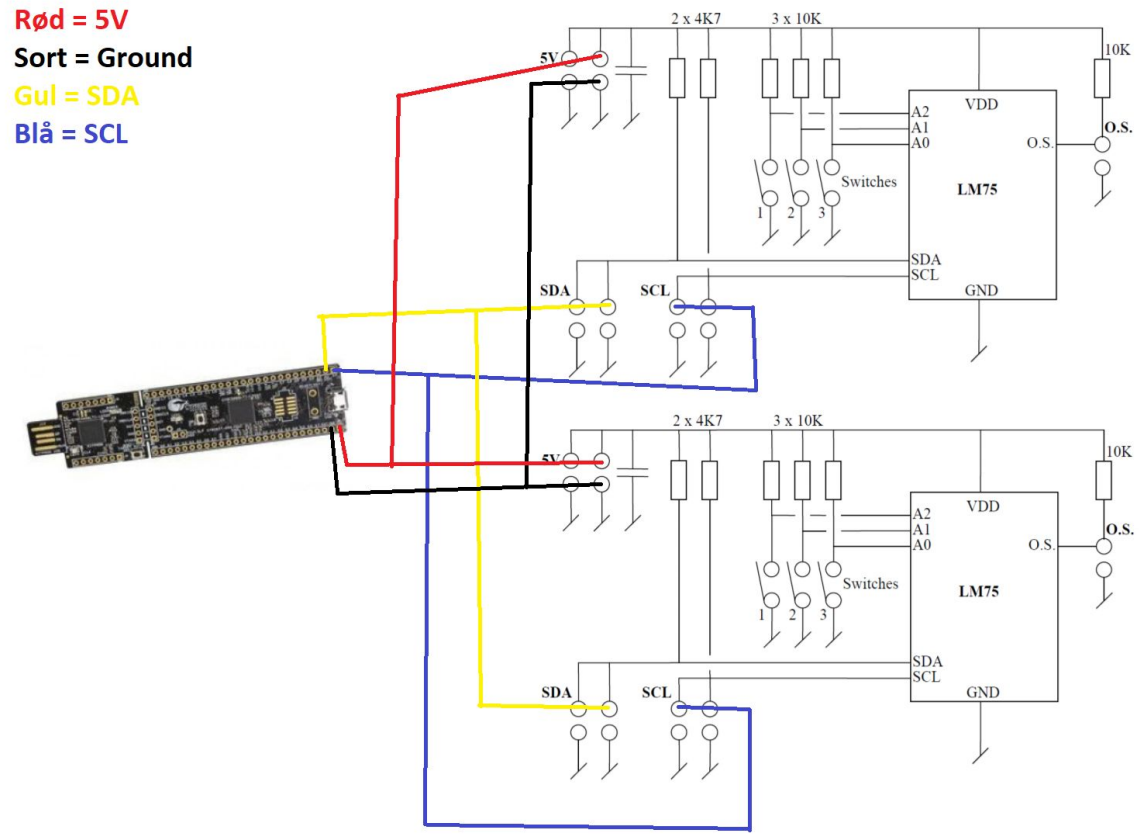
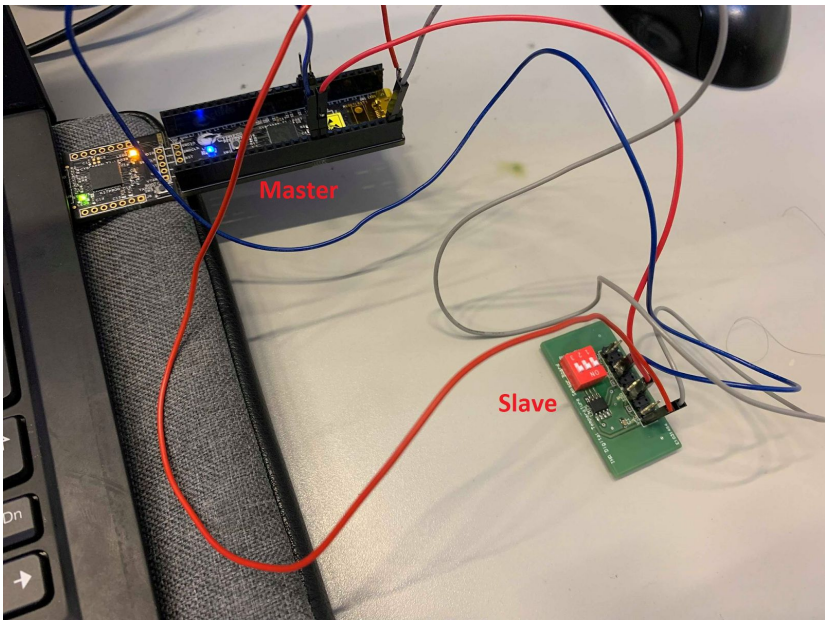



*Figure 2 Billede af, hvordan PSoC (Master) er sat til de 2 temperatursensor prints (Slaver)*

Som det ses på figuren ovenover behøver vi kun to ledninger mellem master og slaverne. SCL er styring af clocken, som master styrer, og ifølge databladet skal denne være minimum 2.5 mikrosekunder, hvilket svarer til et maksimal frekvens på 400k Hz. Vi har valgt vores frekvens til at være 100 kHz, som altså er 100 kBit/s.

