

Projektdokumentation
AutoGreen
Gruppe 9

3. Semesterprojekt E3PRJ3-02
Ingeniørhøjskolen, Aarhus Universitet
Vejleder: Tore Arne Skogberg

14. maj 2015

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	1
1.1 Version	1
1.2 Beskrivelse	1
1.3 MoSCoW prioritering	2
1.4 Rigt Billede	3
2 Kravspecifikation	4
2.1 Version	4
2.2 Systembeskrivelse	4
2.3 Ordforklaring	6
2.4 Brugerfladen	7
2.5 Aktør Kontekst Diagram	8
2.5.1 Aktørbeskrivelser	8
2.6 Funktionelle Krav	9
2.7 Ikke Funktionelle Krav	9
2.8 Use Case Diagram	10
2.8.1 Use Case beskrivelser - Initiering og Formål	12
2.8.2 Use Case Beskrivelser - Fully Dressed	14
3 Accepttest	23
3.1 Version	23
3.2 Funktionelle Krav	23
3.3 Ikke-funktionelle krav	31
4 Systemarkitektur	34
4.1 Version	34
4.2 Indledning	34
4.3 Hardwarearkitektur	34
4.3.1 BDD for System	35
4.3.2 IBD'er for System	36
4.3.3 IBD for Aktuator	37
4.3.4 IBD for Jordfugt	38
4.3.5 Signalbeskrivelser	39
4.4 Softwarearkitektur	40
4.4.1 Applikationsmodel	40
4.4.2 Controller-Klasser	40
4.4.3 Boundary-Klasser	40
4.4.4 Domain-Klasser	41
4.4.5 Menuoversigt	42
4.4.6 Menubeskrivelse	42
4.5 Protokol for UART	44
4.5.1 UART indstillinger	44
4.5.2 Datavalidering	44
4.5.3 Kommandoer	44

5 Hardware Design	47
5.1 Version	47
5.2 I ² C Protokol	47
5.2.1 Temperatursensor	47
5.2.2 Slave Aktuator	48
5.2.3 Slave Jordfugt	50
5.3 PSoC Master	51
5.3.1 Klassebeskrivelser	52
5.3.2 Sekvensdiagrammer	64
5.4 Aktuator Design (Henrik og Morten)	66
5.4.1 Varmelegeme	66
5.4.2 Blæsere	68
5.4.3 Vinduesmotor	70
5.4.4 PSoC4	72
5.4.5 Drivers til PSoC4	72
5.5 Strømforsyning Design (Henrik og Morten)	74
5.6 Jordfugt Design (Henrik og Morten)	76
6 Software Design	77
6.1 Version	77
6.2 Indledning	77
6.3 Udvidet Applikationsmodel: Sekvensdiagrammer	77
6.3.1 Usecase 1: Start	77
6.3.2 Usecase 2: Stop	78
6.3.3 Usecase 4: Administrer Drivhus	79
6.3.4 Usecase 5: Vis Historik	82
6.3.5 Usecase 6: Administrer Plantedatabase	83
6.3.6 Usecase 7: Konfigurer System	84
6.3.7 Usecase 8: Se Systemlog	88
6.3.8 Usecase 9: Rapportering	89
6.3.9 Usecase 10: Monitorering	90
6.3.10 Usecase 11: Regulering	91
6.4 Klassebeskrivelser	93
6.4.1 Domainklasse Datalog	93
6.4.2 Domainklasse Indstillinger	95
6.4.3 Boundaryklasse Monitor	100
6.4.4 Domainklasse Plantedatabase	101
6.4.5 Boundaryklasse Rapport	103
6.4.6 Boundaryklasse Regulator	104
6.4.7 Domainklasse Systemlog	105
6.4.8 Boundaryklasse UART	106
6.5 Trådhåndtering	107
7 Hardware Implementering	108
7.1 Version	108
7.2 PSoC Master implementering	108
7.2.1 Main implementering	108
7.2.2 I ² C implementering	110
7.2.3 UART implementering	112

7.2.4	DSP implementering	113
7.2.5	Controller implementering	115
7.3	Aktuator	119
7.3.1	HW PSoC4	119
7.3.2	SW PSoC4	120
7.3.3	HW Varmelegeme	127
7.3.4	HW Blæsere	129
7.3.5	HW Vinduesmotor	130
7.4	Strømforsyning	131
7.5	Jordfugt	133
7.5.1	HW PSoC4	133
7.5.2	SW PSoC4	134
8	Software Implementering	135
8.1	Version	135
8.2	GUI - QT	135
Litteraturliste		138

Forfattere

Afsnit	Forfatter(e)
1 Projektformulering	Alle
2 Kravspecifikation	Alle
3 Accepttest	Alle
4 Systemarkitektur	Alle
5 Hardware Design	Kristian T, Philip, Lasse, Henrik og Morten
6 Software Design	David, Kasper og Kristian S.
7.2 PSoC Master implementering	Philip, Lasse og Kristian T
7.3 Aktuator	Henrik og Morten
7.4 Strømforsyning	Henrik og Morten
8 Software Implementering	David, Kasper og Kristian S.

1 Projektformulering

1.1 Version

Dato	Version	Initialer	Ændring
26. februar	1	MHG	Første udkast.
4. marts	2	LBS	Mindre rettelser efter første review.
12. marts	3	MHG	Mindre rettelser efter fælles gennemlæsning.

1.2 Beskrivelse

Mange har prøvet at kaste sig ud i et nyt projekt, som for eksempel at dyrke frugt og grønt i drivhus, men pludselig glemmer man at vande, holde øje med temperaturen og lignende, og så er projektet gået i vasken.

AutoGreen hjælper den nye drivhusbruger med at holde styr på basale parametre som temperatur og fugtighed, men det er også for den mere erfarne drivhusbruger, som ønsker optimale forhold i drivhuset, eller som ønsker at vælge de mest egnede planter ud fra de forhold, der er i drivhuset. Ved dyrkning af planter i et drivhus, er temperaturen en af de vanskeligste ting at kontrollere. Man er ikke altid hjemme, når drivhuset skal åbnes og lukkes, hvilket sjældent er samme tid på dagen; det afhænger af udendørstemperatur, skydække mm. Der findes mekaniske vinduesåbnere, som åbner og lukker et eller flere vinduer i drivhuset vha. en gasfyldt cylinder. Disse er dog forholdsvis upræcise, og reguleringen af temperaturen er langsom. Der er desuden ikke mulighed for at få ekstra varme tilført, hvilket kan være et stort problem, hvis vejret er ustabilt, særligt i starten af sæsonen. AutoGreen styrer temperaturen i drivhuset vha. en vinduesåbner, tovejs luftcirculation og et varmelegeme. Dette giver en hurtig og præcis regulering af temperaturen. Varmelegemet tilfører ekstra varme, hvis der er for koldt i drivhuset. Dette kan meget vel redde planterne, hvis det viser sig, at man har plantet ud for tidligt, og det giver mulighed for at forspire i drivhuset, selv om drivhussæsonen ikke er startet. Hvis der er for varmt i drivhuset, åbner vinduet, og hvis dette ikke er tilstrækkeligt, anvendes også luftcirculationen til at regulere temperaturen. Brugeren har mulighed for at vælge mellem forskellige måder at styre temperaturen på. Ønskes optimale forhold hurtigst muligt døgnet rundt, anvendes både varmelegeme, vinduesåbner og luftcirculation. Brugeren kan også vælge fx at udelade brugen af varmelegemet eller luftcirculationen, hvis en mere økonomisk temperaturregulering ønskes.

En anden vigtig parameter for drivhusplanternes trivsel er selvfølgelig vanding, hvilket ligesom regulering af temperaturen kan være problematisk, hvis man ikke er hjemme, eller man ganske simpelt glemmer det. AutoGreen kan vha. en eller flere fugtmålere i drivhusjorden give brugeren besked om, at det er tid til at vande, ligesom et tilkoblet automatisk vandingssystem kan aktiveres. Et sådant vandingssystem er ikke en del af AutoGreen. Forskellige planter kræver forskellig mængde vand, og brugeren har derfor mulighed for at bruge op til seks fugtmålere, som kan placeres i jorden ved forskellige plantetyper.

AutoGreen mäter desuden luftfugtighed og lysmængde i drivhuset; disse målinger logges sammen med målinger af fugtighed i jorden og temperaturmålinger. Brugeren kan vha. en database med de mest almindelige drivhusplanter vælge, hvad han vil dyrke i sit drivhus, eller han kan forsøge at optimere forholdene i drivhuset, hvis han ønsker bedre forhold for en bestemt type plante. Brugeren har mulighed for at tilføje ekstra planter i databasen.

AutoGreen systemet kontrolleres af brugeren vha. en grafisk brugerflade med touch display, der realiseres på et Embest DevKit8000 Evaluation Board. [1] Alle sensorer og aktuatorer samt systemets masterenhed realiseres vha. PSoC4 udviklingsboards (CY8CKIT-042). [2]

1.3 MoSCoW prioritering

Ambitionen for dette projekt er som absolut minimum at realisere nedenstående punkter under "*skal*". Det forventes desuden at punkterne under "*bør*" realiseres, men de har lavere prioritet. Punkterne under "*kan*" forventes ikke realiseret, og punkterne under "*vil ikke...*" realiseres med sikkerhed ikke. Sidstnævnte punkter kan ses som udviklingsmuligheder i forhold til senere versioner af systemet.

- **Systemet skal:**

- Kunne monitorere temperaturen i drivhuset og regulere temperaturen i drivhuset vha. varmelegeme, åbning af vinduer og luftcirculation.
- Give brugeren mulighed for at vælge varmelegeme og/eller luftcirculation fra, hvis en mere økonomisk regulering af temperaturen ønskes.
- Have et grafisk user interface.

- **Systemet bør:**

- Måle jordfugtighed med op til seks sensorer i drivhuset og give brugeren besked på displayet om, at det er tid til at vande.
- Måle Lysintensitet og luftfugtighed i drivhuset.
- Indeholde en log over alle målte parametre; jordfugtighed, temperatur, luftfugtighed og lysmængde. Dataene præsenteres grafisk for brugeren.
- Indeholde en database over de mest almindelige drivhusplanter, så brugeren kan orientere sig om en plantes optimale forhold.
- Indeholde en systemlog, som noterer vigtige system hændelser.

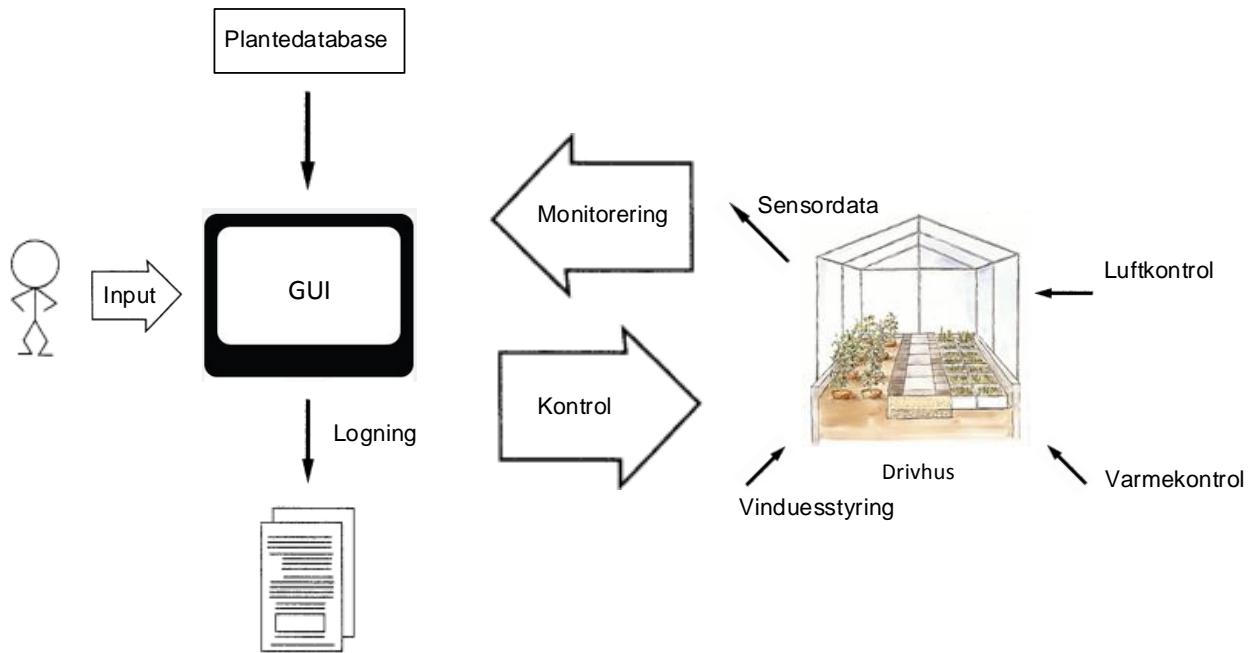
- **Systemet kan:**

- Sende besked til brugeren via email, om at det er tid til at vande. Tilkobles et automatisk vandingssystem, som aktiveres ved behov for vanding.
- Give brugeren mulighed for at tilføje planter i databasen.
- Give brugeren mulighed for at kommunikere trådløst med systemet fra brugerfladen, så denne kan placeres fx inde i brugerens bolig.

- **Systemet vil ikke i denne version:**

- Indeholde et kamera, og tilhørende billedarkiv, som giver brugeren mulighed for at følge planternes udvikling fra dag til dag.
- Give brugeren mulighed for at agere med systemet via en app på dennes mobiltelefon.

1.4 Rigt Billede



Figur 1: AutoGreen Automatiseret Drivhus

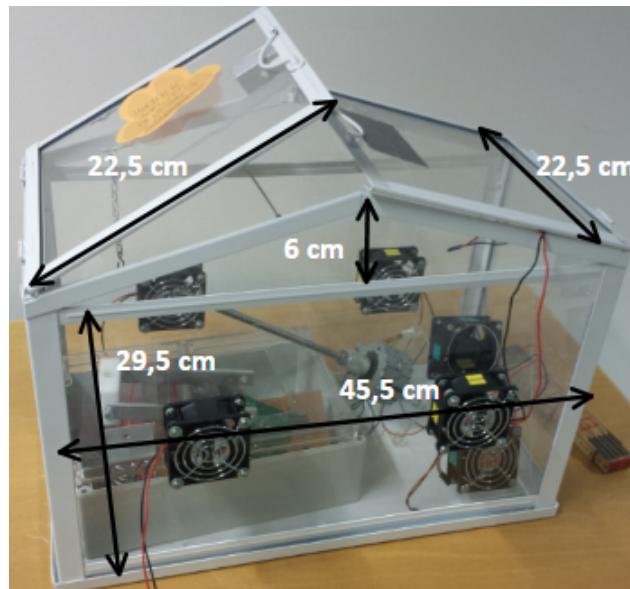
2 Kravspecifikation

2.1 Version

Dato	Version	Initialer	Ændring
26. februar	1	MHG	Første udkast.
6. marts	2	MHG	Rettelser efter review.
12. marts	3	MHG	Mindre rettelser efter fælles gennemlæsning.

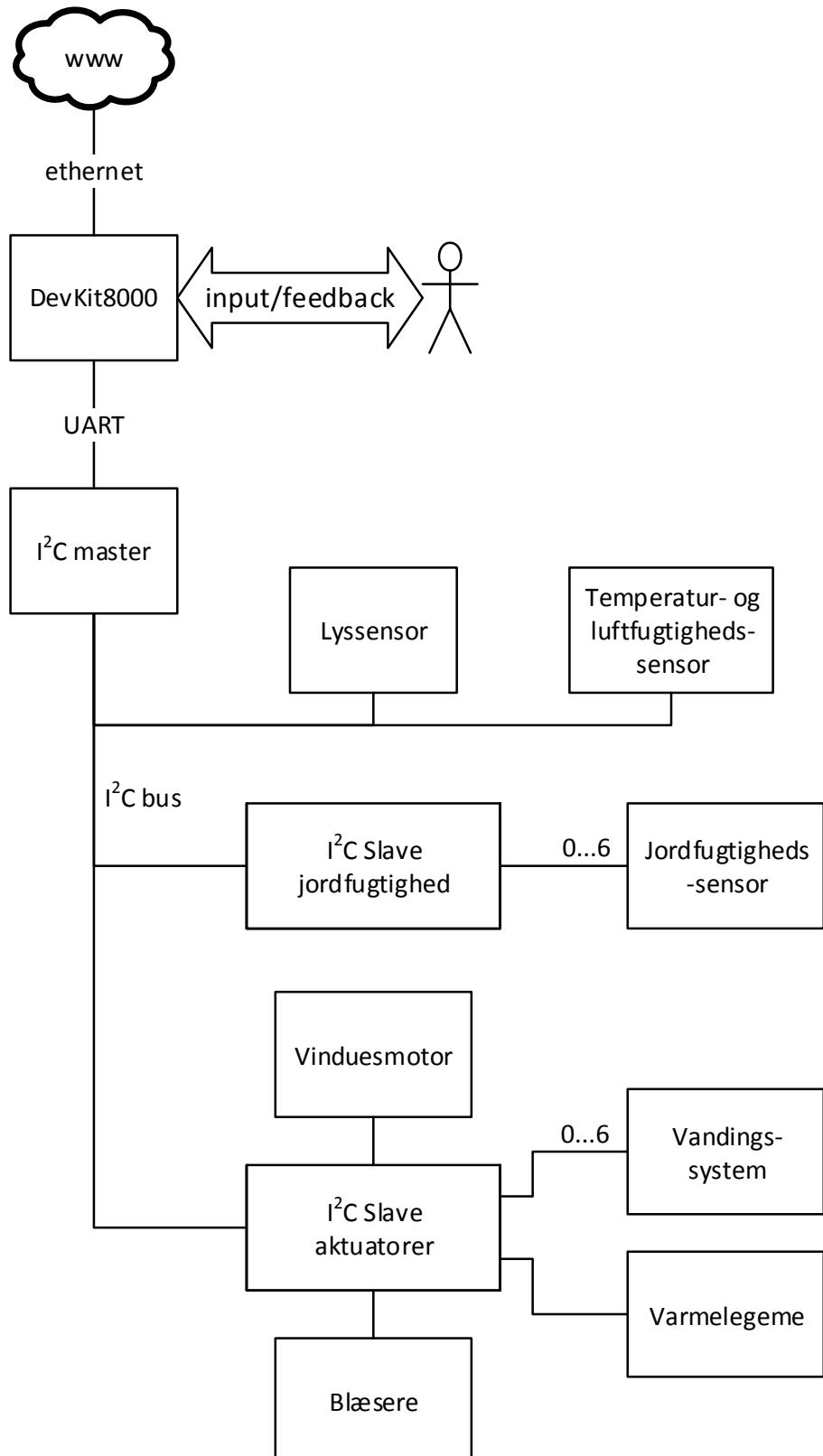
2.2 Systembeskrivelse

Under udviklingen af prototypen for AutoGreen, anvendes en drivhusmodel, der er vist på Figur 2.



Figur 2: Dimensioner for drivhus.

På billedet ses blæsere samt vinduesmotoren (ikke monteret). Disse indgår som en del af systemet, men selve drivhuset gør ikke. Der vil i systemet ydermere være et varmelegeme, som ikke er repræsenteret på billedet.



Figur 3: Oversigt over system

DevKit8000

DevKit8000 er systemets kontrolenhed og brugergrænseflade. DevKit8000 modtager input fra brugeren på dens touch skærm, og den kan give output til brugeren på skærmen og via e-mail; den er koblet til internet via ethernet. DevKit8000 kommunikerer vha. UART med en I²C Master.

I²C Master

I²C Master er realiseret på et PSoC4 udviklingsboard (CY8CKIT-042). I²C Master modtager input fra DevKit8000 og sender/modtager data til/fra I²C Slaver, hvorefter respons sendes retur til DevKit8000.

I²C Slave Jordfugtighed

I²C Slave Jordfugtighed er ansvarlig for alle handlinger og målinger, der har at gøre med vanding i det fysiske drivhus. Der kan tilkobles 0 - 6 jordfugtighedsensorer med tilhørende aktuator til et evt. vandingssystem. Selve vandingssystemet er ikke en del af AutoGreen, en vandingsaktuator er en high/low bool. Enheden er realiseret på et PSoC4 udviklingsboard (CY8CKIT-042).

I²C Slave Aktuatorer

I²C Slave Aktuatorer er ansvarlig for al kommunikation mellem I²C Master og alle aktuatorer i det fysiske drivhus. Enheden er realiseret på et PSoC4 udviklingsboard (CY8CKIT-042).

2.3 Ordforklaring

Plantedatabase

Plantedatabasen indeholder information om ideelle forhold for forskellige typer planter, som brugeren kunne tænkes at plante i sit fysiske drivhus. Informationen i plantedatabasen står til grund for udgangsparametre for nye planter i det virtuelle drivhus. Der findes en række systemplanter, som brugeren ikke kan redigere eller slette, men brugeren kan tilføje egne planter.

Data Log

Systemet er udstyret med en log over de indsamlede data fra sensorer i systemet, der måles og indskrives i loggen hvert minut. Denne er opbygget som en database, hvor hver logning indeholder information fra de diskrete sensorer samt et tidspunkt.

System Log

Systemet er udstyret med en log over hvad systemet foretager sig. Dette kunne f.eks. være et indlæg når systemet foretager en måling, sender en e-mail, regulerer miljøet i drivhuset.

Virtuelt Drivhus

Det virtuelle drivhus er systemets repræsentation af det fysiske drivhus. Brugeren kan tilføje planter fra plantedatabasen i det virtuelle drivhus, og på den måde give systemet indirekte oplysninger om ønskede parametre. Disse informationer lagres i systemets konfigurationsfil.

Fysisk Drivhus

Ved det fysiske drivhus forstås det drivhus hvori systemet er monteret.

Konfigurationsfil

Dette er en automatisk genereret fil, der er placeret på DevKit8000, som indeholder brugerens konfigurationer om blandt andet notifikationer, e-mailadresser, antallet af fugtsensorer og deres unikke id mm.

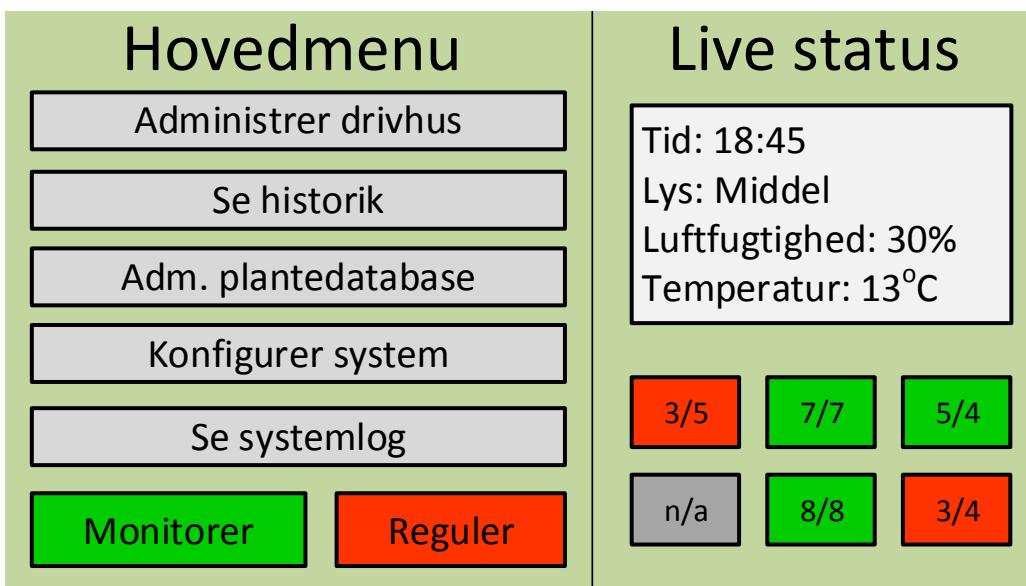
Notifikations E-mail

Dette er en daglig E-mail, som brugeren kan vælge at få tilsendt. Den sendes klokken 12:00, og indeholder informationer om parametrene i det fysiske drivhus.

Advarsels E-mail

Dette er en E-mail, som brugeren kan vælge at få tilsendt. Den sendes, hvis en parameter i det fysiske drivhus kommer uden for tolerancen af den ønskede værdi.

2.4 Brugerfladen



Figur 4: Skitse af hovedmenuen på brugerfladen.

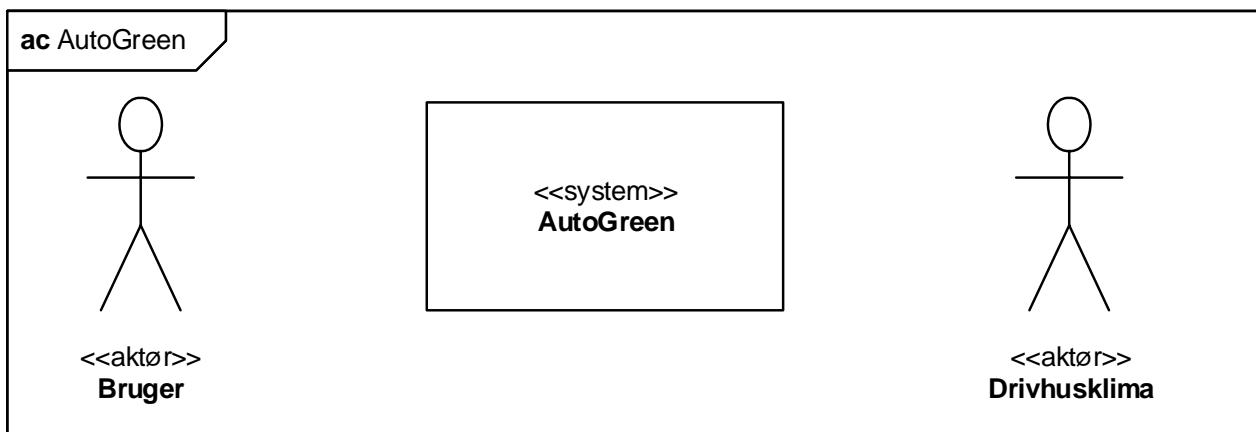
I Figur 4 er vist en skitse over hvordan brugerfladen forventes at se ud. De grå områder under Hovedmenu er knapper, brugeren kan trykke på for at tilgå yderligere menuer.

Nederst ses "Monitorér" og "Regulér" knapper, som kan aktivere eller deaktivere hhv. monitorerings- og reguleringsfunktionalitet.

Til højre ses live status for det fysiske drivhus. Nederst ses live status for jordfugtighed for hver plante. Dette er vist ved seks felter i forskellige farver. Disse symboliserer planter i det virtuelle drivhus, og viser status for den enkelte plante. Grøn betyder at plantens jordfugtighed er indenfor

tolerancerne, hvor rød betyder at den er uden for tolerancen. Grå (Not Available) betyder, at der ikke er placeret en plante i det virtuelle drivhus, for den pågældende fugtighedssensor.

2.5 Aktør Kontekst Diagram



Figur 5: Aktør Kontekst Diagram for AutoGreen

2.5.1 Aktørbeskrivelser

Bruger - Primær Aktør

Brugeren kan:

- Starte og stoppe systemet
- Overvåge det aktuelle klima i drivhuset.
- Administrere drivhuset, hvilket vil sige at han giver systemet input om hvilke planter der er i drivhuset.
- Se historik over klimaet i drivhuset
- Konfigurere systemindstillinger
- Se systemlog
- Modtage rapportering om klimaet i drivhuset
- Administrere planter i plantedatabasen

Drivhusklima - Sekundær Aktør

Drivhusklimaet består af en række parametre, som systemet mäter og/eller regulerer:

- Lufttemperatur

Måles, registreres og reguleres af systemet. Reguleringen sker vha. vinduesåbner, blæsere og varmelegeme.

- Jordfugtighed
 - Måles, registreres og reguleres indirekte af systemet
- Luftfugtighed
 - Måles og registreres af systemet
- Lysintensitet
 - Måles og registreres af systemet

2.6 Funktionelle Krav

Systemet...

1. ... *Skal* give bruger mulighed for at monitorere og konfigurere drivhusklimaet vha. en grafisk brugerflade på et touch display.
2. ... *Skal* have mulighed for at starte og stoppe systemet.
3. ... *Skal* måle lufttemperatur i det fysiske drivhus.
4. ... *Skal* kunne regulere temperatur i det fysiske drivhus.
5. ... *Skal* kunne indstilles til brugerdefineret tid og dato.
6. ... *Skal* kunne give bruger mulighed for at vælge brug af varmelegeme og ventilatorer.
7. ... *Skal* give bruger mulighed for at tilføje en plante i det virtuelle drivhus.
8. ... *Skal* give bruger mulighed for at fjerne en plante i det virtuelle drivhus.
9. ... *Skal* give bruger mulighed for at redigere en plante i det virtuelle drivhus.
10. ... *Skal* kunne regulere drivhusklima automatisk efter behov.
11. ... *Bør* kunne måle jordfugtighed i fysiske drivhus.
12. ... *Bør* kunne måle lysintensitet i det fysiske drivhus.
13. ... *Bør* kunne måle luftfugtighed i det fysiske drivhus.
14. ... *Bør* indeholde informationer om planter i en datastruktur.
15. ... *Bør* kunne fremvise grafisk historik over måledata fra drivhus.
16. ... *Bør* kunne vise planteinformationer fra plantedatabasen.
17. ... *Bør* give bruger mulighed for at se en systemlog over hændelser i systemet.
18. ... *Bør* gemme alt monitorering i en data log.
19. ... *Kan* give bruger mulighed for at redigere og slette planter i plantedatabasen, som bruger selv har tilføjet.
20. ... *Kan* give bruger mulighed for at tilføje/redigere/slette e-mail adresser.
21. ... *Kan* give bruger mulighed for valg af varslings e-mail omhandlende dårligt klima og daglig e-mail.
22. ... *Kan* sende e-mail til bruger, på baggrund af brugerindstillinger.

