

Projektdokumentation
Home Automation System
Gruppe 1

2. Semesterprojekt E2PRJ2-03
Ingeniørhøjskolen, Aarhus Universitet
Vejleder: Arne Justesen

17. december 2014

Navn	Studienummer	Underskrift
Morten Hasseris Gormsen	201370948	
Kristian Thomsen	201311478	
Philip Krogh-Pedersen	201311473	
Lasse Barner Sivertsen	201371048	
Henrik Bagger Jensen	201304157	
David Erik Jensen	11229	
Kasper Torp Samuelsen	201311498	
Kristian Søgaard Sørensen	20115255	

Indhold

Indhold	ii
1 Projektformulering (Alle)	1
1.1 Version	1
2 Kravspecifikation (Alle)	3
2.1 Version	3
2.2 Systembeskrivelse	3
2.3 Aktør-kontekst diagram	5
2.3.1 Aktørbeskrivelse	5
2.4 Use Case diagram	5
2.5 Use Case beskrivelse	6
2.6 Funktionelle Krav	10
2.7 Ikke Funktionelle Krav	10
3 Accepttest (Alle)	11
3.1 Version	11
3.2 Funktionelle Krav	11
3.3 Ikke Funktionelle Krav	14
4 Systemarkitektur (Alle)	16
4.1 Version	16
4.2 SysML diagrammer	16
4.2.1 BDD for system kontekst	16
4.2.2 IBD for system kontekst	17
4.2.3 BDD for systemet	18
4.2.4 IBD for systemet	19
4.2.5 IBD for Transmitter-blokken	20
4.2.6 IBD for X.10-enhed i transmitter-blokken	20
4.2.7 IBD for receivers	21
4.2.8 IBD for X.10 enhed i lys-blokken	22
4.3 Signalbeskrivelser	23
4.3.1 Signaltyper	23
4.3.2 Grænseflader	24
4.4 Softwarearkitektur	24
4.4.1 Sekvensdiagram for PC [Use Case 1: Opret Scenarie]	24
4.4.2 Sekvensdiagram for PC [Use Case 2: Vælg Scenarie]	26
4.4.3 Sekvensdiagram for PC [Use Case 3: Stop Scenarie]	27
4.4.4 Klassediagram for PC	28
4.4.5 Sekvensdiagram for Transmitter [Use Case 1: Opret Scenarie]	29
4.4.6 Sekvensdiagram for Transmitter [Use Case 3: Stop Scenarie]	30
4.4.7 Sekvensdiagram for Transmitter [Use Case 4: Afvikl Scenarie]	31
4.4.8 Klassediagram for Transmitter	32
4.4.9 Sekvensdiagram for Receiver [Use Case 4: Afvikl Scenarie]	33
4.4.10 Klassediagram for Receiver	34
5 Hardware Design	35

5.1	Version	35
5.2	Blokopdeling	35
5.3	Transmitter	36
5.3.1	18V AC Transformer	36
5.3.2	Spændingsforsyning (Morten)	36
5.3.3	Kodelås (Lasse og Henrik)	36
5.3.4	Zerocross Detector (Morten og Philip)	36
5.3.5	Carrier Generator (Philip og Lasse)	38
5.4	Receiver	40
5.4.1	Lysmodul (Morten)	40
5.4.2	Carrier Detector (Morten, Henrik og Lasse)	40
6	Software Design	44
6.1	Version	44
6.2	PC Blokken	44
6.2.1	Action (Kristian T.)	44
6.2.2	PC Controller	46
6.2.3	Scenario (Kristian S.)	46
6.2.4	UART (Kasper)	47
6.2.5	UI (Kasper)	49
6.3	Transmitter Blokken	52
6.3.1	Action (Kristian T.)	52
6.3.2	Codelock (David)	53
6.3.3	Time (Kasper)	53
6.3.4	Transmitter (Kristian S.)	55
6.3.5	Transmitter UART (Kasper)	56
6.3.6	Tx10 (Kristian T.)	57
6.4	Receiver Blokken	58
6.4.1	Lampe (David)	58
6.4.2	Receiver (David)	59
6.4.3	Rx10 (Kristian T.)	60
6.5	Protokoller (Kristian T., Kristian S., David og Kasper)	61
6.5.1	Protokol for UART	61
6.5.2	Protokol for X.10	62
6.6	Porte (Kristian T., Kristian S., David og Kasper)	63
6.7	Statemachines (Kristian S. og David)	64
7	Hardwareimplementering	67
7.1	Version	67
7.2	Spændingsforsyning (Lasse)	67
7.3	Kodelås (Philip)	67
7.4	Zerocross Detector (Morten og Henrik)	70
7.5	Carrier Generator (Lasse og Philip)	72
7.6	Lysmodul (Kristian S.)	76
7.7	Carrier Detector (Morten og David)	77
8	Softwareimplementering	80
8.1	Version	80
8.2	PC blokken	80

8.2.1	Scenario (Kristian S.)	80
8.2.2	Action (Kristian S.)	81
8.2.3	UI (Kasper)	82
8.2.4	PC UART (Kasper)	84
8.2.5	PC Main (Henrik)	86
8.3	Generelt om Microcontrollers (David)	87
8.3.1	C++	87
8.3.2	C++ Compiler på Microcontrollerne	87
8.3.3	C++ og Interrupts	87
8.3.4	C++ på AVR Object Kommunikation	87
8.4	Transmitter Blokken	88
8.4.1	Action Klassen (Kristian T.)	88
8.4.2	Codelock Klassen (Kristian T.)	88
8.4.3	Time Klassen (Kristian S.)	88
8.4.4	Transmitter Klassen (Kristian T.)	88
8.4.5	Tx10 Klassen (Kristian T.)	89
8.4.6	TxUART Klassen (Kristian T.)	92
8.5	Receiver	94
8.5.1	Rx10 Klassen (David)	94
8.5.2	Receiver Control Klassen (David og Lasse)	96

Litteraturliste

97

1 Projektformulering (Alle)

1.1 Version

Dato	Version	Initialer	Ændring
8. oktober	1	KT	Første udkast af dokumentet efter review.
15. december	2	LS	Mindre rettelser. Endelig version.

Vores mål med dette projekt er at udvikle et home automation system, som kan simulering tilstedsvarsel i en bolig. Gennem styring af de tilkoblede enheder, som fx TV, lys og radio, kan udefrakommende ledes til at tro, at der er nogen tilstede, selvom brugeren ikke er hjemme. Udover home security vil systemet også kunne hjælpe brugeren i dagligdagen ved at automatisere visse aspekter af hverdagen, fx ved at indstille et fast tidspunkt for aktivering af radio om morgenen eller mulighed for at tænde og styre TV. Systemet er tilpasset én bestemt bruger og dennes særlige behov. Brugeren er en familie, hvor alle er beskæftigede i dagstimerne og derfor vil sikre sig mod tyveri. Familien er ofte på ferie og er generelt meget hjemmefra. Når de endelig er hjemme, vil de gerne have en mere automatiseret hverdag i deres hjem. Familien kan via systemet styre to lamper, et TV og en radio.

Brugeren kan via systemet styre følgende funktioner i enhederne:

- TV:
 - Tænde- og slukke
 - Skifte kanal (op/ned)
 - Justere volumen
 - Valg af specifikt kanalnummer
- Radio:
 - Tænde og slukke
 - Kanalsøgning (op/ned)
 - Justere volumen
 - Valg af forudindstillet kanal
- Lamper:
 - Tænde og slukke enkeltvis
 - Justere lysintensitet enkeltvis

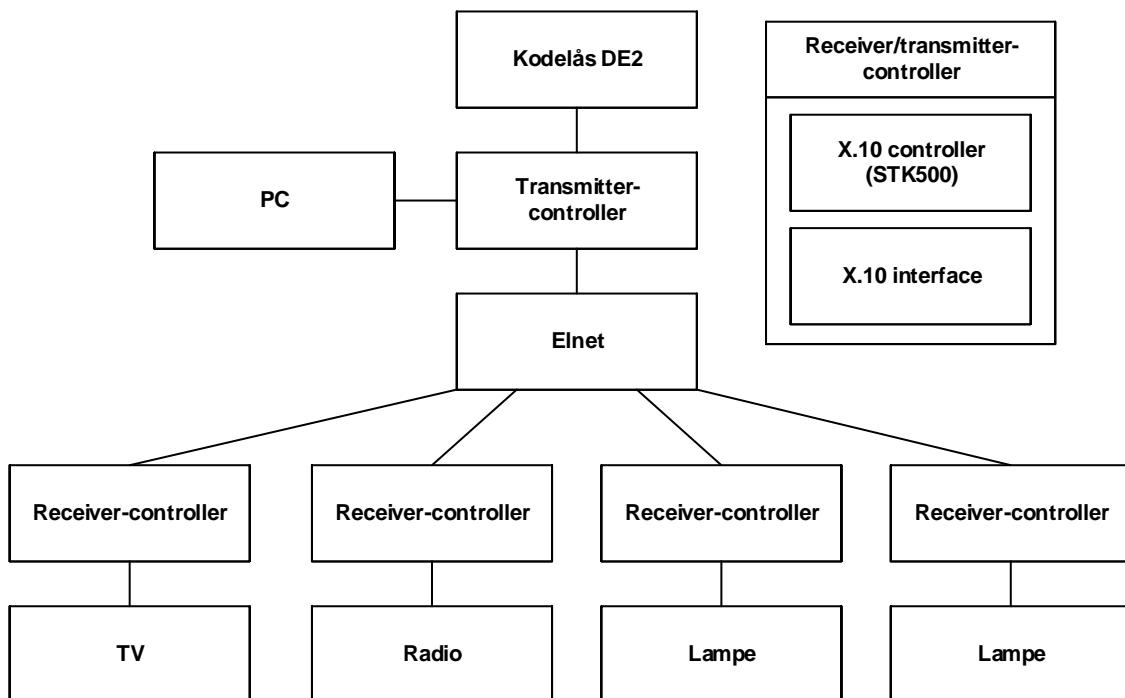
Disse enheders funktioner kan herefter indstilles af brugeren via den centrale computer til at ske på bestemte tidspunkter ved at konfigurere eller vælge forudbestemte scenarier. Disse scenarier har en varighed på 24 timer og gentages automatisk indtil de stoppes af brugeren. For at forhindre uautoriseret adgang til systemets brugerflade er systemet beskyttet med en kodelås, som brugeren selv ved tidligere lejlighed har indstillet.

Systemet består af:

- Software til installation på brugerens PC
- En X.10 transmitter-controller, bestående af et STK500 kit samt et X.10 interface

- Fire X.10 receiver-controllere, bestående af STK500 kit samt X.10 interface
- En kodelås på et Altera DE2 udviklingsboard

Systemet er forbundet til det eksisterende elnet i hjemmet via X.10 controllerne. Brugeren indstiller altting via softwareen på sin PC, som gemmer brugerens preferencer på transmittereren. Denne skal være koblet til både kodelås samt X.10 interfacet og sørger for at afvikle det indstillede program, selv når brugerens PC er slukket. For hver enhed der skal styres, er der en specifik receiver-controller, som understøtter den tilkoblede enhed. For radio og TV receiver-controllerne er der på hver koblet en IR-transmitter på, som styres af det pågældende STK500 kit. Dette sikrer kommunikation til radio/TV. Controllerne er hver især tilkoblet elnettet og deres respektive enheder. Hele denne opsætning kan ses i Figur 1.



Figur 1: Kommunikationsveje mellem enheder i systemet

2 Kravspecifikation (Alle)

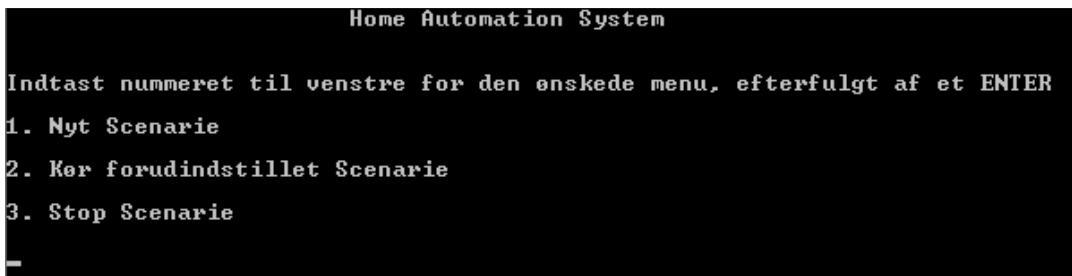
2.1 Version

Dato	Version	Initialer	Ændring
8. oktober	1	KT	Første udkast af dokumentet efter review
19. november	2	KR	Glødepære ændret til LED.
9. december	3	KT	Indsat billede af softwaren samt små rettelser.
15. december	4	LS	Små rettelser. Endelig version.

2.2 Systembeskrivelse

PC-software

Softwareen på brugerens PC er sammen med DE2-boardet grænsefladen til systemet. Når programmet åbnes vil der vises en brugerflade på skærmen, som guider brugeren gennem opsætning og indstilling af systemet. Systemet fortsætter selvom brugerens PC slukkes eller at softwaren lukkes ned, men ønskede interaktion med systemet skal softwaren startes igen. Figur 2 viser et udkast af hovedmenuen i userinterfacet.



Figur 2: Hovedmenuen for softwaren til PC

Scenarier

Systemet indeholder tre prædefinerede scenarier, som ligger i PC-softwaren. Disse tre scenarier kan ikke ændres og der kan ikke tilføjes flere. Brugeren har derimod mulighed for at oprette et brugerdefineret scenario, som overføres til transmitter-controlleren og eksisterer så længe det afvikles. Ved afvikling af et nyt scenario (brugerdefineret eller prædefineret) overskrives det scenario, som er gemt på transmitter-controlleren. Et scenario består af op til 20 aktioner, et eksempel på en aktion kunne være ”tænd lampe 1 klokken 12:00”, hvor: ”lampe 1” er enheden, ”tænd” er kommandoen og ”12:00” er tidspunktet. I Tabel 2.2 nedenfor vises et eksempel på tre aktioner.

Aktionsnummer	Tidspunkt	Enhed	Kommando
1	12:00	Lampe 1	Tænd
2	13:15	TV	Tænd
3	13:30	TV	Sluk

Tabel 1: Eksempel på scenario.

Transmitter-controller

Controlleren modtager de konfigurerede indstillinger fra PC'en og validerer kodelåsen på DE2-boardet, før brugeren kan tilgå PC-softwaren. Kan koden ikke valideres, sørger transmitter-controlleren for at nægte adgang til systemet. Når systemet er aktiveret, sørger transmitter-controlleren for at eksekvere det valgte scenarie og afsende kommandoer til de valgte enheders receiver-controllere over elnettet.

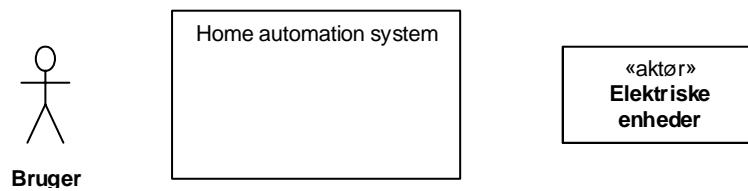
Receiver-controller

Hver receiver-controller har sit eget enheds-ID, som kun den reagerer på. Ydermere er begge lampe-controllere grupperet under samme huskode, hvor TV-controlleren og radio-controlleren har hver sin huskode. Hver type af receiver-controller understøtter forskellige kommandoer.

Kodelås DE2

Kodelåsen bliver indstillet af brugeren ved installation af systemet, denne kan herefter ændres efter brugerens ønske, såfremt den tidligere kode indtastes først. Kodelåsen kan kun indeholde én kode.

2.3 Aktør-kontekst diagram



Figur 3: Aktør-kontekst diagram

2.3.1 Aktørbeskrivelse

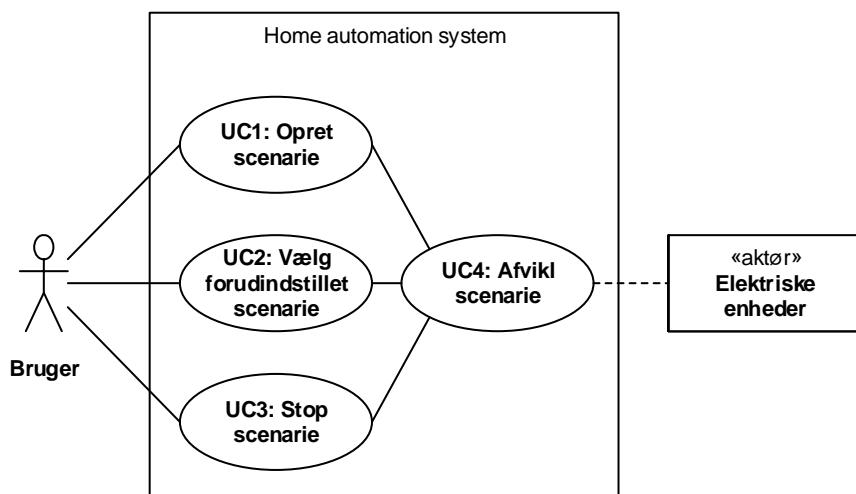
Bruger (Primær aktør)

Brugeren er kunden, som bruger systemet til hverdag. Brugeren ønsker at oprette og afvikle scenarier.

Elektriske enheder (Sekundær aktør)

Elektriske enheder er hhv. to lamper, et fjernsyn og en radio, der er tilkoblet systemet.

2.4 Use Case diagram



Figur 4: Use Case diagram

2.5 Use Case beskrivelse

Navn:	UC1: Opret scenarie
Mål:	Oprette og eksekvere et brugerdefineret scenarie.
Initering:	Bruger
Aktører:	Bruger (primær)
Reference:	UC4: Afvikl scenarie
Antal samtidige forekomster:	En
Forudsætning:	At de tre koder er indtastet korrekt på kodelåsen, systemet er operationelt og brugeren har valgt "Opret scenarie" i programmet på PC'en.
Resultat:	Systemet afvikler brugers scenarie.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Systemet præsenterer en liste af aktioner. 2. Brugeren vælger aktionsnummer. <ul style="list-style-type: none"> • [Ext 1: Ugyldigt input] 3. Systemet præsenterer en liste af enheder. 4. Brugeren vælger enhed for aktion. <ul style="list-style-type: none"> • [Ext 1: Ugyldigt input] 5. Systemet præsenterer en liste af kommandoer. 6. Brugeren vælger kommando for aktion. <ul style="list-style-type: none"> • [Ext 1: Ugyldigt input] 7. Systemet beder om indtastning af et tidspunkt for aktion. 8. Brugeren indtaster tid for aktion. <ul style="list-style-type: none"> • [Ext 1: Ugyldigt input] 9. Brugeren præsenteres med listen af aktioner. 10. Brugeren vælger aktionsnummer (gentag fra punkt 3.) eller at igangsætte scenariet. <ul style="list-style-type: none"> • [Ext 1: Ugyldigt input] 11. Programmet sender det konfigurerede scenarie til transmitter-controlleren. 12. UC4: Afvikl scenarie påbegyndes. 13. PC-softwaren vender tilbage til hovedmenuen.
Udvidelser:	<p>[Ext 1: Ugyldigt input]</p> <ol style="list-style-type: none"> 1. Systemet informerer brugeren om at der er foretaget en forkert indtastning og informerer brugeren om at prøve igen.

Tabel 2: UC1: Opret scenarie

Navn:	UC2: Vælg forudindstillet scenarie
Mål:	Eksekvere et forudindstillet scenarie.
Initering:	Bruger
Aktører:	Bruger (primær)
Reference:	UC4: Afvikl scenarie
Antal samtidige forekomster:	En
Forudsætning:	At de tre koder er indtastet korrekt på kodelåsen, systemet er operationelt og brugeren har valgt ”Vælg forudindstillet scenarie” i programmet på PC’en.
Resultat:	Systemet afvikler brugerens scenarie.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Systemet præsenterer en liste af scenarier. 2. Brugeren vælger scenarie. <ul style="list-style-type: none"> • [Ext 1: Ugyldigt input] 3. Programmet sender det konfigurerede scenarie til transmitter-controlleren. 4. UC4: Afvikl scenarie påbegyndes. 5. PC-softwaren vender tilbage til hovedmenuen.
Udvidelser:	<p>[Ext 1: Ugyldigt input]</p> <ol style="list-style-type: none"> 1. Systemet informerer brugeren om at der er foretaget en forkert indtastning og informerer brugeren om at prøve igen. 2. Systemet vender tilbage til hovedscenariet ved forrige punkt.

Tabel 3: UC2: Vælg forudindstillet scenarie

Navn:	UC3: Stop scenarie
Mål:	At stoppe al aktivitet på de elektriske enheder.
Initering:	Bruger
Aktører:	Bruger (primær)
Reference:	UC4: Afvikl scenarie
Antal samtidige forekomster:	En
Forudsætning:	At de tre koder er indtastet korrekt, at systemet er operationelt og at der er et scenarie under afvikling og brugeren har valgt "Stop scenarie" i programmet på PC'en.
Resultat:	Systemet stopper igangværende scenarie.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Systemet prompter brugeren om vedkommende er sikker. 2. Brugere vælger "ja". <ul style="list-style-type: none"> • [Ext. 1: Brugeren vælger "nej"] 3. UC4: Afvikl scenarie afsluttes. 4. Der sendes en sluk-kommando til alle enheder. 5. PC-softwaren vender tilbage til hovedmenuen.
Udvidelser:	<p>[Ext 1: Brugeren vælger "nej"]</p> <ol style="list-style-type: none"> 1. Systemet vender tilbage til hovedmenuen og UC3: Stop scenarie afsluttes.

Tabel 4: UC3: Stop scenarie

Navn:	UC4: Afvikl scenarie
Mål:	Afvikler kontinuerligt det valgte scenarie.
Initering:	UC1 eller UC2
Aktører:	Elektriske enheder (sekundære)
Reference:	UC1, UC2 og UC3
Antal samtidige forekomster:	En
Forudsætning:	Systemet er operationelt.
Resultat:	Systemet afvikler det valgte scenarie.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Systemet tæller tiden et sekund opad. 2. Systemet tjekker om tiden stemmer overens med en aktion. 3. Tiden stemmer ikke overens med en aktion. <ul style="list-style-type: none"> • [Ext 1: Tiden stemmer overens med en aktion] 4. Hovedscenariet gentages fra Punkt 1..
Udvidelser:	<p>[Ext 1: Tiden stemmer overens med en aktion]</p> <ol style="list-style-type: none"> 1. Transmitter-controlleren sender den pågældende kommando. 2. Den pågældende receiver-controller afkoder og udfører den pågældende kommando.

Tabel 5: UC4: Afvikl scenarie

2.6 Funktionelle Krav

Systemet...

1. ... *Skal* kunne tænde og slukke for lamper.
2. ... *Skal* kunne dimme op og ned for lamper.
3. ... *Skal* kunne tænde og slukke for et TV.
4. ... *Skal* kunne inkrementere og dekrementere kanal på et TV.
5. ... *Skal* kunne justere volumen op og ned på et TV.
6. ... *Skal* kunne vælge en specifik kanal på et TV.
7. ... *Skal* kunne tænde og slukke for en radio.
8. ... *Skal* kunne søge efter næste eller forrige kanal.
9. ... *Skal* kunne justere volumen op og ned på en radio.
10. ... *Skal* kunne vælge en forudindstillet radiokanal.
11. ... *Må* ikke kunne anvendes uden korrekt kode.
12. ... *Skal* have en GUI.
13. ... *Skal* kunne interageres med via en PC.
14. ... *Skal* give bruger mulighed for at oprette og eksekvere et scenarie.
15. ... *Skal* give bruger mulighed for at vælge og eksekvere et forudindstillet scenarie.
16. ... *Skal* give bruger mulighed for at standse det igangværende scenarie.
17. ... *Skal* kunne håndtere ugyldigt input fra bruger.
18. ... *Skal* fungere selvom den tilkoblede PC er slukket.
19. ... *Skal* kunne kommunikere med den tilkoblede PC via en seriell forbindelse.

2.7 Ikke Funktionelle Krav

Systemet...

1. ... *Skal* fungere på et $18VAC$ elnet med en frekvens på $50Hz$.
2. ... *Skal* via brugerfladen kunne præsentere en aktionsliste med op til 20 aktioner.
3. ... *Skal* kunne oprette et scenarie med op til 20 aktioner.
4. ... *Skal* kunne afvikle en aktion med en præcision på ± 1 minut, i forhold til den valgte tid.
5. ... *Skal* bruge X.10 protokol [11] til kommunikation via elnettet.
6. ... *Skal* kunne afvikle minimum en aktion i minuttet.
7. ... *Skal* rumme 3 prædefinerede scenerier.
8. ... *Skal* kunne iværksætte et prædefineret scenarie ved højst 5 indtastninger fra brugerens side.
9. ... *Skal* kunne håndtere 2 lamper, et TV og en radio.
10. ... *Skal* via dimmefunktionen i trin af $10\% \pm 1\%$ kunne regulere middelspændingen over en 5mm gul L53-YD LED eller tilsvarende fra 5% til 95% af elnettets spænding.
11. ... *Skal* have en min. up-time på 80% over en time.
12. Teksten i UI'en *skal* være grå med hex-farvekode `#C0C0C0` med sort baggrund med hex koden `#000000`.
13. Når en lampe er slukket, *skal* middelstrømmen gennem denne være $0A \pm 50mA$ over en periode på 10 sekunder.
14. Transmitter- og receiver-controllererne *skal* være afskærmede af individuelle kasser med dimensioner på maksimalt $50cm \times 50cm \times 20cm$.
15. Brugerfladen *skal* være på dansk.

3 Accepttest (Alle)

3.1 Version

Dato	Version	Initialer	Ændring
8. oktober	1	KT	Første udkast af dokumentet efter review.
8. december	2	KT	Accepttest udført.
15. december	3	LS	Mindre rettelser. Endelig version.

3.2 Funktionelle Krav

Use case under test		UC1: "Opret Scenarie"		
Scenarie		Hovedscenarie		
Forudsætning		Koden er indtastet korrekt på kodelåsen og systemet er operationelt.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
1.1	Brugeren har startet programmet og vælger "Opret Scenarie".	Systemet præsenterer en liste af aktioner.	Systemet præsenterer en liste af aktioner	Godkendt
1.2a	Brugeren vælger et ugyldigt aktionsnummer	Systemet giver en fejlmeddeelse og beder om nyt input.	Systemet giver en fejlmeddeelse og beder om nyt input	Godkendt
1.2b	Brugeren vælger et gyldigt aktionsnummer	Brugeren bliver præsenteret for en liste af enheder.	Brugeren bliver præsenteret for en liste af enheder.	Godkendt
1.3	Brugeren indtaster en enhed for aktion.	Systemet præsenterer en liste af enheder.	Systemet præsenterer en liste af enheder.	Godkendt
1.4	Brugeren vælger en kommando for aktion.	Systemet præsenterer en liste af kommandoer.	Systemet præsenterer en liste af kommandoer.	Godkendt
1.5	Brugeren vælger tid for aktion	Systemet beder om indtastning af et tidspunkt for aktion.	Systemet beder om indtastning af et tidspunkt for aktion.	Godkendt
1.6	Brugeren gentager pkt. 1.2b til 1.5.	Brugeren præsenteres med listen af aktioner.	Brugeren kan vælge mellem kører scenariet eller rediger den	Godkendt / Anden implementation end use case beskrivelse
1.7	Brugeren vælger "Eksekvr scenarie"	Hovedmenuen præsenteres og scenariet eksekveres som planlagt.	Hovedmenuen præsenteres og scenariet eksekveres som planlagt.	Godkendt

Tabel 6: Accepttest for UC1: Opret scenarie

Use case under test		UC2: "Vælg Forudindstillet Scenarie"		
Scenarie		Hovedscenarie		
Forudsætning		Koden er indtastet korrekt på kodelåsen og systemet er operationelt.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
2.1	Brugeren starter programmet og vælger "Vælg forudindstillet scenarie".	Systemet præsenterer en liste af scenarier.	Systemet præsenterer en liste af scenarier.	Godkendt
2.2	Brugeren vælger scenarie.	Hovedmenuen præsenteres og scenariet eksekveres som planlagt.	Hovedmenuen præsenteres og scenariet eksekveres som planlagt.	Godkendt

Tabel 7: Accepttest for UC2: Vælg forudindstillet scenarie

Use case under test		UC3: "Stop Scenarie"		
Scenarie		Hovedscenarie		
Forudsætning		Koden er indtastet korrekt på kodelåsen og systemet er operationelt.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
3.1	Brugeren starter programmet og vælger "Stop scenarie".	Systemet prompter brugeren om vedkommende er sikker.	Systemet prompter brugeren om vedkommende er sikker.	Godkendt
3.2	Brugeren vælger "nej".	Systemet vender tilbage til hovedmenuen; eksekvering af scenarie fortsætter.	Systemet vender tilbage til hovedmenuen; eksekvering af scenarie fortsætter.	Godkendt
3.3	Brugeren vælger "Stop scenarie".	Systemet prompter brugeren om vedkommende er sikker.	Systemet prompter brugeren om vedkommende er sikker.	Godkendt
3.4	Brugeren vælger "ja".	Eksekvering af igangværende scenarie stopper, hovedmenuen præsenteres og samtlige elektriske enheder slukkes.	Eksekvering af igangværende scenarie stopper, hovedmenuen præsenteres og samtlige elektriske enheder slukkes.	Godkendt

Tabel 8: Accepttest for UC3: Stop scenarie

Use case under test		UC4: "Afvikl Scenarie"		
Scenarie		Hovedscenarie		
Forudsætning		Systemet er operationelt.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
4.1	Brugeren opretter og eksekverer et testscenarie i følge bilag [16]	Programmet viser hovedmenuen	Programmet fremviser hovedmenuen	Godkendt
4.2	Brugeren slukker for PC'en og venter 2 minutter.	Lampen tænder	Lampen tænder	Godkendt
4.3	Brugeren venter 5 minutter	Lampen dimmer ned	Lampen dimmer ned	Godkendt
4.4	Brugeren venter 5 minutter	Lampen dimmer op	Lampen dimmer op	Godkendt
4.5	Brugeren venter 5 minutter	Lampen slukker	Lampen slukker	Godkendt
4.6	Brugeren venter 5 minutter	TV'et tænder	Ikke implementeret	Ikke godkendt
4.7	Brugeren venter 5 minutter	Næste kanal på TV'et vælges	Ikke implementeret	Ikke godkendt
4.8	Brugeren venter 5 minutter	Forrige kanal på TV'et vælges	Ikke implementeret	Ikke godkendt
4.9	Brugeren venter 5 minutter	Volumen på TV'et øges	Ikke implementeret	Ikke godkendt
4.10	Brugeren venter 5 minutter	Volumen på TV'et dæmpes	Ikke implementeret	Ikke godkendt
4.11	Brugeren venter 5 minutter	TV'et skifter til kanalnummer 5	Ikke implementeret	Ikke godkendt
4.12	Brugeren venter 5 minutter	TV'et slukker	Ikke implementeret	Ikke godkendt
4.13	Brugeren venter 5 minutter	Radioen tænder	Ikke implementeret	Ikke godkendt
4.14	Brugeren venter 5 minutter	Næste kanal på radioen søgeres	Ikke implementeret	Ikke godkendt
4.15	Brugeren venter 5 minutter	Forrige kanal på radioen søgeres	Ikke implementeret	Ikke godkendt
4.16	Brugeren venter 5 minutter	Volumen på radioen øges	Ikke implementeret	Ikke godkendt
4.17	Brugeren venter 5 minutter	Volumen på radioen dæmpes	Ikke implementeret	Ikke godkendt
4.18	Brugeren venter 5 minutter	Den forudindstillede radiokanal vælges	Ikke implementeret	Ikke godkendt
4.19	Brugeren venter 5 minutter	Radioen slukker	Ikke implementeret	Ikke godkendt

Tabel 9: Accepttest for UC4: Afvikl scenarie

3.3 Ikke Funktionelle Krav

Krav	Test	Forventet resultat	Resultat	Godkendt / kommentar
1	Systemet tilkobles den 230V AC til 18 V AC transformator og UC2 gennemføres	UC2 gennemføres planlagt.	Visuel test: UC2 kører som forventet.	Godkendt
2	Brugeren vælger ”opret scenario” i hovedmenuen.	Der vises en liste af aktioner med op til 20 aktioner.	Visuel test: Der vises en list med 20 aktioner.	Godkendt
3	Brugeren vælger ”opret scenario” og indtaster 20 aktioner efter eget valg.	Listen med aktioner vises, uden mulighed for at tilføje flere.	Brugeren kan kun tilføje 20 aktioner.	Godkendt
4	Brugeren gennemfører UC1 og noterer de valgte tidspunkter og venter til de pågældende tidspunkter (målt på PC'en).	Tidspunkterne for aktionerne stemmer overens med de indtastede tidspunkter med en fejlmargin på ± 1 minut.	Aktionerne gennemføres inden for fejlmarginen	Godkendt
5	Et oscilloskop kobles på elnettet mens en kommando afvikles.	Målinger viser 120kHz impulser efter nulgennemgangene, som stemmer overens med X.10 protokollen.	oscilloskopet viser 120kHz impluser efter en nulgennemgang.	Godkendt
6	Der oprettes et scenario med tre samtidigt forekommende aktioner.	Samtlige tre aktioner udføres indenfor tre minutter fra det indstillede tidspunkt.	De tre aktioner blev udført inden for 2 sekunder.	Godkendt.
7	Brugeren vælger ”Vælg forudinstilledede scenarier” i hovedmenuen.	Der vises tre forudindstillede scenarier at vælge imellem.	Visuel test: Der vises tre scenarier.	Godkendt.
8	Brugeren gennemfører UC2 og noterer antal indtastninger.	Antallet af indtastninger overgår ikke 5.	Brugeren skal kun bruge 4 indtastinger.	Godkendt.
9	Der oprettes et scenario med fire enheder, to lamper, et tv og en radio.	Scenariet gennemføres korrekt.	Ikke implementeret.	Ikke godkendt.

