AARHUS UNIVERSITET
INGENIØRHØJSKOLEN

Home Security System

- Styring og monitorering af enheder over lysnettet

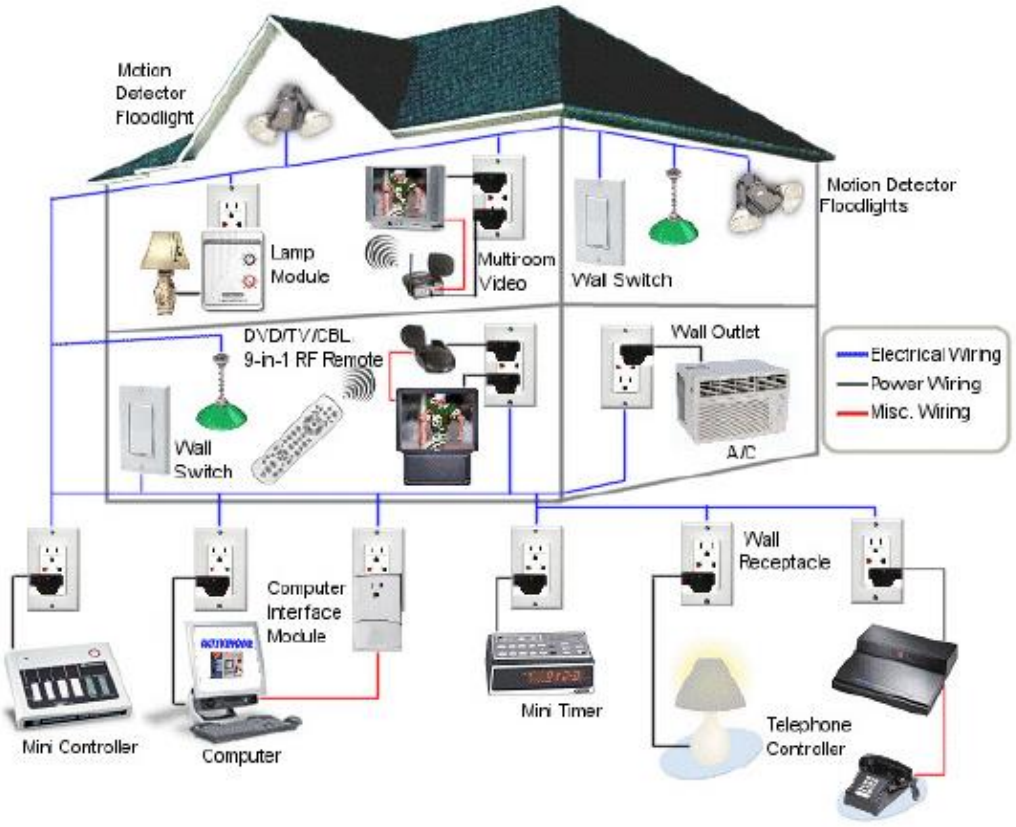
E2PRJ2 - Projektrapport

- 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	



Resume

Formålet med dette projekt er at der ud fra et problemoplæg, skal kunne færdigøre et system, rapport og dokumentation heraf. Der skal desuden læres at arbejde sammen som en gruppe, hvor der fordeles ansvarsområder ud. Der skal gøres rede for, hvordan X.10 protokollen virker og udforme en systembeskrivelse ved hjælp af HW/SW blokke.

Hvad skal der til for at forebygge tyveri? Kan man simulere et scenarie som forvirrer tyven, inden vedkommende bryder ind i hjemmet?

Der foretages løsninger valgt ud fra X.10 protokollen som beskriver, hvordan et system til tyveri forebyggelse kan laves. Hardware løsningen er hentet fra denne X.10 protokol som inspiration, og der er bygget de nødvendige kredsløb for at løse opgaverne. Softwaren er skrevet ud fra X.10 standarder, som får enheder til at tænde over el-nettet.

Der har været anvendt nogle forskellige metoder til færdiggørelse af projektet som inkluderer følgende – Århus modellen, V-modellen, og vandfalds-metoden.

Gruppen har formodet at få en carrier generator for sender og carrier detektor for modtager i HW til at virke, og der er givet et forventet resultat. Der er desuden blevet testet om SW til sender og modtager virker og yderligere givet resultat som forventet.

Abstract

The goal of this project is that we should be able to create a system, a report and a documentation of that from a given problem presentation. Furthermore, we should learn how to work together as a group, where the different areas of responsibility are dealt out to each individual member of the group. There need to be accounted for how the X10-protocol work by forming a system description of SysML HW/SW blocks.

How do we prevent thievery? Is it possible to simulate a scenario that will deceive the thief before he enters our home?

Our solutions are chosen from the X10-protocol that describes how a system can be used to prevent thievery is made. The hardware solutions are inspired by the X10-protocol and build of the necessary circuits to ensure the solutions is capable of solving the tasks for an X10-protocol. The software is written out of the X10 standards for transmitting and receiving bits, which will enable different units to turn ON/OFF within a closed power-grid.

Throughout the project we have used a few different methods for completion of the project. These include methods such as the Aarhus model, the V-model and the waterfall-model.

At the end of the project we have been able to get the hardware for our X10transmitter and receiver to work as intended, and give the expected result. Furthermore, the software for the X10transmitter is tested and working. The result which was provided by this test is also giving the expected result.

Indholdsfortegnelse

Indledning

Dette 2. semester projekt er et tværfagligt projekt mellem 5 andre kurser på samme semester fra IHA Katrinebjerg i Aarhus N. Projektoplæg hedder Home Security System, hvor har været forskellige underemner der kunne vælges i mellem. Der er i det følgende blevet valgt at fokusere på tyveri forebyggelse, da emnet i første øjekast har virket som et emne med meget indhold i.

Som problem omkring emnet er der lagt vægt på, at hjemmetyve er blevet mere intelligente. Tyvene i dagens samfund har nemmere at se om folk virkelig er hjemme eller ikke er hjemme i forhold til, at der er tændte apparater eller lys i hjemmet. Derfor laves et system, der kan variere i tændetid i forhold det nuværende system, der er konstant tændt eller slukket ved manuelle kontakter.

Der laves derfor et system som kører på daværende system men med mulighed for at kunne implementere variabler til valgte enheder. Udover dette skal bruger selv kunne indstille de tilsluttede enheder via et bruger-interface.

Projektet er opbygget efter V-modellen, og der er brugt bottoms-up test rækkefølge. Først bliver der beskrevet hvad der skal bruges for at kunne opfylde vores problem. Efterfølgende undersøges det hvordan hele systemet skal se ud, hvor der herefter dykkes ned i systemet, hvor de enkelte dele beskrives. Efter færdiggørelse af de enkelte dele, bliver der foretaget en enhedstest. Lykkedes dette integrationstestes hele systemet, hvor HW komponenter og SW skal kunne kommunikere med hinanden.

Dette står beskrevet i denne rapport og med flere detaljeret i projekt dokumentationen.

Opgaveformulering

Hvem har ikke oplevet at komme hjem til en rodede lejlighed, hvor ting er blevet stjålet, som har været uerstattelige for vedkommende? Men ikke nok med det har indbrudstyven sluppet godt fra det. Derfor er der blevet et større behov for at tyveriforbygge sit hjem, da indbrud i hjemmet stiger med hele 47[1] procent lige nu. Med den teknologi der er i dag, kan en løsning være et X.10 sikkerhedssystem. Systemet skal programmeres med X.10 enheder til at simulere, at der er ”nogen hjemme”. Her kan der anvendes en central PC, hvor en bruger får mulighed for at vælge et scenarie, der skal simuleres i hjemmet. Det kunne være et lys, der automatisk bliver tændt i bestemte intervaller, bruger selv vælger.
Ud fra overstående problemstilling skal der designes et system, hvor en bruger vælger kommandoer på en central PC, som kommunikere med et lys og evt. dæmper lyset eller slukker lyset i det interval, bruger vælger.

Interface

[1] http://www.tekniq.dk/PresseOgNyheder/Pressemeddelelser/2012/december/Juleindbrud.aspx.

- Brugervenlig Windows prompt, hvor bruger trykker på nogle taster, vælger ekstern enhed der skal kommunikeres med og taster det tidsinterval, lyset skal være tændt i. Her skal der være mulighed for automatisk eller manuelt regulering af tiden.

Input

- Bruger scenarie – lys konfiguration.
- Serial kommunikation mellem PC og X.10 kontroller (STK 500).
- X.10 interface (AN236) tilkoblet en central PC.
- DE2 kodelås indtastning
- 230 VAC Power-line kommunikation.

Output

- 18 VAC Power-line kommunikation
- DE2 kodelås system aktivering/deaktivering
- Automatisk eller manuelt styring af lys.

Projektafgrænsning

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.

I projektet skal der kunne simuleres op til 16 enheder i 16 forskellige i huse. Men der i det følgende kun taget udgangspunkt i ét hus med 3 enheder der skal styres og monitorers af bruger på PC. Årsagen til dette er tidsmæssigt pres og forståelse for X.10 protokollen. Dog er det et godt udgangspunkt, hvor der i et fremtidigt arbejde blot kan tilføjes flere huse og enheder ved brug af samme metode, som beskrevet i denne rapport.

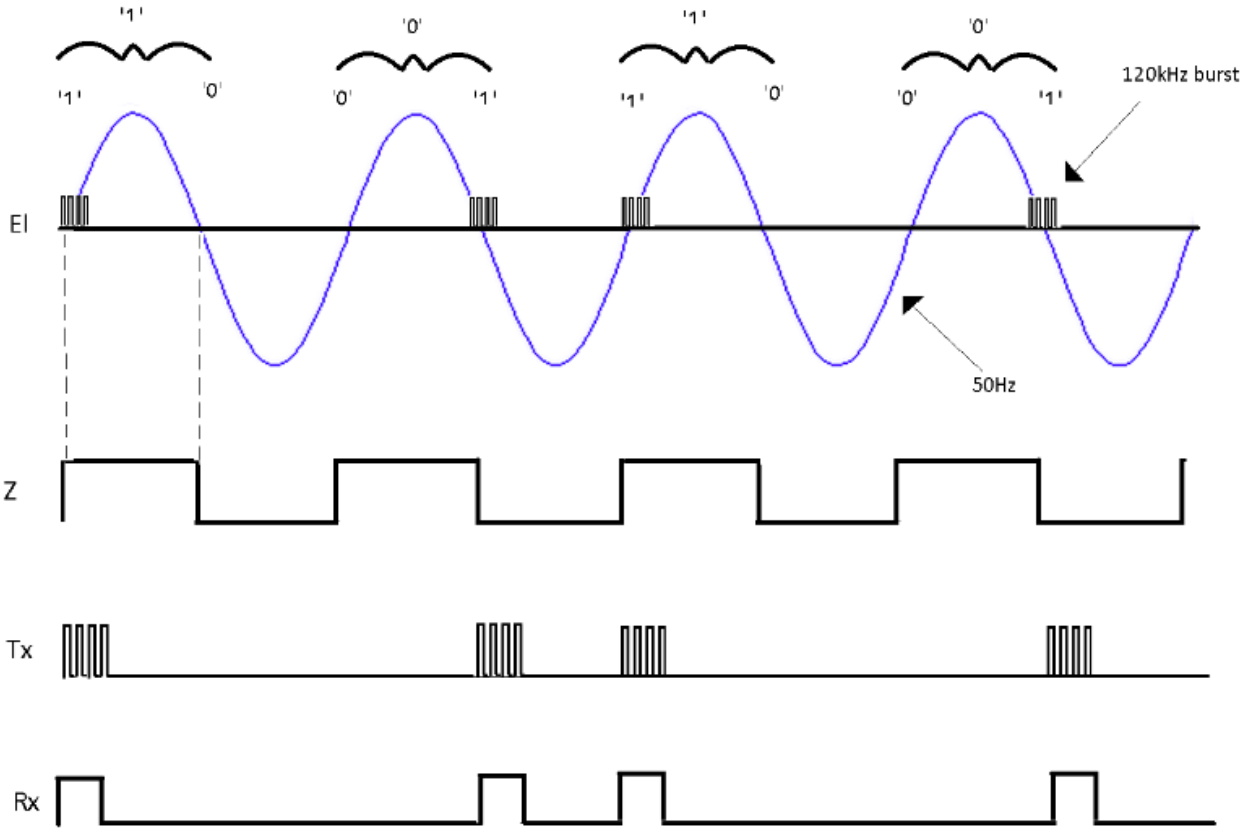
Derfor laves én modtager i HW med tilhørende SW og én sender i HW med tilhørende SW. Med én sender driver og én modtager driver gør testning nemmere at færdiggøre.

Systembeskrivelse

For at designe og implementere et system med overstående opgaveformulering, er det vigtigt at have forståelse for X10 protokollen.

Fra PC bliver der som sagt sendt kommandoer til eksterne enheder, tilkoblet et lukket 18 Volts AC lysnet. Dette lukket 18VAC lysnet består af et 50 Hz sinusformet signal. Metoden der bliver brugt vha. X10 er baseret på simple data frames med op til 8 bits (1 byte) som er forudsat for en bestemt start kode. Det komplicerede del af denne teknologi er ikke de binære data for systemet, men metoden til at sende fra en enhed(sender) til en anden enhed(modtager).

Princippet er at hver enhed skal have en ”zero-crossing” detektor som detekterer nulgennemgang af det 50 Hz sinusformet signal på lysnettet. Når der er nulgennemgang genereres et højt signal som bliver ved med at være højt indtil næste nulgennemgang, hvor signalet toggles og bliver lavt. På denne måde genereres et PWM signal på 50 % duty cycle.



Figur 1 - Billedbeskrivelse af X.10 Protokollen for systemet

Krav

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-kontroller (STK-500). X.10 kontrolleren 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) til X.10 kontrolleren således at systemet kun kan tilgås af brugeren når systemet er låst op. Nedenstående use case diagram sammenfatter aktører og use cases for systemet.

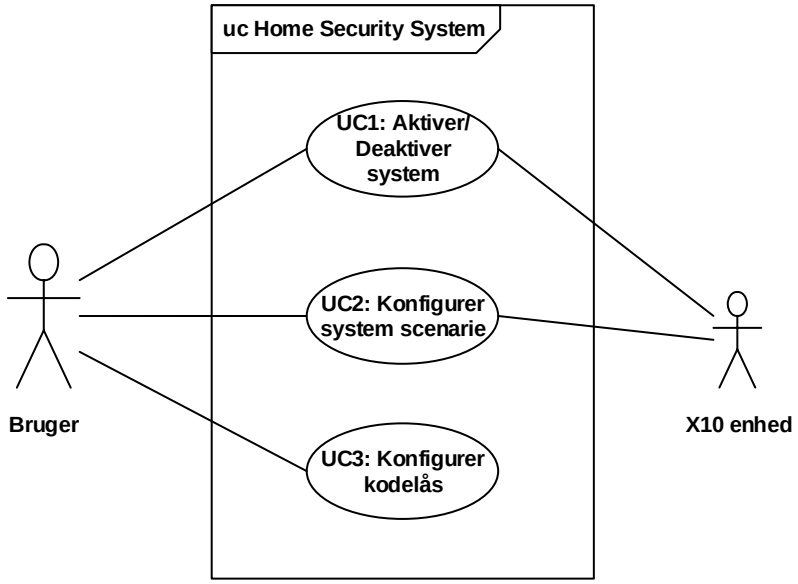


Figure 2 Use Case diagram for Home Security System

UC1 er beskrevet ved, at bruger aktiver eller deaktiver systemet via. DE2 Boardet. Herefter får bruger adgang til bruger-interface, hvilket giver bruger mulighed for at tænde eller slukke en X.10 enhed.

