

0.1 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.

0.2 Overvejelser

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

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

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

0.2.1 LM75 \iff 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 datasheetet¹. Først skal vi sætte den adresse, som vi skal kommunikere med LM75'en igennem. Dette gør vi med følgende besked struktur.

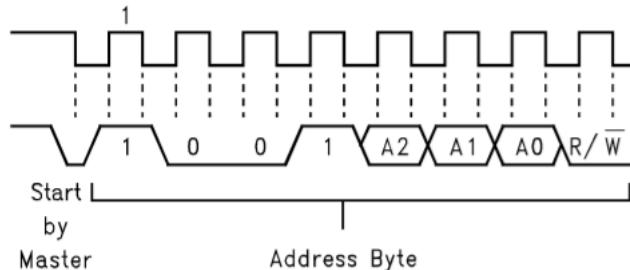


Figure 1: I2C adresse til LM75

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

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

¹<https://www.ti.com/lit/ds/symlink/lm75b.pdf>

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 invitere 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.

0.2.2 PSoC \iff 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 forholdsvis nemt ved at følge en guide² 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".

0.3 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.

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

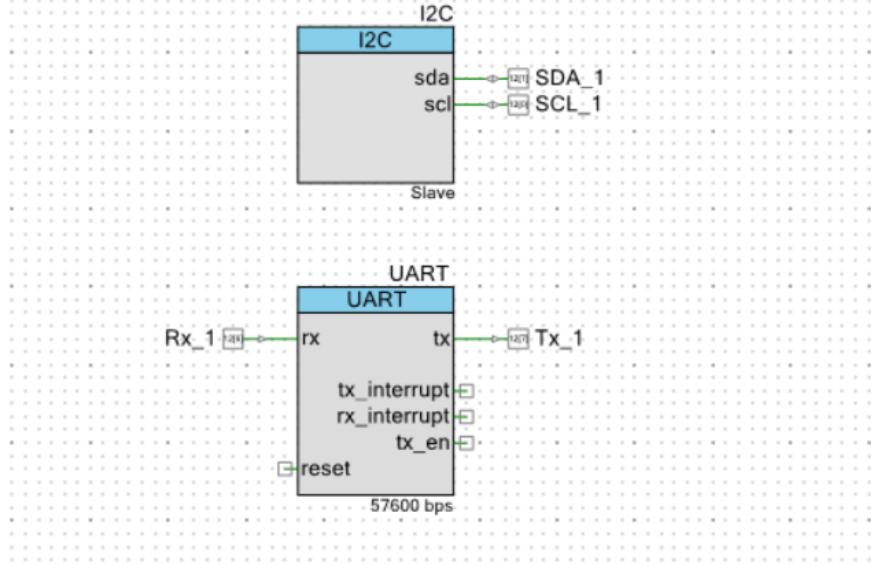


Figure 3: Top design til I2C master

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

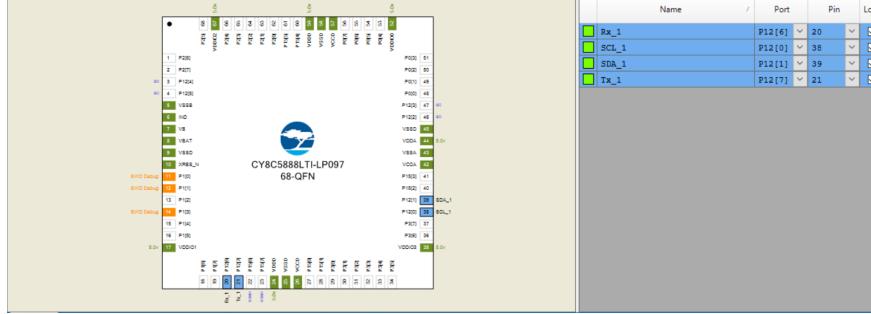


Figure 4: Pin setup på I2C master

I koden er der 2 primære dele:

1. Indlæsning af data fra I2C
2. Print af float over UART

Disse dele vil følgende blive gennemgået og der vil blive beskrevet, hvordan koden modtager og håndterer vores input.

0.3.1 Indlæsning af data fra I2C

Efter at vi i *main()* loopet har startet og slået interrupten for I2C til, skal vi ha lavet en funktion som modtager input fra en I2C adresse og sender det ud som en float. Vores prototype hedder:

```
1 uint8 i2cRead(uint8 addr, float *result);
```

Først har vi returntypen uint8, den bruges til at vise om vi har fået en fejl eller ej, hvis der ikke er en fejl returnere vi '1'. Herefter har vi adressen til vores I2C device, den specificere hvilket device vi gerne vil have data fra. Til sidst har vi resultatet. Dette modtager vi som en adresse for at kunne parse det ud af funktionen samtidigt med at vi har en fejlkode fra returværdien.

