Lab 9: Sensorer og I2C

Målet for denne laben er å få en introduksjon til sensorer med et digital grensnitt, noe som blir med og mer vanlig å benytte.

I kurset så langt har vi sett mye på analoge komponenter, men vi skal i denne labøvelsen se på hvordan man kommuniserer og henter ut data fra en sensor med et digitalt grensesnitt. Mange moderne sensorer er rene ICer eller ferdig utviklet kretser med et digitalt grensesnitt. Brukeren ikke har tilgang til den analoge delen av sensoren må da forholde seg til databladet for som forteller om egenskapen og hvordan komponenten virker.

Temperatur sensor
fra MicroChip

Ultralyd sensor
fra MaxBotix

RGB sensor
fra TAOS

For å kommunisere med slike sensorer bynyttes ofte en buss, men det kan også være at kommunikasjon med sensoren er definert i et eget digitalt grensesnitt fra produsenten. Den mest vanlige bussene er I2C, men også andre busser benyttes til kommunikasjon med sensorer .feks SPI, IEEE1451.2, and IEEE1451.3. Mer informasjon om ulike busser benytte mot sensoer finner du her (https://www.google.no/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0ahUKEwj-tl_1jbbTAhXMApoKHe0EB-sQFggjMAA&url=http%3A%2F%2Fwww.mdpi.com%2F1424-8220%2F2%2F7%2F244%2Fpdf&usg=AFQjCNEr-ZuilFhR8RSKsg-6Ov2bDvAxSw&sig2=6vTqzSCHG-jSciN5jW02Gg).

I denne labben skal vi se på en sensor for å detektere farger som kommunisererover I2C. Microbiten som egentlig er en liten microcontroller har bygget inn støtte for denne bussen. Den bruker faktisk også denne bussen til å kommunisere med de sensorene du finner på Microbiten, akselerometret og kompasset.

Hvilke fordeler gir så IC (pakket) sensor?

- Sensoren gir oftest en bedre verdi enn hva du oppnår selv ved å bygge det du trenger av elektronikk rundt sensoren.
- Sensoren gir ofte en lineær verdi ut fordi ulineæriteter allerede er korrigert for på ICen.
- Du spare tid både i forhold til design og kalibrering.
- Redusert strømforbruk.
- Et digitalt grensesnitt er med robust mot støy, særlig hvis det er lengre avstander og man må benytte ledninger for å koble til.

Ulemper er at:

- Du må støtte det digitale Interfacet i designet ditt.
- Du mister kontrollen over hvordan sensoren oppfører seg og må stole på leverandørens spesifikasjoner.
- Det krever kunnskap om hvordan du kommuniserer med sensorer noe som ofte innebærer C kompetanse eller VHDL kompetanse. I dette tilfellet bruker vi micopython.

I2C protokoll

I2C er en Two-wire interface og består alltid av to linjer: serial data (SDA) og serial clock (SCL). Protokollen tillater multi-master og multi-slave koblet sammen, men for å gjøre det enklere skal vi bare se på en master med flere slaver. Se tabell 1 for noen definisjoner.

Table 1. Definition of I²C-bus terminology

Term	Description
Transmitter	the device which sends data to the bus
Receiver	the device which receives data from the bus
Master	the device which initiates a transfer, generates clock signals and terminates a transfer
Slave	the device addressed by a master

Tabell 1

Et oppsett hvor det er kun en master må inneholde et minimum av egenskaper. Av tabellen kan det sees at disse er START og STOP condition, Acknowledge og 7-bit slave-adresse. En sensor kan ha flere egenskaper enn disse fire, men det skal vi ikke gå nærmere inn på i denne laben.

Table 2. Applicability of I²C-bus protocol features
M = mandatory; O = optional; n/a = not applicable.

Feature	Configuration	
	Single master	Multi-master
START condition	M	M
STOP condition	M	M
Acknowledge	M	M
Synchronization	n/a	M
Arbitration	n/a	M
Clock stretching	O[2]	O[2]
7-bit slave address	M	M
10-bit slave address	O	O
General Call address	O	O
Software Reset	O	O
START byte	n/a	O[3]
Device ID	n/a	n/a

Tabell 2

Disse fire funksjonene er helt nødvendige for at kommunikasjonen skal virke. Selve kommunikasjonen mellom enheter forgår ved at SDA veksler mellom høy og lav. Både SDA og SCL er normalt høy når det ikke foregår en overføring. All overføring over SDA må være delt inn i bytes (8-bit) etterfulgt av en acknowledge (1-bit). Det er derimot ingen begrensninger på hvor mange bytes som kan sendes etter hverandre.

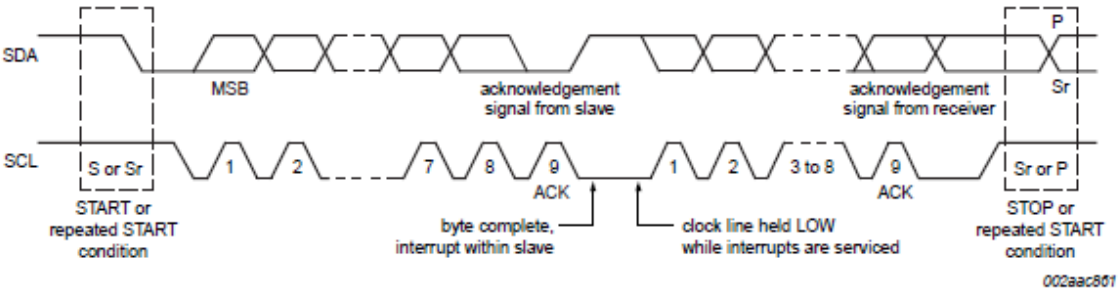


Fig 6. Data transfer on the I²C-bus

Figur x

START/STOP

START and STOP conditions

All transactions begin with a START (S) and are terminated by a STOP (P) (see Figure 5). A HIGH to LOW transition on the SDA line while SCL is HIGH defines a START condition. A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition.