UC2 er beskrevet ved at bruger konfigurer system scenarie. Ved et scenarie menes der, at bruger vælger huskode og adresse på en enhed, og derefter slukker eller tænder en X.10 enhed. Dette sker i bruger-interfacet.

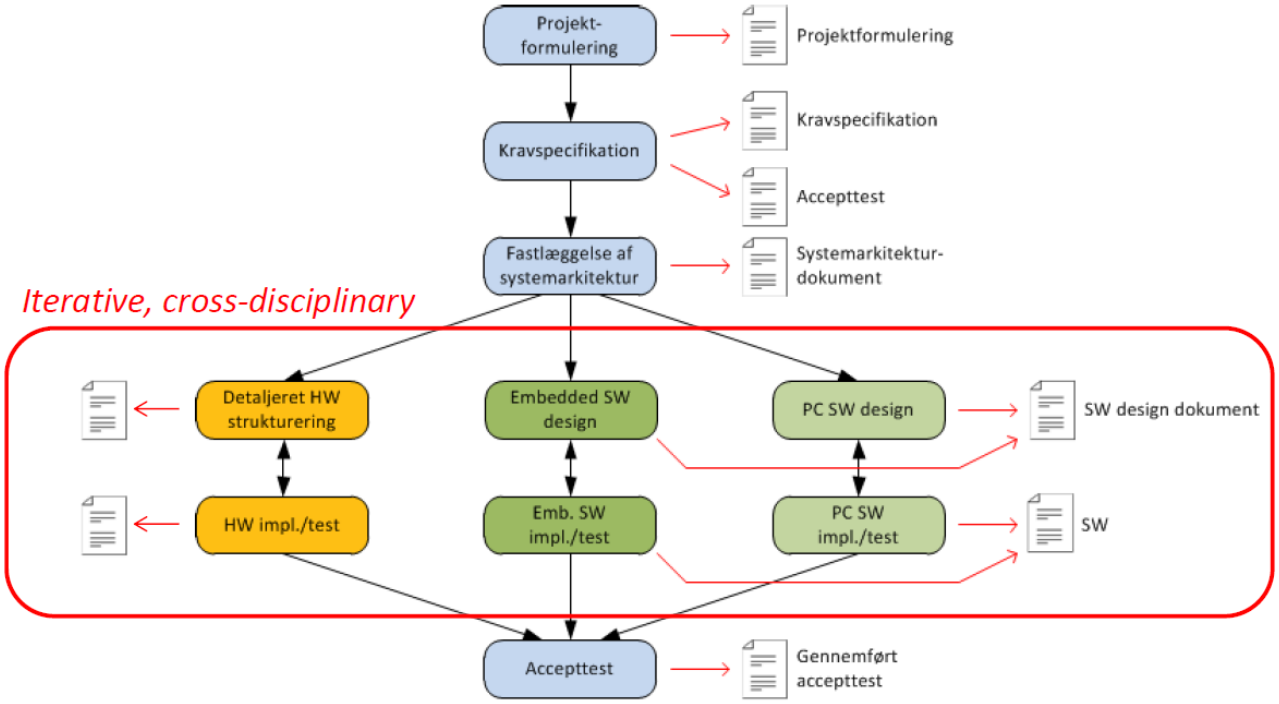
UC3 er beskrevet ved at bruger konfigurer DE2 kodelåsen på DE2 Boardet. Ved konfiguration ændrer bruger sin nuværende kode til en ny kode.

(**Der refereres til kravspecifikationen i projektdokumentationen for yderligere detaljer om hver Use Case**)

Projektbeskrivelse

Projektgennemførelse

Da projektet blev påbegyndt, blev der i gruppen enighed om at dele projektet op i faser vha. Aarhus School of Engineering proces og oprettede en tidsplan som skulle følges.



Figur 3 - Århus modellen, opdeling i faser

	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Intro til emnet	x															
Problemformulering		x	x	x	x											
Krav		x	x	x	x											
Accepttest		x	x	x	x											
Struktur						x	x	x	x	x	x					
Design HW						x	x	x	x	x	x					
Design SW						x	x	x	x	x	x					
Implementering										x	x	x	x	x	x	
Test										x	x	x	x	x	x	
Dokumentation			x	x	x	x	x	x	x	x	x	x	x	x	x	
Aflevering																31. maj

Figur 4 - tidsplan for projektet

På figur 3 kan man se Århus modellen, og hvordan den fungerer. Fællesgruppen diskuterer og laver projektformulering, kravspecifikation og systemarkitektur. Derefter deler medlemmerne sig op i sine fagspecifikke fag – arbejdsfordeleingen er vist længere nede – og laver design faserne samt implementering disse. Til sidst laves accepttesten som en samlet gruppe.

På figur 4 ses gruppens tidsplan som blev oprettet i starten af projektet. Gruppen har forsøgt at følge tidsplanen, så godt som gruppen kunne.

Gruppen har valgt at have faste dage om ugen til at lave projektet i. I projektets begyndelse blev der aftalt faste dage for møder med vejleder (Steen Krøyer) hver tirsdag for at være up-to-date, få vejledning og stille de spørgsmål, som gruppen havde. Gruppen mødtes desuden hver onsdag efter skole for at arbejde på projektet.
Gruppen har udover at bruge Århus modellen også hentet inspiration i V-modellen.

Arbejdsfordeling:

HW:

Carrier generator/detektor – JBM og YS

Zero-crossing detector - JBM

SW:

Carrier generator – DB

Carrier detector – JMH

User-interface – JRL

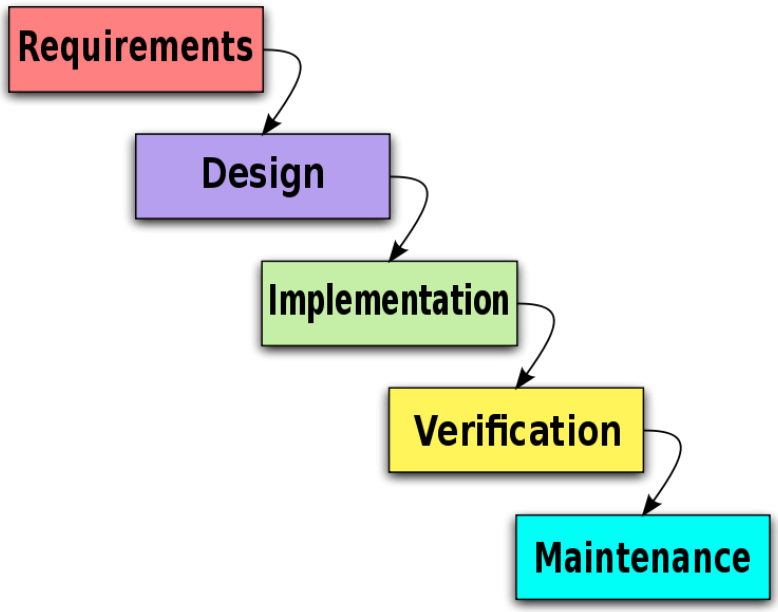
DE2kodelås – JRL og JMH

Metoder

Gruppen har ikke valgt at bruge scrum, da det er nyt og uoverskueligt. I stedet har gruppen valgt at bruge V-modellen, da gruppen er mere vant til at bruge denne model, samt at der stadig er mulighed for nemt at ændre på de forskellige dokumenter.

Gruppen har brugt dropbox som fildelingssted, da det er hurtigt tilgængeligt alle steder med adgang til internettet. Yderligere havde gruppen mulighed for at tilføje og redigere filer. Ulempen er dog at hvis to personer redigere i det samme dokument og gemmer, opstår der en konflikt, og der kan hurtigt mistes data. Derfor er det vigtigt med en åben dialog mellem alle gruppen medlemmer.

Vandfaldsmodel:

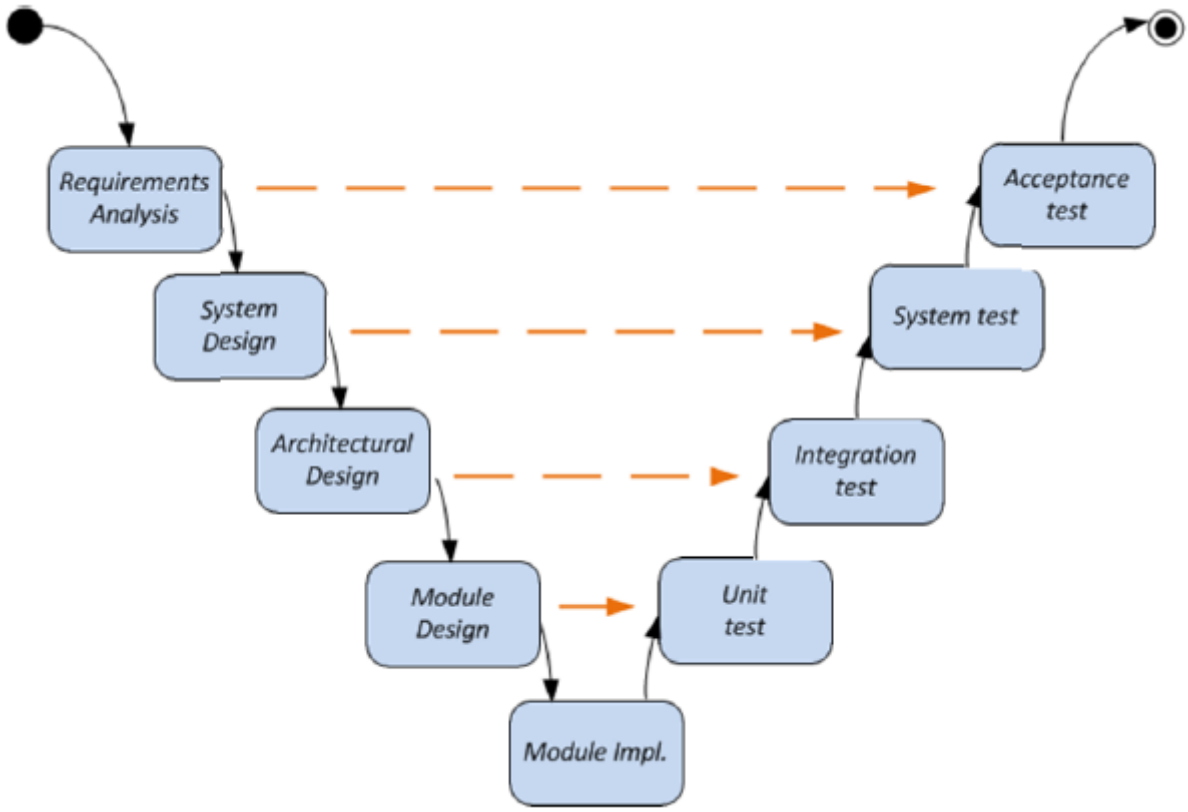


Figur 5 - Lineære proces

Denne model er simpel at følge, da det er en lineær proces, hvilket kan medføre mange fejl, da det kræver meget ekstra arbejde at gå tilbage i modellen og rette på et tidligere afsluttet emne. Derfor skal man være helt sikker på, at den aktivitet der bliver lavet nu er helt færdig før den næste igangsættelse startes.

Selvom det er nemt at lave fejl i denne model, kan den nemt laves om til en iterativ model i stedet for. Denne form for model kan være V-modellen.

V-Model:

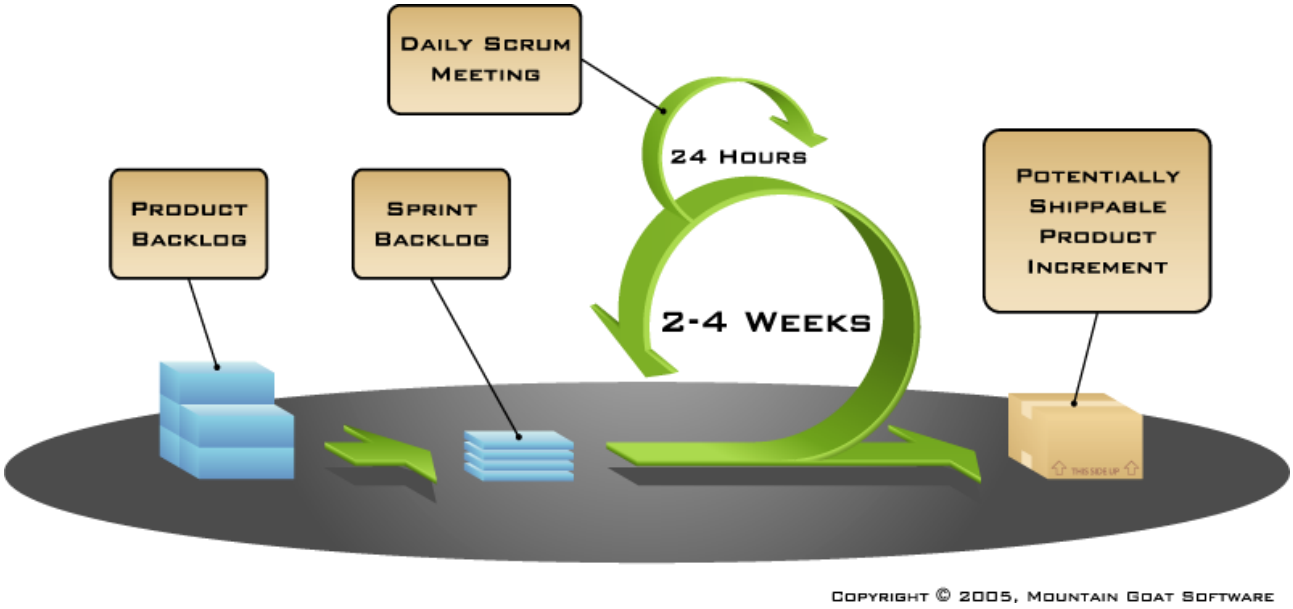


Figur 6 - V-model

Gruppen har som sagt valgt at bruge denne model, fordi den er lige til, men der er stadig mulighed for at ændre i dokumenterne, og når vi skal til at teste, kan vi teste efter bottom-up princippet.

Ved bottom-up test er det de nederste komponenter/enheder, i bdd omkring system enheder, som bliver testet først, og den proces fortsættes til man når de øverste komponenter/enheder. Fordelen ved bottom-up test er, at det er meget nemmere at finde fejl, imens når man tester det fra bunden og op.