Målet med dette eksperiment er at kunne læse fra de to slaver på skift ved blot at ændre adressen, som vi sender afsted fra masteren, når vi beder om at få tilsendt data. For at kunne verificere, at de to slaver rent faktisk på skift sender deres data, kan man gøre brug af en modstand, der bliver varmet op, eller blot ånde på den ene temperatursensor, hvorved den burde blive varmere.
Opsætningen af eksperimentet med en enkelt slave sat til ses nedenunder.



,

*Figure 3 Forsøgsopstilling af en PSoC (Master) med et temperatur print (Slave)*

## 2's komplement konvertering

For at konvertere de tilsendte bytes fra 2's komplement til decimaltal kan de to bytes først lægges sammen i en 16 bit integer med den første byte som den mest betydende byte. Herefter inverteres alle 16 bits, hvorefter vi kan shifte bitsene syv gange til højre, da temperaturen kun er indeholdt i de 9 mest betydende bits. Til sidst skal man huske at lægge en til, for at fuldføre konverteringen.
I vores kode har vi imidlertid ikke gjort brug af denne metode, da vi mente, at det ville være lettere blot at kigge på, om MSB i den mindste byte var 1 eller 0. Hvis den var 1, skulle der lægges 0.5 til temperaturen, og hvis den var 0, skulle der ikke lægges en halv grad til.

## Ændring af delay

Der er i koden et delay efter hver gang, der bliver aflæst en temperatur, konverteret og til sidst skrevet ud i terminalen. Dette er i koden som standard sat til 1000 millisekunder for at være sikker på, at LM75 har målt en ny temperatur og gemt værdien digitalt i dens registre. Denne konvertering er det, der hedder conversion time. Dette skal være klart, inden der læses igen via I2C for at få et rigtigt resultat. I databladet kan conversion time ses til at være 100 ms

| | | | |
|---|---|---|---|
| Temperature Conversion Time | See[5] | 100 | ms |
| | See[5], $-55°C \le T_A \le 125°C$ | 300 | |

*Figure 4 Billede af datasheet for LM75, hvor conversion time er angivet*

Der er forsøgt at sænke dette delay for at se, om det har en virkning. Selvom delay blev langsomt sænket gav det ikke noget anderledes resultat. Til sidst blev delayet fjernet helt, og der kom stadig tilsyneladende realistiske tal ud.

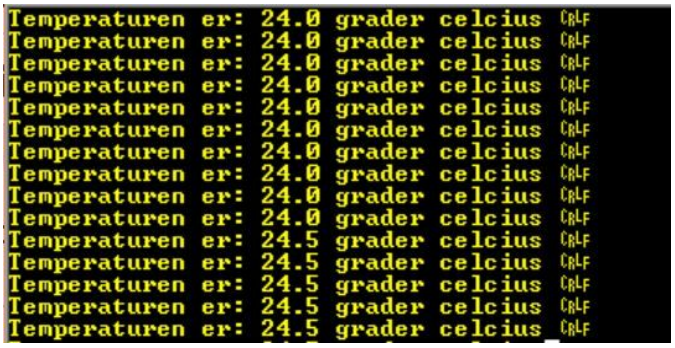



*Figure 5 Billede af test med I2C kommunikation uden delay i bunden af løkken*

Det kan derfor konkluderes, at delayet ikke har nogen betydning for at få den rigtige temperatur. Det har nok noget at gøre med, at vi tjekker status ved start af transmission fra masteren. Derfor vil vi ikke komme videre, hvis der er fejl. Og der ventes derfor til, at status er I2C_MSTR_NO_ERROR. I hvert fald vil vi fortsat med at overholde conversion time for LM75.

## I2C med to slaver

For at tilføje en ekstra slave trækkes blot to ledninger fra SCL og SDA på det første slave print til disse indgange på den anden slave samt to ledninger til hhv. VCC og ground. Vi valgte dog lige at samle forbindelserne på fumlebræt.

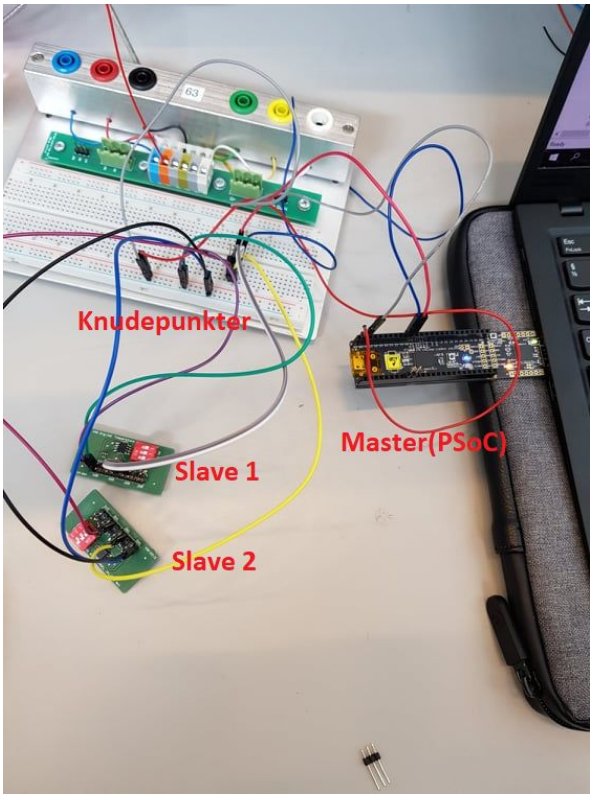



*Figure 6 Billede af opstilling med en PSoC (Master) med 2 temperatursensor-prints (Slaver). Læg mærke til knudepunktet på Breadboardet, hvor de 2 slaver sikres at være forbundet til samme bus med data og clock.*