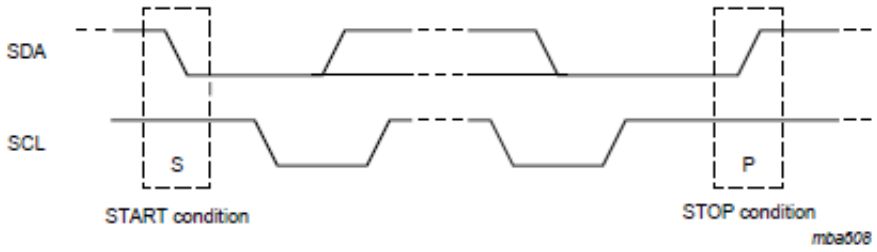


Fig 5. START and STOP conditions

Acknowledge

Acknowledge (ACK) and Not Acknowledge (NACK)

The acknowledge takes place after every byte. The acknowledge bit allows the receiver to signal the transmitter that the byte was successfully received and another byte may be sent. The master generates all clock pulses, including the acknowledge ninth clock pulse.

7-bit slave address

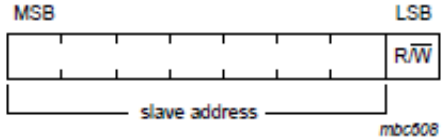
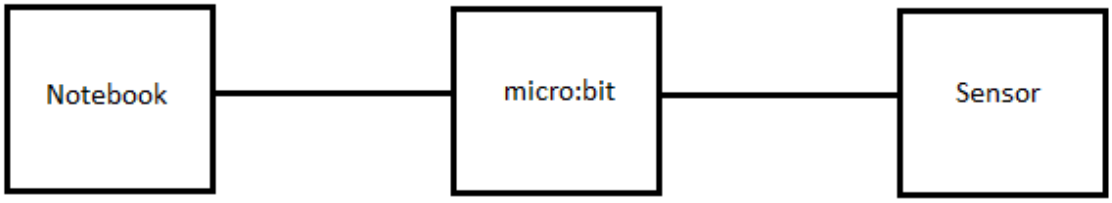


Fig 10. The first byte after the START procedure

Spesifikasjon og brukerveiledning for I2C: NXP-I2C (http://www.nxp.com/documents/user_manual/UM10204.pdf)

Oppsett av sensoren

I denne laben skal du bruke Notebook til å programmere og lese data ut fra micro:bit. micro:biten skal kommunisere med sensoren.

