Home Security System

- Styring og monitorering af enheder over lysnettet

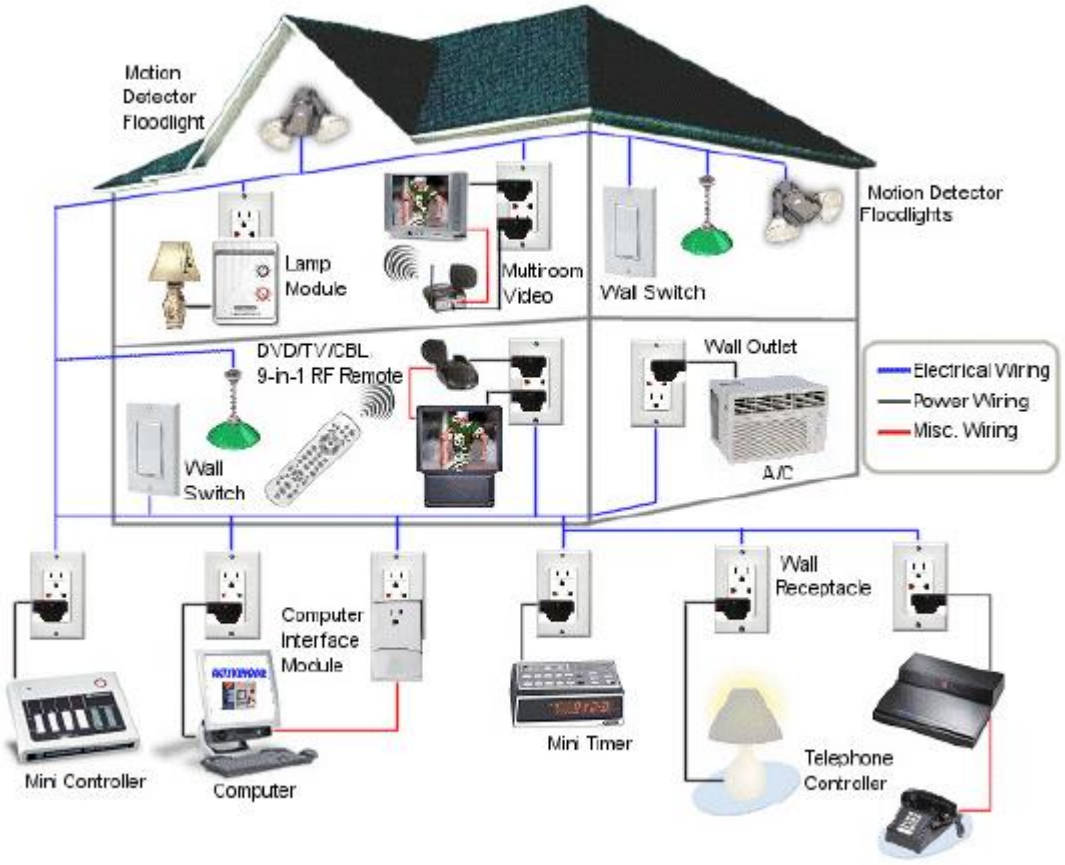
E2PRJ2 – Projektdokumentation

- Gruppe 10

31-05-2013

Vejleder: Steen Fredborg Krøyer

Navn	Initial	Studienummer	Underskrift
David-Samuel Buhauer	**DB**	201270749	
Jacob Mose Hansen	**JMH**	201270410	
Jacob Bak Mortensen	**JBM**	201271011	
Yusuf Sarikaya	**YS**	201271032	
Jonathan R. Loftheim	**JRL**	201270859	



Indholdsfortegnelse

Kravspecifikation

Versionshistorik

Ver.	Dato	Initialer	Beskrivelse
1.00	11.03.13	Alle	Tilpasset faget "Indledende System Engineering"
2.0	24.05.13	Alle	Tilpasset efter sidste review med vejleder. Scenarie er blevet defineret i en fodnote.
2.1	29.05.13	JRL, DB	Rettet nogle små ting omkring hoved forløb i use cases, samt rettet for grammatiske fejl.

Indledning

Dette dokument specificerer kravene til projektet Home Automation/Home Security.

Indhold

Dokumentet indeholder

- Specifikation af funktionelle krav vha. use cases
- Specifikation af ikke-funktionelle krav
- Forslag til brugergrænseflade

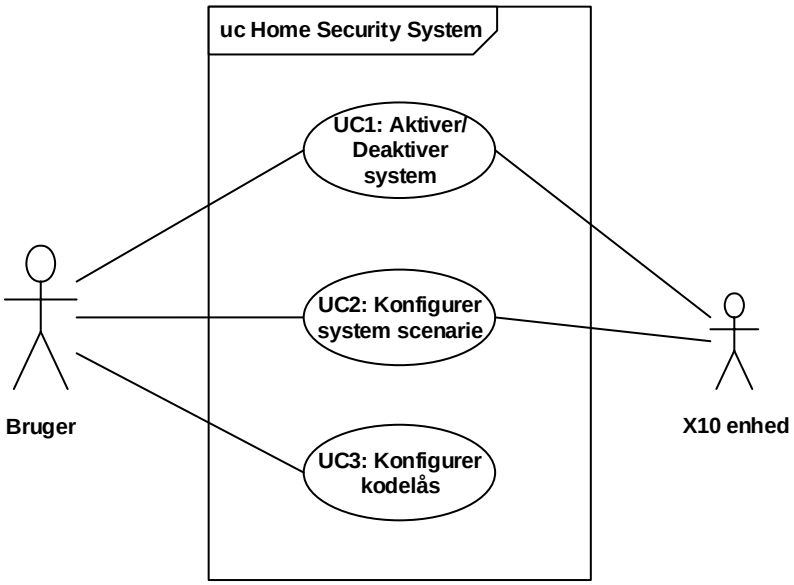
Systembeskrivelse

Security systemet skal kunne simulere aktivitet i hjemmet dermed ment, at det skal se ud som om, at der er nogen hjemme. Systemet skal kunne styres fra en PC som serielt kan kommunikere med X10-controller (STK-500).

X10-controlleren skal kunne styre lyset i huset og holde det tændt eller slukket i tidsintervaller i alle husets rum. Derud over skal der være en kodelås (DE2 board) til X10-controlleren således at systemet kun kan tilgås af brugeren når systemet er låst op.

Funktionelle krav

Nedenstående use case diagram sammenfatter aktører og use cases for systemet. Efterfølgende underafsnit giver en detaljeret beskrivelse af de enkelte use cases.



Figur 1 - Use Case diagram for Home Security System

Aktørbeskrivelse

Aktørnavn:	Bruger
Alternativ reference	-
Type:	Primær
Beskrivelse:	Bruger vælger scenarie[1] for systemet. Det er ligeledes bruger der tænder og slukker systemet. Ligeledes er det bruger der administrerer og konfigurerer systemet alt efter ønskede scenarier. Samtidigt sørger bruger også for opsætning af X.10 enhed.

Aktørnavn:	X.10 enhed
Alternativ reference	-
Type:	Sekundær
Beskrivelse:	Får inputs fra systemet og aktivere/deaktivere alt efter valgte scenarie

[1] Scenariet er defineret ved, at bruger kan vælge enten at tænde eller slukke for en lampe, samt indstille tids intervaller.

	Bruger	X.10 Enhed
Aktiver/Deaktiver system	Primær	Sekundær
Konfigurer system scenarier	Primær	Sekundær
Konfigurer kodelås	Primær	-

Tabel 1 - Beskriver hvilken funktion hver aktør har for bestemte use cases

Use case 1: Aktiver/Deaktiver system

Navn:	**Aktiver/Deaktiver systemet**
Use case ID:	1
Samtidige forekomster:	1
Primær aktør:	Bruger: Ønsker at aktiver/Deaktiver systemet.
Interessenter:	X.10 enhed
Forudsætninger:	UC 3: Konfigurering af kodelås har været succesfuld.
Resultat:	Systemet er aktiveret/deaktiveret.

Hovedforløb:

1. Bruger åbner Windows prompt på PC.
2. Bruger trykker på en specifik knap på bruger-interfacet programmet.
   a. Bruger vælger aktivering af system ved tryk af ON-knap.
      1a. Signal om aktivering sendes til X.10 kontrolleren.
      2a. X.10 kontrolleren modtager signalet og aktiver system.
   b. Bruger vælger deaktivering af system ved tryk af OFF-knap.
      1b. Signal om deaktivering sendes til X.10 kontrolleren.
      2b. X.10 kontrolleren modtager signalet og deaktivere system.

Use Case 2: Konfigurer system scenarie.

Navn:	**Konfigurer system scenarie.**
Use case ID:	2
Samtidige forekomster:	1
Primær aktør:	Bruger: Ønsker at konfigurer systemet efter valgt scenarie.
Interessenter:	X.10 enheder.
Forudsætninger:	UC1: Aktivering af system scenarie har været succesfuld.
Resultat:	Bruger har konfigureret systemet efter valgt scenarie.

Hovedforløb:

1. Bruger åbner Windows prompt på PC og får en velkomst besked.
   a. Bruger trykker på en tast.
      i. '1' for Menu

1. Bruger trykker '1' for at sætte enhed
   [Bruger trykker på en tast forskelligt fra '1' og '2']
   a. Bruger vælger en husadresse mellem 'A' og 'P' på en tast.
   [Bruger trykker på en tast forskelligt fra 'A' og 'P']
   b. Bruger vælger enhed på en tast mellem '1' og '16'
   [Bruger trykker på en tast forskelligt fra '1' og '16']
2. Bruger trykker '2' for at gå tilbage til velkomst skærm.
   [Bruger trykker på en tast forskelligt fra '1' og '2']

ii. '2' for at aktivere enhed
[Ingen enhed er valgt eller huskode sat]
iii. '3' for at deaktivere enhed
[Ingen enhed er valgt eller huskode sat]

2. Det valgte scenarie overføres til X.10 kontrolleren og konfigurere scenarie til X.10 enhed.

Undtagelser:

1. *[Bruger trykker på en tast forskelligt fra '1' og '2']*
   a. Bruger får besked om at indtaste '1' eller '2'.
2. *[Bruger trykker på en tast forskelligt fra 'A' og 'P']*
   b. Bruger får besked om at indtaste et bogstav mellem 'A' og 'P'.
3. *[Bruger trykker på en tast forskelligt fra '1' og '16']*
   c. Bruger får besked om at indtaste et tal mellem '1' og '16'
4. *[Ingen enhed er valgt eller huskode sat]*
   d. Bruger får besked om at ingen enhed er valgt eller huskode sat.

Use Case 3: Konfigurer kodelås

Navn:	**Konfigurer kodelås**
Use case ID:	3
Samtidige forekomster:	1
Primær aktør:	Bruger: Ønsker at konfigurer kodelåsen på DE2 boardet.
Interessenter:	-
Frekvens:	-
Forudsætninger:	-

Bruger har konfigureret kodelås til systemet, som kan aktiveres eller deaktiveres.

Hovedforløb:

1. Bruger indtaster gamle kode.
   [*Gamle kode er indtastet forkert tre gange]*
2. Bruger indtaster den nye kode to gange.
   [*Nye kode er indtastet forkert anden gang]*

3. Den nye kode erstattes med den gamle kode.
4. Kodelåsen er konfigureret til systemet og systemet kan nu aktiveres eller deaktiveres.

Undtagelser:

1. [*Gamle kode er indtastet forkert]*
   a. Udskriver meddelelse om at den gamle indtastede kode er forkert og bruger må prøve igen.
2. [*Nye kode er indtastet forkert anden gang]*
   a. Udskriver meddelelse om at den nye indtastede kode ikke stemmer ens med det første indtastet, og at bruger skal prøve igen.

Ikke-funktionelle krav

Dette afsnit beskriver gennemgående, ikke-funktionelle krav til systemet.

Brugergrænseflader

- Brugergrænsefladen skal være Windows prompt.
- Skal have forskelige (cases) Scenarier som kan vælges ved tryk på angivne knapper
- Skal Implementeres i C++

PC med mulighed for seriel-kommunikation med X.10 kontroller

- PC skal have Windows styresystem
- Serielport

X.10 kontroller

- Tænde eller slukke lyset i huset
- Kunne lade lyset være tændt i valgte intervaller og slukke det efter
- Skal kunne sende en Adresse(destination af signalet) samt handlingskode
- Skal Implementeres og opsættes i C

DE2 Board

- Skal kunne fungere som ”kodelås” til aktivering af system
- Kodelåsen skal skrives i VHDL

Mikrochip (AN236) X.10 interface

- Skal kunne distribuere signaler fra X.10 kontroller til adressen på det lys der skal tændes/slukkes

Forsyningsnet

- Skal kunne levere en spænding på 18 V AC.

Krav til udviklingsproces og teknologi

- Modificeret V-model
  - Semesterprojekt modellen
- Dokumentation
  - Dansk
  - SysML
- Teknologier og værktøjer
  - Udvikles i C, C++ og VHDL
  - Microsoft Visual Studio 2010 anvendes til C++ og at implementere brugergrænseflade
  - Atmel studio 6 Bruges i forbindelse med C og implementering af X.10 kontroller
  - Quartus II 12.1 Bruges i forbindelse med VHDL
  - X.10 kontroller(stk500)
  - X.10 interface (AN236) mikrochip
  - Standard PC
  - DE2 Board
- Andet
  - Dokumentation udarbejdes i Microsoft Word 2010

Accepttest

Indledning

Dette dokument specificerer accepttesten af projektet Home Security jf. kravspecifikationen til samme.

Formål

Dokumentet specificerer accepttests og vil i udfyldt stand udgøre accepttestrapporten.

Før accepttesten påbegyndes udføres der tre niveauer af test:

1. Enhedstest:
Dette omfatter test af de enkelte funktioner implementeret i komponenter og klasserne (modulerne), som produktet bestående af hardware og software er sammenstykket af. Dette sker under test af hardware og software.

2. Integrationstest:
Dette omfatter test af grænseflader mellem komponenter og klasser (moduler), der indgår i det samlede system eller produkt.

3. Accepttest:
Dette omfatter en samlet test af funktionelle krav fra kravspecifikationen for hele systemets funktionalitet.

Dette afsnit omhandler testniveau 3. Accepttesten.

Omfang

Accepttesten dækker over relevante tests af de forskellige use cases beskrevet i kravspecifikation dokumentet.

Testspecifikation

Testspecifikation for **Home Security.**

Hardware- og Softwarekomponenter

Følgende komponenter indgår i denne accepttest. De forskellige komponenter identificeres entydigt ved navn, versionsnummer samt releasedato.

Software der skal testes:

Software	Version	Release dato	Bemærkning
Brugergrænsefladen, Windows prompt.	1.0	27/05-13	Det meste af koden virker, men mangler kodelås og aktivering/deaktivering af enhed.
C kodestump til STK-500:	1.1	27/05-13	XSenderDriver virker og er enhedstestet på STK-500 ved observation af LED'er der viser '1' og '0'
VHDL kode til DE2boadet.	1.0	13/05-13	Virker

Hardware der skal testes:

Hardware	Version	Release dato	Bemærkning
AN236 X.10 enheder X.10 sender			

Test case 1: Aktiver/Deaktiver systemet

Use case under test: 1: Aktiver/Deaktiver systemet.

Forudsætninger: -

Hovedscenarie

Step	Handling	Forventet observation	Resultat
1	Bruger indtaster kode på kodelåsen = "110"	Lys diode tænder ud fra den nedtrykkede knap	De rigtige lys dioder lyser ud fra de rigtige nedtrykkede knapper
2	Kodelåsen tjekker kodens gyldighed.	Grøn lys diode lyser	Grøn lys diode lyser konstant
3	Signal om	Lys led indikator på STK500	Lys led tændes og slukkes på

	aktivering/deaktivering sendes til X.10 kontrolleren.	lyser alt efter hvilket bit mønster der bliver sendt.	STK500 i takt med et bestemt bit mønster.
4	X.10 kontrolleren udfører det valgte scenarie	Det valgte scenarie igangsættes.	De valgte enheder tænder eller slukker

Undtagelse 1: Kodelåsen er ugyldig

Step	Handling	Forventet observation	Resultat
1	Systemet beder bruger om at indtaste kode igen.	Start lys diode tændes på DE2 board, hvilket indikerer at systemet er klar til ny kode input forsøg.	Start lys dioden tændes og der kan igen trykkes en kode.

Test case 2: Konfigure system scenarie

Use case under test: 2: Konfigurer system scenarie

Forudsætninger: Test af Use case 1: Aktiveringen/Deaktivering af systemet har været succesfuld.

Hovedscenarie

Step	Handling	Forventet observation	Resultat
1	Bruger indtaster kode på kodelåsen = ”110”	Lys diode tænder ud fra den nedtrykkede knap	De rigtige lys dioder lyser ud fra de rigtige nedtrykkede knapper
2	Kodelåsen tjekker kodens gyldighed.	Grøn lys diode lyser	Grøn lys diode lyser konstant
3	Signal om aktivering af system sendes til X.10 kontrolleren.	X.10 kontrolleren modtager signalet og aktiver systemet, samt tænder for en led indikator	Led indikator er tændt på X.10 kontrolleren
4	Bruger vælger hvilken enhed der skal have et nyt senarie.	Konsol app. Accepterer de nye værdier for enheden.	Konsol app. Viser en bekræftelses besked.
5	Bruger vælger om X.10 enhed skal køre det ny ændret scenarie.	Det valgte scenarie igangsættes.	De valgte enheder tænder eller slukker

Undtagelse 1: Kodelåsen er ugyldig

Step	Handling	Forventet observation	Resultat
1	Systemet beder bruger om at indtaste kode igen.	Start lys diode tændes på DE2 board, hvilket indikerer at systemet er klar til ny kode input forsøg.	Start lys dioden tændes og der kan igen trykkes en kode.

Test case 3: Konfigurer kodelås

Use case under test: 3: Konfigure kodelåse

Forudsætninger: -

Hovedscenarie

Step	Handling	Forventet observation	Resultat
1	Bruger åbner konfigurationsprogrammet	Konfigurationsprogram er åbent.	Windows konsol applikation er åbnet.
2	Brugeren vælger ny kode kombination.	Implementeringen af den nye kode er succes.	4 grønne leds på DE2 board lyser
3	Bruger trykker den nye kode	Kodelåsen tænder grøn led ud fra den trykkede knap	De rigtige lys dioder lyser ud fra de rigtige nedtrykkede knapper
4	Kodelåsen tjekker kodens gyldighed.	Grøn lys diode lyser.	Grøn lys diode lyser konstant

Undtagelse 1: Input data fejl

Step	Handling	Forventet observation	Resultat
1	Bruger får besked om at der er opstået en fejl. Use case afsluttes	Bruger får besked om fejl og programmet lukkes ned.	Konsol app. Viser at der er en fejl og lukker herefter ned.

Systemarkitektur

Versionshistorik

Ver.	Dato	Initialer	Description
1.00	05.04.13	Alle	Tilpasset faget “Indledende System Engineering”
2.00	29.04.13	Alle	Tilpasset Sys ML diagrammer med beskrivelse efter møde med Kim.
2.01	08.05.13	Alle	Opdateret signalbeskrivelse og SW sekvensdiagram for kodelås.
2.1	29.05.13	JRL	Rettet det til, så det passer mere med vores use cases og resten af projektet

Indledning

Dette dokument beskriver systemarkitekturen for projektet ”Home Security System”, som formuleret i projektbeskrivelsen og specificeret i kravspecifikationen.

Formål

Formålet med dette afsnit er:

- At fastlægge systemets overordnede HW- og SW-komponenter, drevet af kravene specificeret i kravspecifikationen
- At fastlægge grænsefladen mellem systemets overordnede komponenter
- At identificere arbejdsopgaver for projektets design- og implementeringsfase

Refereret dokumentation

- Kravspecifikation for projektet

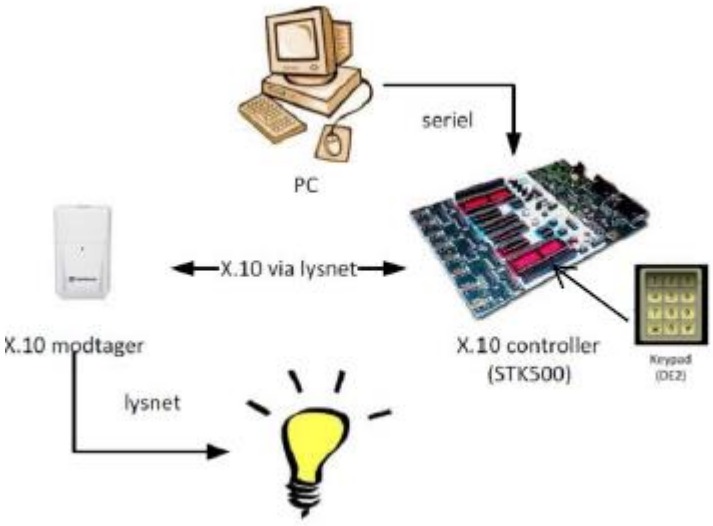
Systemkomponenter

Der er gennemført en analyse af kravene som givet i kravspecifikationen, og på baggrund af disse er der truffet følgende beslutninger om systemets komponenter og deres placering:

- Brugerens tilgang til systemet er en PC, der kommunikerer med X.10 kontrolleren bestående af en 32 bit Microcontroller over et elektrisk interface "AN236, som link mellem PC og X.10 nettet. På Pc'en ligger ligeledes et brugervenligt program til at kontrollere eksterne X.10 enheder i systemet som vælger et scenarie.

- *DE2 board*
  Programmeres i VHDL og bruges som kodelås til systemet, så brugeren skal have indtastet en rigtig kode kombination, før brugeren kan ændre i X.10 kontrollerens forskellige værdier.
- *STK500 KIT*
  Programmeres i C og bruges som X.10 kontroller, hvor systemet videre sender et bit mønstre, som kommer fra bruger interface, til en X.10 enhed.

- *X.10 enhed(er)*
  Er den enhed som modtager et bit mønstre fra X.10 kontroller og udføre bestemt handling alt efter hvilket mønstre der bliver modtaget. I vores tilfælde skal den kunne ændre tænde og slukke tiden på en lampe/lys diode.

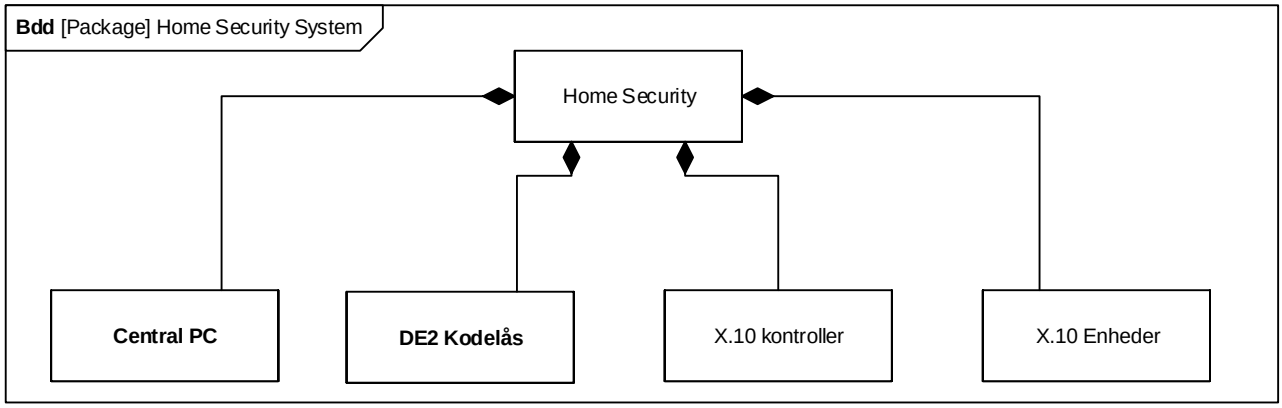


Figur 2 Overordnet ide om system komponent sammenhæng

Hardware Arkitektur

Block Definitions Diagram

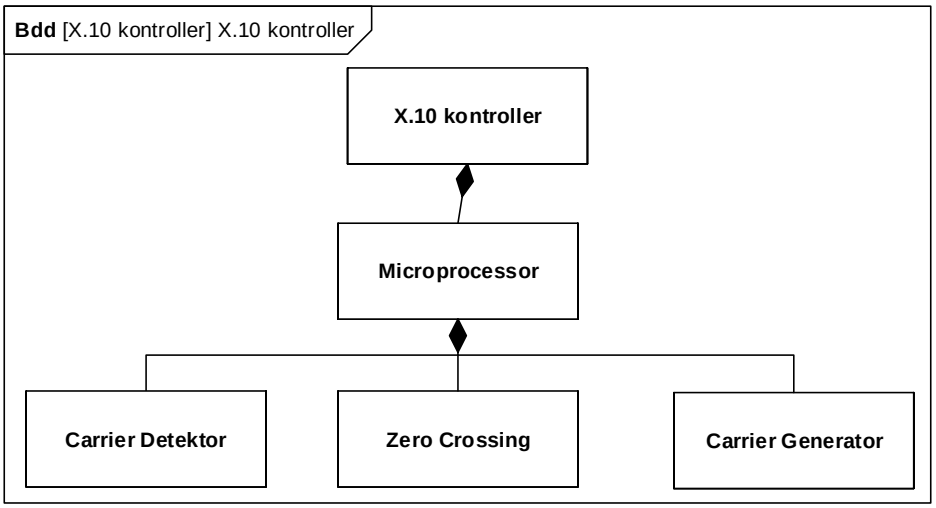
Home Security System



Figur 3 Block Definitions Diagram for hele system

Overstående er en oversigt over hele systemet. Home Security Systemet består af central PC, en DE2 Kodelås, en X.10 kontroller og en x antal X.10 Enheder.

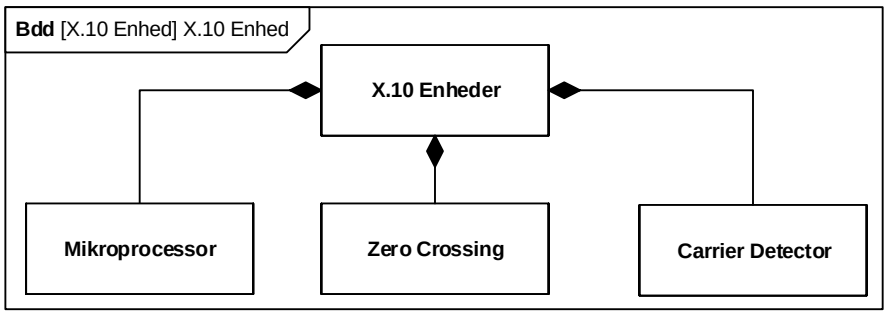
X.10 Kontroller



Figur 4 - BDD for X.10 kontroller

Figur 4 viser blokstrukturen for X.10 kontrolleren. Denne består af en Mikroprocessor, en Zero-Crossing funktion og en Carrie Generator.