Kodemæssigt for at understøtte denne udvidelse, implementerede vi en counter funktion. Her angav vi at x(counter variablen som tæller op) modulo 2, skulle udskrive den ene slave adresse. Hvis så at det gav 0, så skulle den udskrive den anden slave adresse, herved sikrede vi os at den skifter mellem de to adresser hver gang den tæller op. Til slut tilføjes counter++ funktionen, som tæller variablen én op.

## Målinger med diskussion

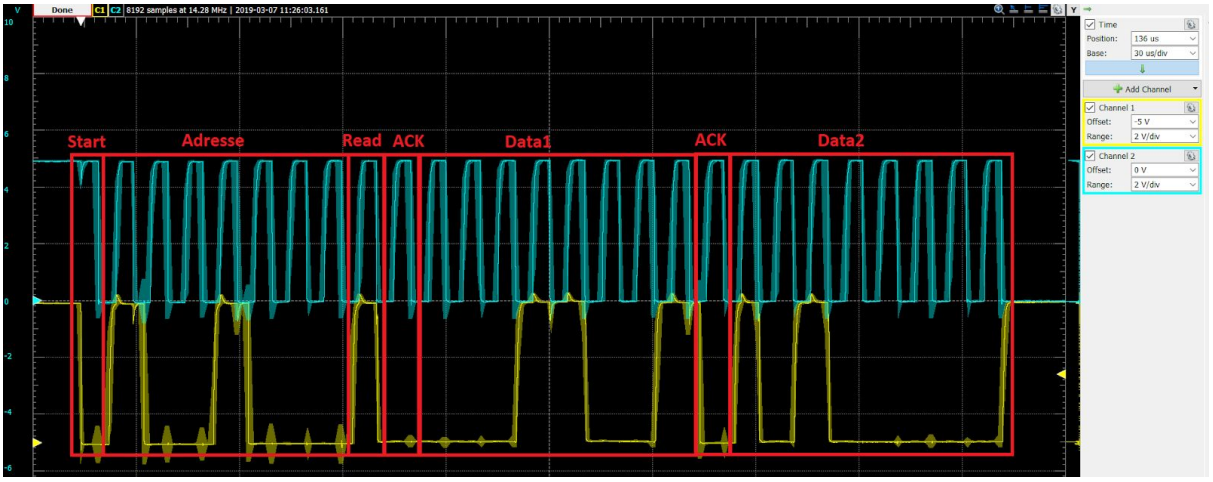



*Figure 7 Billede af scoopet for den ene temperatursensor LM75. Det blå scope er clocken SCL og det gule scope er SDA.*

På billedet på figur 7 ses clocken (SCL) foroven med det blå scope, og data i SDA ses i bunden med det gule scope.
Der kan altså ses på oscilloskopet, at der startes med et start bit. Herefter sendes adressen på 0x48 (1001000), som vises i adressefeltet på figur 7. Herefter er et bit sat højt, som indikerer at vi læser temperaturen fra LM75. Herefter trækkes en bit lav fra Slavens side, der indikerer et ACK (acknowledge). Hvis det lykkedes sendes de næste 8 bit. Hvis ikke; sendes de samme 8 bits igen, eller Masteren forsøger at stoppe transmissionen. Herefter sendes 8 bits med det mest betydende byte. Et bit trækkes lav ved ACK. Herefter sendes det mindst betydende byte.
Der kan herefter anes på oscilloscope-billedet på figur 7, at der trækkes en bit højt på SDA (gult scope), der står for vellykket NACK (not acknowledge). Den er omvendt, så masteren ved,  hvornår der ikke er flere bytes, der skal sendes.
Herefter sendes et stop-bit, der desværre ikke kan ses på oscilloskopet. Der blev forsøgt at finde frem til det sidste bit for stop, men det kunne ikke lade sig gøre. Vi kan kun finde frem til, at stopbit'et må være sendt, men det kunne ikke ses på målingen. SDA skulle gerne lave en rising edge, mens SCL er højt for at indikere stop.

# Resultater

Her ses temperaturen for hver af de 2 temperatursensor prints hver anden gang, så vi ser forskellen mellem de to print, i terminalen, når den ene er varmet op.