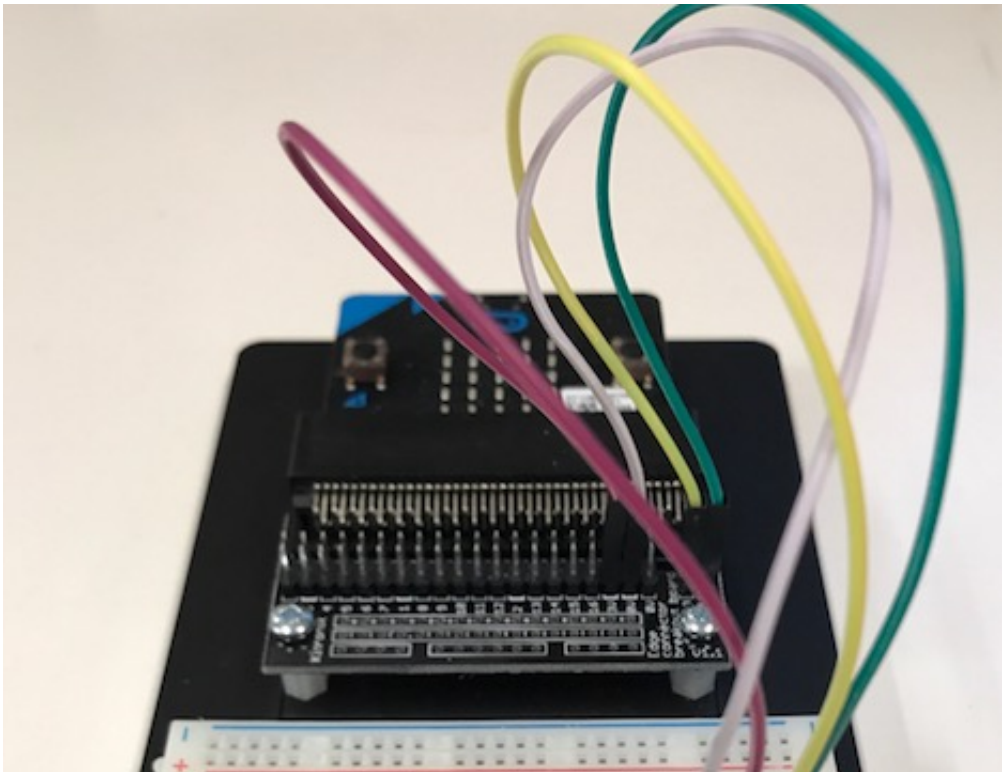


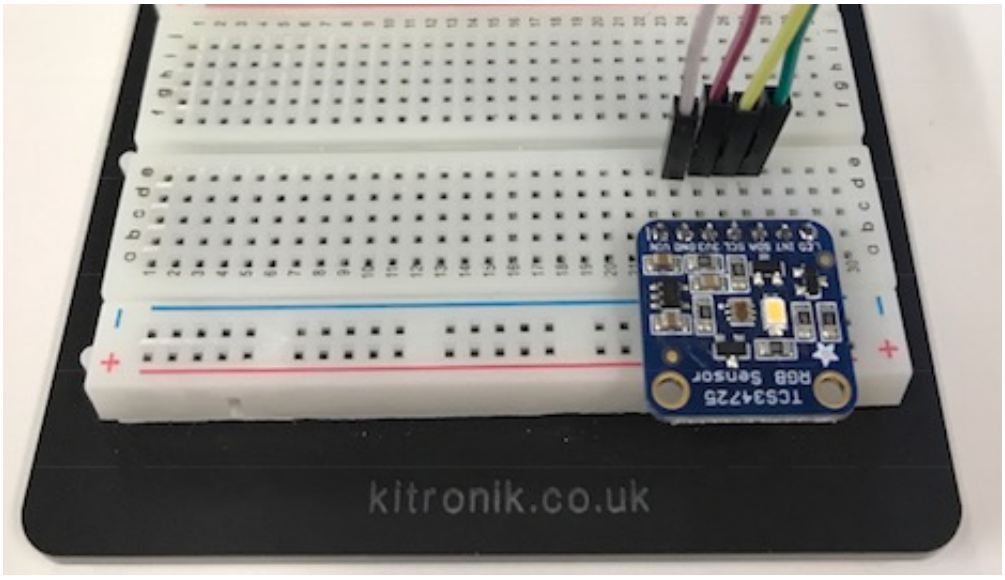
Figur x: Kommunikajson mellom Notebook, micro:bit og sensor.

For å komme igang må vi koble opp micro:biten og sensoren.

Utstyrsliste:

- micro:bit
- Kitronik brødbrett
- 4 ledninger (han-hun av typen som følger med inventor's kit)
- TCS34725 RGB-sensor





Oppsett lab 9

Kobling fra sensor til pin på micro:bit som følger:

- SDA til pin 20
- SCL til pin 19
- 3V3 til 3V
- GND til 0V

Programmering av micro:bit

Laste opp tom fil til micro:bit

Det neste vi må se på er hvordan vi laster opp (programmerer) micro:biten. Kommandoen uflash laster opp skript til micro:bit. Den er lagd av Nicholas Tollervey og ligger på GitHub (https://github.com/ntoll/uflash). Vi begynner med å laste en tom fil til micro:bit som overskriver det som eventuelt måtte ligge der fra før.

In [ ]:
```
!uflash ubit/clean.py
```

Ved bruk av magic'en `%load_ext <extension>` kan man laste inn customized IPython extensions (http://ipython.readthedocs.io/en/stable/config/extensions/index.html). I denne laben skal det brukes en IPython extension laget for micro:bit. Denne extensionen gjør at en kan bruke micro:bit i en vanlig notebook med følgende magics:

Magic	Input	Output
%ub	`line`	-
%%ub	`cell`	-
%ubp	`print(line)`	print

%ubr	`print(line)`	return
%ubport	`line`	Text

Deler av ubmagic extension består av kode til ubit_kernel (https://github.com/takluyver/ubit_kernel) som er lagd av Notebook-utvikler Thomas Kluyver. (ubit_kernel ble brukt i Lab 1 og 2)

Laste IPython extension for micro:bit

In [ ]:
```
%load_ext ubmagic
```

Eksempler på bruk av micro:bit magics

%ub og %%ub

Disse to brukes til å skrive til micro:bit. De har lik funksjonalitet som ubit_kernel. %ub sender linjen til micro:bit, mens %%ub sender cellen. For å få output må det brukes `print()`.

%ub

In [ ]:
```
%ub x = 1
```

In [ ]:
```
%ub print(dir())
```

In [ ]:
```
%ub print(x)
```

%%ub

In [ ]:
```
%%ub
y = 2
z = 3
print(dir())
print(y, z)
```

In [ ]:
```
%%ub
xyz = [x, y, z]
print(dir())
for i in xyz:
    print(i)
```

%ubp

%ubp x er faktisk bare %ub print(x), men den gjør at en slipper å skrive `print(x)` hver gang en skal lese av noe fra micro:bit.

In [ ]:
```
%ubp x
%ubp y, z
%ubp dir()
```

%ubr

%ubr brukes til hente variabler fra micro:bit til notebooken. Det går også ann å sende data fra notebooken til micro:bit ved bruk av `$`-operator før variabelen (f.eks `$x`).

In [ ]:
```
x_nb = %ubr x
print(x_nb)
```

In [ ]:
```
ubdir = %ubr dir()
print(ubdir)
for item in ubdir:
    print(item)
```

Laste tom fil til micro:bit

In [ ]:
```
!uflash ubit/clean.py
```

Bruk `%ubport reconnect` etter flashing til micro:bit med uflash.