Scrum:



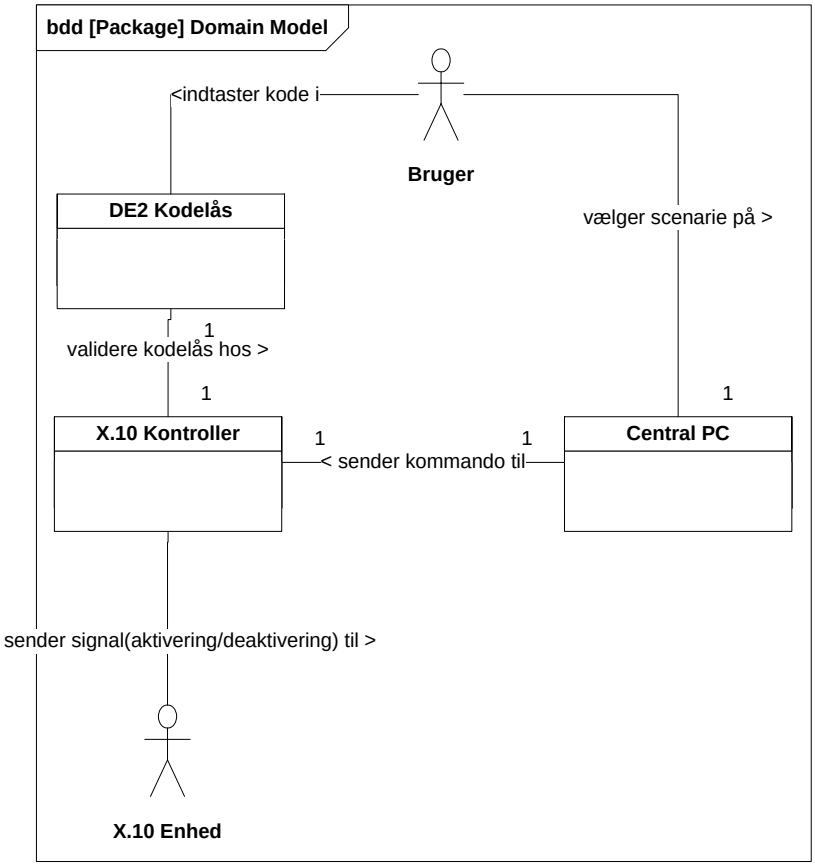
Figur 7 - Illustration af Scrum

Dette er en udviklingsmetode som ikke er en lineær proces. Dette betyder, at man ikke skal implementere de fire aktiviteter fra vandfaldsmetoden i en rækkefølge, men man kan derfor starte, hvor der ønskes og skifte til en anden aktivitet på ethvert tidspunkt. Dette gør scrum til en mere fleksibel og produktiv metode. Det er fleksibelt, idét at gruppen eller den enkelte, selv vælger hvilket område, der skal arbejdes på nu. Der holdes daglige møder for at holde styr på, hvor meget der er lavet i det hele.

Grunden til at scrum er produktivt er fordi, at der bliver kørt sprint, som typisk bliver kørt over 2-4 uger, hvor holdet tager en del fra det samlet produkt og får det færdig gjort inden der køres et nyt sprint, hvilket sprit holdet skal køre er blevet planlagt på et møde på for hånd. Hvis holdet ikke får færdiggjort et helt sprint, sendes resten tilbage til det samlede ufærdige produkt og tages op igen på et senere tidspunkt.

Specifikation og analyse

På baggrund af beskrevne metoder kan systemet opstilles i et domæne model som følgende:



Figur 8 - Domain model der beskriver analysen af systemet

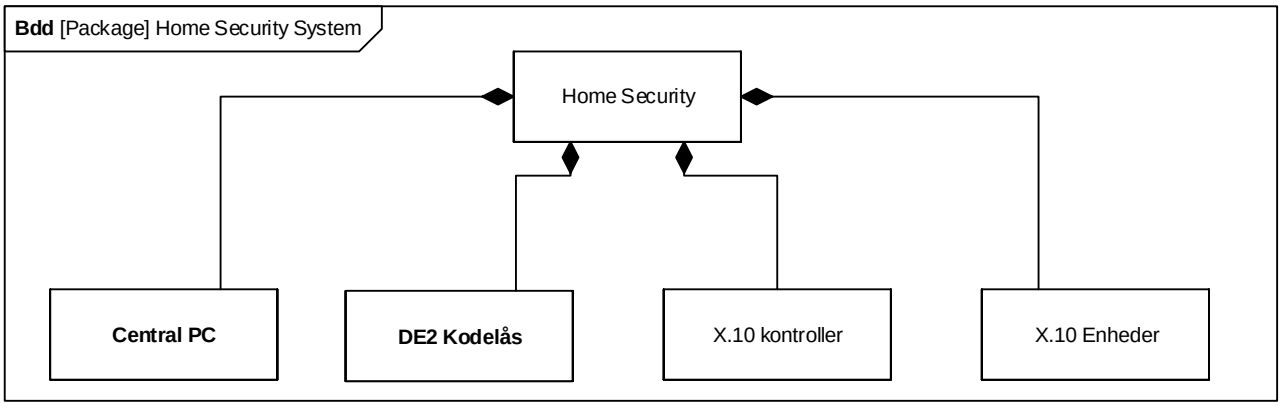
Ved at have analyseret de forskellige Use Cases er der defineret konceptuelle klasser og deres indbyrdes associationer.

Analysemetoden har været at gennemlæse hver Use Case og identificere navneord og tekstuelle beskrivelser, som havde noget med systemet at gøre. Disse kunne på punktform opskrives som kandidater til klasser. Herefter blev der vurderet og diskuteret hvorvidt kandidaten til klassen var en vigtig eller relevant klasse. Yderligere er der tilføjet multiplicity angivet med tal, som fortæller er der 1 Central PC, 1 X.10 Kontroller og 1 DE2 Kodelås.

Beskrivelsen af domænemodellen er som følgende:

Bruger er den primære aktør der igangsætter hele systemet. Brugerens kommunikation til systemets sekundære aktør, X.10 enheden, sker via Central PC. For at bruger kan foretage indstillinger og konfigurationer på Central PC, indtaster bruger kode i DE2 Kodelås som valideres hos X.10 Kontroller. Herefter vælger bruger scenarie på Central PC, som sender kommandoer til X.10 Kontroller. Herefter vil der blive sendt signal om aktivering eller deaktivering til X.10 enheden, som modtager signalet og udfører en handling alt efter, hvad bruger har valgt af scenarie.

Systemarkitektur



Figur 9 - Bdd for hele system

I den overordnet bdd for systemet ses det, at der til hele systemet skal bruges fire forskellige mindre systemer, som igen består at nogle forskellige dele. For vores system skal gruppen bruge noget der kan sende en kommando. Her bruges der en del der kan modtage den sendte kommando og en enhed der reagere på den modtagende kommando, samt et del system der kan låse resten af systemet, hvis der ikke bliver trykket en korrekt kode. Alle disse dele er viderebeskrevet i resten af dokumentet, hvor at den centrale PC er bruger-interface, DE2 kodelåsen sker på et DE2 board, X.10 kontrolleren er en sender og X.10 enhed er modtager.

Systemkomponenter

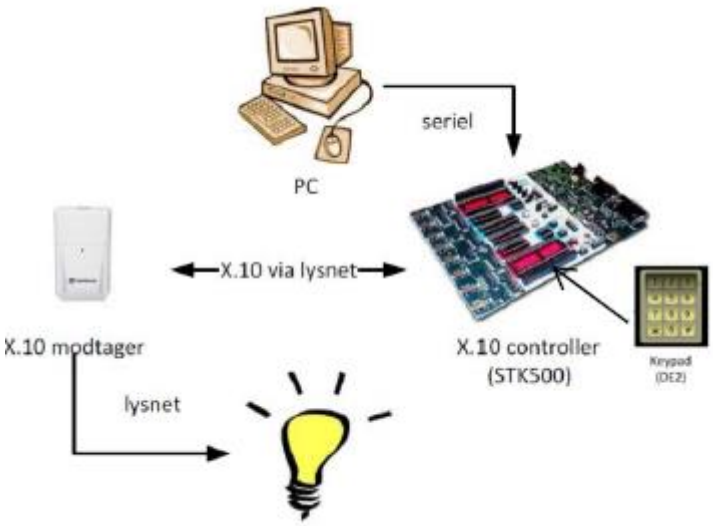
Der er gennemført en analyse af kravene som givet i kravspecifikationen, og på baggrund af disse er der truffet følgende beslutninger om systemets komponenter og deres placering:

- Brugers tilgang til systemet er en PC, der kommunikerer med X.10 kontroller bestående af en 32 bit Micro controller over et elektrisk interface ”AN236, som link mellem PC og X.10 nettet. På PC 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. Bruger skal have indtastet en rigtig kode kombination, før bruger kan ændre i X.10 kontrollers forskellige værdier.

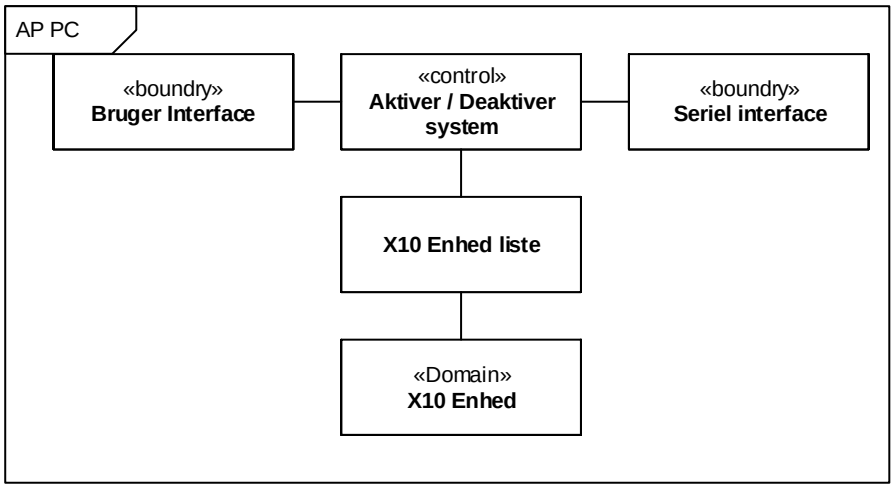
- *STK500 KIT*
  Programmeres i C og bruges som X.10 kontroller, hvor systemet videre sender et bit mønstre, som kommer fra bruger interface, til en X.10 enhed.

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



Figur 10 - Illustration af system komponenterne og deres indbyrdes kommunikation

System Applikations model for UC1: Aktiver/Deaktiver system



Figur 11 - Applikationsmodel for UC1: Aktiver/Deaktiver system

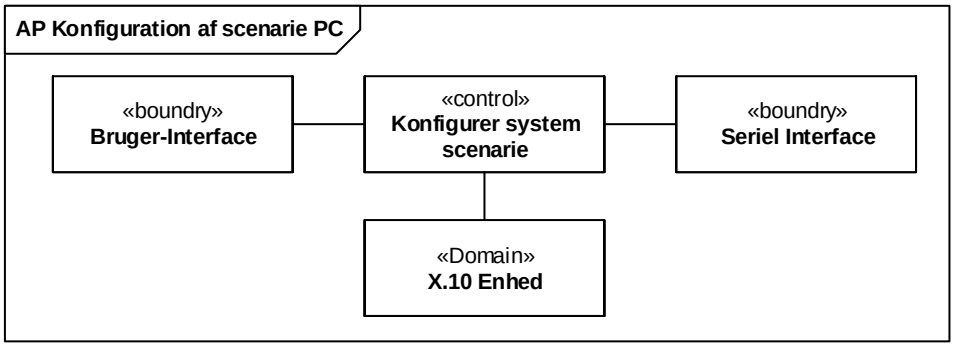
Beskrivelse:

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

System Applikations model for UC2: Konfigurer system scenarie

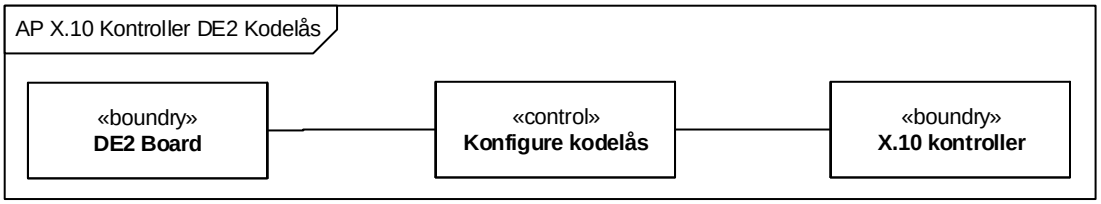


Figur 12 - Applikationsmodel for UC2: Konfigurer system scenarie

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.

System Applikations model for UC3: Konfigurer kodelås



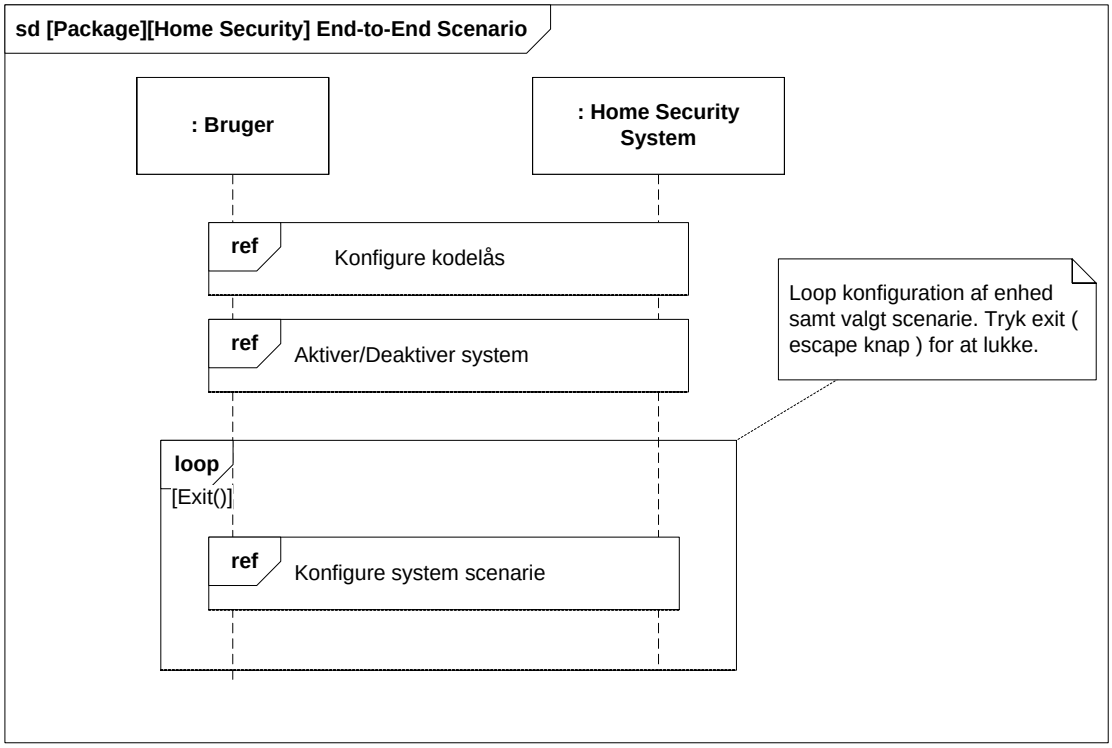
Figur 13 - Applikationsmodel for UC3: Konfigurer kodelås

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.

Sekvensdiagram Home Security System End-to-End Scenario



Figur 14 - Sekvensdiagram for hele System, der viser End-to-End Scenario

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.