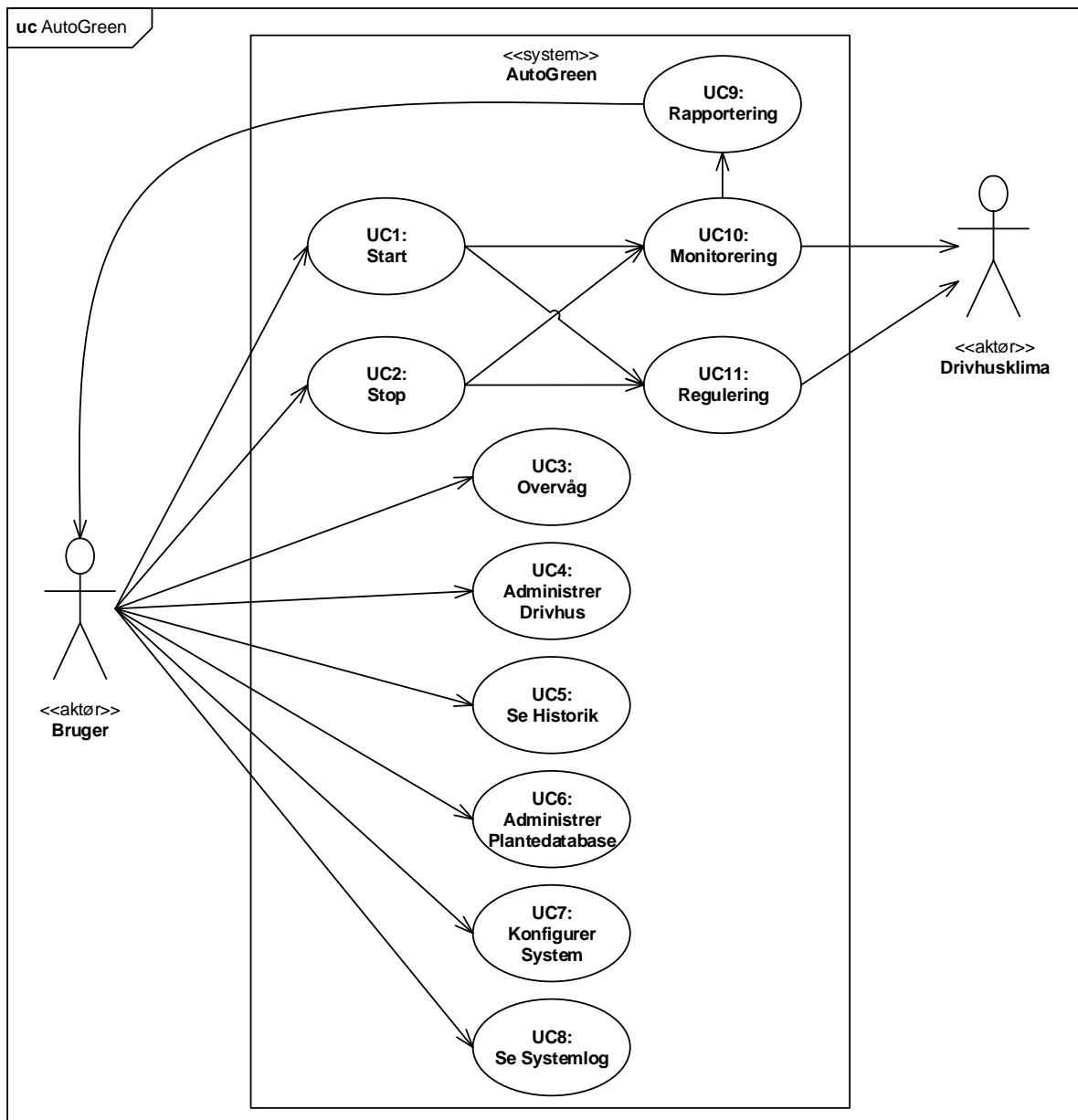
2.7 Ikke Funktionelle Krav

Systemet...

1. ... *Skal* minimum måle parametre i det fysiske drivhus med 1 minuts mellemrum +/- 5 sekunder.

2. ... *Skal* kunne justere temperaturen i det fysiske drivhus til det ønskede niveau på højest 30 minutter ved en starttemperatur der ligger højest 10 grader fra det ønskede niveau, når alle tre aktuatorer anvendes.
3. ... *Skal* kunne måle jordfugtighed i trin á 10, hvor 10 er mest fugtigt.
4. ... *Skal* kunne indeholde op til seks fugtmålere.
5. ... *Skal* kunne indeholde op til 100 planter i plantedatabasen.
6. ... *Skal* kunne indeholde måledata et år tilbage i tiden.
7. ... *Skal* kunne måle temperaturen med en præcision på +/- 1 grad celcius ved 20 grader.
8. ... *Skal* kunne indeholde op til tre e-mail adresser.
9. ... *Bør* kunne justere temperaturen til 25 grader celcius i det fysiske drivhus med en præcision på +/- 2 grad, når drivhuset er placeret i et rum ved stuetemperatur (ca. 20 grader).
10. ... *Kan* sende mail til brugeren højest 1 minut efter et for lavt jordfugtighedsniveau er målt, hvis den er indstillet til dette.

2.8 Use Case Diagram



Figur 6: Use Case Diagram for AutoGreen

2.8.1 Use Case beskrivelser - Initiering og Formål

UC1: Start

Initieres af: Bruger

Denne UC giver brugeren mulighed for at starte systemet, dvs. monitorering og regulering af drivhusklimaet. Brugeren har mulighed for kun at starte monitorering. Use Case'en kan initiere UC10 Rapportering og UC11 Monitorering.

UC2: Stop

Initieres af: Bruger

Denne UC giver brugeren mulighed for at stoppe systemet, dvs. monitorering og regulering af drivhusklimaet. Brugeren har mulighed for kun at stoppe regulering. Use Case'en kan stoppe UC10 Rapportering og UC11 Monitorering.

UC3: Overvåg

Initieres af: Bruger

Når UC10 Monitorering er startet, vises der i user interfacets hovedmenu live opdaterede måleværdier. Såfremt UC11 Regulering er startet, kan værdierne for lufttemperatur og jordfugtighed være røde, hvis de ikke passer med de ønskede værdier.

UC4: Administrerer Drivhus

Initieres af: Bruger

Denne UC giver brugeren mulighed for at informere systemet om hvilke planter der er i drivhuset. Brugeren kan tilføje op til seks planter fra plantedatabasen i det virtuelle drivhus, og brugeren kan redigere parametre for disse, hvis brugeren ønsker andre parametre end dem, der fremgår i plantedatabasen. Hver af disse planter kan forbindes med en jordfugtighedsmåler.

UC5: Se Historik

Initieres af: Bruger

Denne Use Case giver brugeren mulighed for at se grafisk historik over de fire målte parametre i drivhuset. Brugeren kan se data op til et år tilbage i tiden.

UC6: Administrerer Plantedatabase

Initieres af: Bruger

Denne UC giver brugeren mulighed for at se på planter i databasen. Brugeren kan desuden tilføje og fjerne egne planter i databasen, og brugeren kan redigere i de planter brugeren tidligere har tilføjet.

UC7: Konfigurer System

Initieres af: Bruger

Denne UC giver brugeren mulighed for at rette i systemindstillinger, herunder:

- Indstille tid og dato
- Tilføje/fjerne/rette e-mail adresse
- Aktivering/deaktivering af advarsels E-mail
- Aktivering/deaktivering af notifikations E-mail
- Aktivering/deaktivering af varmelegeme

- Aktivering/deaktivering af luftcirculation

UC8: Se System Log

Initieres af: Bruger

Denne UC giver brugeren mulighed for at se en liste over systemhændelser, herunder:

- Start og stop af system
- Manglende kontakt til sensorer
- Afsendte e-mails
- Tilføjede/fjernede/redigerede planter i drivhuset
- Tilføjede/fjernede/redigerede planter i plantedatabasen
- Konfigurationsændringer
- Fejl i registrering i data log
- Fejl på vinduesåbner
- Fejl på luftcirculation
- Fejl på varmelegeme
- Foretaget regulering

UC9: Rapportering

Initieres af: UC10 Monitorering

Denne Use Case rapporterer til brugeren ud fra de indstillinger brugeren har valgt under UC7 Konfigurer System. Dette sker ved afsendelse af e-mail til den eller de adresser, som brugeren ligeledes har tilføjet under UC7 Konfigurer System.

UC10: Monitorering

Initieres af: UC1 Start.

Denne Use Case lagrer målinger af lufttemperatur, jordfugtighed, luftfugtighed og lysintensitet i en data log fil. Lagringen sker minimum en gang i minuttet.

UC11: Regulering

Initieres af: UC1 Start.

Denne Use Case regulerer temperaturen i drivhuset, vha. vinduesåbner, varmelegeme og luftcirculation, med mindre brugeren har slået varmelegeme og/eller luftcirculation fra. Det kan ske uden luftcirculation og/eller varmelegeme, hvis brugeren har valgt dette under UC7 Konfigurer System. Det er ikke muligt at aktivere regulering uden at UC10 Monitorering er aktiveret.

2.8.2 Use Case Beskrivelser - Fully Dressed

For alle Use Cases hvor brugeren nавигerer i undermenuer af hovedmenuen, gælder det, at brugeren har mulighed for at gå et skridt tilbage ved at trykke på en "tilbage knap". Fremover ved benævningen "Systemet er operationelt" menes, at systemet er tilsluttet strømforsyning og at alt fungerer samt at systemet er tilsluttet ethernet.

Navn:	UC1: Start
Mål:	At starte systemet helt eller delvist.
Initering:	Bruger
Aktører:	Bruger (primær)
Reference:	UC10: Monitorering, UC11: Regulering
Antal samtidige forekomster:	Én
Forudsætning:	Systemet er stoppet helt, er operationelt og viser hovedmenuen.
Resultat:	UC10: Monitorering og evt. UC11: Regulering er startet, systemet viser Hovedmenuen.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Bruger trykker på "Monitorering". 2. System aktiverer UC10: Monitorering. 3. Bruger trykker på "Regulering". <ul style="list-style-type: none"> • [Ext 3.a : Bruger trykker ikke "Regulering"]. 4. Systemet aktiverer UC11: Regulering. 5. UC1 afsluttes.
Udvidelser:	[Ext 3.a : Bruger vælger kun monitorering.] <ol style="list-style-type: none"> 1. Systemet fortsætter ved pkt. 5 i hovedscenariet.

Tabel 1: UC1: Start

Navn:	UC2: Stop
Mål:	At stoppe systemet helt eller delvist.
Initering:	Bruger
Aktører:	Bruger (primær)
Reference:	UC10: Monitorering, UC11: Regulering
Antal samtidige forekomster:	Én
Forudsætning:	Både UC10: Monitorering og UC11: Regulering er startet, systemet er operationelt og viser hovedmenuen.
Resultat:	UC10: Monitorering og evt. UC11: Regulering er stoppet, systemet viser Hovedmenuen.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Bruger trykker på monitorerings knap. <ul style="list-style-type: none"> • [Ext 1.a: Bruger trykker på regulerings knap.] 2. System stopper UC10: Monitorering og UC11: Regulering. 3. UC2 afsluttes.
Udvidelser:	<p>[Ext 1.a : Bruger trykker på regulerings knap.]</p> <ol style="list-style-type: none"> 1. Systemet stopper UC11: Regulering. 2. Systemet fortsætter ved pkt. 3 i hovedscenariet.

Tabel 2: UC2: Stop

Navn:	UC3: Overvåg
Mål:	At se aktuel status i det fysiske drivhus i Hovedmenuen.
Initering:	Bruger
Aktører:	Bruger (primær)
Reference:	Ingen
Antal samtidige forekomster:	Én
Forudsætning:	UC10: Monitorering er aktiv, systemet er operationelt og hovedmenuen vises.
Resultat:	Brugeren har set et live feed af parametre for det fysiske drivhus.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Bruger aflæser måleværdier på brugerfladen. 2. UC3 afsluttes.
Udvidelser:	Ingen

Tabel 3: UC3: Overvåg

Navn:	UC4: Administrer Drivhus
Mål:	Bruger har informeret systemet om hvilke planter, der er i drivhuset.
Initering:	Bruger
Aktører:	Bruger (primær)
Reference:	Ingen
Antal samtidige forekomster:	Én
Forudsætning:	Systemet er operationelt og hovedmenuen vises.
Resultat:	Konfigureringsfilen er opdateret med informationer fra brugeren.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Bruger trykker på "Administrer drivhus" i hovedmenu. 2. System viser "Virtuel Drivhus Menu". 3. Bruger trykker på "Tilføj plante". <ul style="list-style-type: none"> • [Alt 3.a : Bruger trykker på en plante, der skal fjernes.] • [Alt 3.b : Bruger trykker på en plante, der skal redigeres.] 4. System præsenterer bruger for liste af planter i Plantedatabasen. 5. Bruger trykker på den plante, der skal tilføjes. 6. Systemet opretter planten i det virtuelle drivhus med parametrene fra plantedatabasen. 7. System viser "Planteredigeringsmenu". 8. Bruger redigerer ønskede parametre og trykker "OK". 9. Systemet gemmer brugerens valg i konfigurationsfilen og præsenterer "Virtuel Drivhus Menu". 10. Bruger trykker "Tilbage". 11. UC4 afsluttes og systemet viser Hovedmenu.
Alternativ:	<p>[Alt 3.a : Bruger trykker på en plante, der skal fjernes.]</p> <ol style="list-style-type: none"> 4. System viser "Planteredigeringsmenu". 5. Bruger trykker på "Fjern". 6. Systemet fjerner planten fra det virtuelle drivhus og markerer planten som fjernet i data loggen. 7. Systemet præsenterer "Virtuel Drivhus Menu". 8. UC4 fortsætter fra pkt. 10 i hovedscenariet. <p>[Alt 3.b : Bruger trykker på en plante, der skal redigeres.]</p> <ol style="list-style-type: none"> 4. UC4 fortsætter fra pkt. 7 i hovedscenariet.
Udvidelser:	Ingen

Tabel 4: UC4: Administrer Drivhus

Navn:	UC5: Se Historik
Mål:	At se historik fra data loggen op til et år tilbage.
Initering:	Bruger
Aktører:	Bruger (primær)
Reference:	Ingen
Antal samtidige forekomster:	Én
Forudsætning:	Systemet er operationelt og hovedmenuen vises.
Resultat:	Brugeren har set historik, systemet viser Hovedmenuen.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Bruger trykker "Se Historik" i hovedmenu. 2. System viser "Historikmenu". 3. Bruger vælger den ønskede tidshorisont (uge/måned/år). 4. Systemet viser en graf over den valgte periode. 5. Bruger kan nu vælge at deaktivere nogle måleværdier. Lys, temperatur, luftfugtighed kan deaktiveres således at de kan vises hver for sig eller samtidigt. Desuden kan brugeren vælge mellem jordfugtighed for planter i drivhuset. 6. Bruger trykker "Tilbage", UC5 afsluttes og Hovedmenuen vises.

Tabel 5: UC5: Se Historik

Navn:	UC6: Administrer Plantedatabase
Mål:	At administrere planter i plantedatabasen.
Initering:	Bruge
Aktører:	Bruge (primær)
Reference:	Ingen
Antal samtidige forekomster:	Én
Forudsætning:	Systemet er operationelt og hovedmenuen vises.
Resultat:	Bruge har tilføjet, redigeret og/eller fjernet plante i plantedatabasen. Systemet viser Hovedmenuen.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Bruge trykker "Administrer Plantedatabase" i hovedmenu. 2. System viser "Plantedatabasemenu". 3. Bruge trykker på "Tilføj Data". <ul style="list-style-type: none"> • [Alt 3.a : Bruge trykker på en plante, der skal fjernes.] • [Alt 3.b : Bruge trykker på en plante, der skal redigeres.] 4. Systemet opretter en plante med standardparametre og præsenterer "Databaseredigeringsmenu". 5. Bruge redigerer ønskede parametre og trykker "OK". 6. Systemet gemmer brugerens valg og viser "Plantedatabasemenu". 7. Bruge trykker "Tilbage". 8. UC6 afsluttes og systemet viser Hovedmenuen.
Alternativ:	<p>[Alt 3.a : Bruge trykker på en plante, der skal fjernes.]</p> <ol style="list-style-type: none"> 4. System viser "Databaseredigeringsmenu". 5. Bruge vælger "Fjern Data" og trykker "OK". 6. Systemet fjerner planten fra Plantedatabasen og viser "Plantedatabasemenu". 7. UC6 fortsætter fra pkt. 7 i hovedscenariet. <p>[Alt 3.b : Bruge trykker på en plante, der skal redigeres.]</p> <ol style="list-style-type: none"> 4. Systemet viser "Databaseredigeringsmenu". 5. UC6 fortsætter fra pkt. 5 i hovedscenariet.
Udvidelser:	Ingen

Tabel 6: UC6: Administrer Plantedatabase

Navn:	UC7: Konfigurer System
Mål:	At konfigurere indstillinger for systemet.
Initering:	Bruger
Aktører:	Bruger (primær)
Reference:	Ingen
Antal samtidige forekomster:	Én
Forudsætning:	Systemet er operationelt, regulering er aktiveret og hovedmenuen er vist.
Resultat:	Systemet er konfigureret efter brugerens ønske. Hovedmenuen vises.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Bruger trykker "Konfigurer System". 2. System viser "Konfigurationsmenu". 3. Bruger vælger "E-mail Adresser". <ul style="list-style-type: none"> • [Alt 3.a : Bruger vælger "Notifikationer"]. • [Alt 3.b : Bruger vælger "Indstil dato/tid"]. • [Alt 3.c : Bruger vælger "Hardware indstillinger"]. 4. Systemet viser "E-mail menu". 5. Brugeren indtaster op til tre ønskede E-mail adresser og trykker "OK". 6. Systemet gemmer E-mail adresserne i konfigurationsfilen. Systemet viser "Konfigurationsmenu". 7. Brugeren trykker "tilbage". 8. UC7 afsluttes og Hovedmenuen vises.
Alternativ:	<p>[Alt 3.a : Bruger vælger "Notifikationer"].</p> <ol style="list-style-type: none"> 4. System viser "Notifikationsmenu". 5. Brugeren indtaster ønskede indstillinger for notifikationer. 6. Brugeren trykker "OK". 7. Systemet gemmer indstillingerne i konfigurationsfilen og viser "Konfigurationsmenu". 8. UC7 fortsætter fra punkt 7 i hovedscenariet. <p>[Alt 3.b : Bruger vælger "Indstil dato/tid"].</p> <ol style="list-style-type: none"> 4. Systemet viser "Tid- og datomenu". 5. Bruger indtaster dato og tid. 6. Brugeren trykker "OK". 7. System gemmer de indtastede data i konfigurationsfilen og viser "Konfigurationsmenu". 8. UC7 fortsætter fra punkt 7. <p>[Alt 3.c : Bruger vælger "Hardware indstillinger"].</p> <ol style="list-style-type: none"> 4. System viser "Hardware Indstillingsmenu". 5. Brugeren vælger blæser on/off og/eller varmelegeme on/off. 6. Brugeren trykker "OK". 7. System gemmer de indtastede indstillinger i konfigurationsfilen og viser "Konfigurationsmenu". 8. UC7 fortsætter fra punkt 7 i hovedscenariet.

Tabel 7: UC7: Konfigurer System

Navn:	UC8: Se Systemlog
Mål:	Brugeren aflæser data i system log.
Initiering:	Bruger
Aktører:	Bruger
Reference:	Ingen
Antal samtidige forekomster:	Én
Forudsætning:	Systemet er operationelt og hovedmenu vises.
Resultat:	Brugeren har set system log og Hovedmenuen er vist.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Bruger vælger "Se Systemlog". 2. Systemet viser en "Systemlogmenu", der indeholder en liste over hændelser i systemet. 3. Bruger vælger "Tilbage". 4. UC8 afsluttes og hovedmenuen vises.
Udvidelser:	Ingen

Tabel 8: UC8: Se Systemlog

Navn:	UC9: Rapportering
Mål:	Bruger modtager notifikations- og advarsels E-mails.
Initiering:	UC10: Monitorering
Aktører:	Bruger
Reference:	UC10: Monitorering
Antal samtidige forekomster:	Én
Forudsætning:	UC10 er aktiv, systemet er operationelt og notifikations- og advarselemail er slået til.
Resultat:	Systemet har sendt notifikations- og advarsels E-mail.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Systemet sender daglig notifikations E-mail klokken 12. 2. Systemet sender advarsels E-mail, hvis en parameter i det fysiske drivhus er under den ønskede værdi.
Udvidelser:	Ingen

Tabel 9: UC9: Rapportering

Navn:	UC10: Monitorering
Mål:	At opdatere live parametre i Hovedmenuen.
Initering:	UC1: Start
Aktører:	Drivhusklima
Reference:	UC1: Start, UC2: Stop, UC9: Rapportering
Antal samtidige forekomster:	Én
Forudsætning:	UC1 er gennemført og systemet er operationelt.
Resultat:	Hovedmenuen er opdateret med nyeste data fra data loggen.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Systemet indlæser konfigurationsfilen. 2. Systemet aflæser måleværdier fra sensorer og gemmer dem i data loggen. 3. Systemet opdaterer live-status i hovedmenuen med de målte værdier. 4. Systemet sammenligner aflæste værdier fra sensorerne med ønskede værdier fra det virtuelle drivhus. 5. Målte værdier ligger inden for tolerancerne i forhold til ønskede værdier. <ul style="list-style-type: none"> • [Ext 4.a : Værdierne ligger ikke inden for tolerancerne.] 6. Systemet farver alle datafelter for jordfugtighed grønne. 7. Systemet venter et minut og fortsætter fra pkt. 1 i hovedscenariet.
Udvidelser:	<p>[Ext 4.a : Værdierne ligger ikke inden for tolerancerne.]</p> <ol style="list-style-type: none"> 1. Systemet aktiverer UC9: Rapportering. 2. Systemet markerer datafelter, der ligger udenfor tolerancerne røde. 3. Systemet fortsætter fra pkt. 7 i hovedscenariet.

Tabel 10: UC10: Monitorering

Navn:	UC11: Regulering
Mål:	At regulere temperaturen i det fysiske drivhus til ønsket værdi, samt at jordfugtigheden for hver plante stemmer overens med den angivne jordfugtighed i det virtuelle drivhus.
Initiering:	UC1: Start
Aktører:	Drivhusklima
Reference:	UC1: Start
Antal samtidige forekomster:	Én
Forudsætning:	Systemet er operationelt og regulering er aktiveret.
Resultat:	Aktuatorer for vindue, blæsere, varmelegeme og vanding er evt. aktiveret.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Systemet indlæser konfigurationsfilen. 2. Systemet sammenligner nyeste værdier for jordfugtighed fra data loggen med ønskede værdier fra det virtuelle drivhus. 3. Jordfugtværdierne ligger inden for tolerancerne. <ul style="list-style-type: none"> • [Ext 3.a : En eller flere jordfugtværdier ligger under tolerancen.] 4. Systemet sammenligner nyeste værdi for temperatur fra data loggen med angiven værdi i det virtuelle drivhus. 5. Værdien ligger inden for tolerancen. <ul style="list-style-type: none"> • [Ext 5.a : Værdien for temperatur ligger over tolerancen.] • [Ext 5.b : Værdien for temperatur ligger under tolerancen.] 6. Systemet venter 1 minut og fortsætter fra pkt. 1 i hovedscenariet.
Udvidelser:	<p>[Ext 3.a : En eller flere jordfugtværdier ligger under tolerancen.]</p> <ol style="list-style-type: none"> 1. Systemet aktiverer aktuator for vanding for den eller de planter der er under tolerancen. 2. Systemet fortsætter fra pkt. 4 i hovedscenariet. <p>[Ext 5.a : Værdien for temperatur ligger over tolerancen.]</p> <ol style="list-style-type: none"> 1. Systemet regulerer temperaturen nedad jf. konfigurationsfilen. 2. Systemet fortsætter fra pkt. 6 i hovedscenariet. <p>[Ext 5.b : Værdien for temperatur ligger under tolerancen.]</p> <ol style="list-style-type: none"> 1. Systemet regulerer temperaturen opad jf. konfigurationsfilen. 2. Systemet fortsætter fra pkt. 6 i hovedscenariet.

Tabel 11: UC11: Regulering

3 Accepttest

3.1 Version

Dato	Version	Initialer	Ændring
26. februar	1	KS	Første udkast.
6. marts	2	MHG	Rettelser efter review.
13. marts	3	MHG	Mindre rettelser efter fælles gennemlæsning.

3.2 Funktionelle Krav

Use case under test		UC1: Start og UC2: Stop		
Scenarie		Hovedscenarie		
Forudsætning		Systemet er stoppet helt, er operationelt og viser hovedmenuen.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
1.1	Bruger trykker på "Monitorering".	Visuel test: Bruger observerer at "Monitorering"skifter farve fra rød til grøn.		
1.2	Bruger trykker på "Monitorering".	Visuel test: Bruger observerer at "Monitorering"skifter farve fra grøn til rød.		
1.3	Bruger trykker på "Regulering".	Visuel test: Bruger observerer at "Monitorering" og "Regulering" skifter farve fra rød til grøn.		
1.4	Bruger trykker på "Regulering".	Visuel test: Bruger observerer at "Regulering"skifter farve fra grøn til rød.		
1.5	Bruger trykker på "Monitorering".	Visuel test: Bruger observerer at "Monitorering"skifter farve fra grøn til rød.		
1.6	Bruger trykker på "Regulering".	Visuel test: Bruger observerer at "Monitorering" og "Regulering" skifter farve fra rød til grøn.		

1.7	Bruger trykker på "Monitorering".	Visuel test: Bruger observerer at "Monitorering" og "Regulering" skifter farve fra grøn til rød.		
-----	-----------------------------------	--	--	--

Tabel 12: Accepttest for UC1: Start og UC2: Stop

Use case under test		UC3: Overvåg		
Scenarie		Hovedscenarie		
Forudsætning		UC 10 er aktiv, systemet er operationelt og hovedmenuen vises.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
3.1	Bruger ser på brugergrænsefladen.	Visuel test: Der observeres tilstedeværelse af oplysninger om temperatur.		
3.2	Bruger ser på brugergrænsefladen.	Visuel test: Der observeres tilstedeværelse af oplysninger om luftfugtighed.		
3.3	Bruger ser på brugergrænsefladen.	Visuel test: Der observeres tilstedeværelse af oplysninger om lysintensitet.		
3.4	Bruger ser på brugergrænsefladen.	Visuel test: Der observeres tilstedeværelse af oplysninger om jordfugtighed for jordfugtmåler 1-6.		

Tabel 13: Accepttest for UC3: Overvåg

Use case under test		UC4: Administrer drivhus		
Scenarie		Hovedscenarie		
Forudsætning		Systemet er operationelt og hovedmenuen vises.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
4.1	Bruger trykker "Administrer Drivhus".	Visuel test: Bruger observerer en undermenuen "Virtuelt Drivhus Menu" på brugerfladen.		
4.2	Bruger trykker "Tilføj plante".	Visuel test: Bruger observerer en liste over planterne i Plantedatabasen på brugerfladen.		
4.3	Bruger vælger den øverste plante på listen.	Visuel test: Systemet viser undermenuen "Planteredigerings-menu".		
4.4	Bruger indtaster parametre for planten, temperatur: 25 grader, fugtgthed: 10, og trykker "OK".	Visuel test: Undermenuen "Virtuelt Drivhus Menu" vises. Den redigerede plante vises.		
4.5	Bruger trykker på den plante, der blev tilføjet under pkt. 4.4.	Visuel test: Systemet viser undermenuen "Planteredigerings-menu".		
4.6	Bruger trykker "Fjern" og trykker "OK".	Visuel test: "Virtuelt Drivhus Menu" vises. Den fjernede plante er ikke længere i det virtuelle drivhus.		
4.9	Bruger trykker "Tilbage".	Visuel test: Hovedmenuen vises.		

Tabel 14: Accepttest for UC4: Administrer drivhus

Use case under test		UC5: Se historik		
Scenarie		Hovedscenarie		
Forudsætning		Systemet er operationelt og hovedmenuen vises.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
5.1	Bruger trykker "Se Historik".	Visuel test: Systemet viser "Historikmenu".		
5.2	Bruger trykker "Uge".	Visuel test: Systemet viser en graf med historik for en uge.		
5.3	Bruger trykker "Måned".	Visuel test: Systemet viser en graf med historik for en måned.		
5.4	Bruger trykker "År".	Visuel test: Systemet viser en graf med historik for et år.		
5.5	Bruger trykker "Temperatur".	Visuel test: Historik for temperatur vises ikke på grafen.		
5.6	Bruger trykker "Luftfugtighed".	Visuel test: Historik for luftfugtighed vises ikke på grafen.		
5.7	Bruger trykker "Lys".	Visuel test: Historik for lys vises ikke på grafen.		
5.8	Bruger trykker "Temperatur".	Visuel test: Historik for temperatur vises på grafen.		
5.9	Bruger trykker "Luftfugtighed".	Visuel test: Historik for luftfugtighed vises på grafen.		
5.10	Bruger trykker "Lys".	Visuel test: Historik for lys vises på grafen.		
5.11	Bruger trykker "Jordfugtighed" for plante nr 1.	Visuel test: Jordfugtigheden for plante nr. 1 vises på grafen.		
5.12	Pkt. 5.11 gentages for planterne 2-6.	Visuel test: Jordfugtigheden for planterne vises på grafen.		

5.13	Bruger trykker "Jordfugtighed" for planten nr 1.	Visuel test: Jordfugtigheden for planten nr. 1 vises ikke på grafen.		
5.14	Pkt. 5.12 gentages for planten 2-6.	Visuel test: Jordfugtigheden for planterne vises ikke på grafen.		
5.15	Bruger trykker på "Tilbage"	Visuel test: Hovedmenuen vises.		