In [ ]:
```
%ubport reconnect
```

Sensor

Den eneste måten å finne ut hvordan kommunisere med en sensor med I2C er å lese tilhørende datablad. For å spare tid får du utvalgte utdrag fra databladet til TCS34725 i den hensikt at du skal få et inntrykk av:

- Hva sensoren gjør
- Sensorens adresse
- Sensorens oppbygning og protokoll
- Oversikt over registere

TCS34725 generelt

TCS34725 gir digital data for nivåer av rød, grønn, blå (RGB) og klart lys (C). Sensoren består av 12 fotodioder (3 x 4), 3 for hver farge og 3 for klart lys (omtaler klart lys som farge videre). Hver av diodene har et optisk filter (et fysisk bellegg) som blokkerer ut de delene av lysspekteret de ikke skal måle, i tillegg til lyskomponenter fra IR-spekteret. Ulikt lys gir ulike spenninger over hver diode, som igjen er koblet til fire ADCer som integrerer disse spenningene til en 16-bit digital verdi for hver farge. Dataen lagres i 8 dataregistere, 2 for hver farge. Når disse registerene får kommandoer fra m:bit, kan data om enten R,G,B, eller C avleses.

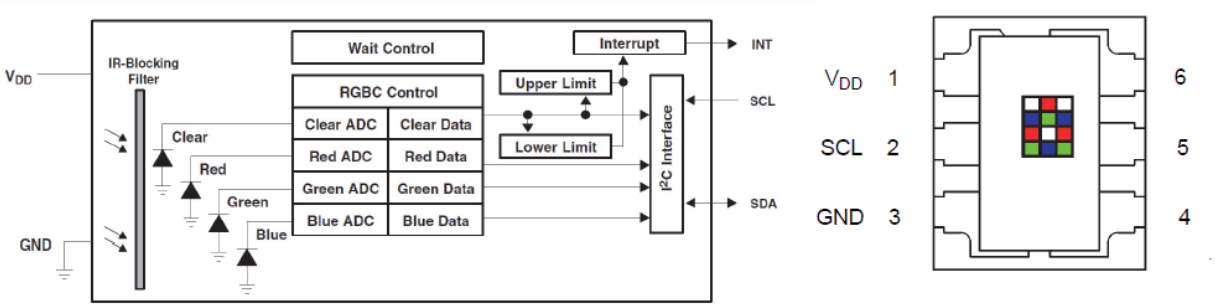


Fig x: Oppbygning og pins

TERMINAL NAME	NO.	TYPE	DESCRIPTION
GND	3		Power supply ground. All voltages are referenced to GND.
INT	5	O	Interrupt — open drain (active low).
NC	4	O	No connect — do not connect.
SCL	2	I	I^2C serial clock input terminal — clock signal for I^2C serial data.
SDA	6	I/O	I^2C serial data I/O terminal — serial data I/O for I^2C .
V_{DD}	1		Supply voltage.

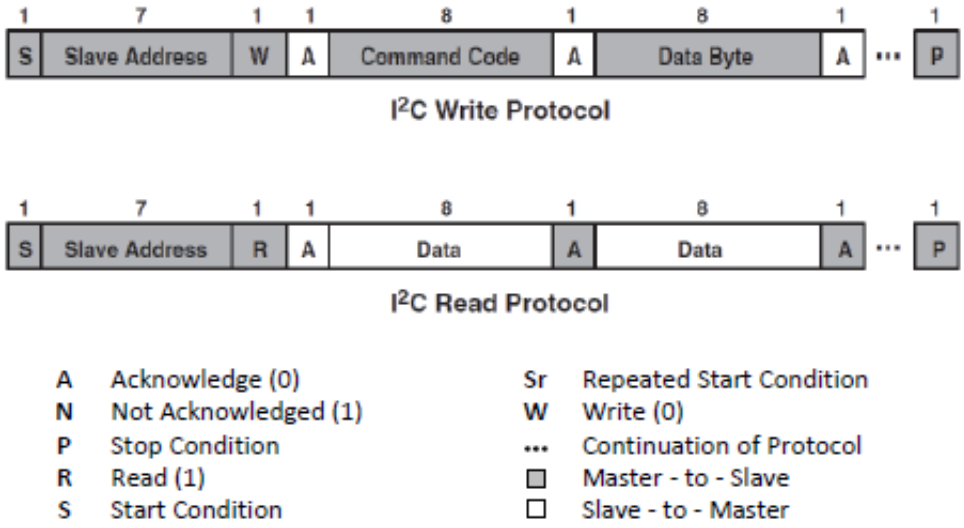
Fig x: Oversiktstabell

Avlesing fra TCS34725

I2C gir mulighet for tre typer dataoverføring mellom master og slave; `read`, `write`, og `combined protocol`. I lab 9 konsenterer vi om de to første. I2C-protokollen bestemmer formatet dataoverføringene gjøres i. Før du kan programmere m:bit til å hente ut data fra registerene må du derfor først lære om protokollen som er implementert spesifikt på sensoren av produsenten.