X.10 Enhed



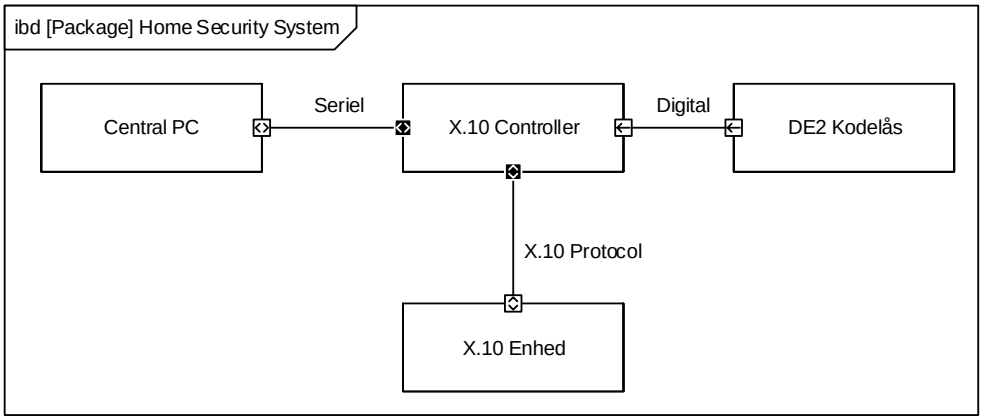
Figur 5 – BDD for X.10 Enhed

Figur 5 viser blokstrukturen for X.10 enhed. Denne består af en Mikroprocessor, en Zero-Crossing funktion og en Carrie Detektor.

Intern Block Diagram

Systemets struktur er i dette afsnit beskrevet ved hjælp af Sys ML struktur-diagrammer.
Dette diagram definerer systemets grænseflader, og interne blokke.

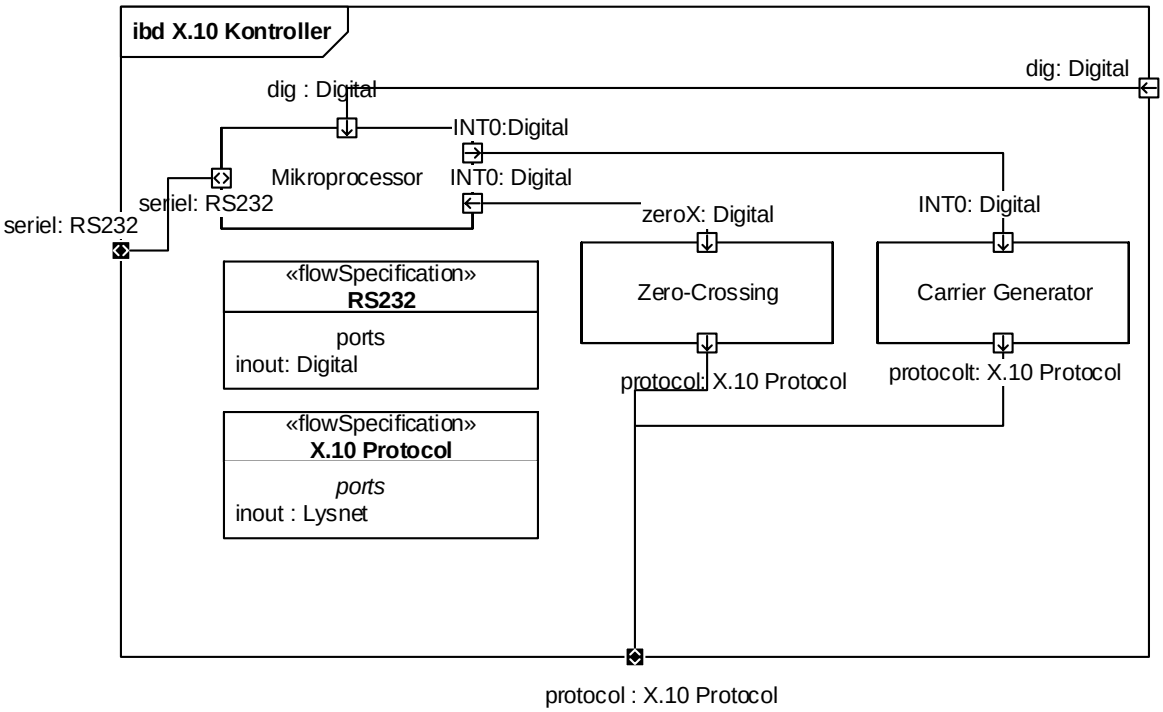
Home Security System



Figur 6 - Internal Block Diagram for hele Home Security System

Viser hvordan de forskellige ting er forbundet I forhold til hinanden og med hvilke signaler der bliver brugt.

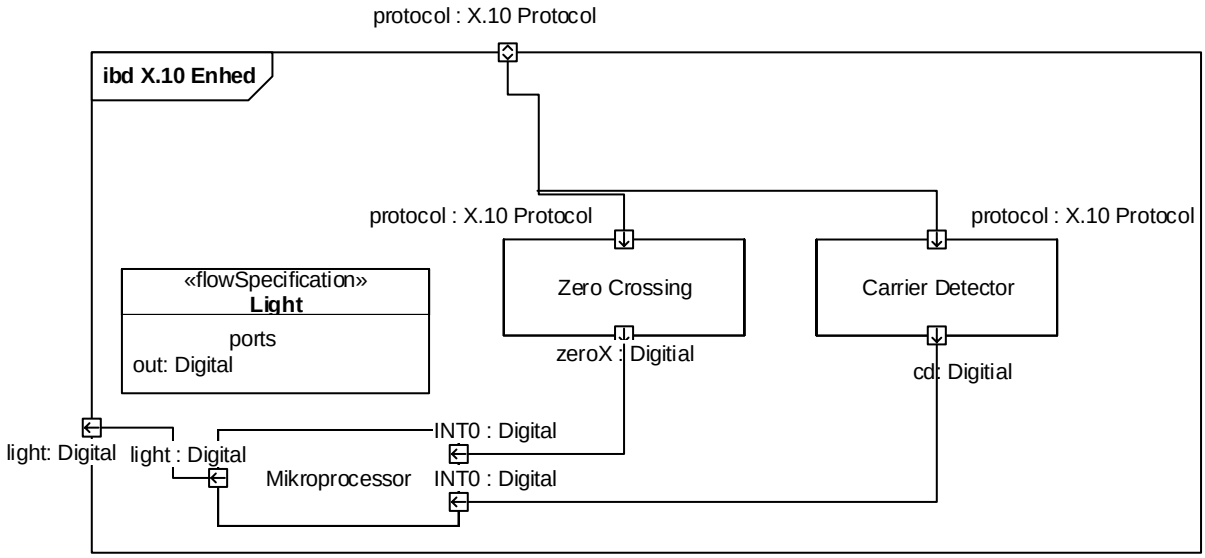
X.10 Kontroller



Figur 7 - IBD for X.10 Controller

Viser hvilke enheder X.10 kontrolleren består af og hvordan de snakker sammen, samt hvilke i/o-porte den har.

X.10 Enhed

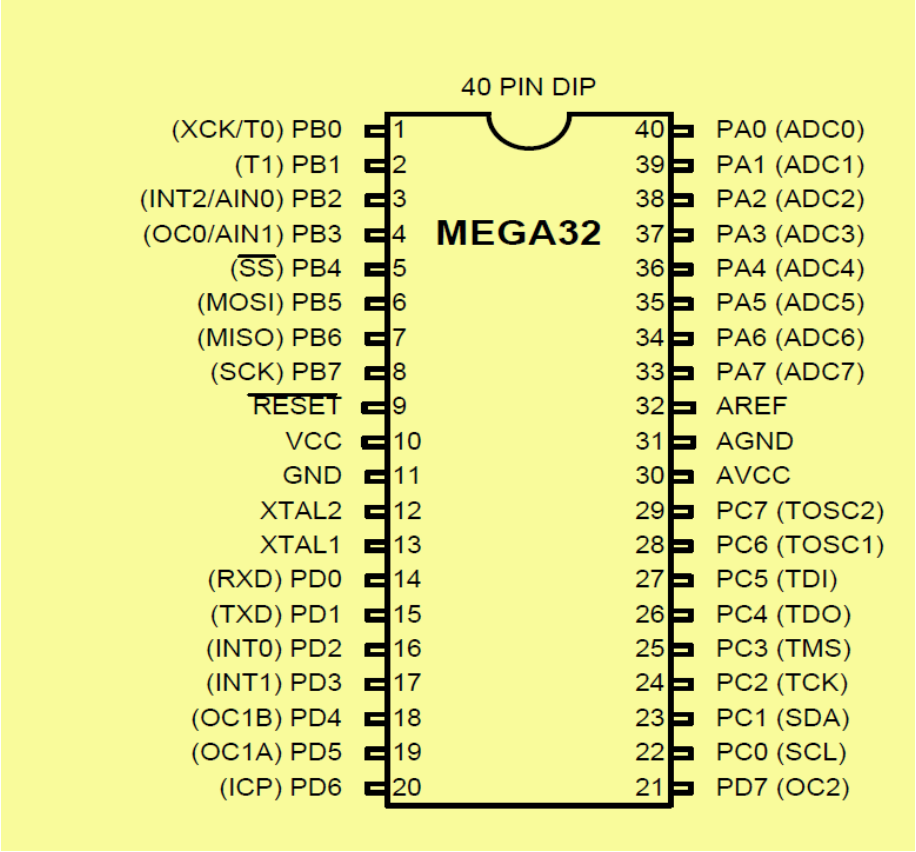


Figur 8 - IBD for X.10 Enhed

Er en oversigt til X.10 enhedens indre dele og hvordan de høre sammen både internt, men også hvordan dens i/o porte er.

Signalbeskrivelse:

Navn	Signal	Type	Spænding	Tolerance
dig	Kodelås signal aktiveringssignal til mikroprocessor. Sender et logisk 1 eller 0.	Digital	5V DC	+/- 0,5V
seriel	Seriel forbindelse mellem PC og mikroprocessor	RS232	5V DC	+/- 0,5V
INT0	Ekstern interrupt på PD2, når zero crossing er sket.	Digital	5V DC	+/- 0,5V
protokol	120 KHz burst genereret af carrier generatoren ved zero-crossing detektion, der overleger 50 Hz AC lysnet med et DC logisk højt signal.	Lysnet: AC/DC	5V DC/18VAC	+/- 0,5V
cd	120 KHz burst overleget lysnettet kommer ind i carrier detektoren.	Lysnet: AC/DC	5V DC/18VAC	+/- 0,5V
ZeroX		Digital	5V DC	+/- 0,5V
light	Højt signal for at tænde lampe og lavt signal for at slukke lampe.	Digital	5V DC	+/- 0,5V



PORTE:

PD2 – ZeroCrossing Interrupt INT0 (Falling edge)

PD1 – TXD Seriel sender bit

PD0 – RXD Seriel modtager bit

PD5 – – Timer1 (OCR1A), 120 KHz burst udgang

PBn – Hvis Zerocrossing, alle bits høj, ellers 0.

PB0 – Timer0, ca. 1 ms delay.

PD3 – DE2 Kodelås indgang, højt eller lavt, INT1

PC3 – 120kHz Input

PC4 – Input fra DE2 board(GPIO_0[0])

PC1 – Signal Output fra modtager – aktiv høj lysstyring

Beskrivelse af funktionaliteten af de enkelte blokke.

Zero-Crossing

I vores X.10 controller bliver beskeder og information tilpasset efter zero-crossing.
En Zero-crossing Detektor bliver brugt til at detektere hvornår et AC signal går gennem 0.
De 0-gennemgange der bliver detekteret bliver oversat til et digitalt signal og de forskellige logiske niveauer bliver detekteret inden for CMOS områder.
De logiske niveauer vil blive aflæst i bestemte intervaller (if. X10 protokol AN236) og disse intervaller vil blive set som ”House” og ”Key” koder der aktiverer og deaktivere de forskellige X.10 enheder og fortæller bestemte kommandoer.

Carrier Generator

X.10 enheder anvender et 120kHz moduleret signal til at videre give oplysning over el nettet med 50Hz.
Det er muligt at generer 120kHz moduleret signal med et eksternt oscillator kredsløb eller et STK 500 kit.
Det er vigtigt at generer et præcis 120kHz frekvens og til denne opgave vil det være let at bruge STK 500kit til at lever 120kHz firkantsignal. Der findes mange måder at designe et filter, som tillader 120kHz igennem og dæmper 50Hz. Det valgte filter er et passivt højpas filter.

Carrier Detektor

Målet med den 120 kHz Carrier Detektor er at tjekke om der er en carrier tilstede eller ikke. Dette sker ved at, for hver zero-crossing interrupt bliver PD2 tjekket indenfor en 1ms transmission kurve om der er en Carrier tilstede. X-10 senderen sender så en besked med et ’1’ hvis der er en Carrier eller et ’0’ hvis der ingen Carrier er. For at Carrier detektoren kan modtage X-10 signaler er det nødvendigt at opfange tilstedeværelsen af det 120kHz signal som er i AC strømnettet.

Software Arkitektur

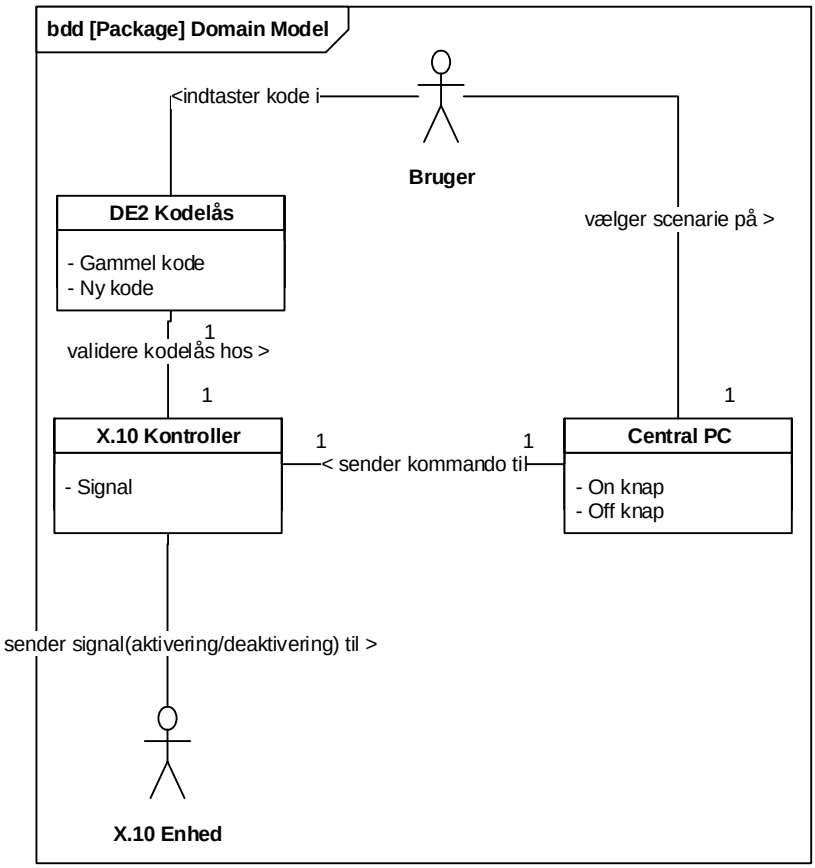
Indledning

Følgende indeholder domain analyser samt applikationsmodeller over use cases beskrevet i Kravspecifikationsdokumentet. Først identificeres de konceptuelle klasser ud fra valgte use cases, ved at analysere navneordene. Dernæst beskrives de konceptuelle klasser ved brug af en kategori liste, hvor kategorier bestemmes for relevante domain objekter. Og dernæst skrives domain modeller og applikationsmodeller op for hver use case.

Identifikation af konceptuelle klasser ud fra use cases.

Navneord	Specifikation
Bruger	Aktør
Gammel kode	Attribut
Ny kode	Attribut
DE2 Kodelås	Klasse
Signal	Attribut
X.10 kontroller	Klasse
Konfigurer system scenarie	Klasse
Central PC	Klasse
Bruger-Interface program	Klasse
ON/OFF knap	Attribut

System Domain Model for Home Security System



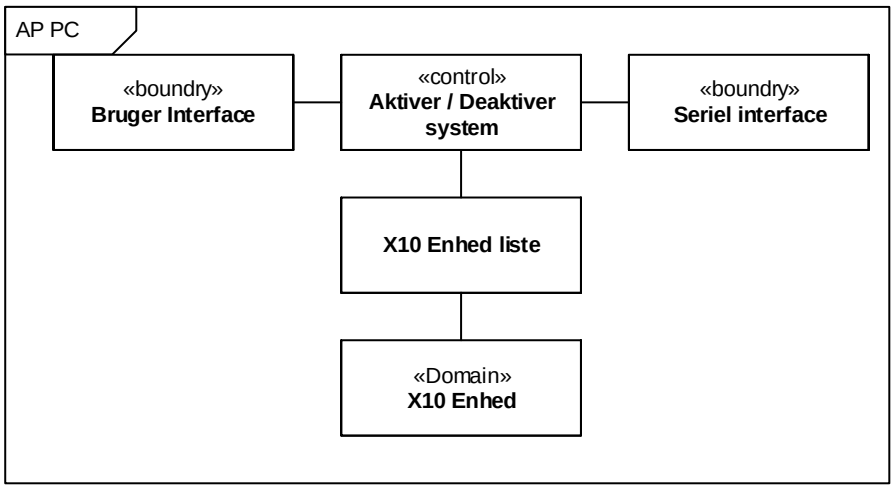
Figur 9 - Domain Model for hele system.

Beskrivelse:

Her vises domain model for hele system. De er sat sammen for simplificering, da de sender kommandoer fra samme sted og ved samme kode indtastning.

Bruger indtaster kode *i DE2 Kodelås*. Hernæst vælger Bruger et scenarie på *Central PC*, som sender kommando videre til *X.10 kontroller*. Afhængig af hvad Bruger har valgt, aktiveres eller deaktiveres systemet og dermed *X.10 Enhed*.

System Applikations model for UC1: Aktiver/Deaktiver system



Figur 10 - Applikations model for PC

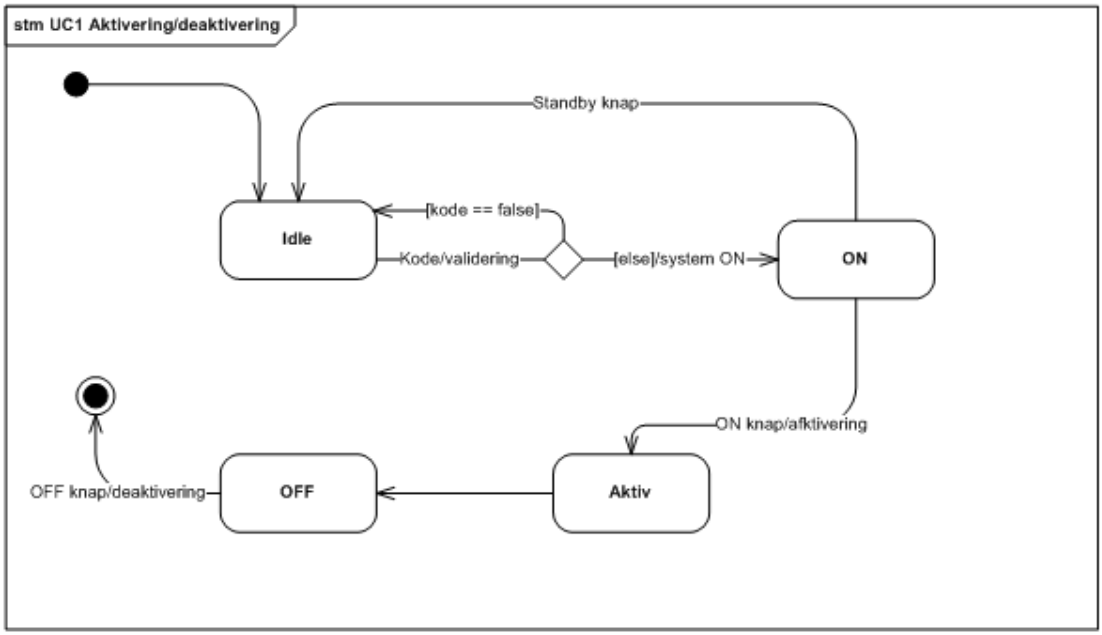
Beskrivelse:

Bruger-Interface i denne use case giver brugeren mulighed for at styre X.10 Enhederne via kommandoer og valgte scenarier. Klassen her er en grænseflade der håndterer inputs.

Seriel-Interface i denne use case er den grænseflade mellem PC og X.10 kontroller, der sørger for den serielle forbindelse. Klassen her håndterer input og output mellem PC og X.10 kontroller.

X.10 Enhed i denne use case er et domain objekt, som primært håndterer informationer som bruger har givet til systemet, f.eks. at en lyslampe skal tænde og dernæst slukke

Tilstandsdiagram for UC1: Aktiver/Deaktiver system

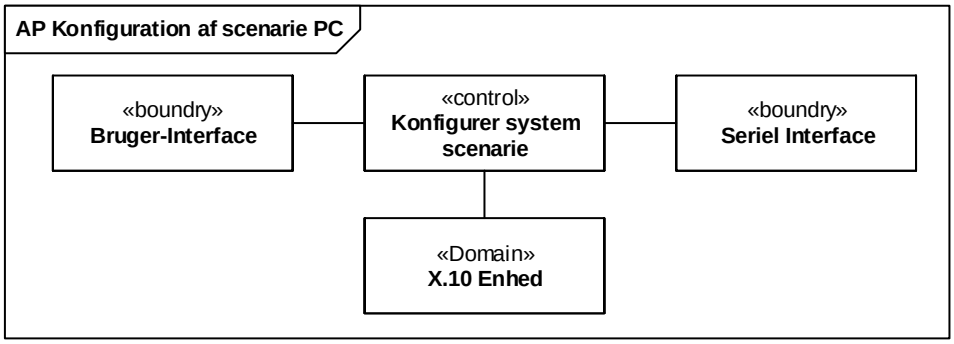


Figur 11 - Tilstandsdiagram for Use Case 1: Aktiver/Deaktiver system

Beskrivelse:

Systemet, som er Home Security System, står i standby også kaldt ”Idle” tilstand. Herefter sker der en indtastning af kode i systemet der valideres. Hvis koden er falsk vendes tilbage til ”Idle” tilstand. Hvis koden er korrekt indtastet, er systemet klar til at modtage kommandoer og befinder sig i en tændt tilstand, kaldt ”ON”. Herefter venter systemet på at en ON knap trykkes og er dette tilfælde, befinder sig systemet i en aktiv tilstand, som foretager en eller anden form for kommando, brugeren har valgt. Hvis der trykkes på OFF knap, deaktiveres systemet.

System Applikations model for UC2: Konfigurer system scenarie

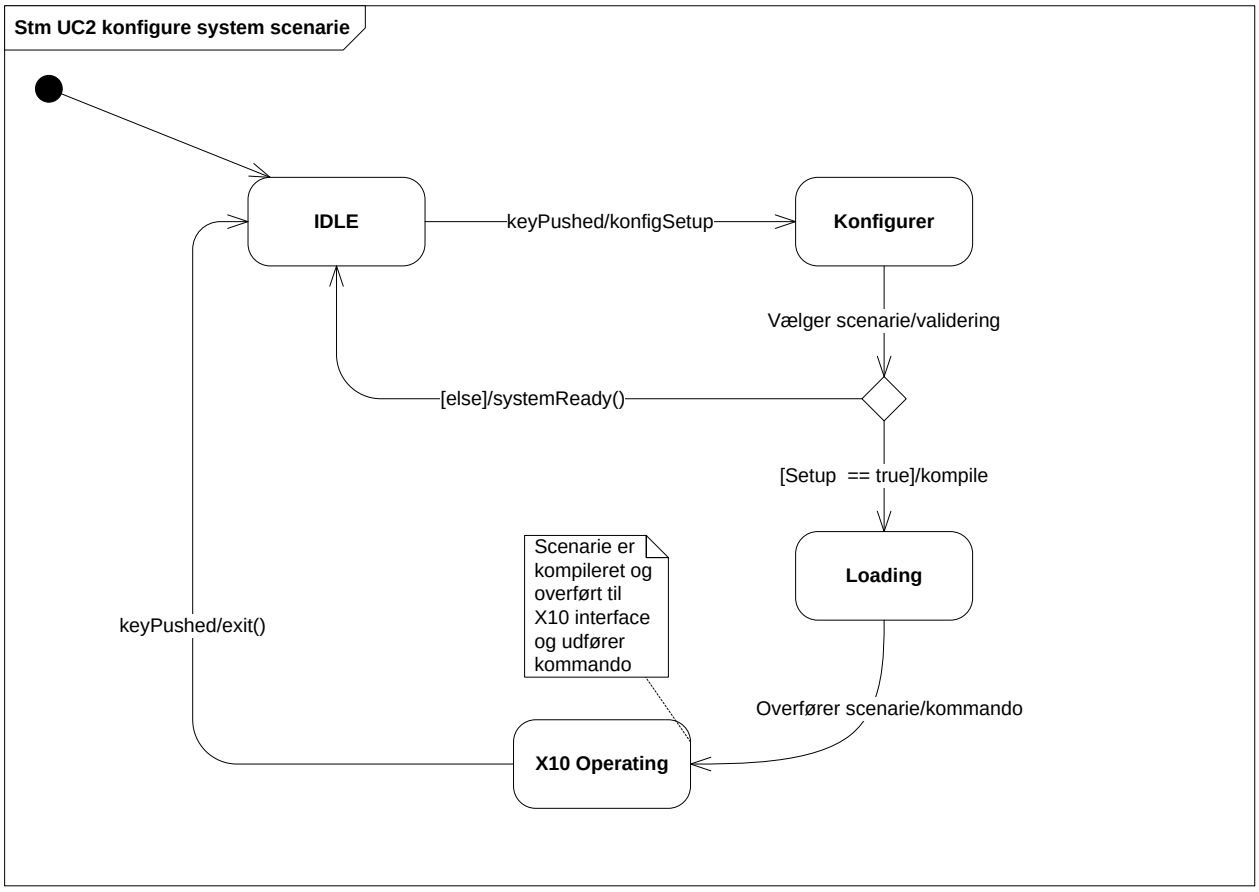


Figur 12 - AP for UC2

Beskrivelse:

Denne applikationsmodel tager udgangspunkt i *UC2: Konfigurer system scenarie* og her er der valgt de samme grænseflader og domain klasse, da det er kommandoer fra samme bruger-interface, der igangsætter systemet.