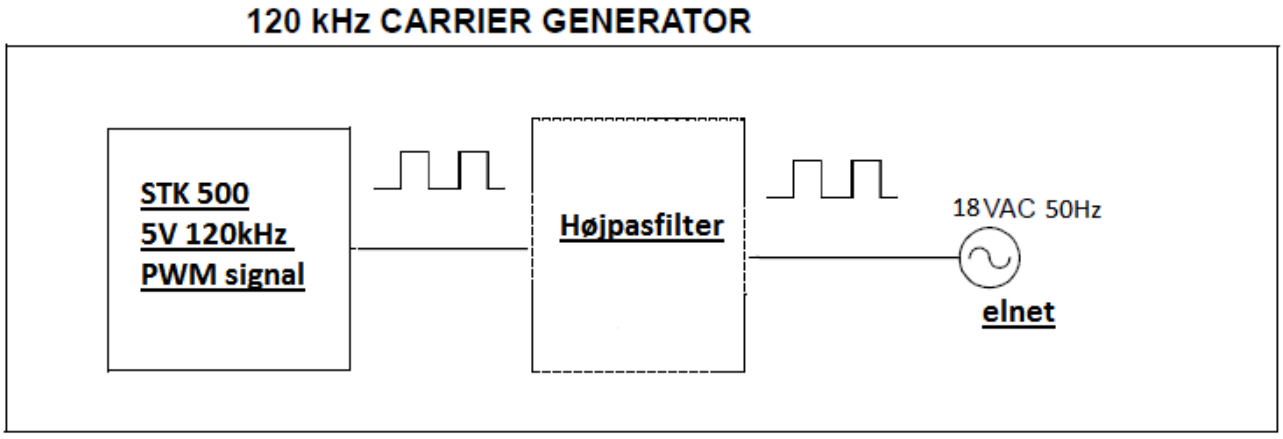
Design(HW)

Initial: YS & JBM

Sender

Da vi i forvejen havde god kenskab til stk500 og mikroprocessor ATmega32 har vi valgt at benytte disse til vores X10 controller.

Senderen skal transmittere de signaler som STK500 sender, og videreføre dem til el nettet. Senderen har også den funktion at dæmpe 50 Hz signalet og lade 120 kHz signal komme igennem, hvilket kræver et filter. Vi overvejede i starten at lave et båndpas-filter, men da støjen var minimal besluttede vi i stedet at lave et passivt højpas-filter som var mere simpelt og tog kortere tid samt gav samme resultat. Et passivt højpas-filter er også billigere at realisere.



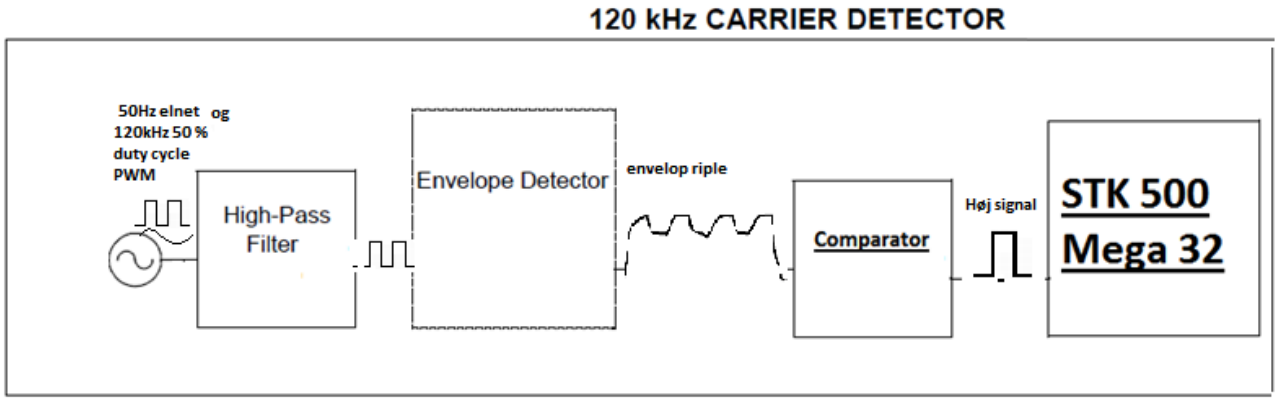
Figur 15:Sender system beskrivelse

Derudover har vi valgt at benytte en transistor til at generere 120 kHz PWM signal, hvor at et PWM signal fra stk500 sættes på basen af transistoren, og samtidigt sættes VCC (5 V) på ”Collectoren” som når transistoren slår fra og til laver de 120 kHz signal. En transistor er valgt for at stk500 ikke skal trække systemet og bruge for meget strøm.

Modtager

Der er taget udgangspunkt i X10 protokollen (Figur 4: 120kHz Carrier Detektor side 4), hvor i stedet for den opstilling der er i protokollen er der benyttet nedenstående løsning:

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



Figur 16: Modtager system beskrivelse

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

Angående højpas-filteret er det blevet realiseret som et RC led. Altså et passivt filter der kan dæmpe de 50 Hz fra el nettet og lade 120 kHz passerer. Det er valgt på baggrund af at det var meget simplere end forslaget i protokollen og at funktionaliteten var den samme når man tilføjer Comparatoren og laver lidt om på ”Envelope Detektoren”.

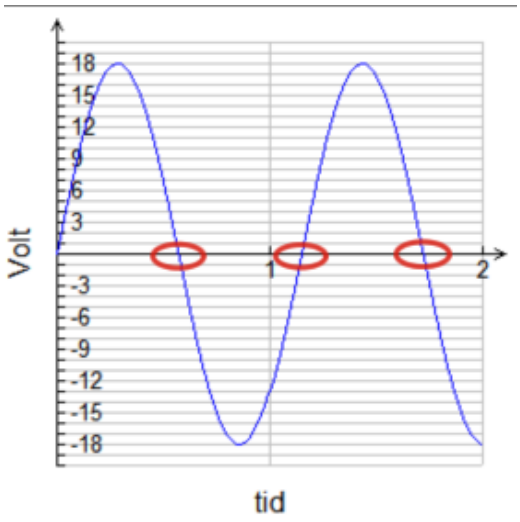
Envelope Detektoren er realiseret med en diode i serie med en kondensator og en modstand i parallel. De ”ripples” der kom på outputtet fra Envelope detektoren kunne vi se som mindre betydelige da Comparatoren lavede det om til henholdsvis høj (5 V) og lav (0 V) signal.

Comparatoren er lavet ud fra en Op Amp og en spændingsdeler til referencespænding samt VCC (5 V) til selve forsyningen på Op Amp, hvor at VSS er sat til ”ground” (**Reference til Datasheet for MCP6002 – OP AMP**).

Zero-Crossings detektor

Initial: 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).



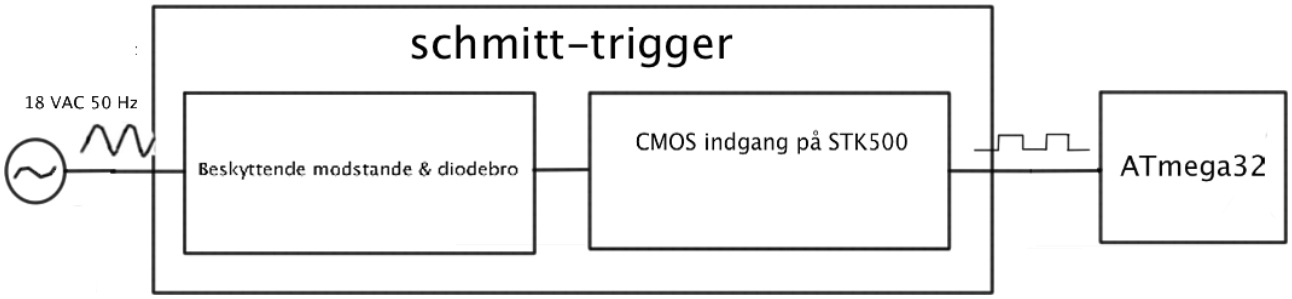
Figur 17: Periodediagram 18sin(314t).

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 output signalet fra detektoren til stk500 skifter fra 1 til 0 V.

Den første tanke var at lave en Comparator til at tjekke niveauerne på el nettet for ”zero crossing” eller muligvis en IC med en Schmitt-trigger.

Herefter blev det endeligt besluttet at bruge de CMOS indgange der allerede var på stk500 og dermed benytte CMOS niveauer(thresholds) til detektere ”zero crossing” og samtidig bruge modstande til at begrænse strømmen fra el nettet samt en diodebro så CMOS ikke blev påtrykket negative spændinger.



Figur 18:Nulpunktsdetektor system beskrivelse

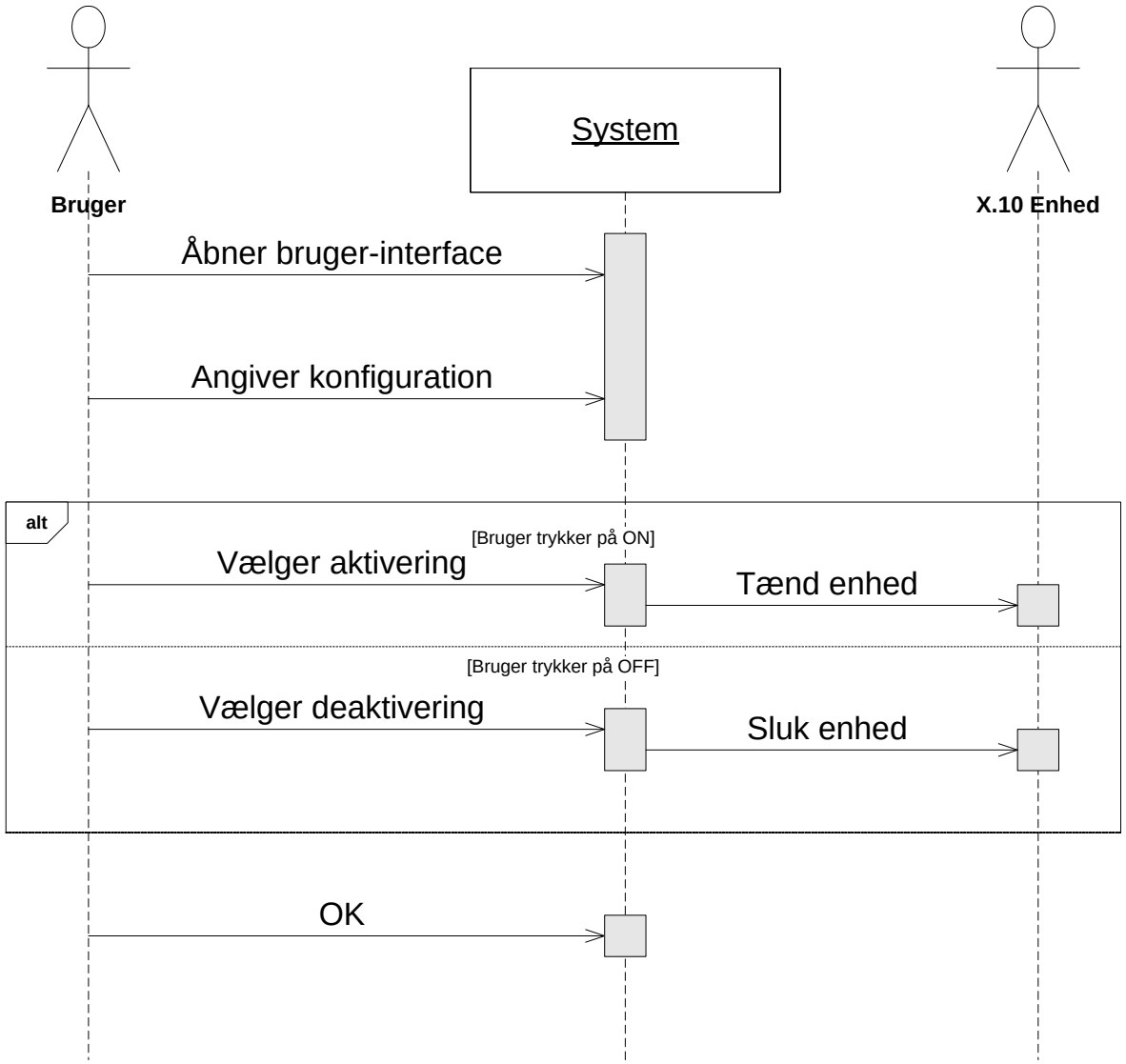
Design(SW)

Initial: DB, JMH, JRL

Beskrivelse af designsprocessen og overvejelser.

For at kunne give bruger mulighed for at styre og monitorer enheder over lysnettet, skal der i SW laves et bruger-interface, hvor bruger kan vælge enheder og bestemme scenarier, dvs. slukke eller tænde for et lys i et hus.

Diagrammet nedenunder her illustrerer sekvensen for brugerens interaktion med systemet for hovedscenariet, som knytter sig til de enkelte Use Cases beskrevet i projektdokumentationen. Den interaktion, der findes på systemsekvensdiagrammet, kan også genfindes på de detaljerede sekvensdiagrammer på projektdokumentationen.



Figur 19 - Sekvensdiagram for hele Home Security System.

Bruger åbner bruger-interface og angiver konfiguration. Ved konfiguration forstås det, at bruger skriver konkrete inputs til systemet, som tager sig af validering og eventuelle fejlbeskeder (ikke beskrevet her). Bruger kan herefter vælge at aktivere eller deaktivere systemet, som hhv. tænder eller slukker en X.10 enhed. I overstående diagram er der taget udgangspunkt i et succes scenarie, og der henvises til projektdokumentation for alternative/undtagelser i systemet.

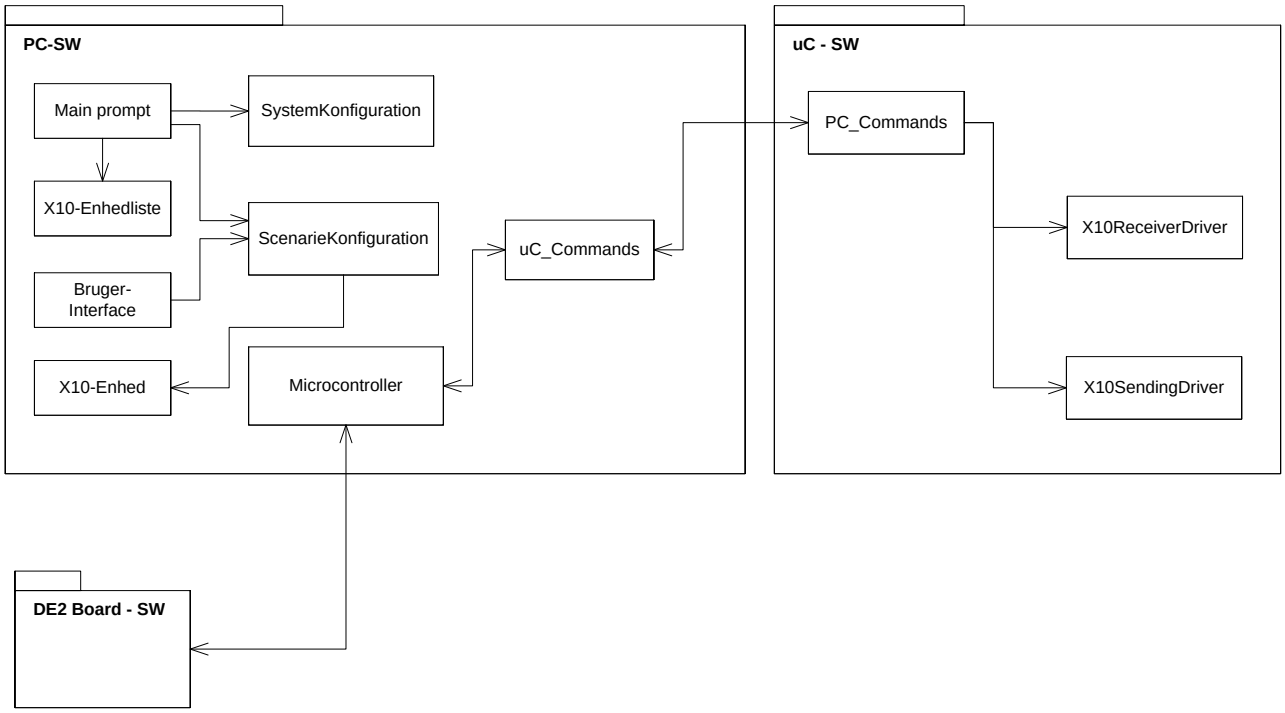
Overordnet klassediagram

Initial: DB

Bruger-interfacet består af et Windows prompt, som kan køre på et almindelig Windows styresystem på en PC. På PC'en er der yderligere SW der via. seriel kommunikation kommunikerer med SW på microcontroller. Microcontroller har software der kan sende sende og modtage data fra bruger. Yderligere er der SW på et DE2 Board der kommunikerer med microcontroller og kan aktivere eller deaktivere systemet afhængig af, om bruger indtaster korrekt kode eller forkert kode på DE2 Board.