Listing 1: i2cRead()

```
1 uint8 i2cRead(uint8 addr, float *result){
2     uint8 buffer[BUFFER_LEN];
3     I2C_Init();
4
5     I2C_MasterReadBuf(addr, buffer, BUFFER_LEN, I2C_MODE_COMPLETE_XFER);
6
7 // Wait for transfer to complete
8     while(I2C_MasterStatus() == I2C_MSTAT_XFER_INP);
9     if(I2C_MasterStatus() != I2C_MSTAT_RD_CMPLT){
10         // Re-init I2C after fault
11         I2C_Init();
12         // Tranfer err
13         return 0;
14     }
15
16 // Interpret LM75 2s compliment into proper float
17 // Follow the steps explained in report, section 2.2.1
18     buffer[1] >>= 7;
19     _Bool comp = buffer[0] & 0b10000000;
20     buffer[0] <<= 1;
21     uint8 total = buffer[0] | buffer[1];
22     if(comp)
23     {
24         --total;
25         total ^= 0xff;
26     }
27
28     *result = (float)total * 0.5;
29
30     //no err
31     return 1;
32 }
```

Variablen BUFFER_LEN er defineret som 2.

På listing 1 kan man se, hvordan vi i første halvdel opretter en buffer, som vi bruger til at opbevare vores data fra I2C forbindelsen. Herefter tjekker vi for fejl og genstarter I2C, hvis der er fejl.

Efter dette følger vi bare de trin der blevet specificere i afsnit 0.2.1 for at gøre det til en enkelt byte. Denne

byte smider vi ind på adressen fra result og herefter returnerer vi '1' da der ikke har været nogle fejl.

0.3.2 Print af float over UART

På listing 2 kan man se vores main funktion. Kort beskrevet, så modtager den floats fra *i2cRead()*, kopiere det over i vores printBuf sammen med vores forklarende tekst streng og til sidst printer den det over UART til vores computer, hvor vi kører RealTerm til at modtage dataen fra UART fobindelsen.

Listing 2: main()

```
1 int main(void)
2 {
3     CyGlobalIntEnable; /* Enable global interrupts. */
4
5     // Initiate UART & I2C
6     UART_Start();
7     I2C_Start();
8     I2C_EnableInt();
9
10    for (;;)
11    {
12        if (!i2cRead(0x48, &temp1)){
13            //temp error value
14            temp1 = -0.1;
15        }
16        sprintf(printBuf, "Temperaturen_paa_slave_1_er : %.1f\r\n", temp1);
17        UART_PutString(printBuf);
18        CyDelay(1);
19
20        if (!i2cRead(0x49, &temp2)){
21            //temp error value
22            temp2 = -0.1;
23        }
24        sprintf(printBuf, "Temperaturen_paa_slave_2_er : %.1f\r\n", temp2);
25        UART_PutString(printBuf);
26        CyDelay(500);
27    }
28 }
```

Variablen PRINT_LEN er defineret til 50.

En vigtig del af vores main loop er det delay vi indsætter efter vi har læst fra begge LM75'ere. Dette gør vi for ikke at læse for hyppigt på slaverne. Hvis vi gjorde det risikerede vi at overophede slaverne og dermed få en forkert temperatur, da LM75'eren har en anden temperatur end den omkringværende.

Fra datasheetet³ lyder det nemlig at

"The LM75 should not be accessed continuously with a wait time of less than 300 ms."

Hvis vi spørger oftere end 300ms risikerer vi at afbryde en temperatur læsning hos LM75'en. Så starter den

³<https://www.ti.com/lit/ds/symlink/lm75b.pdf?> - Side 5, Afsnit 6.5, note (5)

bare forfra og dette kan risikere, at den skaber en unødig varme. Derfor har vi valgt at indsætte et delay på 500ms. Se linje 29 på listing 2.

Efter vi har klaret alt dette kan vi nu lave vores opstilling med PSoC, LM75 boards og computer.