Tilstandsdiagram for UC2: Konfigure system scenarie

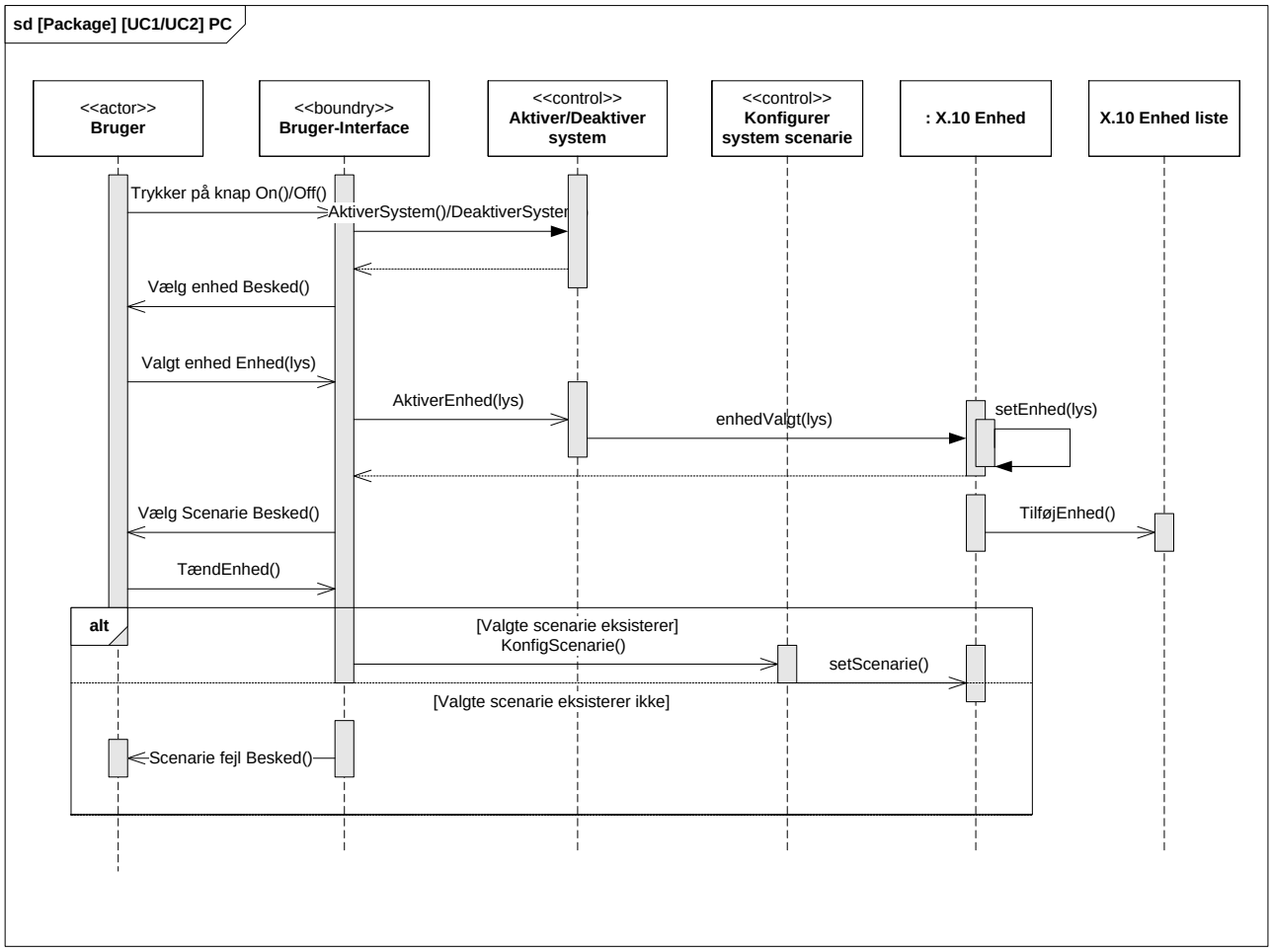


Figur 13 - Tilstandsdiagram for Use Case 2: Konfigure system scenarie

Beskrivelse:

Systemet starter i en standby tilstand ”IDLE”. Hvis der trykkes på en knap, i dette tilfælde hvis bruger f.eks. trykker på menu, vil systemet gå i en konfigurations tilstand, hvor bruger vælger scenarie samt en validering sker. Hvis det valgte scenarie, dvs. bruger trykker andet end bruger får besked på, vendes systemet i ”IDLE” tilstand igen. Hvis det valgte scenarie er korrekt indtastet, går systemet i en ”indlæsende” tilstand, hvor koden i X.10Sender driveren påbegyndes og konfigureres efter brugers valg. Valget overføres og modtages af en X.10 enhed, bruger har valgt, hvor X.10 enhed herefter tændes eller slukkes, og befinder sig i den såkaldte ”X10 Operating” tilstand. Trykker bruger på exit knap, vendes systemet tilbage til ”IDLE” tilstand.

Sekvensdiagram for UC1 og UC2

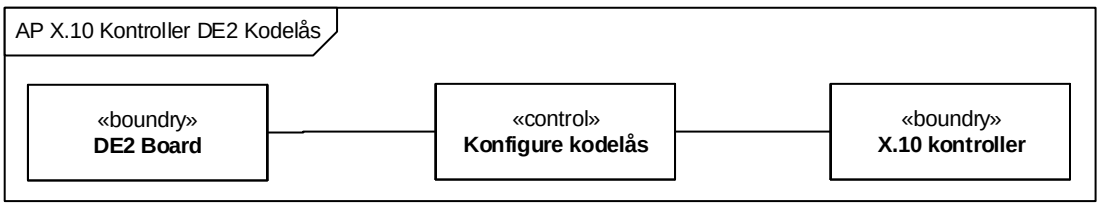


Figur 14 - Sekvensdiagram for UC1 og UC2

Sekvensbeskrivelse:

Sekvensdiagrammet tager udgangspunkt i to sammenlagte use cases, use case 1 og 2 da bruger-interface håndterer de samme kommandoer. At initialisere/aktivere eller deaktiver en enhed samt konfigurer et system scenarie.

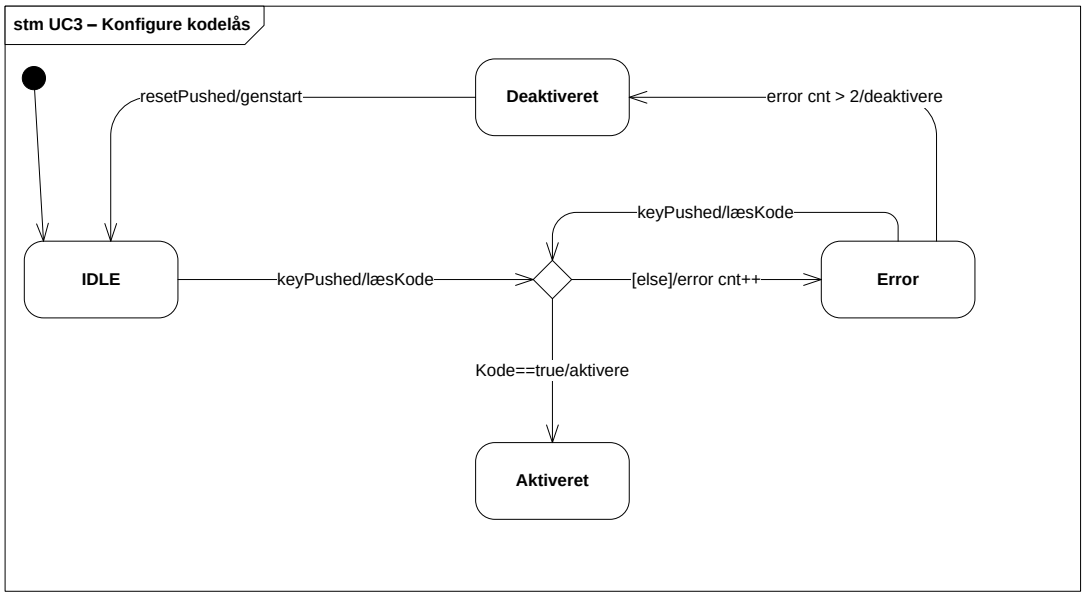
System Applikations model for UC3: Konfigurer kodelås



Figur 15 - AP for UC3

Beskrivelse:
DE2 Board i denne use case er klassen, som håndterer indtastning af koden og videre sendelse af signalet til X.10 kontrolleren for validering. *X.10 kontroller* i denne use case sørger for at låse op for systemet via STK-500 og altså afgøre om at systemet kan aktiveres eller deaktiveres. Denne håndterer inputs men også outputs.

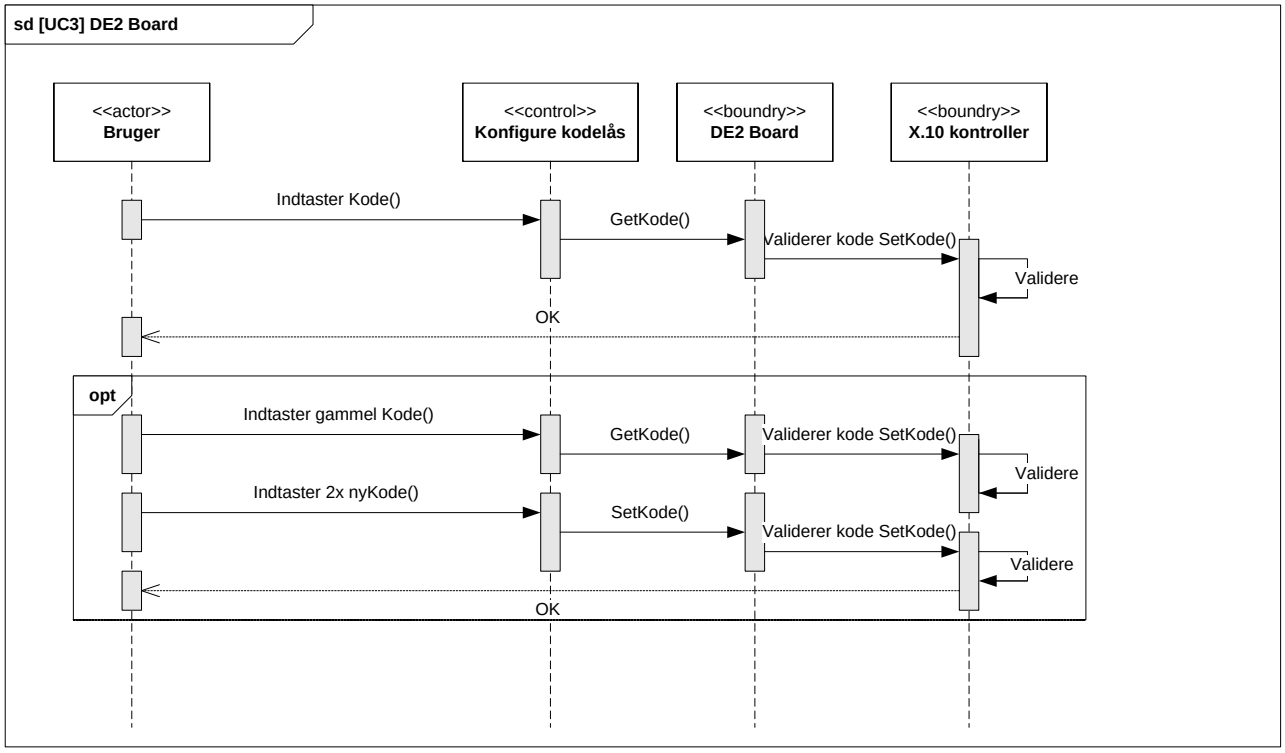
Tilstandsdiagram for UC3: Konfigure kodelås



Beskrivelse:

I dette tilstandsdiagram beskrives de enkelte tilstande, DE2 Kodelåsen på DE2 Boardet befinder sig i. Der er valgt en ”IDLE” tilstand, som systemet starter i. Hvis der trykkes på en knap, læses koden og DE2 Boardet validerer koden. Hvis koden er forkert, påbegyndes en fejltæller, der tæller antallet af forkerte indtastninger. DE2 Boardet vil herefter befinde sig i en fejltilstand, indtil der indtastes igen. Er koden forkert mere end 3 gange, deaktiveres DE2 Boardet og systemet vil befinde sig i et deaktiveret tilstand, indtil der trykkes på reset. Er dette tilfældet vendes der tilbage til ”IDLE” tilstand. Hvis koden er korrekt indtastet, aktiveres DE2 Boardet og vil befinde sig i en aktiveret tilstand.

Sekvensdiagram for UC3: Konfigurer kodelås

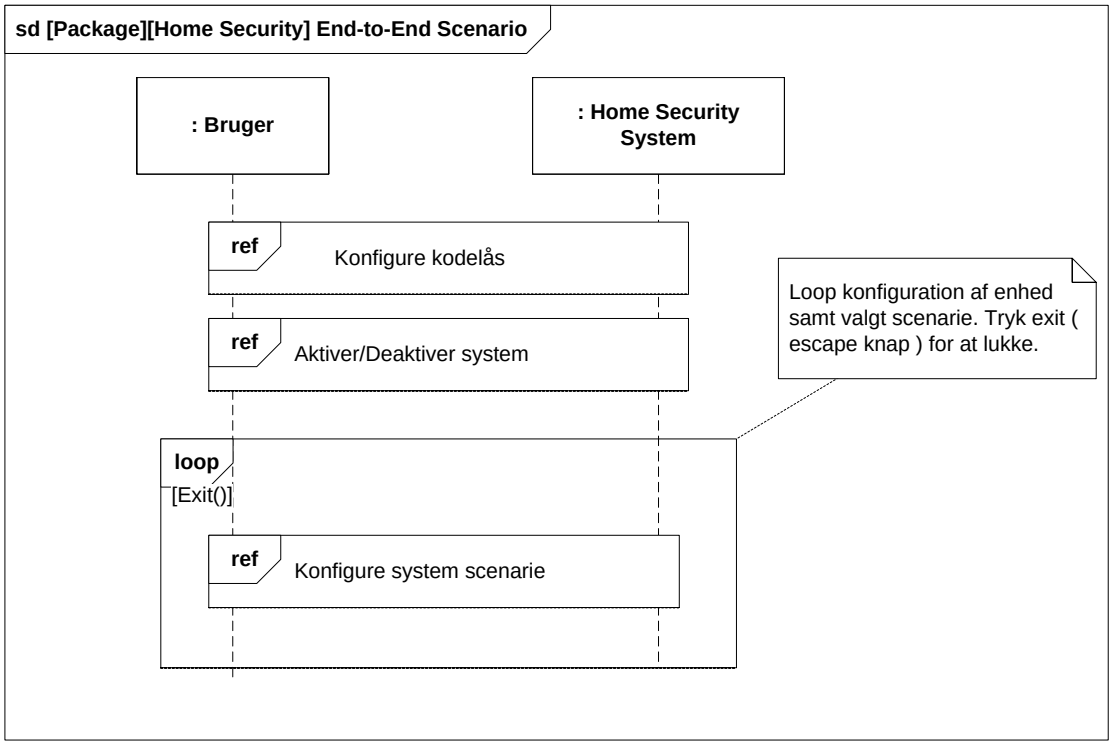


Figur 16 - SD for UC3

Beskrivelse:

Dette sekvensdiagram beskriver forløbet, hvor *Bruger* konfigurere DE2 kodelås. *Bruger* indtaster gammel kode som herefter sender signal til *DE2 Boardet* (programmeret i VHDL) om, at en kode er blevet indtastet og bliver verificeret eller falsificeret (som udgangspunkt er der i det overstående valgt en succes scenarie dvs. verificeret) af STK-500 (programmeret i c) fra *X.10 kontrolleren*. Yderligere er der også lagt fokus på ifl. use case 3, at *Bruger* har mulighed for at indtaste en ny kode, som skal indtastes to gange, for at verificere at begge indtastninger er ens. Herefter er Home Security systemet klargjort og klar til at modtage kommandoer fra *Bruger.*

Sekvensdiagram Home Security System End-to-End Scenario



Figur 17 - SD for End to End scenarie

Beskrivelse

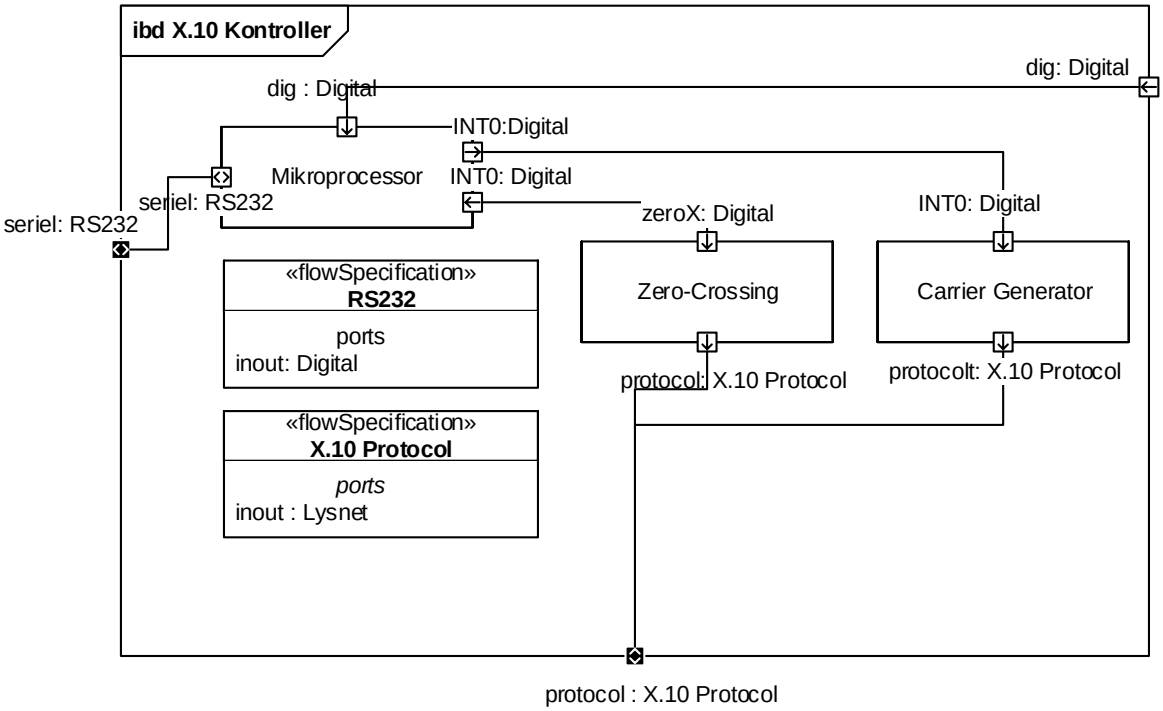
Overstående diagram viser et End-to-End Scenario forløb mellem systemets primær aktør *Bruger* og selve systemet. Forløbet er sekventielt og der refereres til de respektive sekvensdiagrammer. *Bruger* konfigurer kodelås og aktiverer eller deaktivere system. *Bruger* konfigurer enheder og scenarier, så længe *Bruger* ikke trykker på en exit knap, evt. escape knap.

Detaljerede HW design

Sender

Initialer: YS og JBM

Senderen skal transmittere de signaler som STK500 sender, og videreføre dem til el nettet. Senderen har også den funktion at den skal dæmpe 50 Hz signalet fra el nettet og tillade 120 kHz signal komme igennem derfor skal der bruges et filter. I dette tilfældes vælges et passivt højpas filter.



Figur 18 - ibd over X.10 kontroller

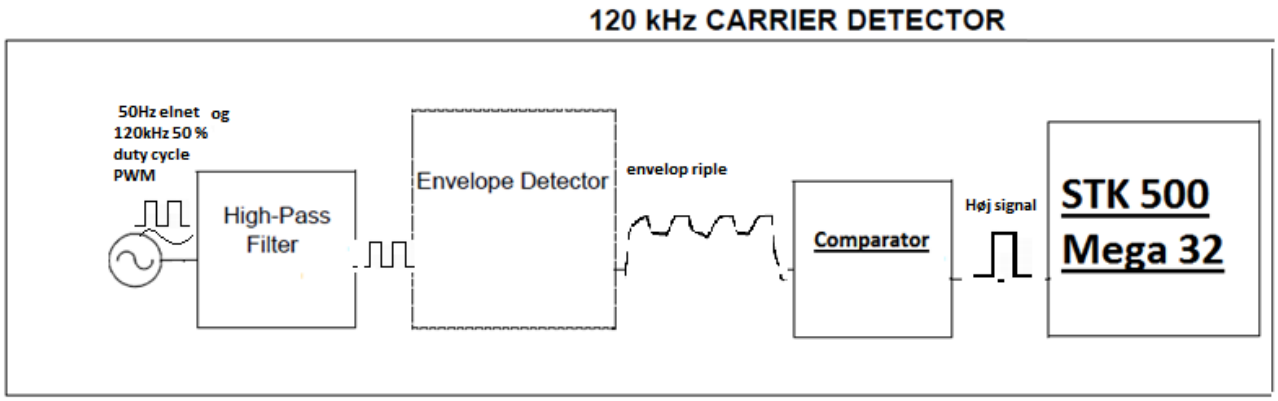
Krav til at sender

Krav til at senderen kan virke er at den skal modtage 5V signal med 120kHz(+/-3) frekvens og 50 % duty cycle på 430 Ω modstanden som hedder R2 i figur 2.

Modtager

Initialer: YS og JBM

Modtageren skal kunne modtage 120kHz signal og filtrerer el-nettets 50Hz signal fra. Ved tilstedeværelse af et 120kHz PWM signal(burst) skal modtagerens output til STK-500 være højt (5V) og hvis der ikke forekommer noget 120kHz signal, så skal outputtet til STK-500 gå lavt (0V). Til dette system benytter vi os af et højpasfilter, en ”envelop detektor” og en ”comparator”, som vist på billedet:

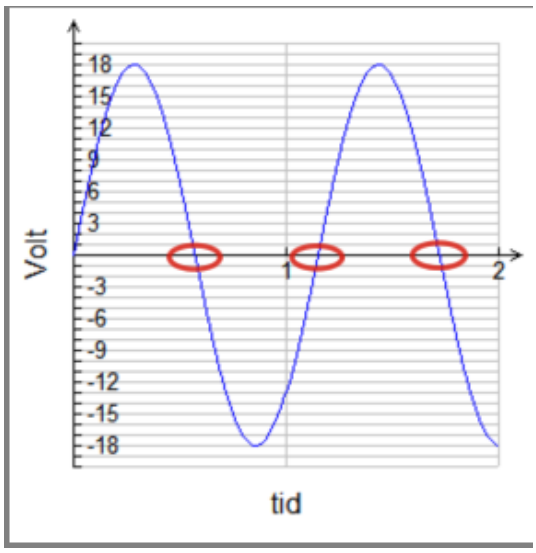


På billedet kan vi se hvordan modtageren behandler de signaler vi får fra el-nettet. Efter højpas filteret har vi kun 120kHz signal tilbage da 50Hz er filtreret fra og efter 120kHz signal har været igennem Envelope Detectoren fås en samlet puls dog med ”ripple” og Comparatoren sørger for at lave det om til et rent digitalt signal.

Zero-Crossing Detector

Initialer: JBM

Den generelle ide med ZC-detektoren er, at detektere de ”nul-gennemgange”, der opstår på det lokale net (18 VAC 50 Hz) . En Detektion foregår således, at ved bestemte spændinger eller ”thresholds” omkring ”nul-gennemgangen”, ved ”rising-edge”, vil ZC-detektoren udsende et logisk ’1’ eller højt signal (5 volt) til micro-processoren(ATmega32) og modsat ved ”falling-edge” vil der blive sendt logisk ’0’ (0 volt).



Figur 19 - Periodediagram 18sin(314t).

Her kan ses de steder hvor ZC-Detektoren skal detektere nul-gennemgangene (røde cirkler) grafen tegnet som et sinus-signal med 50 Hz og en amplitude på 18 V(i henhold til det vi får fra trafo.).

Hver gang der bliver detekteret en nul-gennemgang skal der komme et interrupt på ”falling-edge” når signalet fra detektoren skifter fra 1 til 0 V. Hvis et interrupt forekommer lidt før en nul-gennemgang (rød cirkel) på ”falling-edge” betyder det umiddelbart ikke noget, så længe det sker på samme tidspunkt hver gang.

Krav til Zero-Crossing Detektor

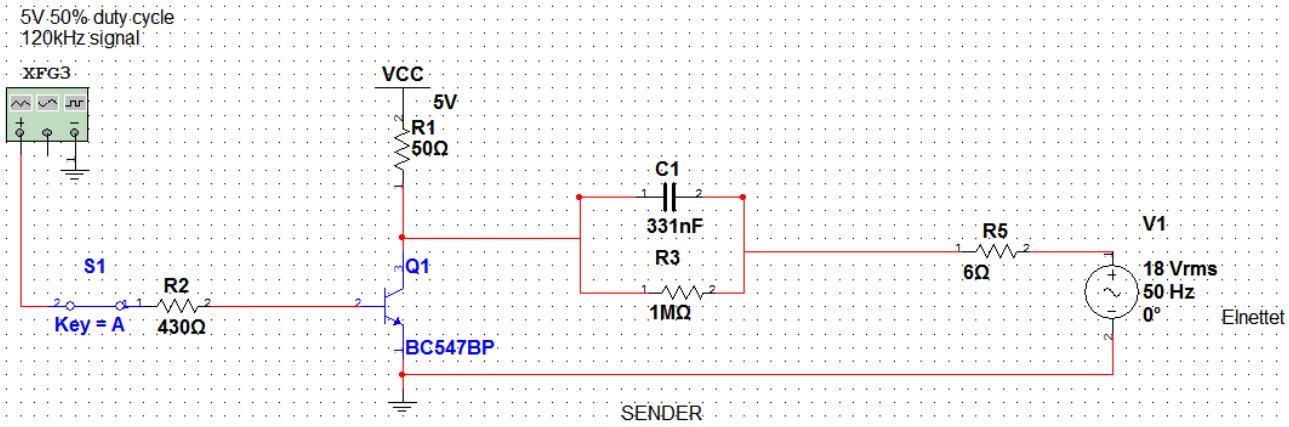
- På ”rising-edge”, på elnettets 50Hz-signal, ved et spændingsniveau på omkring ’3,5 V’ vil ”PC2” på STK-500 gå højt (5 V) idet at 3,5 til 5 V ses som logisk ’1’ på mega32 i kraft af de CMOS-gates der sidder i mikroprocessoren. Signalet fra detektoren må gerne være forsinket i forhold til el nettets signal.
- På ”falling-edge” ved et spændingsniveau på omkring ’1,5 V’ vil der blive sat logisk ’0’ på STK-500 ”PC2” (0 V) idet at CMOS har et område til logisk ’0’ på 0 til 1,5 V. Signalet fra detektoren må gerne være forsinket i forhold til el-nettets signal.

