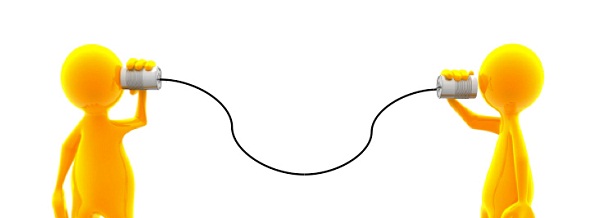
Serial communication – rapport

Oppgaven går ut på å lage et system som kan kommunisere med andre enheter. I denne oppgaven skal vi gå dypt inn i hvordan kommunikasjon foregår, og hvilke typer kommunikasjon vi har.

Kravspesifikasjon:

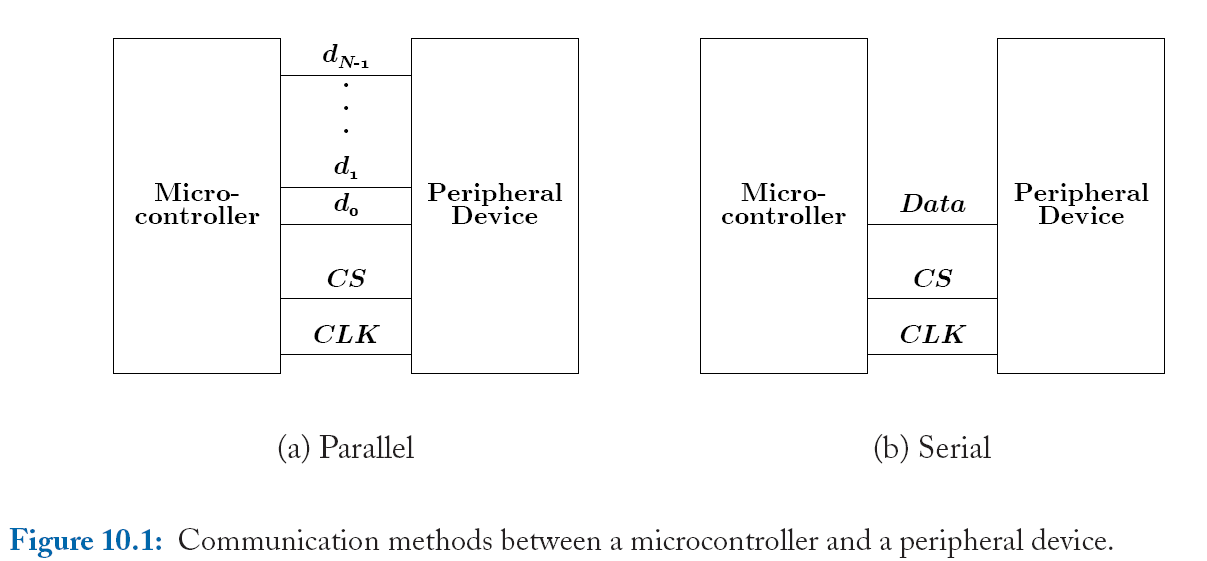
* Vi kan få 2 eller flere arduino boards til å kommunisere med hverandre.
* En av arduinoen skal sende en slags data/informasjon videre til den andre.
* Mottakeren skal kunne behandle informasjonen den har mottatt.

Det viktigste for å kunne løse denne oppgaven er å kunne forstå hvordan kommunikasjonen foregår. Videre skal jeg drøft rundt nettopp det.

Før to eller flere enheter kan i det hele tatt kommunisere, må de bli enige om hvilket språk de skal kommunisere på. Det er akkurat som mennesker. For eksempel hvis to mennesker skal snakke sammen, så hjelper det ikke at den ene kan spansk og den andre tysk. De må bli enige og kommunisere på et språk begge kan, som for eksempel engelsk. Det samme gjelder for Embedded systems. Her heter språket **protokoll**. Akkurat som språk hos mennesker, så finnes det ett sett med regler her og. Dette er noen av reglene.