10	Der oprettes et scenarie, som dimmer en lampe gradvist fra minimum til maksimum. Strømmen over lampen måles under udførel af scenariet.	Middelstrømmen gennem lampen måles til hhv. 5%, 15% ... 95% \pm 3% af lampens I_{max} ved 5V [7].	Alle undtagen 5% virkede.	Ikke godkendt. Efter fejlsøgning, formodes problemet at ligge i software, men fejlen blev ikke lokaliseret og rettet.
11	UC2 afvikles og systemet kører i 2 timer.	Tiden hvor systemet ikke er operationelt udgør mindre end 20% af den samlede tid.	Visuel test: Aktioner udført uden fejl.	Godkendt.
12	Programmet afvikles på en PC og farven på hhv. tekst og baggrund måles med ColorSchemer ColorPix [17].	Tekstfarven er hvid #C0C0C0 og baggrundsfaven er #000000.	ColorPix's resultater stemmer overens med kravene.	Godkendt.
13	Der oprettes et scenarie, som slukker for en af lamperne. Strømmen gennem denne lampe måles efter aktionen er udført.	Der måles en strøm på $0A \pm 50mA$ gennem lampen.	Der måles en strøm på omkring 0mA.	Godkendt.
14	Afskærmningen på transmitter- og receiver-controllerne måles med en tomtestok.	Samtlige afskærmninger er mindre end $50 \times 50 \times 20cm$	Ikke implementeret.	Ikke godkendt.
15	En dansktalende testperson læser brugerfladen.	Brugerfladen konstateres at være på dansk.	Visuel test: Menuen er på dansk.	Godkendt.

Tabel 10: Ikke-funktionelle krav

4 Systemarkitektur (Alle)

4.1 Version

Dato	Version	Initialer	Ændring
29. oktober	1	LS	Første udkast af dokumentet.
19. november	2	KT	Mange mindre rettelser efter review.
02. december	3	HBJ	Rettelser af lampe signaler.
15. december	4	LS	Mindre rettelser. Endelig version.

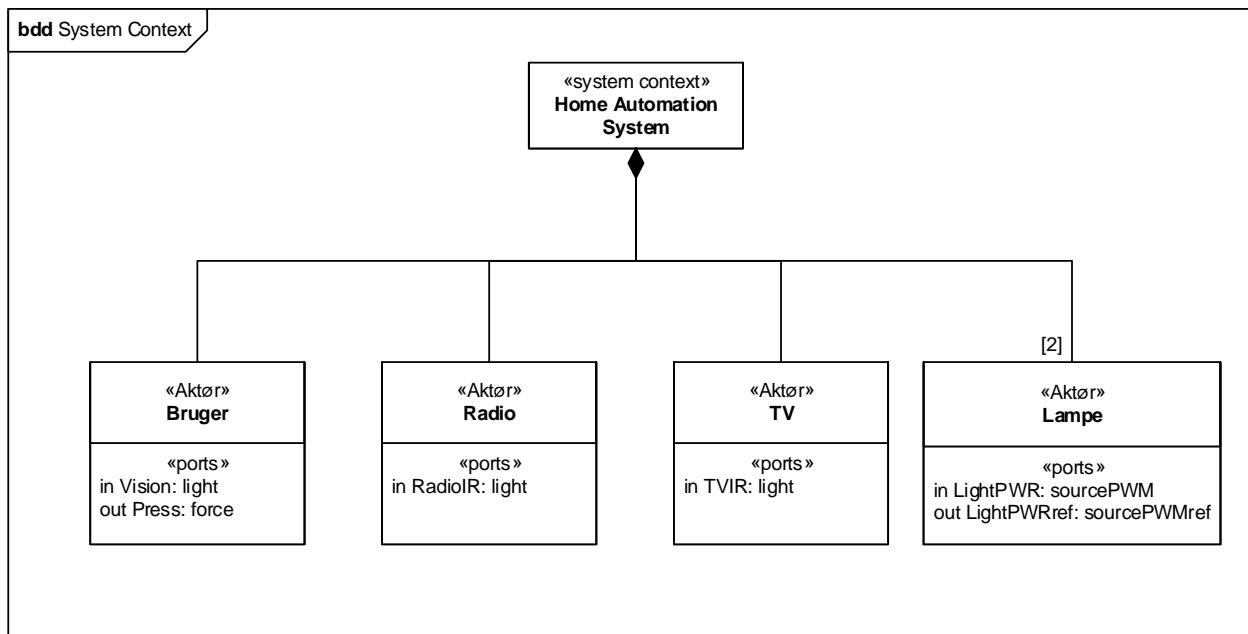
4.2 SysML diagrammer

Beskrivelse af samtlige signaler findes i Tabel 11 på side 23.

4.2.1 BDD for system kontekst

I Figur 5 vises konteksten for systemet, som består af de elektriske enheder samt brugeren af systemet. Yderligere vises porte på aktørerne, som agerer med systemet. Bruger blokken beskriver den person der interagerer med systemet. Radio blokken er en radio med mulighed for styring via en IR fjernbetjening, TV blokken er tilsvarende. De to lampe blokke er to 5V lyskilder, som kan dimmes via pulsbreddemodulation.

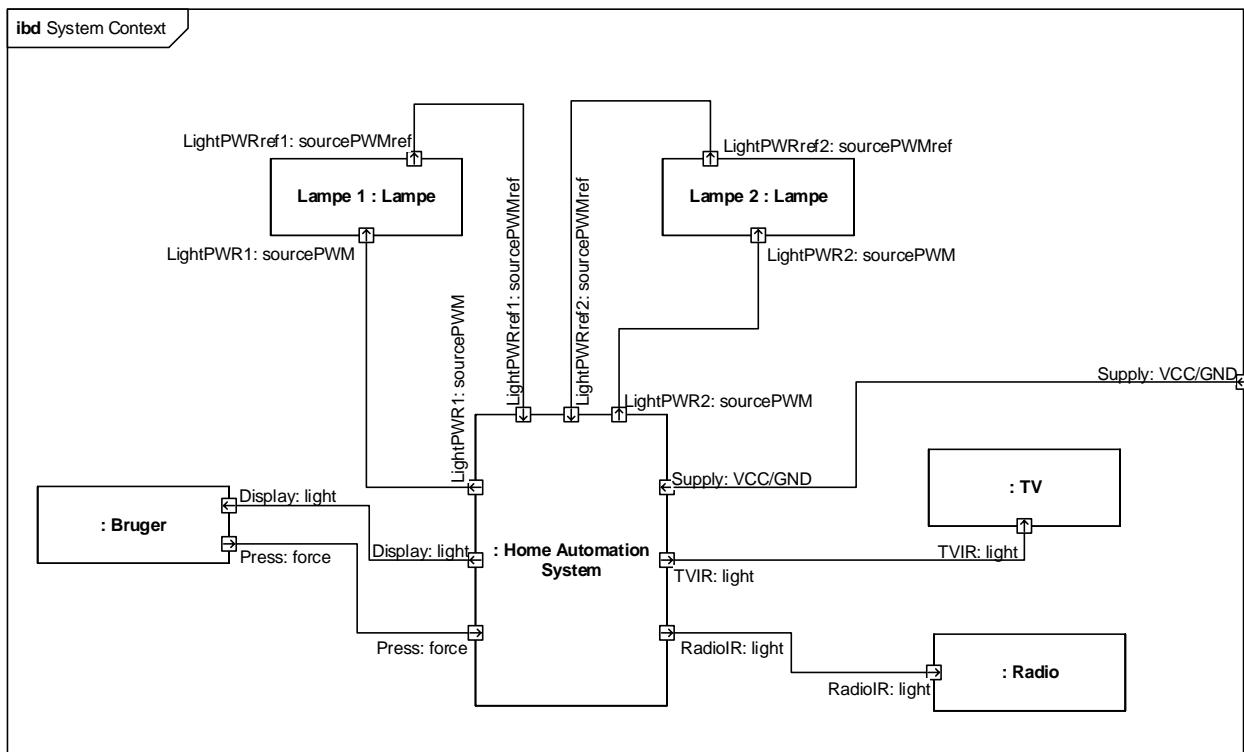
Bemærk at ingen blokke eller enheder forsyner fra 18VAC nettet, da dette kun bruges til X.10 kommunikation. Der benyttes derimod eksterne spændingsforsyninger.



Figur 5: BDD for system konteksten

4.2.2 IBD for system kontekst

I Figur 6 vises systemets eksterne forbindelser til de øvrige blokke omkring systemet.



Figur 6: IBD for system konteksten

Det antages at der til TV samt Radio allerede er tilført spændingsforsyninger, da disse enheder ikke er en del af systemet.

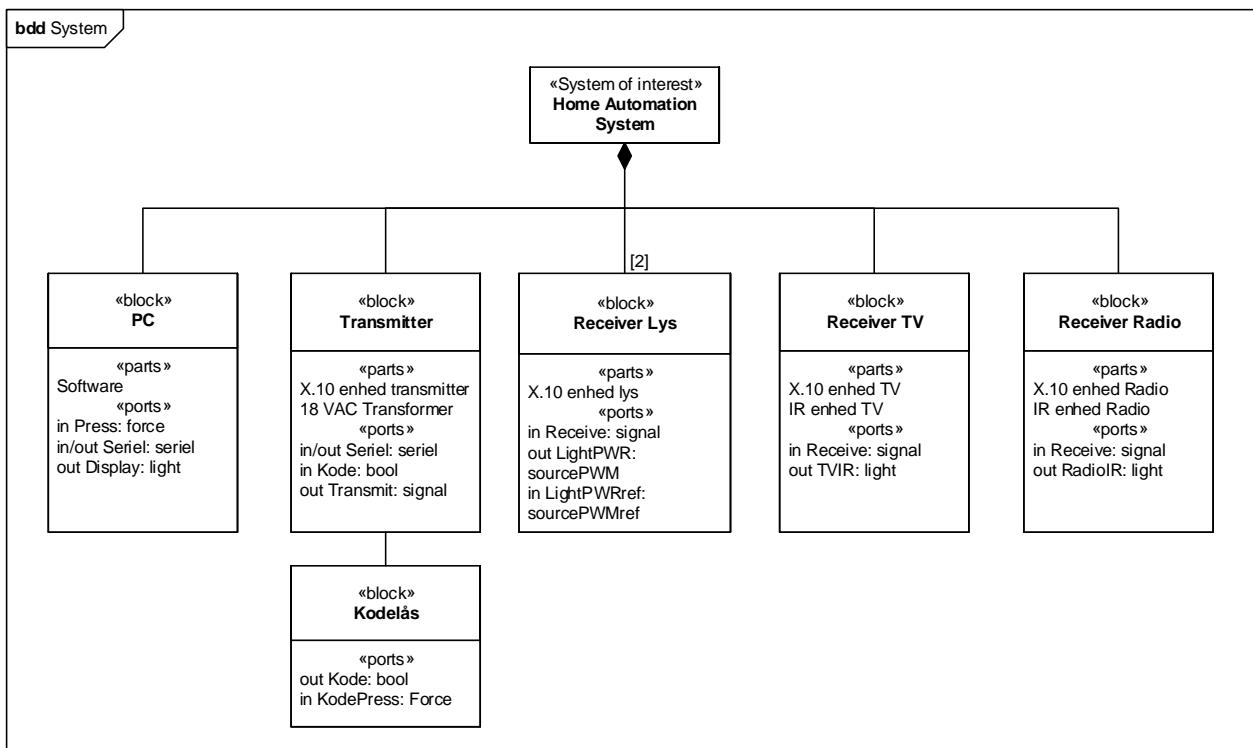
4.2.3 BDD for systemet

I Figur 7 vises et BDD over systemet i sin helhed. Diagrammet viser overordnede blokke i systemet, samt deres ports og parts.

PC blokken er brugerens grænseflade til systemet. Transmitterblokken modtager information fra PC softwaren, såfremt kodelåsen er aktiveret, og sender kommandoer til Receiver-blokkene.

Receiver-blokkene består kun af receivere, dvs. selve TV'et, Radioen og lamperne ikke er en del af systemet.

Der er for overskuelighedens skyld valgt ikke at indskrive systemets porte i «System of interest» blokken. Disse fremgår af Figur 6.

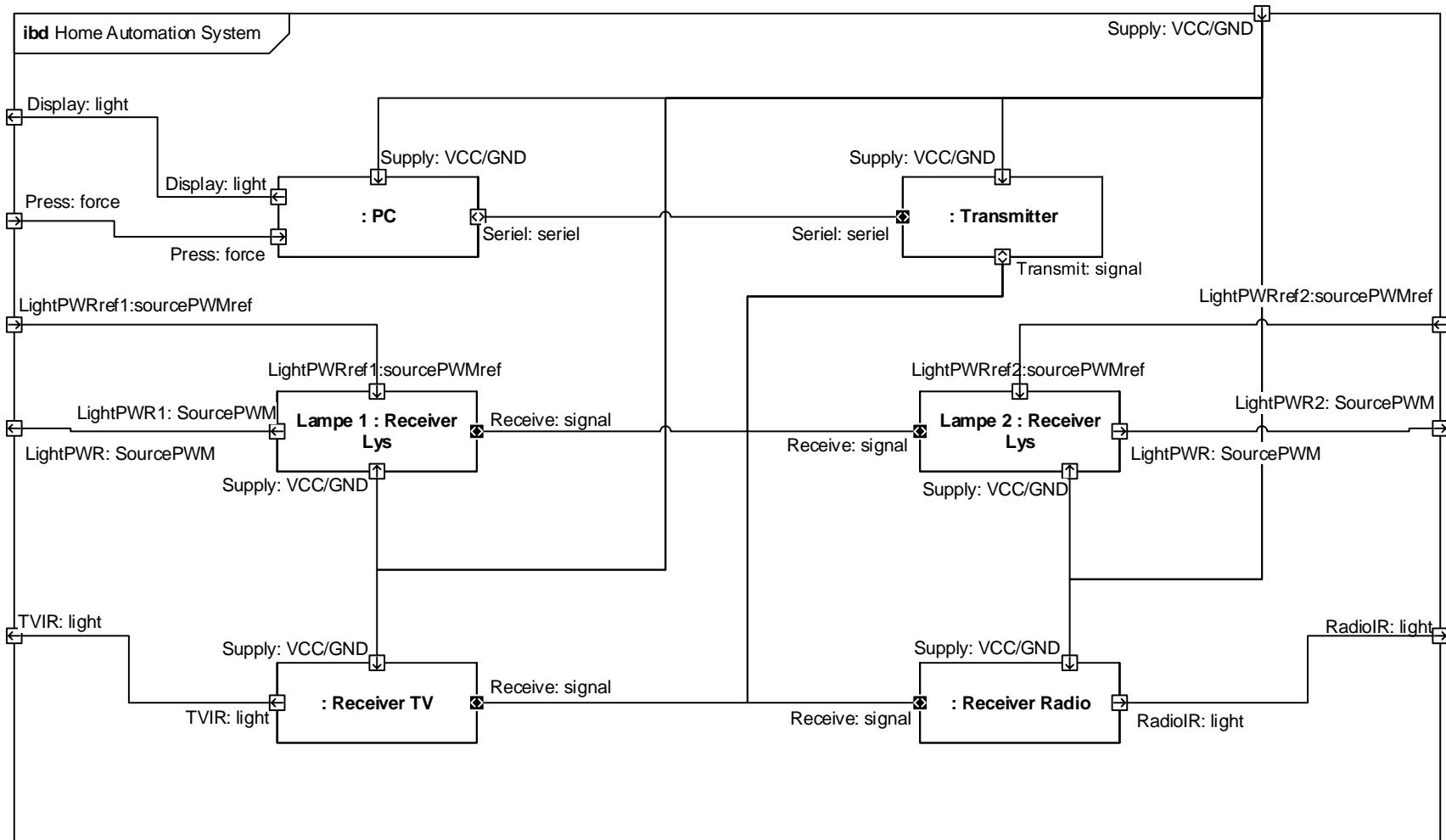


Figur 7: BDD diagram for systemet

4.2.4 IBD for systemet

I Figur 8 vises de interne forbindelser mellem blokke i systemet. Selve lampen får sin forsyning gennem *LightPWR: SourcePWM* og har en reference fra *LightPWRref: SourcePWMref*. Signaltypen *signal* har sin reference via *VCC/GND*.

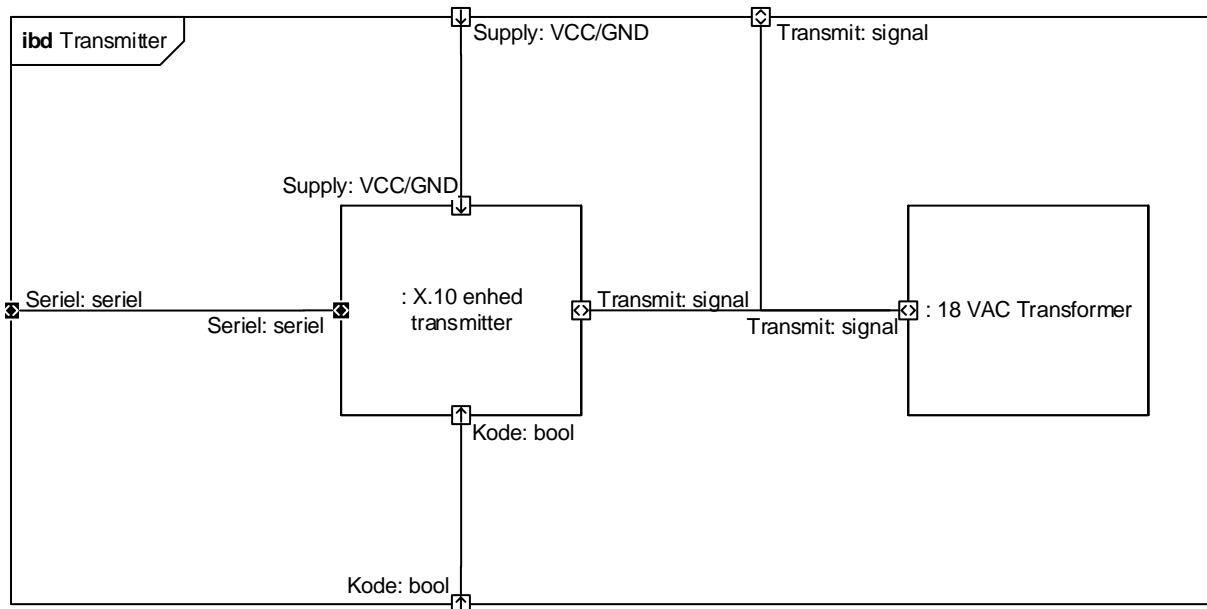
19



Figur 8: IBD diagram for Home Automation systemet

4.2.5 IBD for Transmitter-blokken

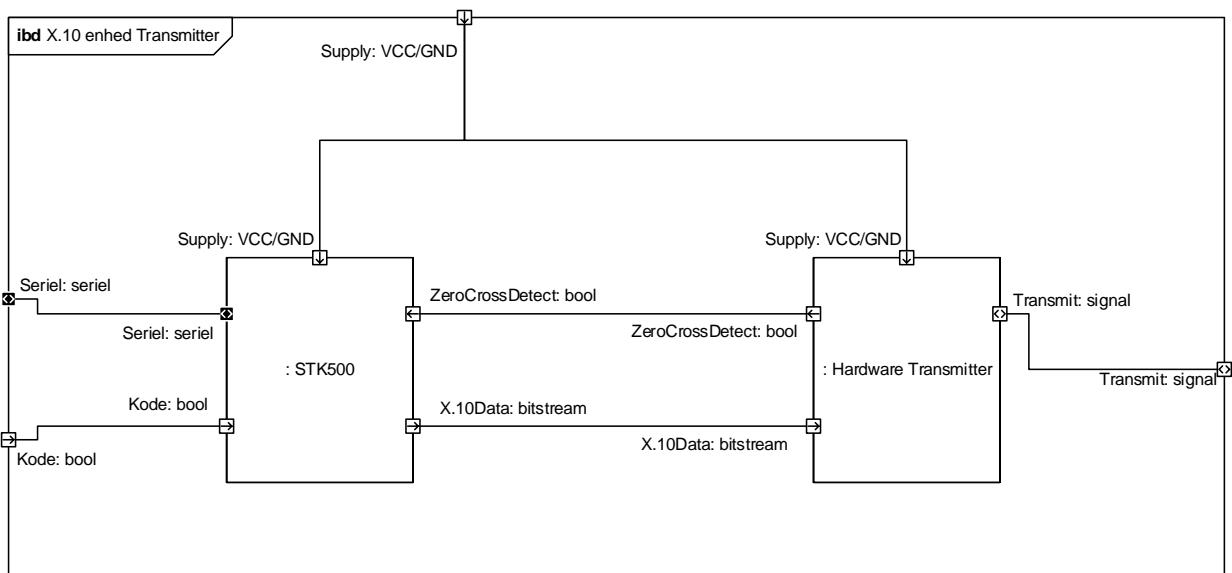
I Figur 9 vises de interne forbindelser mellem parts i Transmitter-blokken.



Figur 9: IBD for Transmitter

4.2.6 IBD for X.10-enhed i transmitter-blokken

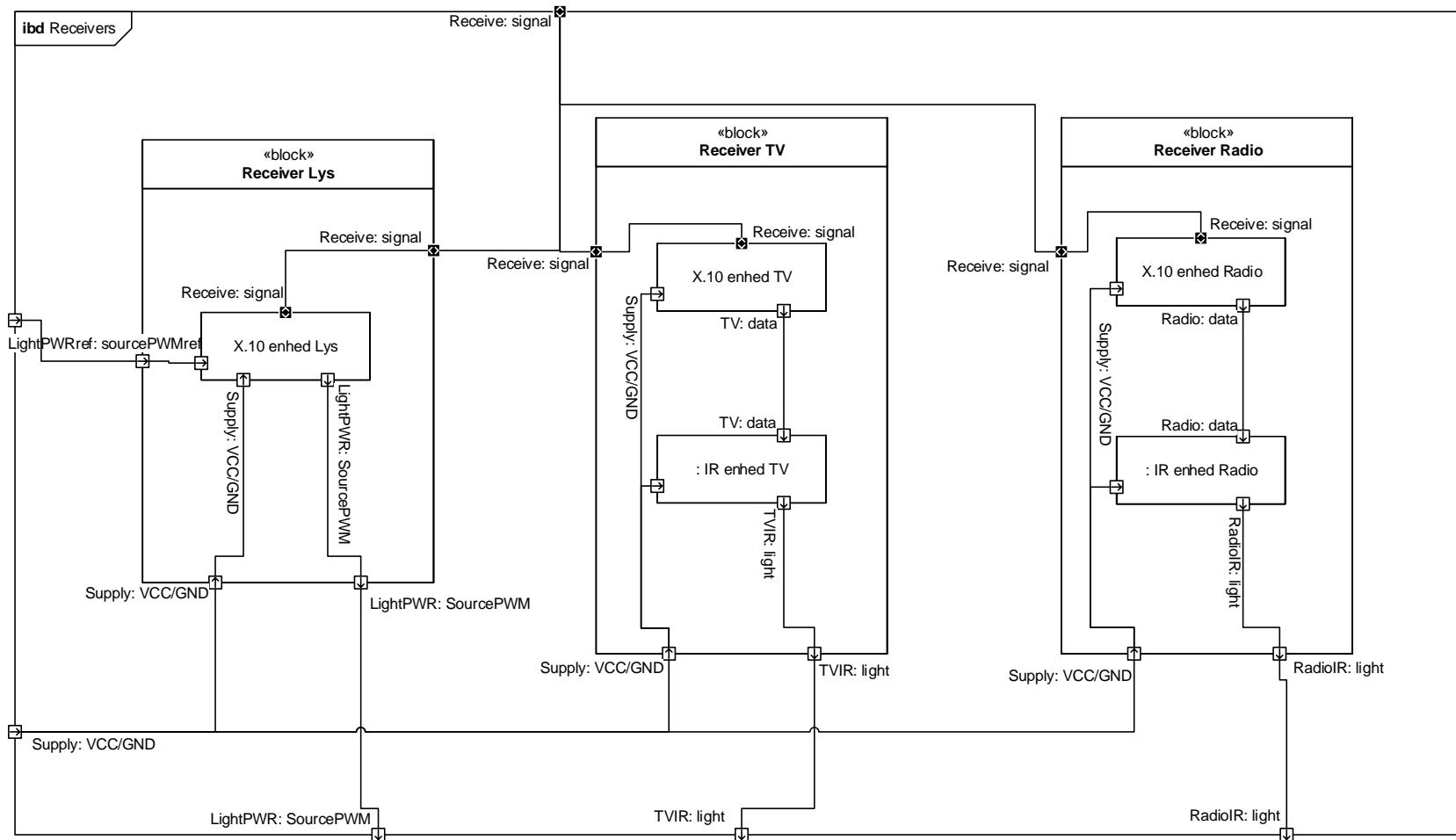
I Figur 10 vises forbindelser mellem STK500 og øvrig hardware i blokken, og dermed grænsefladen mellem software og hardware.



Figur 10: IBD for X.10 enheden i transmitteren

4.2.7 IBD for receivers

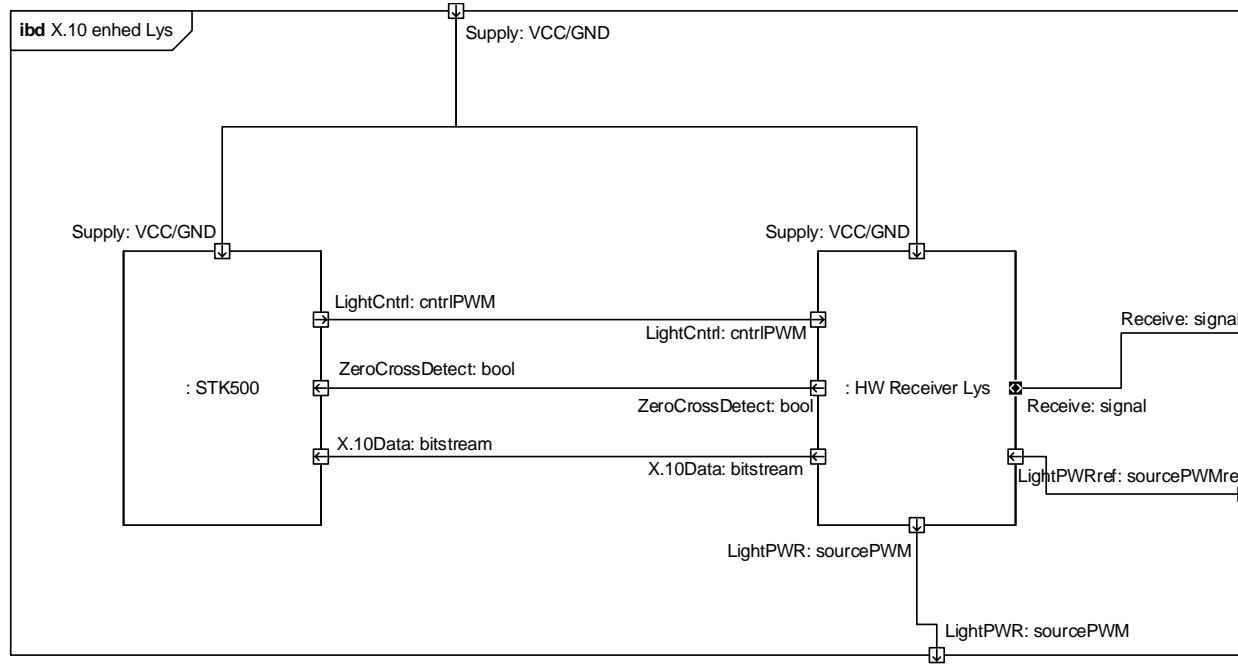
I Figur 11 vises samtlige typer af receivers i systemet. Af hensyn til overskuelighed vises der kun én instans af Receiver Lys, selvom systemet reelt indeholder to, jf multiplicitet i Figur 7 side 18.



Figur 11: IBD for receivers

4.2.8 IBD for X.10 enhed i lys-blokken

I Figur 12 vises forbindelser mellem STK500 og øvrig hardware i blokken, og dermed grænsefladen mellem software og hardware.



Figur 12: IBD for X.10 enheden i transmitteren

NOTE: Bemærk at projektdokumentationen indtil dette punkt har indeholdt det komplette system, men fra dette punkt behandles TV- og Radio-delen af systemet ikke. Dvs. at en del af signalerne ikke er beskrevet samt at videre dokumentationer og design ikke omfatter TV- og Radio-delen.

4.3 Signalbeskrivelser

4.3.1 Signaltyper

Grænseværdierne for samtlige signaler, som har kontakt med STK500 er defineret ud fra tolerancerne på ATMega32's porte [8].

Signaltype	Funktion	Område	Kommentar
bitstream	Serie af 1'er og 0'er	HIGH: $3.1 - 5.4V$, LOW: $-0.4 - 0.9V$	1 = Tilstedeværelse af $120kHz$ signal, som går fra LOW til HIGH. 0 = LOW
cntrlPWM	Firkantformet PWM signal på 1 kHz til regulering af lysstyrke	$0.0 - 5.0V$ maks $10mA$.	PWM duty cycle kan variere i området $5\% - 95\%$ i trin af $10\% \pm 1\%$.
force	Brugerens input på PC		
light	Output på PC'ens skærm		
seriel	Kommunikation mellem PC og transmitter	jf. RS232	
signal	Sammensætning af $18VAC$ og X.10	$18VAC \pm 10\%$ [9]	$18VAC$ leveres fra transformeren og X.10 ($120kHz$ i nulgennemgange) leveres fra Transmitter-blokken.
sourcePWM	Firkantformet PWM spændingsforsyning på 1 kHz til at drive lys	$5.0V \pm 0.1V$	
sourcePWMref	Reference til lampe	$0.0V - (2.9V \pm 0.5V)$ med samme frekvens som cntrlPWM	
VCC/GND	DC spændingsforsyning og reference	VCC: $11.8 - 12.2V$, GND: $0.0V$	Signalet er en sammensætning af to forbindelser, VCC og stel.
bool	Kan være enten HIGH (1) eller LOW (0)	HIGH: $3.1 - 5.4V$, LOW: $-0.4 - 0.9V$	1 = HIGH, 0 = LOW

Tabel 11: Beskrivelse af samtlige signaler.

4.3.2 Grænseflader

- *ZeroCrossDetected: bool*

Skal toggle hver gang der registreres en nulgennemgang på *Transmit: Signal*.

- *Receive: Signal & Transmit: Signal*

Består af et 18VAC ved 50Hz, som for hver nulgennemgang til 1ms efter kan indeholde 120KHz signaler. Hvis der forefindes et 120KHz signal i en nulgennemgang, betragtes det som HIGH. Hvis der ikke forefindes et 120KHz signal, betragtes det som LOW.

- *X.10Data: bitstream*

Sigmalet er grænsefladen mellem STK500 og hhv. transmitter og receiver. HIGH på sigmalet svarer til at der forefindes 120kHz på *signal*. Når sigmalet bruges til at sende med skal det holdes HIGH i 1ms fra nulgennemgang.

- *Kode: bool*

Er LOW hvis koden er indtastet korrekt og HIGH hvis den ikke er indtastet korrekt.

- *Seriell: seriell*

BAUD rate på 9600, 8 bit, 1 startbit, 2 stopbit og ingen paritet.

- *LightCntrl: CntrlPWM*

Firkantformet PWM styresignal til regulering af *LightPWR: SourcePWM*.