Implementering(HW)

Sender

Initialer: YS og JBM

Vores sender er tilsluttet til mikroprocessoren og el nettet med 18 V og 50Hz frekvens. I kredsløbet har vi valgt at simulere PWM signalet fra STK 500 som en funktionsgenerator. El nettets indre modstand er 6Ω, denne skal være med i beregninger da den har betydning for hvor stor kapacitoren skal være i vores højpasfilter som dæmper det 50Hz signal som kommer fra el nettet.

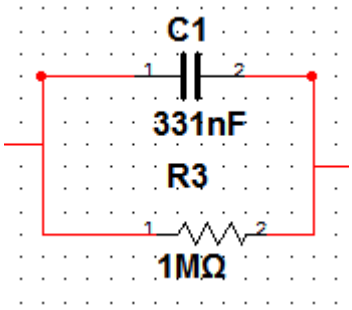


Figur 20 - Sender kredsløb

Vores design med transistor gør at vi beskytter STK 500 dvs. vi undgår at belaste den mere end nødvendigt.

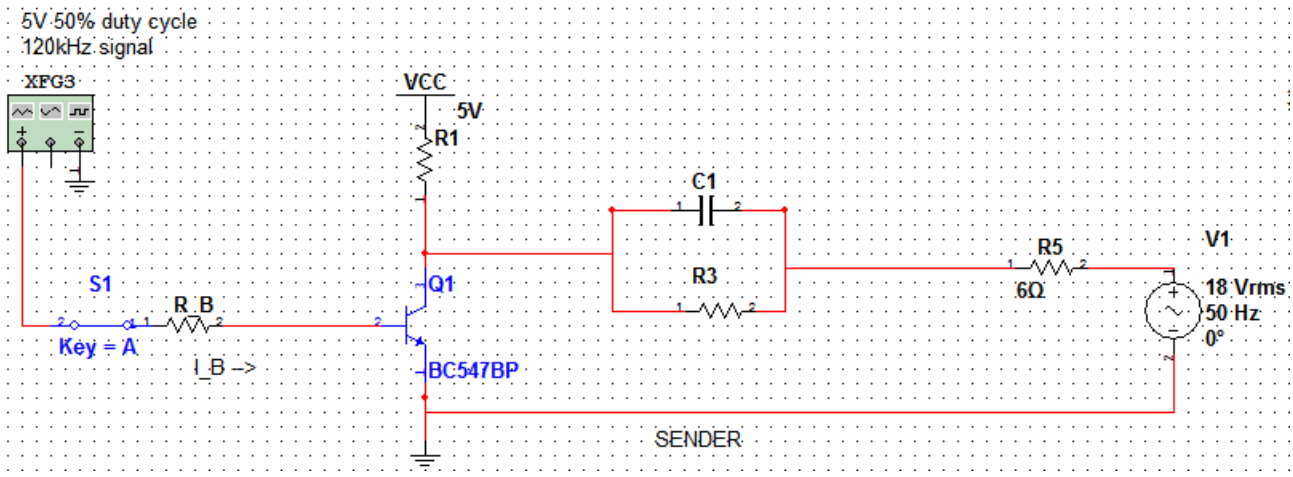
Højpasfilter

Højpasfilteret består af en kapacitor og en modstand parallel, modstanden skal være rigeligt stort til at aflade kapacitoren hurtigt



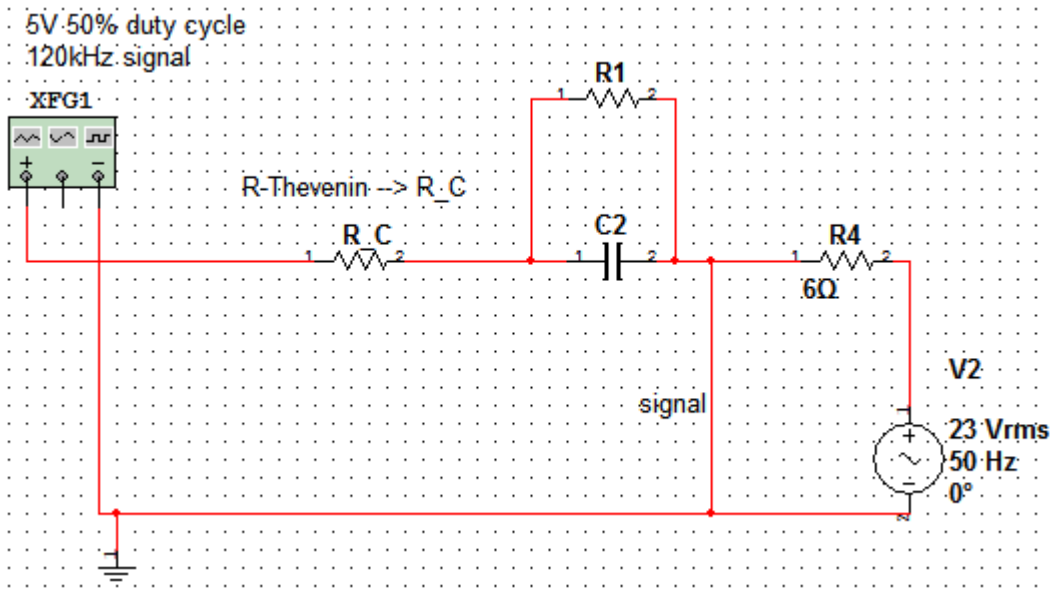
Figur 21 - Højpasfilter

Kapacitorens værdi har vi bestemt ved hjælp af superposition.



Figur 22 - Hele kredsløbet

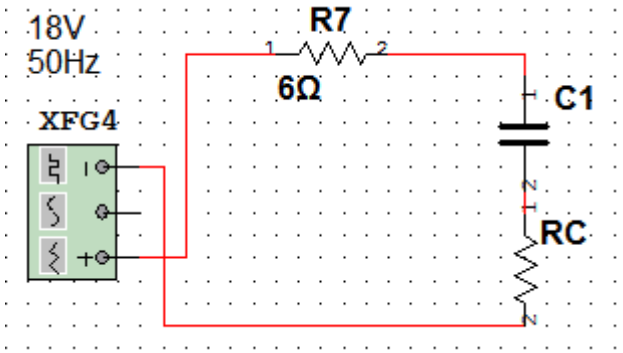
Da vi har et ikke lineært kredsløb pga. transistoren som slår til og fra. Hvis transistoren er slået til (on) bliver R1 trukket til jord og får værdien 0 Ω. Vi laver vores 120kHz signal ved hjælp af vores transistor, som slår til og fra. Vi har valgt at bruge superposition til at bestemme komponent værdierne. Inden vi kan det, skal vi lave kredsløbet om ved hjælp af Thevenin, hvilket resulterer i kredsløbet herunder:



Figur 23 - Thevenin kredsløb

Nu anvendes superposition, hvor hver kilde skiftevis sættes til 0 eller ”slukkes”.

Her sættes kilden med 120 kHz PWM til ”0”:



Figur 21 - Superposition for den ene kilde

Udregninger for 50 Hz kilde:

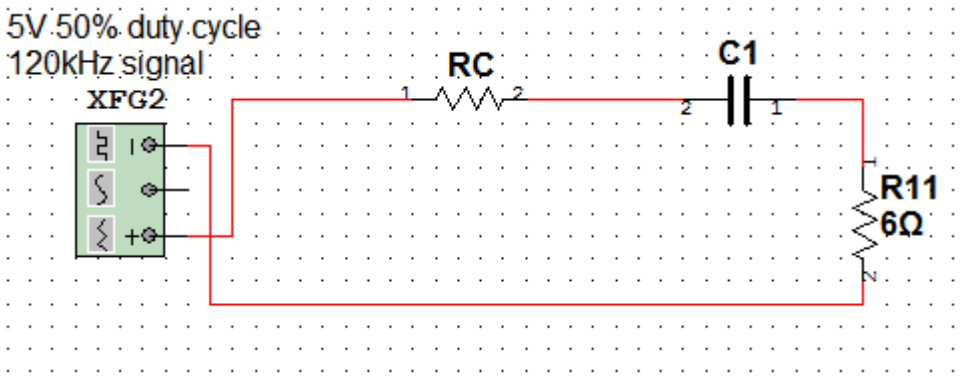
$$Tv(s) = \frac{R_c}{6[\![\Omega]\!] + \frac{1}{s \cdot c_1} + R_c}$$

$$= \frac{R_c \cdot s \cdot c_1}{6[\![\Omega]\!] \cdot s \cdot c_1 \cdot R_c \cdot s \cdot c_1 + 1}$$

$$= \frac{s \cdot \frac{R_c}{R_c + 6[\![\Omega]\!]}}{s + \frac{1}{\left(R_c + 6[\![\Omega]\!]\right) \cdot c_1}}$$

$$= s \cdot \frac{R_c}{R_c + 6[\![\Omega]\!]} \cdot \frac{\frac{1}{\left(R_c + 6[\![\Omega]\!]\right) \cdot c_1}}{s + \frac{1}{\left(R_c + 6[\![\Omega]\!]\right) \cdot c_1}}$$

Her efter sættes El nettets bidrag fra trafo. til ”0”.



Figur 24 - Superposition PWM signal tændt

Udregninger for 120 kHz kilde:

$$Tv(s) = \frac{6[\![\Omega]\!]}{R_c + \frac{1}{s \cdot c_1} + 6[\![\Omega]\!]}$$

$$= \frac{6[\![\Omega]\!] \cdot s \cdot c_1}{s \cdot c_1 \cdot R_c + 6[\![\Omega]\!] \cdot s \cdot c_1 + 1}$$

$$= \frac{s \cdot \frac{6[\![\Omega]\!]}{R_c + 6[\![\Omega]\!]}}{s + \frac{1}{\left(R_c + 6[\![\Omega]\!]\right) \cdot c_1}}$$

$$= s \cdot \frac{6[\![\Omega]\!]}{R_c + 6[\![\Omega]\!]} \cdot \left(R_c + 6[\![\Omega]\!]\right) \cdot c_1 \cdot \frac{\frac{1}{\left(R_c + 6[\![\Omega]\!]\right) \cdot c_1}}{s + \frac{1}{\left(R_c + 6[\![\Omega]\!]\right) \cdot c_1}}$$

Nu har vi fundet det to standard led for begge situationer.

$\frac{\frac{1}{\left(R_c + 6[\![\Omega]\!]\right) \cdot c_1}}{s + \frac{1}{\left(R_c + 6[\![\Omega]\!]\right) \cdot c_1}}$ Dette led er fælles for begge situationer og derved hvis vi finder c_1 på den ene gælder den også for den anden

Vi sætter R_c = 50Ω for at beskytte transistoren selvom vi ønsker forstærkning på 1 dvs.

$$\frac{6[\![\Omega]\!]}{R_c + 6[\![\Omega]\!]} = 1$$

Hvis modstanden er for stor kommer vi længere væk fra vores ønskede værdi derfor har vi valgt en mellemting som 50 Ω er for vores kredsløb.

Vi kender nu R_c modstanden og knæk frekvensen sætter vi til 80.000 og dermed udregner kapacitorens størrelse

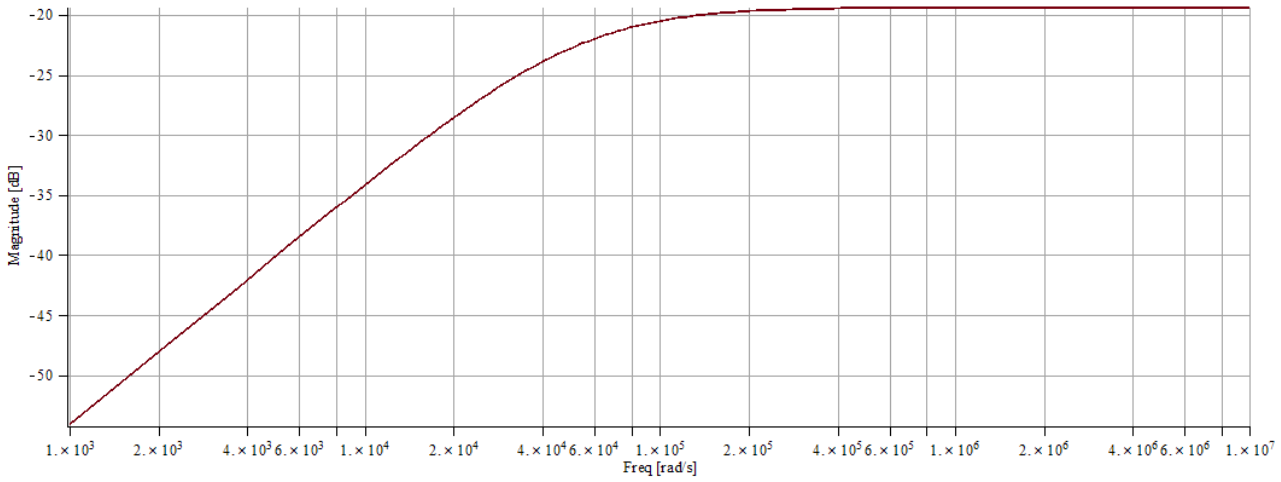
$evalf\left(solve\left(80000 = \frac{1}{2 \cdot \pi \cdot \left(50[\![\Omega]\!] + 6[\![\Omega]\!]\right) \cdot c_1}, c_1\right)\right) = 3.552565694\,10^{-8} \approx 35\text{nF}$ og da vi ikke kunne finde denne størrelse i lab. tog vi 33nF i stedet for. Nu bliver vi nød til at udregne det nye knæk frekvens

$$evalf\left(fcutoff = \frac{1}{2 \cdot \pi \cdot \left(50[\![\Omega]\!] + 6[\![\Omega]\!]\right) \cdot 33 \cdot 10^{-9}}\right) = 86122.80469$$

Vi forsøger nu at tegne frekvens karakteristikken for nedstående overføringsfunktion

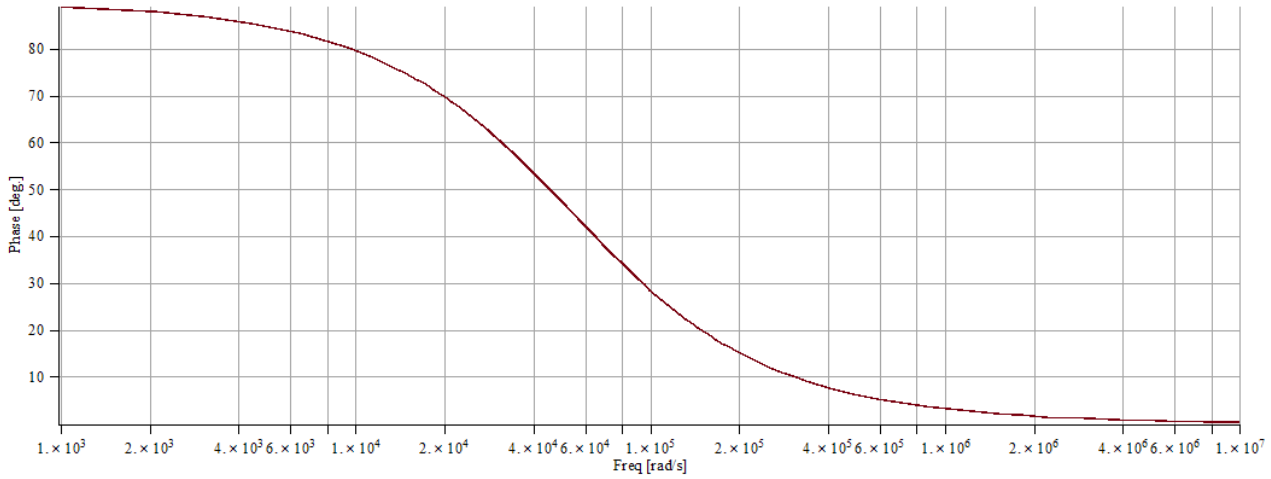
$$s \cdot \frac{6}{50+6} \cdot (50+6) \cdot 330 \cdot 10^{-9} \cdot \frac{\frac{1}{(50+6) \cdot 330 \cdot 10^{-9}}}{s + \frac{1}{(50+6) \cdot 330 \cdot 10^{-9}}}$$

dB plot



Graf 1- dB plot

Phase plot

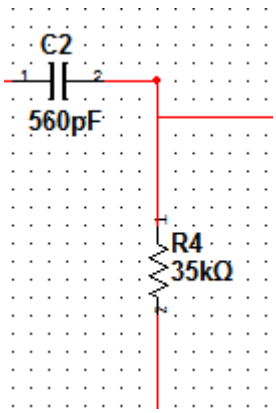


Graf 2 - Fase plot

Modtager

Initialer: YS og JBM

Der er valgt et højpasfilter til at dæmpe det 18V 50Hz signal fra el nettet. Vores højpasfilter består af en 560pf kapacitor og en 35kΩ modstand



Figur 25 - højpasfilter

Der sættes en knæk frekvens på 80000Hz

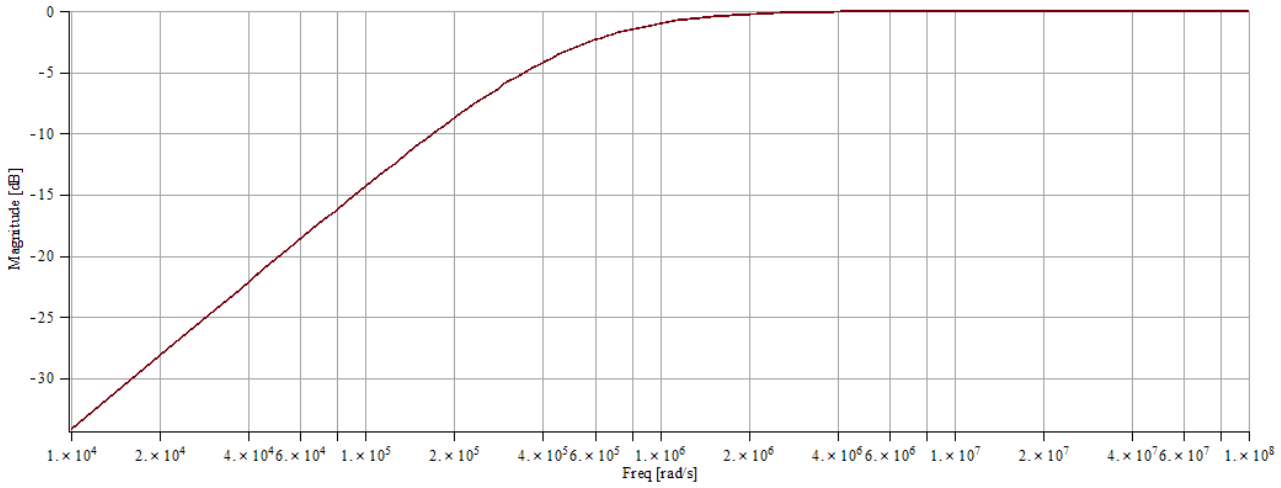
vores knæk frekvens i vinkel

$evalf(80000 \cdot 2 \cdot \pi)$

$$5.026548246\,10^{5}$$

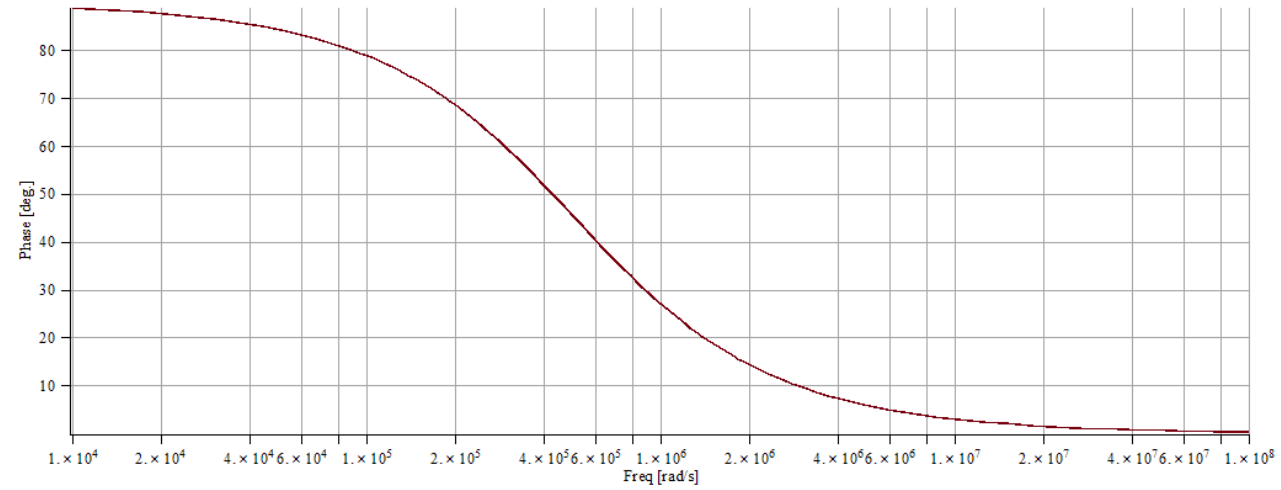
Overførings funktion= $s \cdot 3.5 \cdot 10^{3} \cdot 560 \cdot 10^{-12} \cdot \dfrac{\dfrac{1}{3.5 \cdot 10^{3} \cdot 560 \cdot 10^{-12}}}{s + \dfrac{1}{3.5 \cdot 10^{3} \cdot 560 \cdot 10^{-12}}}$

dB



Graf 3 - dB plot

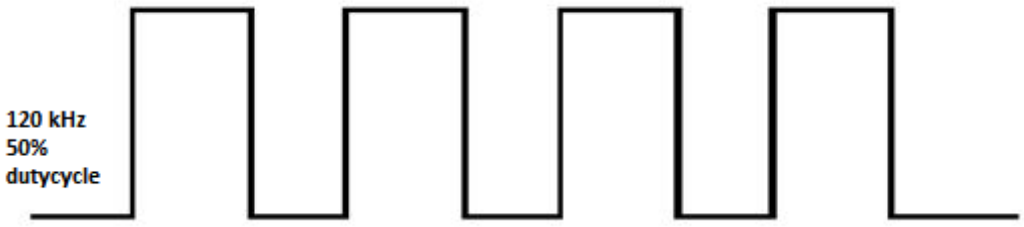
Phase



Graf 4 - Fase plot

Envelop detektor

Meningen med Envelope Detektoren er at den skal kunne modtage 120 kHz signal og lave det om til en puls der går høj.



Figur: input til envelope detektor

Et eksempel på sådan en puls ses på billedet herunder i forhold til billedet ovenover.

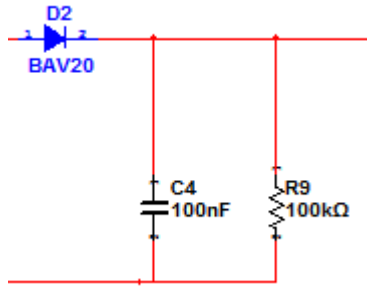
Figur: Et forklarende eksempel på en envelope output

I virkeligheden så ser outputtet på en envelop detektor ikke sådan ud men mere i retning af den figur vi har lavet her:

Her under kan man se vores envelop detektor, hvor at kapacitoren i detektoren bliver opladt, hver gang input signalet er på ”rising-edge” og afladt på ”falling-edge”. Det er denne opladning og afladning der giver ”ripple” på vores signal. Normalt vil man skalerer modstanden og kapacitoren i forhold til tidskonstanten som er:

$$\tau = RC$$

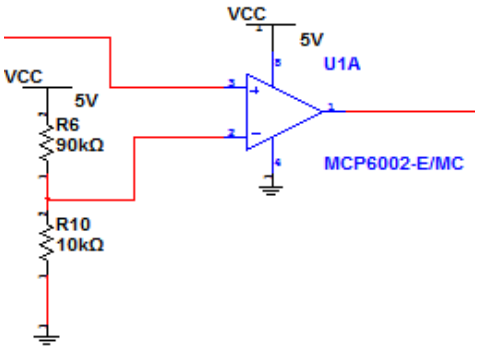
Hvis tidskonstanten er ”stor” vil det tage kapacitoren lang tid at lade op, hvilket også gør den langsommere til at aflade, dermed hvis afladningen tager lang tid vil kapacitoren ikke kunne nå at aflade særligt meget før et nyt ”peak” eller ”rising-edge” starter på indgangssignalet og igen oplader kapcitoren dvs. at en større tidskonstant giver en mindre ”ripple” men et langsommere system. Modsat en ”lille” tidskonstant som giver større ”ripple” men et hurtigere system.



Figur 26 - Envelop detektor

For at undgå det dilemma der hedder enten et langsomt system eller for store ”ripples” (forskel i spænding) benyttes der en ”Comparater” og dermed har vi valgt en nogenlunde kapacitor på 100nF og en stor modstand, for ikke at spilde for meget strøm, på 100kΩ samt en diode for at ensrette strømmen så den ikke løber tilbage til el nettet.

Comparator

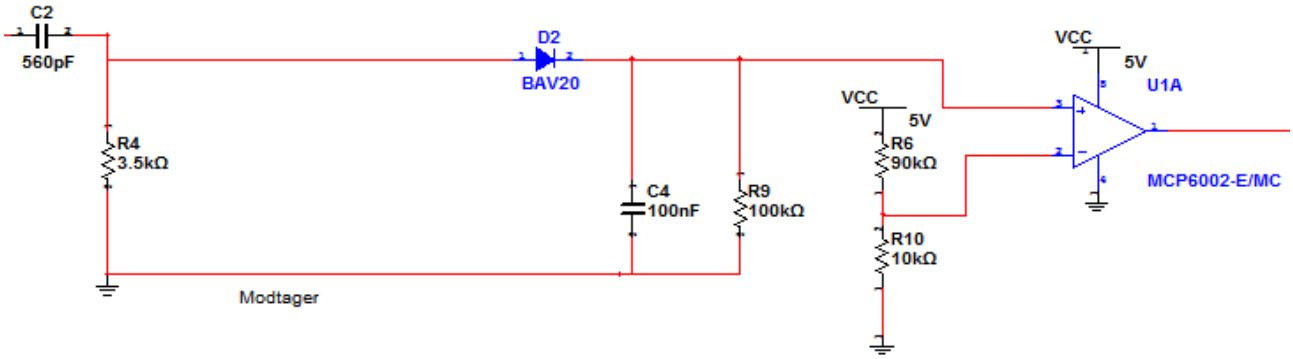


Figur 27 - Comparator

”Comparatoren” skal sammenligne outputtet fra envelope detektoren, på plus benet, med en reference spænding på minus benet og hvis spændingen på plus går over reference spændingen vil ”comparatorens” output blive lig med VCC = 5 V og hvis spændingen på plus derimod går under reference spændingen vil outputtet fra ”comparatoren” blive sat til jord = 0 V.