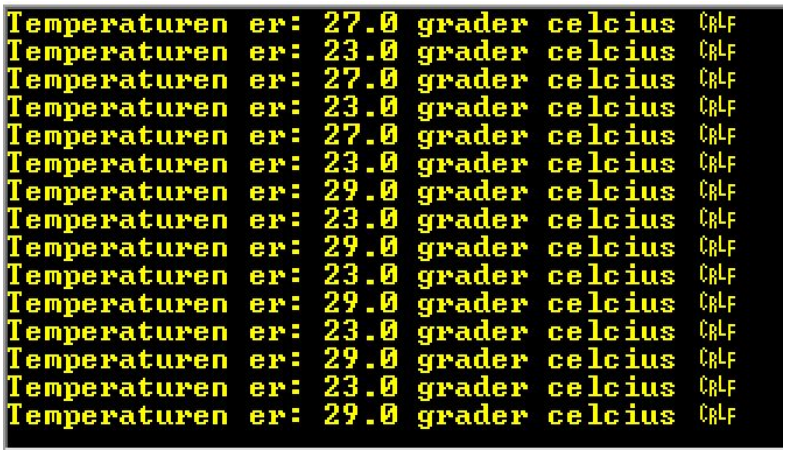



*Figure 8 Billede af temperaturen i Celsius for de 2 prints. Hver prints temperatur skrives ud hver anden linje. Det ses ret tydeligt, at temperaturen på det ene print er en del højere end på den anden.*

# 2. SPI Experiment: PSoC Master and PSoC Slave

## Indledning

I denne del af laboratorieøvelsen skal der kommunikers mellem 2 PSoC's via kommunikationsprotokollen SPI. Den ene er sat som master, som derfor styrer clocken, og den anden er slaven, som modtager den clock, masteren sender ud.
Der skal et program til både master PSoC og slave PSoC. Master modtager en karakter fra PC'en via UART. Dette bliver sendt videre til slaven via SPI, som modtager karakteren og henholdsvis tænder og slukker en LED på PSoC afhængig af den sendte karakter fra slaven. Eftersom SPI er full-duplex sender slaven en karakter tilbage til masteren via SPI, som viser status af switch-knappen på slaven. Denne status sender videre til terminalen på PC'en via UART.

## Teori

Kommunikationen mellem 2 enheder såsom PSoC's fungerer med Master og Slave ligesom i2C. Masteren generer clock-signalet sclk, som slaven følger for at shifte på samme tid som masteren. Hver SPI-enhed har typisk et 8-bit register, der shiftes ved rising edge eller falling edge afhængig af, hvilken Clock Phase (CPHA), der benyttes.
Der sendes og modtages et bit samme tidspunkt. Derfor er SPI full duplex. Man er nødt til at sende noget for at modtaget noget og omvendt.
Clocken polariteten (CPOL) kan være non-inverted og inverteret. Forskellen kan blot ses ved, om clocken starter med at være lav (non-inverted), eller om den starter med at være høj (inverteret).

# Opstilling

Her ses opstillingen af SPI kommunikationen

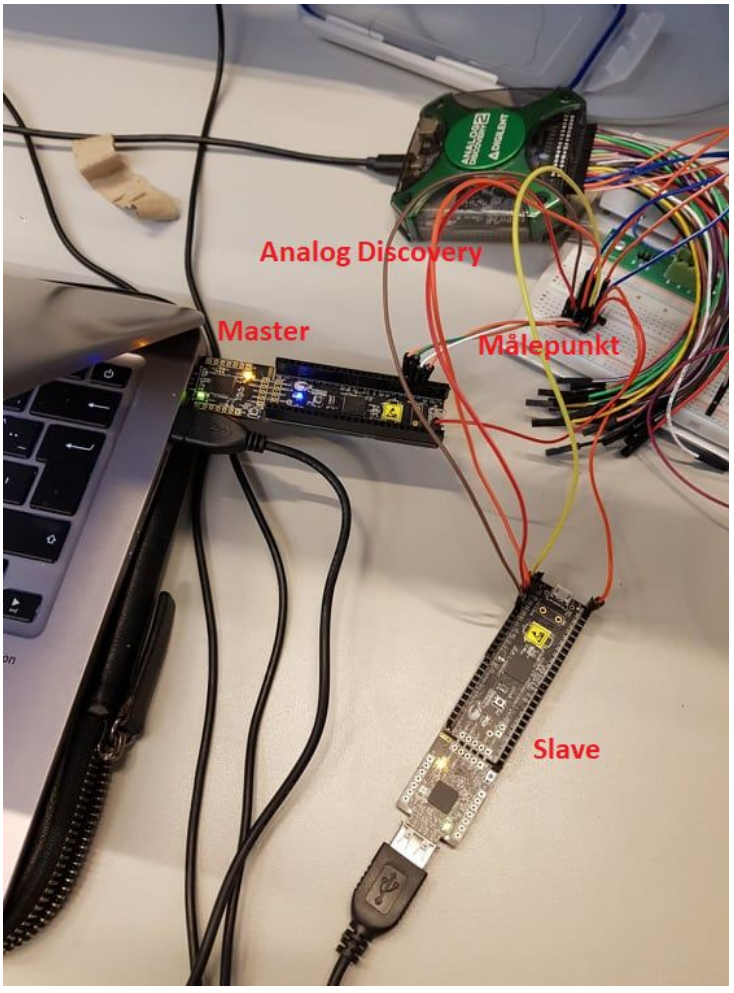



*Figure 9 Billede af opstilling af SPI kommunikation af de to PSoC's, hvor den ene er Master (sat til PC via USB UART), og den anden er Slave.*