I2C-protokoll for TCS34725

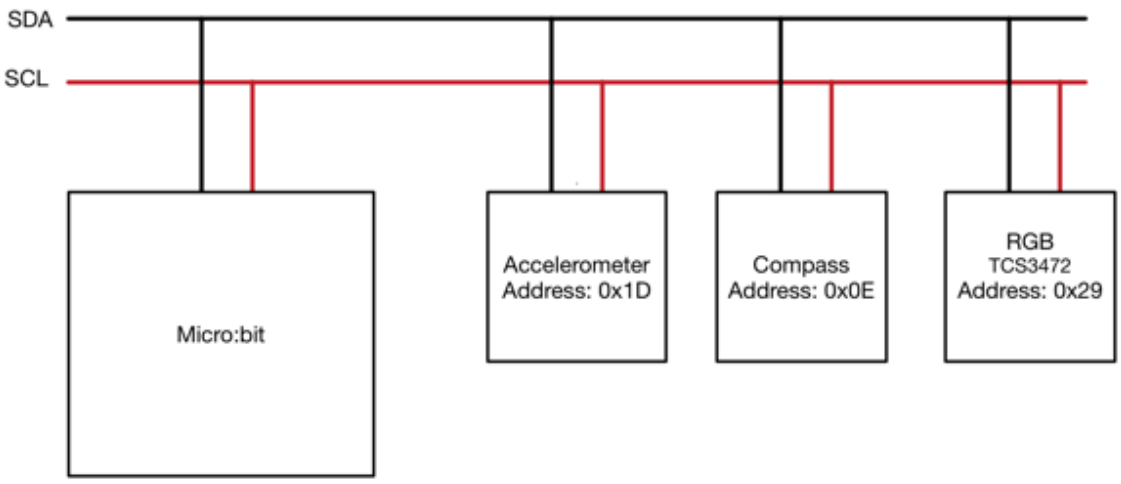
Under ligger figur som viser strukturen på dataoverføring mellom TCS34725 og m:bit, altså I2C-protokollen. Slik det fremkommer fra figuren er informasjonen strukturert i bytes (8-bit). I oppsettet vi skal bruke i lab 9 har vi flere slaver tilgjengelig ettersom I2C-bussen på m:bit også er koblet til de innebygde sensorene (akselerometer og kompass) i tillegg til TCS34725. TCS3472 har adressen: 0x29. Dette er skrevet på hexadesimal form, (0x indikerer hexadesimal). Heksadesimal form brukes fordi det er lettere å lese for mennesker (etterhvert). Vi skal først se på et eksempel på hvilke sensorer som er koblet til micro:bit og deretter et eksempel på hvordan slaveadresse virker.



Figur x: I2C-protokoll for RGB-sensoren; `read/write`

Eksempel: Sensorer koblet til micro:bit

Etter å koblet opp RGB-sensoren skal det nå være tre sensorer med I2C-interface koblet til micro:bit slik som på illustrasjonen.



Under er det en funksjon som scanner addresser fra `0x08` til `0x77`. Det er lett å finne verdien til hex i Notebooken:

In [ ]:
```
print(0x08)
print(0x77)
```

Først lastes funksjonen opp til micro:bit. Deretter kan skriptet kalles ved bruk av `%ub`

`scan()`. (PS. scanningen tar noen sekunder).

In [ ]:
```
%%ub
from microbit import i2c

start = 0x08
end = 0x77

def scan():
    addr_found = []
    print("Scanning I2C bus...")
    for i in range(start, end + 1):
        try:
            i2c.read(i, 1)
        except OSError:
            pass
        else:
            addr_found.append(i)
            print("Found:  [%s]" % hex(i))
    print("Scanning complete")
    return addr_found
```

In [ ]:
```
%ub addr_found = scan()
addr_found = %ubr addr_found
addr_found
```

Eksempel: Multi-slave (slave adresse)

I det første eksempelet ser vi at det finnes tre slaver med hver sin adresse. Vi skal nå se nærmere på to av disse.

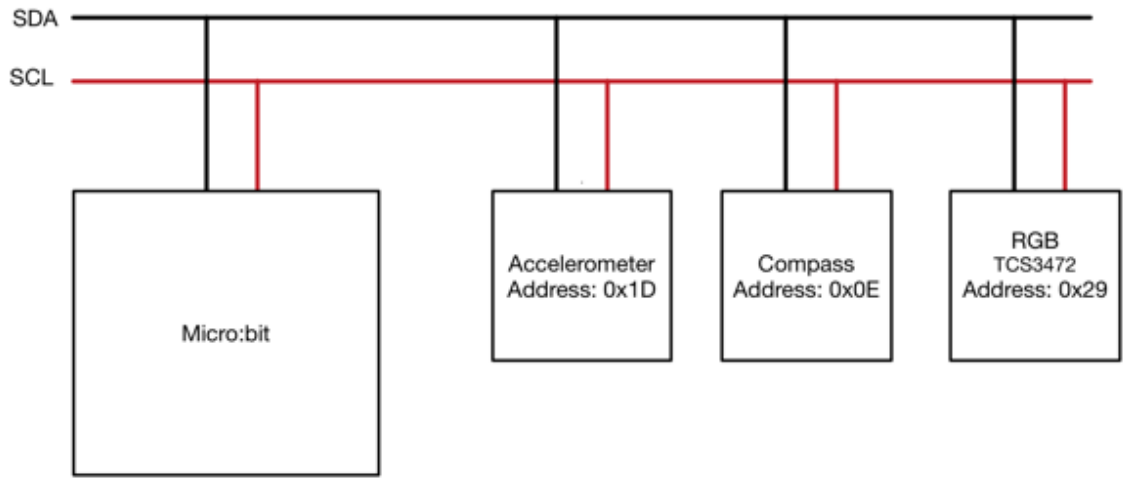
- Byte nummer 1 er alltid slave adresse på 7-bit og 1 bit for write eller read.
- Byte nummer 2 avhenger av hvordan produsenten har designet i2c-interfacet i sensoren.