Alt i alt hvis et 120 kHz-burst er tilstede på el nettet vil der blive sendt et logisk '1' til stk500 og hvis 120 kHz- burst er fraværende vil der blive sendt et logisk '0' til stk500.

Figuren neden under kan vi se hele kredsløbet



Figur 28 - Hele kredsløbet

Som en del af vores Comparator har vi en reference spænding på minusbenet, som er lavet med en spændingsdeler med VCC som input og 2 modstande i serie.

Der vælges en nogenlunde størrelse modstand på 10 kΩ og modstanden efter VCC udregnes så i forhold til at vi godt vil har 0,5 V som reference spænding.

$$Vout = Vin * \frac{R2}{R1 + R2}$$

$$0.5 = 5 \cdot \frac{10000}{R_1 + 10000} => R1 = 90\ k\Omega$$

Dermed fås der som på diagrammet ovenover en modstand på 90kΩ i serie med på 10 kΩ som vil dele spændingen med omkring 4,5 V over de 90kΩ og 0,5 V over de 10kΩ.

Zero-Crossing Detektor

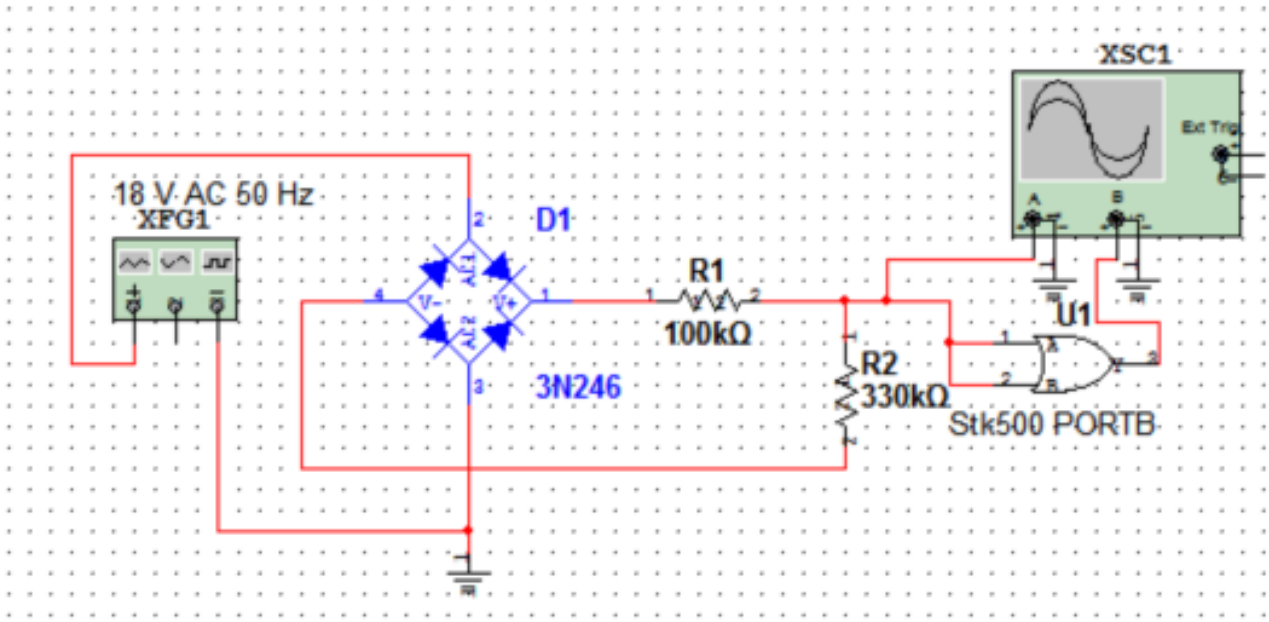
Initialer: JBM

Detektor beskrivelse:

Da stk500 skal detektere på 18 VAC 50 Hz signal fra trafo.(ideelt bliver det 18 V) sættes en relativ stor modstand imellem trafo. og stk500 for at minimere strømmen til et passende niveau som stk500 kan holde til.

Indgangen på Stk500 er CMOS og da der ikke benyttes negative spændinger på CMOS-kredse vil der blive brugt en diodebro-kobling for at ensrette spændingen til positiv retning.

Simulering Detektor (Test med multisim) og realisering.



Figur 29 - Kredsløb over zero crossing

Først vælges en relativ stor modstand på 100 kΩ så udregnes strømmen i modstanden i forhold til de 18 V da vi ideelt set har positiv spænding.

(ohms lov)

$$V = R * I$$

I isoleret

$$I = \frac{V}{R}$$

$V = 18\ V$

$R1 = 100\ k\Omega$

$$I = \frac{18\ V}{100 * 10^3\ \Omega} = 0{,}00018\ A = 180\ \mu A$$

max antal ampere stk500 kan modtage er 40 mA så 180 uA skulle være acceptabelt.

Da strømmen til "ground" skal begrænses vælges endnu en stor modstand på 330 kΩ, da vi kun har interesse for spændingen over modstanden som kommer til udtryk i den spændingsdeling som R1 og R2 skaber.

$$V_{out} = \frac{R2}{R1 + R2} * V_{in}$$

$$V_{out} = \frac{330\ k\Omega}{100\ k\Omega + 330\ k\Omega} * 18\ V = \frac{330\ k\Omega}{430\ k\Omega} * 18\ V \cong 0{,}76 * 18\ V\ \cong 13{,}81\ V$$

så cirka 3/4 af spændingen bliver påtrykket CMOS indgangen på stk500

Diodebroen er valgt efter vores spænding på 18 V og strøm på 180 microA men da de fleste diode kan holde til signaler af denne størrelse kan stort set de fleste diodebroer vælges.

Test(HW)

Sender

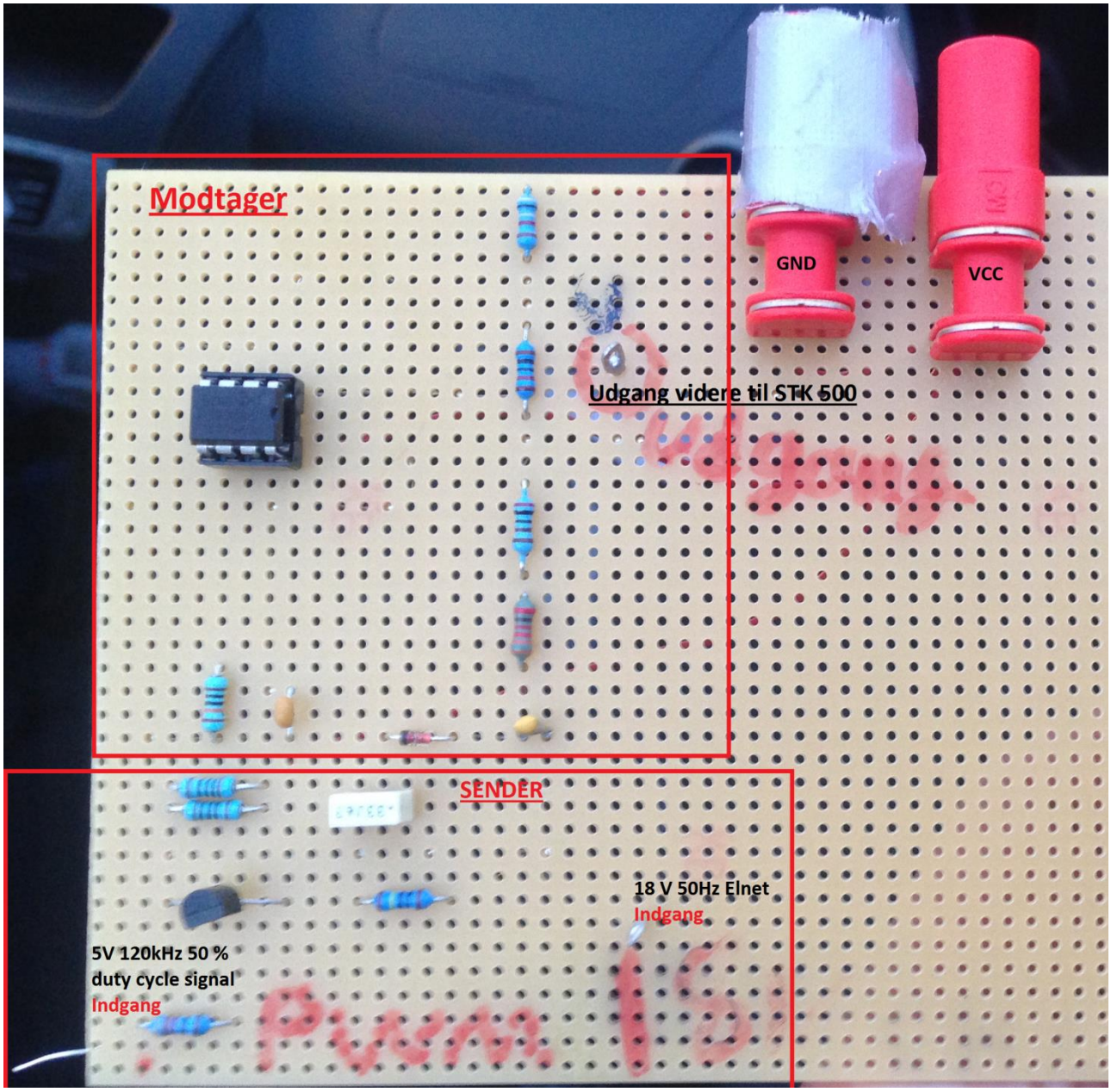
Initialer: YS og JBM

Vi kunne ikke få multisim til at simuler det godt nok og derfor har vi loddet det på print plade og testet det ved hjælp af oscilloskop og multimeter.

Neden under kan man se udgangssignal måling for senderen dvs. det signal vi sender ud på el-nettet.

Figur 30 - Realisering målinger

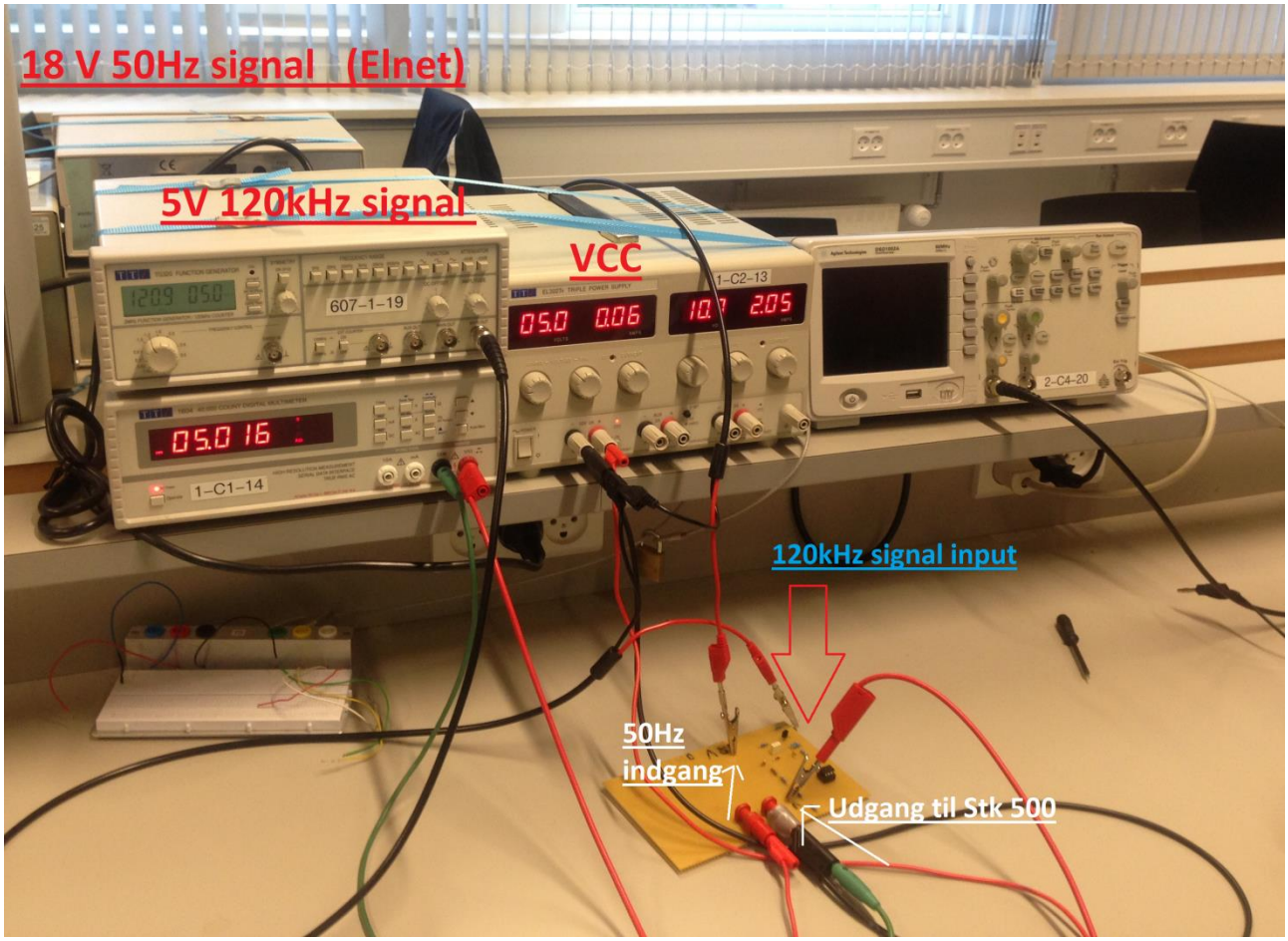
Her under kan man se vores realisering af senderen og modtager i samme printplade



Figur 31 - Billede af sender og modtager.

Modtager

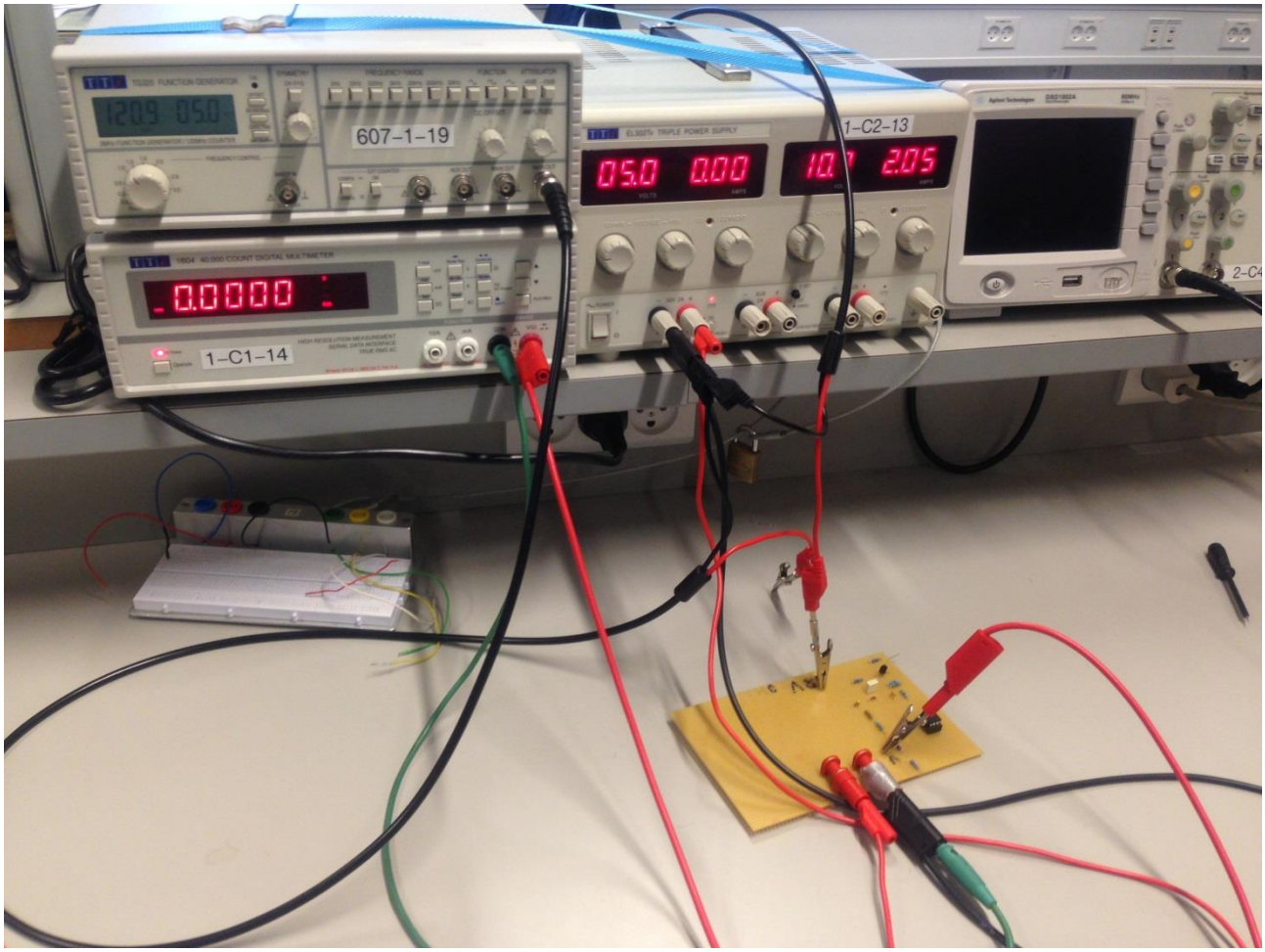
Initialer: YS og JBM



Figur 32 - Realisering af sender og modtager med 120kHz

På figuren ovenover kan vi se realisering af både sender og modtager, hvor der på multimeteret måles 5V fra outputtet fra modtageren, når 120kHz burst genereres fra senderen og 18V 50Hz sættes på indgangen af modtageren. De 5 V der måles på udgangen til stk500 vil sætte stk500 indgangen høj (5 V).

På billedet herunder har vi målt '0' når senderen ikke generere 120 kHz, hvilket må betyde at realisering virker efter hensigten.

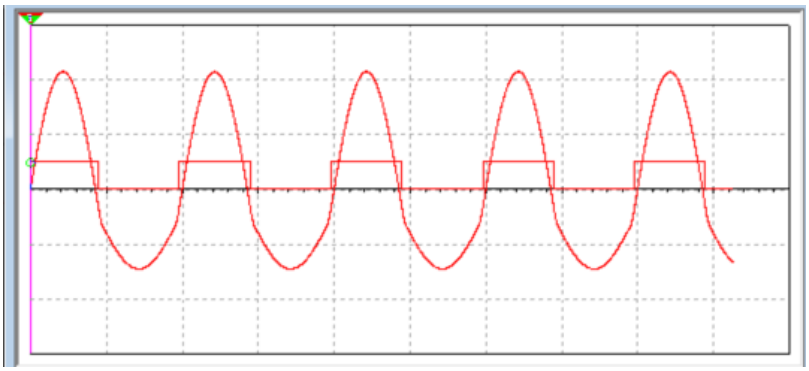


Figur 33 - realisering måling uden 120kHz fra senderen

Zero-Crossing Detektor

Initialer: JBM

(til simulering af CMOS indgangen bruges en OR-gate i multisim).



Figur 34 - Billede af udgangssignalet fra detektoren (sinus 50Hz +/- 2Hz) samt firkant signal 0 til 5V fra CMOS på stk500

Detaljeret SW design

Initialer: DB

Generelt SW design

Indledning

Dette dokument beskriver det detaljerede SW-design for projektet ”Home Security System”, som specificeret i dokumenterne kravspecifikation og systemarkitektur.

Formål

Formålet med dokumentet er:

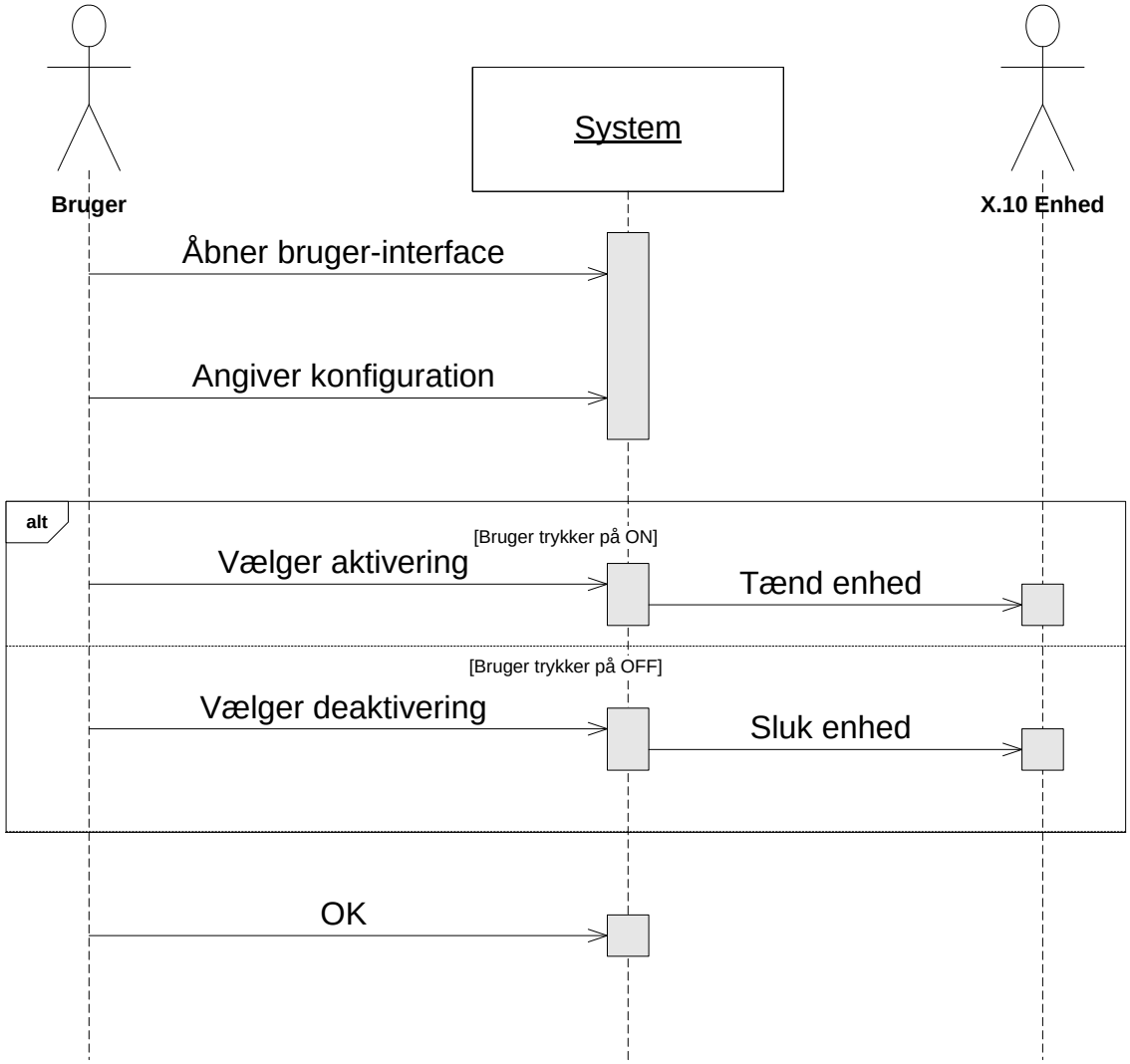
- At fastlægge systemets detaljerede softwarestruktur, drevet af kravene specificeret i kravspecifikationen og beskrivelsen af softwarekomponenterne og disses grænseflader beskrevet i systemarkitektur-dokumentet.
- At fastlægge systemets softwareklasser og disses indbyrdes interaktioner.
- At beskrive de enkelte klassers vigtigste metoder.

Refereret dokumentation

- Kravspecifikation for projektet
- Systemarkitektur-dokument

System-sekvensdiagram.

Dette lille afsnit opsummerer systemets sekvensdiagrammer fra system-arkitektur dokumentet for de enkelte Use Cases defineret i kravspecifikationen. Diagrammet her illustrerer sekvensen for brugerens interaktion med systemet for hovedscenariet og de undtagelser, der knytter sig til de enkelte Use Cases. Den interaktion, der findes på systemsekvensdiagrammet, kan genfindes på de detaljerede sekvensdiagrammer fra system-arkitektur dokumentet.



Figur 35 - Sekvensdiagram for hele systemet

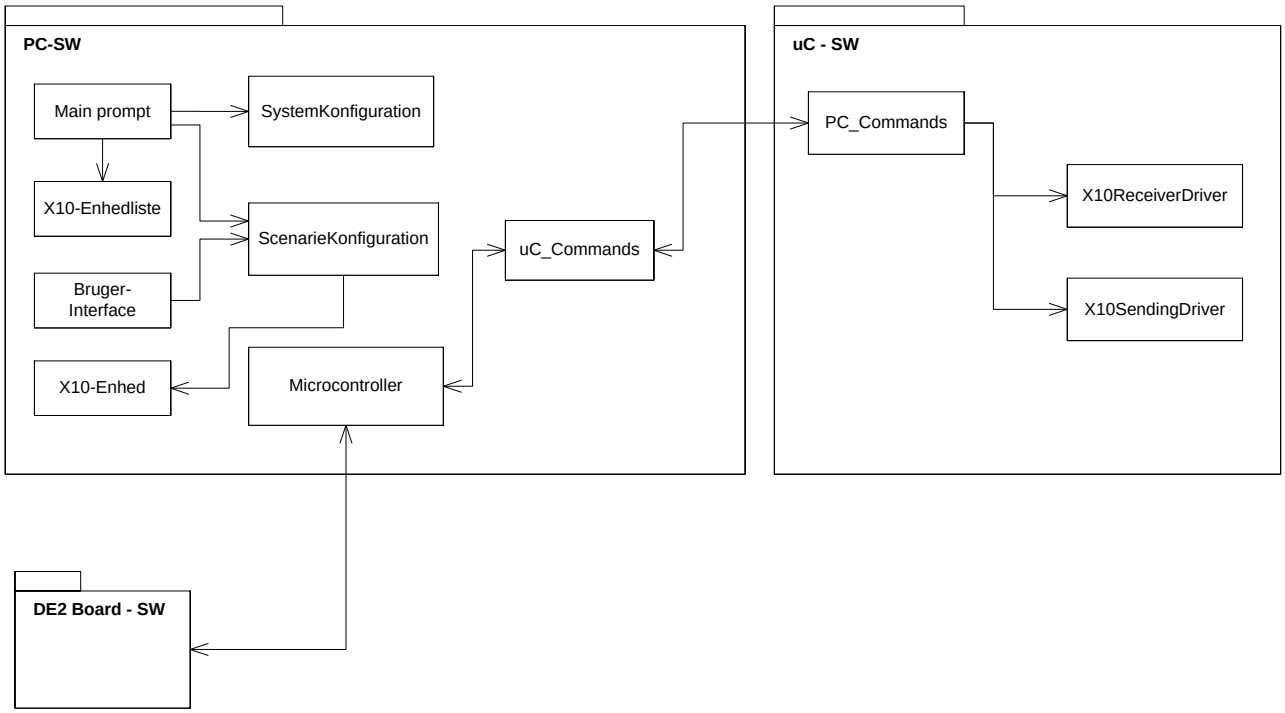
Beskrivelse:

Sekvensdiagrammet er for hele systemet og som udgangspunkt er kodelåsen konfigureret og *bruger* har direkte adgang til bruger-interfacet. Der henvises til Use Case 2 og 3 for alternative/undtagelser i systemet, da der i dette diagram er fokus på et succes scenarie.