Tabel 15: Accepttest for UC5: Se historik

Use case under test		UC6: Administrer Plantedatabase		
Scenarie		Hovedscenarie		
Forudsætning		Systemet er operationelt og hovedmenuen vises.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
6.1	Bruger trykker "Administrer Plantedatabase".	Visuel test: Systemet viser "Plantedatabasemenu".		
6.2	Bruger trykker på "Tilføj Data".	Visuel test: Systemet viser "Databaseredigeringsmenu".		
6.3	Bruger vælger ønskede parametre for planten, temperatur: 22 grader, fugtighed: 8, og trykker "OK".	Visuel test: Systemet viser "Plantedatabasemenu". Planten er tilføjet.		
6.4	Bruger trykker på den planten, der blev tilføjet under pkt. 6.3.	Visuel test: Systemet viser "Databaseredigeringsmenu".		
6.5	Bruger trykker "Fjern".	Visuel test: Systemet viser en dialogboks.		
6.6	Bruger trykker "OK".	Visuel test: Systemet viser "Plantedatabasemenu". Planten er fjernet.		
6.7	Bruger trykker "Tilbage".	Visuel test: Hovedmenuen vises.		

Tabel 16: Accepttest for UC6: Administrer plantedatabase

Use case under test		UC7: Konfigurer system		
Scenarie		Hovedscenarie		
Forudsætning		Systemet er operationelt og hovedmenuen vises. Blæsere og varmelegeme er slået fra.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
7.1	Bruger trykker "Konfigurer System".	Visuel test: System viser "Konfigurationsmenu".		
7.2	Bruger trykker "E-mail Adresser".	Visuel test: System viser "E-mail Menu".		
7.3	Bruger indtaster tre E-mail adresser og trykker "OK".	Visuel test: System viser "Konfigurationsmenu".		
7.4	Bruger trykker "E-mail Adresser".	Visuel test: System viser "E-mail Menu". De tre E-mail adresser fra pkt. 7.3 er synlige.		
7.5	Bruger trykker "Tilbage".	Visuel test: Systemet viser "Konfigurationsmenu".		
7.6	Bruger trykker "Notifikationer"	Visuel test: "Notifikationsmenu" vises.		
7.7	Bruger trykker på "Notifikations E-mail".	Visuel test: "Notifikations E-mail" skifter farve fra rød til grøn.		
7.8	Bruger trykker på "Notifikations E-mail".	Visuel test: "Notifikations E-mail" skifter farve fra grøn til rød.		
7.9	Bruger trykker på "Advarsels E-mail".	Visuel test: "Advarsels E-mail" skifter farve fra rød til grøn.		
7.10	Bruger trykker på "Advarsels E-mail".	Visuel test: "Advarsels E-mail" skifter farve fra grøn til rød.		
7.11	Bruger trykker "Tilbage".	Visuel test: Systemet viser "Konfigurationsmenu".		
7.12	Bruger trykker "Indstil dato/tid".	Visuel test: "Tid- og Datomenu" vises.		

7.13	Bruger indtaster 1. juli klokken 14:15 og trykker "OK".	Visuel test: Systemet går tilbage til "Konfigurationsmenu". Ny dato og tid (1. juli klokken 14:15) vises på brugerfladen.		
7.14	Bruger trykker "Hardware Indstillinger".	Visuel test: Systemet viser "Hardware indstillingsmenu".		
7.15	Bruger trykker på "Blæsere"	Visuel test: "Blæsere" skifter farve fra rød til grøn.		
7.16	Bruger trykker på "Blæsere"	Visuel test: "Blæsere" skifter farve fra grøn til rød.		
7.17	Bruger trykker på "Varmelegeme".	Visuel test: "Varmelegeme" skifter farve fra rød til grøn.		
7.18	Bruger trykker på "Varmelegeme".	Visuel test: "Varmelegeme" skifter farve fra grøn til rød.		
7.12	Bruger trykker på "Tilbage".	Visuel test: "Konfigurationsmenu" vises.		
7.13	Bruger trykker på "Tilbage".	Visuel test: "Hovedmenu" vises.		

Tabel 17: Accepttest for UC7: Konfigurer system

Use case under test		UC8: Se systemlog		
Scenarie		Hovedscenarie		
Forudsætning		Systemet er operationelt og hovedmenuen vises.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
8.1	Bruger trykker "Systemlog".	Visuel test: Systemet viser "Systemlogmenu".		
8.2	Bruger trykker "Tilbage".	Visuel test: Systemet viser Hovedmenuen.		

Tabel 18: Accepttest for UC8: Se systemlog

Use case under test		UC9: Rapportering		
Scenarie		Hovedscenarie		
Forudsætning		UC10 Monitorering er aktiv, systemet er operationelt og E-mail-opsætning er udført af brugeren. Desuden skal brugeren have angivet ønske om at modtage notifikationer. Jordfugtighedsensor 1 er konfigureret til en plante, som har niveau 10 som ønsket jordfugtighedsparameter.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
9.1	Bruger tjekker sin email klokken 12:15.	Visuel test: Bruger har modtaget E-mail med daglig status fra systemet.		
9.2	Bruger tager jordfugtighedsensor 1 op af jorden.	Visuel test: Bruger modtager advarsels E-mail.		

Tabel 19: Accepttest for UC9: Rapportering

Use case under test		UC10: Monitorering		
Scenarie		Hovedscenarie		
Forudsætning		UC1 Start er gennemført og systemet er operationelt.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
10.1	Bruger noterer aktuel værdi for temperatur i Hovedmenuen. Brugeren sprayer kuldespray på temperatursensoren og venter mindst 1 minut.	Visuel test: Værdien i feltet "Temperatur" i Hovedmenuen falder.		
10.2	Bruger deaktiverer monitorering og tilgår systemloggen.	Visuel test: Systemloggen er blevet opdateret med nye data og korrekt tidsstempeling.		

Tabel 20: Accepttest for UC10: Monitorering

Use case under test		UC11: Regulering		
Scenarie		Hovedscenarie		
Forudsætning		Både UC10 Monitorering og UC11 Regulering er startet. Jordfugtighedssensor 1 er konfigureret til en plante, som har niveau 10 som ønsket jordfugtighedsparameter. Varmelegeme og blæsere er aktiveret.		
Step	Handling	Forventet Resultat	Resultat	Godkendt / Kommentar
11.1	Bruger tager jordfugtighedssensor 1 op af jorden; spændingen på aktuator for vanding ved jordfugtighedssensor 1 måles med voltmeter.	Aktuator for vanding ved plante 1 går fra false til true.		
11.2	Bruger sprayer kuldespray på temperatursensoren.	Visuel test: Varmelegemet aktiveres.		
11.3	Det fysiske drivhus varmes op vha. en varmeblæser.	Visuel test: Blæsere aktiveres og vinduet åbnes.		

Tabel 21: Accepttest for UC11: Regulering

3.3 Ikke-funktionelle krav

Krav	Test	Forventet Resultat	Resultat	Godkendt/ kommentar
1.	Start drivhuset med monitoring, noter hvornår værdier bliver tilføjet til systemloggen. Varighed 10 min.	Data loggen genererer 10 datapunkter med 1 min. mellemrum.		
2.	Det fysiske drivhus placeres i et rum ved 20 +/- 1 grader celcius, og opvarmes vha. en varmeblæser til minimum 30 grader celcius. I det virtuelle drivhus sættes en ønsket gennemsnitstemperatur på 25 grader celcius og ventilator er aktiveret.	Inden der er gået 30 min. aflæses temperaturen i drivhuset til 25 +/- 1 grader celcius.		

3.	En potte med tør muld indsættes i det fysiske drivhus. En fugtighedsensor places i mulden og vand hældes langsomt i.	Systemloggen skriver 11 dataværdier med stigende fugtighed og den 11. data værdi er ækvivalent med den 10. værdi.		
4.	6 fugtighedsmålere tilsluttes systemet, hvorefter systemet startes.	Systemloggen indeholder måleværdier for 6 forskellige sensorer efter 1 min.		
5.	100 planter indsættes i plantedatabasen.	Databasen kontrolleres for alle 100 planters eksistens.		
6.	Intervallet for datalogging sættes ned for at simulere et års data.	Historik kan ses med et års data.		
7.	Det fysiske drivhus placeres i et rum ved 20 ± 1 grader celcius. Systemet sættes til at regulere temperaturen i det fysiske drivhus til 25 grader celcius. Et eksternt termometer med en usikkerhed på højest 0.1 grader celcius placeres ved siden af temperatursensoren.	Det eksterne termometer mäter 25 ± 2 grader celcius inden for 30 min.		
8.	Der indtastets tre gyldige E-mail adresser via brugerfladens "E-mailmenu". Daglig E-mail notifikation aktiveres. Testpersonen kontrollerer de tre indtastede E-mailkontos indbakker næste gang klokken har passeret 12.00	Testpersonen har modtaget en E-mail fra systemet på hver af de tre indtastede E-mailadresser.		
9.	Det fysiske drivhus placeres i et rum ved 20 ± 1 grader celcius. I det virtuelle drivhus sættes en ønsket gennemsnitstemperatur på 25 grader celcius.	Inden der er gået 30 min. aflæses temperaturen i drivhuset vha. Hovedmenuen til 25 ± 1 grad celcius.		

10.	En potte med tør muld indsættes i det fysiske drivhus. En fugtighedsensor placeres i mulden og rapportering og Advarsels E-mail er aktiveret. En sensor med ønsket værdi for fugtighed på 10 placeres i mulden.	Før 10 min er forløbet, har brugeren modtaget en Advarsels E-mail.		
-----	---	--	--	--

Tabel 22: Ikke funktionelle krav

4 Systemarkitektur

4.1 Version

Dato	Version	Initialer	Ændring
11. marts	1	KS	Første udkast.
18. marts	2	MHG	Inden review.
23. marts	3	MHG	Rettelser efter review.
16. april	4	MHG	Rettet til så I ² C sensorer forsynes fra MasterPSoC.
1. maj	5	HBJ	Rettet signalbeskrivelse for signaltypen Analog. Passer nu til jordfugt blokken.

4.2 Indledning

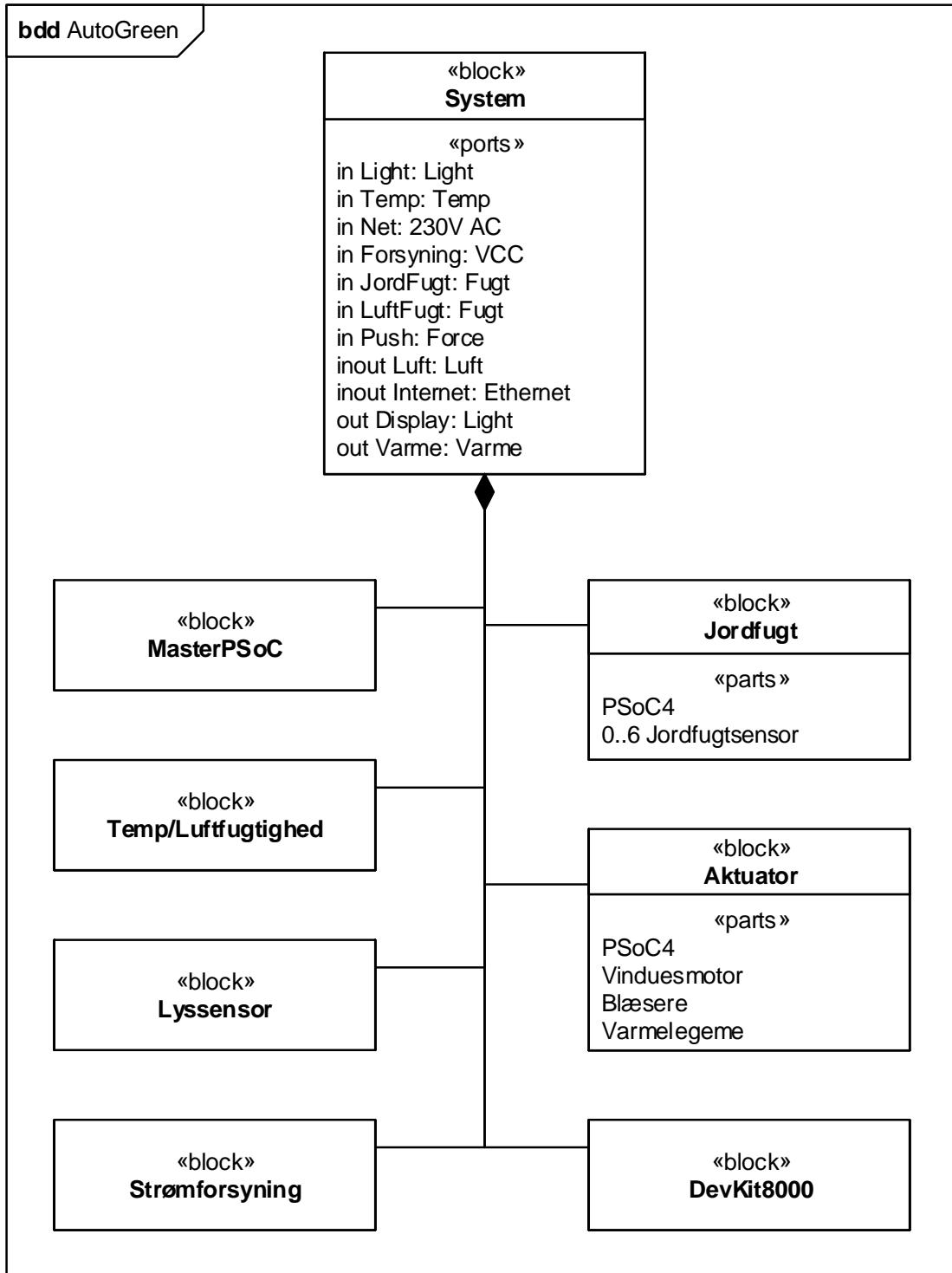
I dette kapitel vil systemarkitekturen for AutoGreen være opdelt i to underdele, hhv. for hardware og for software. Formålet med kapitlet er at gøre systemets grænseflader, både interne og eksterne, klare ift. signaltyper, niveauer og softwaregrænseflader.

4.3 Hardwarearkitektur

Dette afsnit beskriver arkitektur for hardware i AutoGreen.

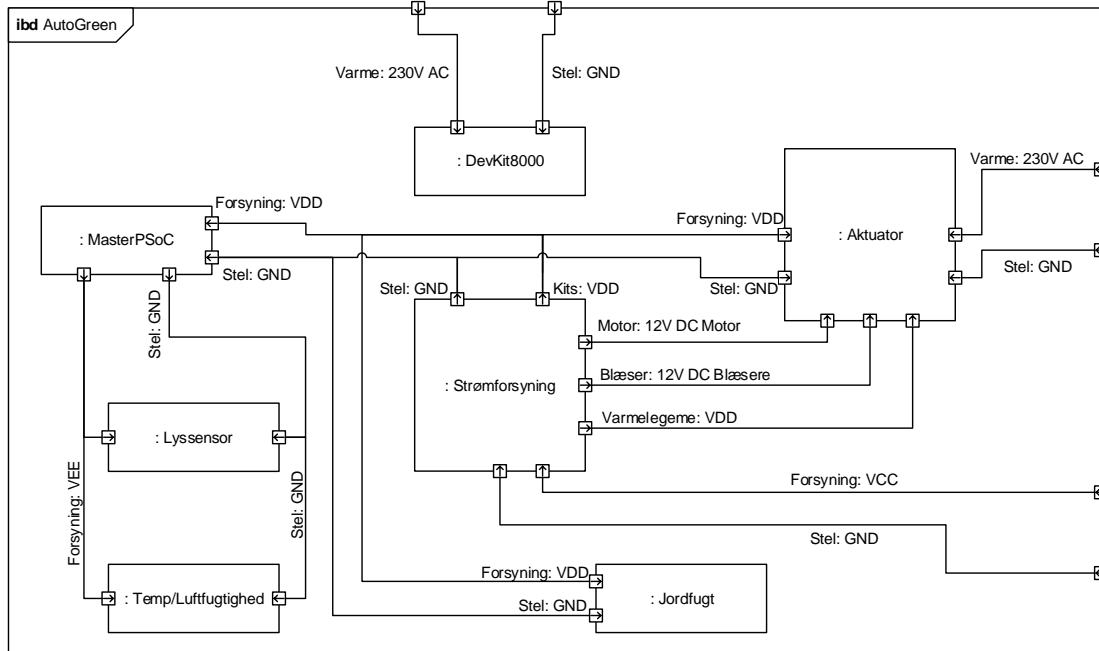
Forsyning til alle blokke er beskrevet på IBD for system, Figur 8. Forsyninger er ikke tegnet ind på øvrige diagrammer for overskuelighedens skyld. Det gælder desuden at alle blokke har fælles reference (GND).

4.3.1 BDD for System

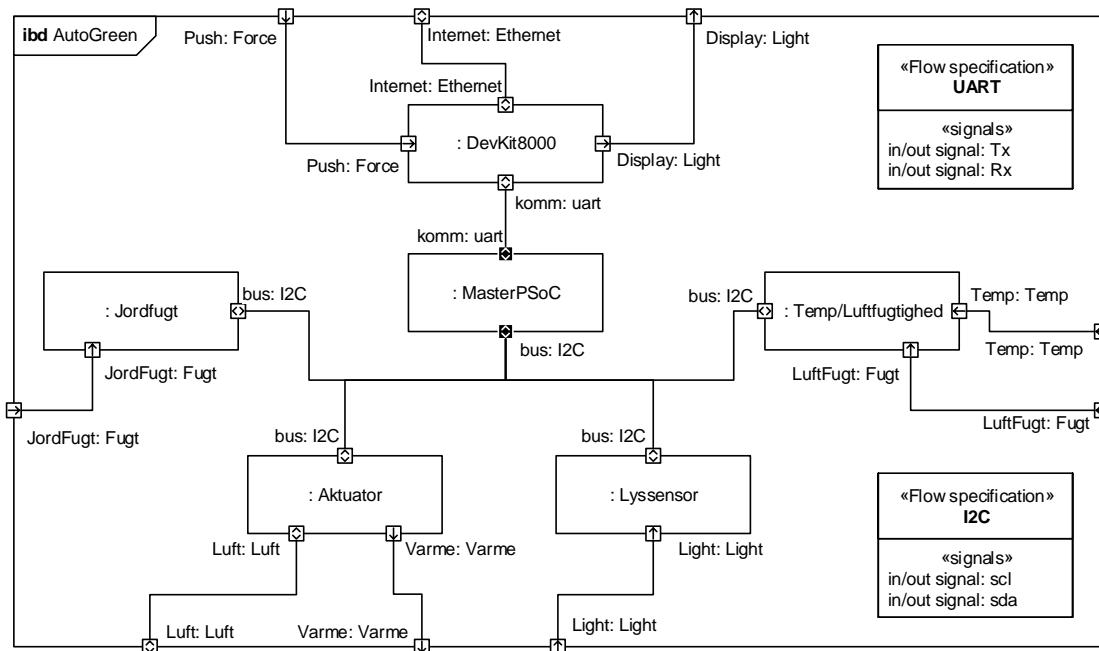


Figur 7: BDD for System.

4.3.2 IBD'er for System



Figur 8: IBD for forsyninger i systemet.



Figur 9: IBD for signaler i systemet.

Strømforsyning

Forsyner øvrig hardware i systemet, undtagen varmelegemet, Devkit8000 samt I²C sensorer. Blokken forsynes fra en laboratorieforsyning.

DevKit8000

Systemets brugerflade, er samtidigt controller for systemet.

MasterPSoC

PSoC4 Pioneer Kit, der har til opgave at kommunikere via UART med DevKit8000 og via I²C med slaver.

Temp/Luftfugtighed

Denne blok indeholder en sensor med I²C interface og mäter temperatur og luftfugtighed i det fysiske drivhus.

Lyssensor

Består af en sensor med I²C interface og mäter lysintensitet i det fysiske drivhus.

Jordfugt

Denne blok indeholder op til seks analoge jordfugtsensorer, som vha. et PSoC4 Pioneer Kit er koblet på systemets I²C bus.

Aktuator

Denne blok indeholder et PSoC4 Pioneer Kit, der fungerer som I²C slave og styrer systemets aktuatorer.

4.3.3 IBD for Aktuator

PSoC4

PSoC blokken består af et PSoC4 Pioneer Kit, der agerer slave på I²C bussen.

Vinduesmotor

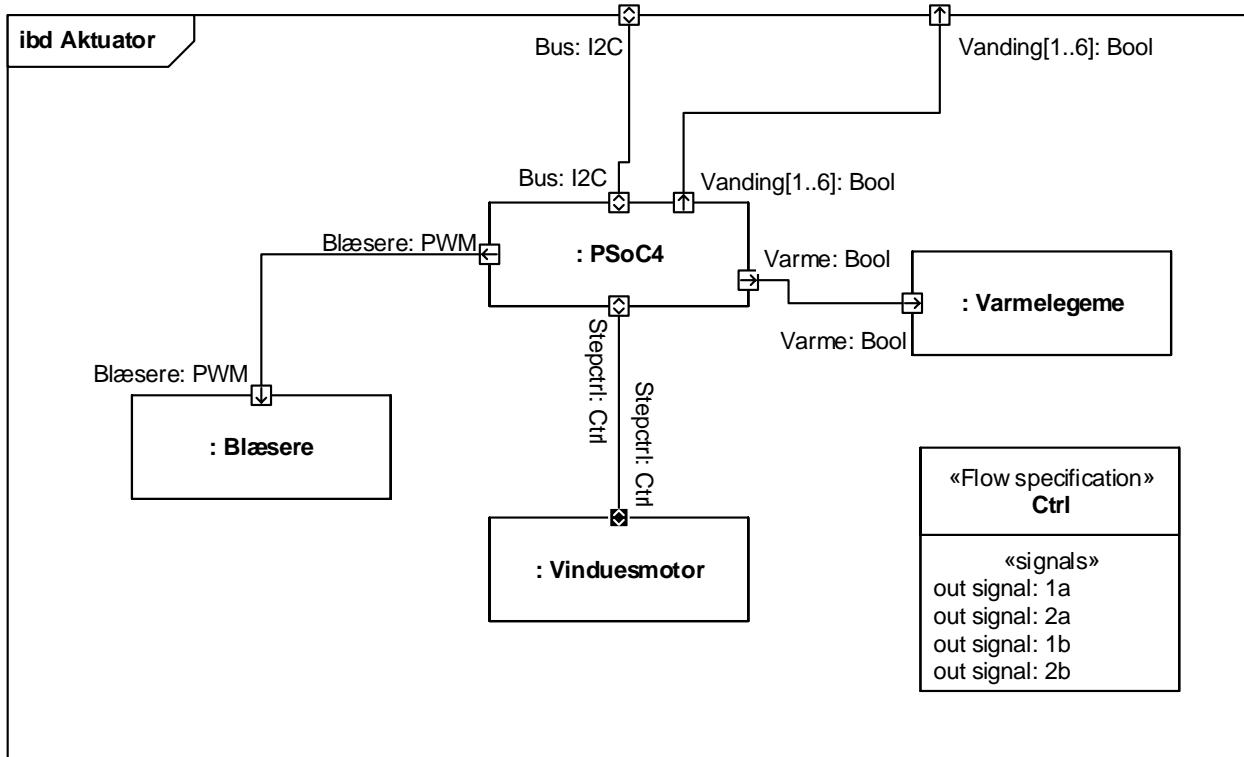
Denne blok består af en steppermotor, der styrer vinduet i det fysiske drivhus.

Varmelegeme

Varmelegeme med formål at hæve temperaturen i det fysiske drivhus. Varmelegemet styres af PSoC4 blokken, og det forsynes direkte fra elnettet (230V AC).

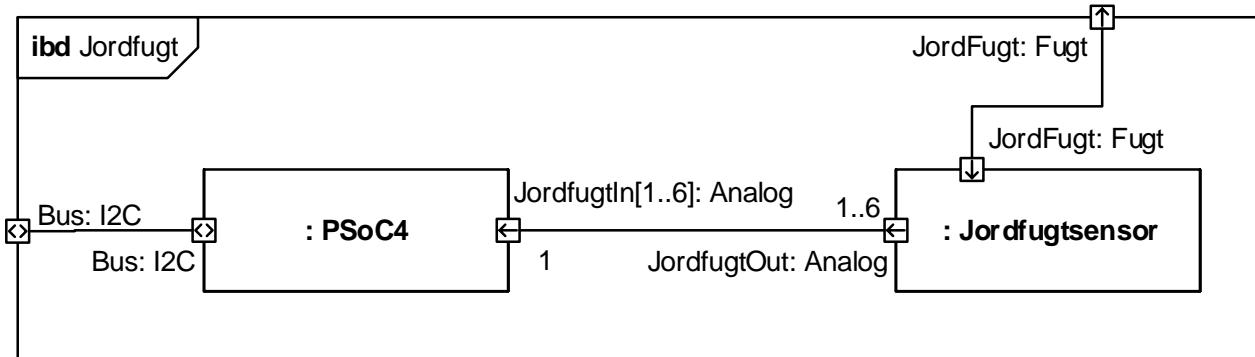
Blæsere

Denne blok består af nogle blæsere, som kan ventilere luften i det fysiske drivhus. Blæserne styres af PSoC4, og de forsynes fra Strømforsyning.



Figur 10: IBD for Aktuator

4.3.4 IBD for Jordfugt



Figur 11: IBD for Jordfugt

PSoC4

PSoC4 Pioneer Kit, der agerer slave på I²C -bussen.

Jordfugtsensor

Denne blok indeholder en analog sensor, der måler jordfugt ved en plante i det fysiske drivhus. Der kan kobles op til seks af disse til PSoC4.

4.3.5 Signalbeskrivelser

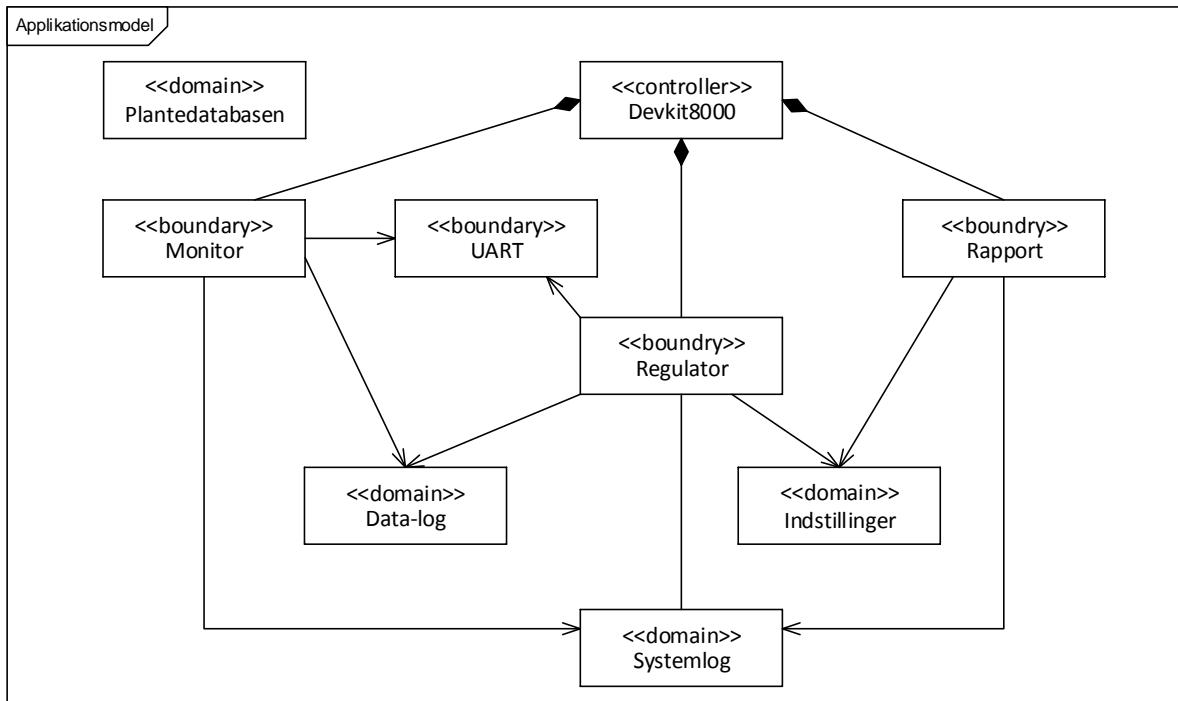
Signaltyp	Funktion	Tolerancer	Kommentar
VCC	Forsyning til strømforsyning	12V ± 0,25V 3A max.	Lab.forsyning
VDD	Forsyning til alle PSoC4 Pioneer Kits.	5V DC ± 0.15V, 0.5A max	-
VEE	Forsyning til sensorer	3.3V DC ± 0.1V, 0.1A max	-
12V DC Blæsere	Forsyning til blæsere.	12V DC ± 0,25V, 140mA max.	-
12V DC Motor	Forsyning til vinduesmotor.	12V ± 0,25V, 500mA max.	-
230V AC	Forsyning til varmelegeme og DevKit8000.	230V AC ± 10%, 50 Hz, 0.3A max	-
Analog	Analogt målesignal fra jordfugtmåler.	0-3.3V ± 0.1V	-
Bool	Digitalt signal til styring af vanding og varmelegeme.	0-3.3V	1=True: 2.8-3.3V 0=False: 0-0.4V
Ctrl	Styring af stepper motor	0-3.3V	1=True: 2.8-3.3V 0=False: 0-0.4V Består af fire signaler: 1a, 2a, 1b, 2b
GND	Stel	0V	Reference
I2C	Kommunikation mellem I ² C enheder.	0-3.3V	1=True: 2.8-3.3V 0=False: 0-0.4V Består af to signaler: sda og scl
UART	Kommunikation mellem DevKit8000 og Master	0-5V	1=True: 4.5-5V 0=False: 0-0.4V Består af 2 signaler: Tx og Rx
PWM	Styring af blæsere vha. pulsbreddemodulation.	0-3.3V 1 kHz	Duty cycle styres fra 0-100% i trin fra 0-255. Hvor 0 svarer til 0% og 255 svarer til 100%

Tabel 23: Beskrivelse af signaler.