- *LightPWR(1 & 2): SourcePWM*

Firkantformet PWM forsyningssignal til at drive lampen. Duty cycle samt frekvens for *LightPWR: SourcePWM* og *LightCntrl: CntrlPWM* skal være ens.

- *LightPWRref(1 & 2): SourcePWMref*

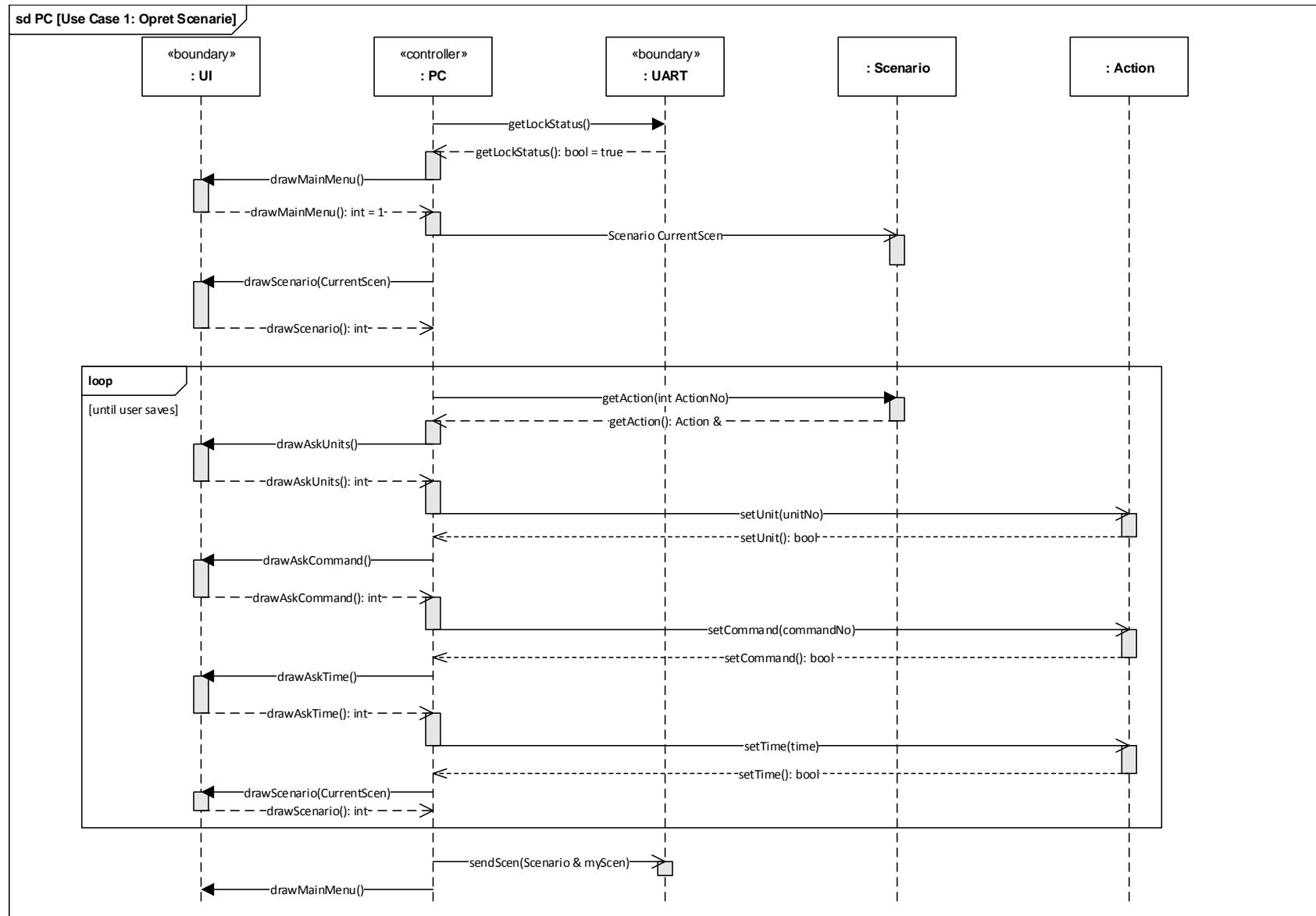
Reference til *LightPWR: SourcePWM*. Samme frekvens og duty cycle som *LightPWR: SourcePWM*.

4.4 Softwarearkitektur

For samtlige diagrammer gælder det at prækonditioner til de enkelte UC er medtaget.

4.4.1 Sekvensdiagram for PC [Use Case 1: Opret Scenarie]

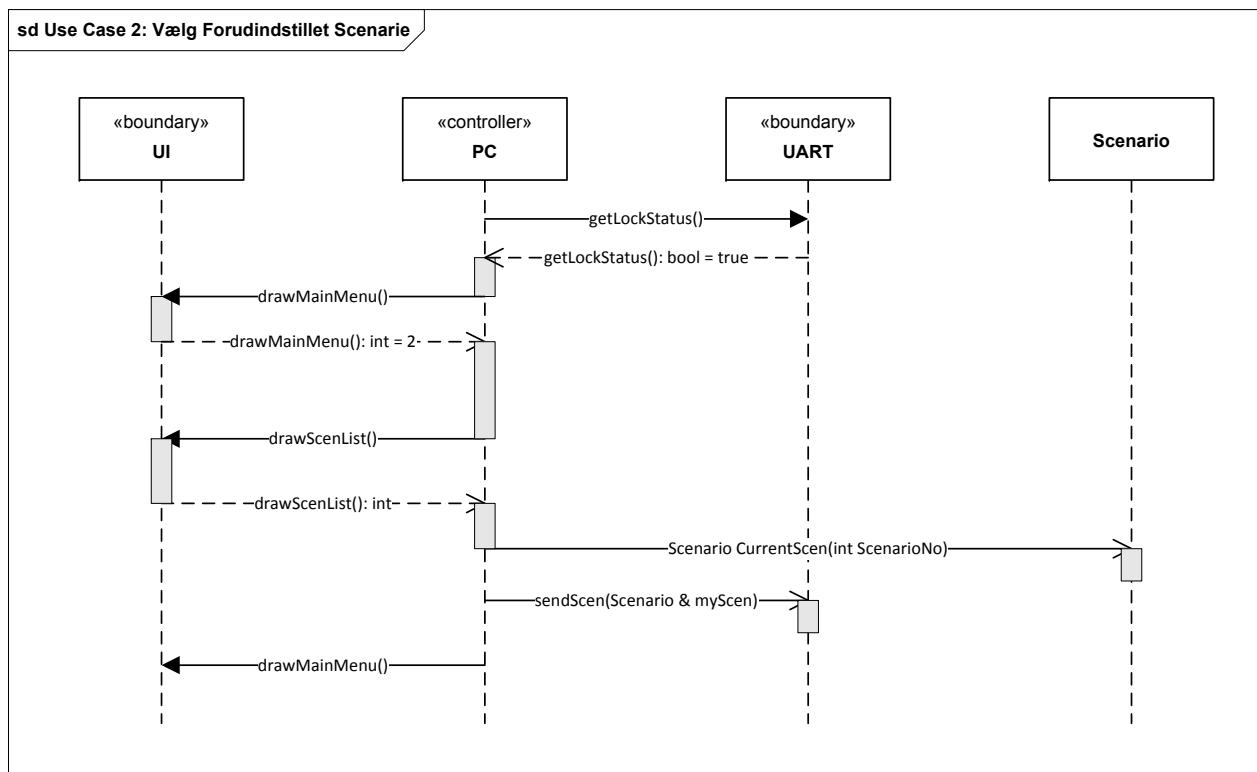
Diagrammet i Figur 13 viser sekvenser for PC'en ved gennemgang af hovedscenariet i UC1. Når der oprettes et objekt af klassen Scenario, oprettes det automatisk med 20 tomme aktioner.



Figur 13: Sekvensdiagram for PC [Use Case 1: Opret Scenarie]

4.4.2 Sekvensdiagram for PC [Use Case 2: Vælg Scenarie]

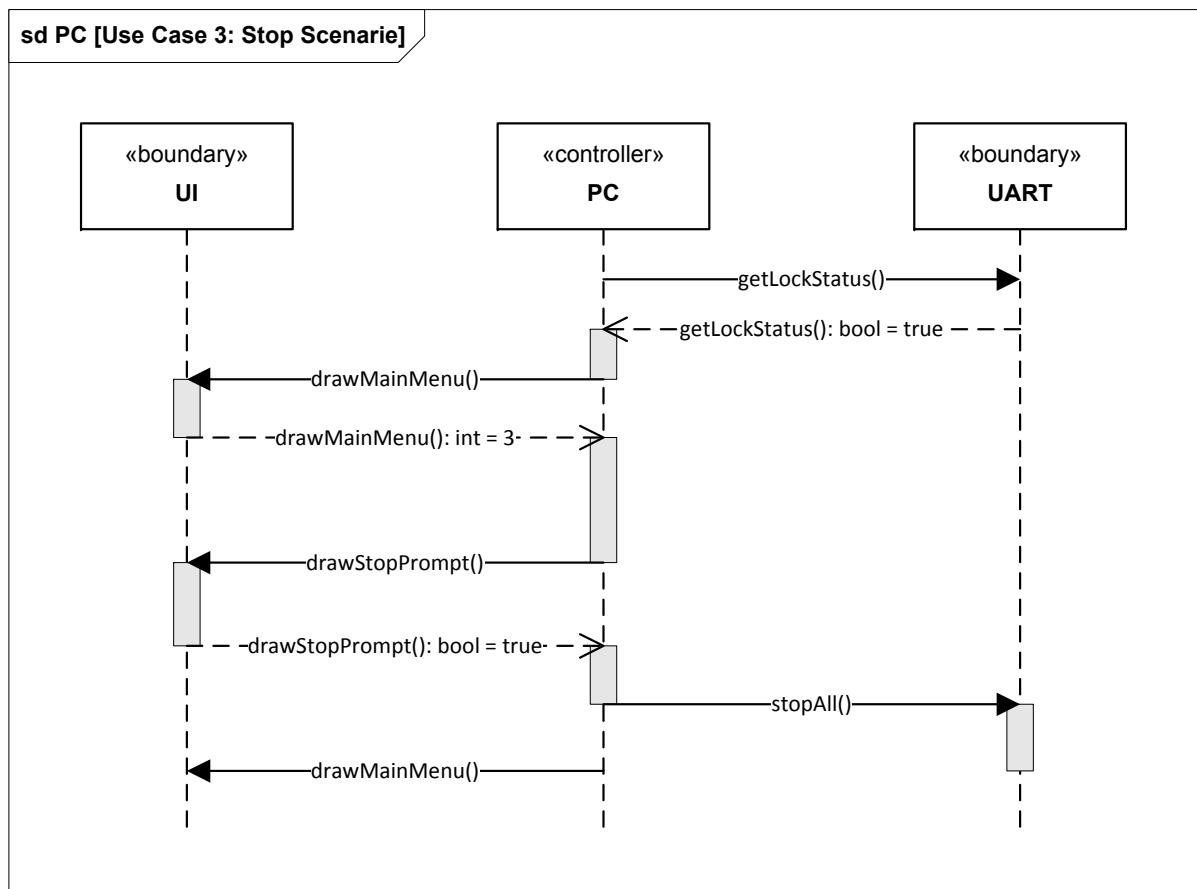
Diagrammet viser sekvenser for PC'en ved gennemgang af hovedscenariet i UC2.



Figur 14: Sekvensdiagram for PC [Use Case 2: Vælg Scenarie]

4.4.3 Sekvensdiagram for PC [Use Case 3: Stop Scenarie]

Diagrammet viser sekvenser for PC'en ved gennemgang af hovedscenariet i UC3.

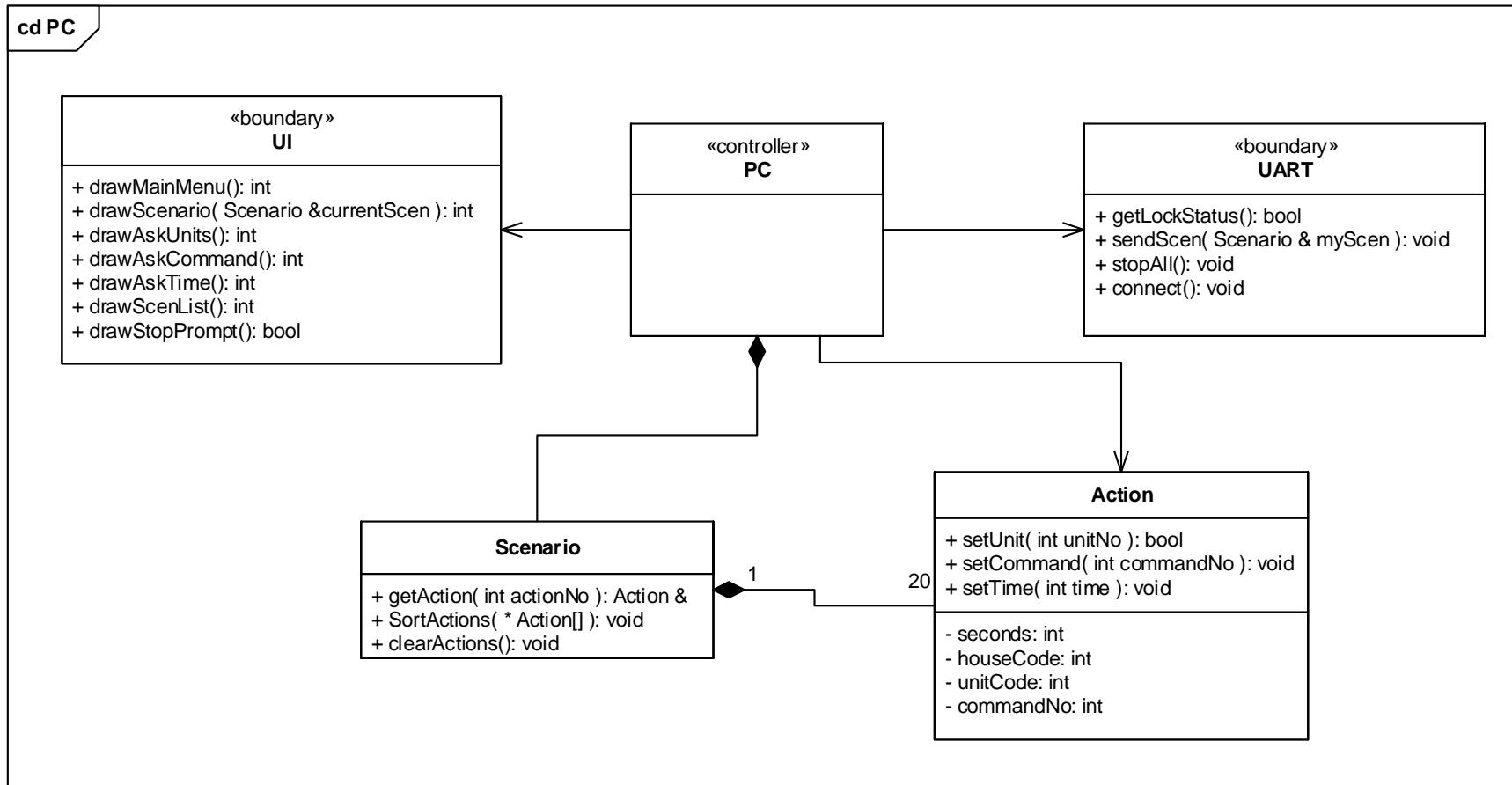


Figur 15: Sekvensdiagram for PC [Use Case 3: Stop Scenarie]

4.4.4 Klassediagram for PC

Diagrammet viser klassediagram for den software, der ligger på PC'en med domæne-, controller- og boundary klasser. Boundaryklassen UI har til formål at formidle kommunikation mellem bruger og controller klassen PC. Boundary klassen UART har ansvar for at kommunikere mellem controller klassen PC og transmitterblokken. Domæneklassen Scenario indeholder op til 20 objekter af domæneklassen Action, der indeholder informationer om aktionen.

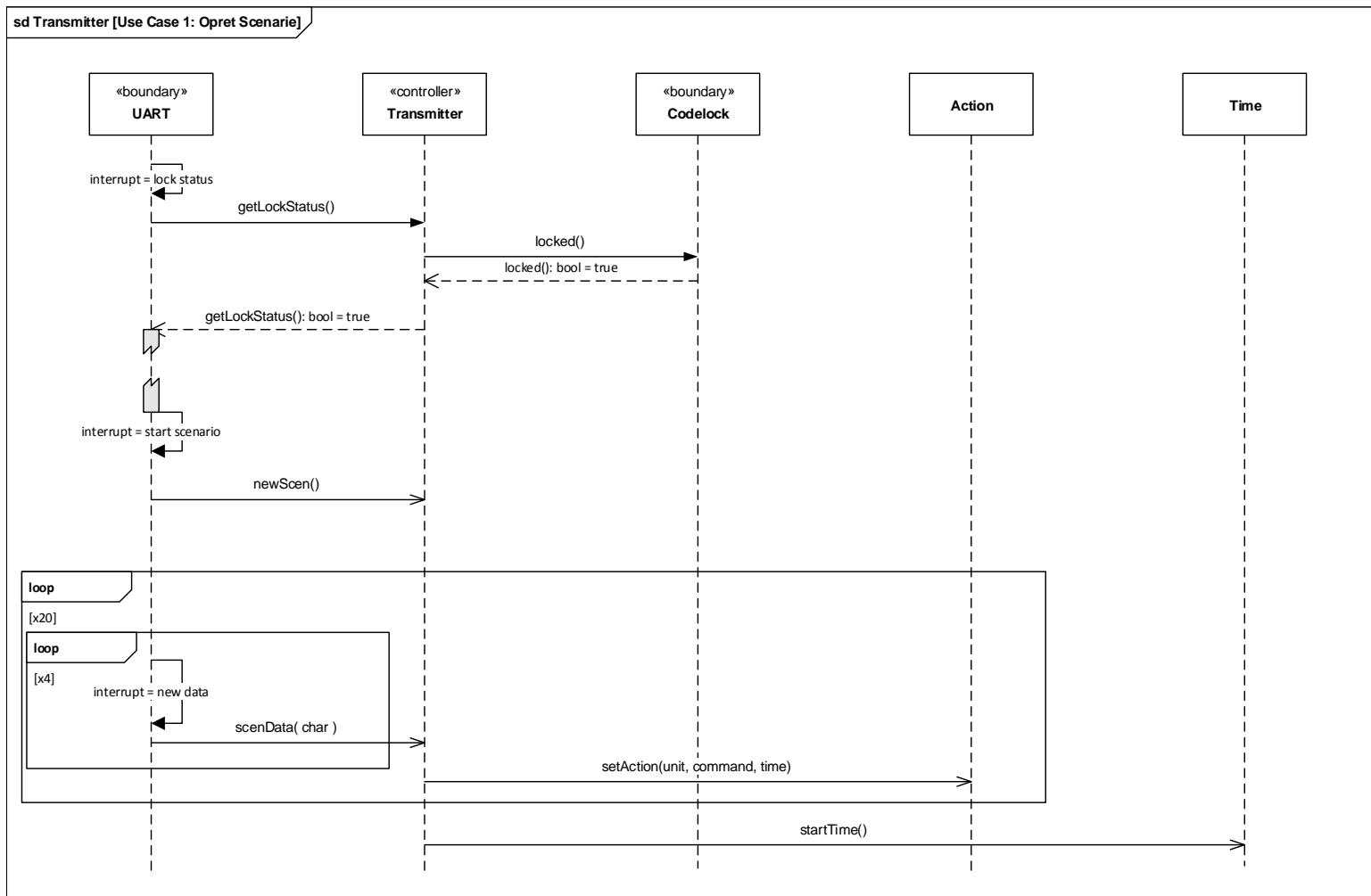
28



Figur 16: Klassediagram for PC.

4.4.5 Sekvensdiagram for Transmitter [Use Case 1: Opret Scenarie]

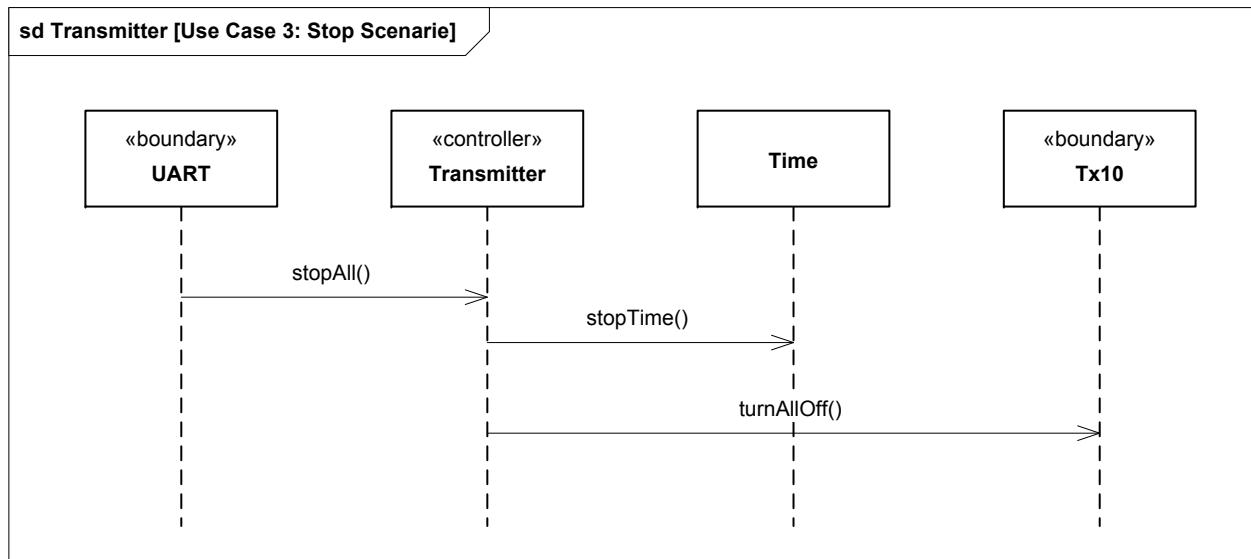
Diagrammet viser sekvenser for transmitteren ved gennemgang af hovedscenariet i UC1.



Figur 17: Sekvensdiagram for transmitter [Use Case 1: Opret Scenarie]

4.4.6 Sekvensdiagram for Transmitter [Use Case 3: Stop Scenarie]

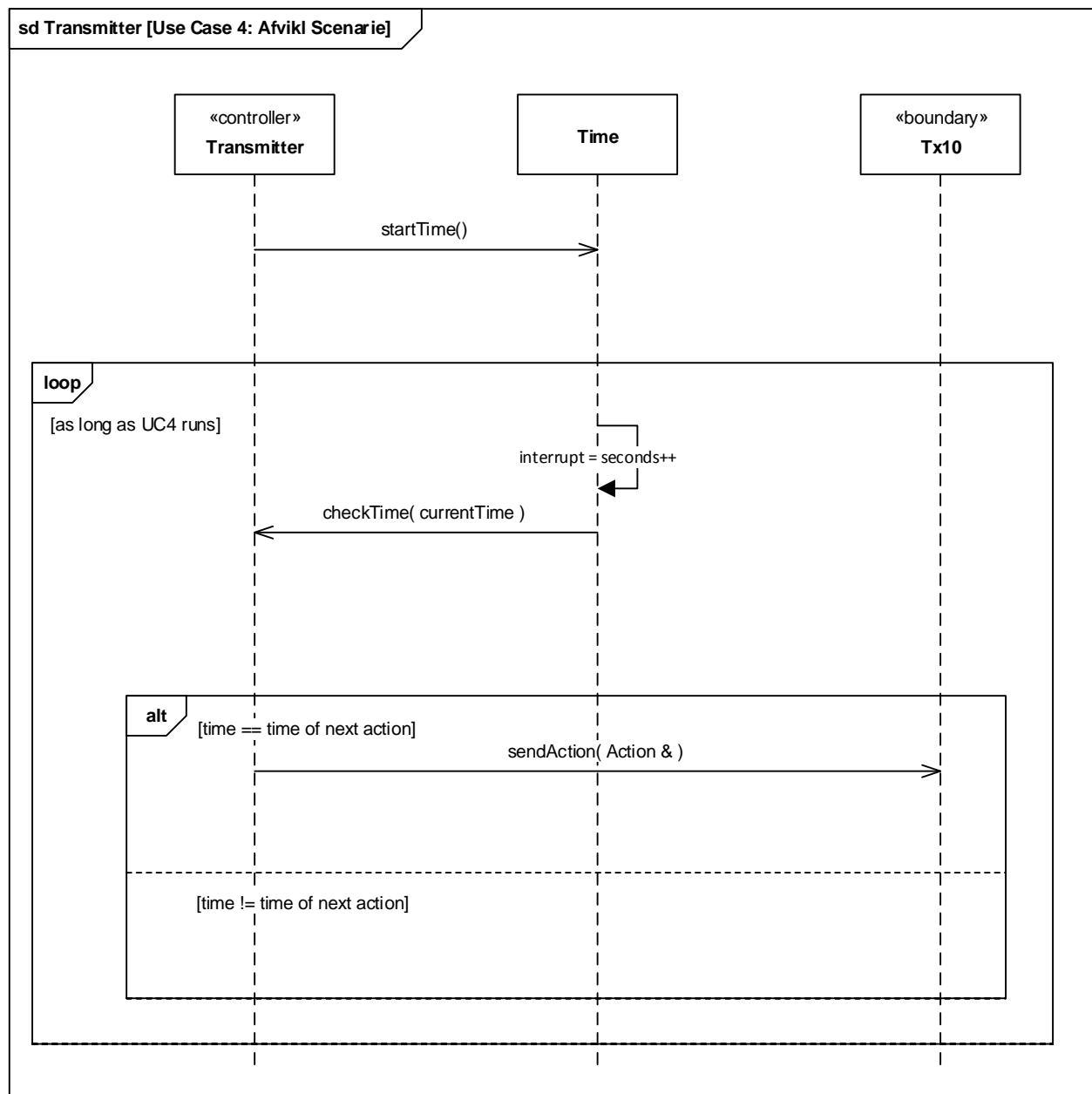
Diagrammet viser sekvenser for transmitteren ved gennemgang af hovedscenariet i UC3.



Figur 18: Sekvensdiagram for transmitter [Use Case 3: Stop Scenarie]

4.4.7 Sekvensdiagram for Transmitter [Use Case 4: Afvikl Scenarie]

Diagrammet viser sekvenser for transmitteren ved gennemgang af hovedscenariet i UC4.

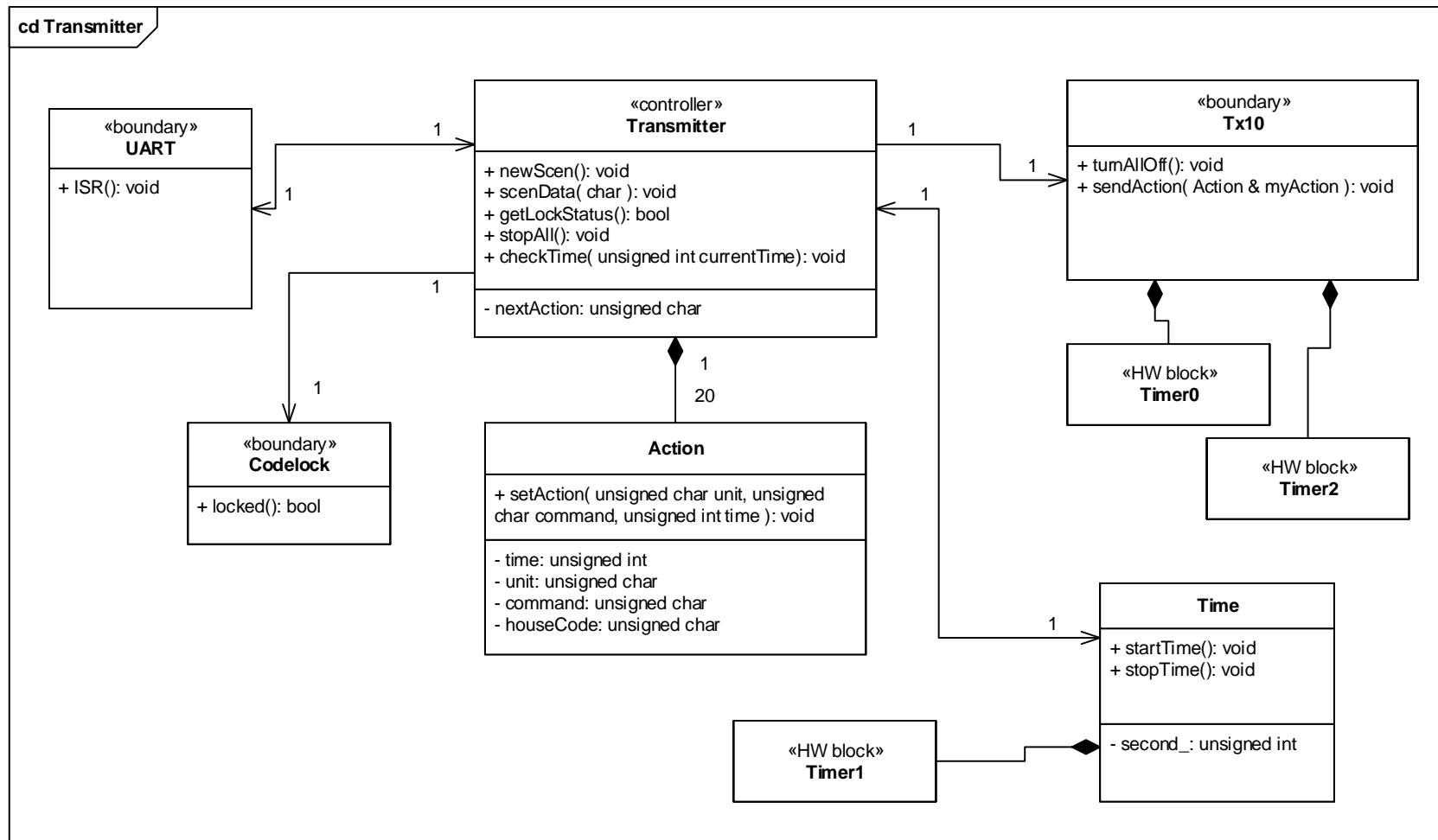


Figur 19: Sekvensdiagram for transmitter [Use Case 4: Afvikl Scenarie]

4.4.8 Klassediagramm for Transmitter

Klassediagramm for transmitter.