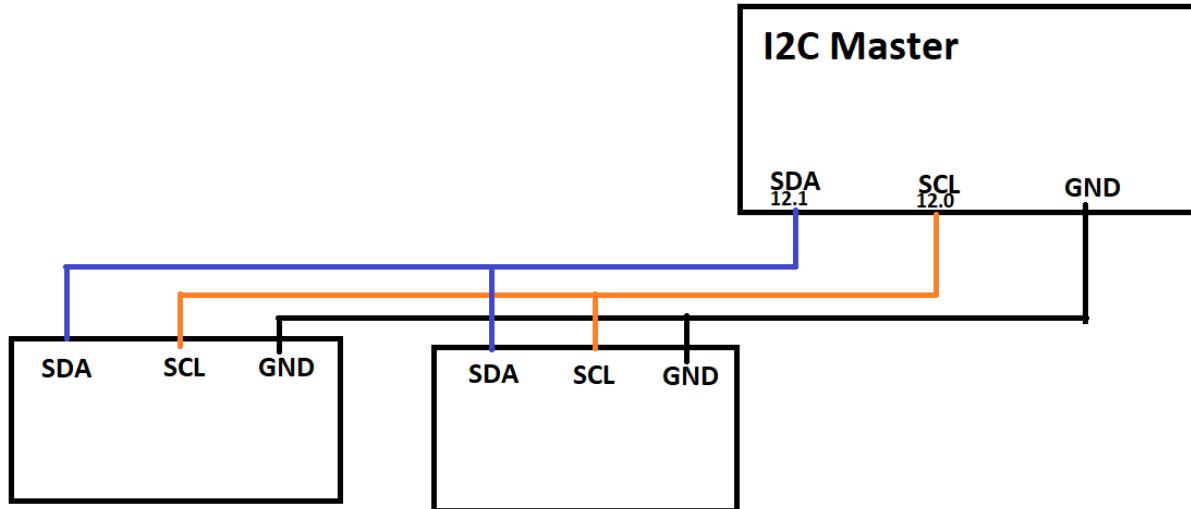


Figure 5: Opstillingsdiagram

På Figure 5 kan man se, at vi udfra top design har lavet et diagram over opstillingen. Dette vil vi bruge til at lave følgende opstillinger med en eller to slave(r).

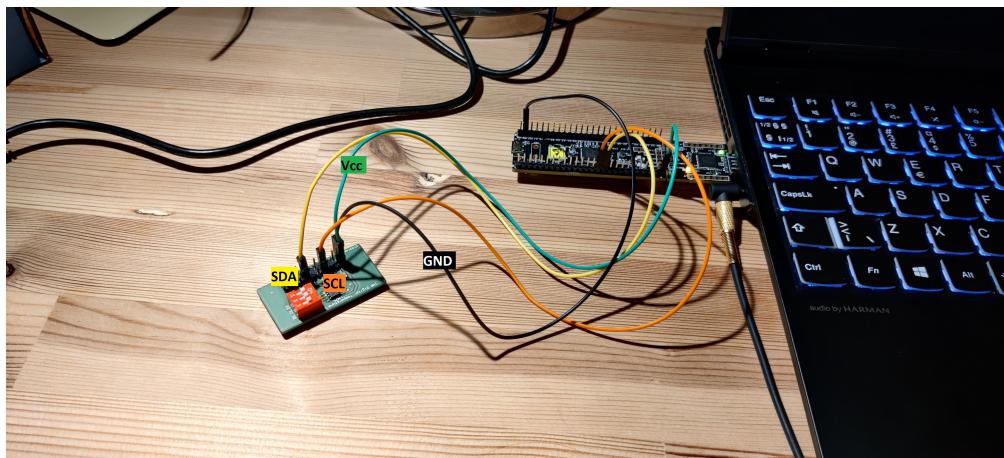


Figure 6: Opstilling af I2C forbindelser

På Figure 6 kan man se opstillingen, hvorpå der er indsatt bokse, der beskriver ledningernes formål.