4.4 Softwarearkitektur

4.4.1 Applikationsmodel

Applikationsmodellen er valgt ud fra udviklernes synspunkt og bruges for at give overblik over hvilke klasser som skal laves, og hvilket ansvar de hver især har. Nedenstående UML skal ses som det overordnede system og menuklasserne er udeladt for at skabe overblik.



Figur 12: Application model for AutoGreen

4.4.2 Controller-Klasser

DevKit8000

DevKit8000 klassen skal initiere systemet og har derfter ansvaret for styring af processerne Regulering og Monitoring. DevKit8000 klassen indeholder alle menuer beskrevet i menuoversigt. Brugeren kan interagere med klassen igennem menuerne. Controller-klassen har igennem menuerne set i menuoversigten tilgang til de andre klasser i systemet.

4.4.3 Boundary-Klasser

Monitor

Monitorklassens primære opgave er at opsamle sensordata fra UART klassen og skrive dem til dataloggen. Derudover skal Monitor skrive til System-log, hvis UART klassen rapporterer fejl ved dataoverførelse.

Regulator

Reguleringsklassen har ansvaret for at planterværdierne bliver overholdt. Den opnår dette ved at læse fra data-loggen og hvis uregelmæssigheder findes blandt disse data, vil klassen tænde de fornødende akutuatorer gennem UART klassen. Der ud over skal Regulator skrive til System-log, hvis UART klassen rapporterer fejl ved data overførelse.

UART

UARTklassen er grænsefladen mellem Devkittet og de sensorer/akutuatorer, der måtte eksistere i AutoGreen systemet.

Rapport

Rapportering indlæser E-mailkonfigurationer fra indstillinger, som bestemmer hvilke funktionaliteter, der skal benyttes. Rapporting skal sende E-mail til brugeren dagligt, når der er kritisk klima i drivhuset, eller både dagligt og ved kritisk klima.

4.4.4 Domain-Klasser

Data-log

Data-loggen styrer en datastruktur. Det er dens opgave at modtage og indsætte målte data from drivhusklima i datastrukturen, samt hente informationer ud fra strukturen.

System-log

System-loggen har til ansvar at styre en datastruktur med henblik på at gemme de vigtigste systemhændelser og skal kunne tilgås af brugeren senere.

Indstillinger

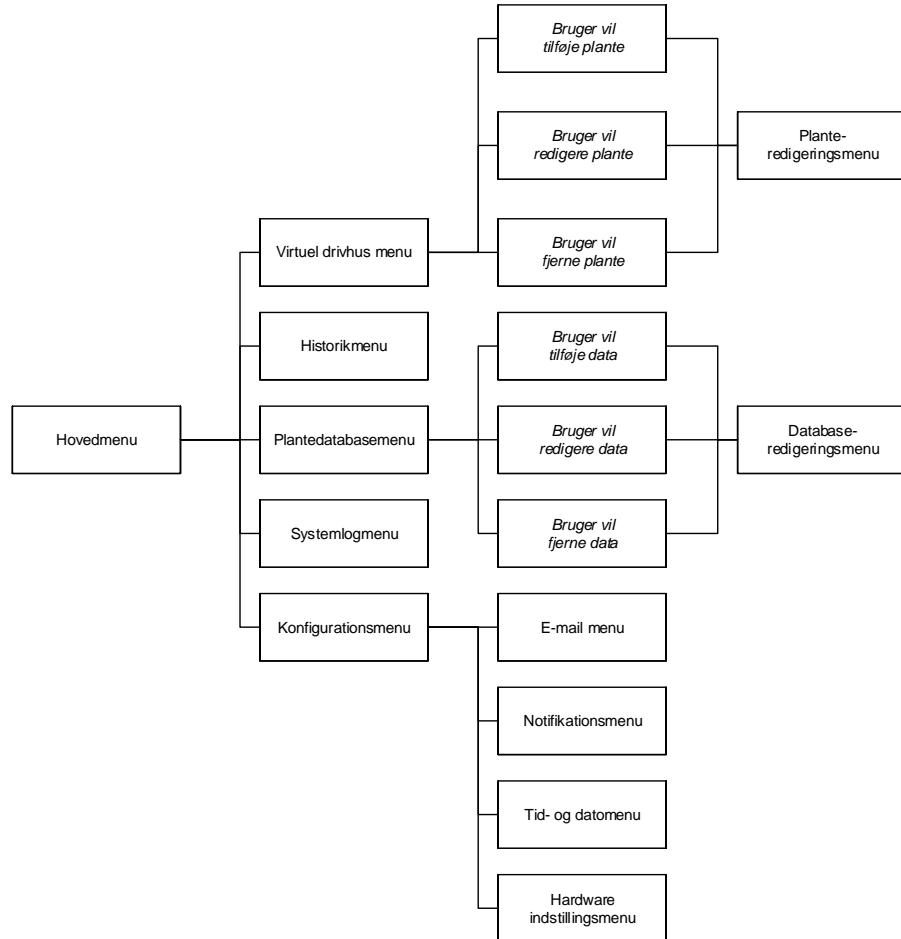
Indstillinger gemmer konfigurationer og indlæser dem i konfigurationsfilen, når regulering eller rapportering startes af brugeren.

Plantedatabasen

Plantedatabasen gemmer parametre for brugerdefinerede planter samt prækonfigurede planter og tilgås via en klasse menu.

4.4.5 Menuoversigt

Menuoversigten giver et overblik over de forskellige menuer og hvilke menuer, der giver tilgang til hinanden.



Figur 13: Oversigt over AutoGreen's menuer

4.4.6 Menubeskrivelse

Menuoversigten er med til at give et overblik over hvordan de forskellige menuer tilgåes igennem systemet, og fra hvilke menuer man kan tilgå andre menuer. Hovedmenuen er som standard stedet, hvor brugeren starter, da er her muligt at monitorere drivhusklimaet. I hovedmenuen har brugeren mulighed for at tilgå de 5 undermenuer: virtuel drivhus-, historik-, plantedatabase-, systemlog- og konfigurationsmenu.

Virtuelle drivhusmenu

I det virtuelle drivhus har brugeren mulighed for at tilføje nye planter til drivhuset, redigere allerede tilstede værende planter, og herunder slette planter fra drivhuset. Uanset ønsket skal brugeren tilgå planteredigeringsmenuen.

Historikmenu

I historikmenuen har brugeren mulighed for at se data over drivhuset op til et år tilbage.

Plantedatabasemenu

I plantedatabasemenuen har brugeren mulighed for at tilføje nye planter til databasen. Ved tryk på 'tilføj plante' oprettes en ny tom virtuel plante i databasen. Denne virtuelle plante åbnes i databaseredigeringsmenuen, hvor dens parametre kan indstilles efter behov. Hvis brugeren ønsker at redigere allerede oprettede planter eller slette disse, kan brugeren trykke på den ønskede plante. Den valgte plante vil blive åbnet gennem databaseredigeringsmenuen, og det er her muligt at redigere eller slette planten.

Systemlogmenu

I systemloggen har brugeren mulighed for at se systemhændelser, f.eks. hvis systemet vælger at åbne et vindue, starte en blæser, eller bruge varmelegemet.

Konfigurationsmenu

I konfigurationsmenu har brugeren mulighed for at tilgå 4 undermenuer: E-mailmenu, Notifikationsmenu, Tid- og datomenu, samt Hardware Indstillingsmenu.

E-mailmenu

I E-mailmenuen, vises 3 kolonner, hvor brugeren har mulighed for at indtaste E-mail adresse, som skal modtage notifikationer.

Notifikationsmenu

I notifikationsmenuen har brugeren mulighed for at slå notifikationer til og fra for både advarselsnotifikationer og daglige notifikationer.

Tids- og datomenu

I Tids- og datomenuen har brugeren mulighed for at ændre dato og tid.

Hardware Indstillingsmenu

I Hardware Indstillingsmenu har brugeren mulighed for at vælge hvilke akutuatorer drivhuset skal bruge. Hvis brugeren ønsker at spare strøm, kan blæser og varmelegeme fravælges til regulering temperaturen.

4.5 Protokol for UART

I projektforløbets senere faser deles arbejdet op mellem en HW- og en SW-gruppe. SW gruppen har ansvar for design og implementering af SW på DevKit8000, mens HW gruppen har ansvar for design og realisering af HW og SW på PSoC4 Pioneer Kits. UART kommunikationen mellem PSoC Master og DevKit8000 defineres derved som grænsefladen mellem HW og SW, omend en del af funktionaliteten på PSoC4 Pioneer Kits realiseres vha. SW.

4.5.1 UART indstillinger

- Baud rate: 9600
- Antal bits: 8
- Antal stop bits: 1
- Paritet: Ingen

4.5.2 Datavalidering

For at sikre validering af data sendt fra DevKit8000 til PSoC4 Master, sendes der altid svar tilbage fra PSoC4 Master til DevKit8000. Svaret består af en gentagelse af den modtagne kommando og evt. nogle dataværdier.

Såfremt der går noget galt i I2C kommunikationen i HW delen af systemet, sendes en fejlkode til DevKit8000. Derved er der mulighed for at SW på DevKit8000 kan logge fejlhændelser i systemloggen, og fx gensende kommandoer eller kassere data.

Da hver enkelt byte, der sendes via UART er vedlagt en paritetsbit sikrer vi os til dels at hver byte overføres korrekt.

Når DevKit8000 sender en kommando via UART skal PSoC Master svare indenfor 2 sekunder. Såfremt dette ikke sker, sendes kommandoen igen mindst to gange. Alle kommandoer udføres serielt, hvilket vil sige at næste kommando ikke sendes før der er modtaget svar på den foregående.

4.5.3 Kommandoer

Kommando (DevKit -> PSoC Master)	Svar (PSoC Master -> DevKit)	Beskrivelse	Bitmønster
RequestTemp		Forespørgsel om data fra tempera- tursensor.	'T'
	Temp	Temperaturværdi fra sensor.	'T' efterfulgt af char med decimal- værdi 1-200. 1 svarer til -19,5 grader, 200 svarer til 80 grader. Der sendes "XT", hvis der ikke er kontakt til temperatursensoren.
RequestLight		Forespørgsel om data fra tempera- tursensor.	'L'

	Light	Lysværdi fra sensor.	'L' efterfulgt af char med decimalværdi 1-100. 1 svarer til XX Lumen, 100 svarer til XX Lumen. Der sendes "XL", hvis der ikke er kontakt til lyssensoren.
RequestAirHum		Forespørgsel om data fra sensor for luftfugtighed.	'A'
	AirHumidity	Luftfugtværdi fra sensor.	'A' efterfulgt af char med decimalværdi 1-100. 1 svarer til 1%, 100 svarer til 100%. Der sendes "XA", hvis der ikke er kontakt til luftfugtsensoren.
RequestSoilHum		Forespørgsel om data fra en bestemt jordfugtsensor.	'S' efterfulgt af char for sensornummer (1-6 i ASCII).
	SoilHum	Jordfugtværdi fra sensor.	'S' efterfulgt af char for sensornummer (1-6 i ASCII). Herefter char med decimalværdi 1-10. 1 svarer til trin 1, 10 svarer til trin 10. Der sendes "XS" efterfulgt af char for sensornummer, hvis der ikke er kontakt til jordfugtsensoren.
TurnHeatOn		Tænder for varmelegeme.	'H'
	HeatIsOn	Ack.	'H' "XH", hvis der ikke er kontakt med aktuator for varmelegeme.
TurnHeatOff		Slukker for varmelegeme.	'K'
	HeatIsOff	Ack.	'K' "XK", hvis der ikke er kontakt med aktuator for varmelegeme.
AdjustWindow		Indstiller vindue.	'W' efterfulgt af ASCII værdien for 0 eller 1. 0 svarer til at lukke vindue, 1 svarer til at åbne vindue.
	WindowState	Ack.	'W' "XW", hvis der ikke er kontakt til aktuator for vindue.
Ventilation		Starter eller stopper blæsere.	'V' efterfulgt af ASCII værdien for 0 eller 1. 0 svarer til at slukke blæsere, 1 svarer til at tænde blæsere.
	VentilationStatus	Ack.	'V' "XV", hvis der ikke er kontakt til aktuator for blæsere.

Watering		Aktivere eller deaktivere vandings-signaler.	'F' efterfulgt af char for plantenummer (1-6 i ASCII). Herefter ASCII værdien for 0 eller 1. 0 svarer til ingen vanding, 1 svarer til vanding.
	WaterStatus	Ack.	'F' Der sendes "XF", hvis der ikke er kontakt til aktuator for vanding.

Tabel 24: Kommando liste for UART kommunikation

5 Hardware Design

5.1 Version

Dato	Version	Initialer	Ændring
30. marts	1	MHG	I2C Protokol og design af Aktuator.
8. april	2	MHG	Mindre rettelser i Aktuatordesign.
10. april	3	MHG	Skrevet det sidste (varmelegeme) design i aktuator blokken.
23. april	4	MHG	Lavet mindre rettelser i et par diagrammer.
27. april	5	MHG	Skrevet design for Strømforsyning.
1. maj	6	MHG	Skrevet design for Slave Jordfugt.
13. maj	7	PKP	I ² C protokol opdateret med ny temperatursensor og PSoC Master funktionsbeskrivelser opdateret.

5.2 I²C Protokol

Dette afsnit omhandler kommunikationen mellem alle I²C kommunikerende komponenter i projektet.

5.2.1 Temperatursensor

Temperatursensoren er valgt til at være LM75 [8]. Informationer til protokollen er fundet i databladet.

Slave:	Temperatursensor
Adresse:	0x48
Bemærkninger:	scl: min 400kHz

Tabel 25: I²C Oplysninger for temperatursensor

Når der skal læses data fra sensoren, sker det jf. Figur 14, fundet i det respektive datablad [8].

UPPER BYTE								LOWER BYTE							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Sign bit 1= Negative 0 = Positive	MSB 64°C	32°C	16°C	8°C	4°C	2°C	1°C	LSB 0.5°C	X	X	X	X	X	X	X

X = Don't care.

Figur 14: I²C read protokol for temperatursensor

5.2.2 Slave Aktuator

Formålet med Slave Aktuatoren er at udføre forskellige opgaver, via et varmelegeme, blæsere, en motor og seks vandings bools. Disse opgaver bliver anmodet af Master PSoC. Under dette afsnit vil protokolen mellem Slave Aktuator og Master PSoC blive gennemgået.

Adresse: 0x42	
Kommando	Beskrivelse
WriteAdjustWindow	Åbning/Lukning af vindue
WriteAdjustHeat	Tænd/Sluk for varme
WriteAdjustVentilation	Juster ventilation
WriteAdjustIrrigation	Juster vanding
ReadStatus	Anmodning om status

Tabel 26: I²C Kommandoer for Slave Aktuator

W7	W6	W5	W4	W3	W2	W1	W0
0x0	Don't Cares			Position for vindue, 0x0 = lukket, 0xF = åben			

Tabel 27: I²C Kommando WriteAdjustWindow

For at gøre systemet opgraderbar er der valgt at klargøre fire bits til position af vindue. Systemet som det er skal kun have mulighed for enten at åbne eller lukke vinduet. Det er dog mulighed for at give vinduet flere positioner mellem fuld åben og lukket.

H7	H6	H5	H4	H3	H2	H1	H0
0x1	Don't Care			Tænd/Sluk varmelegeme, 0x0 = off, 0x7 = on			

Tabel 28: I²C Kommando WriteAdjustHeat

På samme måde som vinduet har systemet kun mulighed for enten at tænde eller slukke for varmelegemet. Der er der klargjort tre bits for at gøre det muligt at opgradere denne funktionalitet til et PWM signal, således at dette kan reguleres til trin mellem tændt og slukket.

V7	V6	V5	V4	V3	V2	V1	V0
0x2	Don't Care						Tænd/Sluk ventilation, 0x0 = off, 0x7 = on

Tabel 29: I²C Kommando WriteAdjustVentilation

Til regulering af blæsere er der klargjort tre bits. Systemet skal blot kunne regulere blæserne mellem tændt og slukket. Dermed er der mulighed for at udvide funktionaliteten af reguleringen til trin mellem tændt og slukket.

I7	I6	I5	I4	I3	I2	I1	I0
0x3	Værdi for pins til vanding, I5: nr. 6 – I0: nr. 1, 1 = on, 0 = off						

Tabel 30: I²C Kommando WriteAdjustIrrigation

Til regulering af de seks vandingsbools er der klargjort seks bits. Da disse er bools, altså tændt/-slukket signaler, er der ikke nogen grund til at klargøre mere plads til fremtidig opgraderinger.

W3	W2	W1	W0	H2	H1	H0	V2	V1	V0	I5	I4	I3	I2	I1	I0
Position for vindue, 0x0 = lukket, 0xF = åben				Status for Varmelegeme, 1 = on, 0 = off			Status for ventilation, 0x0 = off, 0x7 = on			Status for pins til vanding, I5: nr. 6 – I0: nr. 1, 1 = on, 0 = off					

Tabel 31: I²C Kommando ReadStatus

ReadStatus gør det muligt for Master PSoC at spørge hvilken status de forskellige aktuatorer har. Dette passer overens med de klargjorte bits til de forskellige reguleringer for at simplificere processen.

5.2.3 Slave Jordfugt

Formålet med Slave Jordfugt er at måle og bearbejde værdier læst fra op til seks jordfugt sensorer. Disse data bliver anmodet af Master PSoC og herefter spurgt efter. Under dette afsnit vil protokolen mellem Slave Jordfugt og Master PSoC blive gennemgået.

Adresse: 0x32	
Kommando	Beskrivelse
WriteDesiredSensor	Sensor der ønskes data fra
ReadDesiredSensor	Anmodning om sensor data

Tabel 32: I²C Kommandoer for Slave Jordfugt

S7	S6	S5	S4	S3	S2	S1	S0
Don't Care							Værdi for sensor nummer der ønskes læsning af, ved næste ReadStatus; 0x0 = sensor 1, 0x5 = sensor 6

Tabel 33: I²C Kommando WriteDesiredSensor

Master PSoC har via denne kommando mulighed for, at fortælle Slave Jordfugt hvilken sensor der ønskes data fra ved næste læsning. Da der er seks sensorer er der klargjort tre bits til værdi af sensor nummer.

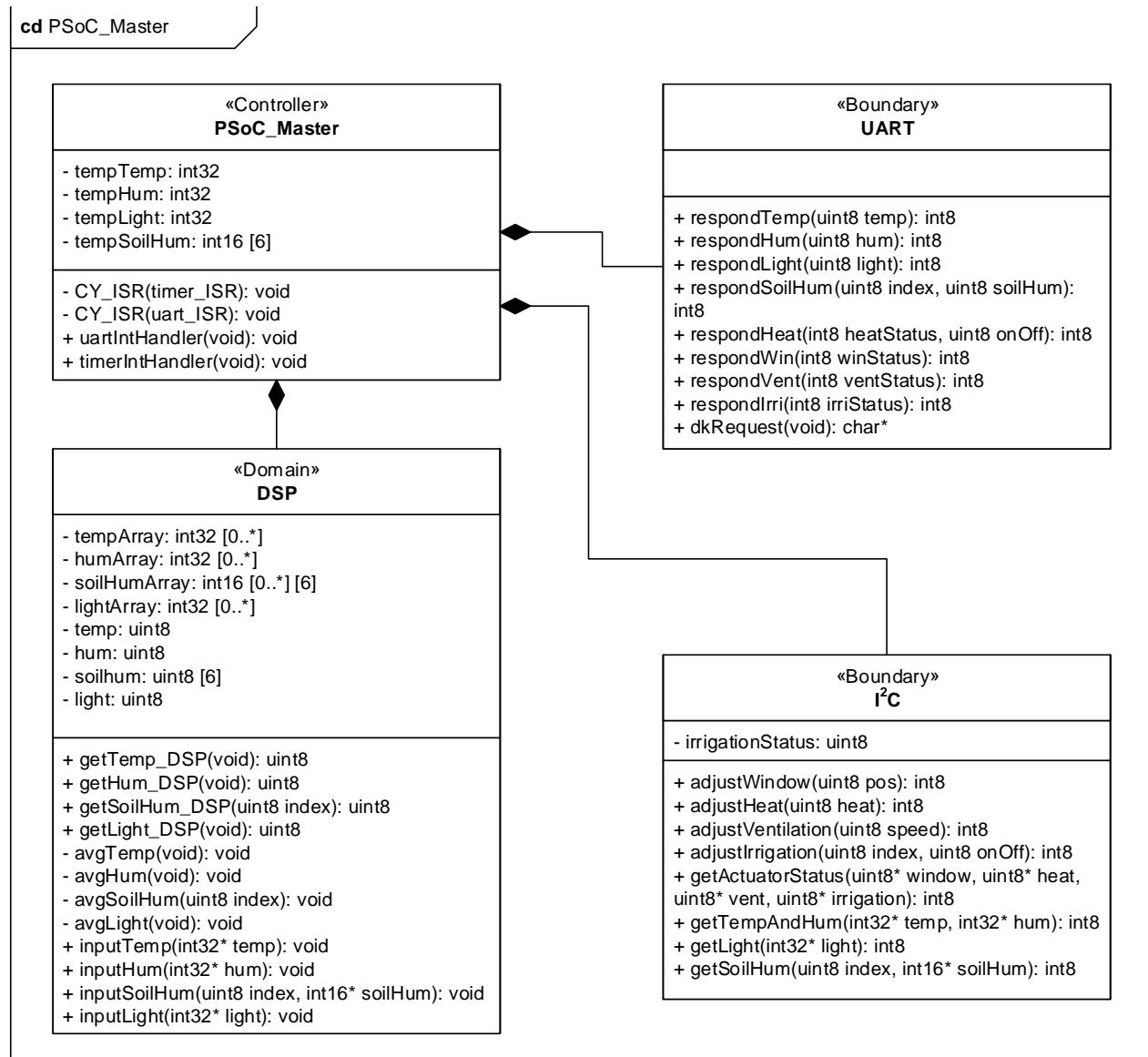
S7	S6	S5	S4	S3	S2	S1	S0
Sensor status, 0x1 = deaktive, 0x0 = aktive							Sensor værdi, 0x1 - 0x64 (1 - 100)

Tabel 34: I²C Kommando ReadDesiredSensor

Ved læsning vil Slave Jordfugt besvare Master PSoC med en værdi, mellem 1 og 100, fra den sensor Master PSoC'en har bedt om data fra via WriteDesiredSensor. I tilfælde af at der er opstået en fejl vil MSB være høj.

5.3 PSoC Master

På Figur 15 ses klassediagrammet for Master PSoC. Der er blevet designet 4 klasser der håndterer hver sin del af arbejdet, som masteren skal udføre. Der er valgt to boundaryklasser, som håndterer kommunikation over hhv. UART og I²C . Udeover dette er der en domænekasse, som indeholder alle de målte dataværdier, der er modtaget af sensorerne.



Figur 15: Klassediagramm für Master PSoC

5.3.1 Klassebeskrivelser

Controllerklasse PSoC_Master

Attributter

Navn	Type	Beskrivelse
tempTemp	int32	Midlertidig variabel til opbevaring af temperatur.
tempHum	int32	Midlertidig variabel til opbevaring af luftfugtighed.
tempLight	int32	Midlertidig variabel til opbevaring af lysintensitet.
tempSoilHum	int16 [6]	Array af midlertidige variabler til opbevaring af jordfugtighed.

Tabel 35: Attributter for klassen PSoC_Master

Metoder

Prototype	<code>void CY_ISR(timer_ISR)</code>
Parametre	<code>timer_ISR</code> Vector for den givne interrupt servicerutine.
Returværdi	-
Beskrivelse	Denne interrupt service rutine bliver kaldt, når en ønsket tidsperiode er forløbet. Rutinen kalder metoder i boundaryklassen I ² C, der varetager indhentning af data fra sensorer.

Tabel 36: CY_ISR(timer_ISR)

Prototype	<code>void CY_ISR(uart_ISR)</code>
Parametre	<code>uart_ISR</code> Vector for den givne interrupt servicerutine.
Returværdi	-
Beskrivelse	Denne interrupt service rutine bliver kaldt, når der modtages noget på UART fra DevKit8000. Rutinen kalder metoder i boundaryklasserne I ² C og UART. Formålet med interrupten er at håndtere forskellige typer af forespørgsler.

Tabel 37: CY_ISR(uart_ISR)

Prototype	<code>void timerIntHandler(void)</code>
Parametre	-
Returværdi	-
Beskrivelse	Kontrollerer et flag sat af timer_ISR. Hvis flag er sat, spørger den på I ² C om sensor-værdier.

Tabel 38: timerIntHandler()

Prototype	<code>void uartIntHandler(void)</code>
Parametre	-
Returværdi	-
Beskrivelse	Kontrollerer et flag sat af <code>uart_ISR</code> . Hvis flaget er sat, spørger den på I ² C om sensor-værdier.

Tabel 39: timerIntHandler()

Boundaryklasse UART

Metoder

Prototype	<code>int8 respondTemp(uint8 temp)</code>
Parametre	<code>uint8 temp</code> Den nyeste midlede temperatur hentet fra domæneklassen DSP.
Returværdi	<code>int8</code> Er denne værdi nul er der blevet returneret en gyldig værdi. Hvis værdien er -1 er der blevet returneret 'XT' over UART.
Beskrivelse	Hvis parameter værdien ligger inden for decimalværdien 1-200 (begge inklusive), skal metoden sende en char 'T', efterfulgt af temp parameteren, over UART. Er værdien er lig nul, skal metoden sende strengen "XT".

Tabel 40: respondTemp

Prototype	<code>int8 respondHum(uint8 hum)</code>
Parametre	<code>uint8 hum</code> Den nyeste midlede luftfugtighed hentet fra domæneklassen DSP.
Returværdi	<code>int8</code> Er denne værdi nul er der blevet returneret en gyldig værdi. Hvis værdien er -1 er der blevet returneret 'XA' over UART.
Beskrivelse	Hvis parameter værdien ligger inden for decimalværdien 1-100 (begge inklusive), skal metoden sende en char 'A', efterfulgt af hum parameteren, over UART. Hvis værdien er lig nul, skal metoden sende strengen "XA".

Tabel 41: respondHum

Prototype	<code>int8 respondLight(uint8 light)</code>
Parametre	<code>uint8 light</code> Den nyeste midlede lysintensitet hentet fra domæneklassen DSP.
Returværdi	<code>int8</code> Er denne værdi nul er der blevet returneret en gyldig værdi. Hvis værdien er -1 er der blevet returneret 'XL' over UART.
Beskrivelse	Hvis parameter værdien ligger inden for decimalværdien 1-100 (begge inklusive), skal metoden sende en char 'L' efterfulgt af light parameteren over UART. Hvis værdien er lig nul, skal metoden sende strengen "XL".

Tabel 42: respondLight

Prototype	<code>int8 respondSoilHum(uint8 index, uint8 soilHum)</code>
Parametre	<code>uint8 soilHum</code> Den nyeste midlede jordfugtighed hentet fra domæneklassen DSP. <code>uint8 index</code> Indexet fortæller hvilket sensornummer der svares fra. Kan være værdierne 0-5.
Returværdi	<code>int8</code> Er denne værdi nul er der blevet returneret en gyldig værdi. Hvis værdien er -1 er der blevet returneret 'XS' over UART.
Beskrivelse	Hvis parameter værdien ligger inden for decimalværdien 1-10 (begge inklusive), skal metoden sende en char 'S', efterfulgt af index og soilHum parametrene over UART. Hvis værdien for soilHum er lig nul, skal metoden sende strengen "XS".

Tabel 43: respondSoilHum

Prototype	<code>int8 respondHeat(uint8 heatStatus, uint8 On)</code>
Parametre	<code>uint8 heatStatus</code> Returværdien fra funktionen adjustHeat i boundaryklassen I ² C ; fortæller om kommunikationen over I ² C er gået godt. <code>uint8 On</code> Returværdi til UART afhængig af hvilken kommando, der blev kaldt. 0 = off, 0 != on.
Returværdi	<code>int8</code> Er denne værdi nul er kommunikationen over I ² C gennemført. Hvis værdien er -1 er der blevet returneret en værdi tilsvarende requesten fra DevKit8000 over UART (se UART protokol side ??) og der er sket en fejl i kommunikationen over I ² C .
Beskrivelse	Hvis parameter værdien er lig nul, sendes en char tilsvarende requesten over UART. Hvis værdien er lig -1, sendes sendes en tilsvarende fejlmeldelse.

Tabel 44: respondHeat

Prototype	<code>int8 respondWin(int8 winStatus)</code>
Parametre	<code>int8 winStatus</code> Returværdien fra funktionen adjustWin i boundaryklassen I ² C . Fortæller om kommunikationen via. I ² C til vinduesaktuatoren er forløbet godt.
Returværdi	<code>int8</code> Er denne værdi nul er kommunikationen over I ² C gennemført. Hvis værdien er -1 er der blevet returneret 'XK' over UART og der er sket en fejl i kommunikationen over I ² C .
Beskrivelse	Hvis parameter værdien er lig nul, sendes en char 'K' over UART. Hvis værdien er lig -1, sendes strengen "XK".

Tabel 45: respondWin

Prototype	<code>int8 respondVent(int8 ventStatus)</code>
Parametre	<code>int8 ventStatus</code> Returværdien fra funktionen adjustVent i boundaryklassen I ² C . Fortæller om kommunikationen via I ² C til ventilatoraktuatoren er forløbet uden problemer.
Returværdi	<code>int8</code> Er denne værdi 0, er kommunikationen over I ² C gennemført. Hvis værdien er -1 er der blevet returneret 'XV' over UART og der er sket en fejl i kommunikationen over I ² C .
Beskrivelse	Hvis parameterværdien er lig nul, sendes en char 'V' over UART, hvilket indikerer at det er gået godt. Hvis værdien er lig -1, sendes strengen "XV".

