# X10\_Sender (CBJ)

Sender softwaren er kodet i C++ og er lavet som en klasse. Det er designet til at kunne virke med en ATmega2560. I vores tilfælde har vi brugt en Arduino 2560. X10\_sender har til opgave at kompilere en data pakke. Dette gør den ud fra hvem modtageren er, hvad type besked det er og hvad data der skal sendes. Herefter sendes pakken som et X10-indkodet-signal over lysnettet.

Ved X10-indkodet-signal menes der at der sendes et logisk 1 på en rising edge fra zero-cross, og der sendes et logisk 0 på en faling edge fra zero-cross. Så når det næste bit der skal sendes er et logisk 1, bliver der sendt et signal når der er en rising edge fra zero-crossen.

Signalet der bliver sendt er et 1 ms burst af et 120 kHz firkant signal. Dette bliver realiseret via af timer 1 der er indstillet til Mode 8: PWM phase and frequency correct, som så toggler OC1A. ICR1 er loadet med en værdi der sammen med prescaleren giver en frekvens på de 120 kHz. For at det kun bliver sendt i 1 ms bruges timer 0 i normal mode sat op med en interupt, så den i interupt rutinen stopper PWM signalet.

Senderen er implementeret således at der via et enkelt metodekald kan sendes en ønsket besked til modtageren. Der kompileres en data pakke og programmet bliver sat til send mode. Når den er i denne mode begynder senderen at sende vært bit ved det relevante interupt fra zero-cross. Efter at en pakke er blevet sendt går senderen ud af send mode.

Der blev først testet for om der kunne produceres et PWM signal med den korrekte frekvens. Det blev derefter testet for at PWM signalet kunne blive sendt et 1 ms burst. Alt dette blev målt og verificeret med et Analog Discovery oscilloskop.

Til at teste senderens evne til at kompilere en pakke og derefter sende denne pakke. En af knapperne på arduino shielded blev sat op til at sende en bestemt pakke. I første test blev de to interupt fra zero-cross simuleret via to knapper. Der blev så observeret på oscilloskopet at der blev sendt et burst når det rigtige interupt blev triggert, og der ikke skete noget ved det modsatte interupt. Der blev derefter sammen lignet med de binære værdier som pakken bestod af i koden, og de binære værdier der kunne observeres der blev sendt. Der var nogle problemer med at sende pakken i starten, som hovedsagligt var på grund af nogle mindre fejl i en del af funktionerne. Da de var blevet rettet kunne der sendes en hel pakke.

Det blev så testet med de forskellige typer af beskeder også virkede.

Sidste test var at køre senderen i fuld hastighed, og at det virkede med vores hardware. Der kunne aflæsses på vores oscilloskop at pakken blev afsendt og at der kom et signal burst på de rigtige steder på de 50 Hz sinus. Der var også et modtager kredsløb der var sat op til at vise hvilken type pakke den havde modtaget. Der var ingen problemer med senderen i denne test.

# X10\_Modtager (CBJ)

Modtager softwaren er kodet i C++ og er lavet som en klasse. Det er designet til at kunne virke med en ATmega2560. I vores tilfælde har vi brugt en Arduino 2560. X10\_Modtager har til opgave at modtage X10 kommunikation og løbende validere om pakken er til sig selv, hvad data pakken indeholder og om hele pakken er sendt intakt.

Modtageren er sat op med de samme to interupt til zero-crossen, som i sender koden. Når der kommer et interupt på rising eller faling, tjekkes der i 1 ms om der er et højt signal fra envelopen. Hvis der ses et højt signal tager den det som enten et Logisk 1 hvis det skete på rising edge, ellers som et logisk 0 hvis det sker på faling edge. Kommer der ikke noget på hverken rising eller falling, resettes modtageren.

Koden er implementeret så at modtageren altid vil kunne modtage et signal, medmindre der er bliver sendt X10\_Sender fra samme mikrokontroller. Når der først bliver modtaget nået data, blive det tjekket om den overholder protokolen. I starten af en pakke bliver der tjekket om den har de rigtige start bit. Efter dette valideres at adressen passer med mikrokontrollerens egen. Hvis en af disse to fejler i validere, stopper modtageren i at gemme pakken. Den venter så på de 6 slut bit, eller til der ikke er blevet sendt data i et stykke tid.

Sidste del af pakken indeholder hvilken type besked det modtages, og dens data. Her efter modtages et paritets bit, efter fuldt af de seks slut bit. Der tjekkes så om der er paritets fejl. Der sættes så et flag om at der er modtaget en pakke.

Det er designet så der via et funktions kald kan tjekkes om der er data klar til at blive læst. Når der er data klar kan man derefter via et andet funktions kald hente hele pakkens data, og sammentid resettes modtageren så der kan modtages data igen.

Der blev i første omgang testet om en bestemt type pakke kunne modtages. Interupt blev simuleret med tryk knapper, det samme med signalet fra envelopen. Der var mange forskellige testere til forskellige iteretioner, på grund af de mange fejl der var i starten. Der blev brugt LED’erne fra shielded til at debugge programmet. I en af de første iterationer blev LED’erne brugt til at se om modtageren kom ind i de rigtige dele af funktionen på de rigtige tidspunkter. Det viste sig at problemet skyldtes fejl i trackingen af hvor meget af pakken der var blevet modtaget. Det tog et stykke tid at fikse problemet da det skyldtes et design problem, og det derfor var alle steder hvor en form for tracking af positionen blev brugt. Da fejlen var udbedret fungerede koden som den skulle.

Der blev så kørt en test hvor LED’erne viste hvilken pakke der blev modtaget. Denne test var en succes, og viste præcis den pakke der blev sendt.

Den sidste test var at kunne modtage data fra senderen gennem hardwaren, og med 50 Hz AC. Dette virkede ikke i første omgang. Efter nogle rettelser i interupt rutinerene for zero-crossen, kunne der nu også modtages data i fuld hastighed fra senderen