Forbindelserne er som udgangspunkt lavet således

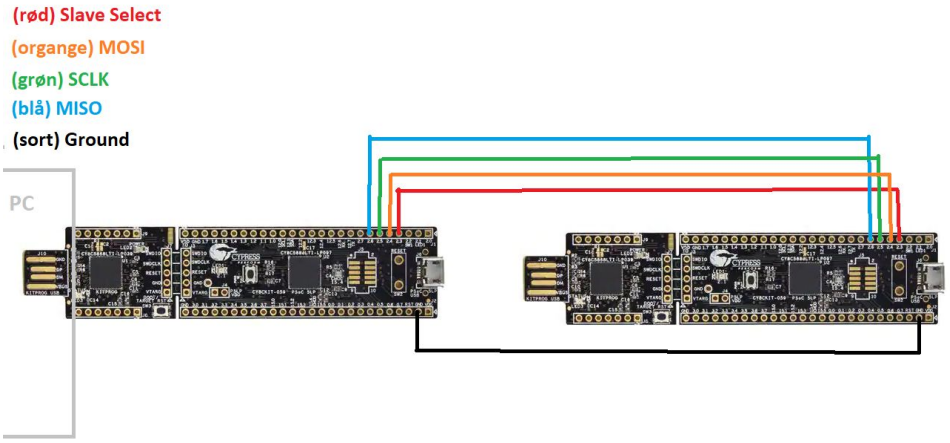



*Figure 10 Billederne af forbindelserne mellem Master og Slave for SPI kommunikationen. Master er sat til PC via UART.*

# Implementering

SPI kommunikationen mellem de 2 PSoC's er implementeret via 2 projekter, hvor den ene er et masterprojekt, og den anden er en slave projekt. I TopDesign for både Master projektet og Slave projektet er der blevet sat en komponent i hver af henholdsvis SPI Master og SPI Slave.

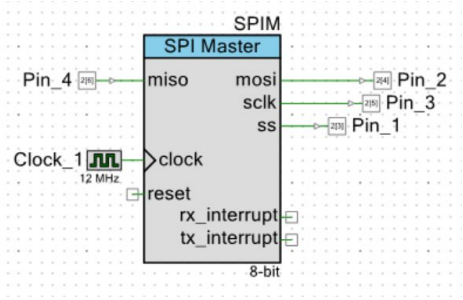



*Figure 11 Billede af Master komponent*

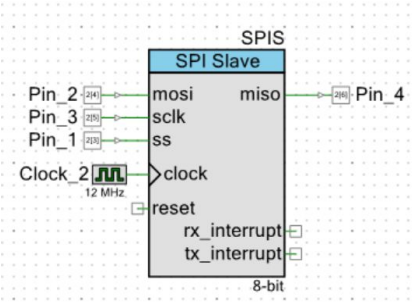



*Figure 12 Billede af Slave komponent*

## Slaven

I Slaven sendes der en karakter '1' tilbage til Master, når der ikke trykkes på knappen. Derimod sendes der karakteren '0' tilbage, når de er trykket på knappen. Dette kan blive sendt ved at tjekke det digitale Pin på P2_2, der er koblet til switchen

```
if(Pin_LN1_Read()) {
    SPIS_WriteTxData('1');
} else {
    SPIS_WriteTxData('0');
}
```

*Figure 13 Billede af kode med læsning af Switch*

Herefter tjekkes der, om der er data klar. Hvis der er, modtages dette, og der tjekkes, om det er en karakter '1' eller '0'. Hvis der er modtaget karakteren '1', så tændes LED på Slaven. Hvis ikke, tændes der ikke. Hvis man vil have koblet til UART, så kan det læses på terminalen, men det er der ikke i dette forsøg. Derfor går programmet videre til at clear bufferen for sikker transmission næste gang

```
while(!SPIS_ReadRxStatus());
uint8 data = SPIS_ReadRxData();
if(data=='1') {
Pin_EN_Write(1);
} else {
Pin_EN_Write(0);
}
UART_1_PutChar(data);
UART_1_PutString("\r\n");
}
SPIS_ClearRxBuffer();
```

*Figure 14 Billede af kode, hvor LED tændes og slukkes afhængig af modtaget char.*

### Master

Masterens kode har store ligheder med Slaven. her bliver der sendt en char2, som bliver sat hver gang, der skrives en karakter ud på terminalen. Når der er klar til data, modtages data fra Slaven og gemmes i variablen data. Dette bliver skrevet ud i terminalen. Der er enten '1' eller '0' afhængig af, om der er trykket på Switchen på Slaven eller ej. Til sidst clears bufferen i Master for at sikre transmissionen næste gang.

```
SPIM_WriteTxData(char2);
while(!SPIM_ReadRxStatus());
uint8 data = SPIM_ReadRxData();
UART_1_PutChar(data);
UART_1_PutString("\r\n");
}
SPIM_ClearRxBuffer();
```

*Figure 15 Billede af kode for Master med afsendelse og modtagelse af char*

## Målinger

Vi har valgt at sætte SPI mode 0, hvor CPOL=0 og CPHA=0. Det kan ses på målingen herunder, hvor clocken starter med at være lav ved denne 8-bit overførsel. Derfor er clock polariteten 0 (CPOL=0). Det kan ses ved den sidste bit på scopet, at den sampler ved rising edge. Den gule kanal 1 er MOSI. Den gule kanal 1 viser altså pulserne, af de bits, der sendes ud af Masteren. Samples bliver lavet i midten af pulsbredden for en puls. Der ses ret tydeligt i slutningen af scopet, at rising edge sker ved midten af denne puls. Derfor passer det, at clock phasen er 0 (CPHA=0).