Tabel 46: respondVent

Prototype	<code>int8 respondIrri(int8 irriStatus)</code>
Parametre	<code>int8 irriStatus</code> Returværdien fra funktionen adjustIrrigation i boundaryklassen I ² C . Fortæller om kommunikationen via I ² C til irrigationsaktuatoren er forløbet uden problemer.
Returværdi	<code>int8</code> Er denne værdi nul, er kommunikationen over I ² C gennemført. Hvis værdien er -1 er der blevet returneret 'XF' over UART og der er sket en fejl i kommunikationen over I ² C .
Beskrivelse	Hvis parameterværdien er lig nul, sendes en char 'F' over UART, hvilket indikerer at det er gået godt. Hvis værdien er lig -1, sendes strengen "XF".

Tabel 47: respondIrri

Boundaryklasse I²C

Attributter

Navn	Type	Beskrivelse
irrigationStatus	uint8	Indeholder den aktuelle status for vandingsaktuatorer (tændt eller slukkede). Bit 0 – 5 er hhv. aktuatorerne fra 1 – 6. Nul betyder slukket og et betyder tændt.

Tabel 48: Attributter for klassen I²C

Metoder

Prototype	<code>int8 adjustWindow(uint8 pos)</code>
Parametre	<code>uint8 pos</code> Den ønskede status for vinduet. Kan være hhv. 0xFF for åben og 0x00 for lukket.
Returværdi	<code>int8</code> Er værdien 0, er kommunikation via I ² C gået godt. Hvis værdien er -1, er der sket en fejl.
Beskrivelse	Metoden kan justere positionen for vinduet i drivhuset. Sender kommandoen "WriteAdjustWindow" via I ² C bussen (se I ² C protokol på side 47).

Tabel 49: adjustWindow

Prototype	<code>int8 adjustHeat(uint8 heat)</code>
Parametre	<code>uint8 heat</code> Bestemmer intensiteten af varmen, 0x00 er ingen varme, 0xFF er fuld varme.
Returværdi	<code>int8</code> Er værdien 0, er kommunikation via I ² C gået godt. Hvis værdien er -1, er der sket en fejl (se I ² C protokol på side 47).
Beskrivelse	Slukker eller tænder for varmeaktuatoren. Sender komandoen "WriteAdjustHeat" via I ² C bussen (se I ² C protokol på side 47).

Tabel 50: adjustHeat

Prototype	<code>int8 adjustVentilation(uint8 speed)</code>
Parametre	<code>uint8 speed</code> Beskriver ventilatoraktuatornes tilstand. 0x00 svarer til slukket og 0xFF svarer til fuld hastighed.
Returværdi	<code>int8</code> Er værdien 0, er kommunikation via I ² C gået godt. Hvis værdien er -1, er der sket en fejl (se I ² C protokol på side 47).
Beskrivelse	Slukker eller tænder for ventilation. Sender kommandoen "WriteAdjustVentilation" via I ² C bussen (se I ² C protokol på side 47).

Tabel 51: adjustVentilation

Prototype	<code>int8 adjustIrrigation(uint8 index, uint8 on)</code>
Parametre	<code>uint8 index</code> Indeksoperator for hvilken vandingsaktuator der skal aktiveres. Første = 0, sidste = 5. <code>uint8 on</code> Beskriver tilstanden for vandingsaktuatoren. 0x00 svarer til slukket og 0xFF svarer til tændt.
Returværdi	<code>int8</code> Er værdien 0, er kommunikation via I ² C gået godt. Hvis værdien er -1, er der sket en fejl (se I ² C protokol på side 47).
Beskrivelse	Slukker eller tænder for individuelle vandingsaktuatorer. Sender komandoen "WriteAdjustIrrigation" via I ² C bussen (se I ² C protokol på side 47).

Tabel 52: adjustIrrigation

Prototype	<code>int8 getActuatorStatus(uint8* window, uint8* heat, uint8* vent, uint8* irrigation)</code>
Parametre	<code>uint8* window</code> Pointer til variable som status for vinduet skrives i. <code>uint8* heat</code> Pointer til variable som status for varmelegeme skrives i. <code>uint8* vent</code> Pointer til variable som status for ventilator skrives i. <code>uint8* irrigation</code> Pointer til variable som status for vandingsaktuatorer skrives i.
Returværdi	<code>int8</code> Er værdien 0, er kommunikation via I ² C gået godt. Hvis værdien er -1, er der sket en fejl.
Beskrivelse	Giver overblik over aktuatorslavens tilstand. Der henvises til I ² C protokollen på side 47 for yderligere information.

Tabel 53: getActuatorStatus

Prototype	<code>int8 getTempAndHum(int32* temp, int32* hum)</code>
Parametre	<code>int32* temp</code> Pointer til variabel, hvori ubehandlet temperaturdata i drivhuset skrives. <code>int32* hum</code> Pointer til variabel, hvori ubehandlet luftfugtighedsdata i drivhuset skrives. Omregningsformel kan findes i databladet for sensoren. [5]
Returværdi	<code>int8</code> Er værdien 0, er kommunikation via I ² C gået godt. Hvis værdien er -1, er der sket en fejl.
Beskrivelse	Metoden skriver ubehandlet data fra sensoren i to variable. Der henvises til I ² C protokollen på side 47 og til sensorens datablad [5] for yderligere information.

Tabel 54: getTempAndHum

Prototype	<code>int8 getLight(int32* light)</code>
Parametre	<code>int32* light</code> Pointer til variabel, hvori ubehandlet lysintensitetsdata i drivhuset skrives. Omregningsformel kan findes i databladet for sensoren. [6].
Returværdi	<code>int8</code> Er værdien 0, er kommunikation via I ² C gået godt. Hvis værdien er -1, er der sket en fejl.
Beskrivelse	Metoden skriver ubehandlet data fra sensoren i en variabel. Der henvises til I ² C protokollen på side 47 og til sensorens datablad [6] for yderligere information.

Tabel 55: getLight

Prototype	<code>int8 getSoilHum(uint8 index, int16* soilHum)</code>
Parametre	<code>uint8* index</code> Indeks for hvilken jordfugtsensor der ønskes at læse fra, den første sensor hedder 0 og den sidste 5. <code>int16* soilHum</code> Pointer til variabel, hvori ubehandlet jordfugtighedsdata i drivhuset skrives.
Returværdi	<code>int8</code> Er værdien 0, er kommunikation via I ² C gået godt. Hvis værdien er -1, er der sket en fejl.
Beskrivelse	Metoden skriver ubehandlet data fra sensoren i en variabel.

Tabel 56: getSoilHum

Domainklasse DSP

Attributter

Navn	Type	Beskrivelse
tempArray	int32	Array der indeholder måleværdier fra temperaturmåler.
humArray	int32	Array der indeholder måleværdier fra fugtighedsmåler.
soilHumArray	int16	To-dimentionelt array der indeholder målinger fra jordfugtsensorerne.
lightArray	int32	Array der indeholder måleværdier fra lyssensoren.
temp	uint8	Variabel der indeholder den midlede og konverterede værdi af temperaturen, som DevKit8000 kan efterspørge.
hum	uint8	Variabel der indeholder den midlede værdi af fugtigheden, som DevKit8000 kan efterspørge.
soilHum	uint8	Variabel der indeholder den midlede værdi af jordfugtigheden, som DevKit8000 kan efterspørge.
light	uint8	Variabel der indeholder den midlede værdi af lysintensiteten, som DevKit8000 kan efterspørge.

Tabel 57: Attributter for klassen DSP

Metoder

Navn	Type	Beskrivelse
tempArray	int32[0...*]	Pointer til et array på en given størrelse, som indeholder ubearbejdede målinger af temperaturen, indhentet fra boundaryklassen I ² C .
humArray	int32[0...*]	Pointer til et array på en given størrelse, som indeholder ubearbejdede målinger af luftfugtigheden, indhentet fra boundaryklassen I ² C .
soilHumArray	int16[0...*][6]	Todimensionelt array på 6 pladser, som indeholder arrays med ubearbejdede målinger af jordfugtigheden, indhentet fra boundaryklassen I ² C .
lightArray	int32[0...*]	Pointer til et array på en given størrelse, som indeholder ubearbejdede målinger af luftfugtigheden, indhentet fra boundaryklassen I ² C .
temp	uint8	Indeholder den nyeste midlede temperaturværdi.
hum	uint8	Indeholder den nyeste midlede luftfugtighedsværdi.
soilHum	uint8[6]	Array der indeholder de nyeste midlede jordfugtighedsværdier.
light	uint8	Indeholder den nyeste midlede lysintensitetsværdi.

Tabel 58: Attributter for klassen I²C

Prototype	int8 getTemp_DSP(void)
Parametre	-
Returværdi	int8 Metoden returnerer variablen temp.
Beskrivelse	Kan bruges til at hente den midlede temperatur, der skal sendes via UART til DevKit8000.

Tabel 59: getTemp_DSP

Prototype	int8 getHum_DSP(void)
Parametre	-
Returværdi	int8 Metoden returnerer variablen hum.
Beskrivelse	Kan bruges til at hente den midlede luftfugtighed, der skal sendes via UART til DevKit8000.

Tabel 60: getHum_DSP

Prototype	<code>int8 getSoilHum_DSP(uint8 index)</code>
Parametre	<code>uint8 index</code> Fortæller hvilken jordfugtighedssensor der returneres værdier fra.
Returværdi	<code>int8</code> Returnerer variablen soilHum der tilsvarer parametret index.
Beskrivelse	Metoden bruges til at hente den midlede jordfugtighed, der skal sendes via UART til DevKit8000.

Tabel 61: getSoilHum_DSP

Prototype	<code>int8 getLight_DSP(void)</code>
Parametre	-
Returværdi	<code>int8</code> Metoden returnerer variablen light.
Beskrivelse	Kan bruges til at hente den midlede lysintensitet, der skal sendes via UART til DevKit8000.

Tabel 62: getLight_DSP

Prototype	<code>void avgTemp(void)</code>
Parametre	-
Returværdi	-
Beskrivelse	Metoden beregner en middelværdi for temperaturene i tempArray og konverterer svaret til et passende format og gemmer det i temp. Se UART protokollen side ?? og databladet for temperatursensoren [5].

Tabel 63: avgTemp

Prototype	<code>void avgHum(void)</code>
Parametre	-
Returværdi	-
Beskrivelse	Metoden beregner en middelværdi for luftfugtigheden i humArray og konverterer svaret til et passende format og gemmer det i hum. Se UART protokollen side ?? og databladet for temperatursensoren [5].

Tabel 64: avgHum

Prototype	<code>void avgSoilHum(void)</code>
Parametre	-
Returværdi	-
Beskrivelse	Metoden beregner en middelværdi for hver sæt værdier for jordfugtighed i soilHumArray og konverterer svaret til et passende format og gemmer det i soilHum arrayet. Se UART protokollen side ??.

Tabel 65: avgSoilHum

Prototype	<code>void avgLight(void)</code>
Parametre	-
Returværdi	-
Beskrivelse	Metoden beregner en middelværdi for lysintensiten i lightArray og konverterer svaret til et passende format og gemmer det i light. Se UART protokollen side ?? og databladet for temperatursensoren [6].

Tabel 66: avgLight

Prototype	<code>void inputTemp(int32* temp)</code>
Parametre	<code>int32* temp</code> En pointer til en variabel der ønskes indlæst i tempArray.
Returværdi	-
Beskrivelse	Metoden har til formål at indsætte værdien fra tempTemp i tempArray, og sørger for at flytte tempArray pointeren til næste plads der skal skrives i næste gang den bliver kaldt. Ydermere kalder den metoden avgTemp.

Tabel 67: inputTemp

Prototype	<code>void inputHum(int32* hum)</code>
Parametre	<code>int32* hum</code> En pointer til en variabel der ønskes indlæst i humArray.
Returværdi	-
Beskrivelse	Metoden har til formål at indsætte værdien fra tempHum i humArray, og sørger for at flytte humArray pointeren til næste plads der skal skrives i næste gang den bliver kaldt. Ydermere kalder den metoden avgHum.

Tabel 68: inputHum

Prototype	<code>void inputSoilHum(uint8 index, int16* soilHum)</code>
Parametre	<p><code>uint8 index</code> Indeksoperator der fortæller hvilket jordfugtarray der skal skrives i. 0 er den første sensor og 5 er den sidste.</p> <p><code>int16* soilHum</code> En pointer til en variabel der ønskes indlæst i det givne soilHumArray. Peger på variablen tempSoilHum med tilsvarene indeks i PSoC_Master klassen.</p>
Returværdi	-
Beskrivelse	Metoden indsætter værdien soilHum i soilHumArray, og sørger fra at flytte soilHumArray pointeren til næste plads der skal skrives i næste gang funktionen bliver kaldt. Ydermere kalder den funktionen avgSoilHum.

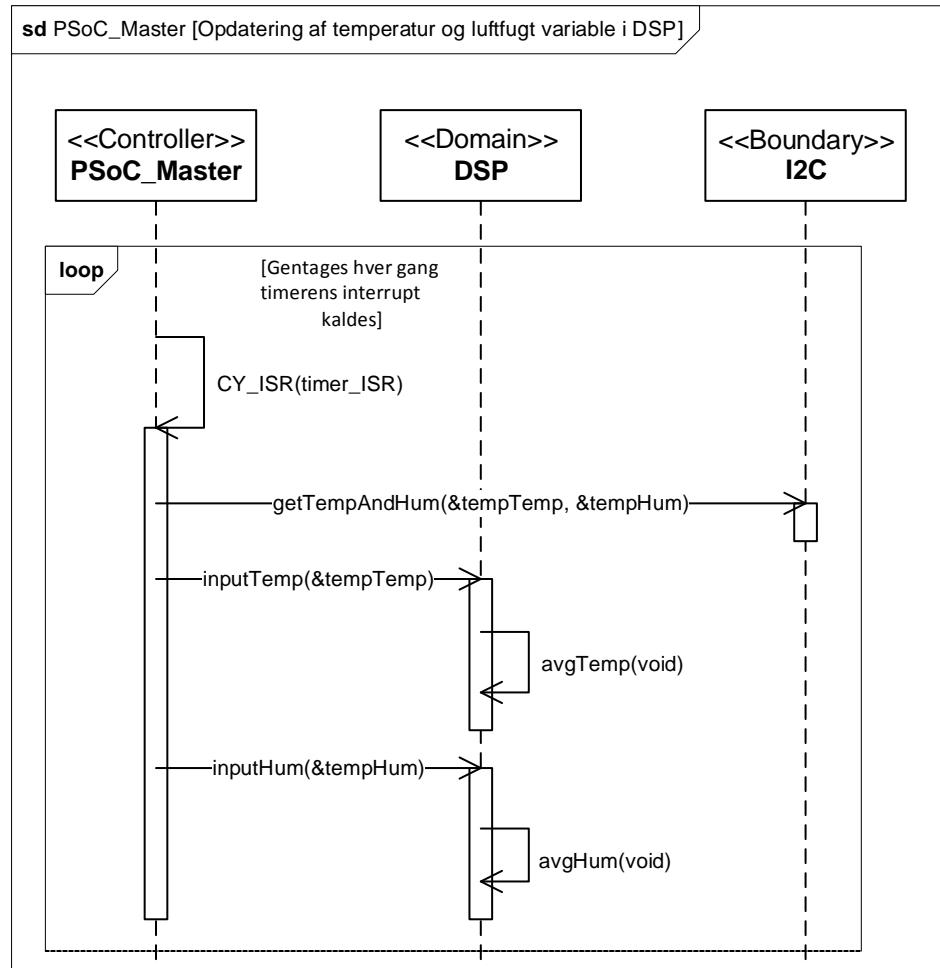
Tabel 69: inputSoilHum

Prototype	<code>void inputLight(int32* light)</code>
Parametre	<p><code>int32* light</code> En pointer til en variabel der ønskes indlæst i lightArray.</p>
Returværdi	-
Beskrivelse	Metoden har til formål at indsætte værdien fra tempLight i lightArray, og sørger fra at flytte lightArray pointeren til næste plads der skal skrives i, næste gang funktionen bliver kaldt. Ydermere kalder den metoden avgLight.

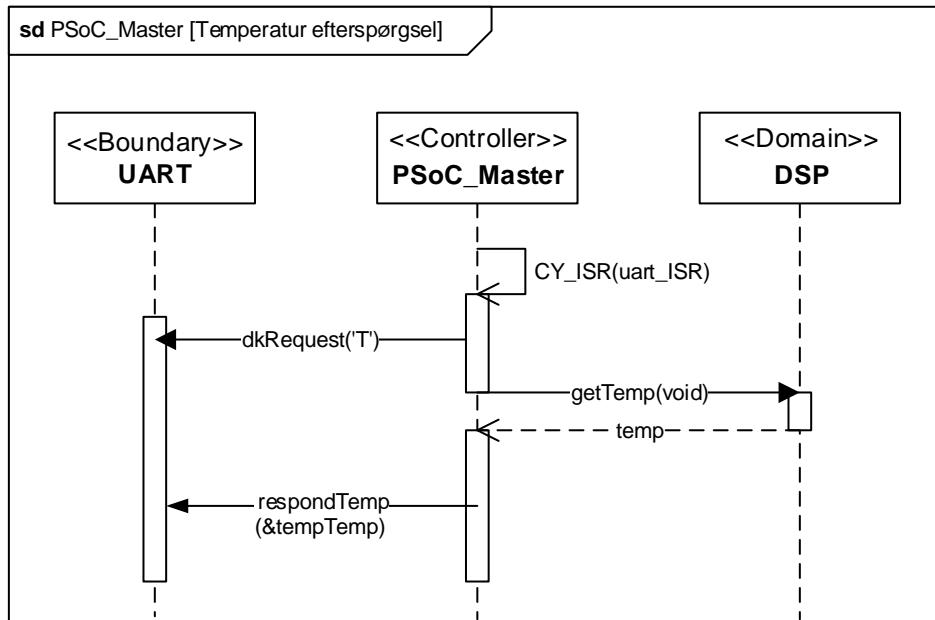
Tabel 70: inputLight

5.3.2 Sekvensdiagrammer

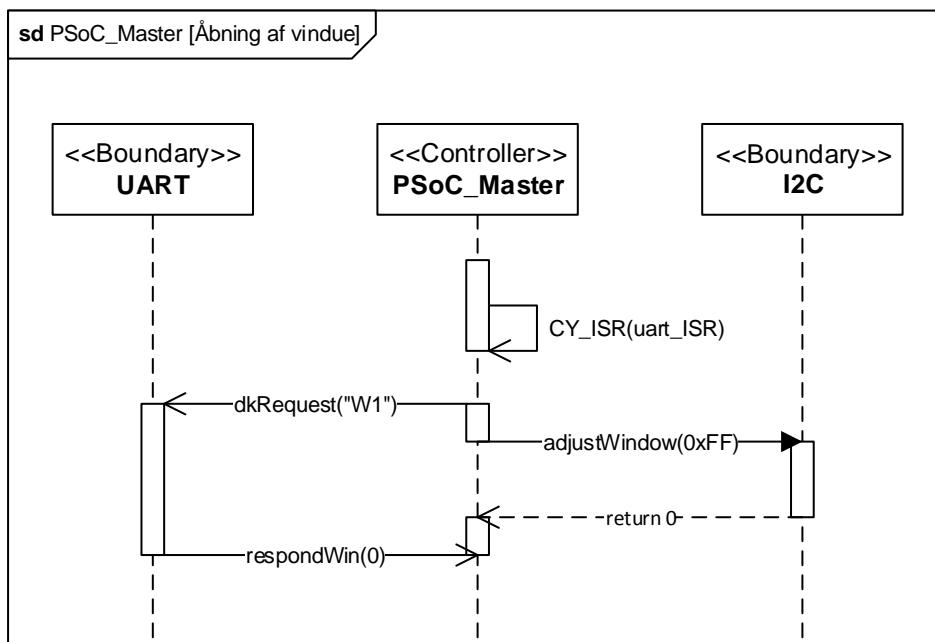
På Figur 16, 17 og 18, ses der forskellige forløb af masterens aktivitet. Sekvensdiagrammerne er eksempler på den funktionalitet der forekommer i masteren.



Figur 16: Sekvensdiagram over opdatering af sensorer.



Figur 17: Sekvensdiagram over forespørgsel af temperatur.

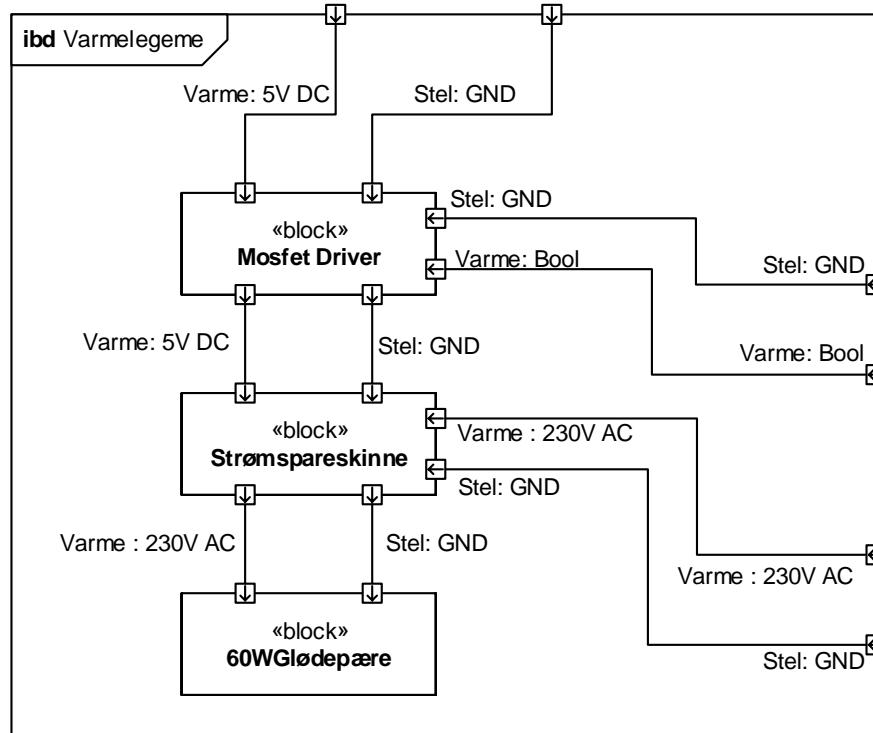


Figur 18: Sekvensdiagram over åbning af vindue.

5.4 Aktuator Design (Henrik og Morten)

Dette afsnit omhandler design af blokken Aktuator. Den opdeles i underblokkene Varmelegeme, Blæsere, Vinduesmotor og PSoC4.

5.4.1 Varmelegeme

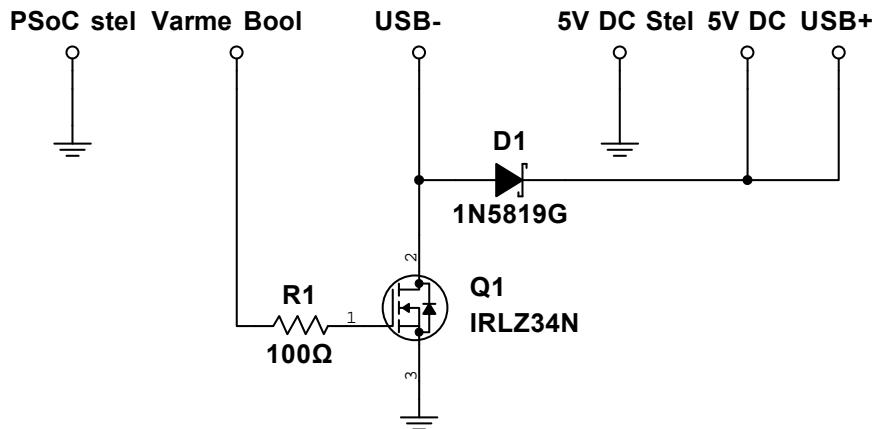


Figur 19: IBD for underblokken Varmelegeme i Aktuator

Ovenstående diagram (Figur 19) viser interne forbindelser i underblokken Varmelegeme i Aktuator. For at undgå håndtering af 230V AC, består underblokken af en USB strømspareskinne, så selve varmelegemet (1-3 stk. 100W Glødepærer) kan tændes og slukkes med et 5V DC signal. Antallet af tilkoblede glødepærer bestemmes under senere tests.

Aktuatorens SW er designet således at der nemt kan opgraderes til PWM styring af varmelegemet. Dette viser sig desværre at være umuligt med denne opstilling, da USB strømspareskinnen indeholder et mekanisk relæ; det er ikke muligt at opnå en frekvens hvor lyset ikke blinker. Dette vil sandsynligvis resultere i en sprunget glødepære. En mulig løsning på problemet kunne være at tænde og slukke de 230V AC direkte med Mosfet transistoren, men vi må ikke håndtere så høje spændinger. En anden mulig løsning er at anvende for eksempel 12V glødepærer i stedet. Der skal dog nok temmelig mange til for at opnå samme effekt.

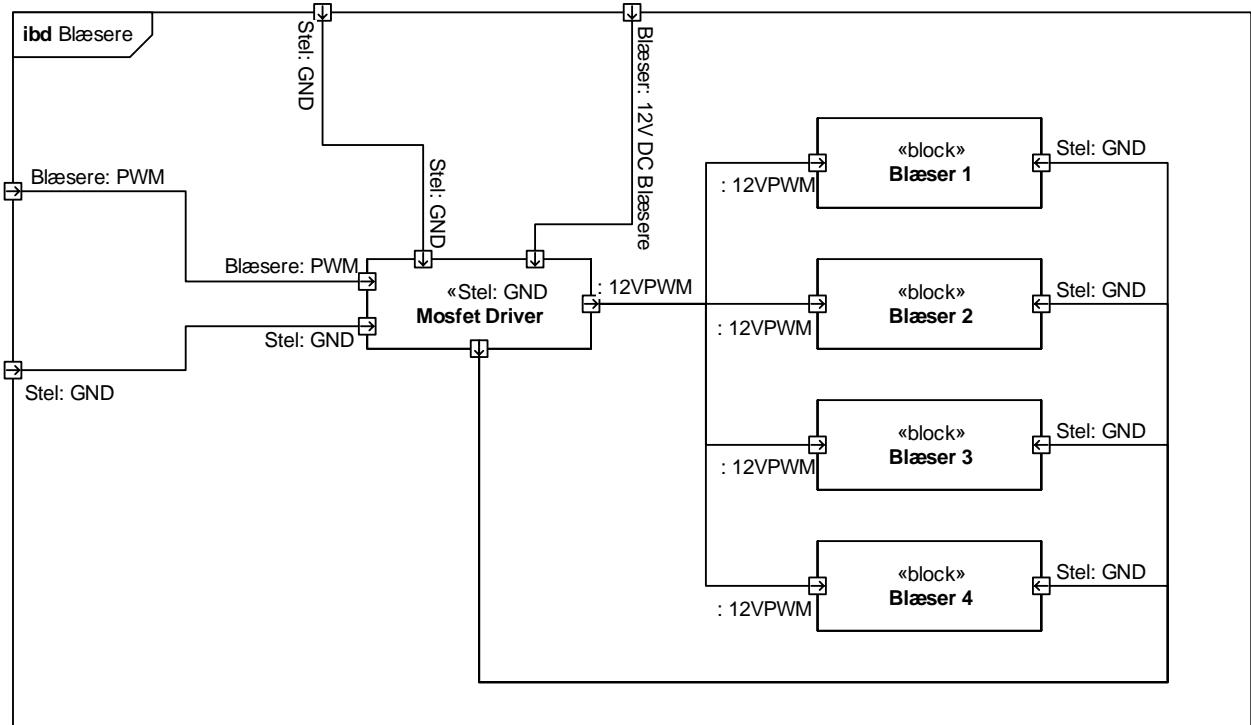
Såfremt det senere vælges at opgradere til PWM styring, skal man tage højde for - eller se bort fra - at effekten ikke er lineært sammenhængende med dtycyclen. Dette skyldes dels at effekt har en sammenhæng med kvadratet af strømmen, dels at modstanden i glødetråden afhænger af temperaturen.



Figur 20: Kredsløb for Mosfet Driver i underblokken Varmelegeme

Når Varme Bool på Figur 19 går høj, lukker mosfet transistoren, og tilslutter derved stel til USB Strømspareskinne; varmelegemet forsynes med 230V AC. Når Varme Bool går lav, åbner mosfet transistoren, og derved afbrydes stel til Strømspareskinne; varmelegemet forsynes ikke. D1 er indsat for at sikre transistoren mod peakspændinger fra USB skinne, når den slukkes. Dette er sandsynligvis ikke nødvendigt, men da vi ikke har indblik i hvordan USB strømspareskinne rent faktisk virker, er dioden indsat for en sikkerheds skyld. Modstanden R1 er en beskyttelsesmodstand, som beskytter PSoC4, hvis Mosfet transistoren brænnes af.