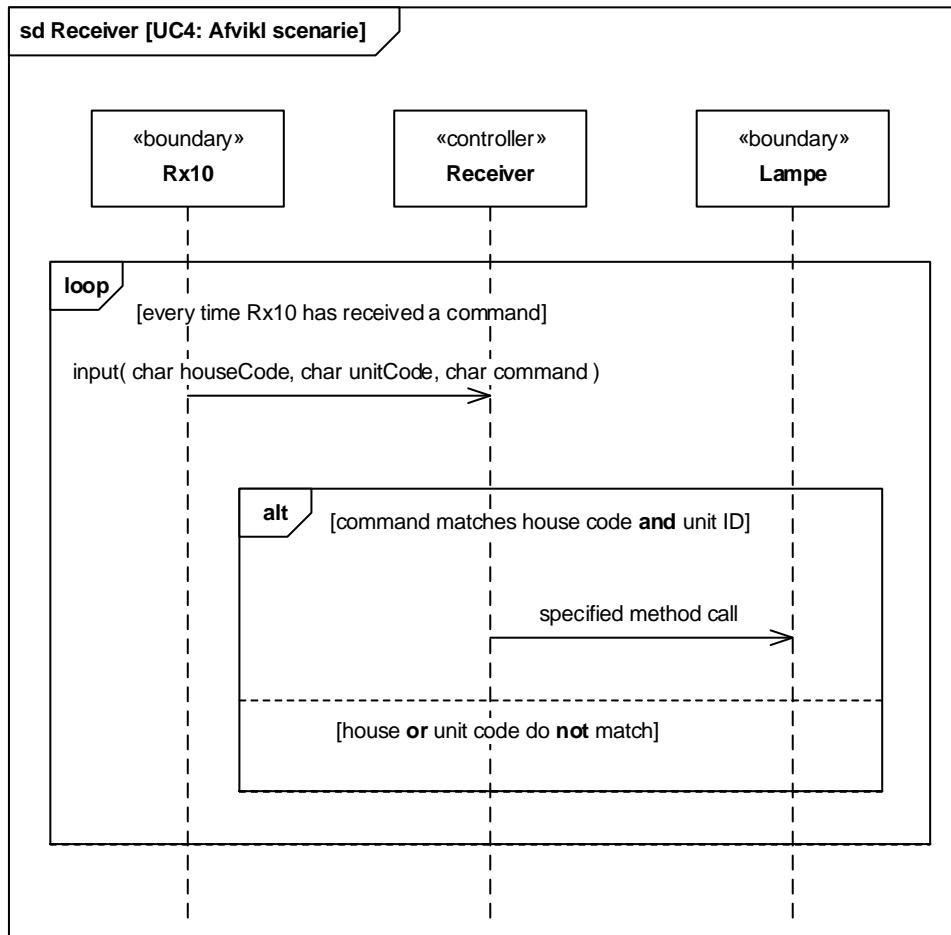
32



Figur 20: Klassediagramm for transmitter

4.4.9 Sekvensdiagram for Receiver [Use Case 4: Afvikl Scenarie]

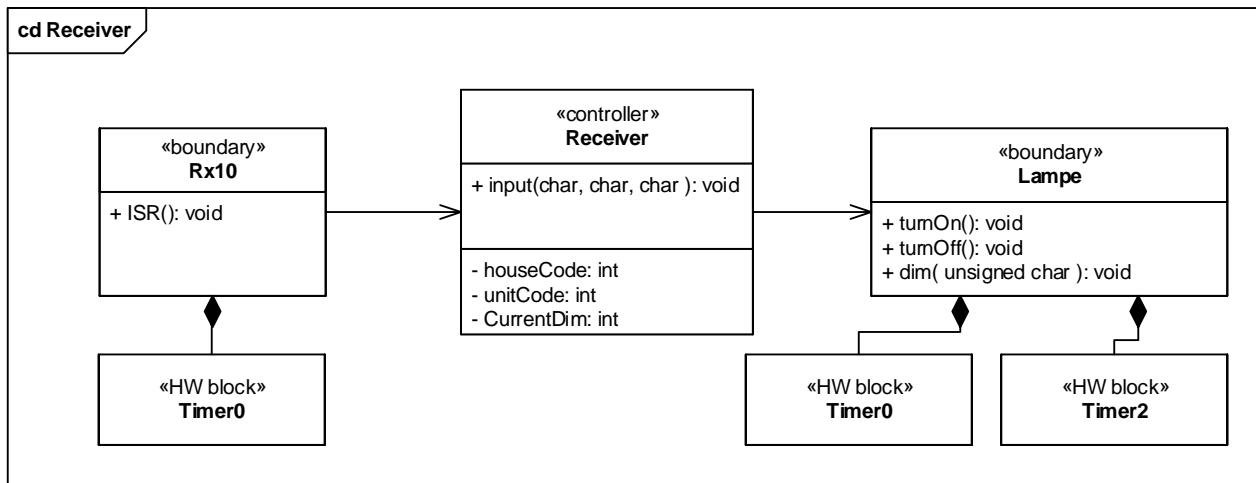
Diagrammet viser sekvenser for receiveren ved gennemgang af hovedscenariet i UC4. Rx10 bufferen kan forklares ved en buffer, der gemmer de seneste 8 nulgennemgange fra X.10. *specified method call* er et kald af en af metoderne i klassen Lampe, se Figur 22.



Figur 21: Sekvensdiagram for transmitter [Use Case 4: Afvikl Scenarie]

4.4.10 Klassediagram for Receiver

Diagrammet viser klassediagram for den software, der ligger på receiveren med controller- og boundary klasse(r). Boundary klassen Rx10 har til formål at fortolke information fra hardware receiver blokken og gør det tilgængeligt for controller klassen Receiver. Boundary klassen Lampe har til formål at formidle kommunikation mellem controller klassen Receiver og hardware receiver blokken.



Figur 22: Klasse diagram for receiver

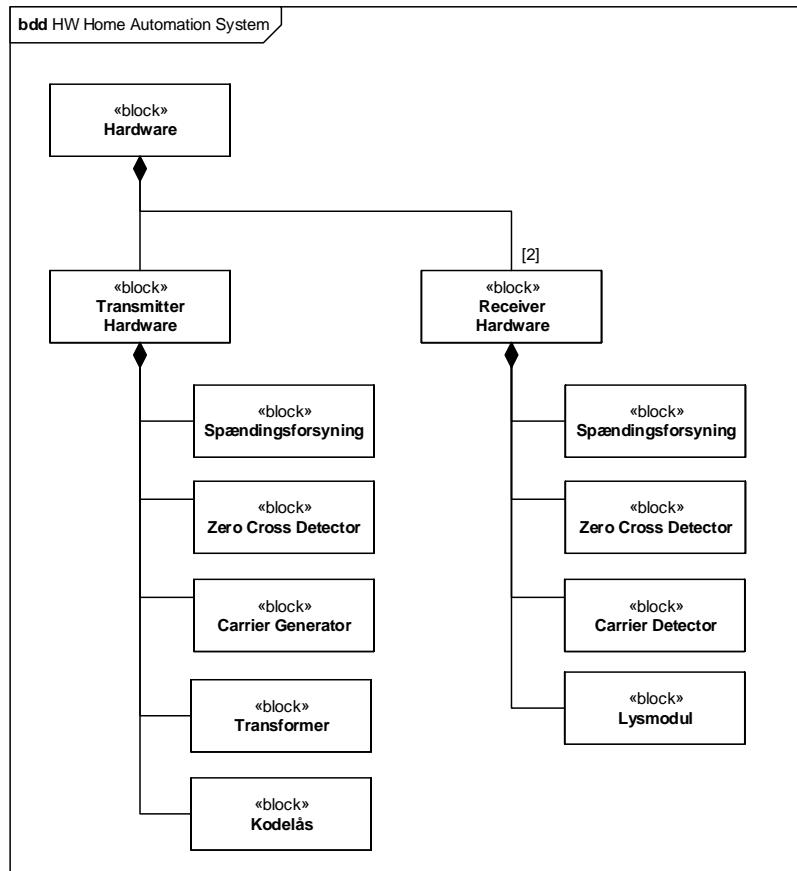
5 Hardware Design

5.1 Version

Dato	Version	Initialer	Ændring.
01. december	1	LS	Første version af dokumentet.
02. december	2	HBJ	Diverse design ændringer.
15. december	3	LS	Små rettelser. Endelige version.

5.2 Blokopdeling

Systemets hardware er opdelt i en transmitter- og to receiverdele. De underblokke der optræder flere steder i systemet er ens, og beskrives derfor kun én gang. For samtlige diagrammer gælder, at de resterende ikke anvendte inputs på IC'er, der ikke er vist, er koblet til stel.



Figur 23: BDD-diagram over hardware

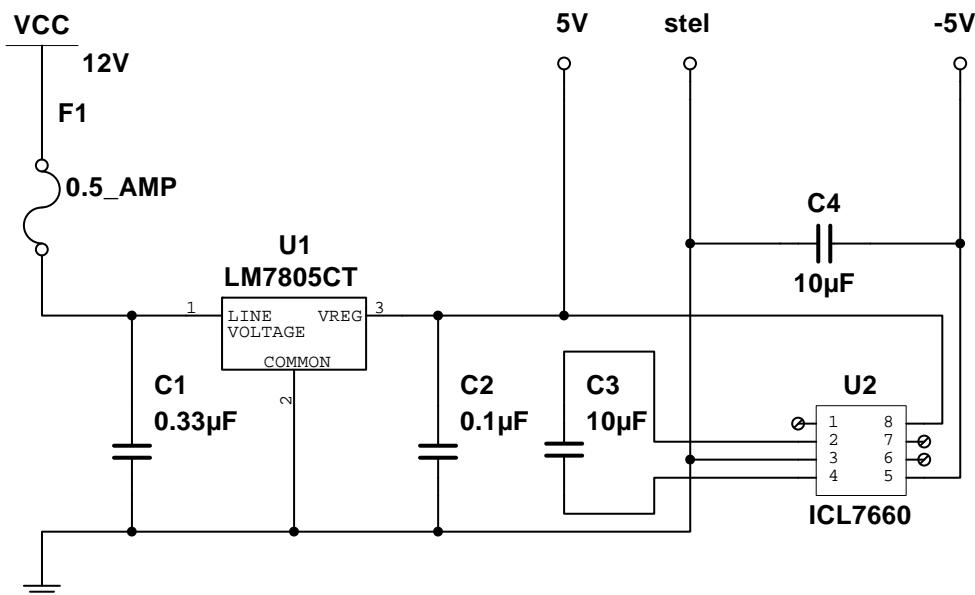
5.3 Transmitter

5.3.1 18V AC Transformer

Transformeren (8610AC) [9] transformerer 230VAC til 18VAC, således at det er muligt at simulere et elnet.

5.3.2 Spændingsforsyning (Morten)

Systemet skal forsynes med en 5V og en -5V spænding. Spændingsforsyningen forsyner med 12V DC fra en ekstern spændingskilde. Der er anvendt en positiv fastspændingsregulator (*LM7805*) [1] og en spændings-inverter(*ICL7660*) [2], og de er opsat jf. standardapplikationen i deres respektive datablade. 500mA er den den mindste tilgængelige sikring, derfor er denne valgt, da der ikke forventes en samlet strøm, der er større end dette i nogen af de tre spændingsforsyninger (transmitter og to receivere). Dette giver et forholdsvis stort spændingsfald over spændingsregulatoren, hvorfor den monteres med køleplade, for at undgå overophedning. Kredsløbets design er vist i Figur 24.



Figur 24: Spændingsforsyning

5.3.3 Kodelås (Lasse og Henrik)

Kodelåsen sender et *HIGH* signal ud på DE2-boardet's ben JP1-10 (*GPIO_0[9]*), når der ikke er indtastet tre rigtige koder. Så smart koderne er indtastet korrekt, vil dette signal skifte fra *HIGH* til *LOW*. Hvis der indtastes forkert kode 3 gange, vil kodelåsen gå i en permanent låsetilstand, som kræver manuel reset. Desuden er der fælles ground fra JP1-12 (*Ground*). Denne blok er lavet under en øvelse i faget Digital System Design, og er derfor ikke forklaret yderligere under dette afsnit.

5.3.4 Zerocross Detector (Morten og Philip)

Zerocross detectoren (Figur 25) har til formål at toggle ZeroCrossDetect signalet, når der registreres en nul-gennemgang på 18VAC - 50Hz nettet. Kredsløbet består af et højpasfilter, to dio-

der (1N4148) [3], der har til formål at begrænse spændingen til $0.7V$ og en operationsforstærker (TS912) [4]. Operationsforstærkeren giver et output alt efter om inputtet på plus-benet er højere eller lavere end inputtet på minus-benet, der er koblet til stel.

Da der vil være risiko for prel i nul-gennemgangene på udgangen af operationsforstærkeren, opbygges kredsløbet som i Figur 25, således at der fremkommer en hysterese. Tærskelværdierne for hysteresen sættes til $\pm 100mV$ således at den samlede hysterese bliver $200mV$. På baggrund af dette kan værdierne for R_2 og R_3 bestemmes.

Beregningerne er fortaget for det tilfælde, lige før en nedadgående nul-gennemgang finder sted.

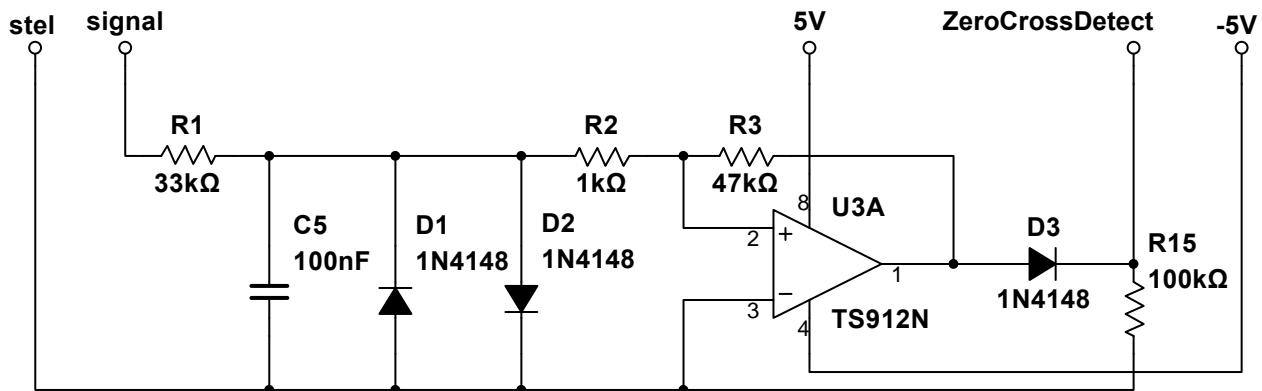
Spændingsfaldet over R_2 skal være $V_{R_2} = 100mV$

R_2 sættes til $1k\Omega$

Når operationsforstærkeren skal gå fra høj til lav er spændingen på udgangen $V_{out} = 5V$.

$$V_{R_2} = V_{out} \cdot \frac{R_2}{R_2 + R_3} \Rightarrow R_3 = \frac{R_2 \cdot V_{out}}{V_{R_2}} - R_2 = \frac{1k\Omega \cdot 5V}{100mV} - 1k\Omega = 49k\Omega \approx 47k\Omega$$

For at undgå negativ spænding på ZeroCrossDetect signalet, er der placeret en diode (D_3), som forhindrer dette. Der er et mindre spændingfald over dioden, men dette har ingen betydning. Derudover er der forbundet en pull-down modstand (R_{15}) efter dioden, for at sikre at signalet ligger stabilt på $0V$, når operationsforstærkeren leverer negativ spænding.



Figur 25: Zerocross detector

For lavpasfiltret R_1 og C_5 fås overføringsfunktionen:

$$T_v(s) \approx \frac{\frac{1}{C_5 \cdot s}}{\frac{1}{C_5 \cdot s} + R_1} = \frac{\frac{1}{R_1 C_5}}{s + \frac{1}{R_1 C_5}}$$

For lavpasfiltret sættes knækfrekvensen til:

$$\omega_{c1} = 2 \cdot \pi \cdot 50Hz = 100\pi rad/s$$

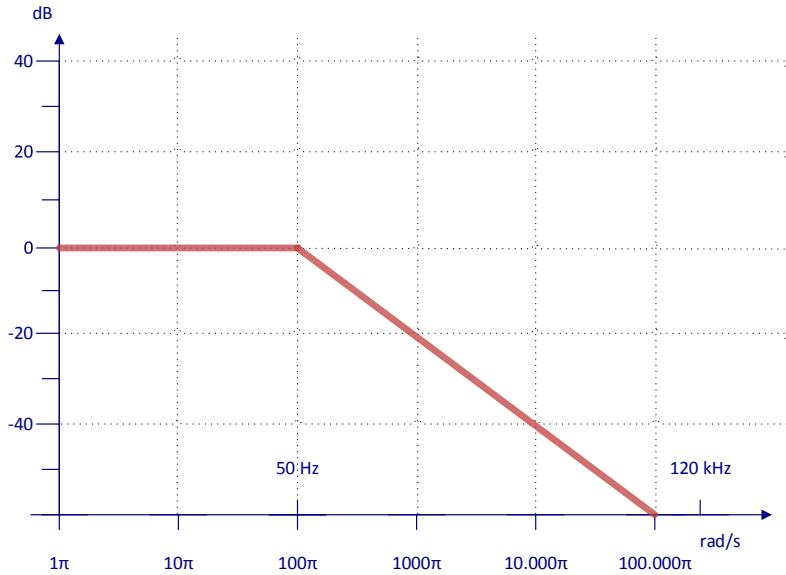
Det vides at knækfrekvensen ω_{c1} svarer til ledet $\frac{1}{R_1 C_5}$

C_5 sættes til at være $100nF$, derfor:

$$R_1 = \frac{1}{\omega_{c1} \cdot C_5} = \frac{1}{100\pi s^{-1} \cdot 100 \cdot 10^{-9} F} = 31.8k\Omega \approx 33k\Omega$$

Der opstilles et bodeplot med asymptotiske linjer for lavpasfiltret, Figur 26.

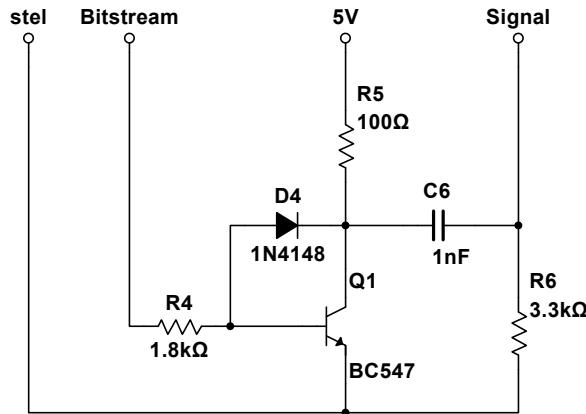
Gain i knækfrekvensen ω_{c1} er $-3dB$, men det har ikke nogen funktionel betydning.



Figur 26: Bodeplot med asymptotiske linjer for lavpasfilter

5.3.5 Carrier Generator (Philip og Lasse)

For at transmitteren uforstyrret kan sende et 120kHz signal på el-nettet, designes et kredsløb, der skal forhindre påvirkning fra $18\text{VAC} - 50\text{Hz}$ nettet på transmitter-hardwaren. Dette gøres vha. et højpasfilter, samt et transistor-kredsløb som vist i Figur 27.



Figur 27: Carrier generator

Komponenternes værdier er udregnet på følgende måde:

For at transistoren (*BC547*) [5] er i mætning, skal basestrømmen I_b være max. 20 gange lavere end kollektorstrømmen I_c , dvs. transistorens β -værdi ≤ 20 . Der vælges en kollektor modstand $R_5 = 100\Omega$, derefter kan vælges en basemodstand der max er 20 gange større end kollektormodstanden, for at ovenstående krav er overholdt. Grunden til dette er at *Bitstream* har samme spænding som de 5V på spændingsforsyningen når den er HIGH.

Der vælges derfor en basemodstand $R_4 = 1.8\text{k}\Omega$, hvilket gør at der kun bliver trukket $\frac{5\text{V}}{1.8 \cdot 10^3 \Omega} = 2.8\text{mA}$ fra *Bitstream*. Basestrømmen bliver trukket fra STK500, og det ønskes derfor ikke, at der bliver trukket en særlig stor strøm derfra. Der er placeret en diode (*D4*) mellem transistorens base-

og kollektor ben, med spærreretning mod kollektoren, for at hjælpe tranistoren med at aflade positiv opladning. Af samme årsag er transistoren *BC547* valgt, da denne har et mindre storage delay.

Højpasfilter

For højpasfilteret på Figur 27, kan følgende overføringsfunktion opstilles:

$$T_v(s) \approx \frac{R_6}{\frac{1}{C_6 \cdot s} + R_6} = \frac{R_6 \cdot s}{\frac{1}{C_6} + R_6} = \frac{s}{\frac{1}{R_6 \cdot C_6} + s} = \frac{\frac{1}{R_6 \cdot C_6}}{\frac{1}{R_6 \cdot C_6} + s} \cdot \frac{s}{\frac{1}{R_6 \cdot C_6}}$$

Frekvensen der skal transmitteres igennem filteret er:

$$\omega_{120k} = 120\text{kHz} \cdot 2\pi = 240000\pi\text{rad/s} = 7.54 \cdot 10^5\text{rad/s}$$

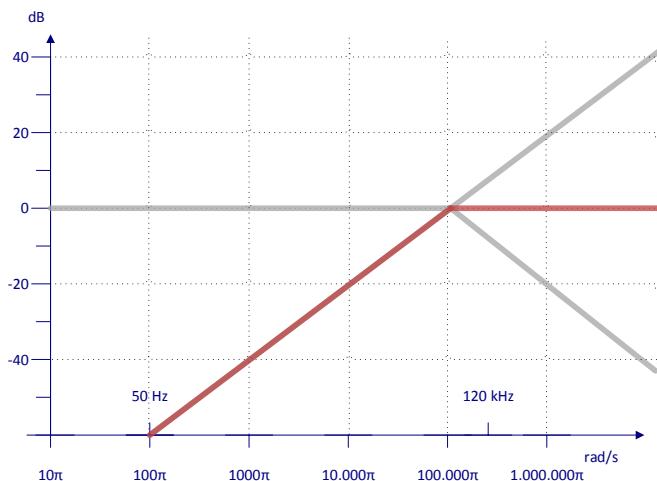
Frekvensen fra transformeren, som *ikke* skal kunne passere igennem filteret, er:

$$\omega_{50} = 50\text{Hz} \cdot 2\pi = 100\pi\text{rad/s} = 314.16\text{rad/s}$$

Med denne viden fastsættes knækfrekvensen ω_{c_2} til:

$$\omega_{c_2} = 2\pi \cdot 50\text{Hz} = 100 \cdot 10^3\pi\text{rad/s}$$

Derved bliver alle frekvenser under $100 \cdot 10^3\pi\text{ rad/s}$ dæmpt, og den endelige overføringsfunktion bliver således:



Figur 28: Bodeplot med asymptotiske linjer for højpasfilter

$$T_v(s) \approx \frac{100 \cdot 10^3\pi\text{rad/s}}{100 \cdot 10^3\pi\text{rad/s} + s} \cdot \frac{s}{100 \cdot 10^3\pi\text{rad/s}}$$

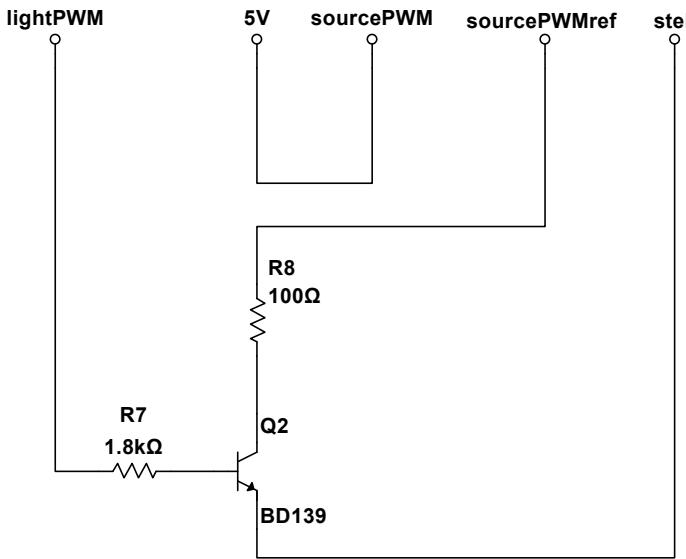
For at bestemme komponentværdierne, fastsættes vores kondensator til $C_6 = 1\text{nF}$. Overføringsfunktionen er omskrevet, så den består af standardled, og man kan derfor fastslå følgende:

$$\omega_{c_2} = \frac{1}{R_6 \cdot C_6} \Leftrightarrow R_6 = \frac{1}{\omega_{c_2} \cdot C_6} = \frac{1}{100 \cdot 10^3\pi\text{rad/s} \cdot 1 \cdot 10^{-9}\text{F}} = 3183\Omega \approx 3.3k\Omega$$

Der laves et bodeplot med asymptotiske linjer, Figur 28, for at se hvordan højpasfilteret vil dæmpe/forstærke forskellige frekvenser.

5.4 Receiver

5.4.1 Lysmodul (Morten)



Figur 29: Lysmodul

Transistoren (*BD139*) [6], Q_2 , er i mætning, og styrer derved PWM-signalen til lampen vha. PWM-signalen fra STK500. På den måde undgås det, at der trækkes strøm fra STK500 til lampen, og det giver mulighed for dæmpning af lysstyrken.

R_8 beregnes udfra strøm gennem lampen og spændingsfaldet over denne.

Som lampe anvendes en Kingbright 5mm gul diode (*L-53YD*) [7], som jf. databladet har et typisk spændingsfald på $2.1V$ og en anbefalet middelstrøm på $30mA$.

$$R_8 = \frac{5V - V_{diode}}{I_{diode}} = \frac{5V - 2.1V}{0.03A} = 97\Omega \approx 100\Omega$$

$$R_7 \leq \beta \cdot R_8 = 20 \cdot 100\Omega = 2000\Omega \approx 1.8k\Omega$$

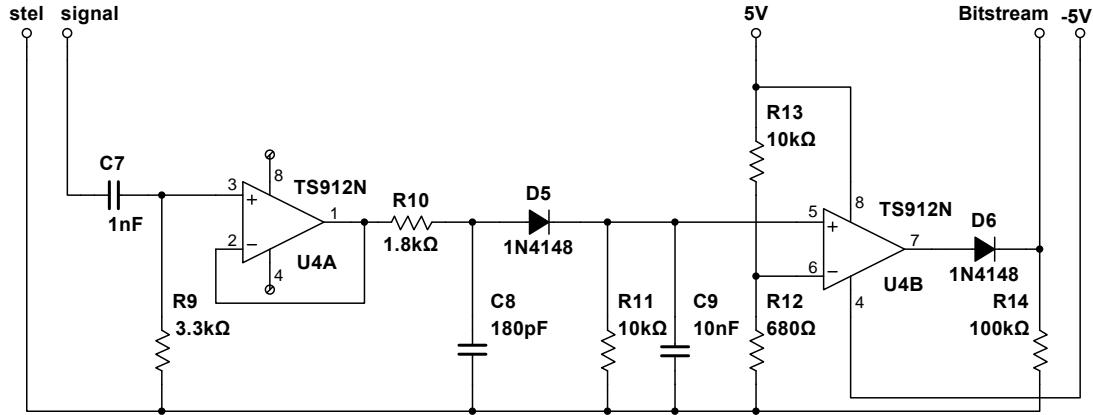
5.4.2 Carrier Detector (Morten, Henrik og Lasse)

Carrier detectoren kan deles op i 2 overordnede dele; et båndpasfilter og en envelope detector. Efter envelope detectoren er der desuden en operationsforstærker. Denne sætter udgangen til $5V$, når envelope detectoren udsender en spænding, der er højere end referencespændingen på operationsforstærkerens inverterende indgangsben, og $-5V$ når det er lavere. For at undgå negativ spænding på *bitstream* signalet, er der koblet en diode på udgangen af operationsforstærkeren.

Båndpasfilteret består hhv. af et højpasfilter (C_7 og R_9), en ikke-inverterende opAmp uden forstærkning (U_3) og et lavpasfilter (R_{10} og C_8).

Båndpasfilter

For at isolere $120kHz$ signalet, benyttes et båndpasfilter. Filteret skal lade $120kHz$ passere, og frasorterer alt andet, herunder $18VAC-50Hz$ nettet.



Figur 30: Carrier detector

Det ønskes at filteret får følgende knækfrekvenser:

$$\omega_{c_2} = 2\pi \cdot 50 \cdot 10^3 Hz = 100\pi \cdot 10^3 rad/s$$

$$\omega_{c_3} = 2\pi \cdot 500 \cdot 10^3 Hz = 1\pi \cdot 10^6 rad/s$$

Da kaskadereglen skal være opfyldt, indsættes et forstærkningsled med en forstærkning på 1. For båndpasfilteret på Figur 30, opstilles følgende overføringsfunktion:

$$T_v(s) \approx \underbrace{\frac{R_9}{\frac{1}{C_7 \cdot s} + R_9}}_{\text{Højpas}} \cdot \underbrace{\text{Gain}}_{\frac{1}{C_8 \cdot s}} \cdot \underbrace{\frac{\frac{1}{C_8 \cdot s}}{\frac{1}{C_8 \cdot s} + R_{10}}}_{\text{Lavpas}}$$

Overføringsfunktionen består af tre dele; et højpasfilter, en forstærkning og et lavpasfilter. Det kan ses at højpasfilteret er det samme som beskrevet i transmitter-afsnittet, og at knækfrekvensen for disse er ens.

$$C_7 = 1nF$$

$$R_9 = \frac{1}{\omega_{c_2} \cdot C_7} = \frac{1}{100 \cdot 10^3 \pi rad/s \cdot 1 \cdot 10^{-9} F} = 3183\Omega \approx 3.3k\Omega$$

Lavpasfiltrets overføringsfunktion omregnes til standard form.

$$T_{lavgpas}(s) \approx \frac{\frac{1}{C_8 \cdot s}}{\frac{1}{C_8 \cdot s} + R_{10}} = \frac{\frac{1}{R_{10} \cdot C_8}}{\frac{1}{R_{10} \cdot C_8} + s}$$

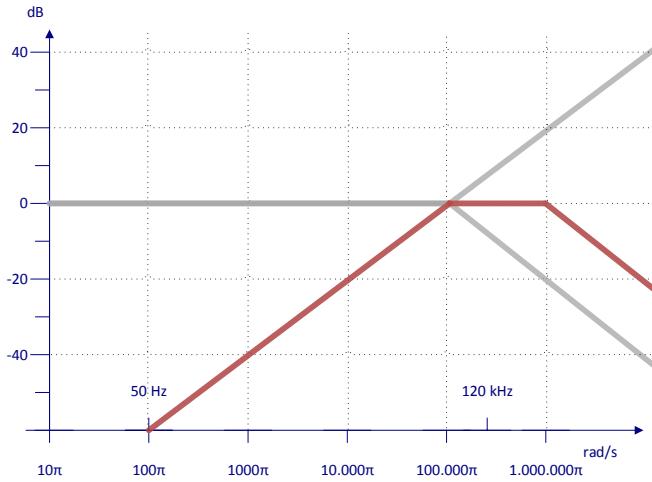
Da lavpasfilteret er på standardform, vides det at:

$$T_v(s) \approx \frac{\frac{1}{R_{10} \cdot C_8}}{\frac{1}{R_{10} \cdot C_8} + s} = \frac{\omega_{c_3}}{\omega_{c_3} + s}$$

Kondensatoren fastsættes til $C_8 = 180pF$

$$\omega_{c_3} = \frac{1}{R_{10} \cdot C_8} \Rightarrow R_{10} = \frac{1}{\omega_{c_3} \cdot C_8} = \frac{1}{\pi \cdot 10^6 rad/s \cdot 180 \cdot 10^{-12} F} = 1768\Omega \approx 1.8k\Omega$$

Der opstilles et bodeplot med asymptotiske linjer for båndpasfilteret, Figur 31.



Figur 31: Bodeplot med assymptotiske linjer for båndpasfilter

Envelope Detector

For at fortolke 120kHz signalet til et logisk signal, der går HIGH når 120kHz signalet er til stede og LOW når det ikke er til stede, er der tilkoblet en envelope detector efter båndpasfilteret.

Envelope detectoren består af modstanden R_{11} , kondensatoren C_9 samt dioden D_5 . Dioden lader kun positive spændinger passere. Dette bevirket at i tilfælde af negativ spændinger før dioden, aflades kondensatoren i envelope detectoren ikke hurtigere end ved 0V.

Hvert peak af inputtet vil oplade kondensatoren, der sørger for, at spændingen i punktet mellem dioden og kondensatoren holdes nær peakværdien. Modstanden i parallel med kondensatoren vil gradvist aflade kondensatoren. For at kondensatoren ikke aflades så hurtigt at spændingen falder til under den ønskede værdi imellem peaks fra 120kHz signalet, skal følgende forudsætning være opfyldt:

$$T \ll \tau$$

Hvor τ er forholdet mellem modstand og kondensator ($\tau = R_{11} \cdot C_9$). T er en tidsperiode ($T = f^{-1}$). Sættes der værdier ind, findes tidsperioden til:

$$T = \frac{1}{120\text{kHz}} \approx 8.3\mu\text{s}$$

Herefter bestemmes τ til at være $100\mu\text{s}$. Derved aflades kondensatoren langsomt nok til, at den ikke kommer under den ønskede tærskel, men ikke for langsomt i forhold til næste nul-gennemgang på 18VAC-50Hz nettet.

Modstanden sættes til $10k\Omega$, dermed bliver kondensatorens størrelse:

$$C_9 = \frac{\tau}{R_{11}} = \frac{100 \cdot 10^{-6}\text{s}}{10 \cdot 10^3\Omega} = 10 \cdot 10^{-9}\text{F} = 10n\text{F}$$

Komparator

Ovenstående del af carrier detectoren er testet på fumlebræt ved brug af carrier generatoren. Ved testen måltes $480 - 600mV$ på envelope detectoren, ved detektion af 120kHz signal, og ca. 0V uden detektion af signal. Derfor er der lavet en spændingsdeler som giver OP-ampens [4] inverterende indgang en reference på:

$$V_{ref} = 5V \cdot \left(\frac{R_{12}}{R_{12} + R_{13}} \right) = 5V \cdot \left(\frac{680\Omega}{680\Omega + 10 \cdot 10^3\Omega} \right) = 318mV$$

Dette betyder at der på OP-ampens udgang er $+5V$ hvis spændingen på envelope detectoren er over referencen, og $-5V$ hvis den er under referencen. For at undgå negativ spænding på *bitstream* signalet, er der plaseret en diode (D_6), som forhindrer dette. Der er et mindre spændingsfald over dioden, men dette har ingen betydning. Derudover er der forbundet en pull-down modstand (R_{14}) efter dioden for at sikre at *bitstream* ligger stabilt på $0V$, når operationsforstærkeren leverer negativ spænding.

