I3GFV - LAB Experiment 2

Communication busses

SW	Gülle Northfarm KnotLate	AU612485
SW	Rásmu$ Müller LangFlod$øn	AU633064
SW	Bâk BakelacheeladensBagerBodega	AU577526

Hold nr. 4

Contents

1 Introduktion

2 I2C Experiment: PSoC Master and LM75 Slave

2.1 Introduktion

I denne opgave skal vi undersøge, hvordan man med en I2C master kan snakke med temperatur sensoren LM75. Hertil skal vi bruge viden fra tidligere opgaver og snakke med vores master gennem en UART forbindelse. Dette gør vi for at aflæse LM75'erens målinger.

2.2 Overvejelser

For at vi kan snakke med LM75'eren er der 2 forbindelser vi skal fokusere på.

1. LM75 $\Longleftrightarrow$ PSoC (I2C Master)
2. PSoC $\Longleftrightarrow$ PC (UART)

Disse forbindelse vil vi følgende beskrive og kigge nærmere på, hvilke udfordringer der opstår og hvordan vi planlægger at overkomme dem.

2.2.1 LM75 $\Longleftrightarrow$ PSoC (I2C Master)

Denne forbindelse foregår via I2C og her skal vi fra PSoC'en sende beskeder som LM75'en modtager og håndtere. Den opbygning vi skal give beskederne kan vi se i datasheetet[1]. Først skal vi sætte den adresse, som vi skal kommunikere med LM75'en igennem. Dette gør vi med følgende besked struktur.

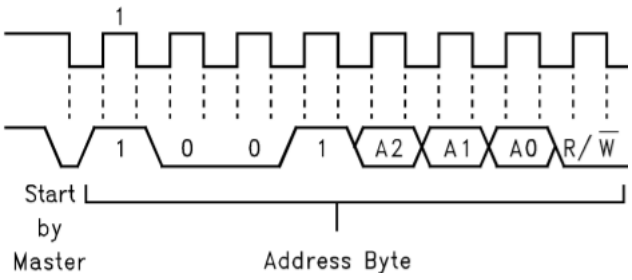

Figure 1: I2C adresse til LM75

Efter at have sat adressen skal vi skal vi modtage temperaturen. Den modtager vi som to 8-bit integers og den følger følgende format:

[1] https://www.ti.com/lit/ds/symlink/lm75b.pdf

Temperature	Digital Output	
	Binary	Hex
125°C	0 1111 1010	0FAh
25°C	0 0011 0010	032h
0.5°C	0 0000 0001	001h
0°C	0 0000 0000	000h
−0.5°C	1 1111 1111	1FFh
−25°C	1 1100 1110	1CEh
−55°C	1 1001 0010	192h

Figure 2: Temperatur formattet fra LM75

Problematikken kommer i at få rykket rundt og behandlet de 2 bytes vi får til kun 1 enkelt byte, hvor vores temperatur er ændret fra 2's kompliment til unsigned. De to bytes vi modtager kommer nogenlunde til at ligne følgende:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MSB								LSB	x	x	x	x	x	x	x

I ovenstående tabel er x brugt til at vise bits som er ligegyldige for os.

Herefter planlægger vi at omskrive det til 1 byte med følgende trin:

- Gem fortegn (+/-) MSB
- Bitshift LSB til plads 15
- Bitshift MSB'en ud og resten af første byte 1 til plads 0
- OR de 2 bytes sammen så vi får LSB ind på plads 7
- Hvis MSB er 1 (-) skal vi invertere plads 0-7 og trække 1 fra for at fjerne 2's compliment
- Returner det halve og cast til en float

Da LM75'en giver os antallet af halve grader halverer vi resultatet og returner det i stedet. Så ved stue temperatur ville man få 40 fra LM75'en i stedet for 20. Dette kan vi herefter sende videre til vores PC gennem UART.

Når denne protokol er opbygget kan vi bruge den til opsætningen af flere LM75'er. Her skal vi bare indstille adressen til en anden og så kan vi forbinde dem alle serielt. Ved at gøre dette kan vi få flere LM75 slaver på samme kommunikations bus. I koden skal vi loope gennem de forskellige adresser og spørge dem en af gangen, når vi så har fået svar skal vi lukke forbindelsen og spørge den næste.