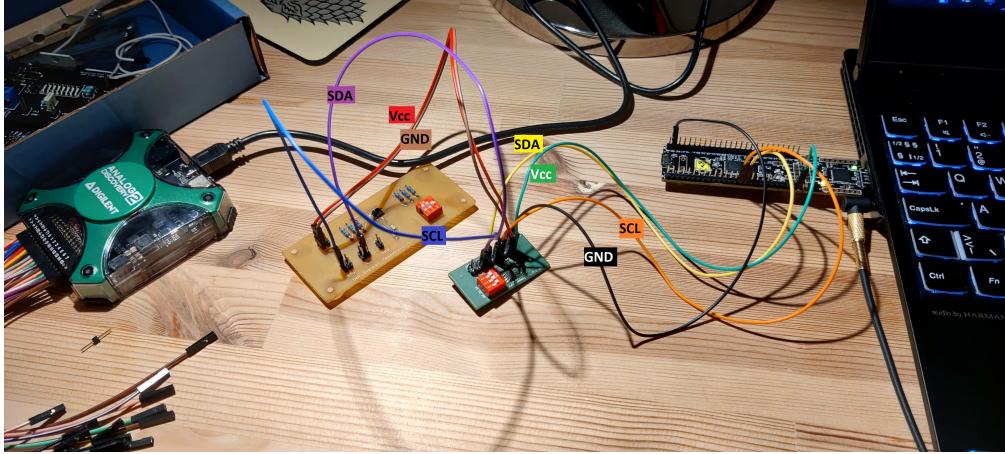


Figure 7: I2C opstilling med 2 LM75 slaver

På Figure 7 kan man se opstillingen, hvor vi har 2 LM75'ere forbundet til kommunikations bussen. Her er der igen indsatt beskrivende bokse.

0.4 Dokumentation

På Figure 8 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.

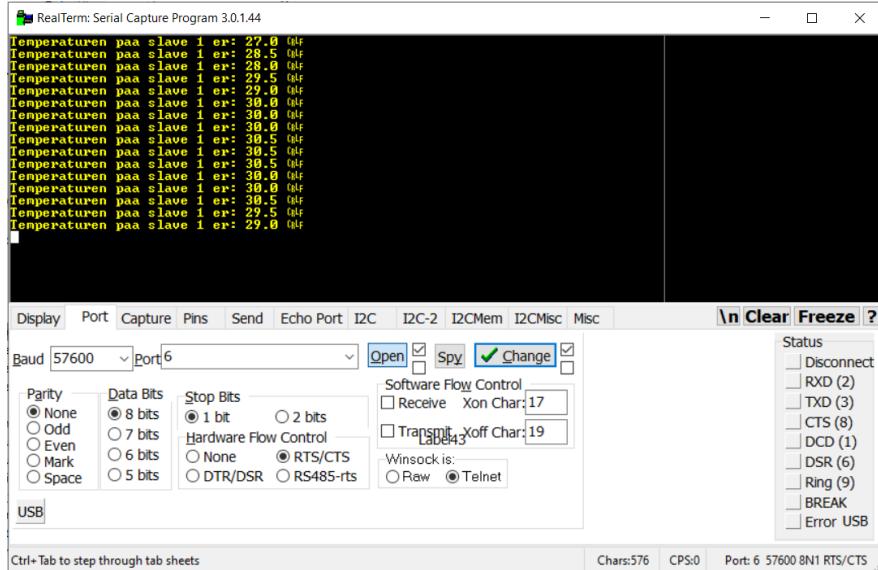


Figure 8: I2C forbindelse med 1 slave

Herefter på Figure 9 kan man se, resultatetet efter vi forbandte 2 slaver til kommunikations bussen.