For å kunne bruke I2C må modulen i2c importeres. Den har bare to funksjoner; read og write.

read/write

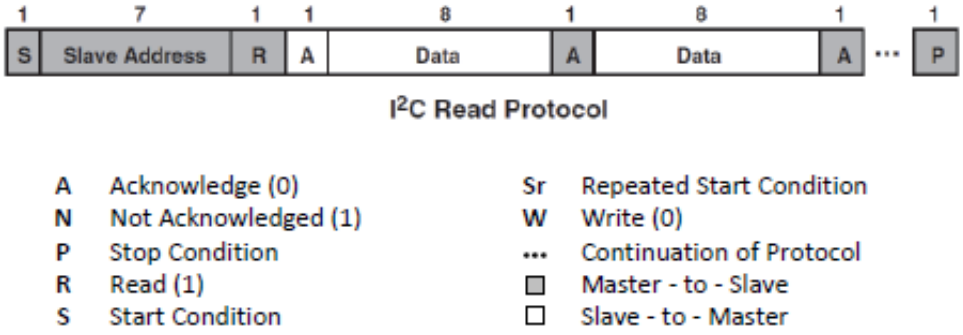
- i2c.write(addr, buf), hvor buf er bytes som skal sendes til sensoren.
- i2c.read(addr, n), hvor n er antall bytes som skal avleses.

Når du legger inn en adresse i disse to funksjonene, er micro:bit programmert slik at den gjør første byte om til `slave address (7 bit) + R/W (1 bit)` slik protokollen krever.



I cellene under er det kode for å lese av device ID for RGB sensoren og akselerometeret. Poenget med disse to cellene er å forstå hvordan slave-adresser brukes, samt at disse to sensorene har forskjellig protokoll, så ikke fortvil om koden er vanskelig å forstå.

RGB-sensor: TCS34725



Figur x: Read protokoll for RGB-sensor

In [ ]:
```
%%ub
addr = 0x29
buf = bytearray(1)
buf[0] = 0x12 + 0x80
i2c.write(addr, buf)
r = i2c.read(addr, 1)
print(hex(r[0]),'er ID, i desimal:', r[0])
```

Akselerometer: MMA8653FC

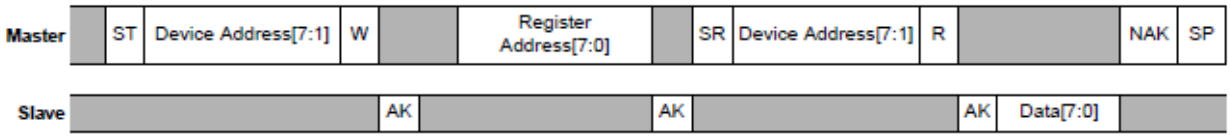


Figure 10. Single-Byte Read timing (I^2C)

Figur x: Read protokoll for akselerometer

In [ ]:
```
%%ub
addr = 0x1d
buf = bytearray(1)
buf[0] = 0x0d
i2c.write(addr, buf, repeat=True)
r = i2c.read(addr, 1)
print(hex(r[0]),'er ID, i desimal:', r[0])
```

Bruk av RGB-sensoren

Write

- "S" - Setter SDA til lav for å indikere at at en overføring starter. Ved bruk av i2c modulen til m:bit skjer dette automatisk.
- "Slave address" - Spesifiserer hvilken slave som skal brukes. Må spesifiseres.
- R/W -må være 0 for write og 1 for read. Settes automatisk av master (m:bit) når `i2c.write` og `i2c.read` brukes.
- "A" - Slave sender "0" om den har mottatt og forstått byte sendt over SDA av master (m:bit). Skjer automatisk.
- "Command code" - Må inneholde en adresse på 5-bit som bestemmer hvilket register det skal skrives til. Må spesifiseres. Oversikt for vår sensor er i figur x, mens detaljer for hvert register står i databladet til sensoren.
- "Data bytes" - Informasjonen som eventuelt skal skrives til ønsket register.
- "P" - Setter SDA til høy for å indikere at overføringen er ferdig. Automatisk ved bruk av i2c modul.

Read

- Første byte - som i "write".
- Andre byte - I Command-registeret er det lagret en adresse til et register. Ved bruk av `i2c.read` vil sensoren returnere data fra registeret med denne adressen. Det er altså viktig å forsikre seg om at Command-registeret inneholder riktig adresse. Dette gjøres ved en write-protokoll før "read" anvendes.
- Tredje byte og utover - Eventuelle flere bytes fra registere. Sensoren vil inkrementere til neste register automatisk av avlesning av flere bytes.

Register

Og husk at et register kun er en enhet som lagrer informasjon. Alle registerne er på 8-bit og har en egen adresse. Data som er større enn 8-bit må fordeles over flere registere.