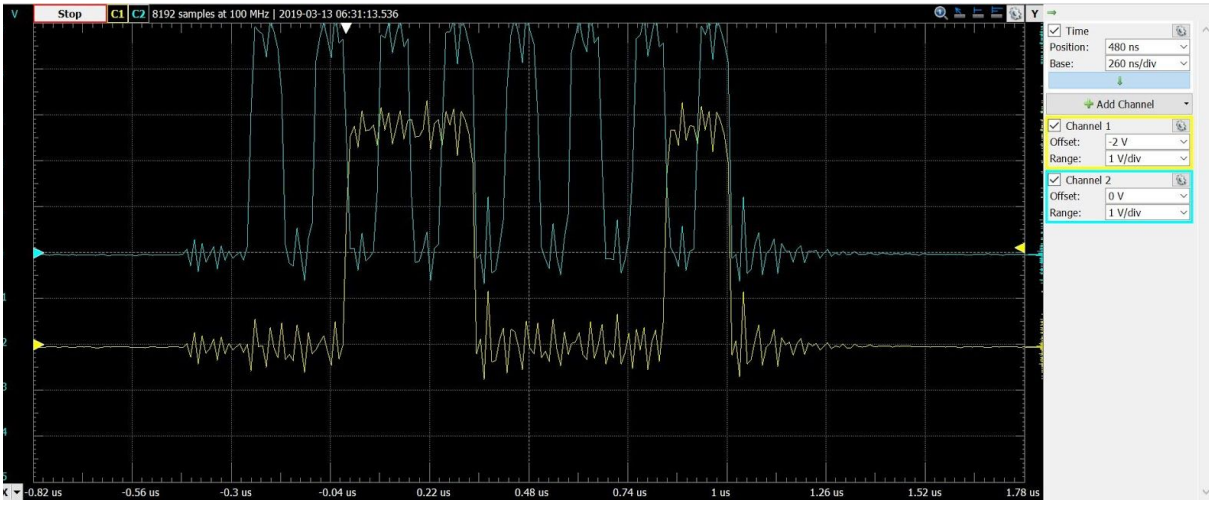



*Figure 16 Billede af oscilloscopet med clock (CLK) på blå scope og MOSI på gul scope. Her sendes et '1' tal ud, så LED tænder på Slaven*

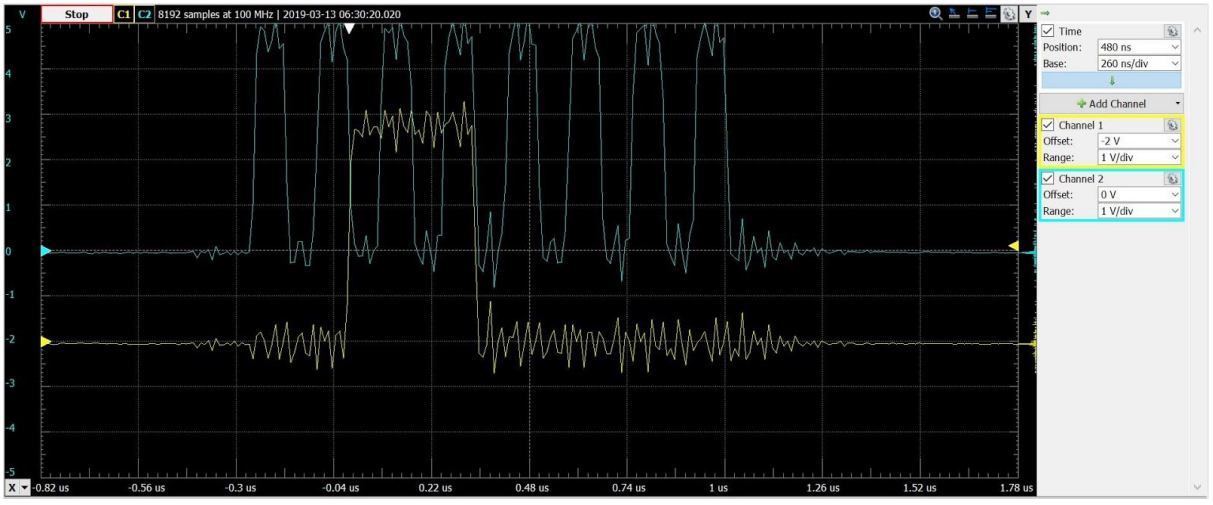



*Figure 17 Billede af oscilloscopet med clock (CLK) på blå scope og MOSI på gul scope. Her sendes en '0' karakter ud, så LED slukker på Slaven*