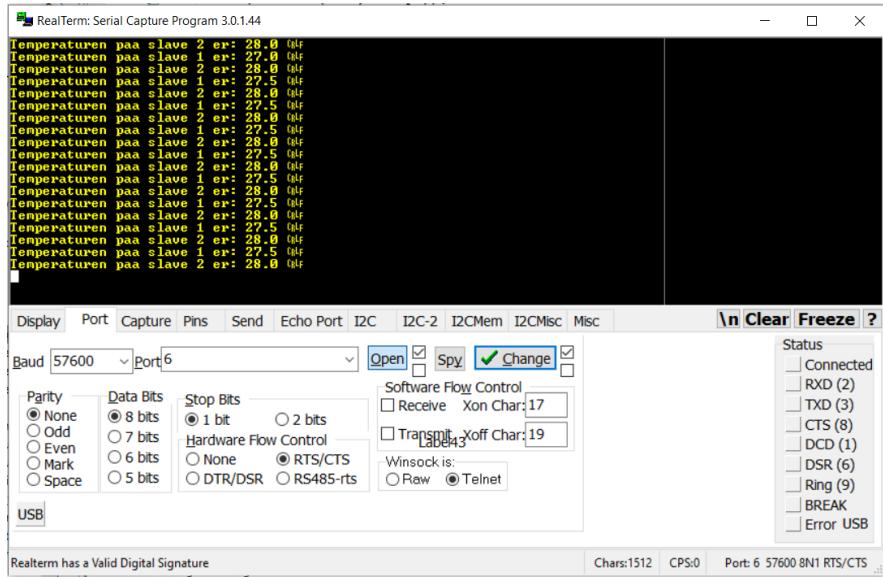


Figure 9: I2C forbindelse med 2 slaver

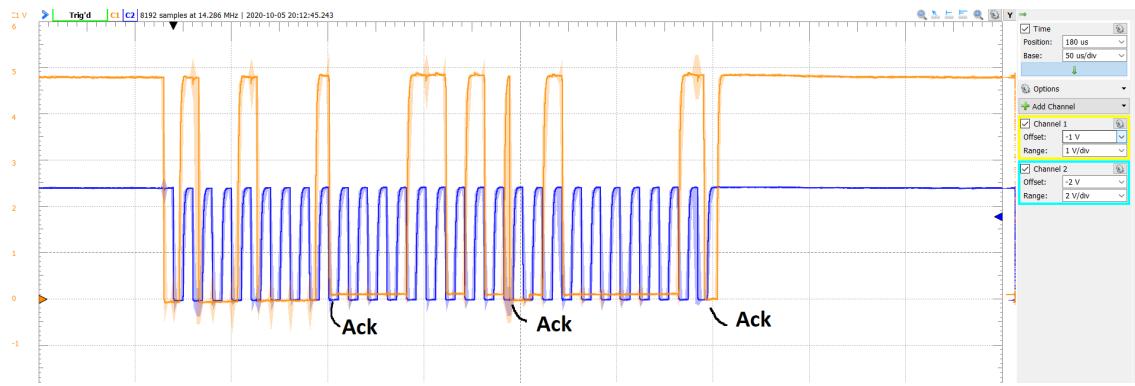


Figure 10: Oscilloscope billede med clock

På Figure 10 kan man se vores I2C data på channel 1 (den orange) og samtidigt på channel 2 (den blå) kan man se clock signalet, der synkronisere det hele.

0.5 Diskussion

På Figure 8 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 finger.

Tilgengæld på Figure 9 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.

På Figure 10 kan man se, hvordan vores protocol følges af signalet. Inden den første acknowledgement har vi PSoC'en der sender adressen ud på data linjen og herefter kan man se, på grund af, at ground bliver forskudt

fra vores PSoC's ground, at det er LM75'en der svarer tilbage. Nu ved vi at LM75'en svarer med 2 bytes af data og vi kan se, at ind imellem disse 2 bytes er der en acknowledge fra masteren. Dette kan man igen se på den lille forskel der sker på vores data forbindelse og at vores LOW bliver skudt en lille smule op fra 0V. Derudover kan man se, at vores data kun skifter når vores clock er LOW, dette er specificeret i protocollen for I2C⁴, hvor den siger:

”The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW (see Figure 4). One clock pulse is generated for each data bit transferred.”

Figure 4, som quoten referer til, kan ses på Figure 11.

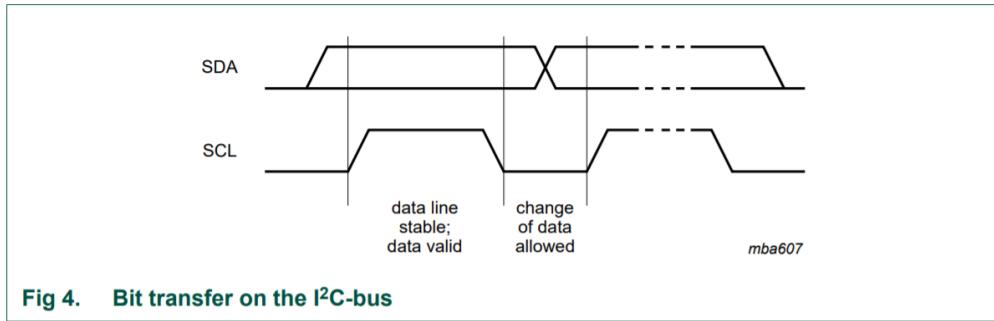


Figure 11: Specifikation for ændring af Data linjen

Et af de problemer vi løb rigtigt meget ind i under vores programmering af PSoC'en var, at når vi havde begge LM75'ere forbundet opstod fik vi ikke ventet på, at den første var færdig med at skrive på bussen og derved når vi skulle læse fra den næste LM75 læste vi bare en masse blandet fra begge slaver. Derfor har vi det lille tjek på linje 8 & 9 i listing 1.

0.6 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. Dog er det vigtigt, at man ikke spørger for ofte, da det kan øge risikoen for at få en forkert temperatur.

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.

⁴<https://www.nxp.com/docs/en/user-guide/UM10204.pdf> - Side 9, Afsnit 3.1.3 Data validity