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 20 - Software pakker mellem PC og uC

Dvs. PC softwaren består af en pakke af andre software ”klasser” og deres indbyrdes kommunikation. PC softwaren kommunikerer med en anden SW pakke på microcontroller, som består af en modtager driver og en sender driver. Disse drivere er vigtigt for systemet, da de sørger for at sende data bits fra brugerens side og sørger for at X.10 enhederne kan modtage disse bits i korrekt format i form af ’1’ og ’0’ som hhv. kan tænde eller slukke en lampe. Dette er beskrevet vha. X.10 Protokollen, hvor der henvises til projektdokumentationen for detaljeret design.

Udviklingsværktøjer

- 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

Erfaringer har vist, at det ikke er alt multisim f.eks. kan simulere og at man skal forholde sig kritisk under simulationer.

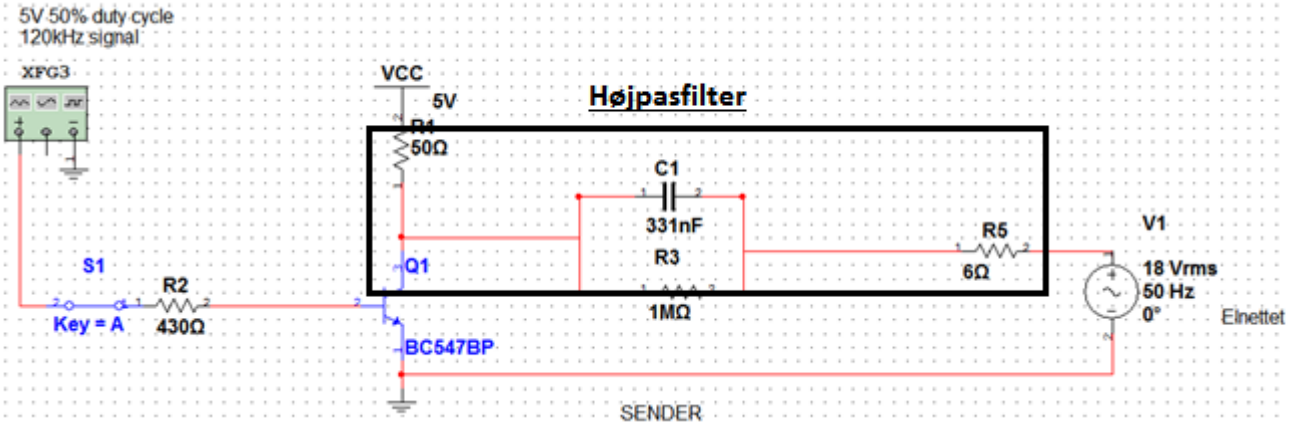
Implementering(HW)

Initial: YS & JBM

Sender

Vi har valgt at lave 120kHz 50% dutycycle signal fra mikroprocessoren da det er en hurtig, billig og præcis løsning som kræver meget lidt programmering.

Til implementering er der taget udgangspunkt i X10 protokollen (Figur 5 under Carrier Generator side 5). ATmega32 mikroprocessoren bruges i stedet for PIC16F87XA til at generere 120kHz PWM signal.



Figur 21: kredsløbet for senderen

højpas-filteret er implementeret sammen med en lille modstand(6Ω), hvilket er en formodet impedans fra el nettet, som blev angivet i en lektion omkring projektet.

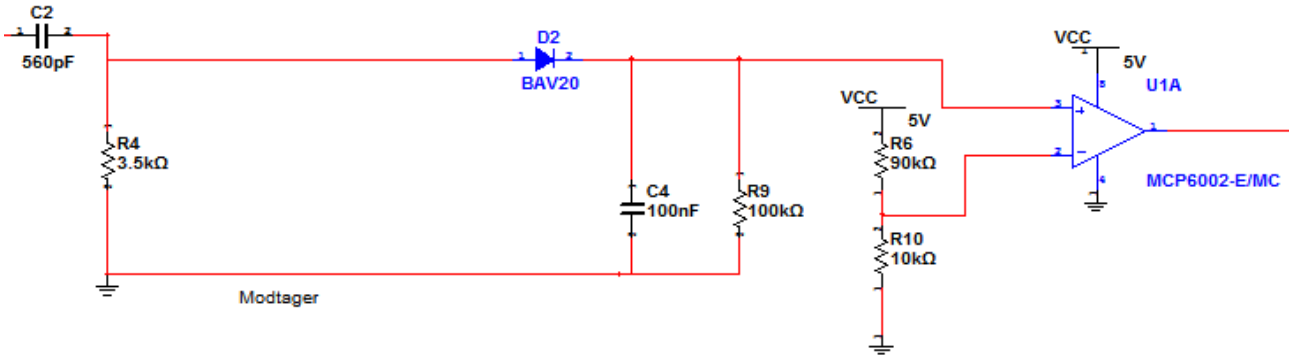
Modtager

Højpas filteret er bestemt ud fra en kondensator der fandtes i lab og at 120 kHz skal kunne passere.

Envelope Detektoren er implementeret ud fra en stor modstand som kunne begrænse strømmen og derved begrænse spild. Ydermere vælges en kondensator igen som findes i lab.

Comparatoren er valgt ud fra det udvalg af OP AMP der var til rådighed og hvilken en af dem der passede bedst til opgaven.

Herunder ses det endelige resultat af implementeringen af modtageren(Carrier Detektor):

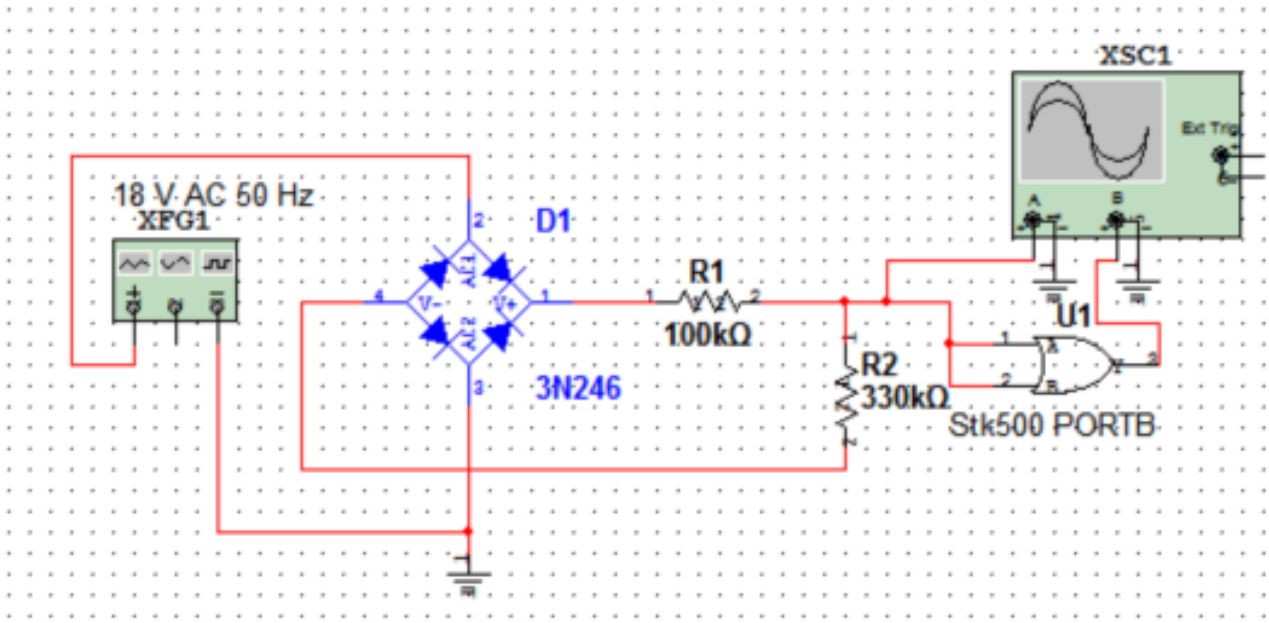


Figur 22: kredsløbet for senderen

Zero-Crossings detektor

Initial: JBM

Til implementering er der taget udgangspunkt i X10 protokollen. I stedet for PIC16F87XA bruger vi STK 500. Vi har beholdt modstanden som det havde brugt i X10 protokollen, men bare med en relativ stor modstand for at tilpasse vores kredsløb til STK500. Vores færdige løsning blev således at vi brugt endnu en stor modstand til og en diodebro til at ensrette strømmen. Den endelig kredsløb kan ses nedenunder og vi bruger OR-gaten til at simulerer CMOS indgang på STK500.



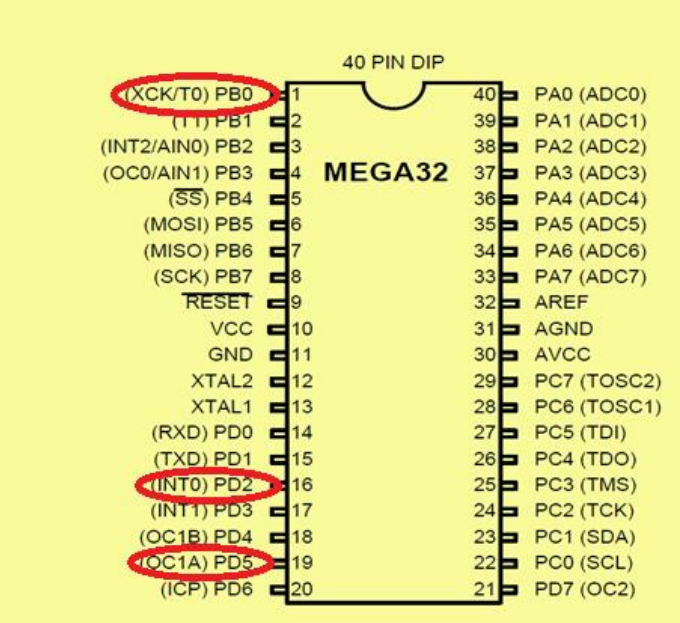
Figur 23: kredsløb for Nulpunktsdetektor

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 overlegnes 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 24 - Anvendte Porte ifm. X10SenderDriver

For at generere et 120 KHz burst, der varer 1 ms, initialiseres en timer0 og en timer1.

(**Der refereres til projektdokumentation for yderligere detaljer for implementering af timerne og beregninger for 120 KHz, der skrives til OCR1A benet, som vidst på figur 1**)

I det følgende beskrives væsentlige kodestumper for interfacet for X10Sender driveren

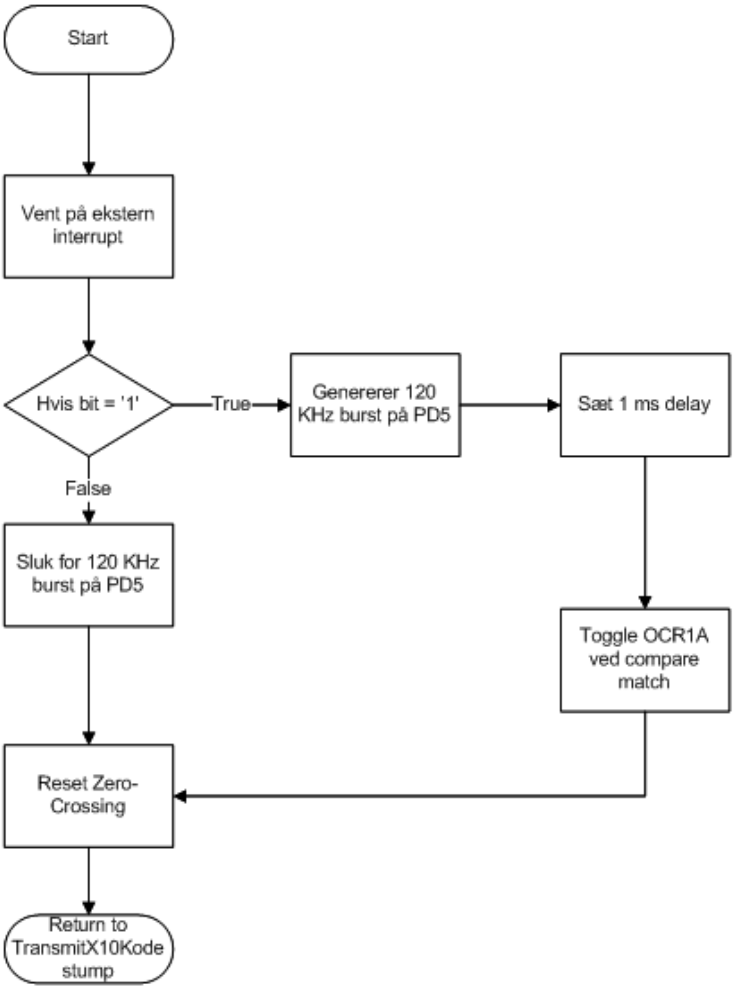
(Der henvises til Projekt Dokumentation for klassebeskrivelse af de enkelte funktioner)

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.

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

Figur 25 - Headerfil/Prototyper for XSender Driveren.