* **Data format** for informasjonen som sendes. Dataen som sendes er delt i to deler, «header» og «data». Selve dataen er lagret i «data» området. «header» inneholder informasjon som er relevant i forhold til protokollen.
* **Adresse format:** Adresser er brukt for å identifisere både sender og mottaker. Adressen er lagret i «header». Dette tillater mottakerne å lese «header» informasjon, videre kan de bestemme om dataen er sendt til den mottakeren leser eller ikke. Mottakerne kan enten prosessere data om den er sendt til dem eller ignorere.

Alt dette er bestemt i protokollen. Og dette og mye mer må de bli enige om før de kan begynne å kommunisere. Akkurat som mennesker. Begge må ha de grunnleggende grammatiske reglene på plass, før de kan begynne å snakke sammen og forstå hverandre.

**Parallell eller serial:** Det finnes to typer kommunikasjon, parallell eller serial. Forskjellen er at ved parallell så finnes det flere «linjer», der kommunikasjonen foregår, og disse linjene ligger parallelle. Mens ved serial så foregår kommunikasjonen over én linje, og alle bits av data sendes i serie, ergo serial.

**Fordeler og ulemper:** Parallell kommunikasjon foregår relativt raskere enn serial, men ulemper ved parallell er at det bruker mer plass og trenger flere ledninger. Det vil være dyrt hvis man skal kommunisere parallelt over 100m, der man ved parallell trenger flere ganger 100m ledninger, så trenger man bare én ledning ved serial. Derfor er parallell mest egnet for små kretser. Selv om serial er langsommere enn parallell er det billig.

**Master og slave:** Under kommunikasjonen får enhetene tildelt rolller. Det finnes en master og det kan være flere slave. Master bestemer hvordan og når kommunikasjonen skal foregår. I tillegg har den «kontroll» over andre enhetene, som kalles for slave. Det er masteren som bestemmer hvilken salve som skal motta dataen, og hvilken som skal sende data tilbake. Når en master skal kommunisere med én bestemt slave, så sender masteren aller første den bestemte adressen til slaven den skal kommunisere med. Når slaven har mottatt og lest adressen, begynner masteren og slaven å kommunisere.

**Duplex og half-duplex:**

**Duplex:** beskriver hvordan kommuniaksjonen foregår mellom enhetene. Duplex vil si at begge enhetene kan sende og motta data på en og samme tid. I dette tilfelle er ikke fare for kollisjon av dataen, så overføringen kan foregå mye raskere. Et eksempel på duplex kommunikasjon er telefon/mobil.

**Half-duplex:** vil si at en enhet kan kommunisere av gangen. Så snart en enhet har sendt sin data, er det den andre enhetens tur til å kommunisere. Hvis begge kommuniserer på samme tid, vil det foresake kollisjon. Et godt eksempel på half-duplex kommunikasjon er walkie talkies.