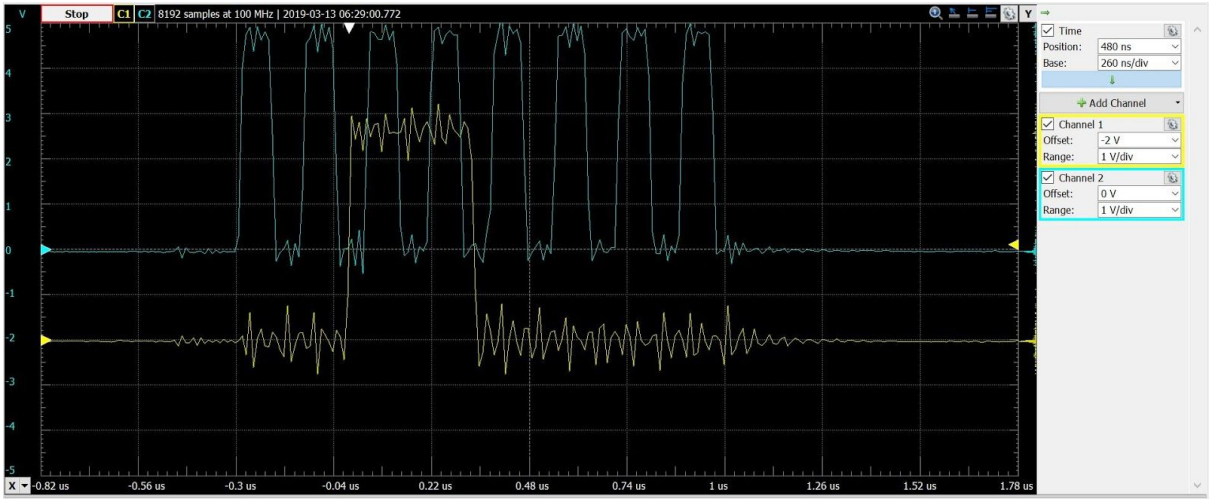



*Figure 18 Billede af oscilloscopet med clock (CLK) på blå scope og MISO på gul scope. Her modtages en '0' karakter, som udtrykker, at der er trykket på switch på Slaven.*

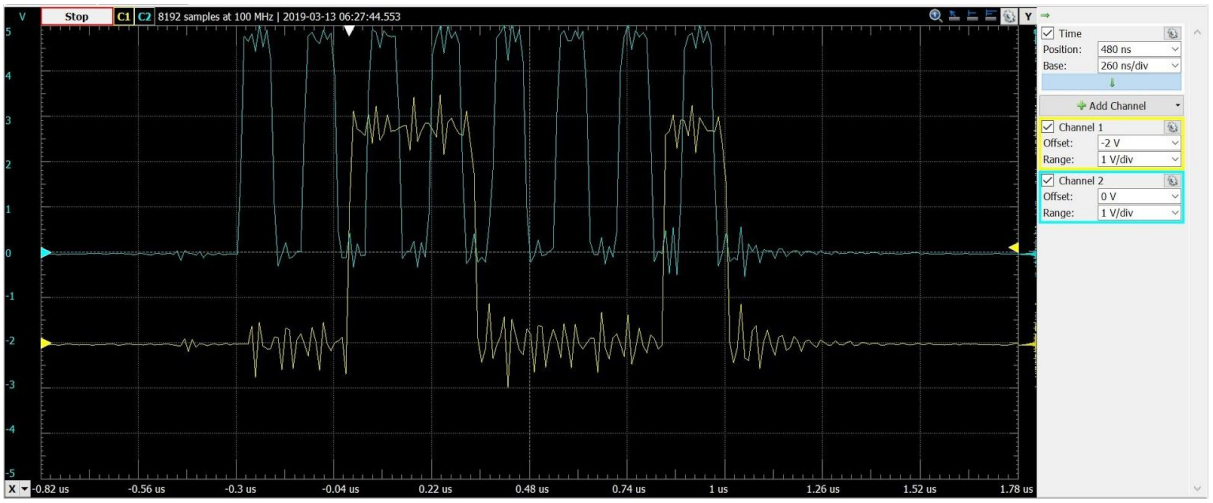



*Figure 19 Billede af oscilloscopet med clock (CLK) på blå scope og MISO på gul scope. Her modtages en '1' karakter, som giver besked til Masteren om, at der ikke er trykket på Switchen på Slaven.*

# Diskussion

Fordelen ved at lade clocken være ikke inverterende er, at den er lav, når transmissionen starter. Det gør, at der ikke skal sendes en spænding ud, når der ikke transmitteres noget. Herudover er clockphasen sat til rising edge. Det gør, at vores sidste bit på MISO og MOSI går lav på omtrent samme tidspunkt som Clocken. Til gengæld vil vores første bit ses før, at clocken går rising edge og derfor skal data være til rådighed inden det. Det ville ses på scopet, hvis vores første bit var højt (1), men eftersom vi udelukkende sender ASCII-værdierne '1' (00110001) og '0' (00110000), så ses det ikke. Det ville iøvrigt ikke være et problem, da data er klar inden den halve clock cycle, og der er relativ stor pause mellem hver byte, der transmitteres.