Den meget generelle proces af driveren ses af følgende flowchart



Figur 26 - Generelle flowchart for X10Sender driver

Flowcharten på figur 27 beskriver det generelle forløb af X10 Senderen. Der afventes et interrupt, når der har været nulgennemgang. 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 27 - 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 28 - Initialisering af globale arrays og pointers

Hvor eksempelvis HOUSEA[2] er predefineret som

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

Disse globale pointers af typen char sættes til at pege på arrayene i funktionen:

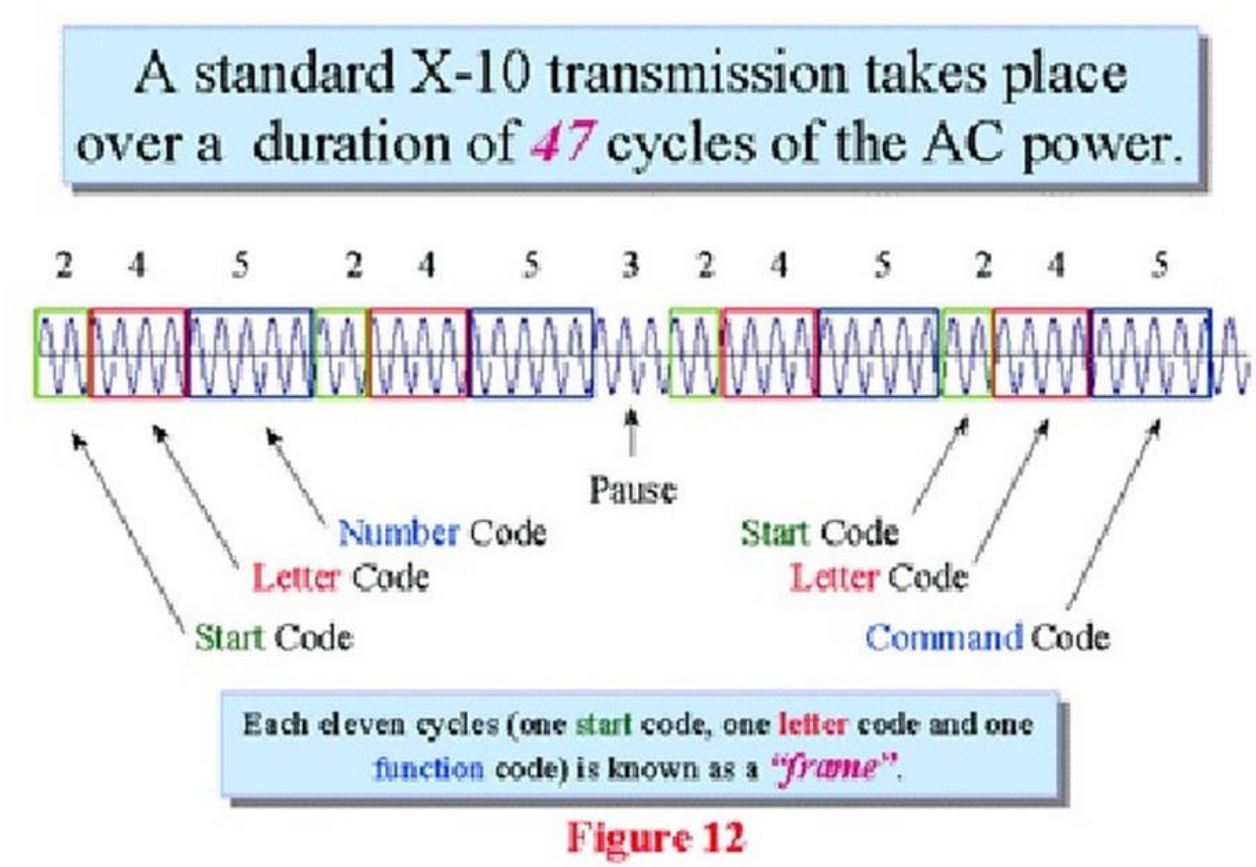
[2] Se AN236 Applikationsnote for X10 Protokoller s. 13 Tabel A-1 og Tabel A-2 for hvordan X10 beskeder ses som en huskode, adresse eller kommando.

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

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

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

Figur 29 - Udsnit af at sende en Frame

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[3]:

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



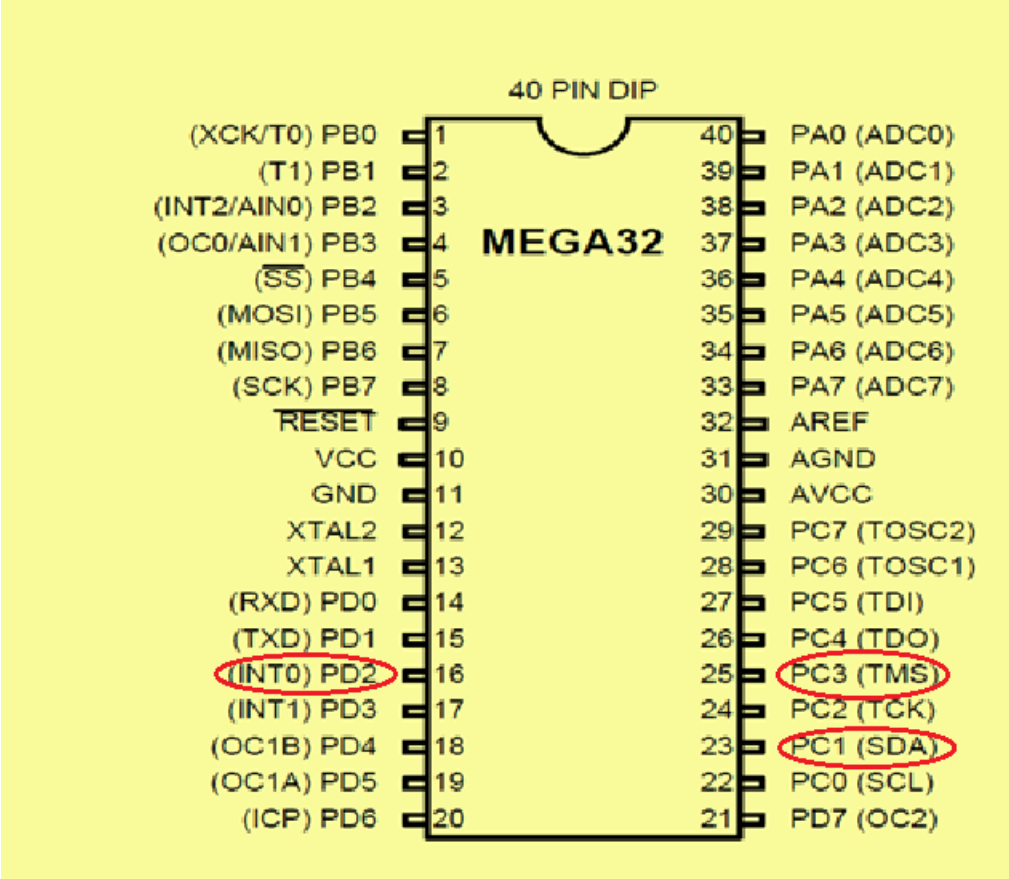
Figur 30 - X10Sender illustration

Figur 31 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.

X10ModtagerDriver

Initial: JMH

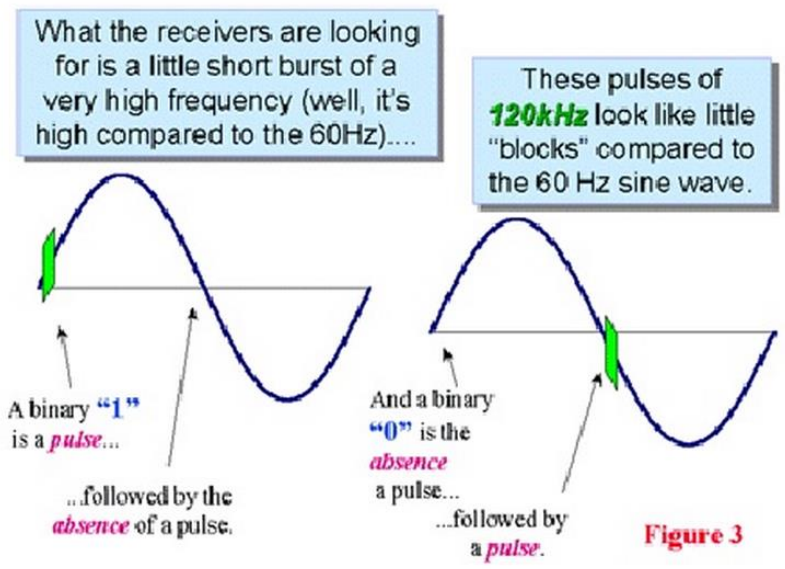
Carrier detektorens funktion er at opfange de 120kHz bursts ved zero-crossing som bliver sendt fra Carrier generatoren og derefter sende signalet videre til enheden/erne indeholdende de kodestumper man forventer.



Figur 31 - Anvendte porte til X10ModtagerDriver

På Figur 32 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.

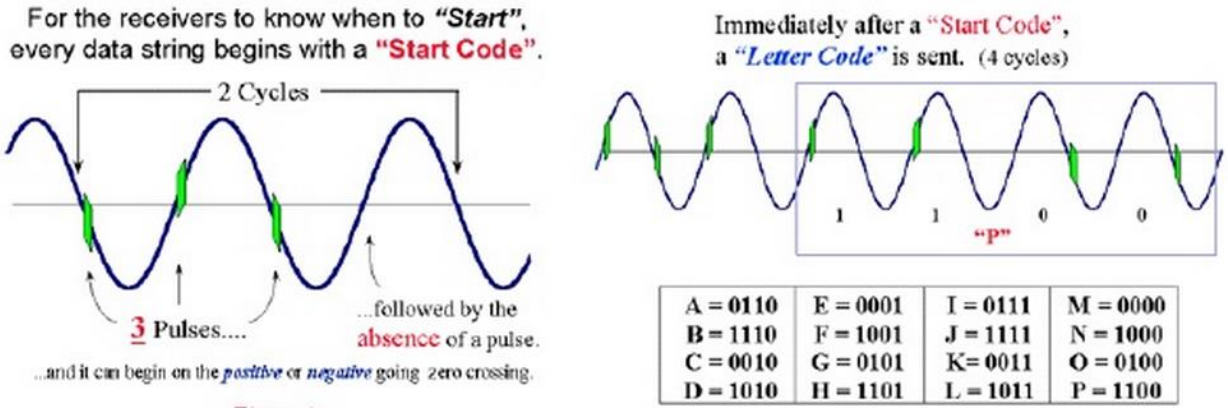
For at opfange de kodestumper sendt fra Carrier generatoren er man nødt til at ”lytte” til zero-crossing detektoren og detekterer om det er et binært 1 eller 0 som er repræsenteret i 120kHz burstet. Dette gøres ved hjælp af funktionen X10ModtagerBit, der måler hver zero-crossing 20 gange og returnerer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



Figur 32 – X10Modtager detektion af binært "1" og "0"

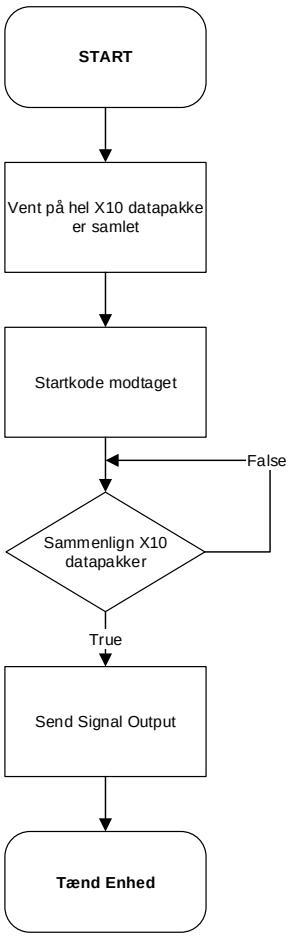
```
void X10ModtagerPakke(char* DataPakkeModtaget)
{
    int i;

    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 33 – Funktion for modtagning af hel datapakke



Figur 34 – Modtagelse af START kommando

Figur 34 viser hvordan disse 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 36 - General flowchart af X10ModtagerDriver

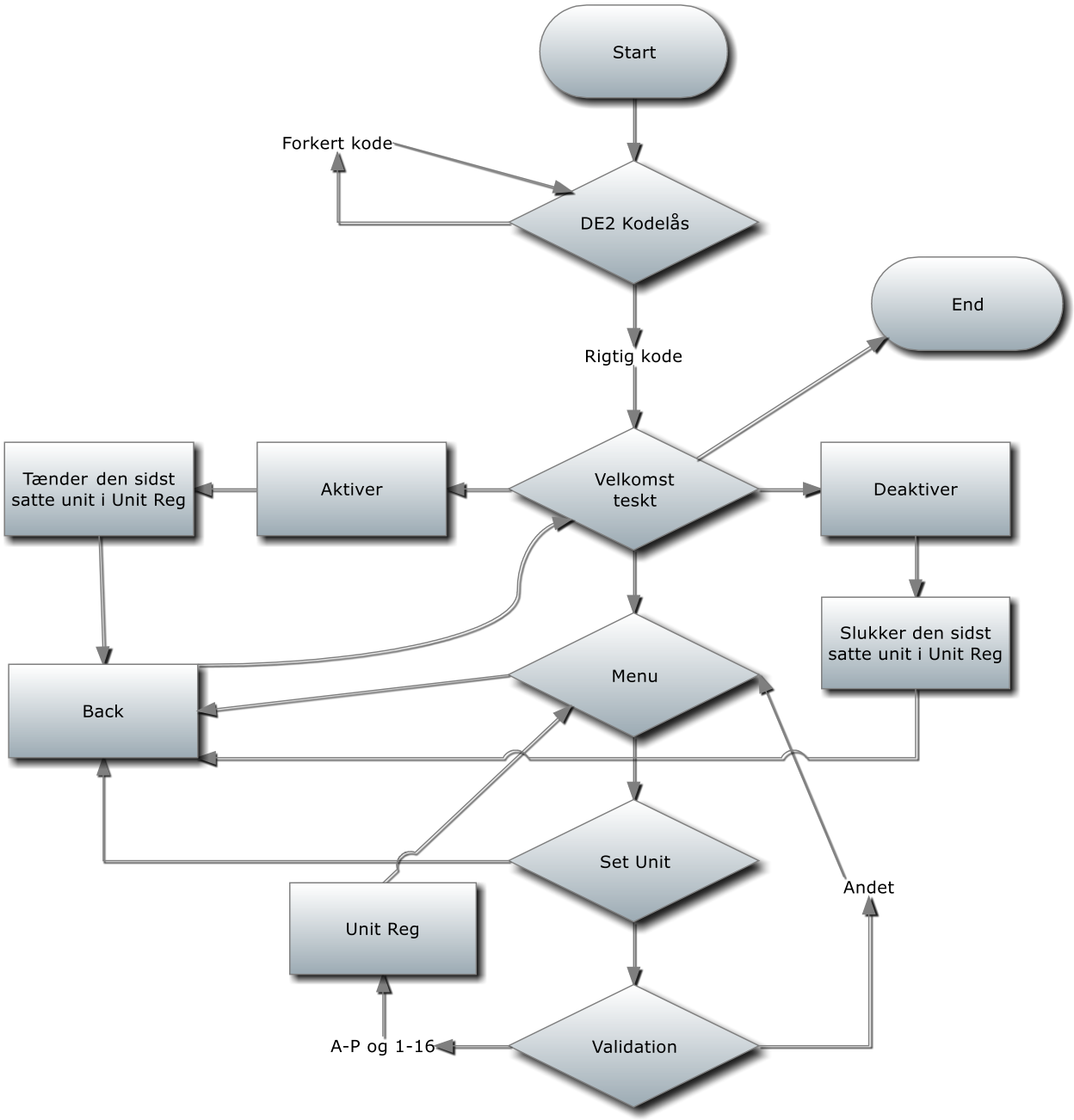
```
unsigned char X10SamligPakke(char* DataPakkeModtaget, char* DataPakkeSamlig)
{
    char samlig = 1;

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

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

Figur 35 – Funktion til at sammenligne to datapakke

Figur 35 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 efterfølgt af 22*2 bits for command koden.
Figur 37 viser et general flowchart af X10ModtagerDriveren, hvor en hel datapakke samles, sammenlignes og sendes til en enhed.