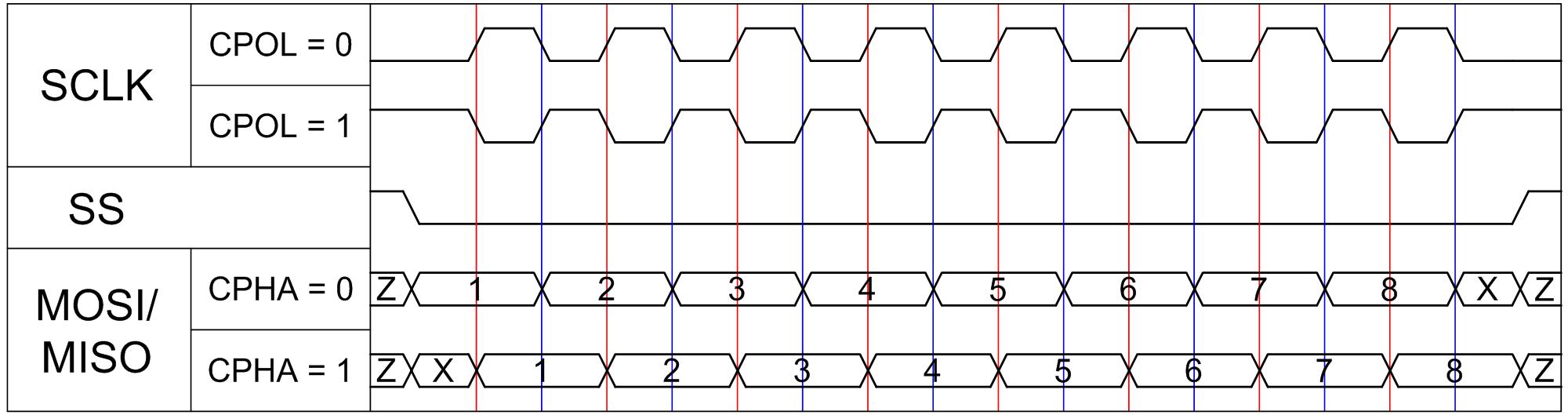
**Eksempler på protokoller:**

**SPI:** Eller uttalt ”spy”, er en serial kommunikasjons type. Det er synkronisert kommunikasjon hvilket betyr at kommunikasjonen er synkronsert etter en bestemt klokke. Klokken har en seperat linje og klokke sørger for at kommuikasjonen er i perfekt sync. Klokken brukes til å fortele mottakeren/slaven når den skal sample på dataen. Det kan feks være når klokken går fra low til high, eller når den går fra high til low. Dette avhenger av protokollen. Når mottakeren oppdager denne endringen vil den gå å sample den neste biten. Siden klokken har en egen linje, så er det ikke nødvendig å spesifere overføringshastigheten.



****Ved SPI er det en enhet som generer klokken (SCK/CLK), og denne enheten kalles for master. Og de andre enhetene kalles for slave. Under hele kommunikasjonen vil det kun være en master, men det kan være flere slaves. SPI krever 4 tilkoblinger, SCK(klokke), MOSI (master-out slave-in), MISO(master-in slave-out) og SS(slave select). Når dataen er sendt fra masterenheten til slaven, så blir dataen sendt over MOSI. Hvis slaven skal respondere tilbake så sender den data tilbake over MISO, og masteren fortsetter å genere en forhåndavtalt klokke sykluser. Det er forhåndsavtalt siden master vet at slaven skal returnere data, og størrelsen på den. Hvis feks masteren skal lese temperaturmåling fra en slave-enhet, så vesPt masteren at slaven skal returnere ett tall.

I tlllegg er SPI duplex, hvilket betyr de har separate linjer for å sende og motta data. I tillegg har en SPI en SS(slave-select) linje, som brukes av masteren til å velge hvilken slave den snakker med.

SS-linjen er activ low, som betyr at den normalt er high. Før dataen sendes til slaven, så blir linje low, og forblir low helt til kommuniikasjonen er over, dette aktiverer slaven. Når kommunikasjonen er over, så blir liinje høy igjen. Dette kan vi se på bildet over. Hele tiden under kommunikasjonen er SS low.

Det sendes det ikke noe adresse, masteren velger en slave via SS, og sender data over MOSI, og mottar på MISO.

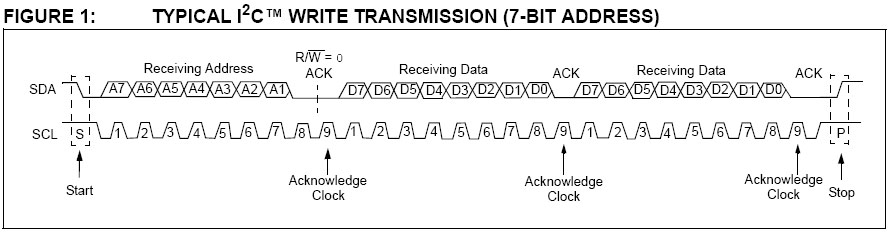
XCCCXXCXCXcxxXCCXCXplsdCXXCPDSÅPDSÅPLXCÅPLDSÅPLc

**UART:** Universal asynchronous reciever/transmitter er en serial kommunikasjonstype. Kommunikasjonen er ikke synkronisert etter forhåndsbestemt klokke, men enhetene må bruke en internal klokke for å finne ut av når den neste bit kommer. Mottakeren og senderen blir enig om en overføringshastighet på forhånd. Som feks 9600 bits per sekund. Det sendes minst en start bit, også en stop bit. Dette betyr at for 10 bits av overføring, er det kun 8 bits som er data. I tillegg er kommunikasjonsenhetene begrenset til kun 2 stykker.