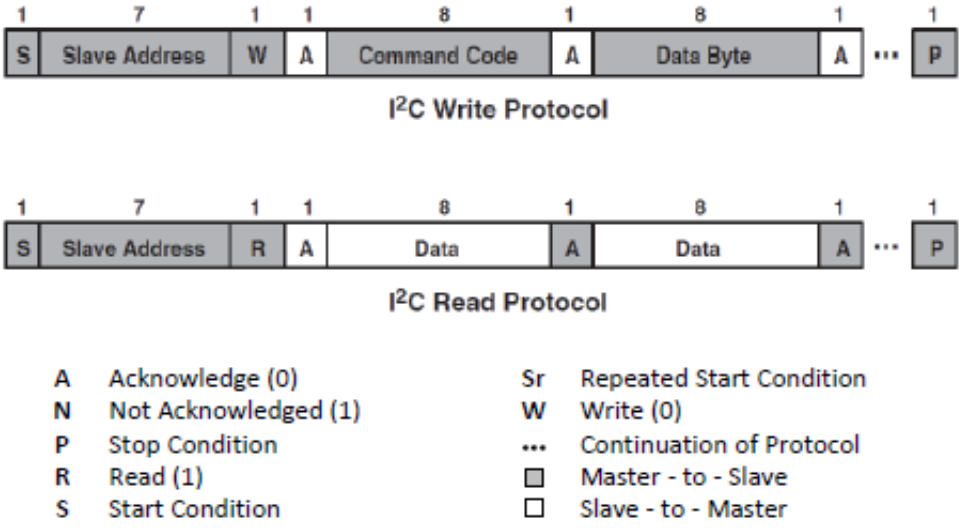


Fig x: I2C-protokoll for "read"/"write"

Address	Register Name	R/W	Register Function	Reset Value
--	COMMAND	W	Specifies register address	0x00
0x00	ENABLE	R/W	Enables states and interrupts	0x00
0x01	ATIME	R/W	RGBC time	0xFF
0x03	WTIME	R/W	Wait time	0xFF
0x04	AILTL	R/W	Clear interrupt low threshold low byte	0x00
0x05	AILTH	R/W	Clear interrupt low threshold high byte	0x00
0x06	AIHTL	R/W	Clear interrupt high threshold low byte	0x00
0x07	AIHTH	R/W	Clear interrupt high threshold high byte	0x00
0x0C	PERS	R/W	Interrupt persistence filter	0x00
0x0D	CONFIG	R/W	Configuration	0x00
0x0F	CONTROL	R/W	Control	0x00
0x12	ID	R	Device ID	ID
0x13	STATUS	R	Device status	0x00
0x14	CDATAL	R	Clear data low byte	0x00
0x15	CDATAH	R	Clear data high byte	0x00
0x16	RDATAL	R	Red data low byte	0x00
0x17	RDATAH	R	Red data high byte	0x00
0x18	GDATAL	R	Green data low byte	0x00
0x19	GDATAH	R	Green data high byte	0x00
0x1A	BDATAL	R	Blue data low byte	0x00
0x1B	BDATAH	R	Blue data high byte	0x00

Fig x: Register Set

Det siste som trengs for å kunne skrive og lese er "Command code"-byten. Most Significant Bit (MSB -bit med størst verdi) må være 1 for at adressen skal lagres i Command-registeret - husk at dette registeret spesifiserer hva som leses av når du bruker read-protokollen. Verdien til MSB er 128 (`0x80/0b10000000`).

Command Register

The command register specifies the address of the target register for future write and read operations.

Figure 25:
Command Register

7	6	5	4	3	2	1	0
CMD	TYPE			ADDR/SF			

Fields	Bits	Description
CMD	7	Select Command Register. Must write as 1 when addressing COMMAND register.

Oppgaver

Sensoren må gjøres klar til bruk for å kunne avlese RGBC. Det betyr at `Power ON` (PON) og `RGBC enable` (AEN) må først settes til 1 i Enable-registeret som har addressen 0x00. Figur x viser en forenkling av hvordan sensoren opererer og hva PON og AEN gjør.

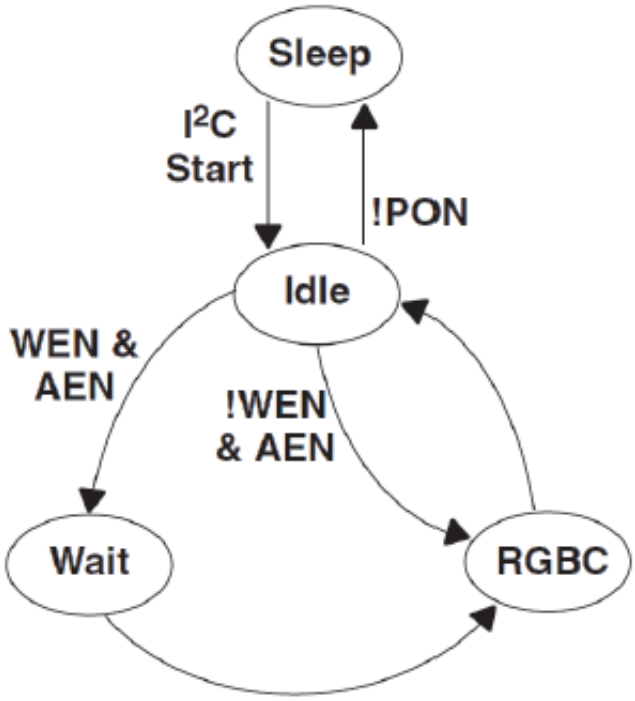


Fig x: Simplified State Diagram

Enable Register (0x00)

The Enable register is used primarily to power the TCS3472 device on and off, and enable functions and interrupts as shown in Table 5.