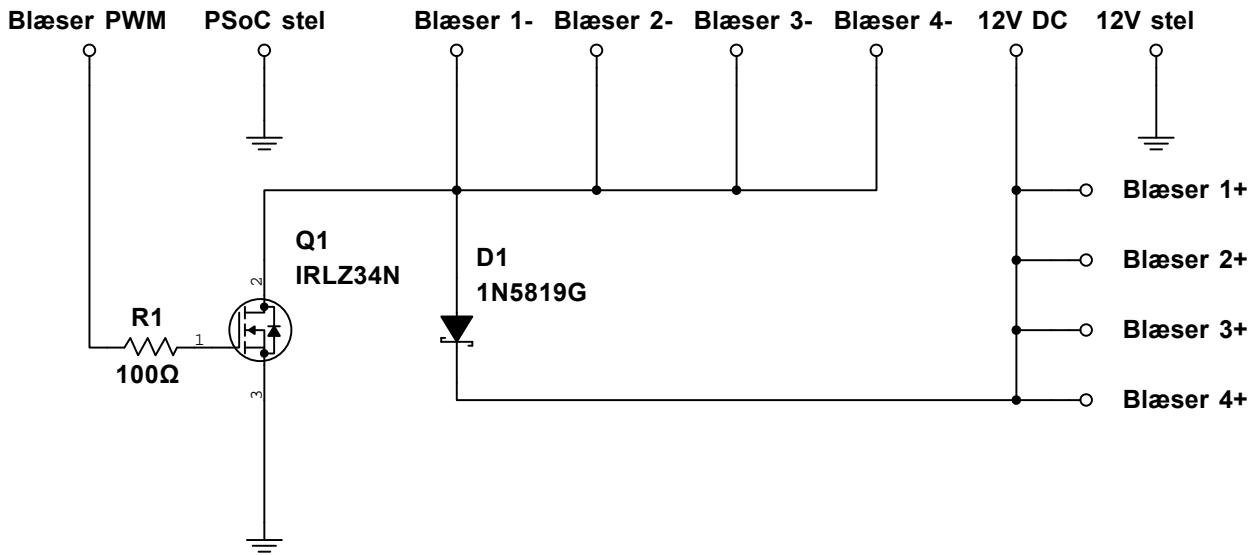
5.4.2 Blæsere



Figur 21: IBD for underblokken Blæsere i Aktuator

Figur 21 viser interne forbindelser i underblokken Blæsere, der består af en Mosfet Driver og fire 12V blæsere. To af blæserne er monteret således at luft blæses ind i drivhuset, mens de to øvrige blæsere blæser luft ud af drivhuset. Det forventes at en dutycycle på 100% udskifter al luft i drivhuset på meget kort tid; dutycyclen for 'tændte blæsere' bestemmes ved praktiske forsøg under realisering af underblokken.

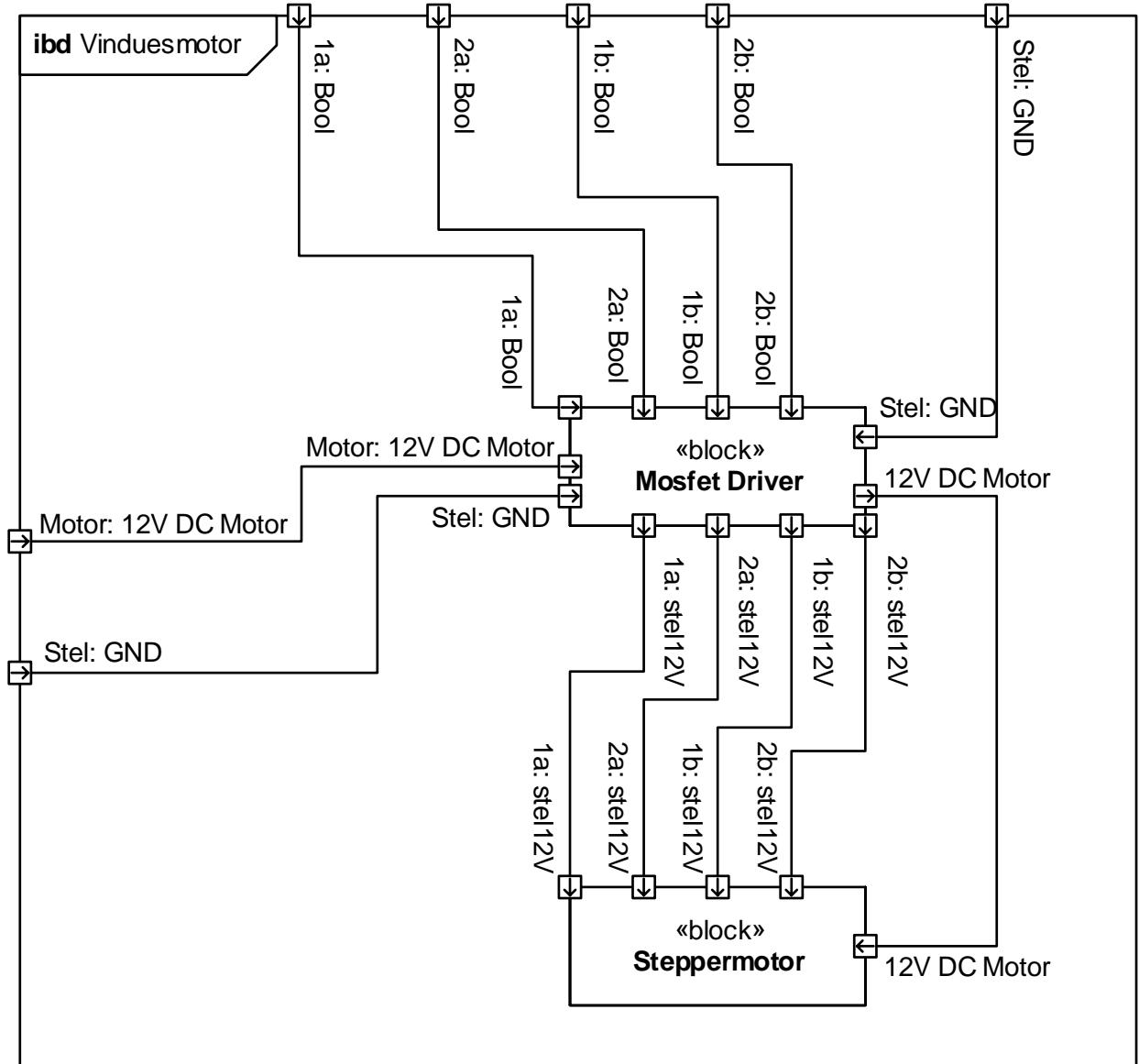
Ved praktiske forsøg, konstateredes det, at en dutycyce på 50% er et fornuftigt maximum. Det konstateredes desuden, at blæserne skal startes på maximum (dutycycle 50%) for at komme i gang. Hvis der startes med en mindre dutycycle, opnår motoren ikke inerti nok til at begynde dreje. Begge dele implementeres i SW.



Figur 22: Kredsløb for Mosfet Driver i underblokken Blæsere

Mosfet Driveren til Blæsere på Figur 22 fungerer i principippet på samme måde som Mosfet Driver for Varmelegeme (Figur 20). Der er blot tilsluttet fire blæsere, der alle styres vha. den samme transistor.

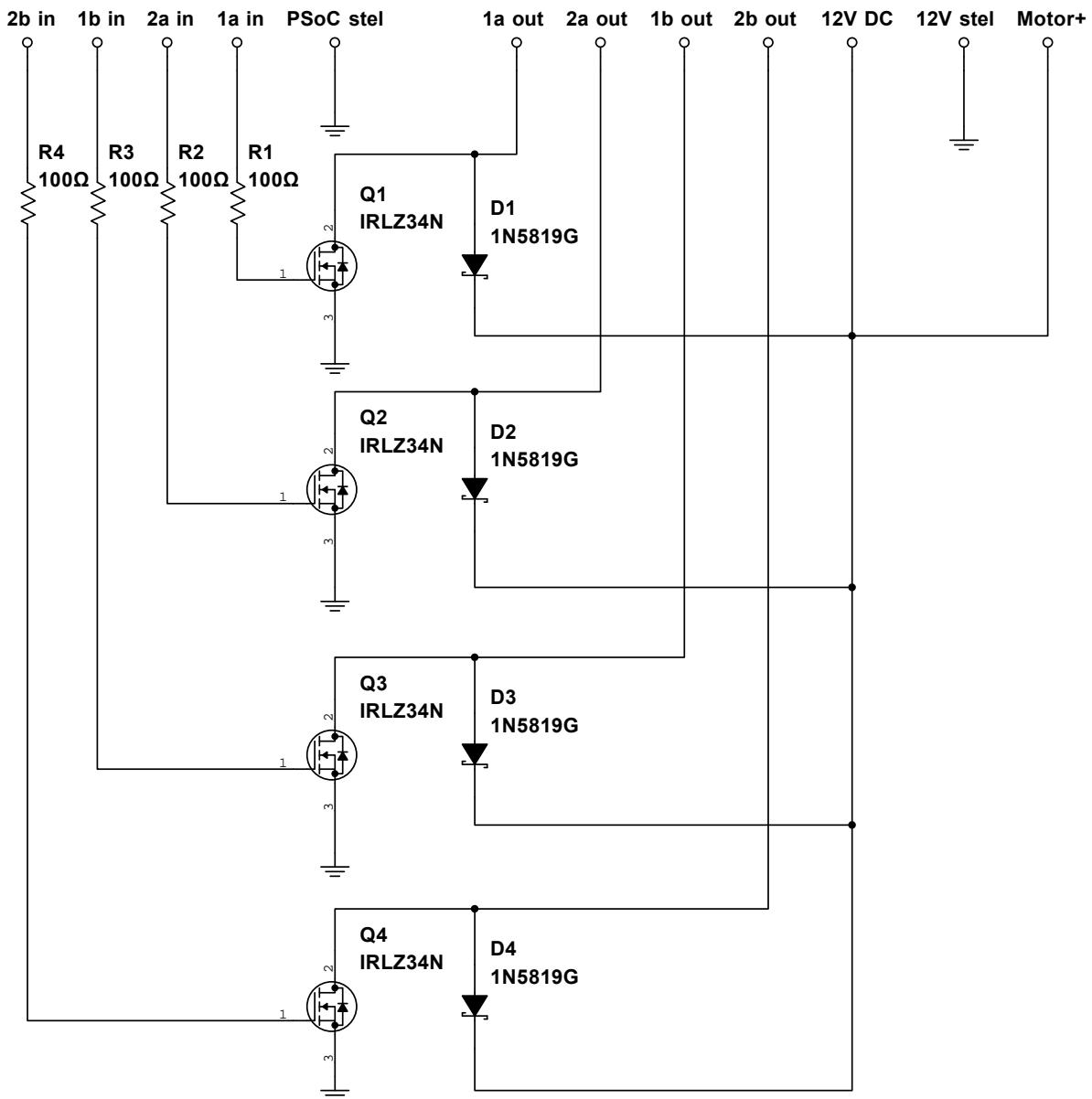
5.4.3 Vinduesmotor



Figur 23: IBD for underblokken Vinduesmotor i Aktuator

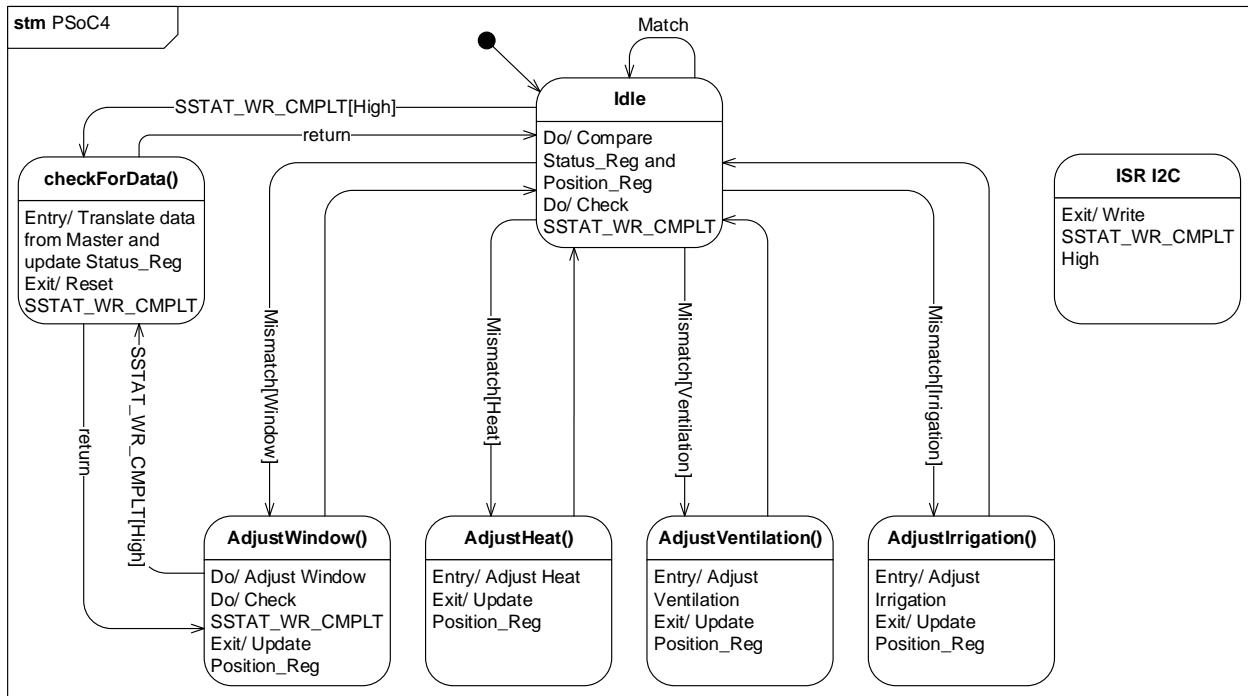
Ovenstående diagram (Figur 23) viser interne forbindelser i underblokken Vinduesmotor, der består af en Steppermotor og en Mosfetdriver. Der åbnes og lukkes for mosfet transistorer i Mosfet Driveren vha. 3,3V signaler fra PSoC4, og derved forsyneres Steppermotor med 12V DC.

Mosfet Driveren til Vinduesmotor på Figur 24 side 71 fungerer i principippet på samme måde som Mosfet Driver for Blæsere (Figur 22). Der er dog den forskel, at de fire signaler styres af hver deres transistor, da de ikke alle skal åbne og lukke samtidigt.



Figur 24: Kredsløb for Mosfet Driver i underblokken Vinduesmotor

5.4.4 PSoC4



Figur 25: State Machine for software på underblokken PSoC4 i Aktuator

Ovenstående diagram (Figur 25) viser en state machine for software i underblokken PSoC4 i blokken Aktuator.

Koden gentager tjek af om ønsket indstilling, af aktuatorer stemmer overens med aktuatorernes aktuelle indstilling og retter dette, såfremt det ikke er tilfældet. Denne rutine kan til enhver tid afbrydes af interrupt fra I2C bussen, der opdaterer slave write complete registeret (SSTAT_WR_CMPLT) fra lav til høj. Dette medfører at checkForData aktiveres som opdaterer ønskede indstillinger af aktuatorer. Herefter vil rutinen genoptages.

Ønskede indstillinger er gemt i registret Status_Reg, mens aktuelle indstillinger er gemt i Position_Reg.

Ved opdaterering af aktuatorer er den prioriterede rækkefølge: Varme, blæsere, vanding og vindue. Vinduet er sidst i rækkefølgen, da det tager lang tid at åbne eller lukke. Koden for åbning eller lukning af vindue skrives således, at slave write complete registeret løbende tjekkes. Hvis dette register er højt vil indstillinger af aktuatorer opdateres, og derefter fortsætte fra vinduets position med de nye indstillinger. Derved undgås det, at vinduet fx skal lukke helt, inden det åbnes, hvis disse to kommandoer modtages med meget kort mellemrum.

5.4.5 Drivers til PSoC4

Dette afsnit beskriver drivere for opdatering af aktuatorer. Disse drivere er opdelt i Varme, Blæsere, Vanding, Vinduesmotor og checkForData. Derved kan systemet nemt opdateres, hvis der ændres på styringen af en bestemt aktuator. Alle drivere består af en header fil med prototyper og en source fil med implementeringer.

Driver Varme

Denne driver indeholder en funktionalitet, der har til formål at tænde eller slukke varmelegemet, samt opdatere aktuatorens bits i Position_Reg. Dette sker ved at sætte en pin på PSoC4 hhv. høj eller lav; det gøres vha. PWM, da der så senere er nem mulighed for at opgradere styringen af varmelegemet til PWM styring.

Driver Blæsere

Driveren for Blæsere har til formål at starte eller stoppe de fire blæsere i drivhuset, samt opdatere aktuatorens bits i Position_Reg. Dette sker ved at sende et PWM signal ud på en pin på PSoC4.

Driver Vand

Denne driver har til formål at aktivere eller deaktivere aktuatorer for vanding, samt opdatere aktuatorens bits i Position_Reg. Dette sker ved at sætte 6 forskellige pins på PSoC4 hhv. høj eller lav.

Driver Vinduesmotor

Driveren for vinduesmotoren har til formål at åbne eller lukke vinduet, samt opdatere aktuatorens bits i Position_Reg. Antallet af omdrejninger for at åbne vinduet bestemmes under realisering ved praktiske forsøg. Driveren skal sammenligne Position_Reg med Status_Reg for hver omgang steppermotoren kører. Derved undgås det fx, at vinduet er nødt til at åbne helt inden det lukkes, hvis disse to kommandoer modtages hurtigt efter hinanden. Koden skrives således at det er nemt senere at opdatere driveren, så vinduet kan indstilles i flere trin mellem åbent og lukket.

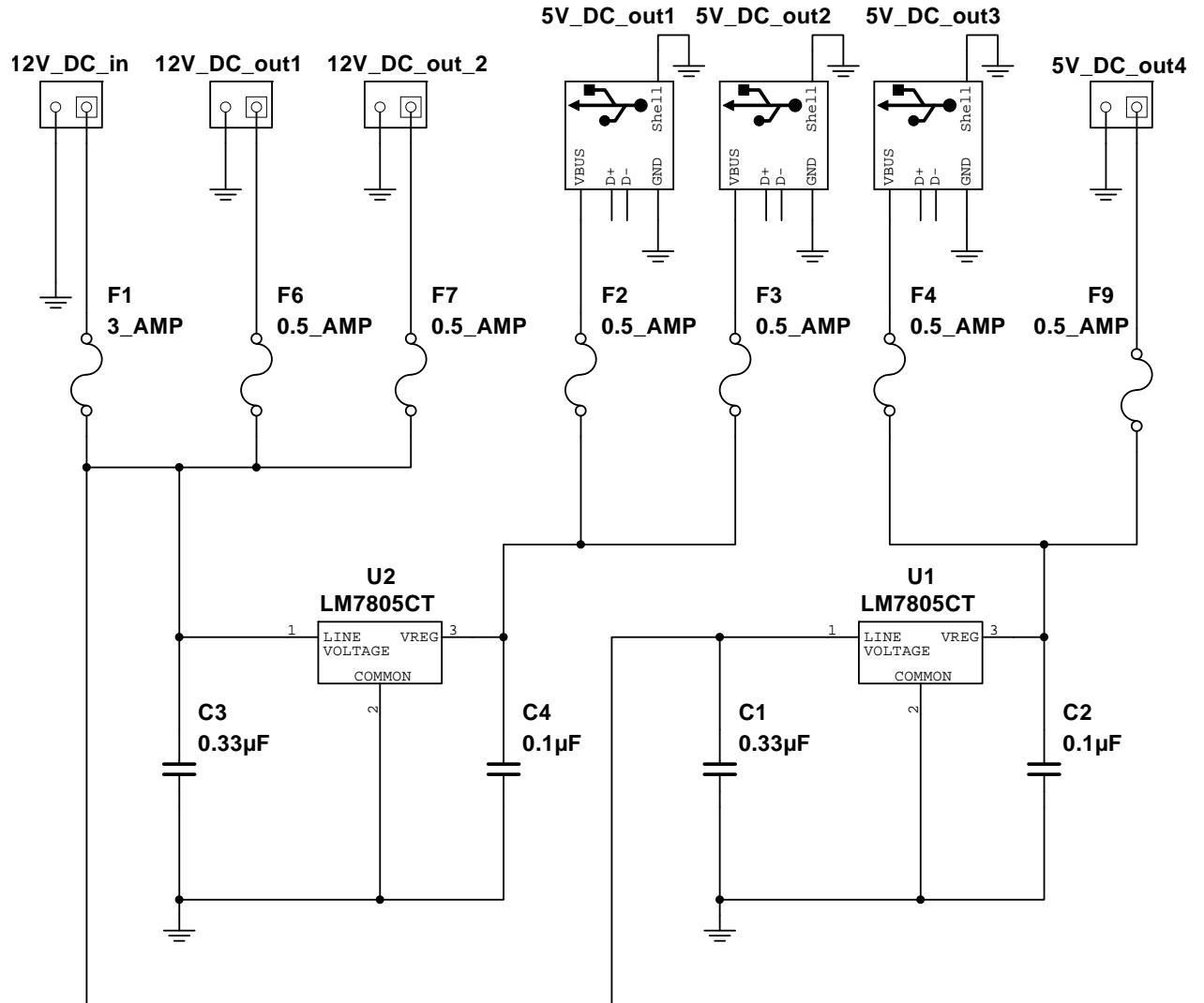
Driver checkForData

Denne driver har til formål at opdatere Status_Reg. Der er mulighed for at kalde denne fra Idle tilstand og under AdjustWindow. Driveren kaldes kun i det tilfælde, at I2C bussen har kaldt et interrupt, hvilket medfører opdatering af slave write complete registeret fra lav til høj.

5.5 Strømforsyning Design (Henrik og Morten)

Som nævnt i systemarkitekturen forsyner strømforsyningen øvrig HW i systemet, undtagen selve varmelegemet og DevKit8000, der forsynes med 230V AC, og sensorerne, der forsynes med VEE (3.3V DC) fra PSoC Master.

Strømforsyningen forsyner med 12V DC max. 3A fra en laboratorieforsyning jf. Signalbeskrivelser på Tabel 23 på side 39. Alternativt kan anvendes en 12V transformør, der kobles til 230V AC. Strømforsyningen skal have 12V DC udgange til motor og blæsere, USB udgange med VDD til PSoC4 Pioneer Kits og en VDD udgang til USB strømspareskinne.



Figur 26: Diagram for blokken Strømforsyning

Figur 26 viser Multisim diagram for designet af blokken Stroemforsyning. De enkelte komponenter og overvejelser herom gennemgås nedfor.

12V DC in (VCC) trækker som sagt max. 3A, derfor monteres denne med en sikring på denne størrelse.

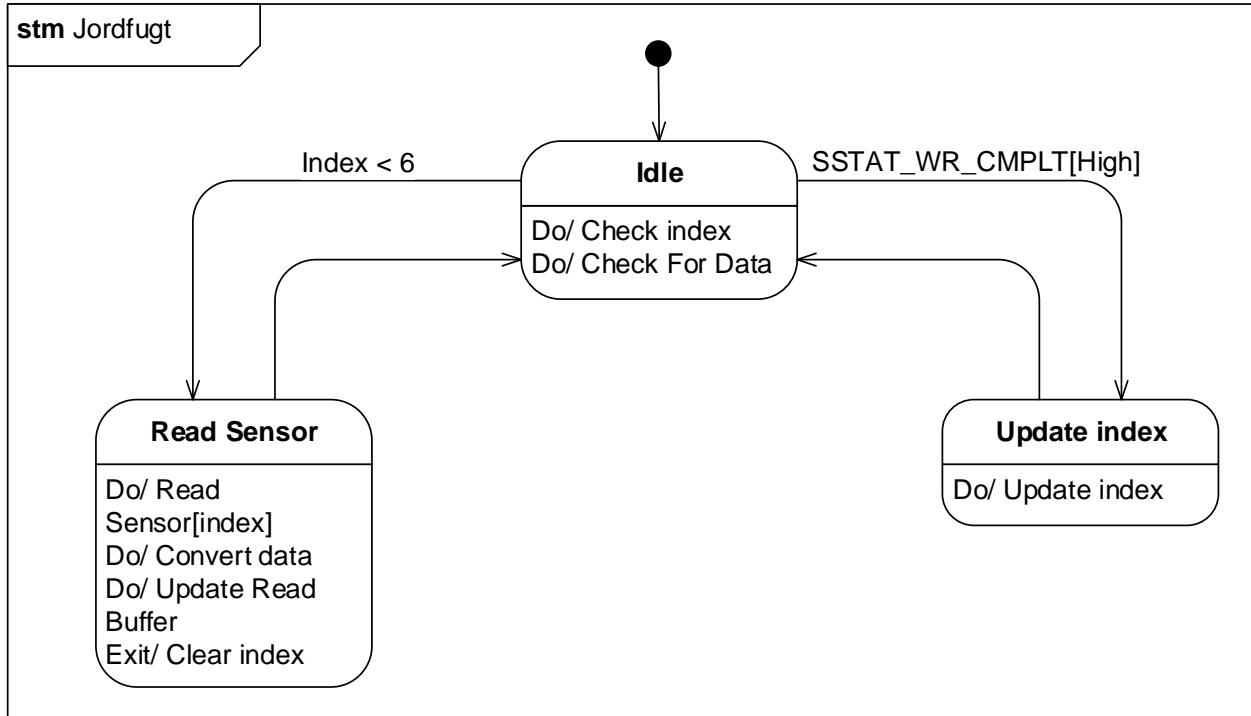
12V DC out1 udgangen til motor trækker max. 500 mA jf. databladet [9] side 38, derfor monteres den med en sikring på 500 mA.

De fire blæsere trækker hver især max. 140 mA ved fuld styrke jf. påtrykt værdi på selve blæserne. Implementeringen af koden i blokken Aktuator er lavet således, at blæserne maximalt kommer til at køre med en dutycycle på 50%. Derfor monteres ligeledes en sikring på 500 mA til de fire blæsere. Ved en praktisk undersøgelse af USB skinnen konstateredes det, at USB indgangen trækker ca. 400 mA, når relæet er slæt til, derfor monteres der også en 500 mA sikring på denne udgang.

Der anvendes to spændingregulatorer LM7805, som begrænser 12V DC til 5V DC. De kan hver især leve 1A, derfor anvendes to stk. De monteres med afkoblinger til stel på ben 1 og 3 jf. standardapplikationen på side 1 i databladet [7].

5.6 Jordfugt Design (Henrik og Morten)

Dette afsnit omhandler design af blokken Jordfugt, der består af et PSoC4 Pioneer kit og 0-6 jordfugtsensorer.



Figur 27: State Machine for SW på PSoC4 i blokken Jordfugt

Ovenstående figur (Figur 27) viser en state machine for SW i PSoC4 i blokken Jordfugt. PSoC'en står hele tiden og poller på om der er modtaget data, og om en indexvariabel er mindre end 6, som er det maximale antal jordfugtsensorer, der kan tilkobles.

Såfremt der er modtaget data via I²C komminikationen, opdateres index variablen til det sensor-nummer, der er modtaget.

Såfremt index variablen er mindre end 6, læses der på den tilhørende sensor, data konverteres til et tal mellem 1 - 100, og read bufferen opdateres med den læste værdi.

Der er i forbindelse med brugen af sensoren lavet en støjundersøgelse i faget Mixed Signal Elektronik. Se journalen [11] for nærmere info. I AutoGreen anvendes jordfugtsensoren på en noget simplere måde end i øvelsen, se afsnittet om implementering af blokken Jordfugt for nærmere info.

6 Software Design

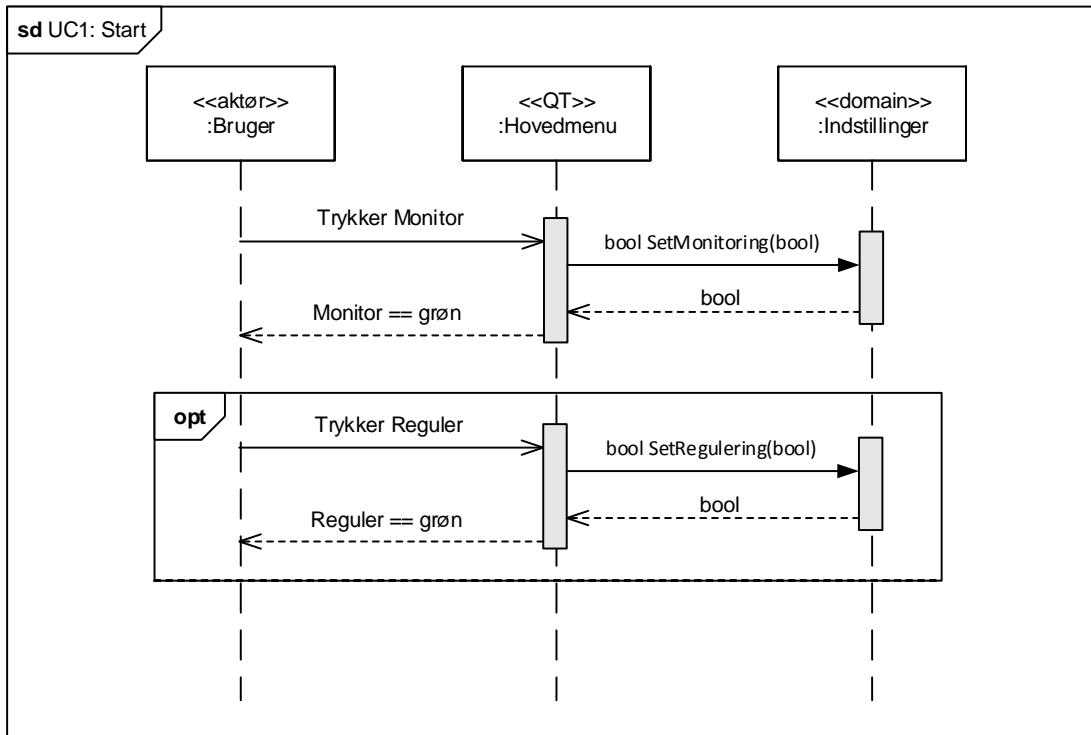
6.1 Version

Dato	Version	Initialer	Ændring
29. marts	1	KS	Første udkast.