2.2.2 PSoC $\Longleftrightarrow$ PC (UART)

For at snakke mellem PSoC'en og vores computer bruger vi et UART komponent. Dette skal sættes op sådan at PSoC'en sender den læste data fra LM75'en til PC'en. På grund af vi ikke bruger computerens input til noget, har vi ikke noget interrupt på RX benet.

Vores computer sættes op med RealTerm til at modtage fra den USB port som PSoC'en er sat til. Selve formaterringen af teksten foregår alt sammen på PSoC'en.

Et af problemer i denne forbindelse er, hvordan vi håndtere at sende vores temperatur værdi som en "floating point" værdi. Dette problem overkommes dog forholdvist nemt ved at følge en guide[2] givet i undervisningen. Først skal man ind i build settings og sætte float formatting til *TRUE* og herefter skal man bare øge heap sizen til *0x200*. Når dette er gjort kan man caste sin unsigned interger til en float og printe den med printf ved brug af formaterrings type fieldet "%f".

2.3 Implementering

På Figure 3 kan man se topdesignet for vores PSoC. Her har vi et I2C modul til at snakke med LM75'eren og UART modulet til at snakke med computeren.

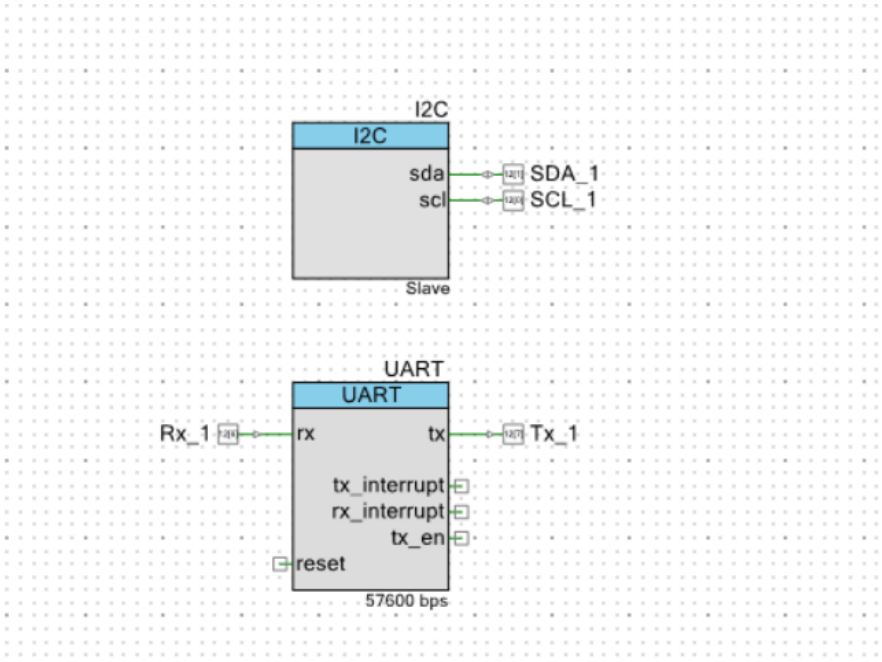

Figure 3: Top design til I2C master

Herefter bestemmer vi, hvor alle pins skal placeres. På Figure 4 kan man se, hvordan pinsne er blevet fordelt.

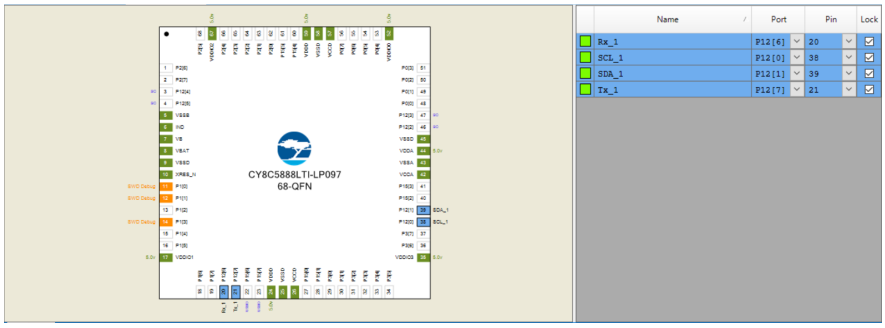

Figure 4: Pin setup på I2C master

[2]GFV Lektion 5 Communication buses - lab experiment Handouts: PSoC-Creator-Printing-Floating-Point.pdf

[KODE AFSNIT]

2.4 Dokumentation

På Figure 5 kan man se vores resultat fra computeren når vi har 1 slave på vores kommunikations bus. Her aflæser vi en temperatur fra 27 grader til 30 grader.