Bruger-Interface

Initial: JRL



Figur 37 Flowchart brugerinterface

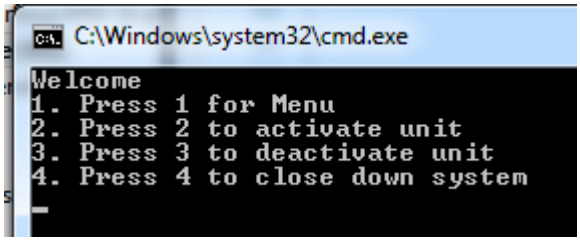
Vores bruger interface er lavet som en konsol applikation hvor brugeren her har mulighed for at aktivere de forskellige enheder som deaktivere dem igen, samt at brugeren selv kan vælge hvilken enhed han vil gøre dette ved. Figuren ovenover beskriver de forskellige muligheder der er omkring bruger interface for at give et større overblik omkring hvordan bruger interface Main program fungere. Blokken der hedder DE2

Kodelås er hardware der bestemmer om brugeren kan få lov til at gøre noget som helst eller ej, dette bliver beskrevet under DE2 kodelås

Konsol Applikation

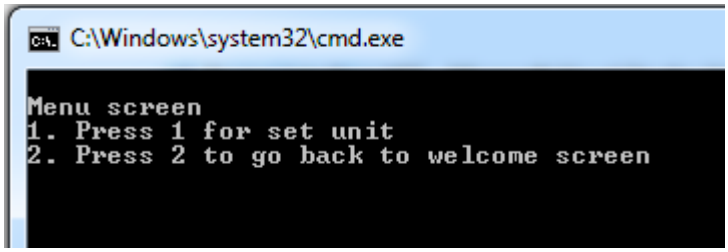
Initial: JRL

Herunder er der beskrevet med billeder om hvordan bruger interface reagere alt efter brugeren indtastning



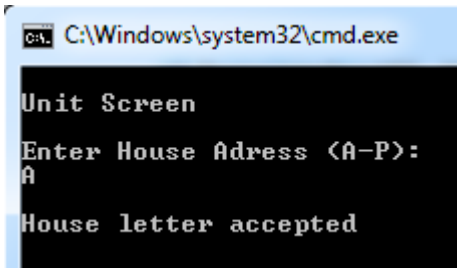
Figur 38 Hovedskærm

Her er brugerens valgmuligheder efter han har tastet koden rigtig på DE2 låsen.



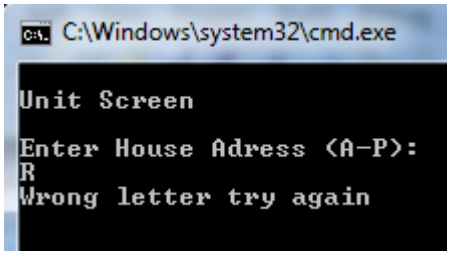
Figur 39 Menu skærm

Efter bruger har trykket på 1 i den forgående skærm, har brugeren nu to nye valgmuligheder, vælge hvilken enhed han vil bruge eller gå tilbage til hovedskærmen.



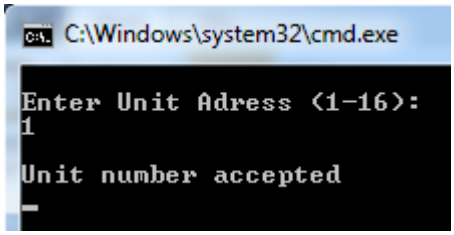
Figur 40 Unit skærm rigtig bogstav indtastning

Hvis brugeren i forgående skærm valgte 1 igen, kommer denne skærm op hvor der kan vælges hvilket hus eller gruppe der skal bruges en enhed fra. På dette billede er der valgt hus A som er et gyldigt input og bruger kan herefter få lov til at gå videre.



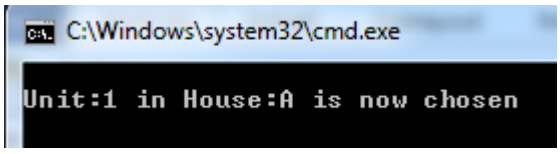
Figur 41 Unit skærm forkert bogstav indtastning

Hvis brugeren ikke indtastede et gyldigt input, bliver brugeren bedt om at prøve igen og bliver sendt tilbage til velkomst skærmen.



Figur 42 Unit skærm rigtigt nummer indtastning

Hvis brugeren tidligere indtastede et gyldigt hus input tidligere, for brugeren nu mulighed for at vælge hvilken enhed, der skal aktiveres eller deaktiveres. Hvis der her igen bliver indtastet et ugyldigt input bliver brugeren igen sendt tilbage til velkomst skærmen.



Figur 43 Konfirmation på valgt unit

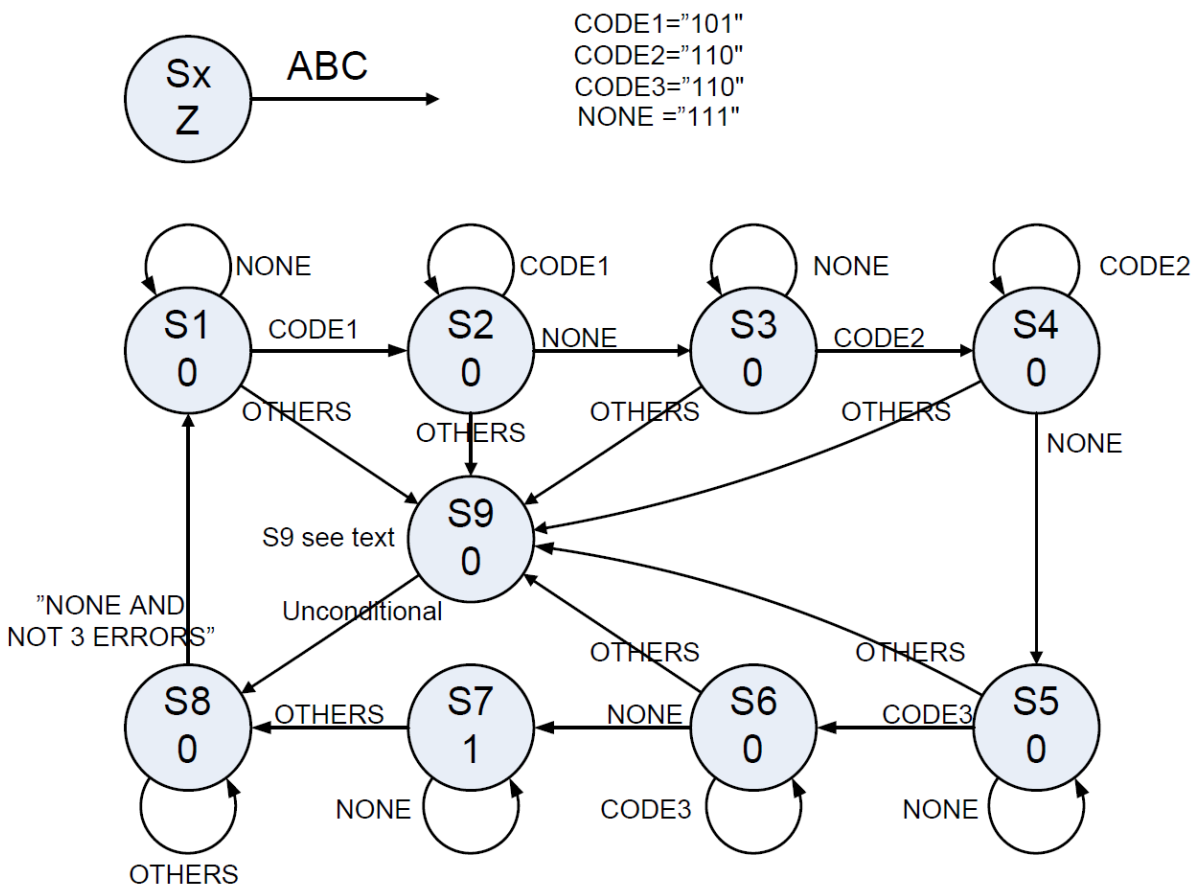
Hvis både hus og enheds input er gyldige, får brugeren en bekræftelse tilbage og bliver herefter sendt tilbage velkomst skærmen, hvor brugeren nu kan vælge om han vil aktivere eller deaktivere den nyeste valgte enhed eller vælge en ny enhed.

Ifølge X.10 protokollen kan der vælges mellem 16 forskellige huse/grupper og i hver hus/gruppe er der 16 enheder. I vores projekt har vi valgt at bruge et hus/gruppe med tre lamper, dette har vi gjort fordi om vi viser at vi kan bruge alle enheder i forhold til X.10 protokollen eller bare nogle få er det samme, da det handler om at vi skal kunne vise hvordan det kan bruges og hvis det skulle være kan vi altid udbygge vores system.

DE2 Kodelås

Initial: JRL & JMH

Kodelåsen er bestående af et Altera DE2 board som er hardware delen og en VHDL kode som er software delen, begge dele kan findes under bilag i projekt dokumentationen.



Figur 44 State diagram kodelås

Ovenover er vist et diagram over hvordan kodelåsen reagere alt efter hvilke taster brugeren trykker på.

S1: Start state, venter på bruger trykker på en knap.

S2: Brugeren holder en knap nede, venter på bruger slipper.

S3: Brugeren har sluppet en knap, venter på nyt tryk fra bruger.

S4-6: Er det samme som S2 og S3, men nye tryk fra bruger.

I vores kode har vi sat den til at den rigtige kode er tast 2-2-1 og hvis dette bliver trykket er systemet i S7 og Z signalet bliver sat til 1. Dette signal skal bruges til at kunne låse eller låse op for bruger interface.

Hvis der bliver tastet forkert mere end tre gange(S9 tæller hvor mange gange der er trykket forkert) går systemet i S8 og låser systemet og kan kun låses op igen ved en restart.

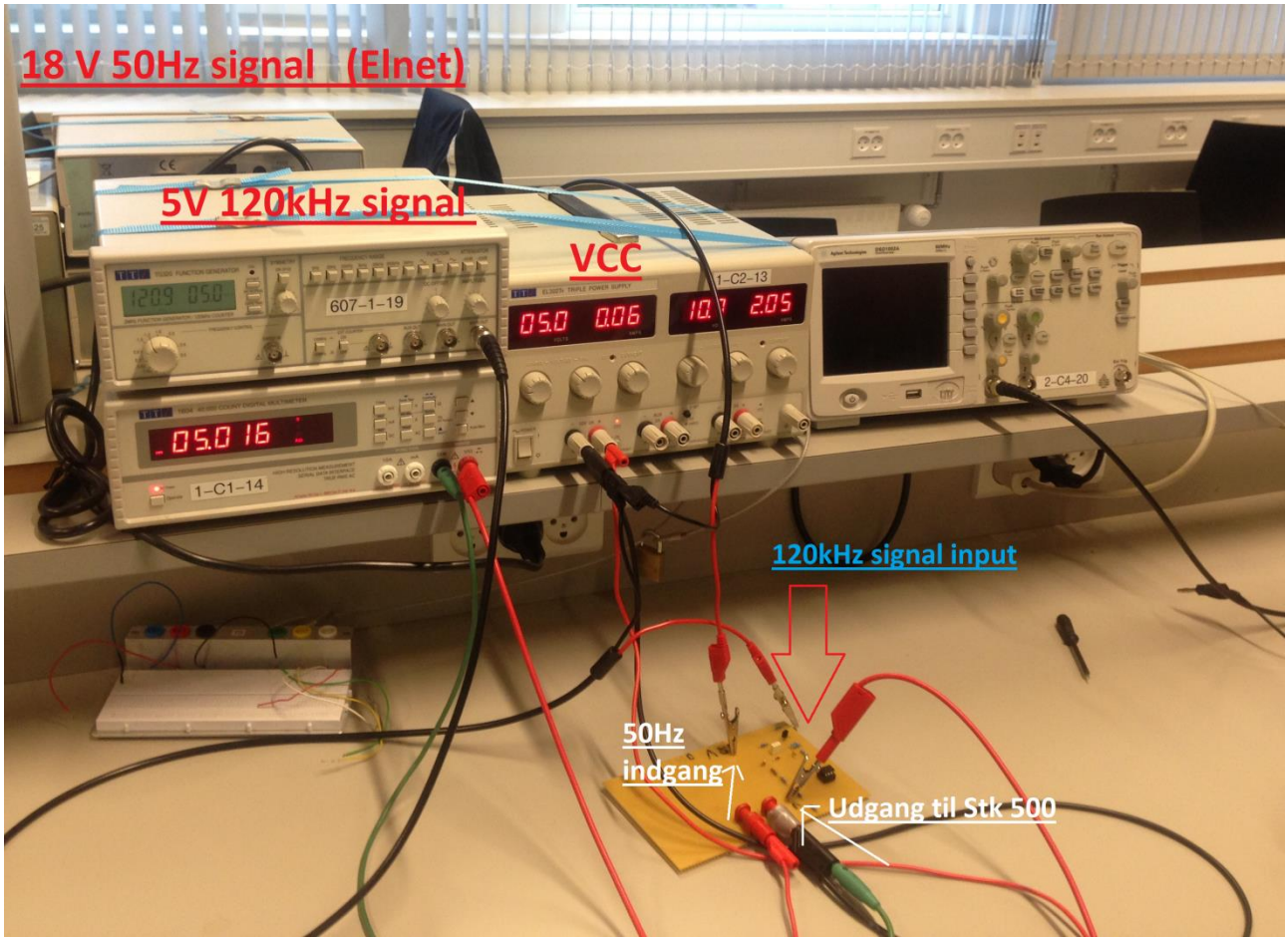
Resultater og diskussion

Vi har valgt at dele resultater/diskussion op i to afsnit, et med hardware og et med software.