**I2C:** er en annen synkronisert protokoll. Settoppet krever to ledninger pr enhet, den kan støtte opp til 1008 slaveenheter, og i motsetting til SPI, kan I2C støtte flere master enheter, som tillater flere master å bruke databusen og kommunisere med slave-enhetene. Det er viktig å huske at master-enhetene kan kun kommunisere i tur, og at de kan ikke kommunisere samtidig, og at master-enhetene ikke kan kommunisere med hverandre.

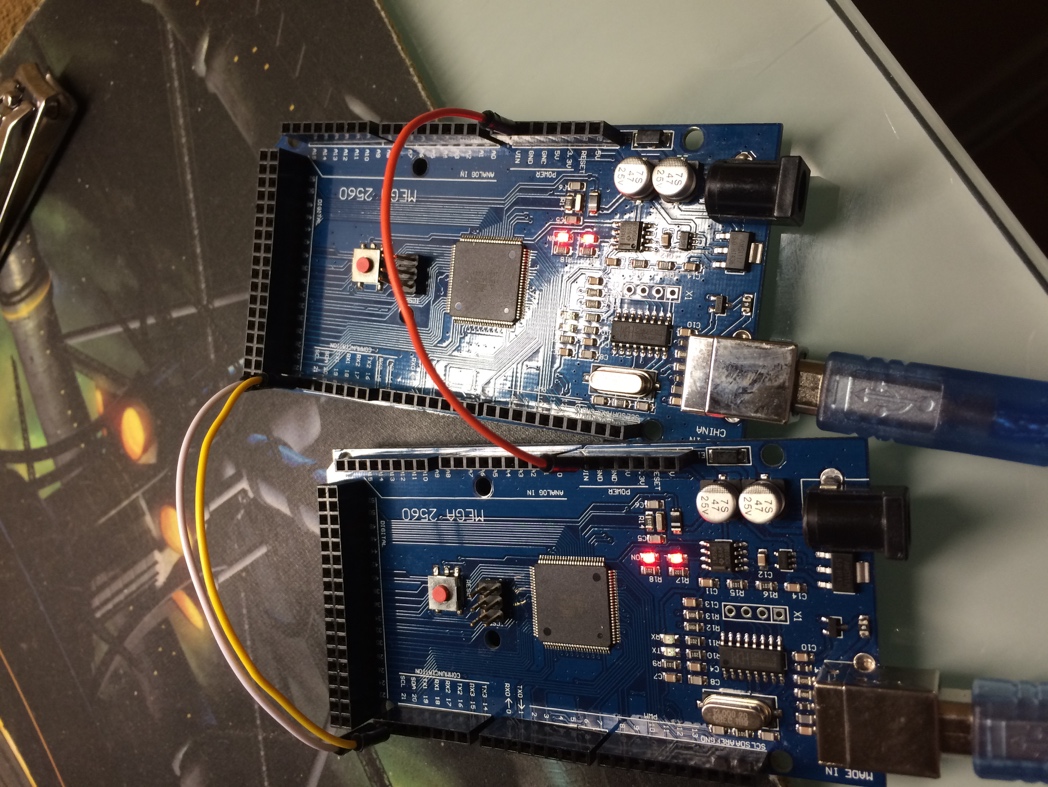
Data hastigheten er mellom synkronisert serial og SPI, som er 100kHz og 400kHz.I2C kommunikasjon krever to linjer, SCL og SDA. SCL er klokke og SDA er data. Klokken er generert av masteren Noen ganger kan slave-enheter manipulere klokke, for å få ekstra tid til å behandle dataen og være klar igjen. I motsettning til SPI og USART foregår kommunikasjonen ved noe som heter ”open drain”, hvilket betyr enhetene kan dra dataen linjen low, men ikke high.