Overordnet klassediagram

Nedenfor angives systemets resulterende klassediagram, der fremkommer som ”foreningsmængden” af de klasser, der er fundet for de enkelte Use Cases sekvens-diagrammer.

Klassernes associationer ses heraf.

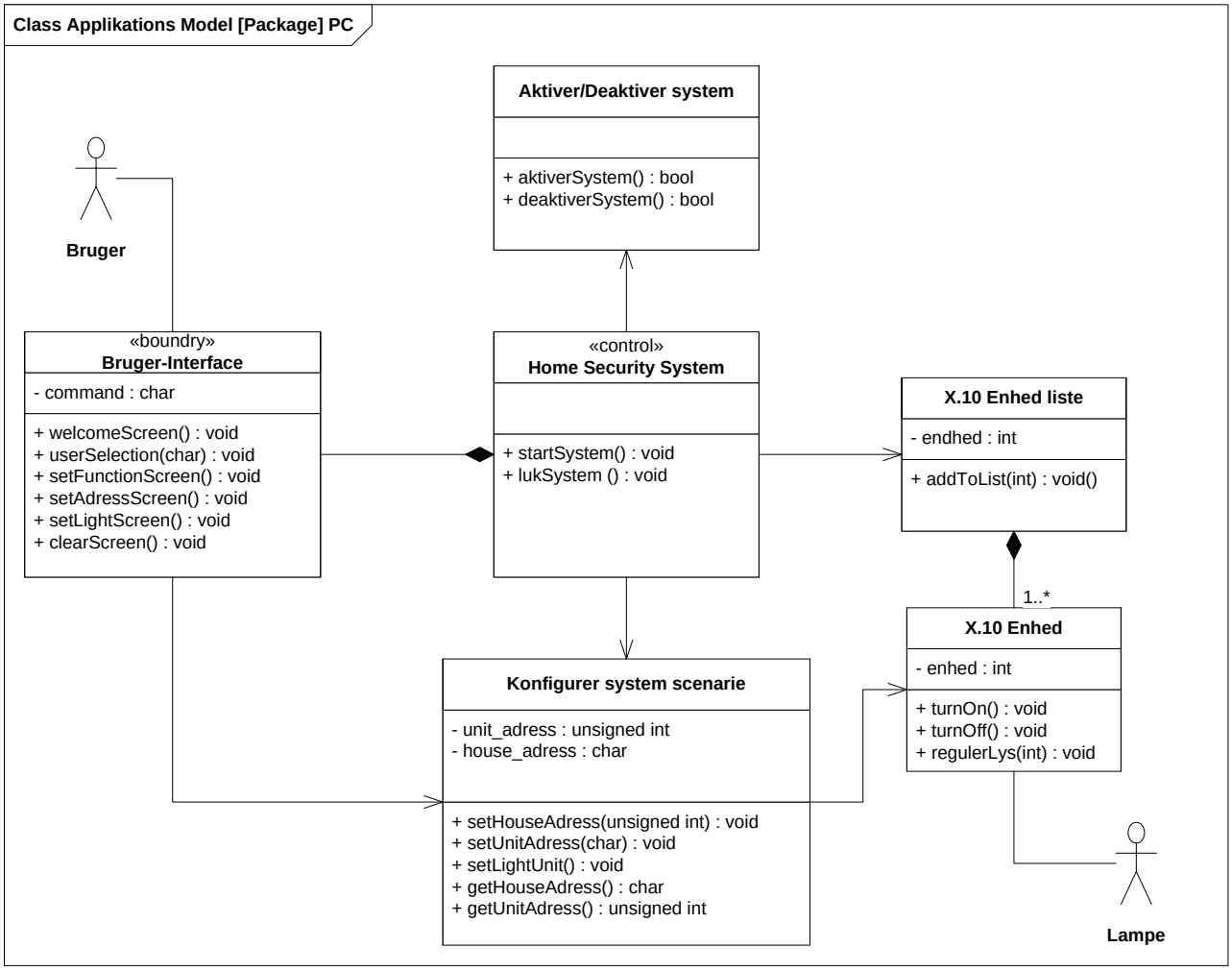


Figur 36 Klasse diagram.

Klassediagram med funktioner og relationer/associationer

PC SW Design

Nedenfor er et klassediagram som er lavet ud fra applikationsmodellen for PC i system-arkitektur dokumentet og specificeret ud fra sekvensdiagrammerne.



Figur 37 - Klasse app for PC

Beskrivelse

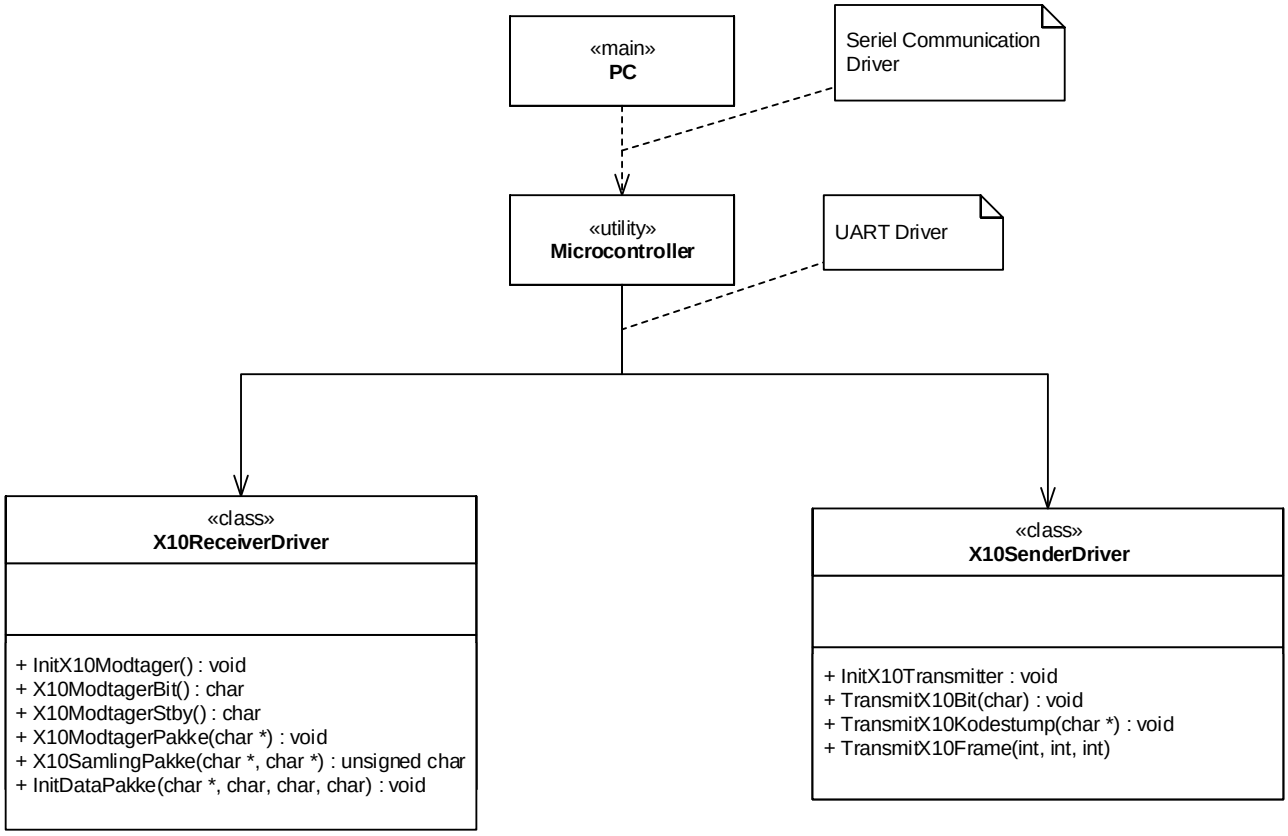
Overstående klassediagram beskriver systemets SW klasser med tilhørende funktioner og medlemsvariabler.

uC SW Design

Dette afsnit beskriver software designet for microcontroller. Der laves et klassediagram for alt softwaren på microcontrolleren og dernæst en klassebeskrivelse for hver driver.

Overordnet klassediagram

Nedenfor ses klassediagram for alt software på microcontrolleren.



Figur 38 – Klassediagram for uC

Vigtig Note:

Den serielle kommunikation mellem PC og microcontroller har vi IKKE valgt at specificere, da denne driver blev udgivet til gruppen af vejleder.

UART driveren der sender og modtager brugerens indtastninger fra terminalen, har vi IKKE valgt at specificere, da det er en driver, der er lavet i en tidligere øvelse, og vi derfor ikke finder det relevant at specificere i overstående klassediagram.

Klassebeskrivelse

X10SenderDriver

Initialer: *DB*

Ansvar: Klassens ansvar er at initialisere X10-Senderenhed samt initialisere zero-crossing detektion.

Metode: `void InitX10Transmitter();`

Parametre: Ingen.

Returværdi: Ingen.

Beskrivelse: Funktionen initialiserer som X10-sendehed og konfigurer PD2 som zero-crossing input

Metode: `void TransmitX10Bit(char bit);`

Parametre: Char bit, der enten kan være '1' eller '0'

Returværdi: Ingen.

Beskrivelse: Funktionen sender en enkelt bit i et forudbestemt tidsrum, når et zero-crossing er detekteret.

Metode: `void TransmitX10Kodestump(char *kodeStump);`

Parametre: Char pointer, der peger på en kodeklump skrevet i et charArray, eks. StartCode

Returværdi: Ingen.

Beskrivelse: Funktionen sender én bit ved hver zero-crossing detektion, så længe pointer parameteren peger på en kodestump, der ikke er nul i et array.

Metode: `void TransmitX10Frame(int _HouseCode, int _UnitCode, int _FunctionCode);`

Parametre: int indexer til houseCode, unitCode samt FunctionCode array. HouseCode kan være 1, UnitCode kan være 1-3 og FunctionCode kan være 1-2, hvor 1 = sluk lys, og 2 = tænd lys.

Returværdi: Ingen.

Beskrivelse: Funktionen sender en X10-Datapakke indeholdende huske, enhed og kommando.

X10ModtagerDriver

Initialer: *JMH*

Ansvar: Klassens ansvar er at initialisere X10-Modtagerenhed samt initialisere zero-crossing detektion.

Metode: `void InitX10Modtager(void);`

Parametre: Ingen.

Returværdi: Ingen.

Beskrivelse: Funktionen initialiserer som X10-modtagerenhed og tillader brug af interrupts. PD2 bliver sat som zero-crossing input og PC3 bliver sat til 120kHz input.

Metode: `char X10ModtagerBit(void);`

Parametre: Ingen.

Returværdi: char.

Beskrivelse: Funktionen modtager én bit ved zero-crossing. Den tager 20 målinger ved hver zero-crossing og returnerer den mest fremkomne værdi.

Metode: `char X10ModtagerStby(void);`

Parametre: Ingen.

Returværdi: char.

Beskrivelse: Funktionen tjekker om X10koden for START er modtaget og forlades ikke før dette er sket. Returnerer 1 hvis START er modtaget.

Metode: `void X10ModtagerPakke(char* DataPakkeModtaget);`

Parametre: char* DataPakkeModtaget pointer som peger et array, der opdateres med de X10 datapakker den modtager.

Returværdi: Ingen.

Beskrivelse: Funktionen modtager en hel X10 datapakke som bliver opdateret i DataPakkeModtaget.

Metode: `unsigned char X10SamligPakke(char* DataPakkeModtaget, char* DataPakkeSamlig);`

Parametre: char* DataPakkeModtaget som indeholder de X10 datapakker der er modtaget. Char* DataPakkaSamlig pointer som peger på et array der indeholder en X10 datapakke om sammenligner dem.

Returværdi: Returnerer 1 hvis datapakkerne er ens eller returnerer 0 hvis datapakkerne er forskellige

Beskrivelse: Funktionen sammenligner to X10 datapakker med hinanden.

Metode : `void InitDataPakke(char* DataPakke, char housecode, char unitcode, char FunctionCode);`

Parametre: Char* Datapakke pointer som peger på et array der er opdateres med X10 datapakker som består af housecode, unitcode og FunctionCode. Char housecode som skrives ind i X10 datapakken. Char unitcode som skrives ind i X10 datapakken. Char FunctionCode som skrives ind i X10 datapakken.

Returværdi: Ingen.

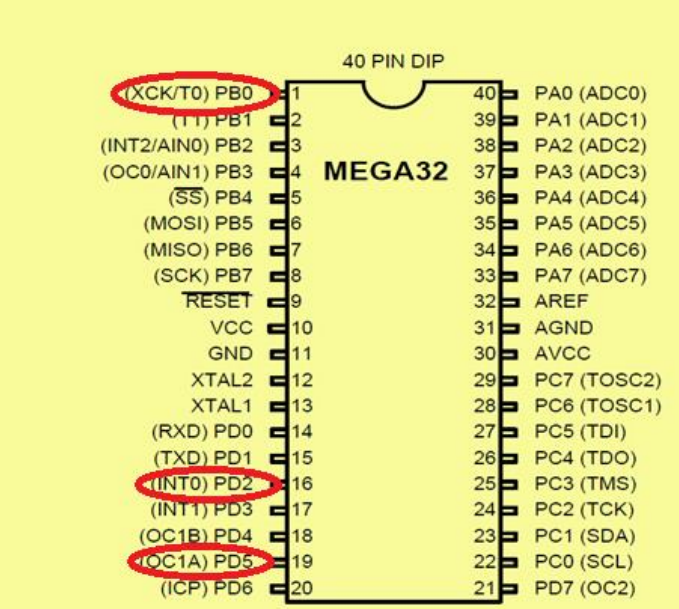
Beskrivelse: Funktionen initialiserer og opbygger X10 datapakker for enheden og sammenligner med modtaget X10 datapakke.

Implementering(SW)

X10SenderDriver

Initial : DB

For at kunne være i stand til at sende signaler over 18 VAC lysnettet bruges 120 KHz burst ved zero-crossing detektion. Dvs. når der er et nulgennemgang i det sinusformet signal på 50 Hz, genereres ét eksternt interrupt, og da overlejres 50 Hz signalet fra lysnyttet med et 120 KHz burst, som skal varer 1 ms, idet 120 KHz signalet er aktivt. Et binært '1' er derved repræsenteret ved et 1 ms langt burst af 120 kHz i nærheden af zero-crossing. Og et binært '0' er repræsenteret af manglen på 120 KHz burst.



Figur 39 - Anvendte Porte ifm. X10SenderDriver

For at generere et 120 KHz burst, der varer 1 ms, initialiseres en timer0 og en timer1, der har følgende interface:

```
// timer 0
void _1MSDelay();
// timer 1
void InitTimer1();
```

Figur 40 - Prototyper

```
#include <avr/io.h>

void _1MSDelay()
{
    // 3686400 Hz /1024 = 3600 Hz
    // 3600 Hz / 4 ~ 1000 Hz = 0,001 s = 1 ms. ( 0,9 ms beregnet )
    TCNT0 = 256-4;
    // Timer 0 i Normal mode og PS = 1024
    TCCR0 = 0b00000101;

    // Vent på Timer 0 overflow flag
    while ((TIFR & (1<<TOV0))==0)
    {}
    // Nulstil Timer0
    TCCR0 = 0;
    // Nulstil Timer 0 overflow flag
    TIFR = 0b00000001;
}
```

Figur 41 - Implementering af Timer0 for 1 ms delay

Af figur 1 markeret med rødt ses det hvilke porte der benyttes ifm. Timer0 og Timer1 samt ekstern interrupt.

Figur 2 viser prototypen for Timer0 og Timer1 funktionerne der alle har ingen parameter og returnerer void.

Figur 3 viser implementering af Timer0, der benytter et tællerregister til at tælle pulser op, der svarer til ca. 1 ms. Dette er beregnet ud fra mega32 intern oscillator på 3,686400 MHz, hvor en prescaler på 1024 bryder denne clock ned, så det svarer til en tusinde del af et second, dvs. 1000 Hz og altså ca. 1 ms. Yderligere sættes Timer0 kontrolregistret i Normal mode og på en prescaler på 1024.

```
void InitTimer1()
{
    // Sæt PD5 som indgang.
    PORTD &= 0b11011111;
    // CTC Mode
    TCCR1A = 0b00000000;
    TCCR1B  = 0b00001001;

    // Konfigurer PD5 - OCR1A til udgang
    DDRD |= 0b00100000;
}
```

Figur 42 - Implementering af Timer1 for 120 KHz burst

Figur 4 viser implementering af Timer1, der benyttes til at generere et 120 KHz timer signal, der beregnes ud fra følgende formel:

Ben frekvens = f_{osc} / (2 * N * (1 + OCRn))
N er timerens prescaler-værdi

Formel 1 - Udregning af OCRn for 120 KHz benfrekvens

Først findes 120 KHz bens frekvens.

Given

$$Ben_{frekvens} := 120000.0Hz$$

$$Ben_{frekvens} = \frac{f_{osc}}{[2 \cdot N \cdot (1 + OCRn)]}$$

$$Find(OCRn) \rightarrow \frac{0.00001536 \cdot MHz}{Hz} - 1.0 = 14.36$$

Dette skrives til OCR1A, efter afrunding.

Figur 43 - Udregning af OCR1A for 120 KHz bens frekvens

Ud fra figur 5 kan der ud fra formel 1 beregnes, hvad der skal skrives til OCR1A benet, dvs. port D ben 5, når ben frekvensen skal være 120 KHz og med en prescaler på N = 1.

I det følgende beskrives kodestumper for interfacet for X10Sender driveren:

```
void InitX10Transmitter();
void TransmitX10Bit( char bit );
void TransmitX10Kodestump( char *kodeStump );
void TransmitX10Frame( int _HouseCode, int _UnitCode, int _FunctionCode );
```

Figur 44 - Headerfil/Prototyper for XSender Driveren.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 3686400UL
#include <util/delay.h>
#include "Timers.h"
#include "X10Sender.h"
```

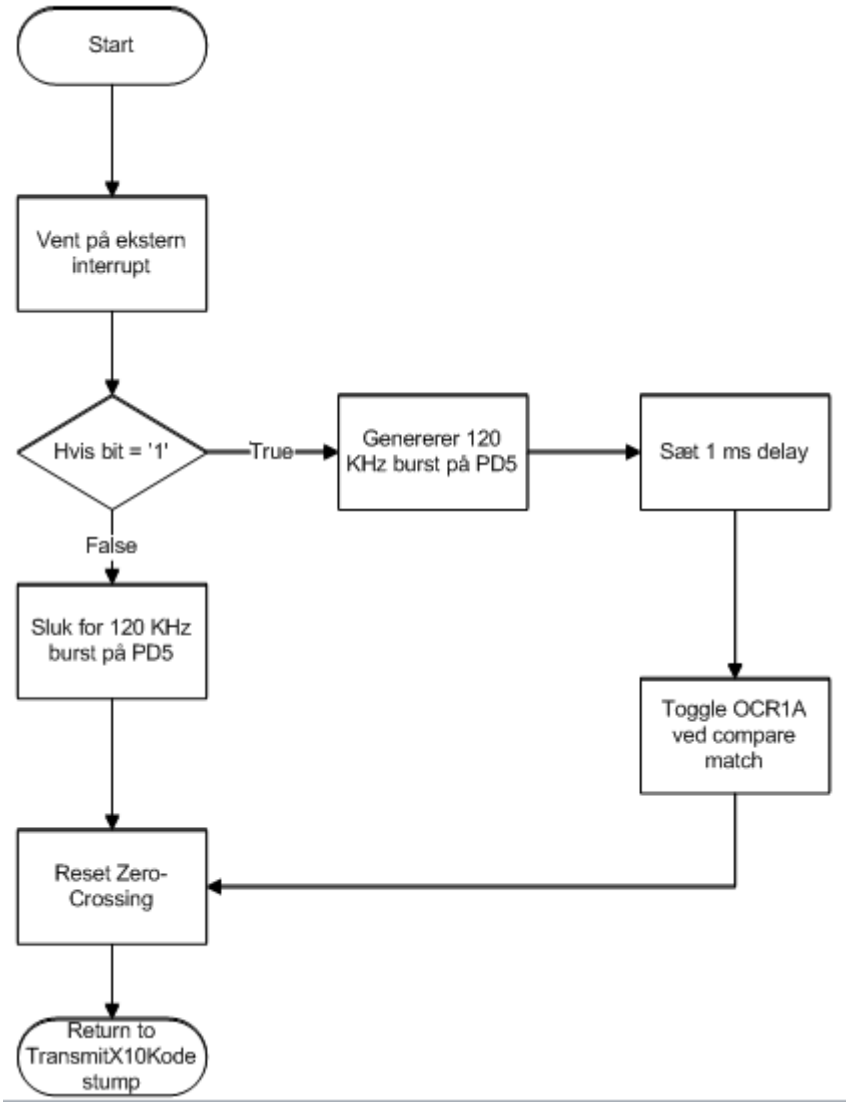
Figur 45 - Header-filer og nødvendige definitioner inkluderet i X10Sender.c

I sourcefilen X10SenderDriver.c inkluderes følgende:

1. Input og output interface for mega32s's porte.
2. Interrupt header-fil, så et ekstern interrupt kan genereres for hvert zero-crossing.
3. Definere mega32 intern clock på 3,68640 MHz for at benytte delay biblioteket.
4. Inkludere selv defineret timer header-fil som beskrevet overstående.
5. Inkludere selv defineret X10Sender header-fil.

Driveren har fire funktioner som sørger for at opfange nulgennemgangen af 50 Hz signalet ved at sætte et interrupt flag eller et højt signal på en port. Den generelle proces af driveren ses af følgende flowchart.

(**Der refereres til bilag for to flow charts der beskriver, hvordan hver kodestump (Housecode, unitcode og functioncode) bliver sendt**)



Figur 46 - Generelle flowchart for X10Sender driver

Flowcharten på figur 7 beskriver det generelle forløb af X10 Senderen. Der afventes et interrupt, når der har været nulgennemgang. Dette interrupt er implementeret i følgende funktion:

```
void InitX10Transmitter()
{
    // Konfigurer PD2 som indgang
    DDRD &= 0b11111011;
    // Trig på for-og bagkant.
    MCUCR |= 0b00000001;

    // Enable ekstern interrupt bit INT0
    GICR |= 0b01000000;

    // No ZeroCrossing => DDRBn = 0x00
    // Yes ZeroCrossing => DDRBn = 0xFF
    ZeroCrossFlag = 0;
    DDRB = 0x00;
}
```

Figur 47 – Aktivering af ekstern interrupt bit INT0

PD2 benyttes som indgang til interrupt bit og hele PORTB sættes til indgang til at toggle zero-crossings detektionen, når følgende interrupt subrutine forekommer, der sætter hele PORTB til udgang.