Hardware:

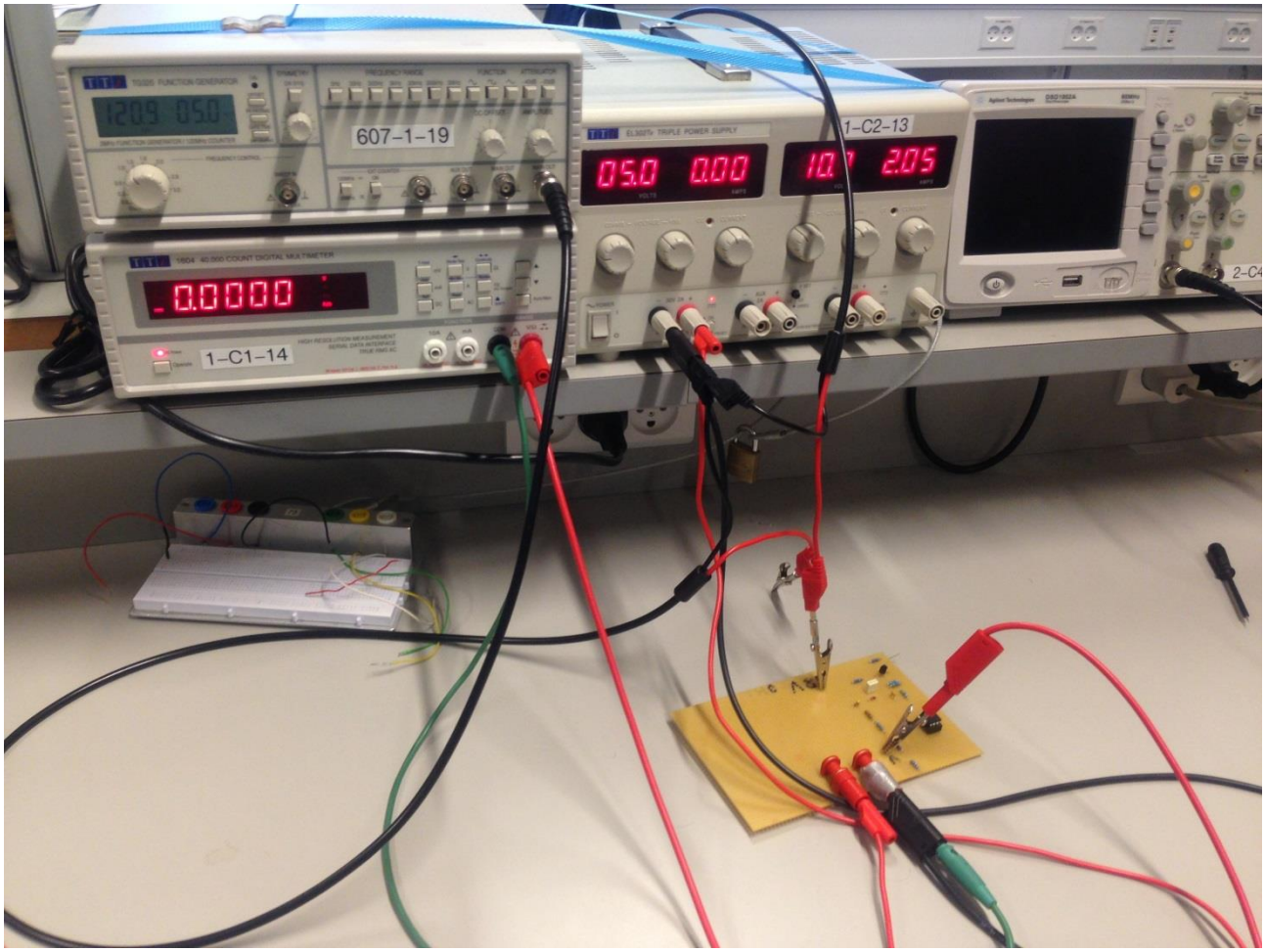
Figur 46 - Realisering af sender

VI tester realiseringen af senderen og måler outputtet, hvor vi kan se på billedet at testen har været succesful. Resultatet har altså været som forventet. Vi kan se at den kun lader det 120kHz signal komme igennem kredsløbet til outputtet. Der vil altså være lidt selvinduktion som også kan ses på billedet hvor der er lidt støj.



Figur 47 - Realisering af sender og modtager med PWM signal.

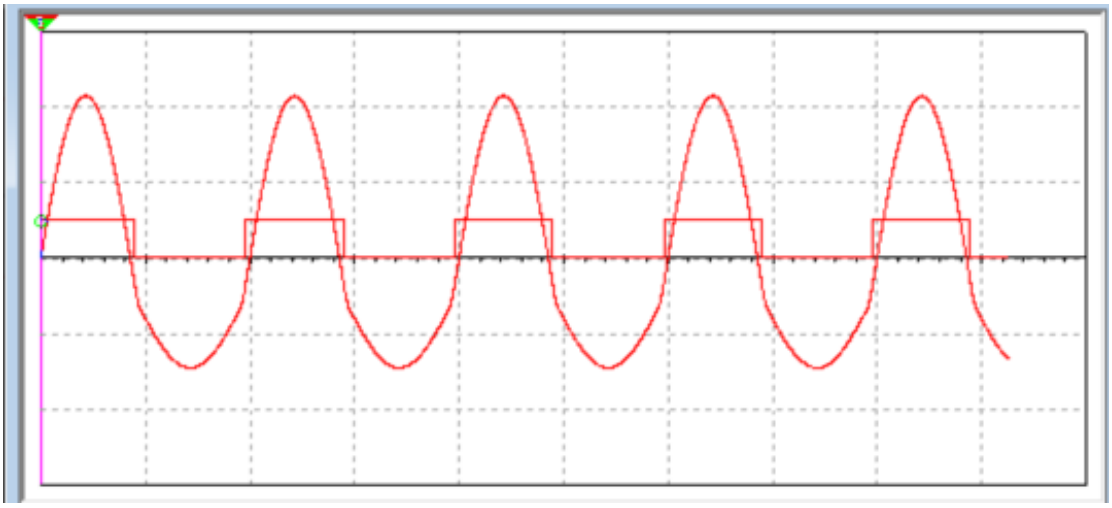
Her testes modtageren med senderen hvor at PWM signalet sendes ind vha. af en funktionsgenerator. Derudover er der også sat et 50Hz signal på for at vise at vores højpasfilter fungerer. Det forventede resultat med det PWM signal vi sendte ind er 5V på outputtet på modtageren og det kan ses på billedet at dette er opnået. Dette resultat er vigtigt da vi nu kan registrerer 120kHz bursts på el-nettet og sende det videre til X10-kontrolleren.



Figur 48 - realisering af modtager og sender uden PWM signal.

Det der ses på dette billede er samme opsætning som på figur 2. Dette er dog uden et PWM signal, og det kan ses på multimeteren at den registrerer 0V på output benet på modtageren. Dette resultat er som forventet.

Det er vist at modtageren både kan gå høj og lav, alt efter om den detekterer et PWM signal eller ej.

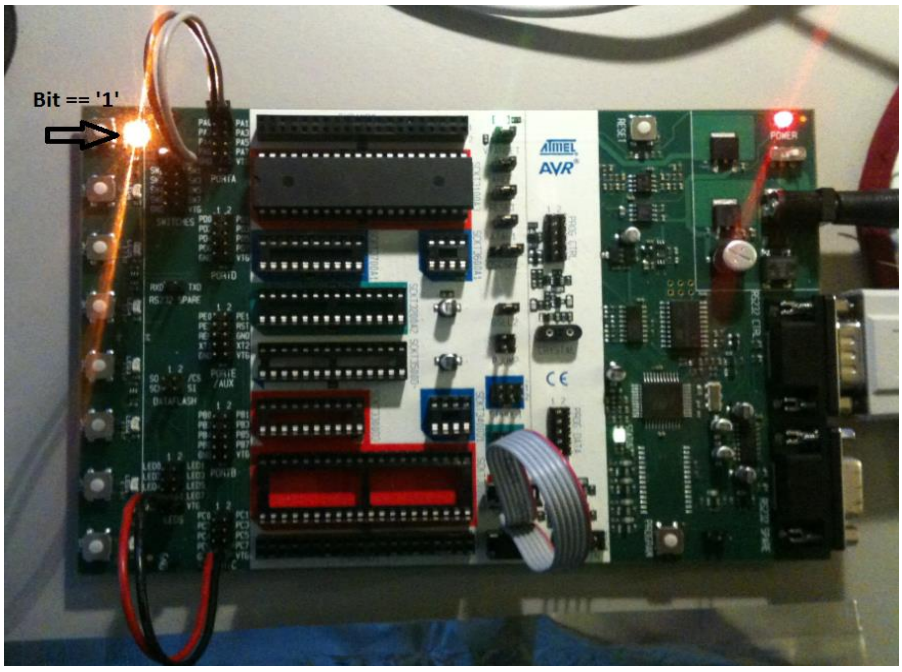


Figur 49 - Simulering af nulpunkts detektor

Sinuskurven af vores el-net på 18V 50Hz og firkant signal er vores zero-crossing detektor som hvis signal går høj på rising edge på el-nettet og går lav på falling edge på el-nettet. Umiddelbart er det et godt tegn at det fungerer i multisim, men det ville have været meget bedre hvis det var realiseret og have været en virkelig måling. Ellers er resultatet som forventet.

Software

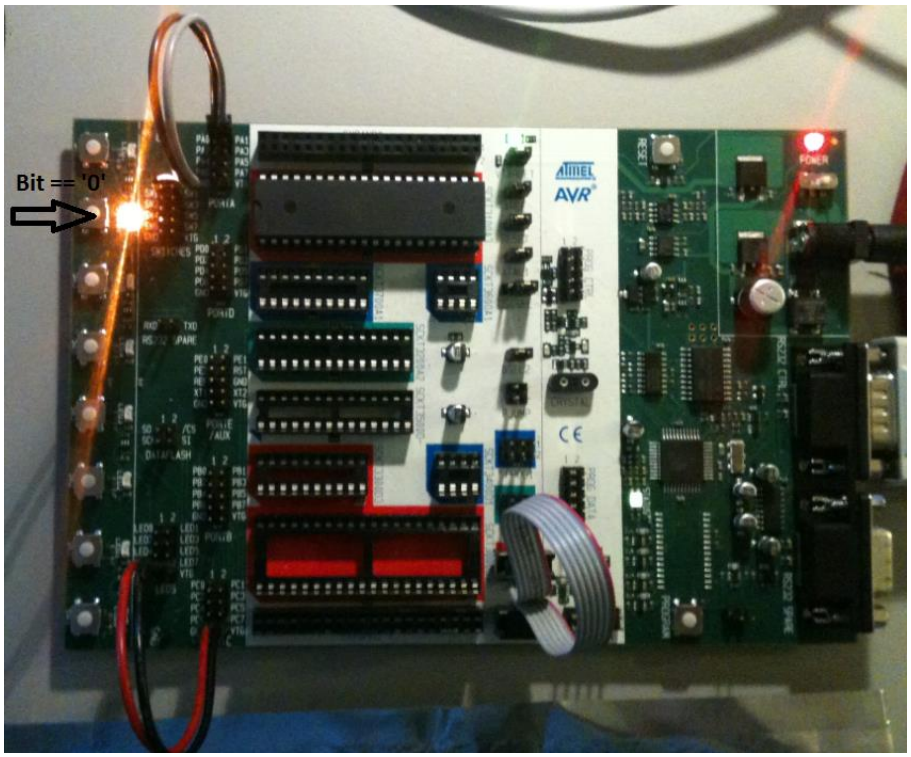
Carrier generatoren i software blev i software testet vha. observation af LED'er på STK-500. Det havde følgende resultat:



Figur 50 - Billede af test på STK-500. LED7 = '1'

Af billedet ovenover ses det at LED 7 på STK-500 lyser. Dette betyder at et zero-crossing er blevet detekteret og et interrupt er blevet sat, hvilket resulterede i, at ét 120 KHz burst er blevet genereret og at det er lykkedes at sende en enkelt høj bit.

Af billedet nedenunder ses det at LED 6 på STK-500 lyser, og at LED 7 er slukket. Dette betyder, at der igen er blevet detekteret et zero-crossing men denne gang er interruptet blevet togglet, og at 120 KHz er blevet overleget af 50 Hz fra lysnettet. Dette resulterede i, at det lykkedes at sende en enkelt lav bit.



Figur 45 - Billede af test på STK-500. LED7 = '0'

Denne form for simulering fortæller, at det er muligt at sende bits, for hvert zero-crossing på lysnettet. Bitsene repræsenterer binære data, som bruger har bedt om, og det er herefter carrier detektorens opgave at lytte efter hver bit detekteret og omskrive det binære data til logiske høje eller lave signaler, som kan tænde eller slukke en enhed.

Carrier Detektor:

Softwaren for carrier detektoren nåede ikke at blive testet. Men hvis men det skulle gøres ville der være sat en UART driver op som er sat sammen mellem STK-500 kittet og X10-modtageren. Her ville der sendes datapakker for at se om koden virker og tænder enheden, som i dette tilfælde skulle være en LED på STK-500 kittet. Modtageren skal altså kunne modtage et binært ”1” eller et binært ”0” fra den pin som det 120kHz bliver sendt ind på. Det resultat som kan forventes af denne test er at en LED vil tænde/slukke på STK-500 kittet hvis det enten er en ON eller OFF datapakke. Som en test for at få et resultat ville være at sætte hele systemet sammen og se om det hele fungerer. Her skulle det forventede resultat være at kunne sende bits ud på det lukkede 18V el-net og samle dem til en gode, hvorefter en enhed skulle tænde/slukke

Opnåede erfaringer

I dette projektforløb har vi opnået et par erfaringer som vi synes kunne hjælpe os med at forbedre projekt forløbet og mere optimeret ydelse i fremtidige projekter. Som gruppe har vi mistet overblikket over projektet under vejs derfor kunne det være en god ide at give en gruppe medlem ansvar for oveblik over systemet og tidsplanen. Vi blev meget presset slutning af projektforløbet pga. mangel på en projektleder.

Endvidere var vi for sent ude med at fordele ansvarsområde, og havde ikke helt styr på hvem der skulle lave hvad. Det ville have været smartere at lave en kontrakt fra starten så alle vidste hvad de skulle lave, og så der var styr på alle opgaver.

Vi manglede at diskutere hvordan systemet virkede og at alle havde samme forståelse for systemet. Vi har erfaret at der er en sekretær i gruppen som havde ansvar for mødereferater og at vi havde en anden en som skrev spørgsmålene til vejlederen (”kunden”) føre den ugentlig møde. Disse to ting har optimeret vores arbejdsproces. Vi aftalte bestemte arbejdsdage i ugen så alle var med på hvor og hvornår vi skulle mødes og kunne planlægge bedre.

Fremtidigt arbejde

Vi er nået så langt med vores projekt at næsten alle enhedstest er overstået og alle virker som de skal. Næste skidt ville have været at bygge de sidste dele færdigt og få dem testet individuelt, herefter skal der laves mindre del test, før der kan laves et samlet system test.

Hvis den sidste samlet test har været succesfuld, kan projektet videreudvikles til at lys enhederne kan variere i lys styrke eller der kan tilføjes enheder med andre funktioner end lys (f.eks. en bevægelses sensor) I bund og grund kan de fleste ting der bruger lysnettet kan blive styret via X.10 udstyr, da de bliver tændt via et digitalt signal, der bliver bestemt fra en PC i stedet for en manuel konkakt.

Med dette system ville man kunne via et pre-sat scenarie kunne lave morgenmad med kaffe/the, æg og ristet brød. Hvis man har en kaffemaskine/the kedel, ægge koger og en brødrister tilsluttet til et X.10 system, hvor brugeren har indstillet til hvornår på morgenen de forskellige maskiner skal tænde. Ud over dette skal brugeren tilføje de nødvendige fødevare til de respektive maskiner.

Konklusion

For os lykkedes det ikke at blive færdige med alt. Vi fik testet alle enkelt dele af systemet (bortset fra X10 modtager driver), men mangler at teste systemet som helhed hvor alt er sat sammen. Hovedårsagen til at vi ikke nåede alle ting var at under design & systemarkitektur mistede vi overblikket, fordi vores vejledning syntes modstridende med teori fra undervisning og kom derfor bagefter med processen.
Hvis man ser bort fra det negative har det, som vi har nået af projektet, været en succes både inden for SW og HW. Implementering er lykkedes på alle fronter.
Umiddelbart har samarbejdet fungeret fint, men punktlighed til møder og det med at komme til tiden er ikke altid lykkes.
I teorien har vi fået besvaret vores problemstilling omhandlende tyverisikring i kraft af de "use cases" vi har beskrevet.
Vi har benyttet os af ASE modellen, hvor vi på grund af mangel på overblik i forhold til systemarkitektur og design har arbejdet utroligt iterativt, idet vi hele tiden har lært noget nyt og har benyttet muligheden til at rette de fejl vi fandt. Udover det har vi også arbejdet lineært, idet vi startede med systemkrav og gik så videre til design og system arkitektur osv.
Generelt viste alle test/målinger det som der var forventet, men der kunne godt have været flere målinger på HW da vi manglede dokumentation for nogle enkelte ting.

Referencer

- Projektdokumentation
- X10 Protokol