6.2 Indledning

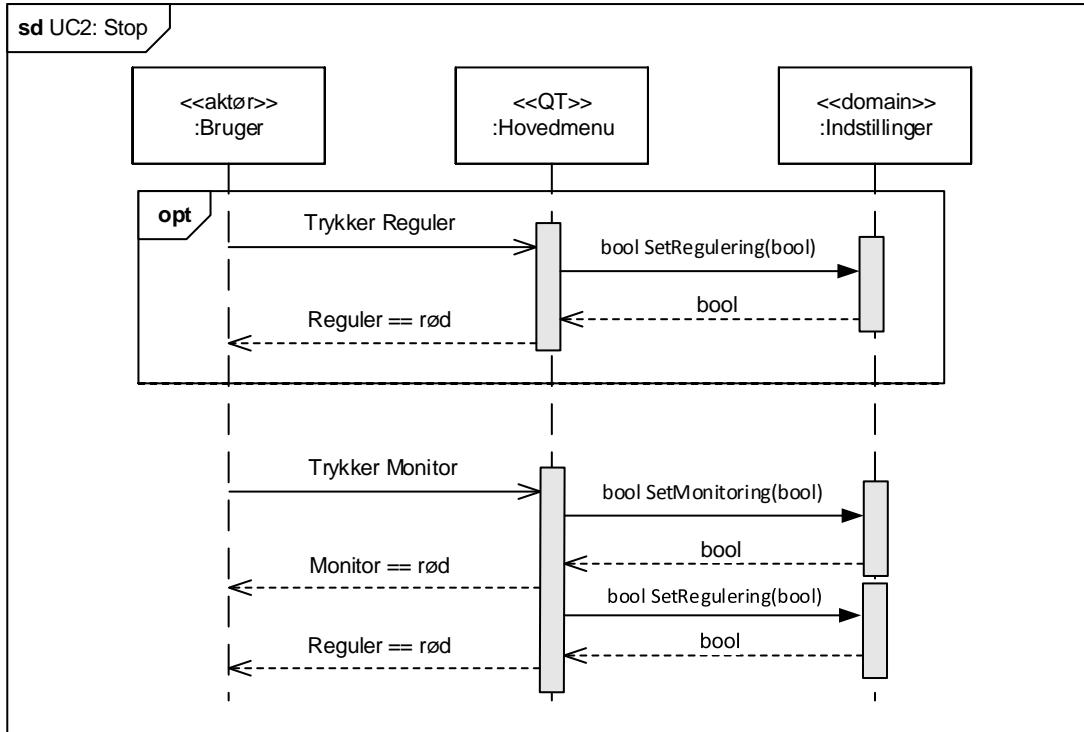
6.3 Udvidet Applikationsmodel: Sekvensdiagrammer

6.3.1 Usecase 1: Start



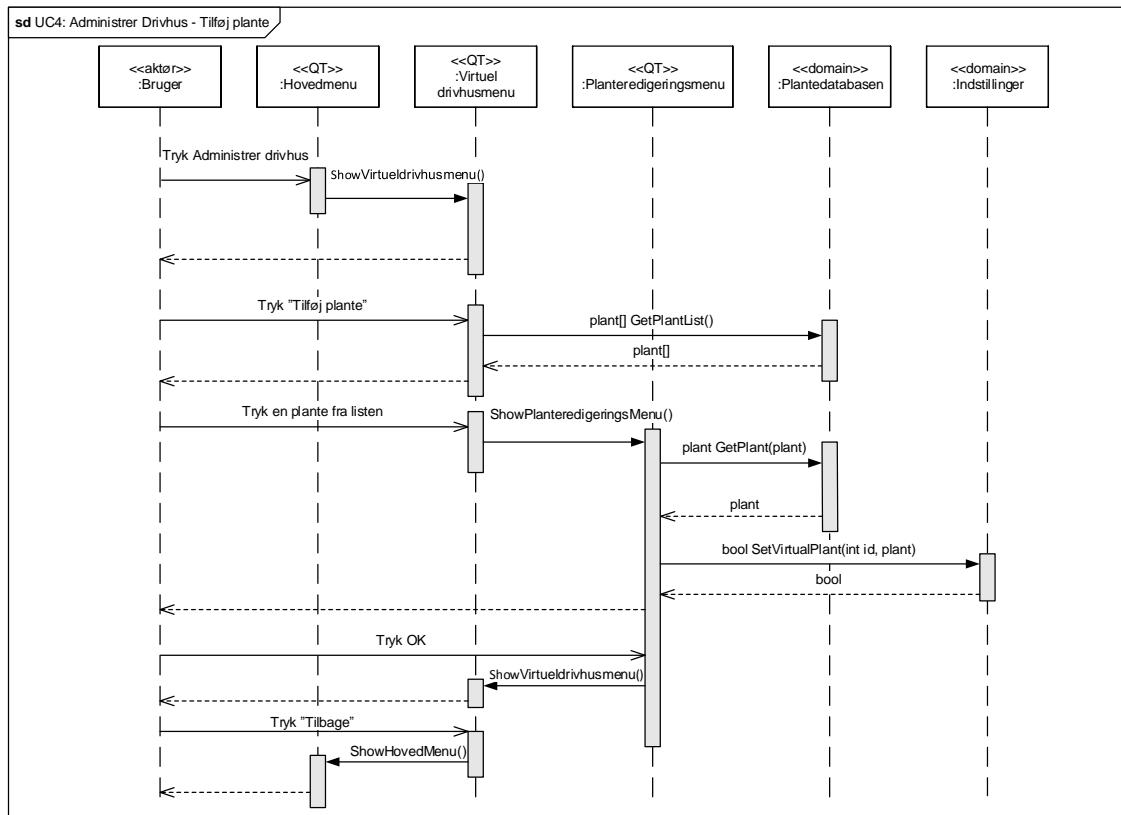
Figur 28: Application model for AutoGreen

6.3.2 Usecase 2: Stop

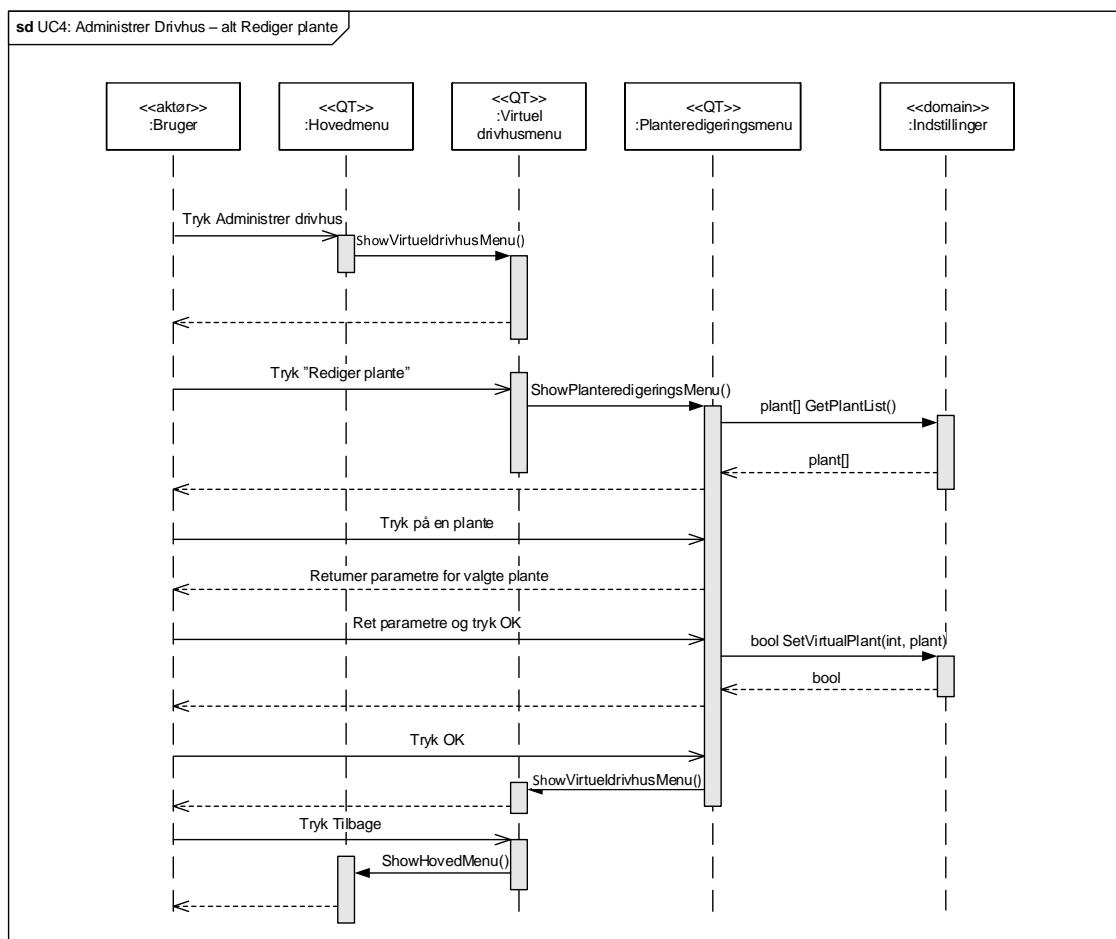


Figur 29: Application model for AutoGreen

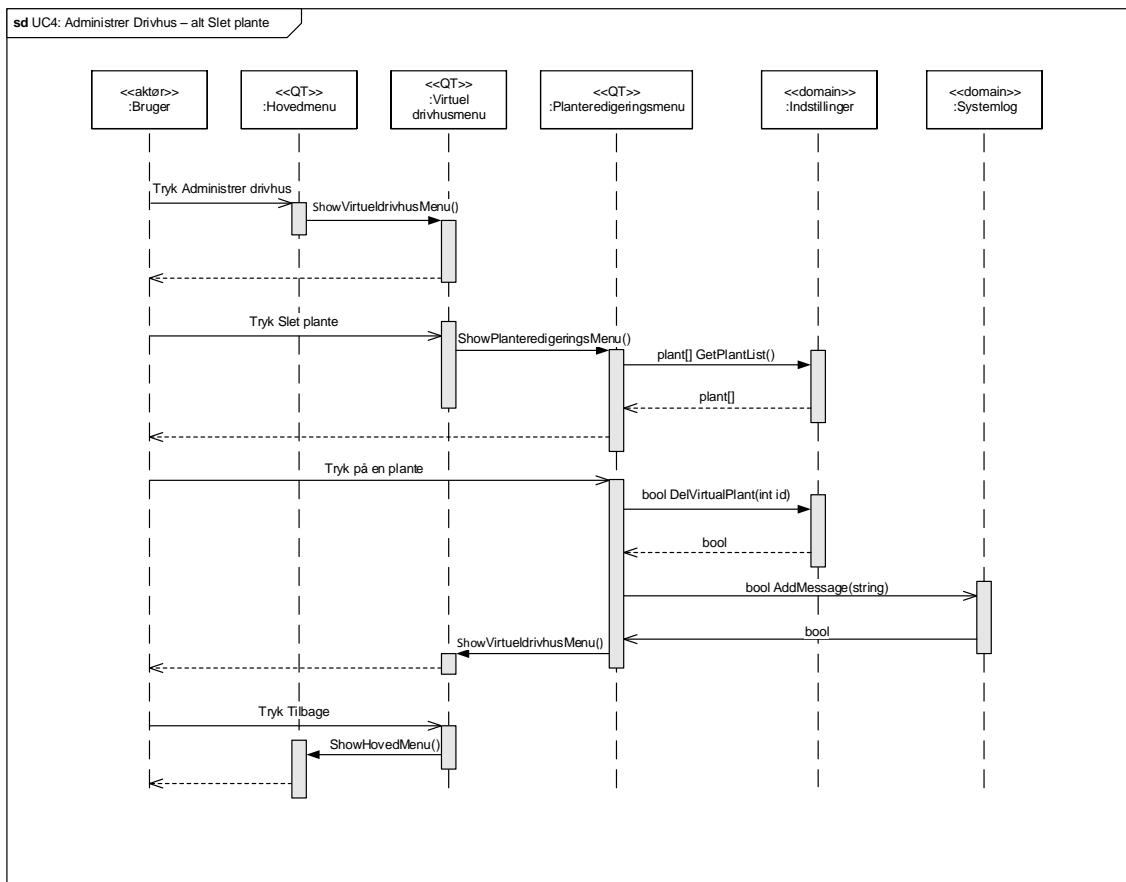
6.3.3 Usecase 4: Administrer Drivhus



Figur 30: Application model for AutoGreen

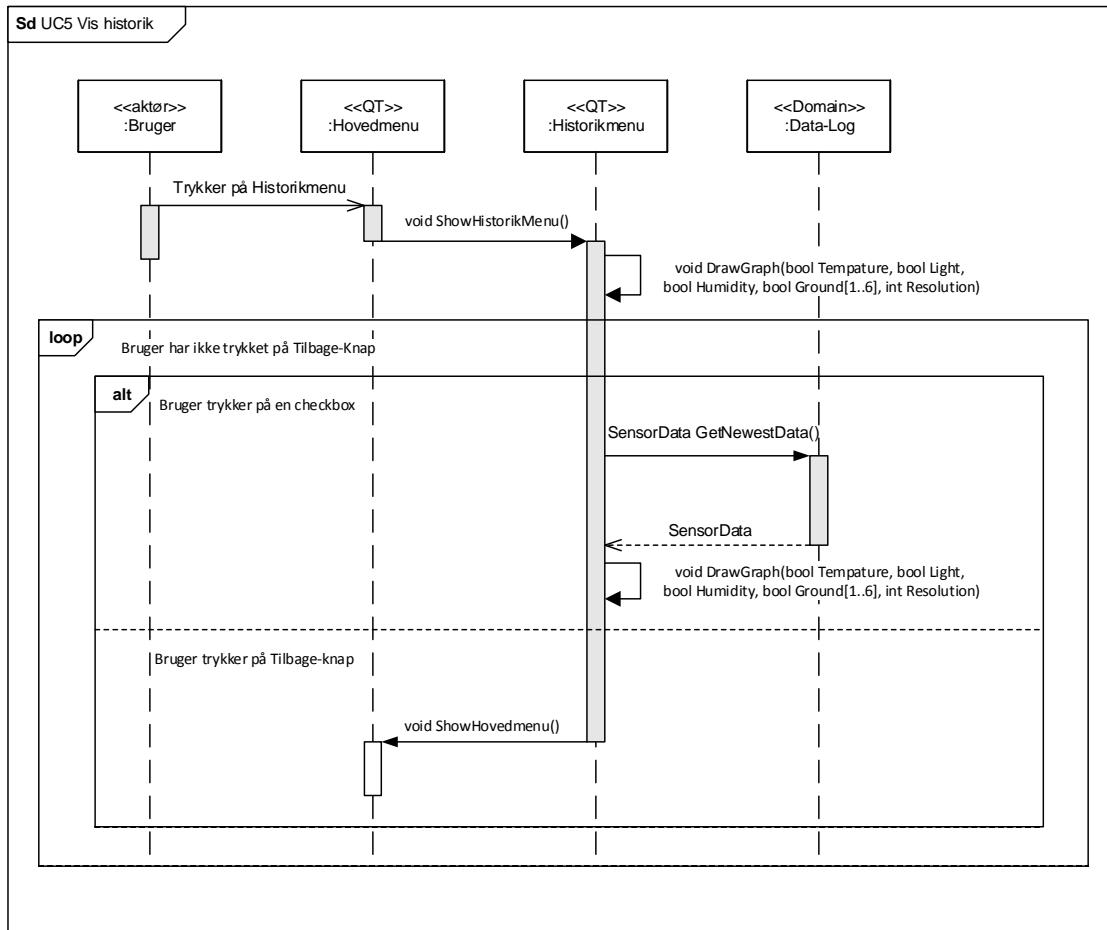


Figur 31: Application model for AutoGreen



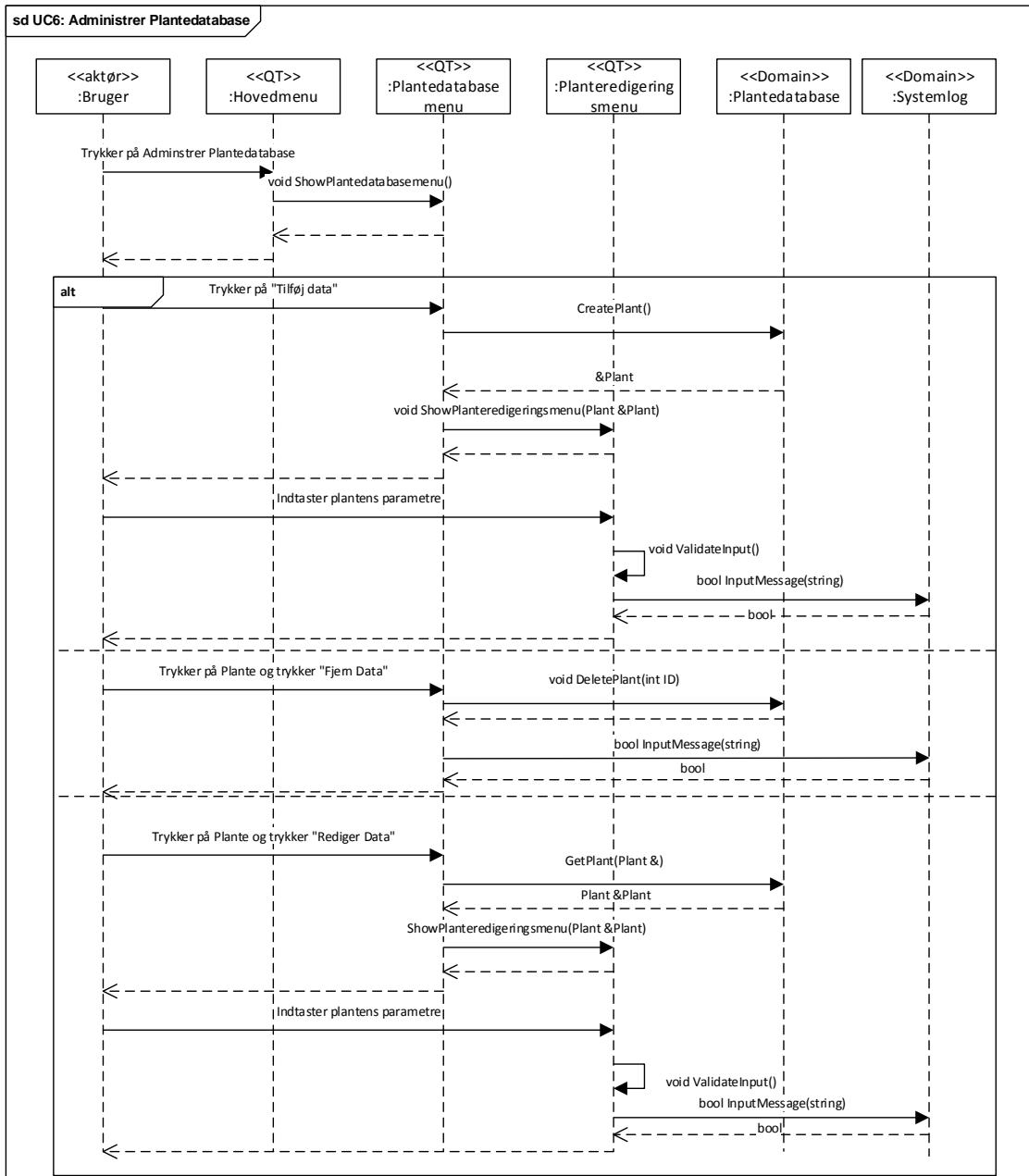
Figur 32: Application model for AutoGreen

6.3.4 Usecase 5: Vis Historik



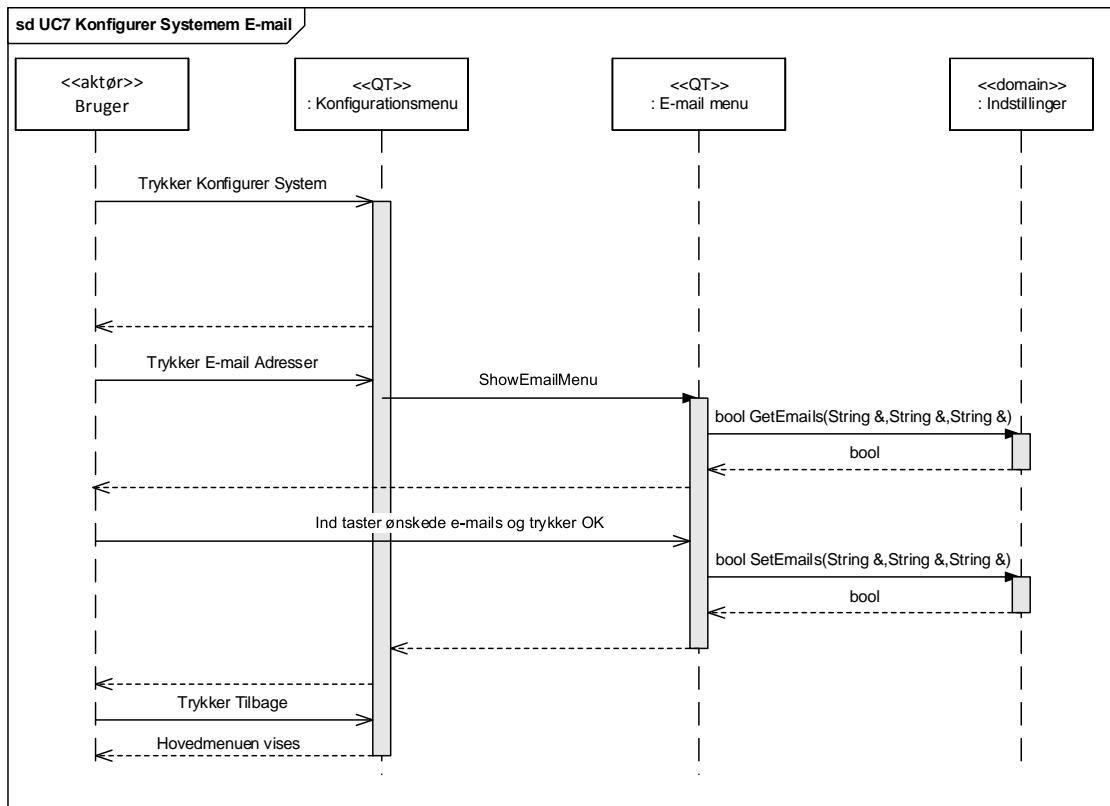
Figur 33: Application model for AutoGreen

6.3.5 Usecase 6: Adminstrer Plantedatabase

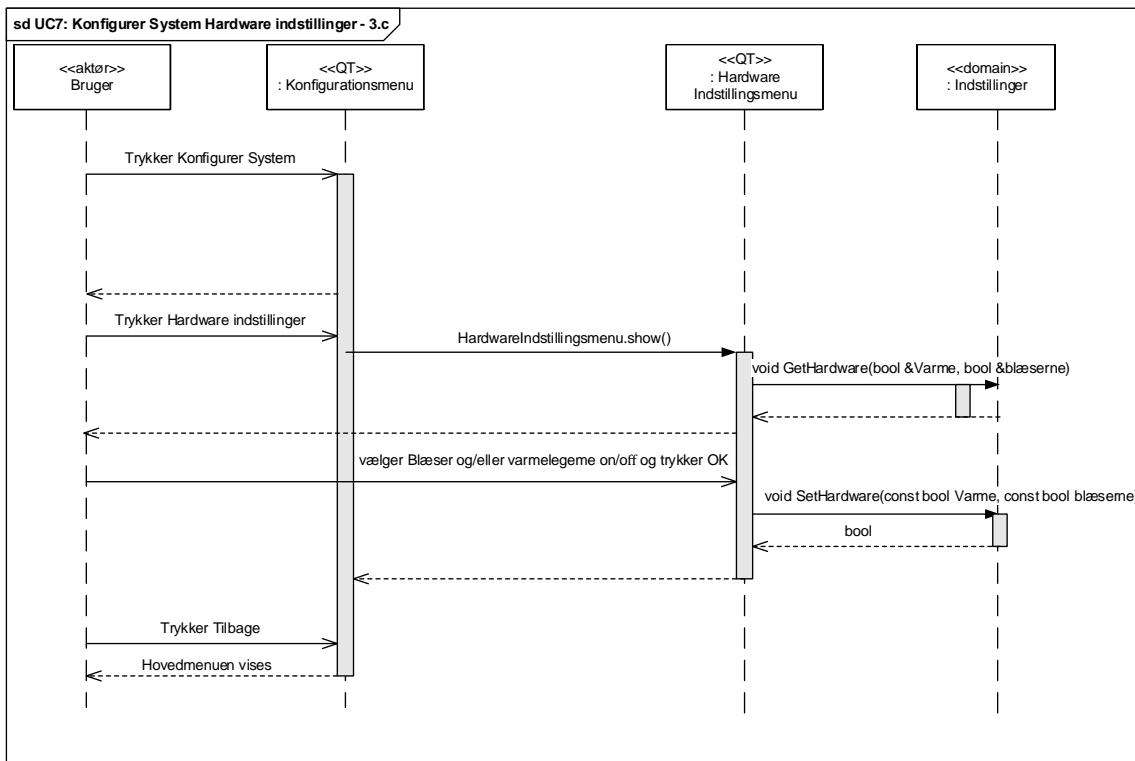


Figur 34: Application model for AutoGreen

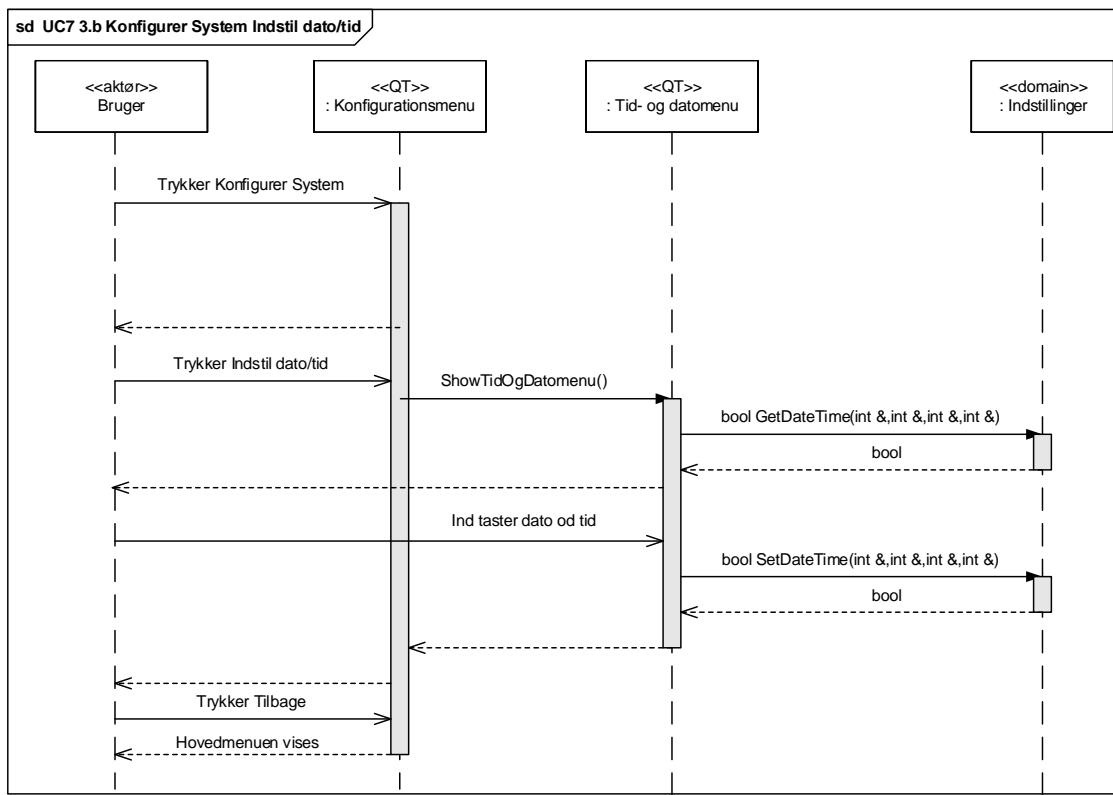
6.3.6 Usecase 7: Konfigurer System



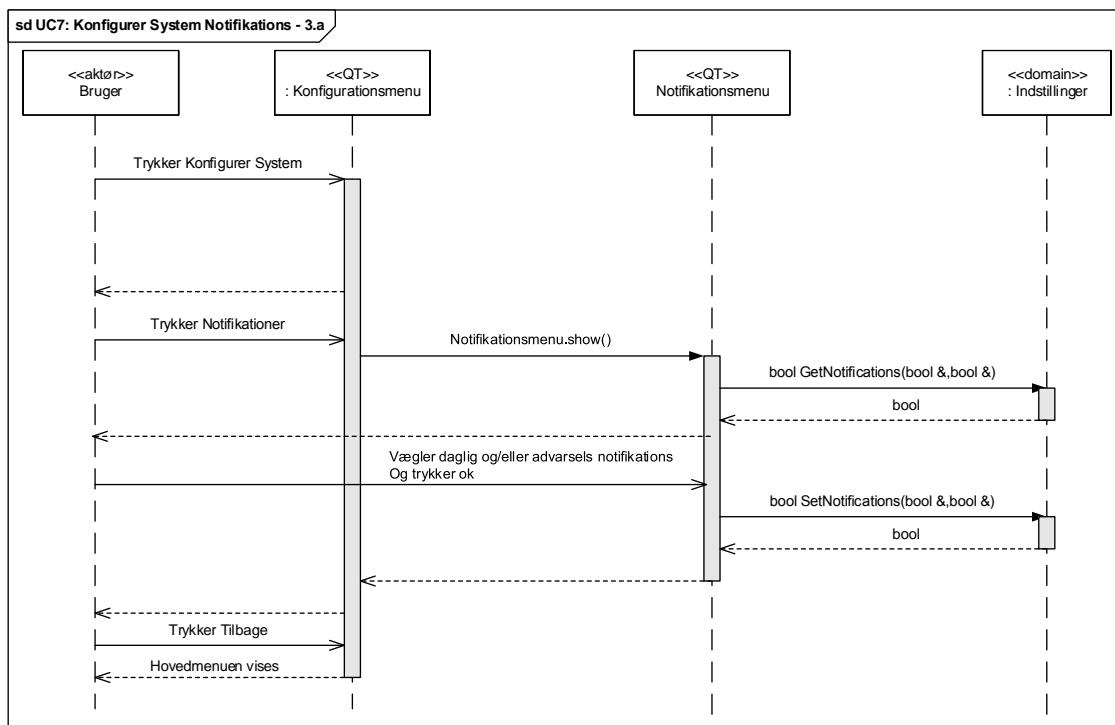
Figur 35: Application model for AutoGreen