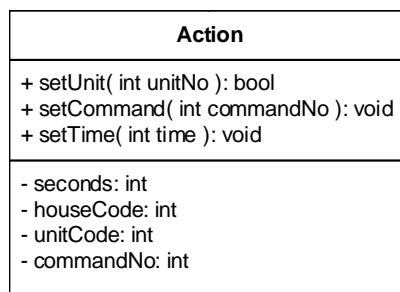
6 Software Design

6.1 Version

Dato	Version	Initialer	Ændring
14. november	1	KT	Første udkast af dokumentet efter review.
15. december	2	LS	Små rettelser. Endelig version.

6.2 PC Blokken

6.2.1 Action (Kristian T.)



Figur 32: Action klassen

setCommand

Operation: `bool setCommand(int commandNo)`

Parametre: Modtager nummeret på den kommando der skal eksekveres.

1 = Tænd

2 = Sluk

3 = Dim 5%

4 = Dim 15%

...

12 = Dim 95%

Returværdi: Returnerer TRUE, hvis input var gyldigt og FALSE hvis ikke.

Beskrivelse: Set-metode til at vælge hvilken kommando, den givne aktion skal indeholde. Metoden ændrer udelukkende på variablerne `commandNo`.

setUnit

Operation: `bool setUnit(int unitNo)`

Parametre: Modtager hvilken enhed aktionen skal manipulere.
 1 = Lampe 1
 2 = Lampe 2
 3 = TV
 4 = Radio
 Alle andre værdier er ugyldige.

Returværdi: Returnerer TRUE, hvis input var gyldigt og FALSE hvis ikke.

Beskrivelse: Set-metode til at vælge hvilken enhed, den givne aktion skal manipulere. Metoden ændrer udelukkende på variablerne `houseCode` og `unitCode`. `houseCode` skal iøvrigt sættes i forhold til den enhed, som vælges. Lampe 1 og Lampe 2 er under huskode 1, TV og Radio er under hhv. 2 og 3.

setTime

Operation: `bool setTime(int time)`

Parametre: Modtager tidspunktet for hvornår en aktion skal udføres i minutter fra kl 00:00. Dvs hvis en aktion skal starte kl 15:30 skal værdien $15 \times 60 + 30 = 930$ indsættes.

Returværdi: Returnerer TRUE, hvis input var gyldigt og FALSE hvis ikke.

Beskrivelse: Set-metode til at vælge hvilket tidspunkt, den givne aktion skal udføres. Metoden ændrer udelukkende på variablen `minutes`.

operator<<

Operation: `ostream & operator<<(ostream & theStream, Action & theAction)`

Parametre: Modtager et `ostream` objekt, som der skal streames til samt en reference til et objekt af klassen `Action`, som skal udskrives.

Returværdi: Returnerer en reference til det `ostream` objekt, som metoden blev kaldt med (tillader cascading).

Beskrivelse: Udskriver en linie med tidspunkt i formattet HH:MM, enhedsnavn og kommando beskrevet i tekstform, afsluttet med et linieskift.

Explicit constructor

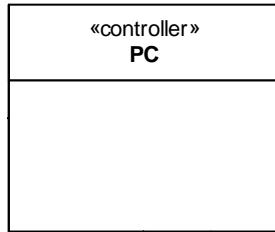
Operation: `Action(int time, int unit, int commandNo)`

Parametre: Modtager parametre for hhv. tid, enhed og kommandonummer.

Beskrivelse: Initierer attributter i objektet, skal sætte attributterne til passende default-værdier, hvis ingen værdier gives.

Attribut	Beskrivelse
<code>int seconds</code>	Tiden fra kl 00:00 til tiden for at den pågældende aktion skal udføres i sekunder.
<code>int houseCode</code>	Huskoden for den enhed, som aktionen skal manipulere.
<code>int unitCode</code>	Enhedskoden for den enhed, som aktionen skal manipulere.
<code>int commandNo</code>	Nummeret på den kommando, som skal eksekveres.

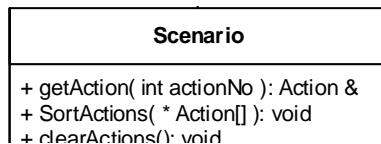
6.2.2 PC Controller



PC controlleren virker som et main program, og holder styr på følgende:

- Kodelås
- Data til UART klassen
- Led mellem Scenario og UI klassen

6.2.3 Scenario (Kristian S.)



Constructor

Operation: void Scenario()
Parametre: -
Retur værdi: -
Beskrivelse: constructoren opretter et array med 20 objekter af klassen action i.

getAction

Operation: Action & getAction(int actionNo)
Parametre: actionNo - ID'et på den action er der skal bruges
Retur værdi: Action & - Reference til den valgte action.
Beskrivelse: Methoden finder den valgte Action ud fra parameteren actionNo, og returnere en reference til den action.

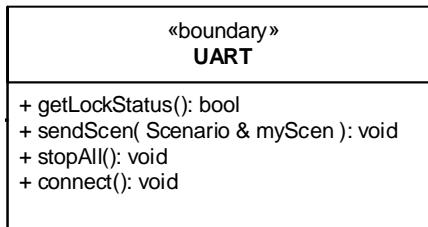
sortActions

Operation: `Action[] & sortActions(Action[])`
 Parametre: Modtager en pointer til et array af `Action` objekter.
 Returværdi: Returnerer en reference til det sorterede array af `Action` objekter.
 Beskrivelse: Sorterer et array af `Action` objekter efter udførelsestidspunkt. Den der skal udføres først lægges i starten af arrayet. Alt skal være relativt til det nuværende tidspunkt.

operator<<

Operation: `ostream & operator<<(ostream & theStream, Scenario & theScen)`
 Parametre: Modtager det `ostream` objekt, som der skal streames til samt hvilket `Scenario` objekt, der skal streames.
 Returværdi: Returnerer en reference til det `ostream` objekt, der modtages som parameter. (tillader cascading).
 Beskrivelse: Skal udskrive ”Aktionsnummer Tidspunkt Enhed Kommando” som titler med et bestemt mellemrum på én linie og herefter udskrive nummeret på den første aktion, den første aktion på den næste linie. Herefter udskrives nummeret på den anden aktion samt den anden aktion på den næste linie osv. Se evt. Tabel 2.2 på side 3 i kravspecifikationen.

6.2.4 UART (Kasper)



UART'en laves med udgangspunkt i funktionerne fra et open-source UART projekt. [10]

Connect

Operation: `Void Connect()`
 Parametre: -
 Returværdi: -
 Beskrivelse: Connect methoden bruges til at finde en gyldig USB port hvor i der er indsat et RS-232 stik som bruges til seriel kommunikation. metoden løber igennem alle gyldige COM-porte på PC'en indtil en port, med RS-232 stik i, findes.

getLockStatus

Operation: `bool getLockStatus(void)`
Parametre: -
Returværdi: bool: returnere `true` / `false` afhængeligt af kodelåsens status.
Beskrivelse: Sender ASCII værdien for et L over til microcontrolleren for at få kodelåsens status. Funktionen venter på at få et svar tilbage fra microcontrolleren, hvilket er ASCII værdien for et U eller et L. Hvis UART'en får et L returneres `True`. Hvis UART'en får et U returneres `False`.

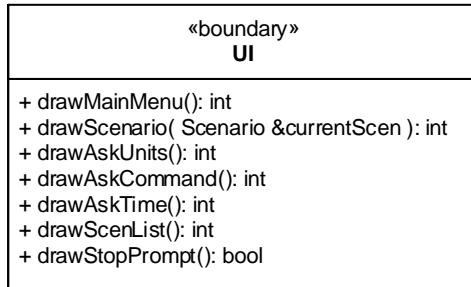
stopAll

Operation: `void stopAll(void)`
Parametre: -
Returværdi: -
Beskrivelse: Funktionen bruges til at slukke for alle enheder tilsluttet systemet. ASCII værdien for et S sendes for til microcontrolleren. Se evt. Protokol for UART side 61.

sendScen

Operation: `void sendScen(Scenario & Scenario)`
Parametre: Scenario & Scenario: reference til scenariet.
Returværdi: -
Beskrivelse: UARTEn transmittere data over serial kommunikation. Rækkefølgen på data der bliver sendt er følgende: Unit (1 char) CMD(1char) Time (3 char). datarækken bliver sendt 20 gange for at få alle kommandoer i scenariet overført til microcontroller.
NOTE: Tiden omreges iforhold til nuværende tid på systemet i computeren. f.eks. hvis klokken er 15.00 på computeren og første aktion sker kl 16.30 bliver tiden sat som 90.

6.2.5 UI (Kasper)



drawMainMenu

Operation: `int drawMainMenu(void)`

Parametre: -

Returværdi: int : Den returnerede integer er hvilken menu der er valgt.

Beskrivelse: Metoden udskriver en menu over de forskellige undermenuer på skærmen og giver brugeren muligheden for at vælge at gå ind i en af undermenuerne (opret nyt scenarie(1), kør eksisterende scenarie(2), stop scenarie(3)). Der fortages et tjek på det indtastede af brugeren, som skal valideres. validationen tjekker at det indtastede er enten 1,2 eller 3 som er de gyldige menu'er. hvis det indtastede er ugyldigt får brugeren en besked om ugyldigt input og får lov at forsøge igen. Dette sker indtil brugeren indtaster noget gyldigt.

drawScenario

Operation: `int drawScenario(Scenario & currentScen)`

Parametre: Scenario & currentScen : reference til klassen Scenario

Returværdi: int : Den returnerede værdi er den valgte action i Scenariet

Beskrivelse: Metoden udskriver en liste over de 20 aktioner der er i scenariet med at bruge ostream operatoeren der er lavet i klassen **Scenarie**. Brugeren kan derefter vælge hvilken af de 20 aktioner der skal redigeres ved at indtaste nummeret på aktionen. Når brugeren har indtastet noget, valideres dette. Hvis inputtet er den af de gyldige værdier (0-19), returneres ID'et/den gyldige værdi. Hvis inputtet er ugyldigt får brugeren en besked om ugyldigt input og får lov at forsøge igen. Dette sker indtil brugeren indtaster noget gyldigt.

drawAskUnits

Operation: `int drawAskUnits(void)`
Parametre: -
Retur værdi: int : Den returnerede integer er hvilken unit der valgt
Beskrivelse: Metoden udskriver en liste over units. hvor unitsne er: Lampe1, Lampe2, TV og Radio. brugeren kan derefter vælge hvilke af de 4 units der skal bruges, ved at indtaste nummeret på unit'en. Når brugeren har indtastet noget, valideres dette. Hvis inputtet er en af de gyldige værdier (1-4) returneres unit ID'en/den gyldige værdi. Hvis inputtet er ugyldigt får brugeren en besked om ugyldigt input og får lov at forsøge igen. Dette sker indtil brugeren indtaster noget gyldigt.

drawAskCommando

Operation: `int drawAskCommand(void)`
Parametre: -
Retur værdi: int : Den returnerede integer er hvilken kommando
Beskrivelse: Metoden udskriver en liste over kommandoer. Brugeren kan derefter vælge hvilken kommando der skal bruges, og ID'en på denne kommando returneres.

drawAskTime

Operation: `int drawAskTime(void)`
Parametre: -
Retur værdi: int : tiden i minutter indtastet af brugeren siden kl 00.00
Beskrivelse: Metoden beder brugeren om at indtaste den ønskede time-tal, hvor en ønsket kommando skal udføres. Brugeren kan derefter indtaste det ønskede time-tal. Metoden beder nu brugeren om at indtaste det ønskede minut-tal i den ønskede time. Brugeren kan nu indtaste det ønskede minut-tal. Metoden validere nu time- og minut-tallene. hvis time-talet er indenfor det gyldige område (0-23) og minut-talet er indenfor dens gyldige område (0-59), udregnes tiden i minutter siden kl 00.00 og returneres. Hvis enten minut- eller time-tal er udenfor det gyldige område, køres metoden fra starten af, og dette sker indtil den gyldig tid indtastes.

drawScenList

Operation: `int drawScenList(void)`
Parametre: -
Retur værdi: int : ID'et på det valgte scenarie
Beskrivelse: Brugeren bliver præsenteret for 3 forudstillede scenarier, med beskrivelser af hvad de gør. Brugeren vælger en af de 3 forudinstillet scenarier, ved at indtaste nummeret på det ønskede scenarie. Det indtastede valideres. Hvis det indtastede er indenfor området 1-3, returneres det indtastede, ellers bliver brugeren bedt om at prøve igen, indtil noget gyldigt er indtastet.

drawStopPromt

Operation: `bool drawStopPromt(void)`

Parametre: -

Returværdi: `bool`: returnere `true` for valgt 'ja' og `false` for valgt nej

Beskrivelse: Brugeren bliver præsenteret med muligheden om brugeren er sikker på at scenariet skal afsluttes. Hvis brugeren trykker på Y / y tasten returneres dette som et `true`, altså et 'ja'. hvis alt andet end Y / y trykkes returneres `false`, svarende til et 'nej'.

6.3 Transmitter Blokken

6.3.1 Action (Kristian T.)

Action
+ setAction(unsigned char unit, unsigned char command, unsigned int time): void
- time: unsigned int - unit: unsigned char - command: unsigned char - houseCode: unsigned char

setAction

Operation: `void setAction(unsigned char unit, unsigned char command, unsigned int time)`

Parametre: Modtager enhed, kommando og tidspunkt relativt til starttidspunktet for aktionen.

Returvaerdi:

-

Beskrivelse: Metodens formål er primært at gemme informationer i en aktion. Metoden skal selv sætte houseCode baseret på hvilken enhed, den kaldes med. De forskellige houseCode værdier kan ses nedenfor:

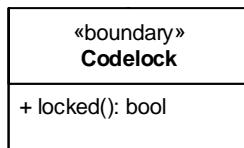
Lamper: 1

TV: 2

Radio: 3

Attribut	Beskrivelse
<code>unsigned int time</code>	Gemmer tiden i minutter for den pågældende aktion relativt til det tidspunkt scenariet sættes igang.
<code>unsigned char unit</code>	Gemmer hvilken enhed, aktionen skal gælde for.
<code>unsigned char command</code>	Gemmer hvilken kommando, aktionen skal sende.
<code>unsigned char houseCode</code>	Gemmer huskoden, for den enhed der skal manipuleres.

6.3.2 Codelock (David)



unlocked

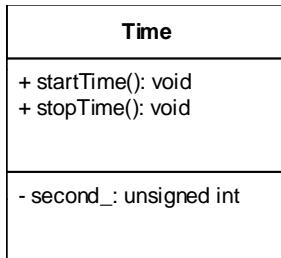
Operation: `bool unlocked(void)`

Parametre: ingen parametre

Returvaerdi: TRUE hvis den er åben. FALSE hvis den ikke er åben.

Beskrivelse: Tjekker om koden er tastet rigtigt ind på vores DE2 board.

6.3.3 Time (Kasper)



Time

Operation: `Void Time()`

Parametre: -

Returvaerdi: -

Beskrivelse: -initialisere timer 1 til at interupt hvert sekund.

startTime

Operation: `Void startTime (void)`
Parametre: -
Returværdi: -
Beskrivelse: -starter tidstælling på timeren. Tiden tælles op hvert minut.

stopTime

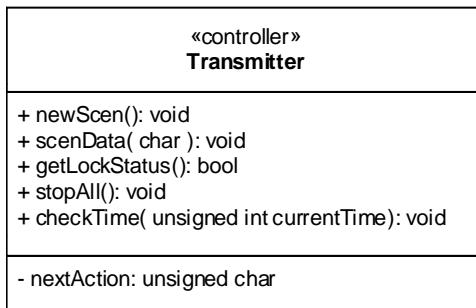
Operation: `Void stopTime (void)`
Parametre: -
Returværdi: -
Beskrivelse: -stopper tidstælling på timeren.

operator++

Operation: `operator++()`
Parametre: -
Returværdi: -
Beskrivelse: -inkremere attributten second med 1.

Attribut	Beskrivelse
<code>unsigned int second</code>	indeholder tiden i sekunder.

6.3.4 Transmitter (Kristian S.)



newScen

Operation: void newScen(void)

Parametre: -

Returvaerdi: -

Beskrivelse: Stopper det igangværende scenerie og sætter nextAction til 0.

scenData

Operation: void ScenData(char)

Parametre: char: Indholder enten information om enhed, kommando eller tidpunkt.

Returvaerdi: -

Beskrivelse: Methoden har ansvaret for at indsætte char'en i det pågældende aktionsobjekt, se UART protokol side 61.

getLockStatus

Operation: bool getLockStatus(void)

Parametre: -

Returvaerdi: bool : TRUE hvis kodelåsen er indtastet korrekt, FALSE hvis den ikke er.

Beskrivelse: Methoden har ansvaret for status angående kodelåsen.

stopAll

Operation: `void stopAll(void)`

Parametre: –

Returværdi: –

Beskrivelse: Methoden har ansvaret for at stoppe det igangværende scenarie og at slukke alle tilkoblede enheder, ved at sende stop-kommandoen ud på Tx10, se X10 protokol side 62.

checkTime

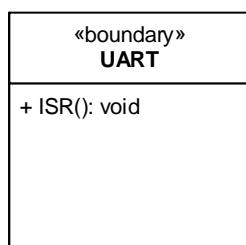
Operation: `void checkTime(unsigned int)`

Parametre: `unsigned int` : Nuværende tid i minutter fra Time-klassen.

Returværdi: –

Beskrivelse: Methoden bliver kaldt fra TimeKlassen hver 5. sekund og er ansvarlig for om den næste aktion skal afvikles. Dette sker kun hvis den givende tid er lig med den næste aktions tid.

6.3.5 Transmitter UART (Kasper)

**constructor**

Operation: `UART`

Parametre: –

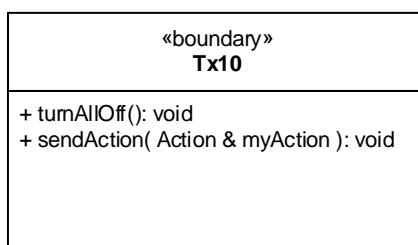
Returværdi: –

Beskrivelse: opsætter forbindelse på RX og TX til UART, sætter op til brug af interrupt.

ISR

Operation: ISR()
Parametre: -
Returværdi: -
Beskrivelse: Ved interupt på UARTEn går transmitteren i 'read mode'. Hvis UART'en læser et L som første char, tjekkes Lockstatus på kodelåsen og returneres til PCen. Hvis et N læses går UARTEN i 'receiving mode' og begynder at aflæse data på receiver pinen. se ref. (state-machine transmitter UART), for lagring af modtagne data. Hvis S modtages sættes transmitteren til 'stop mode'.

6.3.6 Tx10 (Kristian T.)



turnAllOff

Operation: void turnAllOff()
Parametre: -
Returværdi: -
Beskrivelse: Når denne metode kaldes, sendes slukke-kommandoer ud til samtlige enheder på netværket via sendAction().

sendAction

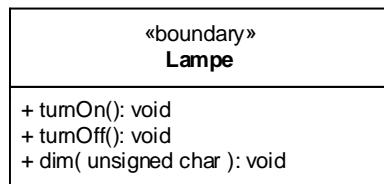
Operation: void sendAction(Action & myAction)
Parametre: Modtager en reference til det pågældende objekt af klassen Action, som skal eksekveres.
Returværdi: -
Beskrivelse: Skal ved hjælp af protokollen for kommunikation over X.10 afsende kommandoer. Der skal sendes én bit hver anden interrupt/nulgennemgang på PD2 (INT0) (interrupt i toggle mode). Ved hver bit skal der ved det efterfølgende interrupt sendes det modsatte af det foregående. Hver gang et 1-tal skal sendes, skal dette være HIGH i 1 ms fra nulgennemgangen/interruptet.

Constructor

Operation: Tx10(void)
Parametre: Ingen
Beskrivelse: Skal initiere Timer0 samt interrupt på PD2 (INT0), men dog ikke aktivere disse.

6.4 Receiver Blokken

6.4.1 Lampe (David)



turnOn

Operation: void turnOn(void)
Parametre: Ingen.
Returværdi: Ingen.
Beskrivelse: Sætter PWM for lampen til 100%.

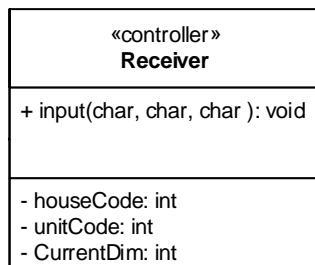
turnOff

Operation: void turnOff(void)
Parametre: Ingen
Returværdi: Ingen
Beskrivelse: Sætter PWM for lampen til 0%.

dim

Operation: `void dim (char)`
Parametre: Modtager en char med en char med den ønskede dimnings værdi.
Returværdi: Ingen
Beskrivelse: Modtager en char som har en værdi mellem 0 og til og med 9, hvor 0 = 5%,
1 = 15% osv. Skal herefter ændre pulsbredden på det ben der er forbundet
til lampen i forhold til den ovenfor angivne procent.

6.4.2 Receiver (David)

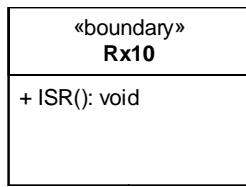


Receiver

Operation: `void input(char)`
Parametre: Modtager en char fra X.10 som bruges til at finde om lampen skal. Tænde,
slukke, dime op, eller dime ned.
Returværdi: ingen retur værdi.
Beskrivelse: Controller for receiveren. Holder styr på lampen nuværerne dimning samt
dens houseCode og unitCode.

Attribut	Beskrivelse
houseCode	Indeholder hus koden for denne Receiver
unitCode	Indeholder unit koden for denne Receiver
CurrentDim	Den nuværne dim værdi.

6.4.3 Rx10 (Kristian T.)



ISR

- Operation: `ISR(INT0_vect)`
 Parametre: Denne servicerutine er indbygget i Atmega32's IO bibliotek og modtager ingen parametre [15].
 Beskrivelse: Denne funktion kaldes ved interrupts på PD2 INT0 og skal være Friend af klassen. Ved interrupts skal denne skifte den nuværende værdi på PA0 ind i `temp`. Der udføres udføres herefter forskellige handlinger afhængigt tilstanden.

checkData

- Operation: `bool checkData(unsigned char)`
 Parametre: Modtager 8 nulgennemgange.
 Returværdi: Returnerer TRUE, hvis de 8 nulgennemgange overholder protokollen for 4 bits og FALSE, hvis den ikke gør.
 Beskrivelse: Denne metode kontrolleer om en given 4-bit størrelse overholder X.10 protokollen. Der modtages 8 nulgennemgange, som deles op i 4 par. Hvert par udgør 1 bit. For at returnere TRUE skal hvert par bestå af ét 1-tal og ét 0.

translate

- Operation: `unsigned char translate(unsigned char)`
 Parametre: Modtager en `char`, som repræsenterer 8 nulgennemgange.
 Returværdi: Returnerer en `char` bestående af 0000 efterfulgt af de 4 bits, som inputtet oversættes til.
 Beskrivelse: Metoden tager parameteren og oversætter den til 4 bits. For at fylde `char`'en sættes bit 7-4 til 0 i returværdien. Bit 3-0 på returværdien sættes til bit 7, 5, 3 og 1 fra parameteren. Dvs hvis metoden kaldes med en `char` 10011010 vil returværdien være: 00001011.

Attribut	Beskrivelse
<code>bool start</code>	Sættes til TRUE, hvis den er igang med at modtage en kommando, ellers sættes den til FALSE.
<code>unsigned char count</code>	Tæller hvor mange nulgennemgange der har været siden sidste hele bit/mønster.
<code>unsigned char temp</code>	Variabel som agerer skifteregister ved modtagelse af nulgennemgange.

6.5 Protokoller (Kristian T., Kristian S., David og Kasper)

6.5.1 Protokol for UART

For kommunikation mellem Transmitter og PC bruges UART med 8 bits bredde. I de to forskellige stadier kan der sendes forskellige ting, under Receiving stadiet skal der sendes: Unit, Command, TimeL, TimeH.

Idle state	
Mode:	Value
Lockstatus	L,ret: L U
Newscen	N
Stop	S

Tabel 12: Tilgængelige værdier i 'Idle' state.

Receiving state					
Unit:	Value	Command:	Value	Time	Value
Arne (L1)	'A'	Dim	'A' - 'J'	bin_val	0x0000 - 0x05FF
Carl (L2)	'C'	Tænd	'T'		
Per (TV)	'P'	Sluk	'S'		
Nils (Radio)	'N'				

Tabel 13: Tilgængelige værdier i 'Receiving' state.

Fra UARTEn på PCen bliver det sendt i rækkefølgen: unit, command, timeLow, timeHigh. Hvor unit er en af de 4 mulige units. Command er en af kommandoerne: T,S eller A til J, hvor A til J er forskellige dimming styrker. Time sendes opdelt i 3 chars. Første char sender de første 4 bit, kodet således at 0 = ASCII værdien for 0, 1 = ASCII værdien for 1 osv. Dvs at Selve tallet + 48 sendes. Det samme gælder for den næste char, som repræsentere bit 4-7. Den sidste char repræsenterer bit 8-11 af den INT, som indeholder tiden. Dette er den maksimale tid, da det er antallet af minutter på et døgn. Der bliver sendt med asynkront, 8bit, 1 stopbit, ingen paritet, og en baudrate på 9600.

6.5.2 Protokol for X.10

Ved kommunikation med X.10, er der lavet en protokol med inspiration fra AN236 Applicationsnote [11]. En bit består af to nulgennemgange, hvor de er hinandens omvendte. Dvs: 1 = 10 og 0 = 01. Udover dette er der et specifikt startmønster "1110" og et specifikt stopmønster "1111", samt en ventekode "000000". Hver kommando skal sendes to gange i stræk, adskilt af ventekoden. I Tabel 14 ses hvordan en kommando er opbygget i X.10.

Startcode	4 bit	4 bit	4 bit	waitcode	4 bit	4 bit	4 bit	Stopcode
1110	House	Unit	Command	000000	House	Unit	Command	1111

Tabel 14: Mønster for hvordan en kommando sendes via X.10

4 bit værdien 'House' erstattes med en af nedenstående huskoder. Hvis koden "All off" modtages, kaldes der i hver receiver en off funktion.

House	Value	Kommentar
All off	0	Slukker alt
Lamps	1	
TV	2	
Radio	3	
Resrvd	4.. 15	ikke taget i brug.

Tabel 15: Tilgængelige huskoder i X.10.

Ligeledes erstattes 'Unit' med en af de nedenstående enheder:

Unit	Value
Lampe 1	1
Lampe 2	2
TV	4
Radio	8

Tabel 16: Tilgængelige huskoder i X.10.

Hver enhed har hver sit sæt af kommandoer, og da kun er valgt at implementere lamper, er her en liste over kommanoder for lamper:

CMD_L	Value
Sluk	0
Tænd	1
Dim 5%	2
Dim 15%	3
Dim ...%	...
Dim 95%	11
Reserved	12... 15

Tabel 17: Tilgængelige kommandoer for lamper.

6.6 Porte (Kristian T., Kristian S., David og Kasper)

Transmitter

- D0 (RXD) & D1 (TXD): Bruges til at kommunikere via UART med PC blokken.
- D2 (INT0): Betragtes som `ZeroCrossDetect: bool` fra Figur 10 side 20.
- B3 (OC0): Forbindes til `X.10Data: bitstream`, bruges til at sende data fra transmitter via X10.
- A0: Henter status fra kodelåsen.

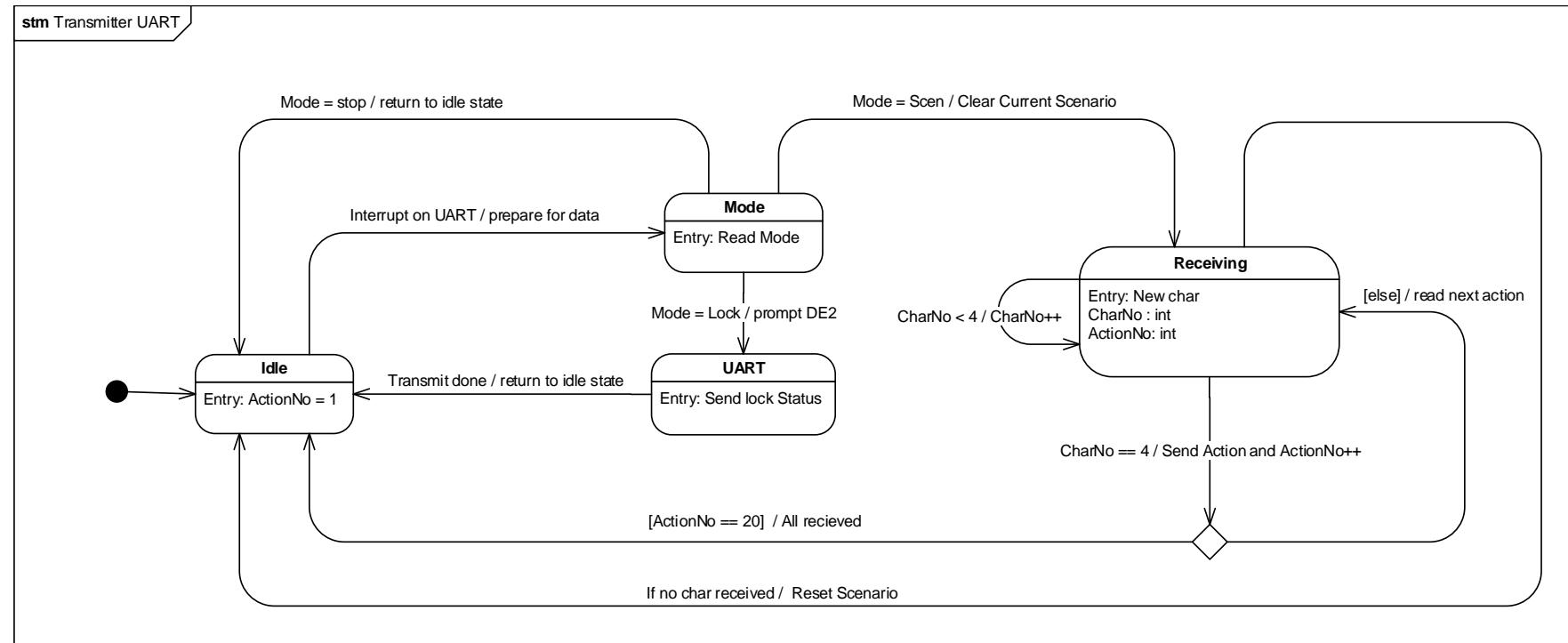
Receiver

Receiver blokken ejer følgende porte på STK500:

- D0 (RXD) & D1 (TXD): Bruges til debugging af Receiveren ved at sende input fra X10 ud via UART.
- A0: Betragtes som `X.10Data: bitstream` fra Figur 12 på side 22.
- D2 (INT0): Betragtes som `ZeroCrossDetect: bool` fra Figur 12 side 22.
- B3 (OC0): Forbindes til `LightCntrl: CntrlPWM` fra ovenstående figur.