```
// Zero cross interrupt
ISR (INT0_vect)
{
    ZeroCrossFlag = 1;
    DDRB = 0xFF;
}
```

Et zerocrossing flag sættes højt men bruges ikke til noget og er derfor blot en hjælp til at forstå, at et zerocrossing er opstået.

På denne måde toggles interruptet, sådan så der kommer et interrupt på hvert zero-crossing.

Herefter hvis den pågældende bit er 1, genereres der 120 KHz burst på PD 5, som sættes til at vare 1 ms, forårsaget af Timer0, og OCR1A benet toggles. Hvis den pågældende bit er 0 slukkes for 120 KHz burst på PD5 og dernæst nulstilles zero-crossing detektoren. Herefter vendes der tilbage til funktionen TransmitX10Kodestump, der er beskrevet som følgende:

```
void TransmitX10Kodestump( char *kodeStump )
{
    while (*kodeStump )
    {
        TransmitX10Bit( *kodeStump );
        kodeStump++;
    }
}
```

Figur 48 - Implementering af en kodestump funktion

Det som den lokale variabel eller parameter, kodestump, peger på er et index i et array, som eksempelvis er implementeret ved:

```
char *HouseCodeArray_[SIZEOF_HOUSECODE] = { HOUSEA };
char *UnitCodeArray_[SIZEOF_UNITCODE] = { UNIT1, UNIT2, UNIT3 };
char *FunctionCodeArray_[SIZEOF_FUNCTIONCODE] = { ON, OFF };

char *HouseCodePtr_;
char *UnitCodePtr_;
char *FunctionCodePtr_;
```

Figur 49 - Initialisering af globale arrays og pointers

I TransmitX10Kodestump funktionen undersøges der, om kodestump pointeren peger på noget, der er forskelligt fra 0, og dernæst kaldes en funktion, der sender hver enkelt bit i kodestumpen:

```
void TransmitX10Bit( char bit )
{

    //Reset ZeroCrossing
    ZeroCrossFlag = 0;
    DDRB = 0x00;

    // Vent på interrupt, zeroX
    while( PINB == 0xFF )
    {
    }

    if ( bit == '1' )
    {
        // Genererer 120 KHz "burst" på PD5
        OCR1A = 14;
        // Længden af ét 120 KHz burst
        _1MSDelay();
        // Toggle OCR1A on compare match
        TCCR1A |= 0b01000000;
    }
    else if ( bit == '0')
    {
        // "Sluk" for 120 KHz burst på PD5
        OCR1A = 0;
        TCCR1A &= 0b10111111;
    }

    TCCR1A  |= 0b00000000;
    // Reset ZeroCrossing
    ZeroCrossFlag = 0;
    DDRB = 0x00;
}
```

Figur 50 - Funktion der sender hver enkelt bit fra kodestumpen.

X10 Beskederne er defineret i toppen af source-filen:

```
#define HOUSEA  "01101001"
#define UNIT1   "01101001"
#define UNIT2   "10101001"
#define UNIT3   "01011001"
#define ON      "01011001"
#define OFF     "01011010"

#define START           "1110"
#define SUFFIX_ADR        "01"  // Unit Adress
#define SUFFIX_CMD        "10"  // Command Adress
#define WAIT          "000000"  // Six Zero-Crossings

#define SIZEOF_HOUSECODE    1
#define SIZEOF_UNITCODE     3
#define SIZEOF_FUNCTIONCODE 2
```

Figur 51 - Definere X10 Beskeder

De globale pointers på figur 10 af typen char sættes til at pege på arrayene i funktionen:

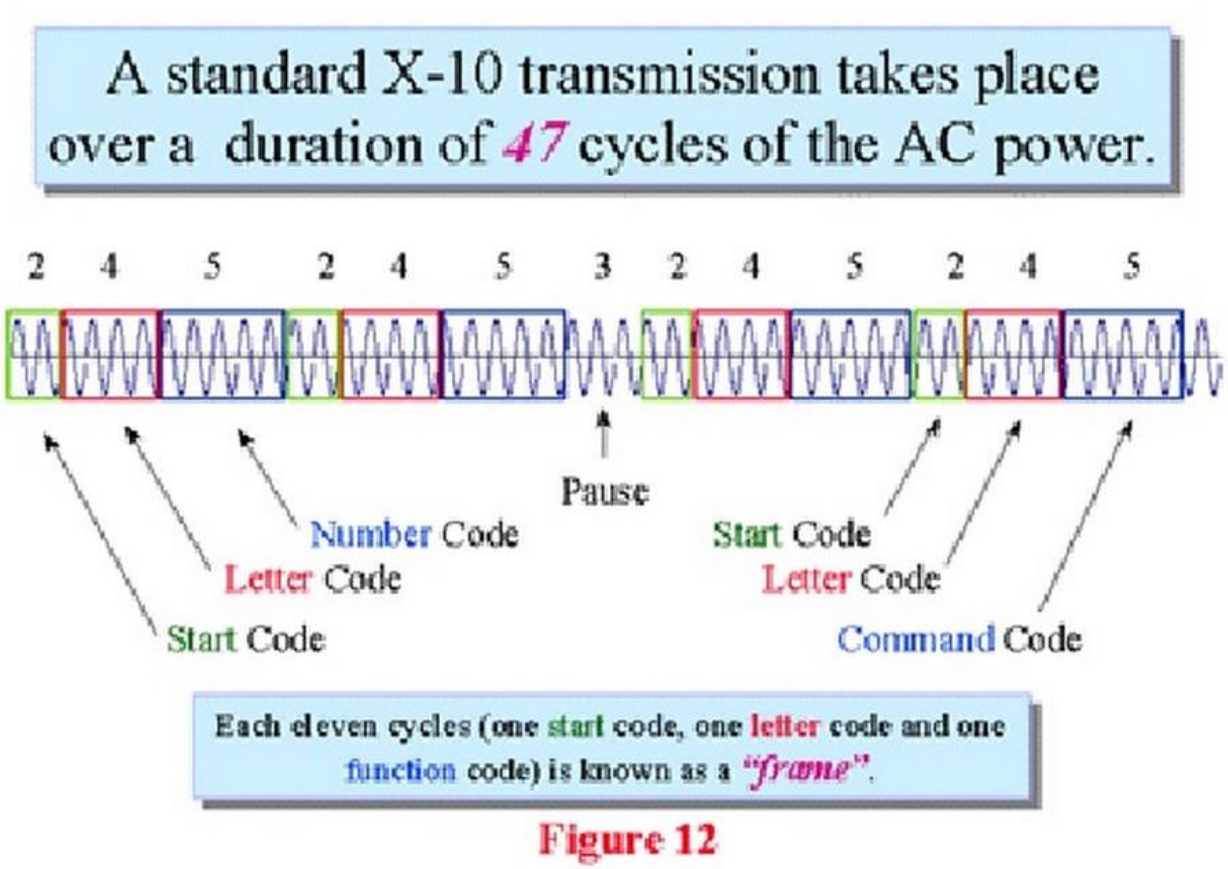
```
void TransmitX10Frame( int _HouseCode, int _UnitCode, int _FunctionCode )
{
    HouseCodePtr_ = HouseCodeArray_[ _HouseCode ];
    UnitCodePtr_ = UnitCodeArray_[ _UnitCode ];
    FunctionCodePtr_ = FunctionCodeArray_[ _FunctionCode ];


    // Send Frame - 11 Cycler:
    // Send adressen første gang
    TransmitX10Kodestump(START);
    TransmitX10Kodestump(HouseCodePtr_);
    TransmitX10Kodestump(UnitCodePtr_);
    TransmitX10Kodestump(SUFFIX_ADR);
    // Send adressen anden gang
    TransmitX10Kodestump(START);
    TransmitX10Kodestump(HouseCodePtr_);
    TransmitX10Kodestump(UnitCodePtr_);
    TransmitX10Kodestump(SUFFIX_ADR);

    // vent 3 cycler( 6 zerocrossing )
    TransmitX10Kodestump(WAIT);

    // Send kommando første gang
    TransmitX10Kodestump(START);
    TransmitX10Kodestump(HouseCodePtr_);
    TransmitX10Kodestump(FunctionCodePtr_);
    TransmitX10Kodestump(SUFFIX_CMD);
    // Send kommando anden gang
    TransmitX10Kodestump(START);
    TransmitX10Kodestump(HouseCodePtr_);
    TransmitX10Kodestump(FunctionCodePtr_);
    TransmitX10Kodestump(SUFFIX_CMD);

    // vent 3 cycler( 6 zerocrossing ) for næste frame
    TransmitX10Kodestump(WAIT);
}
```

Figur 52 - Væsentlig kodestump der tager huskode, unitcode og functioncode som parameter

I figur 14 implementeres en funktion af returtypen void som tager en housecode, unitcode og functioncode som parameter, der alle er indexer til de respektive arrays. F.eks. hvis housecode indexet er 0, er det plads 0 og dermed HOUSE A, som housecode arrayet gennemløber, som der bliver peget på af de respektive, globale pointers. Herefter kaldes TransmitX10Kodestump funktionen beskrevet tidligere, der f.eks. tager det som den globale housecode pointer peger på som parameter, og dermed gennemløbes hver enkelt kodestump. X10ModtagerDriveren beskrevet senere i implementeringsdelen tager disse kodestumper og samler dem til en X10 Datapakke.

Iflg. X10 Protokollen sendes en adresse to gange, hvorefter der ventes 3 cycler eller 6 zero-crossings. Herefter sendes kommandoen to gange og til sidst venter igen 3 cycler før næste frame køres.

For at illustrerer X10 Sender yderligere, vises hermed en figur[2]:



Figur 53 - X10Sender illustration

Figur 11 viser en komplet X10 besked bestående af en såkaldt frame, der er sammensat af en start kode (”1110”), efterfulgt af en huskode, som er efterfulgt af key kode. Key koden kan enten være en enheds adresse eller en funktions kode, afhængig af om beskeden er en adresse eller en kommando. På denne måde sendes en X10 ”Datapakke” indeholdende en huskode, enhed og kommando som bruger vælger fra PC.
Ved en enhedsadresse ses det**,** at den sendte koden slutter på en suffix af ”01”. Ved en funktion kode ses det, hvis suffixen er ”10”.

Når der sendes kode, bruges to zero-crossings til at sende hvert bit som komplementeret bit par. Dvs. et logisk lavt signal, 0, er repræsenteret ved 0-1 og et logisk højst signal, 1, er repræsenteret ved 1-0.

Eksempel på at sende huskoden A, der i C er lavet som en konstant:

```
#define HOUSEA          "01101001"
```

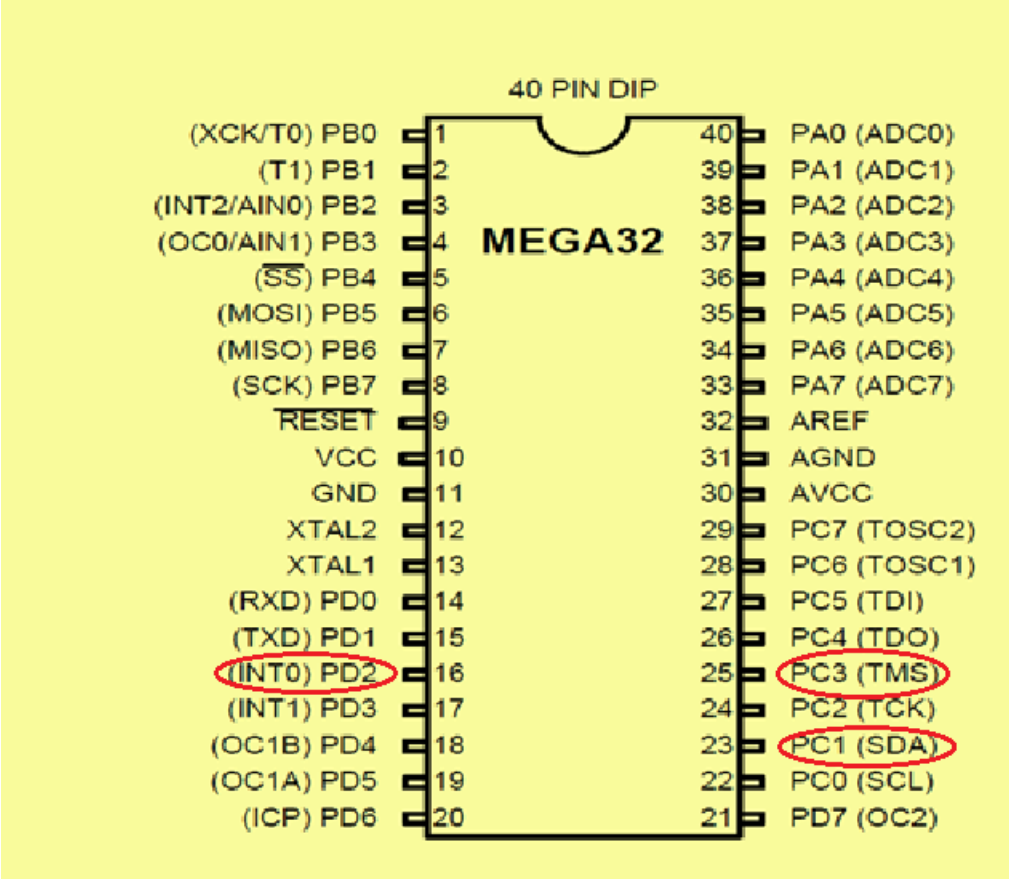
Som altså svarer til Husadressen ”0110”.

[2]Inspirationskilde: http://www.scribd.com/doc/12843158/powerline-communication-using-x10-protocol

X10ModtagerDriver

Initialer: JMH

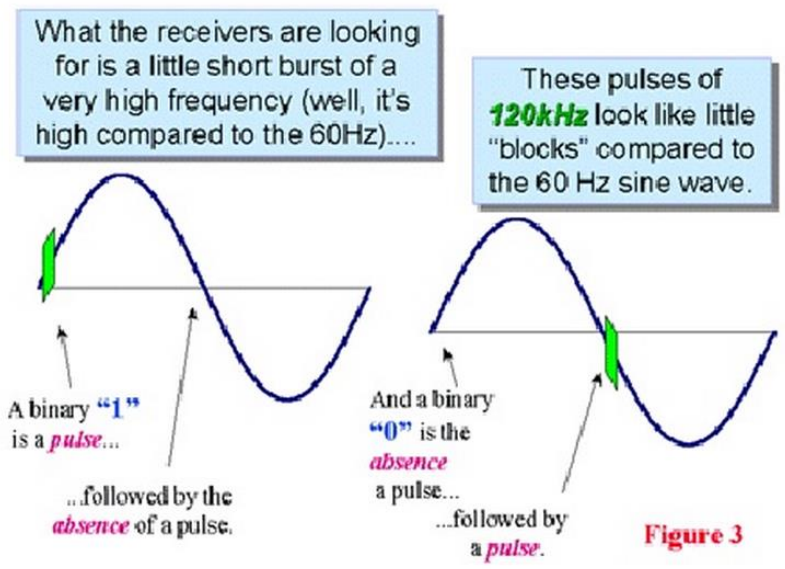
Carrier detektorens funktion er at opfange de 120kHz borsts ved zero-crossing som bliver sendt fra Carrier generatoren og derefter sende signalet videre til enheden/erne indeholdende de kodestumper man forventer.



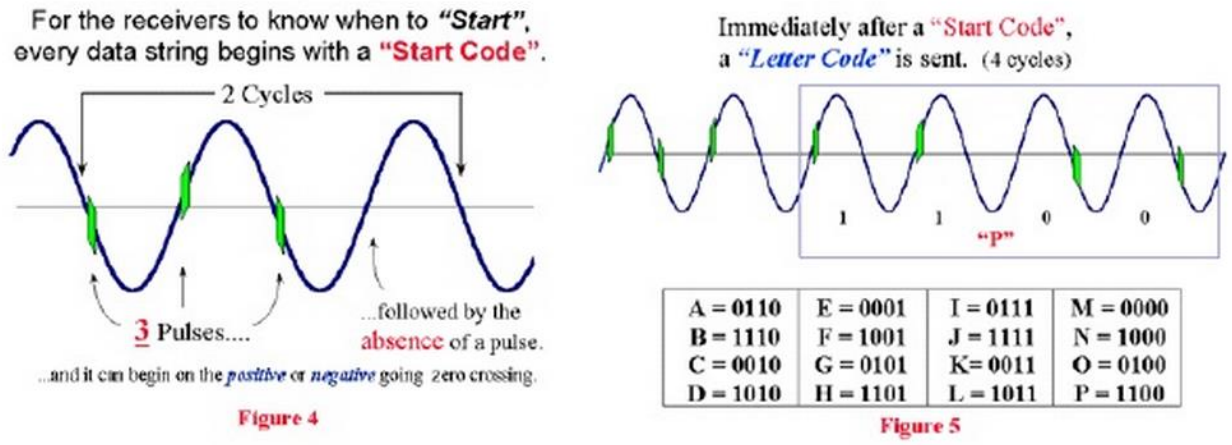
Figur 54 - Anvendte porte til X10ModtagerDriver

På Figur 54 ses de anvendte porte til X10ModtagerDriveren. PD2 bruges som zero-crossing interrupt, PC3 bruges til 120kHz input og PC1 bruges som udgang fra modtageren til enhederne.

For at opfange de kodestumper sendt fra Carrier generatoren er man nødt til at ”lytte” til zero-crossing detectoren og detekterer om det er et binært 1 eller 0 som er repræsenteret i 120kHz burst. Dette gøres ved hjælp af funktionen X10ModtagerBit, der måler hver zero-crossing 20 gange og returnerer den mest fremkomne værdi. Den modtager altså èn bit ved zero-crossing. Det kan ses i figur 2 hvordan Carrier detektoren detekterer de forskellige bits.



Figur 55 – X10Modtager detektion af binært "1" og "0"

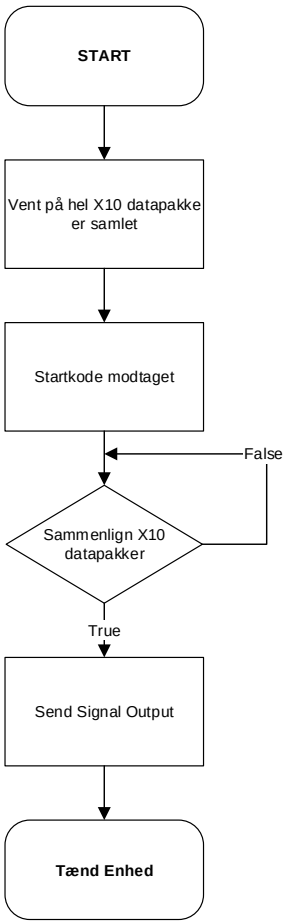


Figur 56 – Modtagelse af START kommando

Bits bliver opsamlet og opdateret i DataPakkeModtaget array og danner en hel X10 datapakke ved hjælp af funktionen X10ModtagerPakke. Vha. X10 datapakken kan man bestemme hvilken enhed der skal tændes. Dette gøres ved at løbe den modtagne datapakke igennem og sammenligne den med en forudbestemt datapakke bestemt af brugeren og se om de er ens. Hvis pakkerne er ens tændes/slukkes enheden som man skriver til.

Figur 56 viser hvordan `#define START "1110"` initialiseres på X10-protokollen, det er derfor nødvendig at have en funktion som står standby og venter på at START kommandoen kommer så datapakken kan blive sendt.
Èn datapakke indeholder 94 bits bestående af 22*2 bits for adresse koden,
6 bits for WAIT efterfulgt af 22*2 bits for command koden.

Figur 57 viser et general flowchart af X10ModtagerDriveren, hvor en hel datapakke samles, sammenlignes og sendes til en enhed.



Figur 57 - General flowchart af X10ModtagerDriver

```
void InitX10Modtager(void)
{
    DDRD &= 0b11111011;
    MCUCR |= 0b00000001;
    GICR |= 0b01000000;

    //PC3 sættes til 120kHz input
    DDRC &= 0b11110111;

    //nulstiller zero-cross flag
    DDRB = 0x00;
};
```

Figur 58 - Funktion til at initialisere X10-modtageren

Funktionen InitX10Modtager initialiserer brugen af Carrier detektoren. Zero-crossing interrupt INT0 enables på PORTD Pin2, MCUCR sættes til at genererer en interrupt anmodning ved alle logiske ændringer på INT0. GICR enabler ekstern interrupt anmodning til INT0.

```
char X10ModtagerBit(void)
{
    DDRB = 0x00;

    int setCounter = 0;

    while(DDRB != 0xFF)
    {

    }

    _delay_us(100);

    int i;
    for(i=0; i<Counter; i++)
    {
        _delay_us(20);

        if(PINC & (1<<PC3))
        {
            setCounter++;
        }
    }

    DDRB = 0x00;

    if(setCounter > Counter/2)
        return '1';
    else
        return '0';
}
```

Figur 59 - Funktion til modtagning af bits fra X10-senderen

Funktionen X10ModtagerBit aflæser de 120kHz bursts sendt fra X10-senderen og bestemmer om det er et binært ”1” eller ”0” som er modtaget. Dette burst kommer ind på PC3 og tjekkes 20 gange for hver eneste zero-crossing for at se om det er binært ”1” eller ”0” som er mest fremkommen.

```
void X10ModtagerPakke(char* DataPakkeModtaget)
{
    int i;

    for(i=0; i<4; i++)
        DataPakkeModtaget[i] = START[i];

    for (i=4; i<94; i++)
        DataPakkeModtaget[i] = X10ModtagerBit();

    for (i=94; i<100; i++)
        X10ModtagerBit();
}
```

Figur 60 - Funktion til at samle hele X10-datapakker

Funktionen X10ModtagerPakke har den rolle at samle alle de bits der er modtaget fra X10ModtagerBit i èn hel X10-datapakke som opdateres i DataPakkeModtaget array. De Bits som er sendt fra X10-senderen indeholder STARTkode, HouseCode, KeyCode (som kan være enten UnitCode eller ON/OFF code) og Suffix. Det er altså disse koder/bits som bliver som danner en X10-datapakke.

```
unsigned char X10SamligPakke(char* DataPakkeModtaget, char* DataPakkeSamlig)
{
    char samlig = 1;

    int i;
    for(i = 0; i < 94; i++)
    {
        if (DataPakkeModtaget[i] != DataPakkeSamlig[i])
        samlig = 0;
    }

    if ( samlig == 1 )
    {
        return 1;
    }
    else
    return 0;
}
```

Figur 61 - Funktion til at sammeligne to datapakker

Funktionen X10SamligPakke's ansvar er at sammenligne to datapakker. Den ene datapakke er den som er blevet samlet i funktionen X10ModtagerPakke hvor den anden datapakke er en som er forudbestemt i main. Grunden til at sammenligne de 2 datapakker er at være sikker på det er de rette kodestumper som er blevet sendt fra X10-senderen ved at sammenligne den modtagende datapakke med den forudbestemte datapakke. Hvis datapakkerne er ens returnerer den true og hvis de ikke er ens returnerer den false.

```
void InitDataPakke(char* datapakke, char housecode, char unitcode, char FunctionCode)
{
    char *HouseKode_ptr = HouseCodeArray_[housecode];
    char *UnitKode_ptr = UnitCodeArray_[unitcode];
    char *FunctionCode_ptr = FunctionCodeArray_[FunctionCode];

    int i, sted = 0;

    // Send adressen første gang
    for(i=0; START[i] != '\0'; i++, sted++)
        datapakke[sted] = START[i];

    for(i=0; HouseKode_ptr[i] != '\0'; i++, sted++)
        datapakke[sted] = HouseKode_ptr[i];

    for(i=0; UnitKode_ptr[i] != '\0'; i++, sted++)
        datapakke[sted] = UnitKode_ptr[i];

    for(i=0; SUFFIX_ADR[i] != '\0'; i++, sted++)
    datapakke[sted] = SUFFIX_ADR[i];


    // Send adressen anden gang
    for(i=0; START[i] != '\0'; i++, sted++)
    datapakke[sted] = START[i];

    for(i=0; HouseKode_ptr[i] != '\0'; i++, sted++)
    datapakke[sted] = HouseKode_ptr[i];

    for(i=0; UnitKode_ptr[i] != '\0'; i++, sted++)
    datapakke[sted] = UnitKode_ptr[i];

    for(i=0; SUFFIX_ADR[i] != '\0'; i++, sted++)
    datapakke[sted] = SUFFIX_ADR[i];


    // Vent 3 cykler (6 zero-crossing)
    for (i = 0; WAIT[i] != '\0'; i++, sted++)
    datapakke[sted] = WAIT[i];
```

```
    //Send kommando første gang
    for (i = 0; START[i] != '\0'; i++, sted++)
    datapakke[sted] = START[i];

    for (i = 0; HouseKode_ptr[i] != '\0'; i++, sted++)
    datapakke[sted] = HouseKode_ptr[i];

    for (i = 0; FunctionCode_ptr[i] != '\0'; i++, sted++)
    datapakke[sted] = FunctionCode_ptr[i];

    for (i = 0; SUFFIX_CMD[i] != '\0'; i++, sted++)
    datapakke[sted] = SUFFIX_CMD[i];

    // Send kommando anden gang
    for (i = 0; START[i] != '\0'; i++, sted++)
    datapakke[sted] = START[i];

    for (i = 0; HouseKode_ptr[i] != '\0'; i++, sted++)
    datapakke[sted] = HouseKode_ptr[i];

    for (i = 0; FunctionCode_ptr[i] != '\0'; i++, sted++)
    datapakke[sted] = FunctionCode_ptr[i];

    for (i = 0; SUFFIX_CMD[i] != '\0'; i++, sted++)
    datapakke[sted] = SUFFIX_CMD[i];
}
```

Figur 62 - Funktion til at initialisere forudbestemt datapakke

Funktionen InitDataPakke bruges til at initialiserer den forudbestemte datapakke. Dette gøres ved at opdaterer arrays med den ønskede datapakke indeholdende STARTcode, HouseCodes, KeyCodes, Suffix. Funktionen løber adressekoderne igennem 2 gange og kommandokoderne 2 gange og genererer ud fra dette den forudbestemte datapakke som opdateres i dataPakkeON/OFF arrayene.

```
char X10ModtagerStby(void)
{
    char StartChk[] = "0000";
    unsigned char StartSamlig;
    int i;

    while(1)
    {
        StartSamlig = 1;

        for(i=0; i<3; i++)
        {
            StartChk[i] = StartChk[i+1];
        }

        StartChk[i] = X10ModtagerBit();

        for(i=0; i<4 && StartSamlig==1; i++)
        {
            if(StartChk[i] != START[i])
                StartSamlig = 0;
        }

        if(StartSamlig == 1)
            return 1;
    }
}
```

Figur 63 - Funktion til at vente på START code

Funktionen X10ModtagerStby har den rolle at tjekke om en START kode er sendt. Den står altså standby og venter på at en START kode kommer, og forlader ikke lykken før der gør. Dette gøres ved at tjekke de modtagene bits fra X10-senderen on tjekke om de passer med START.

```
// Zero-crossing interrupt
ISR(INT0_vect)
{
    DDRB = 0xFF;
}
```

Figur 64 - Funktion til styring af zero-crossing

Funktionen for dette er at sætte alle ben på PORTB høj ved zero-crossing. Der i de funktionen X10ModtagerBit en linje kode (`DDRB = 0x00;`) til at nulstille PORTB for at sætte den lav så der igen kan modtages et zero-crossing. Benene på PORTB skifter altså fra høj til lav konstant.

```
void InitSignalOutput(void)
{
    DDRC |= 0b00000010;
    PORTC |= ( 1<<PC1 );
}
```

Figur 65 - Funktion til at initialiserer Signal Output til enhed

Funktionen InitSignalOutput initialiserer Ben 1 PORTC som output signal fra modtageren til enhed. Dette gøres ved at sætte PC1 til aktiv høj lysstyring.