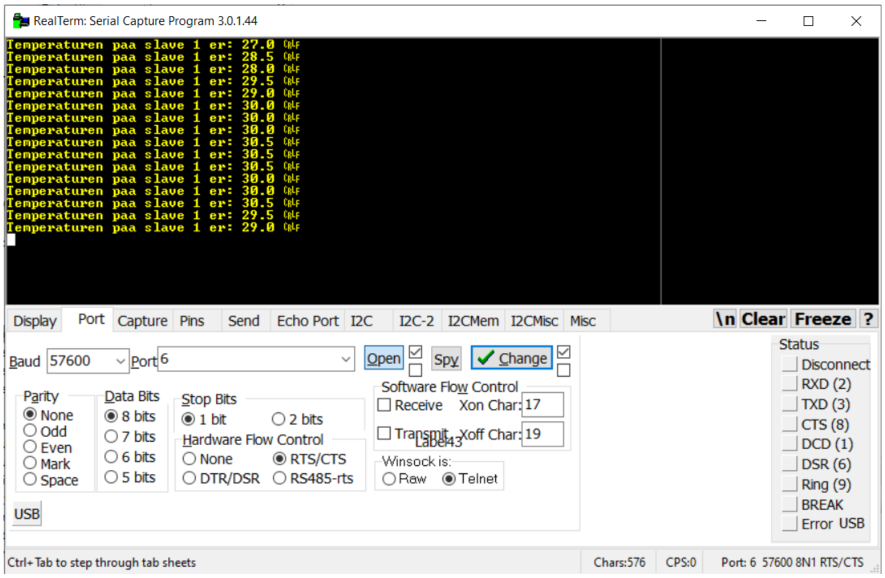

Figure 5: I2C forbindelse med 1 slave

Herefter på Figure 6 kan man se, resultatet efter vi forbandte 2 slaver til kommunikations bussen.