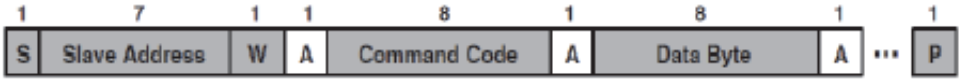
Table 5. Enable Register

	7	6	5	4	3	2	1	0	
ENABLE	Reserved			AIEN	WEN	Reserved	AEN	PON	Address 0x00

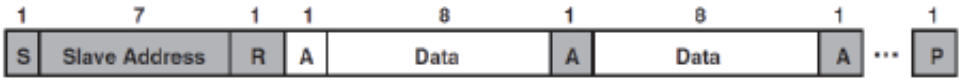
FIELD	BITS	DESCRIPTION
Reserved	7:5	Reserved. Write as 0.
AIEN	4	RGBC interrupt enable. When asserted, permits RGBC interrupts to be generated.
WEN	3	Wait enable. This bit activates the wait feature. Writing a 1 activates the wait timer. Writing a 0 disables the wait timer.
Reserved	2	Reserved. Write as 0.
AEN	1	RGBC enable. This bit actives the two-channel ADC. Writing a 1 activates the RGBC. Writing a 0 disables the RGBC.
PON [1, 2]	0	Power ON. This bit activates the internal oscillator to permit the timers and ADC channels to operate. Writing a 1 activates the oscillator. Writing a 0 disables the oscillator.

NOTES: 1. See Power Management section for more information.
2. A minimum interval of 2.4 ms must pass after PON is asserted before an RGBC can be initiated.

Fig x: Dokumentasjon Enable Register



I²C Write Protocol



I²C Read Protocol

A	Acknowledge (0)	Sr	Repeated Start Condition
N	Not Acknowledged (1)	W	Write (0)
P	Stop Condition	•••	Continuation of Protocol
R	Read (1)	■	Master - to - Slave
S	Start Condition	□	Slave - to - Master

a) Les av enable-register

Sjekk først hva som ligger inne på micro:biten ved å printe directory. Programmer deretter micro:bitten til å skrive inn adressen til enable-registeret i command-registeret, bruk til slutt read til å lese av enable-register.

Husk:

- Buffer må være av typen bytes. Bruk `bytearray` hvis du ikke er vant med å bruke bytes fra tidligere.
- Read returnerer bytes som standard. Bruk [n] for å lese av hver enkelt byte, som da skal ha en verdi mellom 0 og 255.
- Antall bytes som skal leses er én med mindre du leser av fra et farge-register.

In [ ]:
```
### Din kode her ###
```

In [ ]:
```
#skriv inn tilstand på enable her
```

b) Skru på sensor

Bruk micro:bit til å skru på sensoren, deretter sjekk enable-registeret igjen.

In [ ]:
```
### Din kode her ###
```

In [ ]:
```
#skriv inn tilstand på enable her
```

c) Les av fargedata

Programmer micro:bit til å hente inn data fra et av fargeregisterene.

In [ ]:
```
### Din kode her ###
```

Prøv ut RGB-sensoren

Lukk porten brukt av ubmagic

In [ ]:
```
%ubport close
```

Last opp skript til micro:bit

In [ ]:
```
!uflash ubit/tcs34725_ub.py
```

Last inn IPython extension for RGB-sensoren

In [ ]:
```
%load_ext rgbmagic
```

Start grafisk display

Trykk på Button A for å starte og stoppe avlesning. Plasser farget papir over sensoren og se hva som skjer.

In [ ]:
```
%rgb open
```

Endring av innstillinger på sensoren

Prøv å endre innstillingene AGAIN og ATIME.

AGAIN

Control Register (0x0F)

The Control register provides eight bits of miscellaneous control to the analog block. These bits typically control functions such as gain settings and/or diode selection.

Figure 32:
Control Register

7	6	5	4	3	2	1	0
Reserved						AGAIN	

Fields	Bits	Description	
Reserved	7:2	Reserved. Write as 0.	
AGAIN	1:0	RGBC Gain Control.	
		FIELD VALUE	**RGBC GAIN VALUE**
		00	1X gain
		01	4X gain
		10	16X gain
		11	60X gain

In [ ]:
```
%rgb again 3
```

ATIME

RGBC Timing Register (0x01)

The RGBC timing register controls the internal integration time of the RGBC clear and IR channel ADCs in 2.4-ms increments. Max RGBC Count = (256 – ATIME) × 1024 up to a maximum of 65535.

Figure 27:
RGBC Timing Register

Fields	Bits	Description			
		VALUE	INTEG_CYCLES	TIME	MAX COUNT
ATIME	7:0	0xFF	1	2.4 ms	1024
		0xF6	10	24 ms	10240
		0xD5	42	101 ms	43008
		0xC0	64	154 ms	65535
		0x00	256	700 ms	65535

In [ ]:
```
%rgb atime 240
```

Kjør cellen når ferdig

In [ ]:
```
%rgb close
```