For å indikere at masteren sender en adresse, så lar masteren SCL være høy og endre SDA til lav. Dette gjør alle slave enhetene oppmerksomme på at en overføring er i gang.

****

**Meldingene er delt opp i 2 stykker, først en adresse del, hvor master indikerer hvilken slave den snakker med, og en data del hvor en 8-bit data melding er sendt fra masteren til en bestemt slave. Overføringen av data starter når SCL går lav og blir samplet hver gang SCL går høy.**

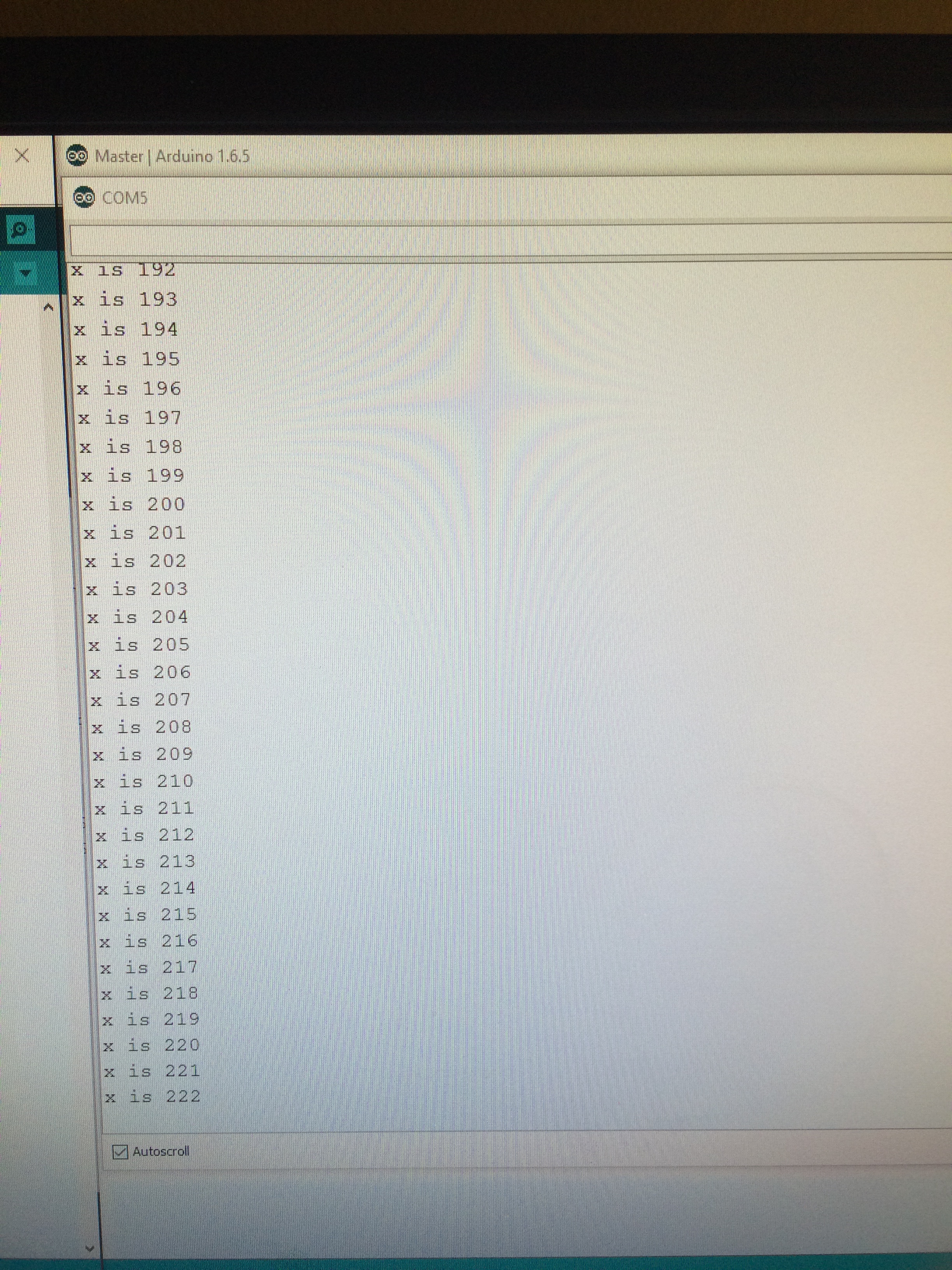
**Vår løsning:**

Vi har brukt TWI, eller som vi kjenner det I2C. Her er den ene arduinoen master og den andre er slave.

**Master:** Vår master etablerer en variable x, og øker den i loop. Masteren sender teksten ”x is” og variable x til slaven.

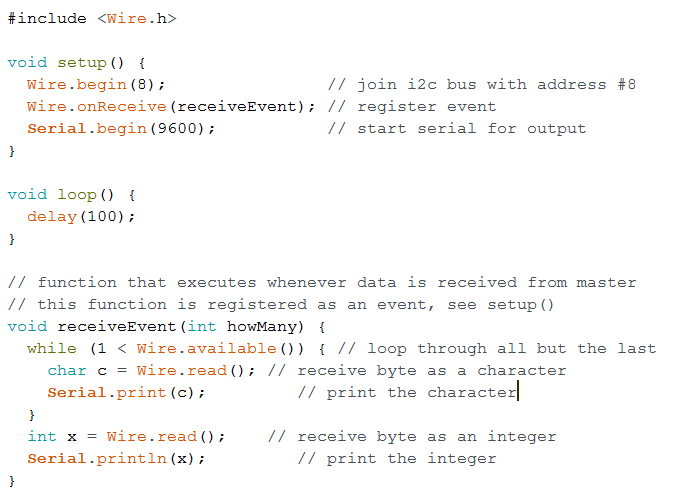