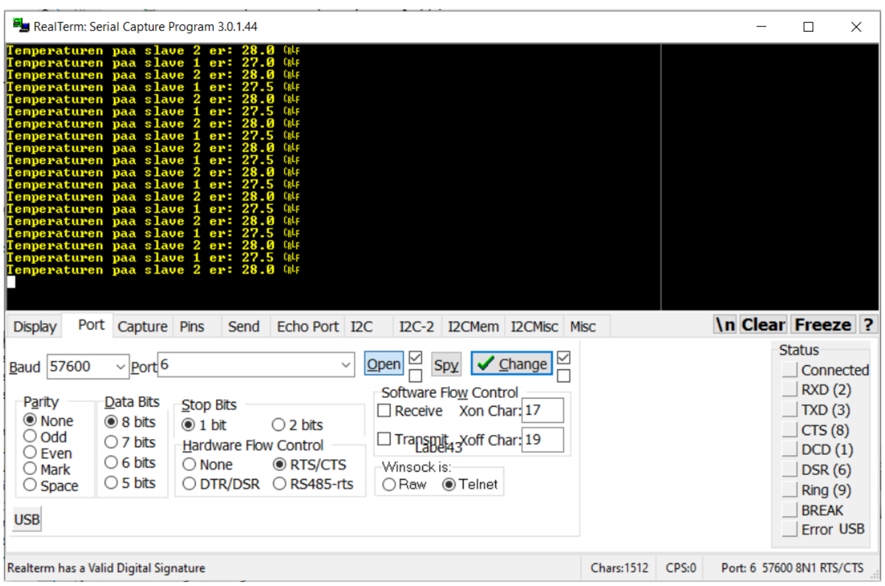

Figure 6: I2C forbindelse med 2 slaver

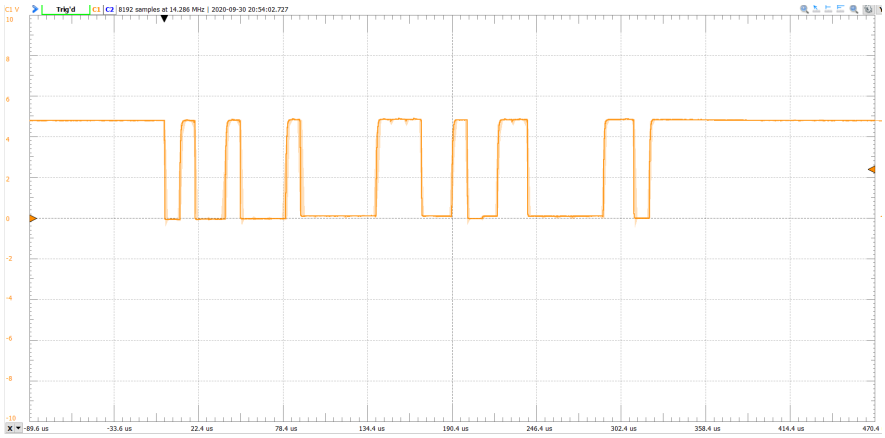

Figure 7: DER SKAL TEKST IGEN

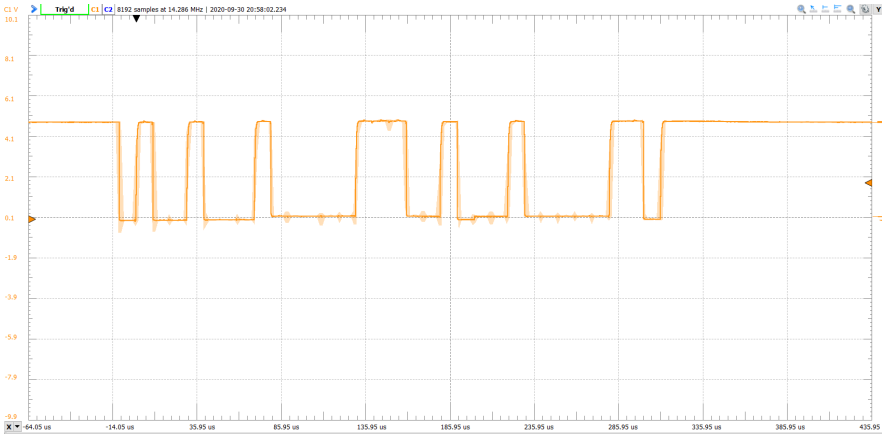

Figure 8: ENDU MERE TEKST

2.5 Diskussion

På Figure 5 kan man se, at temperaturen går mellem 27 grader og 30 grader. Denne ændring kom da vi placerede en finger på sensoren og det passer dermed at temperaturen stiger til 30 grader. Dette passer med forventningen om, at den omkringværende luft er koldere en vores fingrer.

Tilgengæld på Figure 6 kan man ikke se en særlig stor forskel. Denne forskel kommer af, at vi ikke placerede en finger eller et varmt objekt på en af sensorne. Dermed har vi 2 værdier som ligger meget tæt op af hinanden fordi luften omkring dem er ens temperatur.

[Diskuter waveform billeder]

[Diskuter kode ?]

2.6 Konklusion

På baggrund af denne øvelse kan vi konkludere, at man kan kommunikere LM75 med over en I2C forbindelse. Hertil kan man forbinde flere LM75'ere (slaver) på samme kommunikations bus og derved aflæse fra flere slaver over samme forbindelse. Herudover kan vi konkludere, at vi modtager temperaturen over 2 Bytes og med forholdsvis få trin kan man få det omregnet til en enkelt byte, som man herefter meget nemmere kan sende til sin computer og få vist.

3 SPI Experiment: PSoC Master and PSoC Slave

3.1 Introduktion

I denne del af øvelsen vil vi nu arbejde med kommunikations interfacet. Der vil hertil benyttes 2 styk PSoC 5LP, som implementeres som henholdsvist slave og master i systemet. Vha. en UART-forbindelse til PC, vil vi så forsøge at implementere vores master så instrukser sendt fra PC kan sendes gennem SPI-protokollen fra master til slave.