Som nævnt, så udtrykker vi tænd og sluk for LED med ASCII-værdierne '1' og '0'. Ligeledes udtrykker vi Switch trykket ned og op som henholdsvis ASCII-værdierne '0' og '1'. Det kommer af, at switches som standard er aktiv lavt.

# Resultat

Det lykkedes os at sende '1' og '0' via terminalen, så Lysdioden på Slaven lyste. Det var også muligt at læse '1' og '0' på terminalen, der stod for, om knappen på slaven blev holdt nede eller oppe.
Det kan ses her, hvor lysdioden tændes ved at skrive '1' på terminalen:

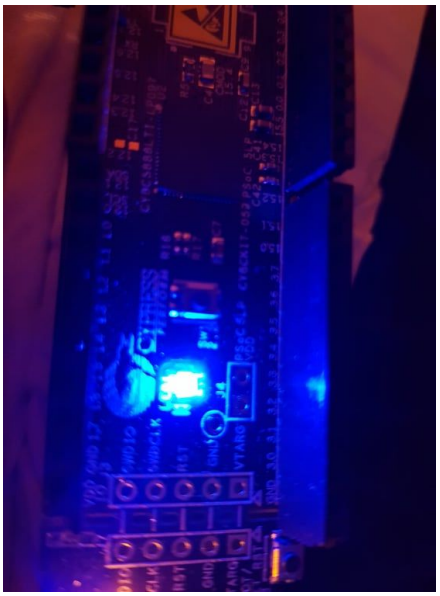


*Figure 20 Billede af LED på Slaven, der lyser, når der sendes '1' på terminalen fra Masterens side.*

Ligeledes kan det ses, at lysdioden slukker ved at skrive 0.

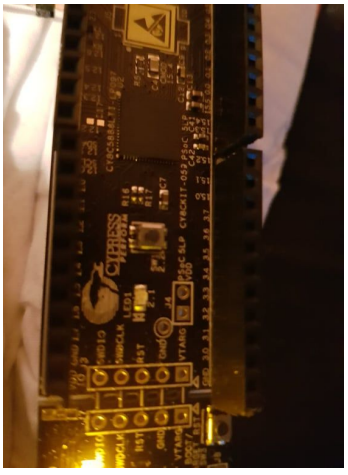


*Figure 21 Billede af LED på Slaven, der ikke lyser, når der sendes '0' på terminalen fra Masterens side.*

Vi har brugt terminalen RealTerm til at aflæse status af switch fra Slaven. Det kan ses i de efterfølgende 2 billeder, at status for tryk og ikke trykket er vist i RealTerm terminalen.

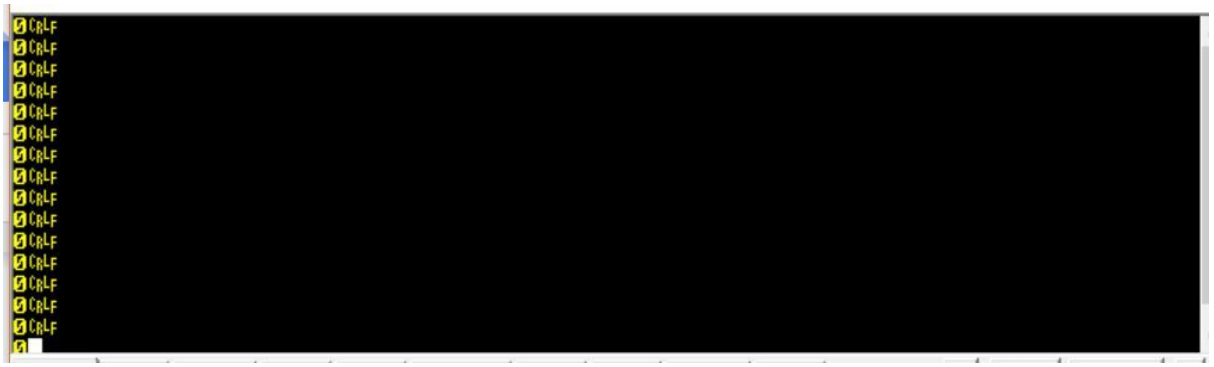



*Figure 22 Billede af terminalen, hvor der trykkes ned på knappen*


*Figure 23 Billede af terminalen, hvor der ikke trykkes ned på knappen*

## Usikkerheder

Vi oplevede, at der var meget støj, da vi skulle måle på MOSI, MISO og SCLK. Vi brugte ekstra ledninger, så vi kunne sætte Analog Discovery imellem. Det magnetiske felt rundt om ledningerne har tilsyneladende påvirket de andre ledninger, således, at digitale 1'er og 0'er (høj og lav) blev lavet om. Det var derfor meget svært at måle, hvor vi havde en fuldendt transmission. Det lykkedes os dog til sidst, som man også kan se på vores scopebilleder. Det kan stadig ses, at der er lidt støj, men det er alligevel ret tydelige puls-signaler, der er repræsenteret på billederne.