**Slave:** Vår slave mottar dataen, og skriver denne ut i serial monitor.

Under bildet av arduinoene kan man se resultatet av prosjektet. Det er en screenshot av serial monitor fra slaven.

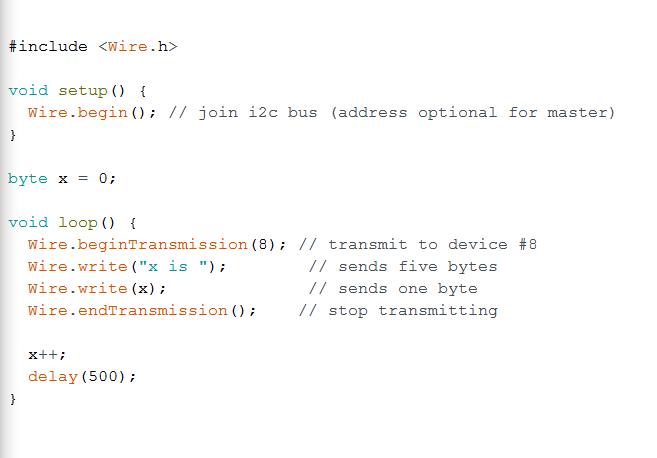


**Kode:**

**Master:**

****

**Slave:**



**Konklusjon:** I dette prosjektet har vi laget et system som gjør det mulig for 2 arduino boards å kommunisere. Vi har brukt TWI eller I2C for å gjøre kommunikasjonen mulig. Dette prosjektet har vært vellykket. Vi har ikke møtt på noen problemer i dette prosjektet, og prosjektet har vært vellykket.

Kilder:

<http://www.cabledepot.com/05DTSERPAR.html>

<https://en.wikipedia.org/wiki/Communications_protocol#Basic_requirements_of_protocols>

<https://learn.sparkfun.com/tutorials/i2c>

<https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>

http://searchnetworking.techtarget.com/answer/The-difference-between-half-duplex-and-full-duplex