3.2 Overvejelser

Kommunikation mellem master og slave vil som sagt foregå gennem SPI. Vores fokus vil derfor ligge i at sikre, at signalet sendt fra master og signalet modtaget af slave benytter den korrekte protokol.

Vi forventer at benytte AD til at teste hvorvidt signalerne sendes og modtages korrekt, og om det rigtige antal bits benyttes. Desuden er vi blevet informeret om at SPI-protokollen i PSoC-Creator benytter en cirkulær buffer, hvilket vi vil uddybbet senere i opgaven.

3.2.1 SPI

SPI fungere som en datastrøm imellem 2 devices. Det er kun muligt at sende på en SPI, hvis der også modtages fra den samtidig. Modtageren og senderen fungerer som en cirkulær buffer, som læses og skrives til hele tiden. Man bør derfor være opmærksomme på ikke at komme bagud i bufferen, så vores data bliver overskreveet. Da protokollen til denne opgave er forhodlsvist nem, og kan anses for binær, er det valgt simpelthen at ignorere bufferen, og nulstille den, hver gang den bruges.

Fobindelsen mellem Master og Slave er forholdsvist simpel i denne implementering. Der skal oprettes en MOSI, MISO og en SCLK. Det er vigtigt vores SPI Master og SPI Slave kører på samme modes. Vi har defor brugt stadardinstillingen på modes, som er $CPHA = 0$ og $CPOL = 0$. CPOL er vores clock polaritet. I mode 0 starter klokken LOW, når der ikke tælles. CPHA fortæller hvornår vores bits bliver læst. Ved 0 er det ved starten af en clock puls der samples, ved 1 er det i slutningen der samples.

Til at opstille Slave modulet, skal vi desuden bruge en Slave Select. Da der i dette system kun eksisterer en enekelt Slave, vælges der at forbinde Slave Select direkte til GND.

3.3 Implementering

3.3.1 SPI Slave

SPI

Listing 1: SPI Slave

```
1  CY_ISR(isr_spi_rx_handler)
2  {
3     uint8_t data = SPIS_ReadByte();
4
5     // Write to terminal for DEBUGGING
6     char buf[20];
7     sprintf(buf, "Data received: %x\r\n", data);
8     UART_PutString(buf);
9
10    // Clear buffer, makes it easier
11    SPIS_ClearRxBuffer();
12
13    switch(data)
14    {
15        case 0xcc:
16        {
17            LED_Write(1);
18        break;
19        }
20        case 0x55:
21        {
22            LED_Write(0);
23            break;
24        }
25        default:
26        {}
27    }
28 }
```

3.4 Dokumentation

På Figure 9 nedenfor ses vores endelige opstilling. Forbindelserne mellem slave og master, beskrevet i implementeringsafsnittet, er etableret som specificeret , og vi ser som forventet at vi kan tænde og slukke for LED'en på slave PSoC'en ved at sende i/o til vores master PSoC.

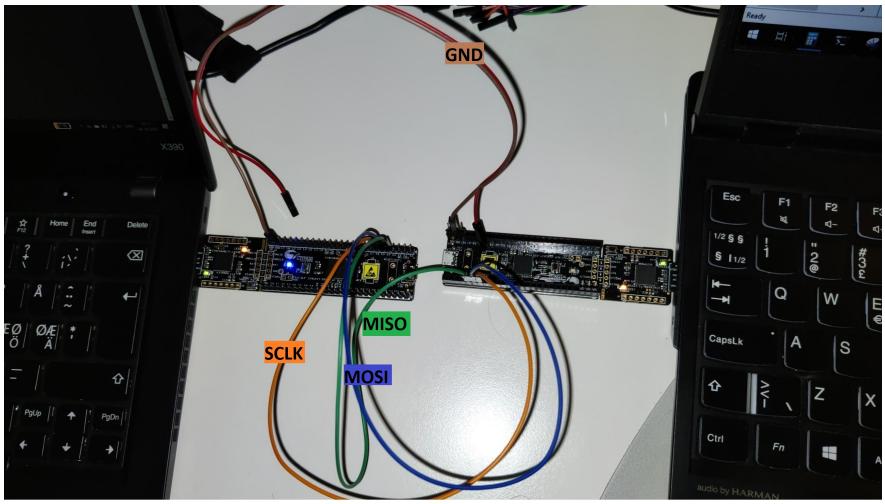

Figure 9: DER SKAL TEKST IGEN

For at få en bedre forståelse af hvad der sker signalmæssigt, og for at sikre systemet ikke bare virker ved et uheld, benytter vi Analog Discovery Oscilloskop til at måle i/o signalet sendt mellem PSoC parret. På Figure 10 ses vores måling af signalet for at tænde LED'en, mens Figure 11 viser signalet sendt for at slukke LED'en.