6.7 Statemachines (Kristian S. og David)

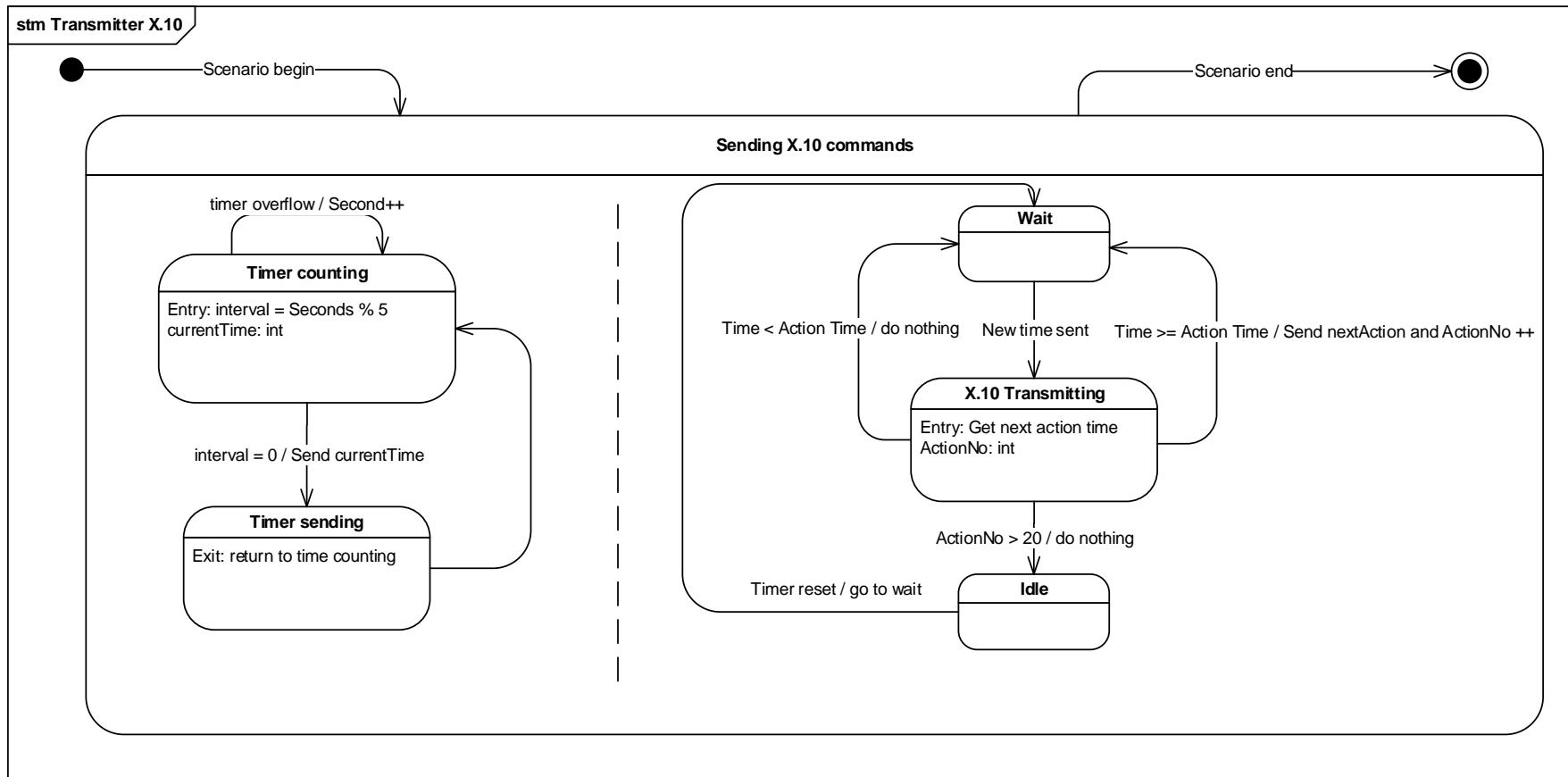
Statemachine over UART/Transmitter



Statemachinen viser Transmitteren's tilstande når PC'en sender data over UART'en. Den første block Mode er et af følgende: Lock, Stop, og Scen(Scenarie). Hvis Scen er sendt, gør transmitter klar til at modtage flere data om det nye scenarie. Efter alt data er modtaget, starter transmitteren det nye scenariet og returner til idle tilstanden.

Statemachine over X.10/Transmitter

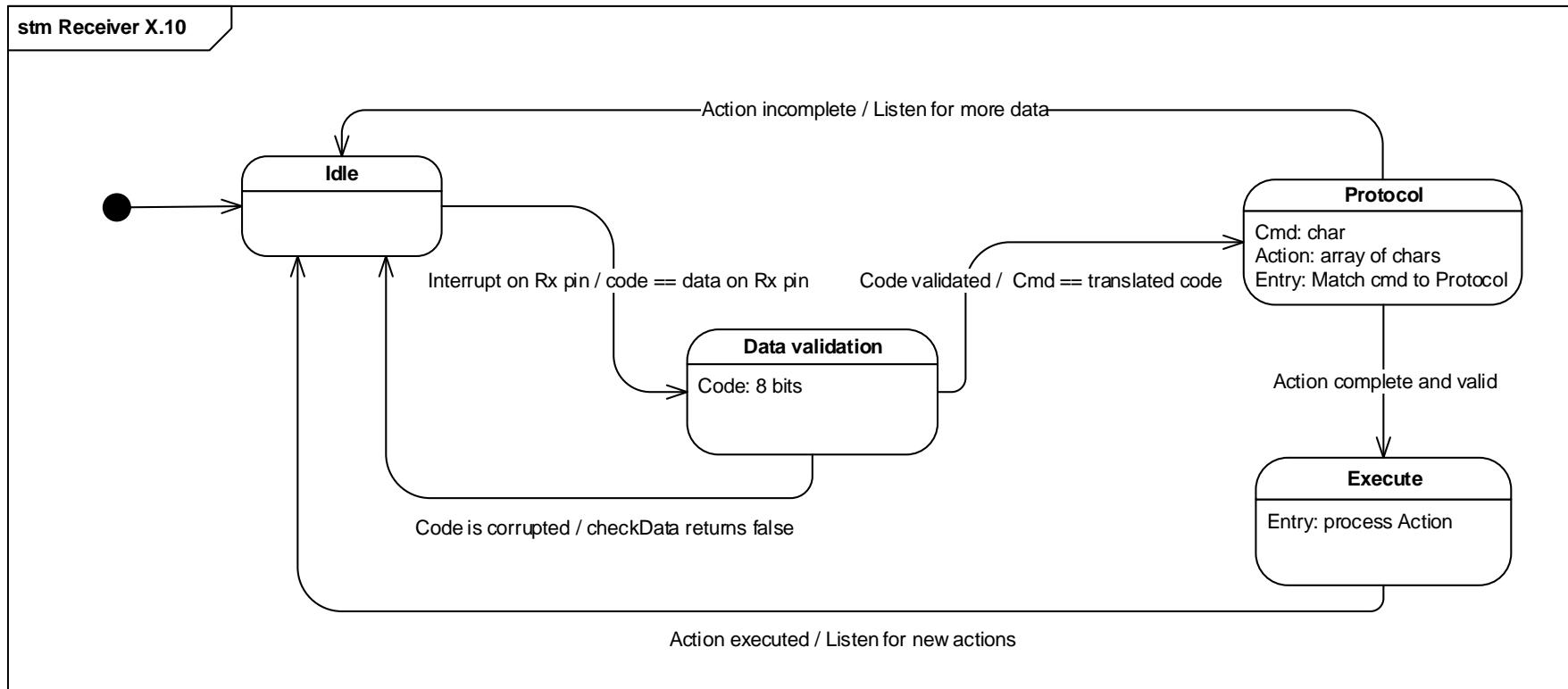
c9



Denne statemachine viser det parallele forløb mellem Time, som holder styr på tiden, og transmitteren, der modtager tid fra Time, samt sender X.10 data ud til Tx10 Klassen.

Statemachine over X.10/Receiver

99



Her ses de forskellige tilstade receiveren skal gennemgå, når data modtages fra X.10 nettet. Når data modtages, loades de ind i en buffer, og checkes op mod X.10 protokollen. Hvis kommandoen er korrekt, vil den blive eksekveret, ellers skal den kastede kommandoen væk. Receiven lytter derefter for mere data.

7 Hardwareimplementering

7.1 Version

Dato	Version	Initialer	Ændring
03. december	1	HBJ	Første version.
15. december	2	LS	Små rettelser. Endelig version.

7.2 Spændingsforsyning (Lasse)

Spændingsforsyningen realiseres i første omgang på fumlebræt, hvorefter det måles med multimeter at modulet leverer hhv. $+5.0\text{og }-5.0\text{V}$.

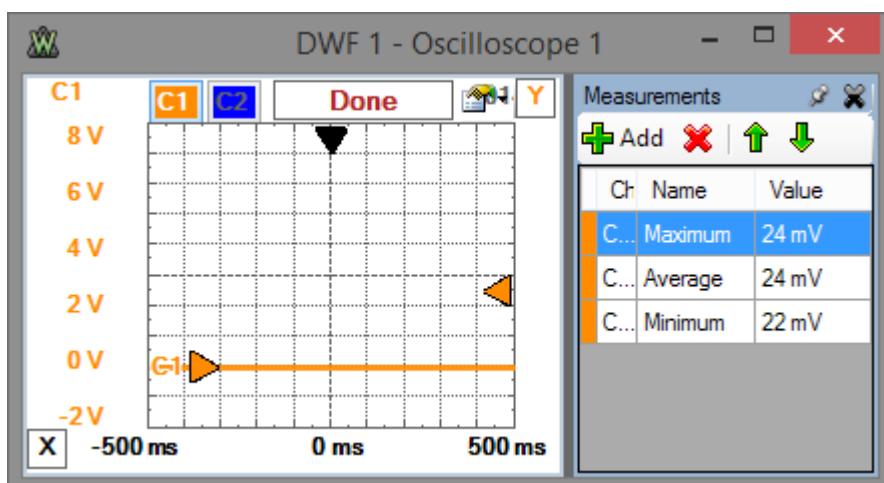
Som nævnt i designafsnittet er der for at sikre mod overophedning, monteret en køleplade på spændingsregulatoren.

Herefter loddes spændingsforsyningen på veroboard. Modultest af veroboardet viser at spændingsforsyningen fungerer korrekt.

7.3 Kodelås (Philip)

I dette afsnit testes hvorvidt kommunikationen mellem Altera DE2 boardet og Atmel STK500 kittet virker.

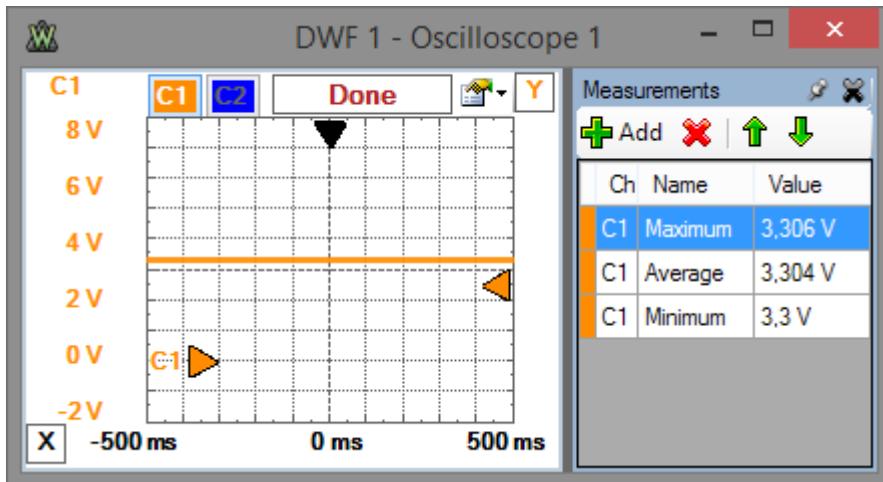
Vhdl filen for kodelåsen [12] downloades til DE2 boardet og der monteres et et Analog discovery oscilloskop på DE2 boardets GPIO(1) pin 0 med en reference på GPIO(1) pin 11(GND).



Figur 33: Analog discovery - Oscilloskop: DE2 LOW output

På figur 33 og 34 kan det ses at et logisk HIGH output på DE2 boardet er målt til en spænding på 3.3V og en logisk LOW har en spænding på 0.0V .

For at teste om dette er en spænding der er høj nok til at STK500-kittet registrer det som HIGH, laves et lille testprogram. Testprogrammet sætter *PORTC* LOW, når *PORTA* pin 0 sættes HIGH og omvendt.



Figur 34: Analog discovery - Oscilloskop: DE2 HIGH output

```

1 #include <avr/io.h>
2
3 void main( void )
4 {
5     DDRA = 0x00; // PORTA is input
6     DDRC = 0xFF; // PORTC is output
7
8     while(1)
9     {
10        if (PINA==0b00000001) // Read PORT A0 and checks if it is HIGH
11        {
12            PORTC = 0x00; // LEDs turns on (PORTC is LOW)
13        }
14        else
15        {
16            PORTC = 0xFF; // LEDs turns off (PORTC is HIGH)
17        }
18    }
19 }
```

Med en Analog discovery's funktionsgenerator laves et sinussignal med en amplitude på 5V som sættes til *PORTA* pin 0. Med analog discovery's oscilloskop måles nu på et tilfældig ben på *PORTC*. Det ses på figur 35 at der skal påtrykkes en spænding på 2.3V for at STK500-kittet til at skifter fra at registrere HIGH til at registrere LOW.

Det ses på figur 36 at der skal påtrykkes en spænding på 2.5V for at STK500-kittet til at skifter fra at registrere LOW til at registrere HIGH.

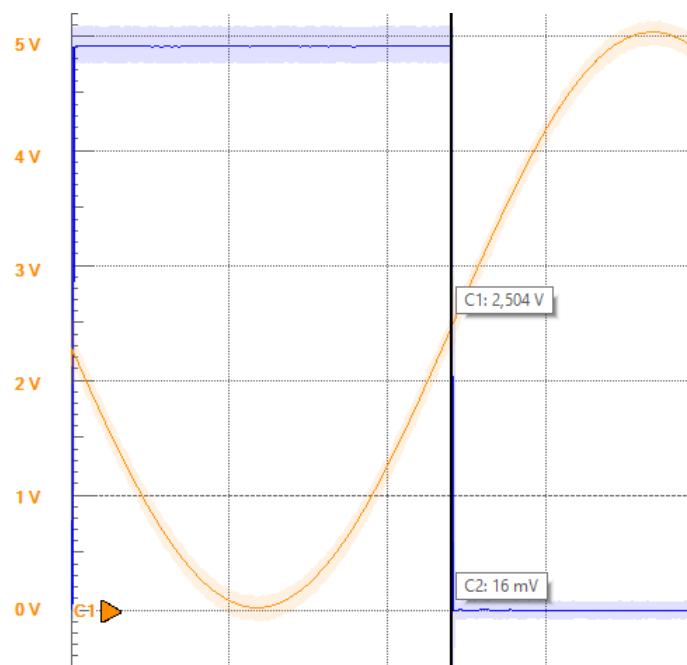
For at kunne udføre en samlet modultest, tilsluttes STK500-kittet PORT A pin 0 til DE2 boardets GPIO(1) pin 0 og et GND-ben på STK500-kittet tilsluttes GPIO(1) pin 11. Ydermere tilsluttes STK500-kittet PORT C til dets indbyggede LED'er.

På både STK500-kittet og DE2 boardet ligger samme software som i de ovenstående test. Det kan nu ses at STK500-kittets LED'er tænder og slukker som forventet, alt efter kom koderne på DE2-boardet er indtastet korrekt.

Altså er kan det konkluderes at vi godt kan bruge koble DE2 boardet direkte til STK500-kittet.



Figur 35: Analog discovery - Oscilloskop: STK-500 Faling edge

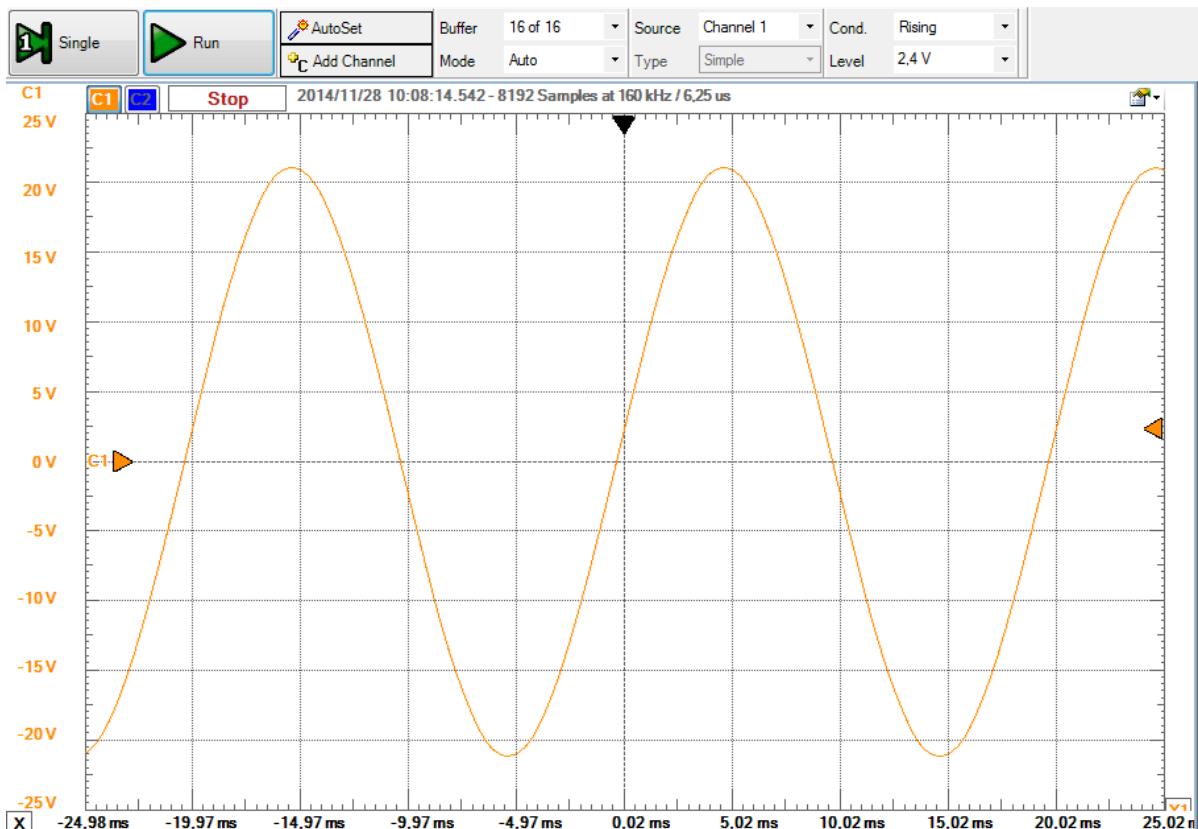


Figur 36: Analog discovery - Oscilloskop: STK-500 Rising edge

7.4 Zerocross Detector (Morten og Henrik)

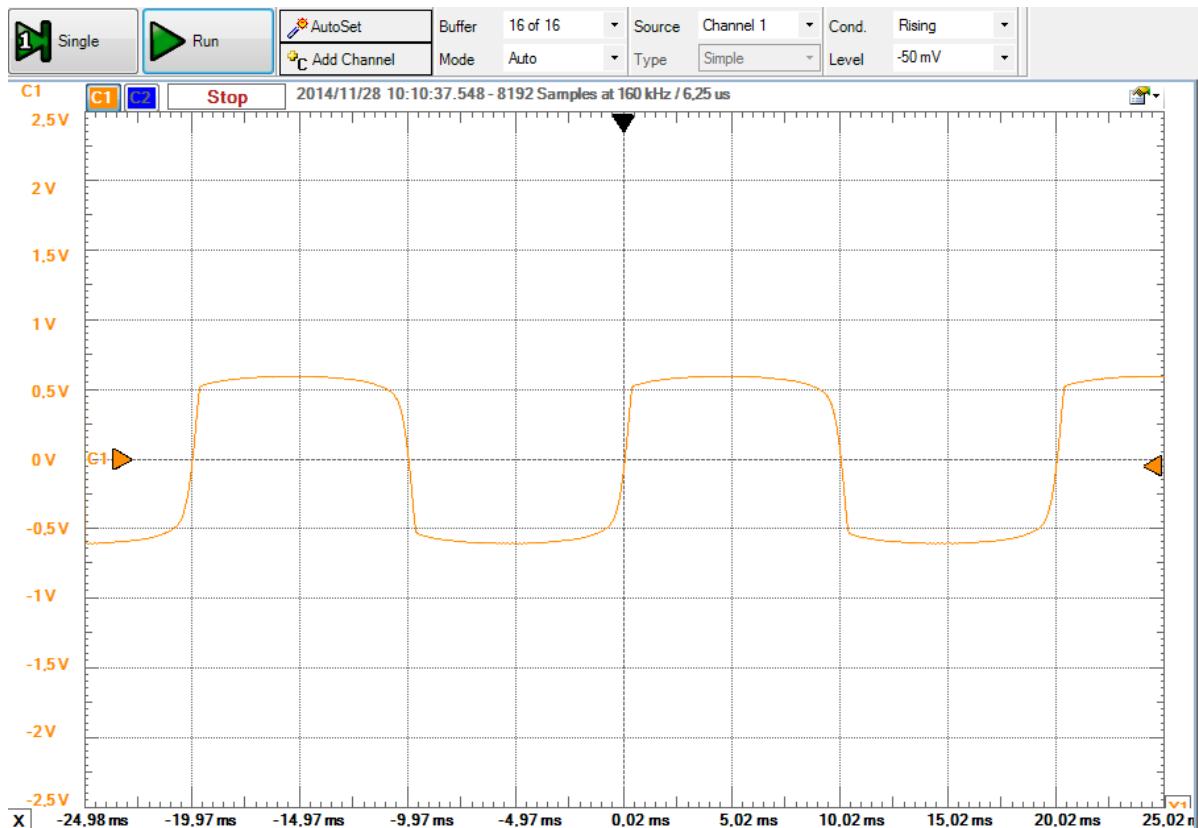
For at sikre at de enkelte dele af zerocross detectoren virker, realiseres kredsløbet på fumlebræt i flere etaper.

Først realiseres kredsløbets lavpasfilter, og det konstateres vha. oscilloskop, at det virker som forventet. Der er valgt en $33k\Omega$ modstand til filteret (R_1), da dette er den nærmeste værdi, der er tilgængelig. Figur 37 viser en mæling af signalet efter lavpasfilteret.



Figur 37: Ren sinus efter lavpasfilter

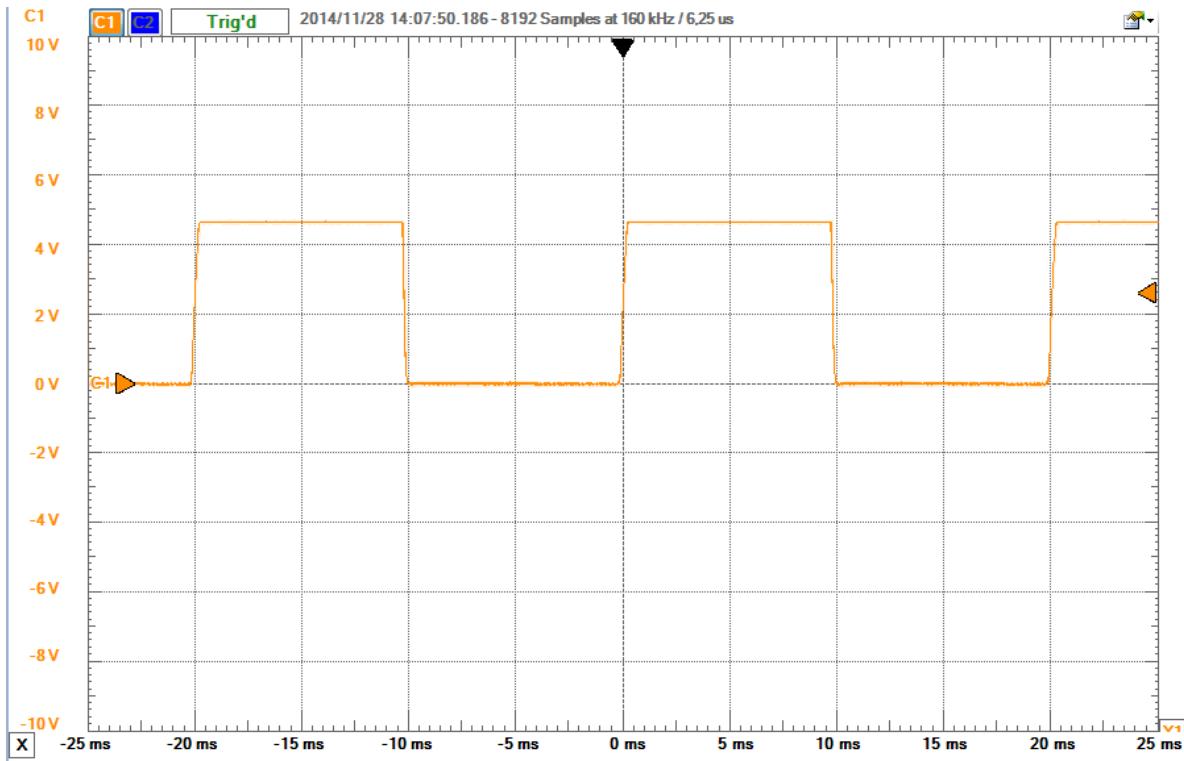
Herefter tilføjes de to dioder (D_1 og D_2) [3] på fumlebrættet, og det ses at signalet begrænses som forventet. Signalet er vist på Figur 38.



Figur 38: Dæmpet sinus efter dioder

Herefter tilføjes komparatoren (TS912) [4], som sammenligner signalet fra Figur 38 med stel. R_2 og R_3 bevirket at der er en hysterese på $200mV$.

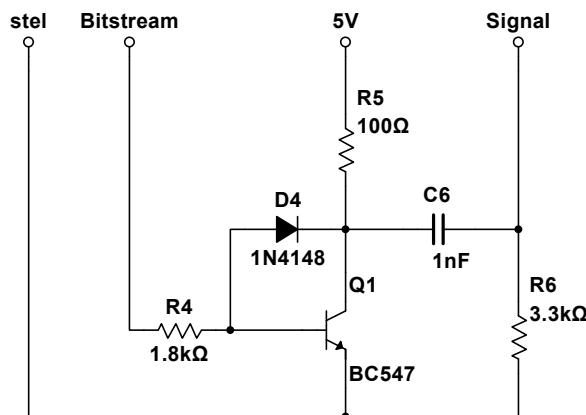
Indgange på ubrugte gates er sat til stel. Dioden efter operationsforstærkeren fungerer som forventet. Udgangssignalet er som ønsket (Figur 39).



Figur 39: ZeroCrossDetect signal efter komparator

7.5 Carrier Generator (Lasse og Philip)

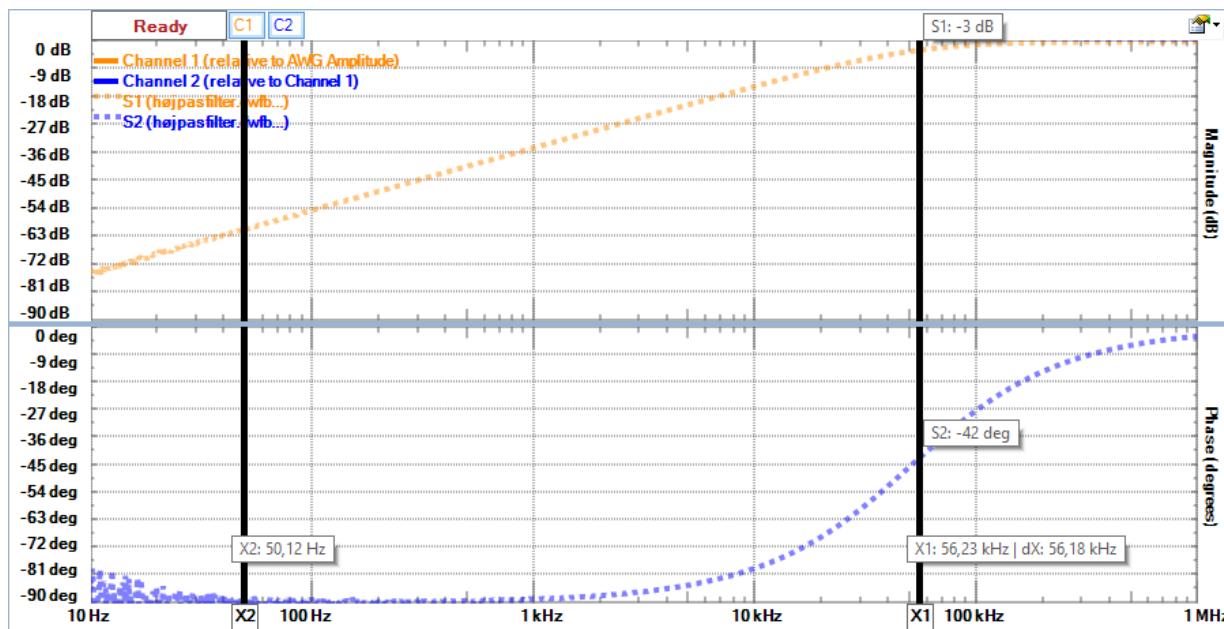
Først opbygges kredsløbet på fumlebræt, hvor de nedenstående del-elementer testes. Når del-elementerne, er testet med succes, laves en samlet modultest for Carrier generatoren. Efter fuldendt modultest, loddes et veroboard med de eventuelle ændringer til designet, der er foretaget.



Figur 40: Carrier generator

Der laves en del-test af højpasfilteret, som er består af C_6 og R_6 , se figur 41. Testen skal sikre at filteret opfører sig jf. bodeplotet i design-dokumentet på Figur 28 på s. 39 i design afsnittet.

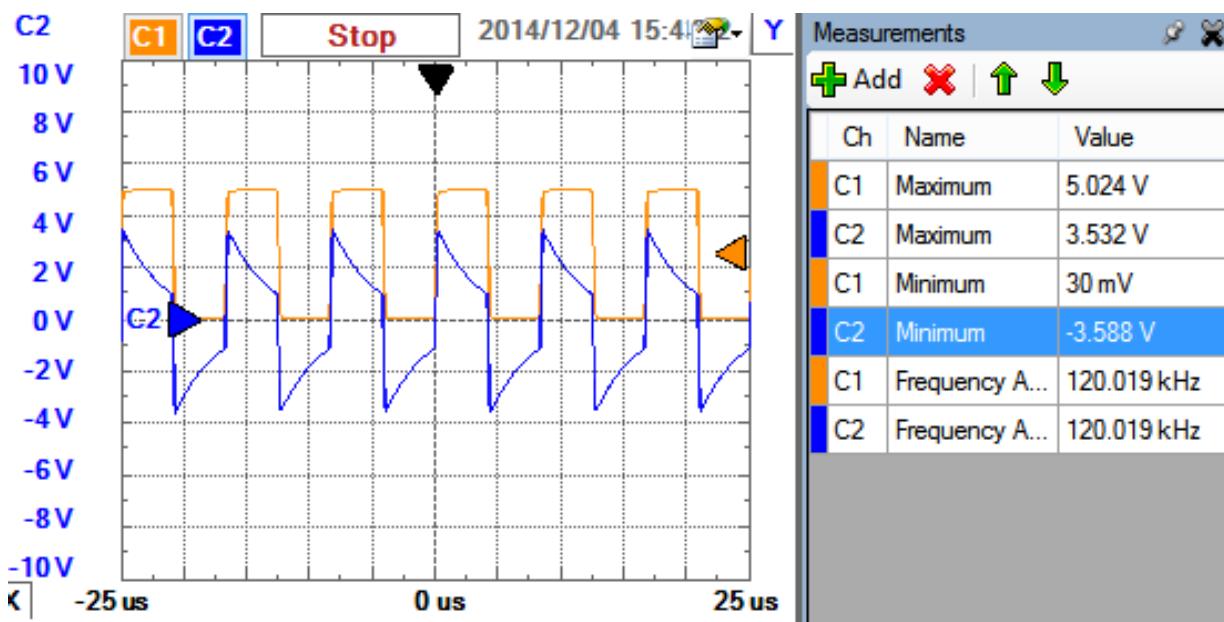
Med *Network Analyzer* funktionen på *Analog discovery* er der konstrueret et bodeplot, der viser amplitude- og frekvenskarakteristikken for højpasfilteret. Se Figur 41. Det ses, at knækfrekvensen



Figur 41: Analog discovery - Network Analyzer: Bodeplot af højpasfilter

ligger som forventet tæt på 50Hz , svarene til $100\pi \cdot 10^3 \text{rad/s}$. Ydermere kan det ses at de 50Hz bliver dæmpet med omkring 60dB , hvilket er en tilstrækkelig dæmpning. Dæmpningen af 120kHz signalet er tæt på 0dB . Overordnet ser det ud som forventet og som det er beskrevet i designdokumentet på side 38.

Ved at sende et 120kHz firkantsignal gennem filteret, fås der et signal der viser opladning og afladning af kondensatoren.



Figur 42: Måling af firkantsignal efter passering af højpasfilter.