```
void SendSignalOutput(char i)
{
    switch (i)
    {
        case 0:
        break;

        case 1:
        PORTC |= ( 1<<PC1 );
        break;

        case 2:
        PORTC &= ( 0<<PC1 );
        break;
    }
}
```

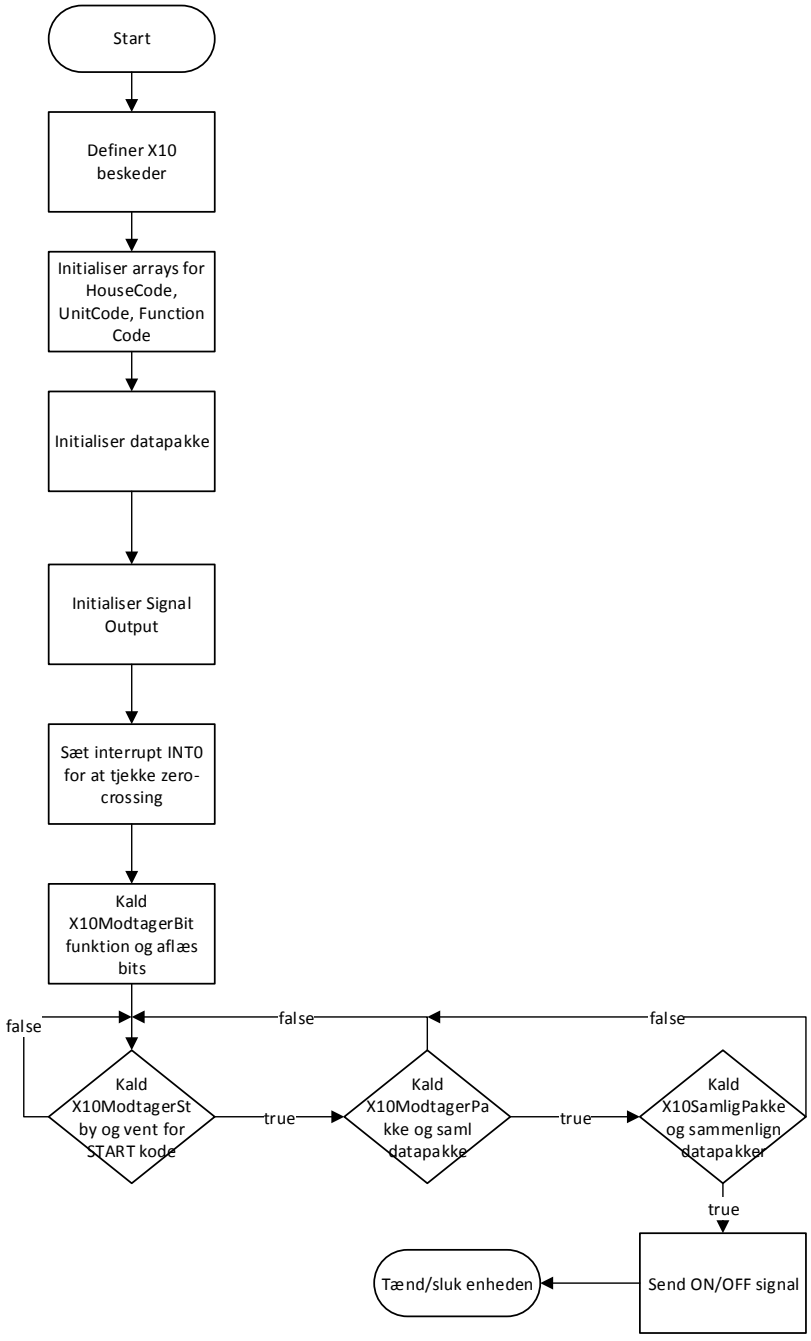
Figur 66 - Funktion til at sende signal til enheder

Funktionen SendSignalOutput har til ansvar at sende et signal som kan tænde/slukke en enhed ud på PC1 som er initialiseret i InitSignalOutput. Dette gøres vha. cases hvor index i sættes til enten 1 eller 2 for sætte PC1 som høj eller lav.

```
#include <avr/io.h>
#include <avr/interrupt.h>

#include "X10ModtagerDriver.h"
#include "Signal Output.h"

#define House_ID 1
#define Unit_ID 2


int main(void)
{
    InitX10Modtager();
    InitSignalOutput();
    sei();

    char dataPakkeON[94];
    char dataPakkeOFF[94];

    char DataPakkeModtaget[94];

    InitDataPakke(dataPakkeON, House_ID, Unit_ID, 2);
    InitDataPakke(dataPakkeOFF, House_ID, Unit_Kald ID, 1);

    char state = 0;

    while(1)
    {
        switch(state)
        {
            case 0:
                if (X10ModtagerStby())
                    state = 1;
                break;

            case 1:
                X10ModtagerPakke(DataPakkeModtaget);
                state = 2;
                break;

            case 2:
                if (X10SamligPakke(DataPakkeModtaget, dataPakkeON))
                    SendSignalOutput(2);

                if (X10SamligPakke(DataPakkeModtaget, dataPakkeOFF))
                    SendSignalOutput(1);
                state = 0;
                break;
        }
    }
}
```

Figur 67 - X10ModtagerDriver main funktion

Det er her i main funktionen hvor koden for X10-modtageren afvikles. Der startes med at initialiserer X10-modtageren samt SignalOutput. Så bliver arrayene dannet og de forudbestemte datapakker initialiseres i funktionen InitDataPakke. Der er valgt at bruge cases til hvordan koden afvikles, hvor den først står standby og tjekker efter START kode, hvis det er true går den videre og samler en hel datapakke modtaget fra X10-senderen, når det er færdigt går den til case2 hvor der funktionen X10SamligPakke sammenligner de to X10-datapakker og bestemmer ud fra disse om der skal sendes ON/OFF signal og om enheden tænder/slukker. Det er desuden vist i flowchartet for main funktionen.



Figur 68 – Flowchart over main funktionen af X10ModtagerDriveren

Figur 68 viser et flowchart over main funktionen i X10ModtagerDriveren, hvor den starter med at initialiserer de nødvendige ting hvorefter at den venter på zero-crossing interrupt for at aflæse de sendte bits fra X10-senderen. Der ventes så på at START koden modtages og hvis dette er sket samler den alle bits i en hel X10 datapakke. Denne pakke bliver sammenlignet med datapakkeON/OFF og hvis de passer sender den et ON/OFF signal til enhederne som vil tænde/slukke.

DE2 kodelås

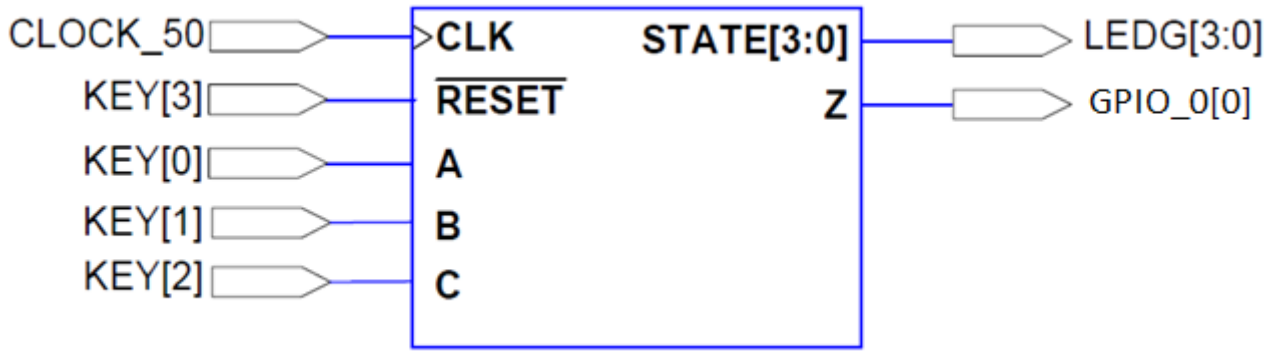
Initialer: JRL og JMH

Denne opgave er taget fra kurset E2DSD øvelse 7, det er en opgave over hvordan man laver en kodelås.

Vi har brugt dette til vores projekt, da vi skal bruge en kodelås til at kunne låse for en bruger interface, inden brugeren har kunnet få lov til at gøre andet.

Code lock.

1.) Create the state machine below using ***two processes***. One ***state_reg*** process and one process merging both ***output*** and ***next_state.***
Download and test the design on the DE2board.
A, B and ***C*** are input switches where a input sequence is entered. When the correct code sequence (illustrated below) is pressed Z is set to '1' (lock is open):
ABC = CODE1 -> NONE -> CODE2 -> NONE -> CODE3 -> NONE
Where the codes are defined as:
- CODE1 = "101" Assert key B
- CODE2 = "110" Assert key A
- CODE3 = "110" Assert key A
- NONE = "111"All keys are released.



Every input code sequence can asserted for an arbitrary length in time.
Z is a Moore-output, that is '1' after a correct input code sequence and stays '1' until one or more keys are asserted. When all keys are released the incorrect attempt counter is cleared and the code lock is ready for a new input code sequence.
After an incorrect input code sequence the code lock abandons the lock sequence and increment (in state S9) the number of incorrect attempts and will unconditionally go to state S8. If the total number of incorrect attempts is greater than 3 the key lock is stuck in S8 until system reset, otherwise the code lock goes to state *S1* and is ready for a new attempt.
RESET is asynchronously and changes state to *S1* and clears the incorrect attempt counter.
STATE shows the current state in binary format, (e.g. S1 = "0001"). This output is solely for test and debug purpose.

```
16  begin
17      state_reg : process ( clk, reset, countattempt)
18      begin
19          if reset = '0' then
20              present_state <= s1;
21              countattempt <= 0;
22          elsif rising_edge(clk) then
23              present_state <= next_state;
24          if (present_state = s9) then
25              countattempt <= countattempt +1;
26          end if;
27      end if;
28      end process;
```

Figur 69 Reset

Her opsættes koden til at der at kunne lave et asynkront reset, at der kan skiftes mellem hver state og at S9 er en fejltæller.

```
29
30      outputs : process (countattempt, present_state, a, b, c)
31      begin
32          case present_state is
33
34              when s1 =>
35                    state <= "0001";
36                    z <= '0';
37                if (a= '1' and b= '1' and c= '1') then
38                    next_state <= present_state;
39                elsif ( a= '1' and b= '0' and c= '1') then
40                    next_state <= s2;
41                else
42                    next_state <= s9;
43                end if;
```

Figur 70 State 1

State 1-6 opsættes næsten alle på samme, ved at de går videre til næste state hvis den rigtige tast er trykket, ellers lægges der 1 til fejltælleren i S9

```
100             when s7 =>
101                   state <= "0111";
102                   z <= '1';
103               if (a= '1' and b= '1' and c= '1') then
104                   next_state <= present_state;
105               else
106                   next_state <= s8;
107               end if;
```

Figur 71 State 7

State 7 er det state der gør at output signalet sættes til 1, hvis koden der er trykket er rigtig.

```
109             when s8 =>
110                   z <= '0';
111                   state <= "1000";
112                if ((a= '1' and b= '1' and c= '1') and countattempt < 3) then
113                   next_state <= s1;
114                else
115                   next_state <= present_state;
116                end if;
```

Figur 72 State 8

Her gå Z signalet lavt igen og hvis koden var trykket rigtigt, går state tilbage til start state, ellers bliver systemet låst og kan kun åbnes igen ved at det bliver "reset"

```
118             when s9 =>
119                   state <= "1001";
120                   next_state <= s8;
```

Figur 73 State 9

Hvis der er blevet trykket forkert mere end tre gange sendes signalet videre til S8 hvor det bliver last.

Bruger interface

Initialer: **JRL**

Der skulle laves et visuelt program som brugeren skal kunne bruge til at styre de forskellige elementer der er i dette projekt.

Vi har valgt at lave vores bruger interface som en konsol applikation i C++ kode, som skal kunne bruges til at sætte forskellige værdier på en valgt enhed, samt tænde eller slukke disse enheder.

Herunder er koden til vores bruger interface, det der står som grøn kommentar er det kode som ikke er nået at blive skrevet.

```
int main()
{
    char choice1, choice2;
    char h;
    int u;

    do
    {
        do
        {
            system( "cls" );
            cout << "Welcome" << endl;
            //PC4 = '1'
            //Dette betyder at når koden på DE2 boardet er tastet korrekt ind,
            //kan man bruge systemet, ellers ikke.
            {
            cout << "1. Press 1 for Menu" << endl;
            cout << "2. Press 2 to activate unit" << endl;
            cout << "3. Press 3 to deactivate unit" << endl;
            cout << "4. Press 4 to close down system" << endl;
            choice1 = getch();

            if( choice1 == '4' ) exit(1);
            }

        }while( choice1 != '1' & choice1 != '2' & choice1 != '3');

            if( choice1 == '1' )
            {
                do
                {
                    system( "cls" );
                    cout << "\nMenu screen" << endl;
                    cout << "1. Press 1 for set unit" << endl;
                    cout << "2. Press 2 to go back to welcome screen" << endl;
                    choice2 = getch();
```

```
    }while(choice2 != '1' & choice2 != '2');

        if(choice2 =='1')
        {
        system( "cls" );
        cout << "\nUnit Screen" << endl;
        cout << "\nEnter House Adress (A-P):" << endl;
        cin >> h;

            if(h >= 'a' & h <= 'q' || h >= 'A' & h <= 'Q')
            {
                cout << "\nHouse letter accepted\n\n";
                choice2 = getch();
                system( "cls" );
                cout << "\nEnter Unit Adress (1-16):" << endl;
                cin >> u;

                if(u >= 1 & u <= 16)
                {
                    cout << "\nUnit number accepted\n";
                    choice2 = getch();
                    system( "cls" );
                    cout << "\nUnit:" << u << " in House:" << h << " is now chosen";
                }
                else
                    cout << "Wrong number try again";
            }
            else
                cout << "Wrong letter try again";

        choice2 = getch();
        }
  }

        if( choice1 == '2' )
        {
            //aktivering af unit
            //denne kode stump skal kunne aktivere den unit som er sidst sat i Unit Reg
        }

        if( choice1 == '3' )
        {
            //deaktivering af unit
            //denne kode stump skal kunne deaktivere den unit som er sidst sat i Unit Reg
        }

    }while(1);
}
```

Det er blevet sat op på den måde at programmet går ind i de forskellige menuer alt efter brugerens valg.

Grunden til at der står at man kan vælge mellem house A-P og unit 1-16, men kun har tidligere beskrevet at vi kun vil bruge hus A og unit 1-3. I vores projekt vil vi kun vise de tre units, men allerede nu har vi gjort klar til at der kan tilføjes flere units uden at skulle ind og ændre i denne kode senere.

Test(SW)

Zero-Crossing Detektor og X10Sender

Initialer: DB

Indhold

Følgende indeholder test af Zero-Crossing Detektor, som nemt kan laves ved at generere et ekstern interrupt på microcontrolleren (mega32) og signalere med et højt signal på en port, hvis det er tilfældet, ellers toggle interruptet så et interrupt forekommer ved hvert zero-crossing og trigges på både forkant og bagkant, dvs. rising edge og falling edge. Yderligere testes X10Senderen, som skal sørge for at sende en enkelt bit i et forudbestemt tidsrum, når zero-crossing er detekteret.

Formål

Formålet med test af Zero-Crossing Detektor samt X10Sender for SW er at illustrere og vise zero-crossing detektion samt at tjekke at de korrekte bits, bliver sendt ved 120 KHz burst, når et interrupt forekommer. Disse bits repræsenterer 1'er og 0'er for en huskode, en adresse på enhed samt en kommando, f.eks. tænd eller slukke en lampe.

Metode

Følgende metode tages udgangspunkt i en LED driver og en SWITCH driver, som kan styre LED'erne og switchene ved simple kald af funktioner. Disse drivers er lavet på forhånd i en tidligere øvelse. Der tændes en LED på et af STK-500's PORT, hvis den sendte bit er et logisk højt signal, '1', og tænde en anden LED på en anden STK-500 PORT, hvis den sendte bit er et logisk lavt signal, '0'. Derved er det muligt at undersøge om de enkelte bits for en frame er sendt korrekt iflg. X10 Protokollen.

Følgende main program tester vha. en switch, at de enkelte bits sendes.

```
int main(void)
{
    initSwitchPort();
    initLEDport();
    InitTimer1();
    InitX10Transmitter();

    sei();

    // Simuler zerocross

    while(1)
    {
        if(switchOn(0))
        {
            TransmitX10Frame(0, 1, 0);
        }
    }

    return 0;
}
```

Figur 56 - Main program der tester sendingen af de enkelte bits

Der sker følgende:

1. Switch porten initialiseres som indgange på hele PORT A.
2. LED porten initialiseres som udgange på hele PORT C.
3. Timer 1 initialiseres, der sætter PD5 som udgang og ”klargør” 120 KHz burst.
4. Funktionskald InitX10Transmitter() der konfigurer PD2 som indgang og aktiverer ekstern interrupt bit INT0 samt zero-crossing som indgang på hele PORTB.
5. Globalt interrupt aktiveres.
6. Uendelig while-løkke påbegyndes.
7. Hvis switchen på PORT A ben0 trykkes ned, kaldes en funktion
8. Funktionskald TransmitX10Frame(0,10)
9. Funktionskald TransmitX10Kodestump(char *kodestump)
10. Funktionskald TransmitX10Bit(*kodestump);

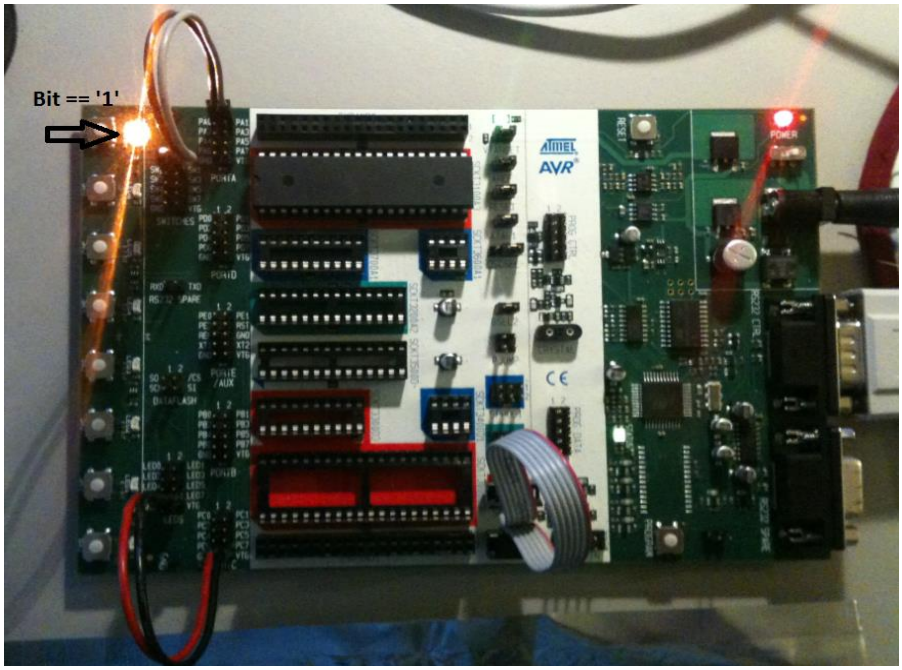
(Der refereres til implementering for SW for yderligere detaljer omkring funktionkaldene)

TransmitX10Bit(*kodestump) funktion:

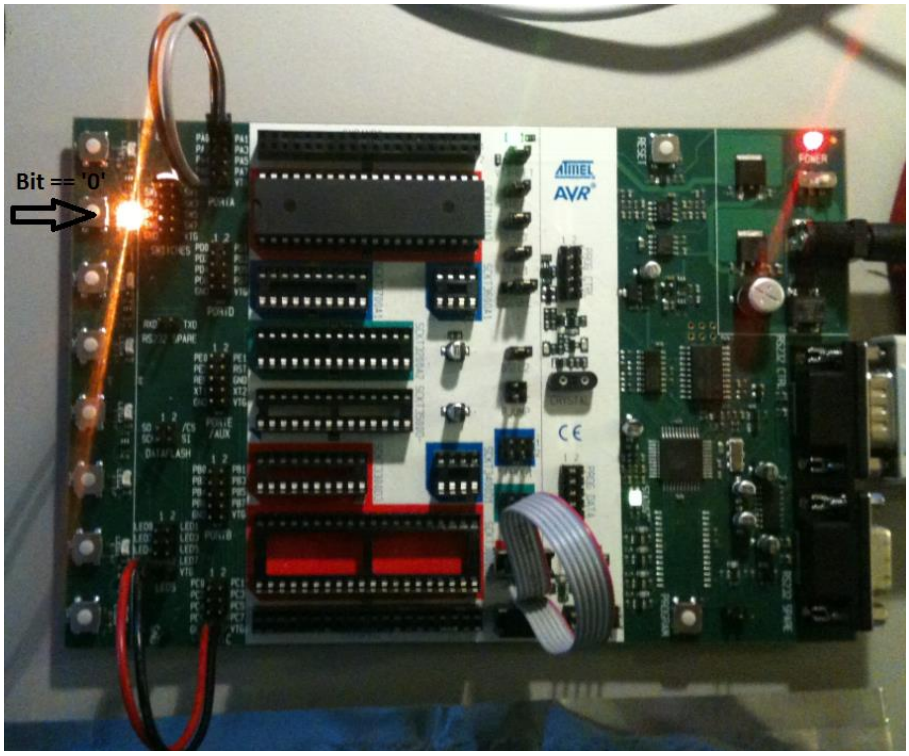
```
void TransmitX10Bit( char bit )
{

    //Reset ZeroCrossing
    ZeroCrossFlag = 0;
    DDRB = 0x00;

    // Vent på interrupt, zeroX
    while( PINB == 0xFF )
    {
    }

    if ( bit == '1' )
    {
        // Genererer 120 KHz "burst" på PD5
        OCR1A = 14;
        // Længden af ét 120 KHz burst
        _1MSDelay();
        // Toggle OCR1A on compare match
        TCCR1A |= 0b01000000;

        turnOnLED(7);
        _delay_ms(1000);
        turnOffLED(7);
        _delay_ms(1000);
    }
    else if ( bit == '0')
    {
        // "Sluk" for 120 KHz burst på PD5
        OCR1A = 0;
        TCCR1A &= 0b10111111;
        turnOnLED(6);
        _delay_ms(1000);
        turnOffLED(6);
        _delay_ms(1000);
    }

    TCCR1A  |= 0b00000000;
    // Reset ZeroCrossing
    ZeroCrossFlag = 0;
    DDRB = 0x00;
}
```

Figur 74 - Kodestump til at sende en enkelt bit

Resultat



Figur 75 - Billede af test på STK-500. LED7 = '1'



Figur 76 - Billede af test på STK-500. LED 6 = '0'

X10ModtagerDriver:

Initialer: JMH

Formål:

Formålet med at teste X10ModtagerDriveren er at se om den virker som forventet, så der kan modtages bits og datapakker fra X10-senderen til at tænde enheder.

Metode:

Uden hardware:

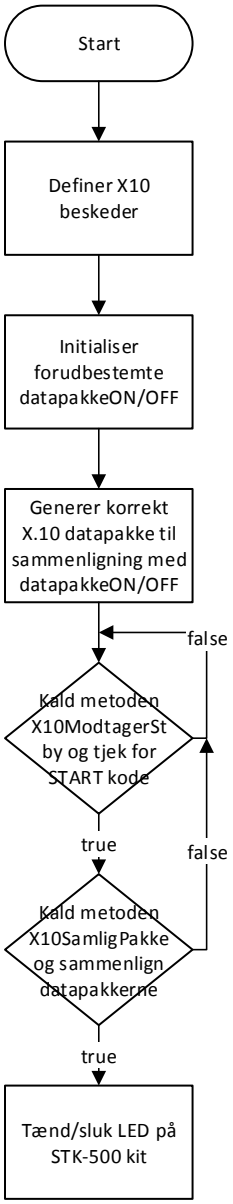
For at teste X10ModtagerDriveren softwaren for sig selv uden hardware vil det være muligt at bruge en to forudbestemte X10-datapakke for at simulerer det som hvis man modtog en X10-datapakke over X10-protokollen. De to forudbestemte X10-datapakker testes i main funktionen som vil tjekke for START koden og sammenligne datapakkerne og vil i stedet for en reel enhed få en LED på STK-500 kittet til at lyse hvis X10-datapakken indeholder datapakkeON kode, og slukke den hvis den indeholder datapakkeOFF kode.

Med hardware:

Efter al softwaren er skrevet og buildet succesfully er det muligt at teste om X10-modtageren egentlig virker som ventet ved at lave en UART forbindelse mellem en mikrocontroller og computeren. Ved at bruge noget af softwaren fra X10-senderen til at generer en X10-datapakke kan man sende den til mikrocontrolleren hvor X10ModtagerDriveren er. Så vil det være muligt at modtage denne datapakke og sammenligne den med en allerede lavet datapakke og se om de stemmer overens. Man vil ud fra denne sammenligning få en bestemt LED på STK500-kittet til at lyse hvis sammenligningen er true (1) eller forblive slukket hvis false (0). Dette er ud fra samme princip som hvis det reelle produkt var opsat.

Resultat:

Da der ikke har været tid til at lave en ordenligt software test af X10ModtagerDriveren bliver der her beskrevet hvordan denne tænkes at lave hvis der havde været tid. Resultaterne for disse test var forventet at tænde for en LED på STK-500 kittet som var det en reel enhed der blev tændt. Samt teste om modtageren opfanger de rigtige bits fra senderen og kan samle dem til en hel datapakke indeholdene to addressekoder med START, HouseCode, UnitCode og Suffix og to kommandokoder med START, HouseCode, ON/OFFcode og Suffix.



Figur 77 - Flowchart for test af X10ModtagerDriver

DE2 kodelås

Initialer: **JRL**

Formål:

Formået med at teste koden til kodelåsen er at vi skal kunne se at den reagere som vi gerne vil, altså om låsen åbner, når koden der bliver trykket er den rigtige og at den låser systemet hvis der bliver trykket forkert et givet antal gange.

Metode:

Måden vi kan se at vi har lavet det rigtige er ved at vi får en lysdiode til at lyse grøn hvis koden er rigtig og rød hvis koden er forkert. Hvis den røde diode lyser, skal systemet åbnes igen ved at trykke på ”reset”.

Udover dette lyser en grøn udfor de forskellige knapper der trykkes på, for at indikere at der er registret et tryk fra brugeren.

Resultat:

Når der blev trykket den rigtige kode, tændte den grønne lys diode og når der blev trykket forkert mere end tre gange lyste den røde lysdiode, samt systemet godt kunne blive ”reset”, så der kunne prøves igen.

Bruger interface

Initialer: **JRL**

Formål:

Formålet med at teste interfacet er at, kunne vide med sikkerhed at de indtaste værdier også er de værdier der bliver sendt videre til X.10 senderen og at programmet reagere som det skal.

Metode:

Der indtastes nogle forskellige værdier, for at se om programmet acceptere dem eller ej, hvis de ikke bliver accepteres skal programmet sende en tilbage til start menuen. Hvis værdierne bliver accepteret skal programmet sende en bekræftelse og gå tilbage til start menuen, herefter skal der være mulighed for aktivere eller deaktivere den valgte enhed.

Resultat:

Da der blev indtastet de rigtige værdier, kom der en bekræftelse tilbage og da der blev tastet forkert igen programmet tilbage til start og man fik her mulighed for at prøve igen.

Vi ville gerne ha haft testet koden til aktivering/deaktivering af enhed, men vi nåede ikke at få skrevet alt koden færdig. Måden dette skulle ha været testet ville ha været at når en enhed blev aktiveret, ville det svare til at en lys diode blev tændt og slukket igen når en deaktiverings kommando bliver sendt.

Bilag

(Se CD-ROM)