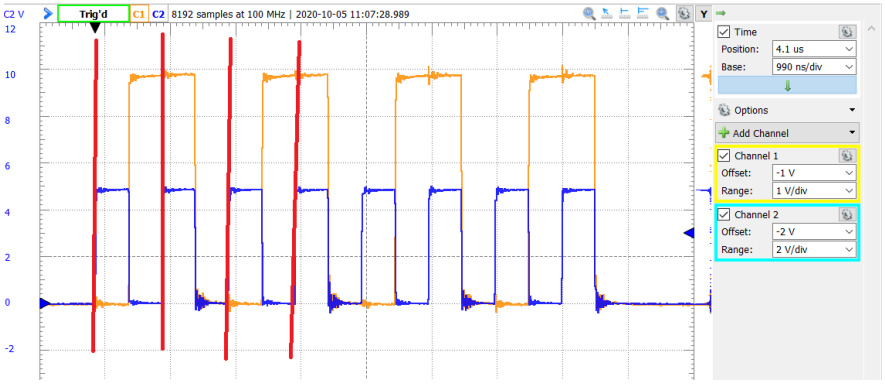

Figure 10: DER SKAL TEKST IGEN

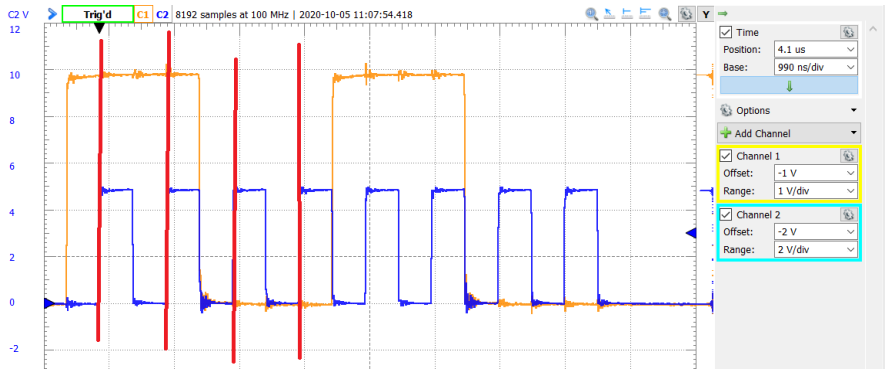

Figure 11: DER SKAL TEKST IGEN

3.5 Diskussion

Som vi først ser på Figure 9 virker vores implementering, og vi kan tænde og slukke for vores slave LED ved at sende et signal til vores master PSoC. Idet vi ville sikre at SPI protokollen blev kørt korrekt, valgte vi specifikke hex-tal til at repræsentere I/O signalerne.

Dette gav dog nogle problemer under implementeringen, da vi ved målinger med Analog Discoverys Spy funktion fandt at signalet modtaget af slave PSoC'en blev forskudt en halv byte ift. hvad der blev sendt af master. Vi fandt at fejlen opstod hvis et tidligere signal ikke var blevet modtaget korrekt. Hvis bufferen så ikke blev tømt efterfølgende, lagde protokollen blot dele af det nye signal ind i bufferen oveni fejlsignalet.

Dette probl

Vi fandt at fejlen skyldtes SPI protokollens mangel af en acknowledge, og derfor

Har ikke noget achknowledge, derfor vis missede enkelt del af modtagning, alle følgende signaler forskudt/-forkert Ikke mulighed for at vælge

Kunne evt. bruge slaveselect, kan sættes til at tømme buffer.

Halvt så stort som clock-signal, os der har ændret på akser så nemmere kan se.

3.6 Konklusion

[TEXT HERE]

4 (Optional)I2C Experiment: PSoC Master and PSoC Slave