Outputtet kan ses på Figur 42. Den orange kanal viser et 120kHz firkantsignal, mens den blå viser

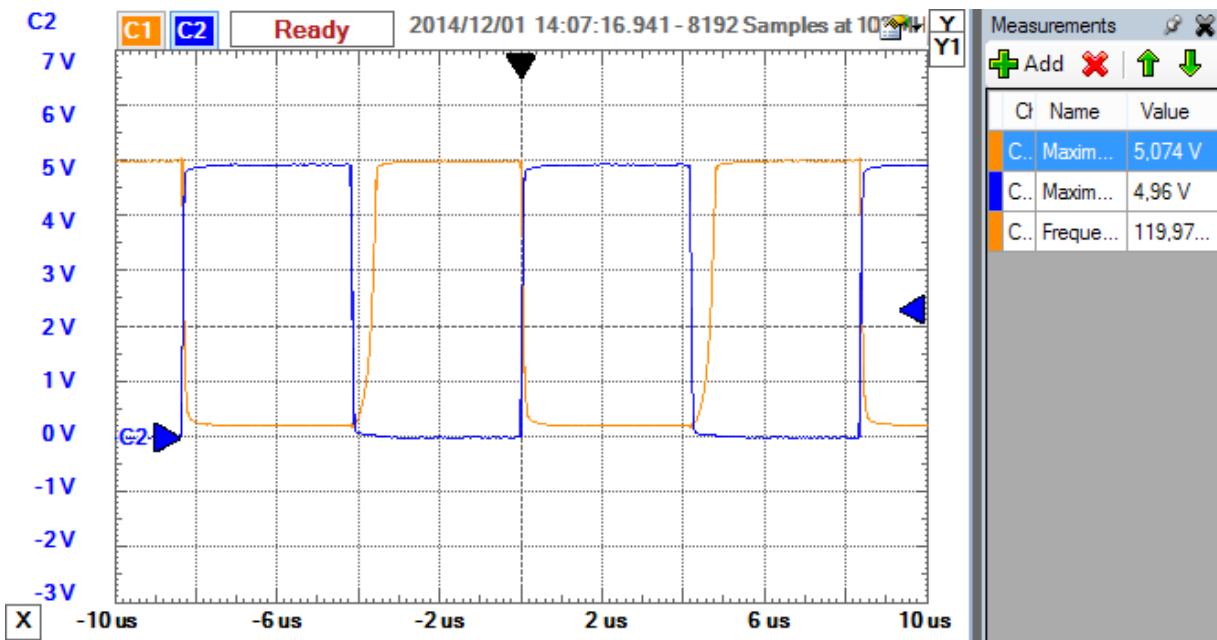
signalet efter passage gennem højpasfilteret.

Oprindeligt blev kredsløbet designet med en transistor (*BD139*) [6], men det viste sig at *Bitstream*, (0V – 5V 120kHz) firkantsignal, ikke kunne passere igennem fra transistorens kollektør til emitter. Efter en fejlsøgning kan det konkluderes, at dette skyldes en positiv opladning, når signalet var HIGH, som transistoren ikke kan nå at aflade inden næste periode.

Hvis transistoren skal anvendes, er det nødvendigt at fjerne den positive ladning der skabes, hurtigere end den selv kan aflade. Problemet kan løses ved at indsætte en diode (*D₄*) [3] parallelt med kollektør-base benene med spærreretning mod kollektor (Baker clamp) [18]. Dette vil hjælpe transistoren til at aflade den positive spænding hurtigere.

En anden løsning er, at skifte transistoren *BD139* ud med en anden transistor, der har et mindre storage delay, fx *BC547* [5]. Konsekvensen heraf vil dog være at transistoren kun kan trække 100mA middelstrøm.

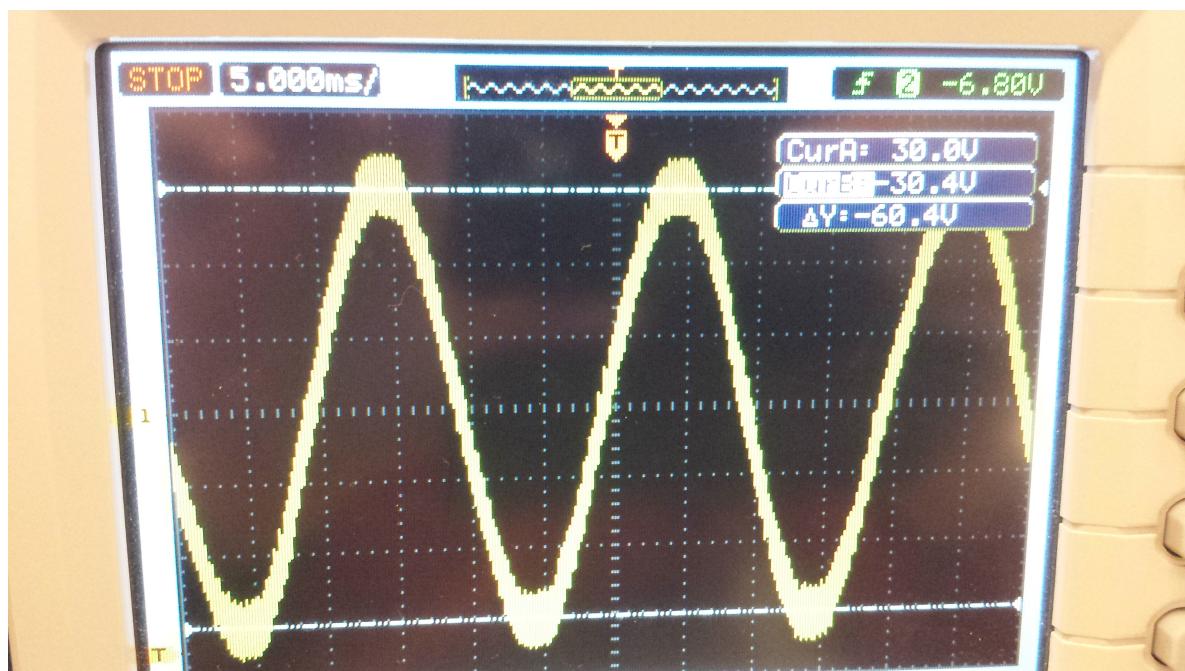
Der vælges en blanding af de to løsninger, således transistoren nu er en *BC547* og der indsættes en diode parallelt med kollektør-base benene.



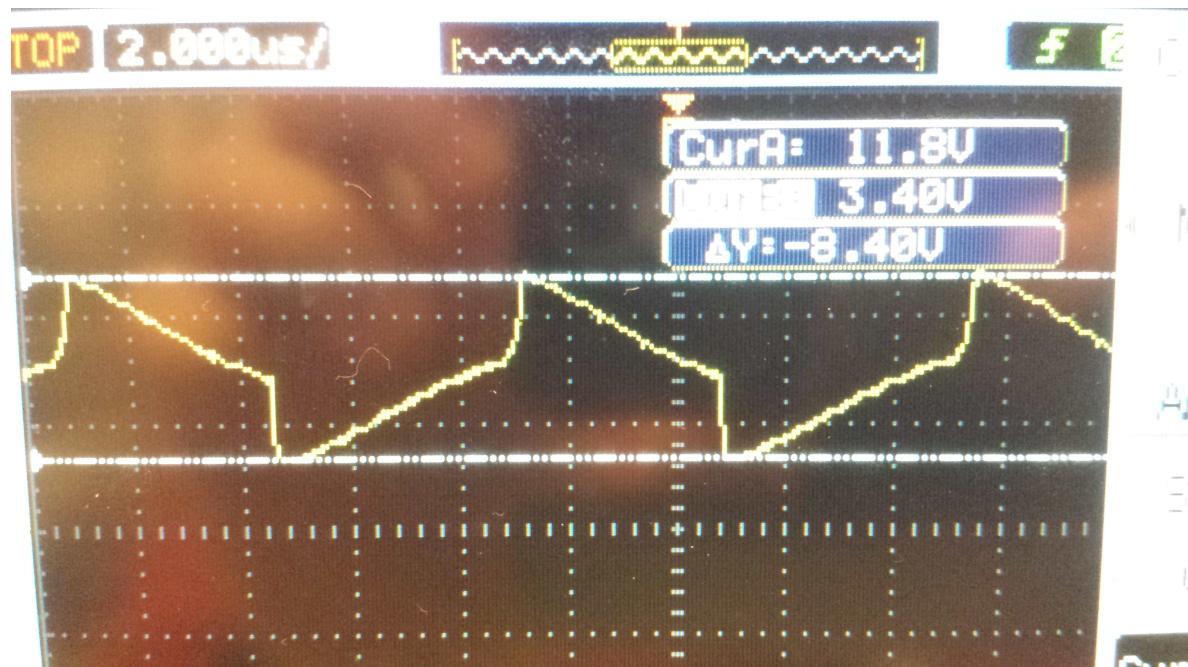
Figur 43: Analog discovery - Oscilloskop: Transistor-kredsløb

På figur 43 ses en oscilloskop-måling af transistor-kredsløbet. Her ses at udgangen *Signal* (den orange), er inverteret i forhold til indgangen *Bitstream* (den blå). Det kan ligeledes ses at der er en lille opladning- og afladningstid på udgangen, der skyldes opladningen af transistoren. Dette har ingen betydning i forhold til anvendelsen af signalet.

En samlet modultest af Carrier generator udføres med et oscilloskop på udgangen af carrier generatoren (*Signal*).



Figur 44: Oscilloskop: Carrier generator modultest



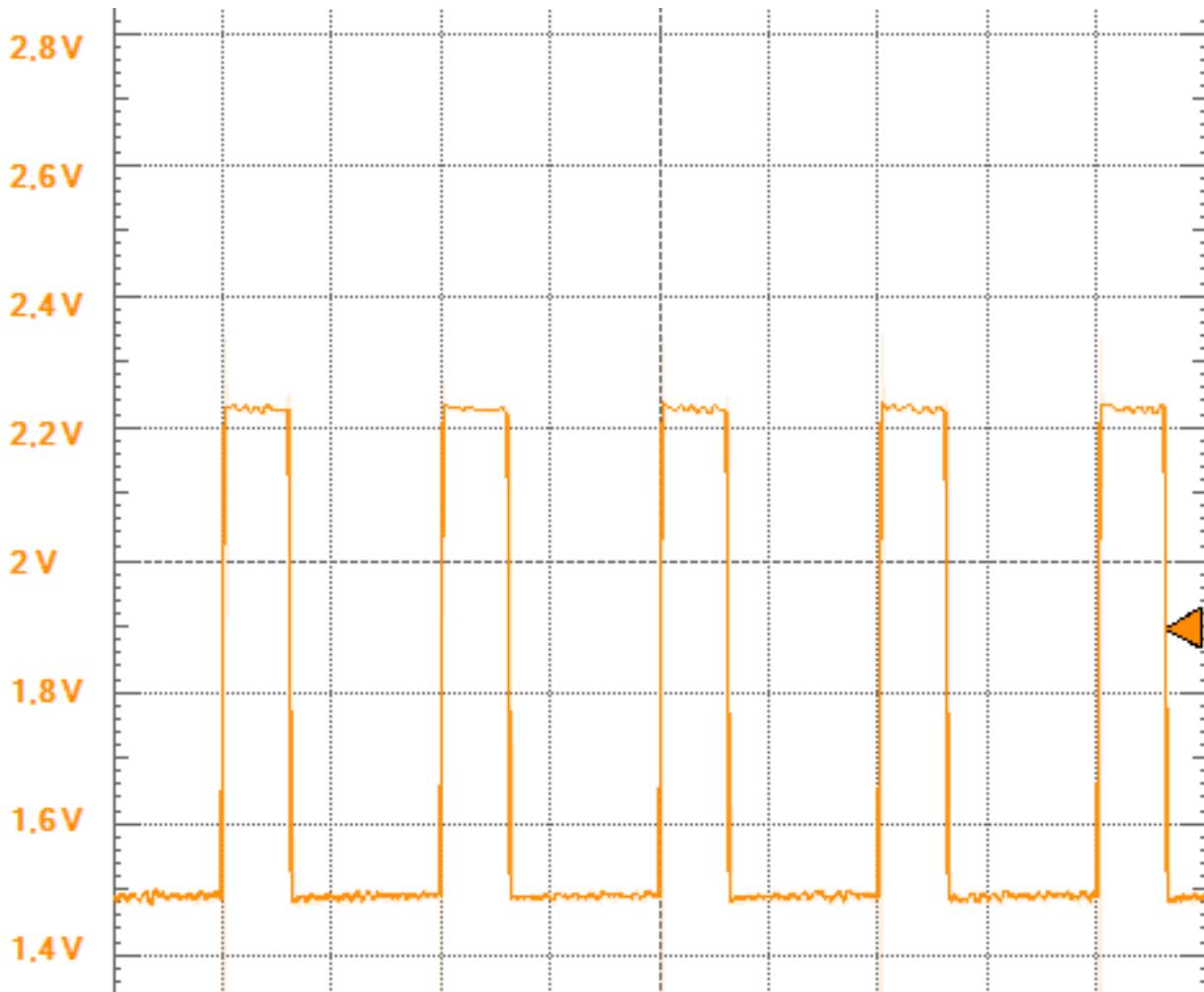
Figur 45: Oscilloskop: Carrier generator modultest zoom

Figur 45 viser samme signal som figur 44. Der er blot zoomet ind så der kun er $2\mu s$ mod $5ms$ pr. streg på tidsakssen. Testen viser at der bliver påtvunget et $120kHz$ signal på udgangen *Signal*, der i forvejen indeholder en $18VAC - 50Hz$ frekvens.

Det bemærkes at signalet har en peak to peak spænding på $3.4V$, hvilket skyldes spændingsdeling i kredsløbet.

7.6 Lysmodul (Kristian S.)

Lysmodulet er opbygget på fumlebræt før det blev loddet på et veroboard og modultestet.



Figur 46: Spænding over lysdiode, ved et $0 - 5V$ firkantsignal, med 30% duty cycle

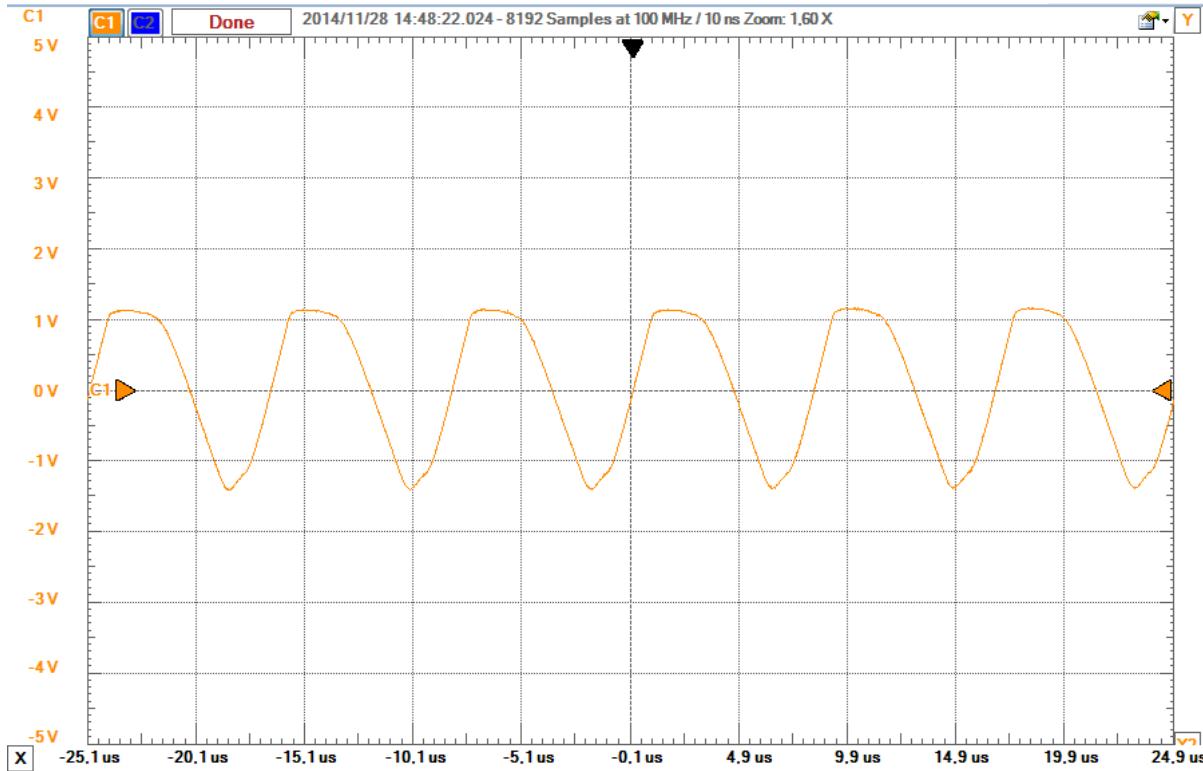
Frekvensen er valgt til $1kHz$, jf. signalbeskrivelsen på side 23, eftersom en alt for lav frekvens vil give et synligt blinkende lys. Duty-cycle blev testet fra 0% til 100% i spring af 10% , og viste et tydeligt skift i lysstyrke.

Efter test på fumlebræt blev modulet implementeret direkte på veroboard.

7.7 Carrier Detector (Morten og David)

For at sikre at alle dele af kredsløbet fungerer, deles det op i mindre blokke, og implementeres på fumlebræt.

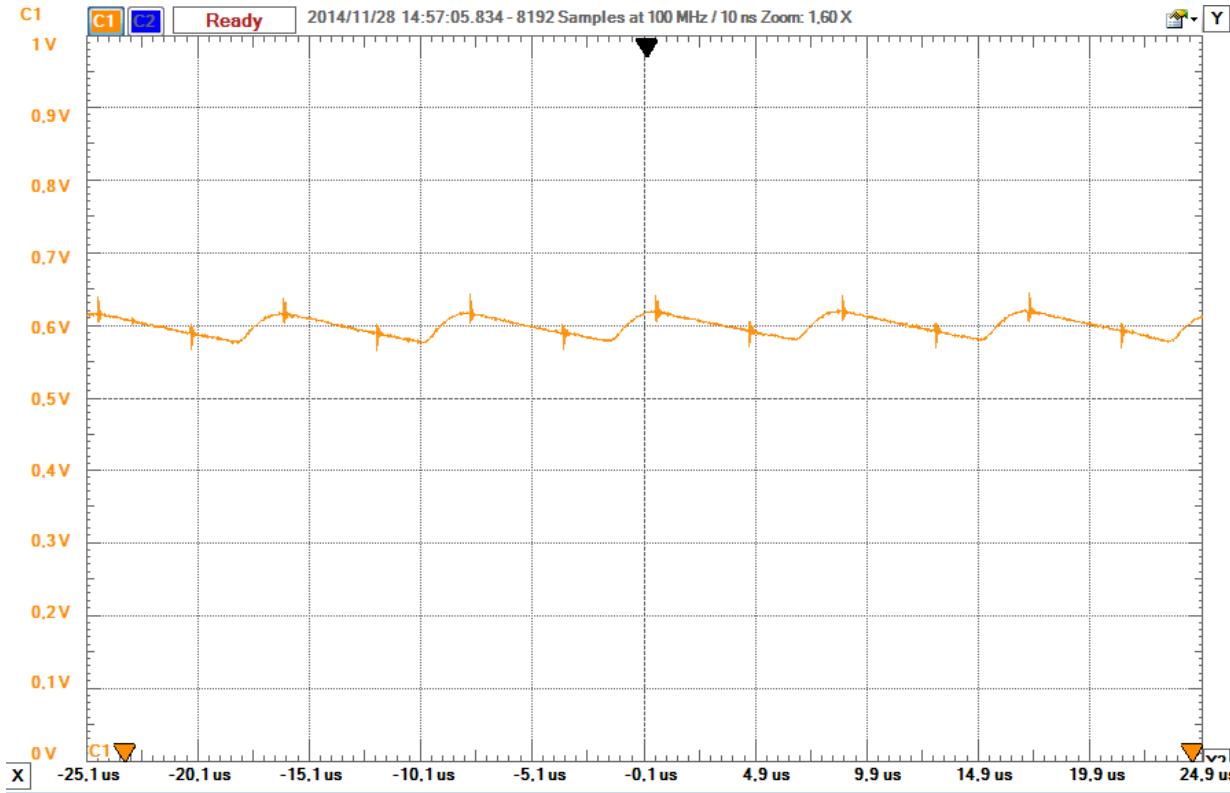
Først realiseres båndpasfilteret for at konstatere, at $18VAC50Hz$ delen er dæmpet, men $120kHz$ signalet fortsat er til stede.



Figur 47: Signal efter båndpas

Det ses på Figur 47, at båndpasfilteret virker som forventet. $18VAC 50Hz$ signalet er væk. $120kHz$ signalet ser noget anderledes ud, men det har ingen betydning.

Herefter realiseres selve envelope detektoren (D_5 [3], R_{11} og C_9), og der måles med oscilloskopet på udgangen. Signalet er vist på Figur 48.

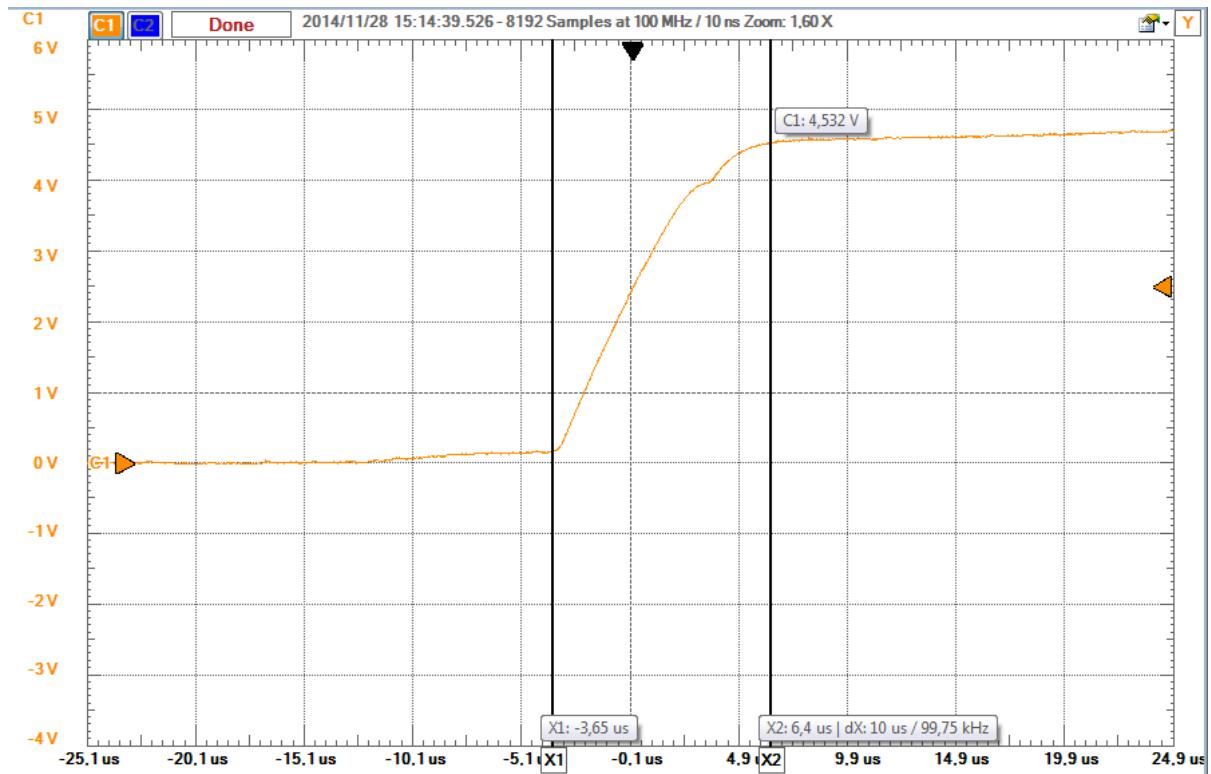


Figur 48: Signal efter envelope detektor med 120kHz

Det ses på Figur 48, enveloppedetektoren fungerer. 120kHz signalet ses tydeligt på takkede signal, og man kan desuden se at forholdet mellem R_{11} og C_9 er passende; afladningstiden er lang nok. Det testedes desuden at udgangen på envelope detektoren gik til 0V , når 120kHz signalet fjernes.

Signalet efter envelope detektoren ligger mellem 480mV og 600mV , og den efterfølgende komparator [4] har derfor en reference på 318mV på den inverterende indgang. Komparatoren U_{4B} og dioden D_6 samt pull-down modstanden R_{14} realiseres på fumlebrættet, og spændingen på udgangen måles. Uden detektion af 120kHz signal måles 0.0V og med detektion af 120kHz signal måles 4.7V , hvilket ligger inden for tolerancen jf. signalbeskrivelsen.

For at sikre at rise time på bitstream er hurtig nok, måles dette med oscilloskopet; resultatet ses på Figur 49.



Figur 49: Rising edge på bitstream

Det ses på Figur 49, at signalet er ca. $10\mu s$ om gå fra lav til høj, hvilket er tilfredsstillende. Det ses desuden, at der ikke er prel på signalet.

8 Softwareimplementering

8.1 Version

Dato	Version	Initialer	Ændring
27. november	1	LS	Første udkast af dokumentet

8.2 PC blokken

8.2.1 Scenario (Kristian S.)

Scenario constructor

Scenario klassen bruger STL(Standard Template Library) vector container format istedet for et array. Constructoren bruger STL's assign funktion til at oprette 20 aktion objekter med default værdier.

getAction

Metoden returner en reference til et aktion objekt i vectoren på baggrund af pladsen i den.

sortActions

sortActions bruger STL's sorterings funktionalitet til at sætte de aktions objekter i numerisk rækkefølge set i forhold til deres tids værdi. For at kunne sammenligne aktion objekter, har aktion klassen fået overloadet dens «"operator. sortAction har ikke længere en returværdi, men muterer et scenarios vector af action objekter.

```
1 void Scenario :: sortActions()
2 {
3     sort(scen_.begin() , scen_.end());
4 }
```

clearActions

clearActions bruger clear funktionen i STL for at slette alle actions objekter, hvor efter de bliver oprettet igen med default værdier.

Ostream Operator

Scenario klassens ostream operator er blevet overloadet til brug i UI klassen. «<"operatoren udskriver nu alle aktion objekters værdier som: Tid, kommando og enhed.

8.2.2 Action (Kristian S.)

Constructors

Aktion klassen indeholder to constructors: En default constructor, der kaldes 20 gange når Scenario objektet først oprettes, og en explicit constructor til de prædefinere scenarier.

setUnit

set-metode til at sætte aktionens enhedsværdi.

setCommand

Set-metode til aktionens kommandoværdi. Funktionen oversætter kommandoens talværdi til en char, der vil blive brugt i UART klassen.

setTime

Set-metode til aktionens tidsværdi.

getUnit

Get-metode til aktionens enhedsværdi.

getCommand

Get-metode til aktionens kommandoværdi.

getTime

Set-metode til aktionens tidsværdi

Ostream Operator

`Ostream` operatoren er overloadet og implementeret som en fri funktion, og kan derfor ikke tilgå de private members direkte.

Less Than Operator

Yderligere er der tilføjet en overload af operatoren `<`, så et Action objekt kan sammenlignes direkte på dens tids værdi.

8.2.3 UI (Kasper)

UI'en er en klasse der består af en række metode-kald. Hver af metoderne clearer skærmen for tidlige menu, og tegner den nye menu alt efter hvilken menu, der bliver kaldt. Skærmen bliver clearet ved brug af kommando'en `system("cls")`. som forekommer i starten af alle metode-kaldene. En metode afsluttes ved at returnere en integere i alle tilfælde, men untagelse af funktionen `drawStopPrompt` som returnerer en boolean.

drawMainMenu

`drawMainMenu` metoden udskriver en liste over mulige menuer på skærmen hvorefter brugeren får mulighed for at indtaste, hvilken menu der ønskes. ved brug af `Cin` til at læse input fra brugeren, kan brugeren også indtaste forkerte inputs som f.eks. "AA3". Til at forhindre at forkerte inputs bliver indlæst, bliver der fortaget en validering af den indtastede int. Det tjekkes at inputtet er imellem 1 og 3, og alle andre inputs resultere i en fejlmeddelse på skærmen, og beder brugeren om at prøve igen.

```
1 cin >> pick;
2     if (pick >= 1 && pick <= 3) // tjekker at input værdien er indenfor gyldige
3         grænser
4     return pick;
5     else
6         cout << "ugyldig menu, pr\x9Bv igen: ";
```

Ved inputs som f.eks. "AA3", vil inputtet ikke kunne valideres ordenligt og resultere i en uendelig løkke af fejlmeddelser. Ved at flush keyboardets buffer, sørges der for at uanset hvor lang en tekst brugeren indtaster, resultere det kun i en enkelt fejlmeddelse.

```
1 cin . clear ();
2 fflush (stdin);
```

drawScenList

Metoden `drawScenList` udskriver en liste med beskrivelsen af 3 forudindstillede scenarier. hvorefter brugeren kan vælge en af de forudindstillede scenarier. Validering af input og valg fungerer på samme måde som `drawMainMenu` metoden.

drawScenario

Metoden `drawScenario` får en reference til et scenarie ind som parameter. Metoden bruger Scenariet ostream operator til at udskrive de 20 aktioner, der ligger i scenariet ud på skærmen.

```
1 int drawScenario (Scenario & Scen)
2 {
3     ...
4     cout << Scen ;
```

Brugeren kan herefter vælge en af de 20 aktioner, ved at indtaste nummeret på aktionen. valg og validering af aktion fungerer på samme måde som `drawMainMenu`, dog er validering af input sat imellem 0 og 19, istedet for 1 og 3.

```
1 if (pick >= 0 && pick <= 19)
```

drawAskUnit

Metoden **drawAskUnit** udskriver en liste over Units der kan vælges til udførsel af en kommando. Brugeren kan vælge mellem 2 lamper, et TV og en radio. Valg og validering af input fungere på samme måde som **drawMainMenu**, dog er validering af input sat imellem 1 og 2, som er de 2 lamper. TV og Radio kan ikke vælges, da de ikke implementeres i systemet.

drawAskCommand

Metoden **drawAskUnit** udskriver en liste over kommandoer der kan udføres af den givne Unit. Brugeren kan vælge mellem at tænde, slukke og dimme. Valg fungere som i **drawMainMenu** metoden. Validering af input tjekker her for om det er enten tænd, sluk eller dim der er valgt. Ved tænd og sluk returnere koderne for det valgte, men hvis det valgte er dim, får brugeren mulighed for at vælge styrken af dimming. Styrken af dimming valideres og returneres. Hvis brugeren indtaster en værdi uden for det gyldige område, bliver metoden kørt fra begyndelsen, hvor tænd, sluk eller dimming skal vælges igen.

drawAskTime

Metoden **AskTime** fungere ved at brugeren bliver bedt om at indtaste hvilket time-tal på dagen den ønskede kommando skal udføres på. Herefter bliver brugeren bedt om at indtaste det ønskede minut-tal på den valgte time. Tiden valideres, og hvis både time-tal og minut-tal er indenfor de gyldige grænser returneres tiden siden midnat i minutter. $timer * 60 + minutter$ Hvis enten time-tal eller minut-tal er udenfor de gyldige grænser, får brugeren besked om at den indtastede tid er ugyldig og keyboard-bufferen tømmes. Når en knap trykkes ved fortsættet programmet, og metoden bliver startet forfra. Til at vente på en knap trykkes er brugt **_getch()** som kan bruges til at hente hvilken knap på keyboardet er trykket. I dette tilfælde er funktionen brugt til at tjekke om en knap bliver trykket på, og ikke for at gemme inputtet af hvad der er trykket på.

```
1 if (hour >= 0 && hour <= 23 && min >= 0 && min <= 59)
2     return (hour * 60 + min); // returnere tiden siden kl 00
3 else
4     cout << "ugyldig tid, pr\x9Bv igen";
5
6 cin.clear();
7 fflush(stdin); // clear og flusher alt gemt i keyboard bufferen
8 _getch(); // venter paa hvilket somhelst tryk paa keyboard
```

drawStopPromt

drawStopPromt metoden udskriver på skærmen en besked om brugeren er sikker på at afviklingen af scenarie skal afsluttes. Ved at trykke på knappen "Y/y" på keyboardet, bekræftes det at det skal afsluttes og der returnes **true**, en hver anden knap resultere i, at der returneres **false**

```
1 get = _getch();
2 if (get == 89 || get == 121) // ascii værdi for y og Y
3     return true;
4 else
5     return false;
```

8.2.4 PC UART (Kasper)

PC'ens UART har til formål at oversætte de data, der er i *Scenario*-objektet, og transmittere dem over på transmitteren. Dette gøres ved brug af et open-source bibliotek kaldet RS232 [10], der følger den protokol det ønskes at transmittere data med.

For at sende data ud til Transmitteren, skal der oprettes en forbindelse gennem en USB port. her bruges **UART_connect** til formålet.

Systemet kræver også at kunne få status på kodelåsen, hvilket metoden **getLockStatus** står for. Kravet for at bruge **getLockstatus** er, at **UART_connect** har været kørt, for at finde en gyldig port at sende og modtage på.

Når data skal transmitteres bruges **sendScen**, der tager en reference til et scenarie som parameter, og får oversat dataene i scenariet til at følge UART protokollen, hvorefter den sender dem over til transmitteren.

UART_connect

Metoden **UART_connect** bruges til at finde en gyldig COM-port på computeren. Metoden kører igennem alle COM-porte på PC'en ved at bruge RS232-bibliotekets funktion **RS232_OpenComport**, til at tjekke om der sidder et RS232 stik i den givne COM-port. Så længe den testede COM-port ikke har et RS232 stik i, tæller metoden **Cport_nr** en op, og derved tester den næste COM-port, indtil en gyldig COM-port er fundet.

```
1 while (RS232_OpenComport(cport_nr, bdrate, mode))  
2 {  
3     printf("ugyldig Com-port\n");  
4     cport_nr++;  
5 }
```

getLockStatus

Metoden **getLockStatus** finder ud af kodelåsens status ved at sende et L ud på COM-porten. Dette sker ved at bruge RS232-bibliotekets funktion **RS232_cputs()**. Parametrene for funktionen er COM-port nummeret **cport_nr** og den mængde af data i form af chars, der skal sendes.

Da det kræver COM-port nummeret, er det derfor også nødvendigt, at metoden **UART_connect** har været kørt, før det er muligt at bruge **getLockStatus**.

```
1 RS232_cputs(cport_nr, "L");
```

Når beskeden om at tjekke status er blevet sendt til transmitteren, begynder metoden at læse på porten og venter på et svar tilbage fra transmitteren. Svaret vil indeholde et L eller et U, afhængig af om kodelåsen er hhv. låst eller oplåst. Til aflæsning på COM-porten bruges RS232-bibliotekets funktion **RS232_PollComport()**. Hvis det aflæste er et L returneres **true**, for at kodelåsen er låst. Hvis et U aflæses returneres **false**, hvilket betyder kodelåsen er låst op.

sendScen

metoden **Send Scen** har til opgave at oversætte data fra scenariet til chars som følger UART protokolen. til at starte med kører metoden koden nedenfor:

```
1 RS232_cputs(cport_nr, "N");
```

N'et betyder at transmitteren skal være klar på at der nu kommer et ny datastrøm fra PCen. Metoden henter herefter den første aktion, så `get` funktioner kan bruges til at hente data ud af aktionen. Kommandoen bliver sat direkte ind, da `get` funktionen for den returnerer en char, derfor sker ingenting med den. Tiden skal regnes om, så transmitteren ved hvor længe der skal gå, før den skal udføre en kommando. Dette sker i koden set nedenfor:

```

1 //udregner tiden til den gældne commando skal ekseveres
2 int timeTillExecution;
3 if (action.getTime() >= Ctime())
4 {
5     timeTillExecution = action.getTime() - Ctime();
6 }
7 else
8 {
9     timeTillExecution = 1440 - (Ctime()-action.getTime());
10 }
11 //tager sig af at omregne tiden fra int til char og dele tiden op i 2 chars,
12 //da den kan ske at fylde mere end 8 bit
13 int timeL1 = timeTillExecution % 16;
14 int timeL2 = int((floor(timeTillExecution / 16) % 16)
15 int timeH = int((floor(timeTillExecution / 256)));
16
17 TimeLow1 = ((char)timeL1 +48);
18 TimeLow2 = ((char)timeL2+48)
      TimeHigh = ((char)timeH)+48);

```

Integeren `timeTillExecution` bliver udregnet ud fra en af 2 udregning. Den valgte udregning afhænger af om tidspunktet allerede har passeret systems klokkeslet. Hvis dette er tilfældet køres udregningen i linje 9. Denne udregning tager hensyn til hvor mange minutter der er tilbage på dagen og kører derfor på det angivne klokkeslet næste dag. Udregningen i linje 5 sker hvis klokkeslettet endnu ikke har passeret tiden, hvor kommandoen skal udføres. Den udregner hvor mange minutter der er fra det gældende klokkeslet og frem til kommandoen skal udføres. `Ctime()` funktionen som fremgår flere steder er en hjælpefunktionen som returnerer en integer med klokkeslettet på PCen. Når `timeTillExecution` er udregnet, skal den omsættes til Chars. Da den maksimale værdi der vil fremkomme er 1440 (antallet af minutter på et døgn), skal der bruges 2 chars til at gemme dataene i.

```

1 int timeL1 = timeTillExecution % 16;
2 int timeL2 = int((floor(timeTillExecution / 16) % 16)
3 int timeH = int((floor(timeTillExecution / 256)));

```

`timeL1` er hvor de 4 første bit gemmes, derfor tages modulus til 16 for at finde resten, og altså den værdi der skal gemmens i `timeL1`. `TimeL2` er hvor de midterste bit gemmes, derfor nedrundes der, og denne omskrives til en int. Derefter tages modulus til 16. `timeH` er hvor de 4 største bit gemmes. For at finde værdien der skal gemmes heri, divideres `timeTillExecution` med 256, og rundes ned, for præcision. Da nedrundingen sker som en double omskrives det til en int, efter nedrundingen. `timeL1`, `timeL2` og `timeH` har efter udregningen en maksimalstørrelse på 16, og kan derfor laves om til en char, med ASCII kode mellem 48 og 64

```

1 TimeLow1 = ((char)timeL1 +48);
2 TimeLow2 = ((char)timeL2+48)
3 TimeHigh = ((char)timeH)+48);

```

Unit oversættes ved brug af en switch set i koden nedenfor:

```

1 switch (action.getUnit()) // tager sig af at oversætte unit til at følge UART
2     protocollen
3     {
4         case 1:
5             unit = 'A';
6             break;
7         case 2:
8             unit = 'C';
9             break;
10        case 3:
11            unit = 'P';
12            break;
13        case 4:
14            unit = 'N';
15            break;
16        default:
17            unit = 'O';
18            break;
19    }

```

Switchen tager uniten ind fra aktionen, og alt efter værdien sættes unit til den char der passer en den gældende unit.

UARTen sender nu dataene fra aktionen ud på COM-porten ved brug af funktionen RS232_cput

```

1 char data[6] = { unit , command, TimeHigh , TimeLow2, TimeLow1 };// data der skal
2     sendes over
3     strcpy(str [c] , data);
4     RS232_cputs(cport_nr , str [c]);

```

når dataene er sendt på på porten, venter UARTen 100milisekunder, hvorefter den begynder forfra med aktion nr.2. Dette sker til alle 20 aktioner er overført til transmitteren, hvorefter metoden afslutter.

stopAll

metoden stopAll er til at slukke alle enheder på system. Dette gør metoden ved at bruge funktionen RS232_cput til at sende et S til transmitteren. Transmitteren ved ifølge protokolen at et S betyder den skal slukke alt.

```
1 RS232_cputs(cport_nr , "S");
```

8.2.5 PC Main (Henrik)

PC'ens main har til formål at stykke alle PC'ens klasser sammen, og styrer alt fra input til output. Den er kodet således at UC1, UC2 og UC3 er opfyldt. Basalt set er dette en main fil som bruger alle funktioner fra PC'ens klasser: Action, PC UART, Scenario og UI.

Ud over dette er det under denne fil at koden til de predefinerede scenarier ligger.

8.3 Generelt om Microcontrollers (David)

I vores projekt har vi valgt at bruge Atmels ATMega32 microcontroller, fordi vi har kendskab til netop denne via MSYS kurset på første semester, den blev også brugt til vores bil på selv samme semester.

8.3.1 C++

I vores projekt har vi valgt at programmere i C++ frem for C. På microcontrolleren får vi den store fordel at vi kan opbygge alt i klasser frem for frie funktioner.

Dette er valgt fordi det gør det nemmere for os som udviklere at kode hver klasse, og kun have den logik, som skal bruges i denne. Dette gør det nemmere at teste koden, samt at dele softwareopgaverne ud blandt flere personer, fordi at så længe klassen overholder vores designdokument, er det nemt at sætte det hele sammen til sidst, hvor alt kode er skrevet.

Dette giver os den store fordel at vi kan gøre brug af de såkaldte "The Three Pillars of Object-Oriented Programming" (indkapsling, arv og polimorfi), hvor det giver mening.

Vi gør dog kun rigtig brug af Encapsulation da vi ikke har vurderet at, der var nogen steder det ville give mening at anvende Inheritance og Polymorphism. Encapsulation giver rigtig meget mening for os, da information hiding hjælper med at beskytte vigtigt private data, som eksisterer eksempelvis i vores boundry klasser til X.10 kommunikation. I praksis betyder det at kun klassens egen metoder må ændre dets private data. Dette er rigtig smart fordi at man så er helt sikker på at der ikke bliver unødig eller fejlagtigt ændret i en variable som potentielt kan ødelække klassens funktionalitet.

8.3.2 C++ Compiler på Microcontrollerne

Da vi skulle undersøge hvordan man kunne anvende C++ på vores microcontroller, så vi først på Atmels hjemmeside [19] for at finde ud af om der er nogle ting der skal gøres specielt opmærksomt på. Vi startede med at skrive vores kode inline, da der har været god erfaring i gruppen med at gøre det i Visual Studio 2013, dog viste dette sig at give massive problemer når der blev kodet inline i Atmel Studio 6.2. Problemerne var bl.a. at indlæse variabler, som blev erklæret i toppen af dokumentet, men længere nede i metoderne. Derfor blev det vedtaget, at der skulle bruges den klassiske struktur med adskilte header og .cpp filer.

8.3.3 C++ og Interrupts

Der var ingen i gruppen, der havde kendskab til hvordan interrupts på microcontrollerne i C++ fungerede, derfor blev der hentet inspiration fra et open source projekt [20]. Det viste sig at fungere ganske godt og uden de store problemer og gruppen var på kort tid i stand til at opstille en test med interrupts i C++.

8.3.4 C++ på AVR Object Kommunikation

Normalt kan objekter oprettes i en separat cpp fil, hvorefter man kan anvende dot-operatoren til at tilgå et objekts metoder og variable. Dette fungerede ikke i Atmel Studio 6.2, da compileren ikke kunne finde de givne objekter, derfor var man nød til at lade de objekter der skulle interagere med hinanden via en fri funktion, der returnerede en pointer til det givne objekt.

8.4 Transmitter Blokken

8.4.1 Action Klassen (Kristian T.)

Selve Action-klassen består af en række get'er metoder og én set motode, som validerer data og genererer houseCode ud fra unitCode parametren. Der er ikke foretaget nogle videre interessante algoritmer eller yderligere hjælpfunktioner uddover dem, der forefindes i kapitlet Software Design.

8.4.2 Codelock Klassen (Kristian T.)

Implementeringen af denne klasse er blot én metode `locked()`, som returnerer true, hvis kodelåsen er låst (benet er højt).

```
1 bool Codelock::unlocked() {
2     //TRUE if unlocked , FALSE is locked .
3     //Signal is LOW when code is correct , HIGH when not correct .
4     return !(PINA & 0b00000001);
5 }
```

8.4.3 Time Klassen (Kristian S.)

Time klassens roller er at give TransmitterCtrl klassen en opdateret værdi i minutter hver 5 sekund. Dette tager højde for flere events, som vil køre på samme tid og giver Tx10 klassen mulighed for at sende hele aktionen før næste aktion påbegyndes. Klassen er blevet implementeret med en compare interrupt på Timer1, hvor der sker et overflow hvert sekund, som inkrementer en `seconds` variabel. Når denne `seconds` er modulo 0 med 5, sendes `minutes` til ctrl klassen, ellers hvis `second` overstiger 59 bliver den sat til 0 og `minutes` tillægges 1.

```
1 ISR(TIMER1_OVF_vect) {
2
3     //increments the time by 1 sec
4     myTime.seconds++;
5
6     //every 60 seconds updates Minutes value .
7     if (myTime.seconds > 58){
8         myTime.seconds = 0;
9         myTime.minutes++;
10    }
11
12    //every 5 seconds updates transmitters next action time .
13    if (myTime.seconds % 5 == 0){
14        myTime.myTransPtr->checkTime( myTime.minutes );
15    }
16 }
```

8.4.4 Transmitter Klassen (Kristian T.)

Constructor

Til implementering af Transmitter klassen er der lavet en constructor, som initierer de to variabler `charCounter` og `scenCounter`. Uddover dette opretter den et array på 20 pladser, som den fylder med tomme `Action` objekter.

scenData(char input)

Denne metode er implementeret, så den tjekker hvor mange dele af en **Action**, der er gemt og når et helt **Action** objekt er overført, gemmes dette i objekt nummer **scenCounter**, som holder styr på hvor mange **Action** objekter der er overført. Når data gemmes oversættes den samtidigt til de kodemønstre, som X10 protokollen bruger.

checkTime(unsigned int theTime)

Denne metode er implementeres således at den kaldes sig selv rekursivt for at tjekke om der er flere aktioner oprettet på samme tidspunkt. Hvis dette er tilfældet fortsætter den med rekursive kald, indtil der ikke er flere aktioner til tidspunktet eller at alle aktioner i hele scenariet er blevet udført (i det tilfælde, hvor alle aktioner i et scenario ligger samtidigt).

```
1 void Transmitter::checkTime(unsigned int theTime){  
2     if (myScenario[nextAction].getTime() == theTime){ //Check to see if the time has  
3         past the next action to be sent  
4         if (firstLoop){ //Check if this is the first recursive call  
5             breakAt = nextAction; //Set the breaking point for recursive calls  
6             firstLoop = false;  
7         }  
8         else{  
9             if (breakAt == nextAction){ //Check if the breaking point has been reached  
10                 firstLoop = true;  
11                 return; //Break out of the recursive calls , if all the actions in the  
12                 Scenario have been sent simultaniously  
13             }  
14         }  
15         myTx10Ptr->sendAction(myScenario[nextAction]);  
16         nextAction++;  
17         if (nextAction == 20){ //If the last action has been sent , queue the first action  
18             nextAction = 0;  
19         }  
20         checkTime(theTime); //Recursive call , to check if several actions are on the  
21             same time.  
22     }  
23     else  
24         firstLoop = true;  
25 }
```

8.4.5 Tx10 Klassen (Kristian T.)

Klassen fungerer ved at der hele tiden kommer interrupts fra **zeroCrossDetected** signalet, 100 interrupts i sekundet. Datamedlemmet **bool start** bestemmer om Tx10 klassen, skal sende noget ud på nettet eller ej.

Constructor

Constructoren sørger for at initialisere samtlige variabler til værdier, som giver mening. Den initialiserer Timer0, Timer2 og INT0 samt PORTC til debugging med LED-lysene, som sidder på STK500.

Timer0 bruges til at danne et 120 kHz clocksignal, til at beregne de initierede værdier er følgende udregninger lavet. Dette er når $f_{osc} = 3.6864\text{MHz}$, $N = 1$ og $OCR0 = 14$.

$$f = \frac{f_{osc}}{2N(1 + OCR0)} = 122.88\text{kHz}$$

unsigned char Translate(unsigned char bitCode)

Hjælpemetode, som oversætter 4-bit tal til en char indeholdende 8 nulgennemgange. Denne er implementeret med **if**-sætninger, som tjekker på de enkelte bits i **bitCode**.

```

1 unsigned char Tx10::translate( unsigned char bitCode ){
2     unsigned char temp = 0;
3     if (bitCode & 0b00001000){
4         temp |= 0b10000000;
5     }
6     else{
7         temp |= 0b01000000;
8     }
9     if (bitCode & 0b00000100){
10        temp |= 0b00100000;
11    }
12    else{
13        temp |= 0b00010000;
14    }
15    if (bitCode & 0b00000010){
16        temp |= 0b00001000;
17    }
18    else{
19        temp |= 0b00000100;
20    }
21    if (bitCode & 0b00000001){
22        temp |= 0b00000010;
23    }
24    else{
25        temp |= 0b00000001;
26    }
27    return temp;
28 }
```

sendAction(Action &)

Metoden starter med at gemme **Housecode**, **Unitcode** og **Command** fra aktionen i **Tx10** klassens egne variabler ved hjælp af **Translate()** og sætter **start** til TRUE.

sendCrossing(unsigned char send)

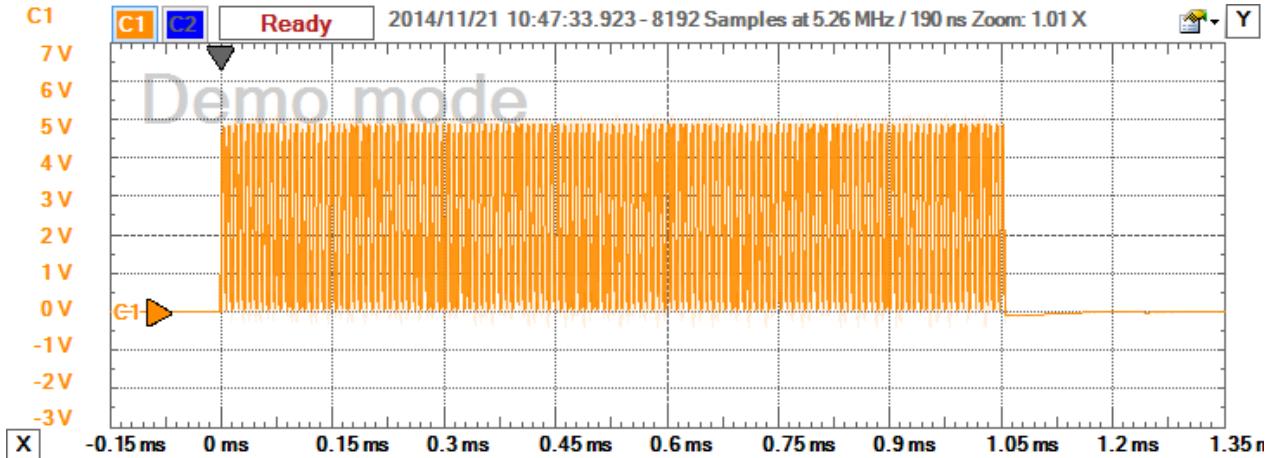
Hjælpemetode, som åbner op for 120 kHz firkantsignal, hvis send er forskellige fra 0.

Metoden er testet til at give outputtet i Figur 50 på STK500 når den kaldes med en char forskellige fra 0.

Frekvensen på firkantsignalet er med Analog Discovery målt til 122 kHz.

waitMs()

Hjælpemetode, som starter timer2 på 1 ms og venter til denne er færdig inden metoden forlades. Tiden er regnet ud ud fra følgende formel:



Figur 50: Output på X.10Data når en der skal sendes 1 i en nulgennemgang.

$$t = \frac{1}{f_{osc}} N(C - 1) = 1.00ms$$

Hvor N er sat til 64 og C er sat til 59. N er prescaleren på clockfrekvensen for ATMega32 og C er antallet af tællinger der skal til. Dvs at der skal skrives $255 - C$ til OCR2 registeret.

intHandler()

Hjælpekode, som håndterer interrupts fra `zeroCrossDetected` benet.

Starter med at tjekke om `start` er TRUE, hvis den er det fyldes `buffer` variablen med de næste nulgennemgange, som skal sendes.

Starter med at sende MSB fra bufferen og skifte bufferen til venstre, indtil at bufferen er tom (`buffCount == 0`). Afhængigt af hvor mange mønstre (`pattCount`), som er sendt fylder den det næste mønster i bufferen. Når alle mønstre er sendt sætter den `start` til false.

```

1 void Tx10::intHandler() {
2     if (start){ //check if the Tx is supposed to transmit
3         if (buffCount == 0){
4             if (pattCount == 0){
5                 buffer = 0b11101111;
6                 buffCount = 4;
7             }
8             else if (pattCount == 1){
9                 buffer = house;
10                buffCount = 8;
11            }
12            else if (pattCount == 2){
13                buffer = unit;
14                buffCount = 8;
15            }
16            else if (pattCount == 3){
17                buffer = command;
18                buffCount = 8;
19            }
20            else if (pattCount == 4){
21                buffer = 0b00000011;
22                buffCount = 6;
23            }
}
```

```

24     else if (pattCount == 5){
25         buffer = house;
26         buffCount = 8;
27     }
28     else if (pattCount == 6){
29         buffer = unit;
30         buffCount = 8;
31     }
32     else if (pattCount == 7){
33         buffer = command;
34         buffCount = 8;
35     }
36     else if (pattCount == 8){
37         buffer = 0b11110000;
38         buffCount = 4;
39     }
40     else if (pattCount == 9){
41         start = false;
42         pattCount = 0;
43     }
44     pattCount++;
45 }
46
47 if (buffCount > 0){
48     sendCrossing((buffer & 0b10000000)); //Send out the MSB
49     buffCount--;
50     buffer = (buffer << 1); //shifts the buffer one time to the left
51 }
52 }
53 PORTC = ~getMyTx10()->buffer; //For debugging
54 }
```

turnAllOff()

Metode der fylder **house** variablen med 0 (koden for at slukke alt) og sætter **start** til TRUE. Dette medfører at ved de efterfølgende nulgennemgang sender mønstret for at slukke alt på nettet.

8.4.6 TxUART Klassen (Kristian T.)

Denne klasse er overordnet implementeret til brug i projektet ved overførsel af scenarier, men bruges også til dels til debugging af de øvrige klasser.

Constructor

Constructoren sørger for at initiere **mode** til 'i' (idle) og initierer ellers UART på almindelig vis i henhold til protokollen.

rxInt()

Hjælpemetode, som håndterer interrupts når der modtages en **char** på UARTEn. Metoden er implementeret ved at den kontrollerer hvilken tilstand, klassen er i. Hvis den er i idle udføres den kommando, som er sendt via UART. Hvis den er i receiving mode og der ikke er send 80 chars endnu, sendes der data til **Transmitter** klassen.

```

1 void TxUART::rxInt() {
2     char input = UDR;
```

```

3 if (mode == 'i'){
4     if (input == 'L'){
5         bool temp = myTrans->getLockStatus();
6         if (temp)
7             sendChar('U');
8         else
9             sendChar('L');
10    }
11    else if (input == 'N'){
12        mode = 'r'; //put in receiving mode
13        myTrans->newScen();
14    }
15    else if (input == 'S'){
16        myTrans->stopAll();
17    }
18    else if (mode == 'r'){
19        if (charNo <= 80){
20            myTrans->scenData(input);
21            charNo++;
22        }
23        if (charNo == 80){ //end of data
24            charNo = 0; //reset counter
25            mode = 'i'; //put in idle mode
26        }
27    }
28 }
29 }
```

sendChar(char)

Hjælpemetode, sender en char via UART.

sendString(const char * sendMe)

Hjælpemetode til debugging, sender en række chars via UART, modtager "string literal", som de forekommer i C++. Dvs man fx kan skrive `sendString("Hello world.")`.

sendNumber(int sendMe)

Sender en integer via UART, som oversættes til ASCII-værdier først. Bruges primært til debugging af systemet.

```

1 void TxUART::sendNumber( int sendMe){
2     sendChar(((sendMe / 1000)%10)+48);
3     sendChar(((sendMe / 100)%10)+48);
4     sendChar(((sendMe / 10)%10)+48);
5     sendChar(( (sendMe)%10)+48);
6 }
```

8.5 Receiver

Receiveren modtager et signal fra ZeroCross detectoren, når der er et 120khz signal på nettet, og derefter læser den fra bitstreamen som indeholder dataen fra transmitteren.

Klassen har en indbygget debugging-funktion, som udnytter RxUART klassen til at "sladre" om hvad der foregår. Debugging kan aktiveres ved at sætte konstanten `debugging` til `true`. Dette hjælper os rigtig meget under implementering da det kan se helt præcis, hvor langt den er nået i modtagelsen af data.

8.5.1 Rx10 Klassen (David)

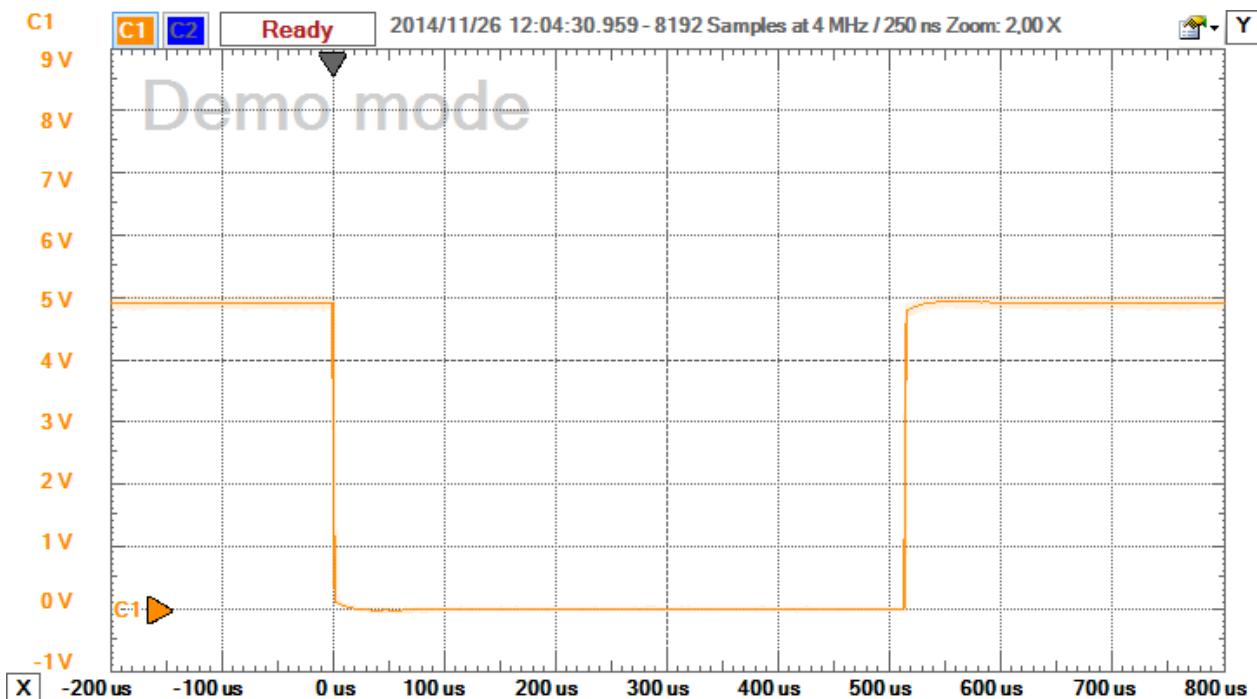
`waitMs()`

Denne hjælpemetode er til for at være sikker på at ramme midt i dataen for en given nulgennemgang. Da disse er $1ms$ lange, skal systemet læse $0.5ms$ efter nulgennemgangen. På denne måde sikres man at ramme dataen, hvis nu timingen skulle skride en smule.

Metoden fungerer ved at starte timer2 på $0.5ms$, dette er regnet ud ud fra følgende udregninger:

$$\begin{aligned} t &= \frac{1}{f_{osc}} N(C - 1) \\ &= \frac{1}{3.6864\text{MHz}} 64(29 - 1) \\ &= 0.5ms \end{aligned}$$

Metoden er testet ved at sætte et ben lavt før og højt igen efter metoden og dette er målt med Analog Discovery i Figur 51. Det ses at timingen passer meget godt med 0.5 ms.



Figur 51: Måling af `waitMs()` metoden.

interruptHandler()

Hjælpemetode til at håndtere interrupts fra `zeroCrossDetected`.

Metoden tjekker i første omgang på hvert interrupt om den har modtaget start-koden fra X.10 protokollen (se s. 62 i protokol afsnittet).

```
1 waitMs(); // wait 0.5 ms
2 buffer = buffer << 1; // Make room for the next bit
3 buffer |= ( PINA & 0b00000001 ); // Adds the bit in the buffer
4 PORTC = ~buffer;
5
6 if (!start){
7     if ( ( buffer & 0b00011111 ) == 0b00001110 ){
8         start = true;
9         pattCount = 2;
10        buffCount = 7;
11        if (debugging){
12            myRxUART->sendString("New command!\n\r");
13        }
14    }
15 }
```

Listing 8.1: Der ventes 0.5 ms og der tjekkes efter startbit.

Når start-koden er fundet, sættes `start` til TRUE og `pattCount` samt `buffCount` initieres. De efterfølgende otte nulgennemgange fyldes bufferen med data fra `bitstream`. Herefter tjekkes det om dataen er valid i forhold til protokollen (så at en høj nulgennemgang efterfølges af en lav og vice versa). Er protokollen overholdt oversættes dataen og gemmes i en variabel.

```
1 if (checkData(buffer)){
2     if (pattCount == 2){ // checks if it is in pattern house
3         house = translate(buffer);
4         buffCount = 7;
5
6         if (debugging){
7             myRxUART->sendString("HouseCode: ");
8             myRxUART->sendNumber(translate(buffer));
9             myRxUART->sendString("\n\r");
10        }
11        ...
12    }
13    ...
14 } else{
15     start = false;
16     if (debugging){
17         myRxUART->sendString("Invalid data!\n\r");
18     }
19 }
```

Dette gentages for tre fyldte buffere (24 interrupts/nulgennemgange), hvorefter der forventes 6 tomme nulgennemgange. Er dette også iorden kontrolleres de næste 24 nulgennemgange om de matcher de foregående, som indeholdte data omkring Actionen. Hvis disse også er okay, ventes der på en slutkode, som beskrevet i protokollen. Er der fejl i blot én af ovenstående check kasseres alt og der ventes igen på start-koden.

Under processen blev det overvejet også at lave en form for fejl retning, men dette ikke gav mening, da der kun er implementeret énvejskommunikation og istedet bliver data dobbeltjekket i receiveren.

translate(unsigned char zeroCrossingCode)

Er implementeret ved at tage de interessante bits og gemme dem over i en lokal variabel, som herefter returneres.

```
1 unsigned char Rx10::translate( unsigned char zeroCrossingCode ){
2     char temp = 0;
3     temp = temp | (0b00000001 & (zeroCrossingCode >> 1) ); // Shifts bit 1 into to temp
4     temp = temp | (0b00000010 & (zeroCrossingCode >> 2) ); // Shifts bit 2 into to temp
5     temp = temp | (0b00000100 & (zeroCrossingCode >> 3) );
6     temp = temp | (0b00001000 & (zeroCrossingCode >> 4) );
7     return temp;
8 }
```

8.5.2 Receiver Control Klassen (David og Lasse)

Receiver klassen fungerer ved, at Rx10 klassen kalder en funktionen *Input()*, med 3 chars, (*houseCode*, *unitCode* og *command*) som parametre. Først tjekker funktionen om *houseCode* er lig nul, da dette er en global commando som betyder sluk alt. Dernæst tjekker den om *houseCode* og *unitcode* passer til den respektive receiver. Hvis disse passer til receiveren, vil funktionen, afhængig af kommandoen kalde en funktion i lampe klassen via en pointer til denne.

Litteraturliste

- [1] Fairchild: *LM7805 datasheet*. "Bilag 017 - LM7805". 2006.
- [2] Maxim: *IC7660 datasheet*. "Bilag 018 - IC7660". 1994.
- [3] NXP: *1N4148 datasheet*. "Bilag 019 - 1N4148". 2010.
- [4] Farnell: *TS912 datasheet*. "Bilag 020 - TS912". 2012.
- [5] Fairchild: *BC547 datasheet*. "Bilag 021 - BC547". 2002.
- [6] Fairchild: *BD139 datasheet*. "Bilag 022 - BD139". 2007.
- [7] Kingbright: *L-53-YD 5mm datasheet*. "Bilag 023 - L-53-YD 5mm". 2006.
- [8] Atmel: *ATMega32 datasheet*. "Bilag 024 - ATMega32". 2011.
- [9] Mascot: *8610 AC/AC adaptor*. "Bilag 010 - 8610AC Adaptor". Ukendt årstal.
- [10] Beelen, Teunis van: *RS-232 for Linux, FreeBSD and Windows*. <http://www.teuniz.net/RS-232/>. 2014-12-11.
- [11] Burroughs, Jon: *AN236*. "Bilag 011 - AN236". 2002.
- [12] Krogh-Pedersen, Philip: *Kode fra Exercise 7.2 DSD*. "Bilag 012 - Codelock". 2014.
- [13] Gruppe 1: *Samarbejds aftale*. "Bilag 013 - Samarbejds aftale". 2014.
- [14] Gruppe 1. *Mødereferater PRJ2*. "Bilag 014 - Mødereferater". 2014.
- [15] Hargaard, Henning: *Summary from the Atmel Mega32 manual*. "Bilag 016 - AtMega32 summary". 2011.
- [16] Gruppe 1: *Testscenarie-dokumentet*. "Bilag 003 - Testscenarie-dokumentet". 2014.
- [17] ColorSchemer: *ColorSchemer - Free screen color picker from ColorSchemer*. http://www.colorschemer.com/colorpix_info.php. 2014-12-14.
- [18] Wikipedia: *Baker clamp*. http://en.wikipedia.org/wiki/Baker_clamp. 2014-12-11.
- [19] Atmel: *C++ compiling*. http://www.atmel.com/webdoc/AVRLibcReferenceManual/FAQ_1faq_cplusplus.html. 2014-10-22.
- [20] Waterproofman: *C++ interrupts* <http://waterproofman.wordpress.com/2007/02/07/avr-interrupts-in-c/>. 2007-02-07.