Figur 36: Application model for AutoGreen

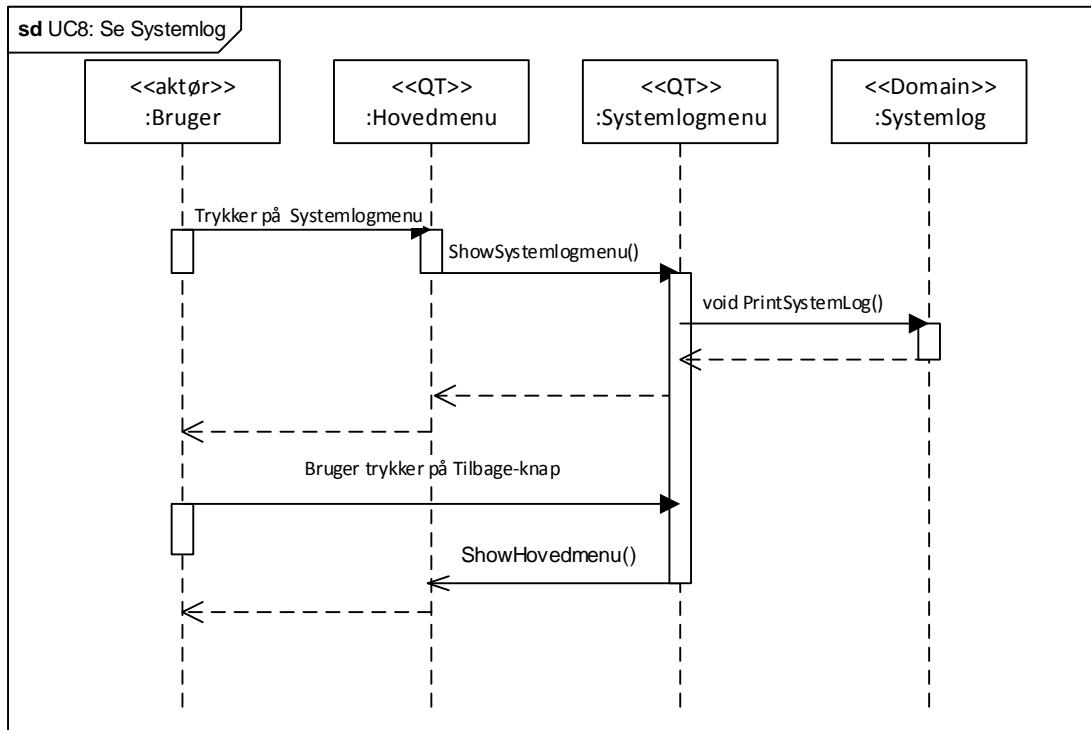


Figur 37: Application model for AutoGreen



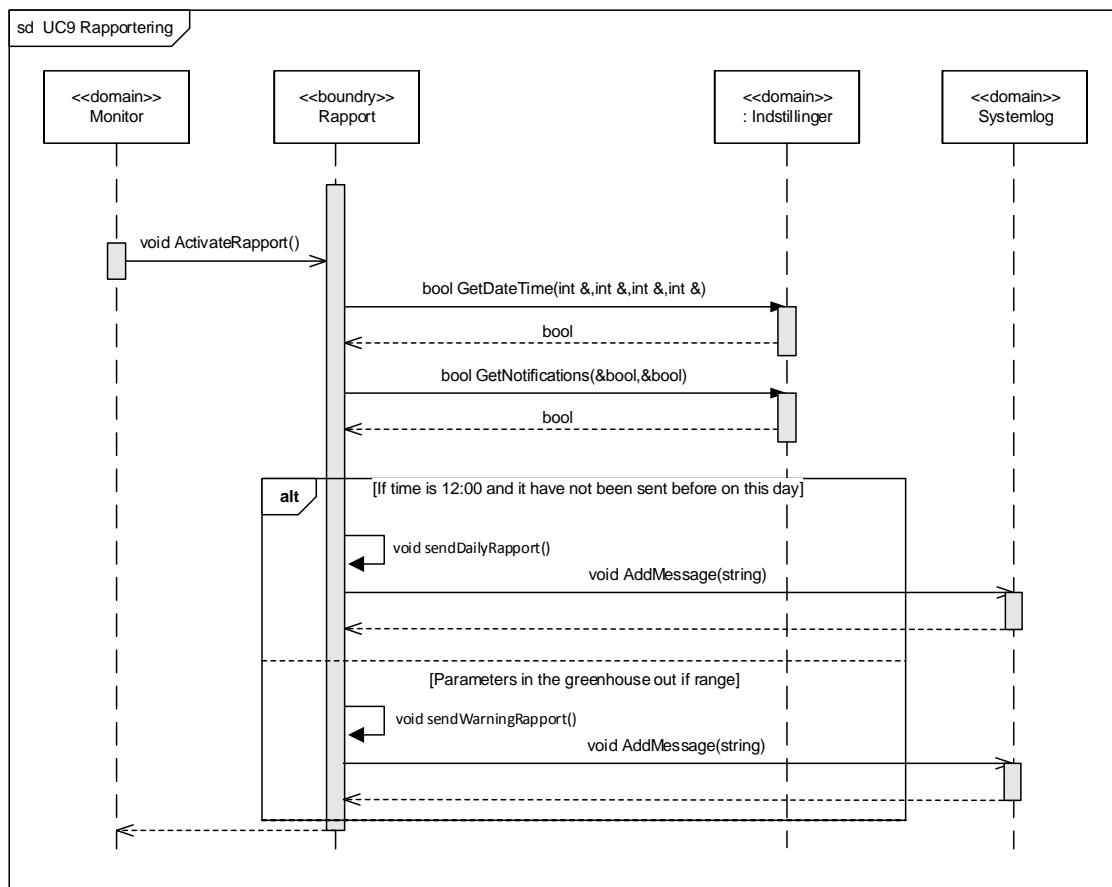
Figur 38: Application model for AutoGreen

6.3.7 Usecase 8: Se Systemlog



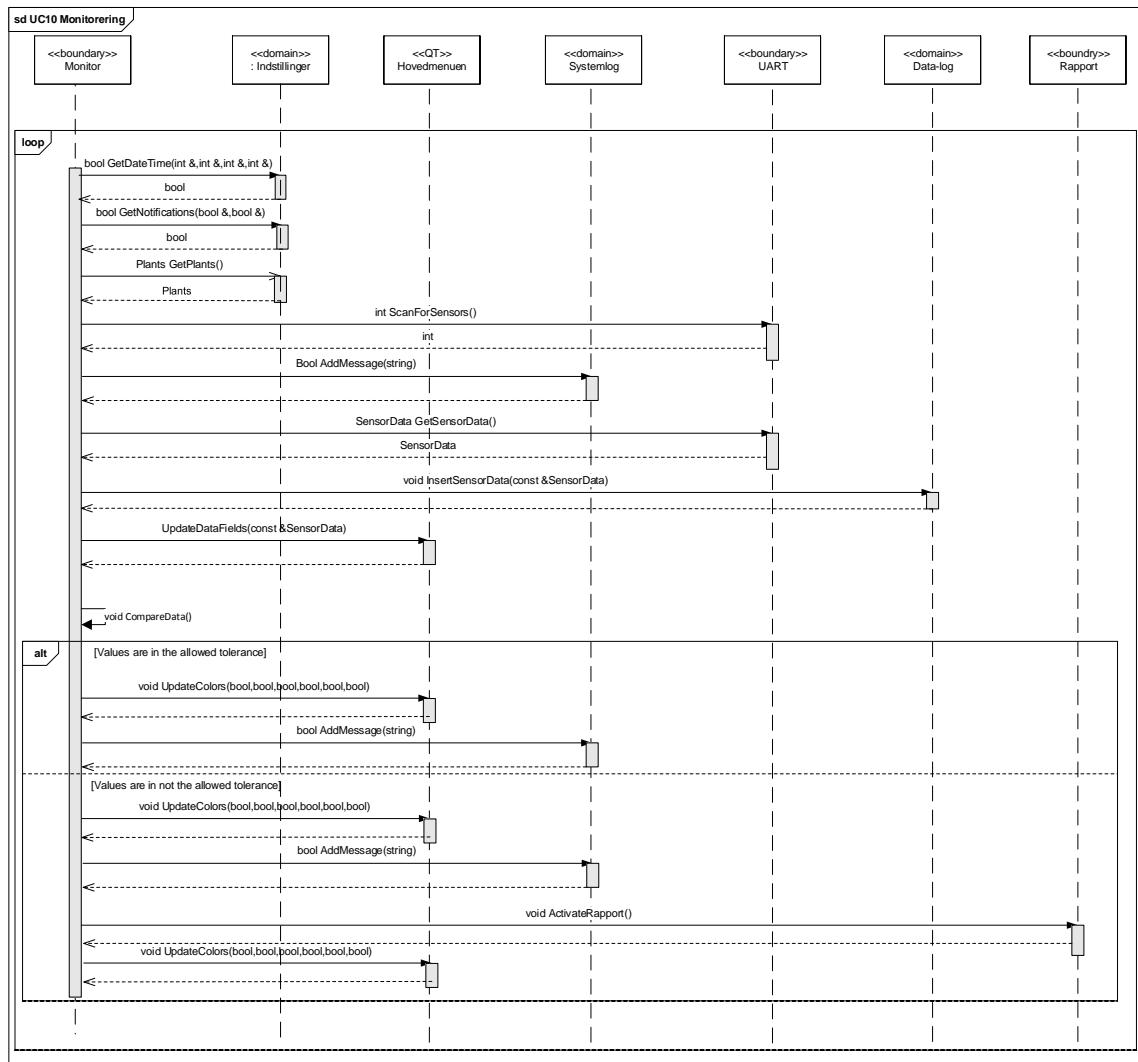
Figur 39: Application model for AutoGreen

6.3.8 Usecase 9: Rapportering



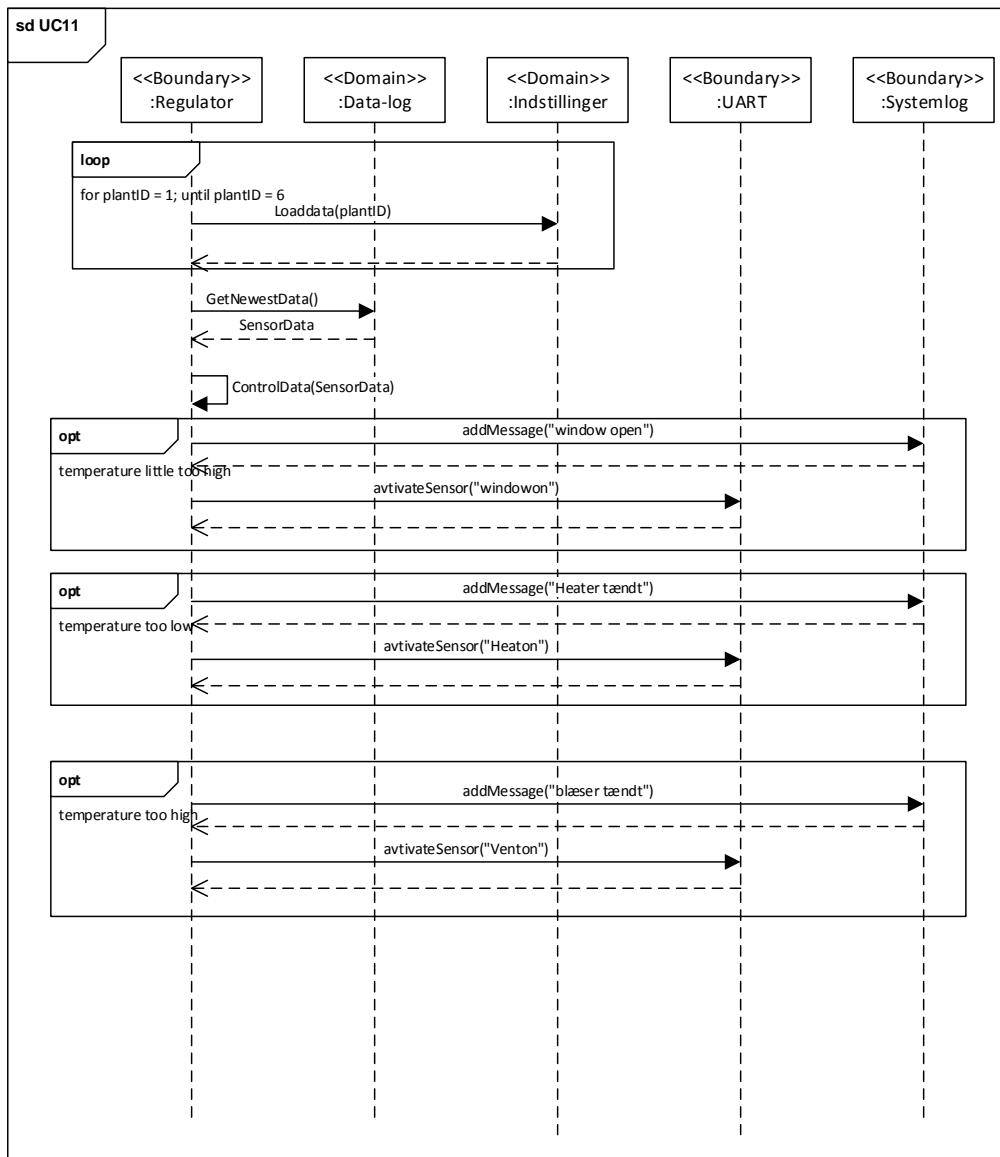
Figur 40: Application model for AutoGreen

6.3.9 Usecase 10: Monitorering



Figur 41: Application model for AutoGreen

6.3.10 Usecase 11: Regulering



Figur 42: Application model for AutoGreen

6.4 Klassebeskrivelser

6.4.1 Domainklasse Datalog

Attributter

DatalogList	datalog	datalogList er en nedarvning af doublylinkedlist, med extra variabler til lagring af tid, temperatur, lysintensitet, luftfugtighed, og 6 jordfugtigheder.
-------------	---------	---

Tabel 71: Attributter for klassen Datalog

Metoder

Prototype	<code>void GetData(int time, int temp, int light, int humidity, int ground)</code>
Parametre	<p><code>int time</code> time er en reference til et array som indeholder tidsstemplere over en hvil periode.</p> <p><code>int temp</code> temp er en reference til et array som indeholder temperaturen på de angivne tidstemplere.</p> <p><code>int light</code> light er en reference til et array som indeholder lysintensiteten på de angivne tidstemplere.</p> <p><code>int humidity</code> ground er en reference til et 2D array, hvor hver kolonne indeholder jordfugtigheden for den givne plante. Hver række indeholder jordfugtigheden på de angivne tidstemplere.</p> <p><code>int ground</code> temp er en reference til et array som indeholder temperaturen på de angivne tidstemplere.</p>
Returværdi	-
Beskrivelse	Metoden har til fordel at hente data ud i et angivet tidsområde, ved at indsætte disse data ind i referencerne, hvorefter metoden afsluttes.

Tabel 72: GetData

Prototype	<code>void InsertSensorData(const SensorData SensorData)</code>
Parametre	<p><code>int time</code> time er en reference til et array som indeholder tidsstemplere over en hvil periode.</p>
Returværdi	-
Beskrivelse	Når metoden kaldes oprettes en ny Node i den linked list hvor alt dataen fra parameteren SensorData bliver lagret i.

Tabel 73: InsertSensorData

Prototype	<code>void GetNewestData(int temp, int humidity, int plant)</code>
Parametre	<p><code>int temp</code> temp er en reference til en int, som indeholder den temperatur fra den nyeste node i linked listen.</p> <p><code>int humidity</code> humidity er en reference til en int, som indeholder luftfugtigheden fra den nyeste node i linked listen.</p> <p><code>int plant</code> ground er en reference til et int array, som indeholde jordfugtigheden for planterne fra den nyeste node i linked listen</p>
Returværdi	-
Beskrivelse	Metoden går ind i link listen fra nyeste data og går tilbage indtil at tiden passer med 24 timer før den nuværende tid, herefter tages data, 24 timer længere tilbage, og regnes sammen til en gennemsnitlig temperatur, luftfugtighed, lysintensitet og for op til 6 jordfugtigheder. De data der udtages af link listen slettes, og en ny Node oprettes på den først udtages plads.

Tabel 74: GetNewestData

Prototype	<code>void Sortweek()</code>
Parametre	-
Returværdi	-
Beskrivelse	Metoden går ind i link listen fra nyeste data og går tilbage indtil at tiden passer med 2 dage før den nuværende tid, herefter tages data, 7 dage længere tilbage, og regnes sammen til en gennemsnitlig temperatur, luftfugtighed, lysintensitet og for op til 6 jordfugtigheder. De data der udtages af link listen slettes, og en ny Node oprettes på den først udtages plads.

Tabel 75: Sortweek

Prototype	<code>void Sortmonth()</code>
Parametre	-
Returværdi	-
Beskrivelse	Metoden går ind i link listen fra nyeste data og går tilbage indtil at tiden passer med 8 dage før den nuværende tid, herefter tages data, 30 dage længere tilbage, og regnes sammen til en gennemsnitlig temperatur, luftfugtighed, lysintensitet og for op til 6 jordfugtigheder. De data der udtages af link listen slettes, og en ny Node oprettes på den først udtages plads.

Tabel 76: Sortmonth

6.4.2 Domäneklasse Indstillinger

Private Attributter

<code>virtuellePlants</code>	<code>Plant[6]</code>	Indholder 6 structs af typen Plant
<code>Email</code>	<code>String[3]</code>	Indholder 3 e-mails
<code>Warning</code>	<code>bool</code>	En bool som fortæller om advarsels e-mails er slæt til eller fra.
<code>daily</code>	<code>bool</code>	En bool som fortæller om daglige e-mails er slæt til eller fra.
<code>Varmelegeme</code>	<code>bool</code>	En bool som fortæller om Varmelegemet er slæt til eller fra.
<code>blæserne</code>	<code>bool</code>	En bool som fortæller om blæserne er slæt til eller fra.
<code>monitor</code>	<code>bool</code>	En bool som fortæller om monitor er slæt til eller fra.
<code>regulering</code>	<code>bool</code>	En bool som fortæller om regulering er slæt til eller fra.

Tabel 77: Attributter for klassen Indstillinge

Private metoder

Prototype	<code>string exec(const char* cmd)</code>
Parametre	en string med den shell kommando der skal eksekveres på systemet.
Returværdi	<code>Plant</code> En string som indeholder outputtet fra den kørende shell kommando
Beskrivelse	Kan køre kommandoer på et Linux system via en pipe til systemet.

Tabel 78: exec

Public Metoder

Prototype	<code>void SetMonitorering(bool active)</code>
Parametre	<code>bool active</code> True hvis den skal køre ellers false.
Returværdi	-
Beskrivelse	Anvendes til sætte monitorering til at kører eller ikke kører.

Tabel 79: SetMonitorering

Prototype	<code>void SetRegulering(bool active)</code>
Parametre	<code>bool active</code> True hvis den skal køre ellers false.
Returværdi	-
Beskrivelse	Anvendes til sætte regulering til at kører eller ikke kører.

Tabel 80: SetRegulering

Prototype	<code>bool getRegulering()</code>
Parametre	-
Returværdi	True hvis den skal kører ellers false.
Beskrivelse	Anvendes tjekke status på regulering, så man kan se om den kører eller ej.

Tabel 81: getRegulering

Prototype	<code>bool getMonitorering()</code>
Parametre	-
Returværdi	True hvis den skal kører ellers false.
Beskrivelse	Anvendes tjekke status på monitorering, så man kan se om den kører eller ej.

Tabel 82: getMonitorering

Prototype	<code>void SetVirtualPlant(int id, Plant plantToPlace)</code>
Parametre	<code>bool active</code> et id som er mellem 1 og 6 som svare til hvor planeten er i det Virtual drivhus. <code>Plant plantToPlace</code> Her er den plante som skal sættes ind i det virtual drivhus.
Returværdi	True hvis den skal kører ellers false.
Beskrivelse	Bruges til at indsætte en plante i det virtual drivhus.

Tabel 83: SetVirtualPlant

Prototype	<code>void DelVirtualPlant(int id)</code>
Parametre	<code>int id</code> et id som er mellem 1 og 6 som svare til hvor planeten er i det Virtual drivhus.
Returværdi	True hvis den skal kører ellers false.
Beskrivelse	Bruges til at slette en virtual plante i det virtual drivhus.

Tabel 84: DelVirtualPlant

Prototype	<code>void GetEmails(string &mail1, string &mail2, string &mail3)</code>
Parametre	<code>string &mail1</code> E-mail et som der kan sendes notifications til. <code>string &mail2</code> E-mail to som der kan sendes notifications til. <code>string &mail3</code> E-mail tre som der kan sendes notifications til.
Returværdi	-
Beskrivelse	Bruges til at hente de tre e-mail adresser som systemet har gemt.

Tabel 85: GetEmails

Prototype	<code>void SetEmails(string &mail1, string &mail2, string &mail3)</code>
Parametre	<code>string &mail1</code> E-mail et som der kan sendes notifications til. <code>string &mail2</code> E-mail to som der kan sendes notifications til. <code>string &mail3</code> E-mail tre som der kan sendes notifications til.
Returværdi	-
Beskrivelse	Bruges til at sætte de tre e-mail adresser som systemet har gemt.

Tabel 86: SetEmails

Prototype	<code>void GetHardware(bool &Varmelegeme, bool &bloeserne)</code>
Parametre	<code>bool &Varmelegeme</code> En reference til en bool, som ændres til den aktuelle status på Varmelegemet. <code>bool &bloeserne</code> En reference til en bool, som ændres til den aktuelle status på blæserne.
Returværdi	-
Beskrivelse	Anvendes til at se om varmelegemet og/eller blæserne kører, hvis værdien for et givet stykke er true kører det stykke hardware, men hvis false kører det ikke.

Tabel 87: GetHardware

Prototype	<code>void SetHardware(const bool Varmelegeme, const bool bloeserne)</code>
Parametre	<code>const bool Varmelegeme</code> Sættes true hvis varmelegemet skal kører ellers false <code>const bool bloeserne</code> Sættes true hvis blæserne skal kører ellers false
Returværdi	-
Beskrivelse	Anvendes at sætte om Varmelegemet og/eller blæserne skal må bruges til regulering i drivhuset.

Tabel 88: SetHardware

Prototype	<code>void setDate(Date time)</code>
Parametre	<code>Date time</code> En struct af type Date som beskriver den tid som det ønskes at systemet skal indstille sig efter.
Returværdi	-
Beskrivelse	Anvendes at sætte tiden på systemet, dette sker ved at eksekvere et script bash script på systemet via en shell som smider de rette parameter ind. Systemet kan sættes til en tid mellem år 2000 og 2050.

Tabel 89: setDate

Prototype	<code>Date getDate()</code>
Parametre	-
Returværdi	En struct af type Date som beskriver den aktuelle tid som på systemet.
Beskrivelse	Anvendes henter den aktuelle tid på systemet.

Tabel 90: getDate

Prototype	<code>void GetNotifications(bool &daily, bool &Warning)</code>
Parametre	<p><code>bool &daily</code> En reference til en bool, som ændres til den aktuelle status på daily notifications, hvis den ændres til true er daily slået til, men hvis den er false er daily ikke slået til.</p> <p><code>bool &Warning</code> En reference til en bool, som ændres til den aktuelle status på warning notifications, hvis den ændres til true er warning slået til, men hvis den er false er warning ikke slået til.</p>
Returværdi	-
Beskrivelse	Anvendes til at se om daily og/eller Warning er slået til, hvis værdien for er true sendes notifications for den, men hvis false sendes den ikke.

Tabel 91: GetNotifications

Prototype	<code>void SetNotifications(const bool &daily, const bool &Warning)</code>
Parametre	<p><code>bool &daily</code> En reference til en bool, som ændres til den aktuelle status på daily notifications, hvis den ændres til true er daily slået til, men hvis den er false er daily ikke slået til.</p> <p><code>bool &Warning</code> En reference til en bool, som ændres til den aktuelle status på warning notifications, hvis den ændres til true er warning slået til, men hvis den er false er warning ikke slået til.</p>
Returværdi	-
Beskrivelse	Anvendes til at se om daily og/eller Warning er slået til, hvis værdien for er true sendes notifications for den, men hvis false sendes den ikke.

Tabel 92: SetNotifications

Prototype	<code>int GetData(int plantId, Plant &plantToedit)</code>
Parametre	<p><code>int plantId</code> Et id som beskriver hvor planeten er henne skal være et tal mellem 1 og 6</p> <p><code>Plant &plantToedit</code> En reference til en plante som du gerne hente ud.</p>
Returværdi	-
Beskrivelse	Anvendes hente en plante ud af systemet .

Tabel 93: GetData

Prototype	<code>plant* GetAll()</code>
Parametre	-
Returværdi	Et array som indeholder de 6 virtuelle planter
Beskrivelse	Anvendes hente en plante ud af systemet .

Tabel 94: GetAll

6.4.3 Boundaryklasse Monitor

Attributter

CurrentTime	Date	Indholder sidste tidpunkt modtaget fra Indstillinger.
Email	Notification	Indholder status for rapportering.
Virtuel	Plant	Indholder array af virtuelle planteparametre.
Real	Plant	Indholder array af faktiske planteparametre.

Tabel 95: Attributter for klassen Monitor

Metoder

Prototype	<code>void CompareData()</code>
Parametre	-
Returværdi	-
Beskrivelse	CompareDatas opgave er en live opdatering af plante sundhedstegn på baggrund af en sammenligningen mellem hvad brugeren har angivet som de ideelle værdier i den virtuelle plantedatabase og de faktiske værdier indsamlet fra sensorer. Hvis de faktiske værdier ligger inden for tolerancen vil Monitor farve plantefelterne i hovedmenuen grønne og hvis de ligger udenfor skal plantefelterne være røde. I tilfælde af en sensor ikke er tilkoblet, så skal plantefeltet for den pågældende sensor være grå. Hver gang en tolerance overskides skal Monitor angive hændelsen til systemloggen og signalerer rapportering for aktivering.

Tabel 96: CompareData

6.4.4 Domäneklasse Plantedatabase

Attributter

<code>dataBase</code>	<code>DoublyLinkedList<Plant></code> Datastrukturen som indeholder alle planterne
-----------------------	---

Tabel 97: Attributter for klassen Plantedatabase

Metoder

Prototype	<code>Plant GetPlantList()</code>
Parametre	-
Returværdi	<code>Plant</code> Et Plant array med alle planter i database.
Beskrivelse	Bruges til at hente alle planter i databasen.

Tabel 98: GetPlantList

Prototype	<code>Plant GetPlant(int id)</code>
Parametre	<code>int id</code> id nummeret for den ønskede plante.
Returværdi	<code>Plant</code> En plante som har det givne id.
Beskrivelse	Henter en plant ud af databasen som har det angivne id.

Tabel 99: GetPlant

Prototype	<code>void InsertPlant(Plant plant)</code>
Parametre	<code>Plant plant</code> Den plante som skal indsættes.
Returværdi	-
Beskrivelse	Indsætter en plante i databasen.

Tabel 100: InsertPlant

Prototype	<code>void DeletePlant(int id)</code>
Parametre	<code>int id</code> id på planeten som skal slettes
Returværdi	-
Beskrivelse	Sletter en planet som har det angivende id.

Tabel 101: DeletePlant

Prototype	<code>plant CreatePlant()</code>
Parametre	-
Returværdi	<code>Plant</code> En ny plante.
Beskrivelse	Opretter en ny plante i Datastrukturen og returnere denne plante, så det er muligt at redigere i data på den ved brug af plantedataredigeringsmenuen.

Tabel 102: CreatePlant

6.4.5 Boundaryklasse Rapport

Attributter

CurrentTime	Date	Indholder sidste tidpunkt modtaget fra Indstillinger.
Email	Notification	Indholder maillingsliste for status emails

Tabel 103: Attributter for klassen Rapport

Metoder

Prototype	<code>void ActivateRapport()</code>
Parametre	-
Returværdi	-
Beskrivelse	Sendes fra Monitor og angiver om rapporterings funktionaliteter skal bruges eller ej.

Tabel 104: ActivateRapport

Prototype	<code>void SendDailyRapport()</code>
Parametre	-
Returværdi	-
Beskrivelse	Sender en E-mail til brugeren med en liste over de seneste systemhændelser siden sidste E-mail.

Tabel 105: SendDailyRapport

Prototype	<code>void SendWarningRapport()</code>
Parametre	-
Returværdi	-
Beskrivelse	Sender en E-mail til brugeren med den kritiske systemhændelse.

Tabel 106: SendWarningRapport

6.4.6 Boundaryklasse Regulator

Attributter

TempHigh	bool	som standard står TempHigh som False, men hvis temperaturen bliver for høj sættes den True.
TempLow	bool	som standard står TempLow som False, men hvis temperaturen bliver for lav sættes den True.
humidityLow	bool	som standard står humidityLow som False, men hvis luftfugtigheden bliver for lav sættes den True.
humidityHigh	bool	som standard står humidityHigh som False, men hvis luftfugtigheden bliver for høj sættes den True.
plante1water 1-6	bool	Plante(1-6)water står som standard til False, men hvis jordfugtigheden bliver 2 niveauer for lavt sættes den til True.

Tabel 107: Attributter for klassen Regulator

Metoder

Prototype	<code>void ControlData()</code>
Parametre	-
Returværdi	-
Beskrivelse	Metoden har til formål at sammenligne den faktiske temperatur i drivhuset med den ønskede temperatur. Det samme fortages med luftfugtighed. I forhold til jordfugtigheden tjekkes hver plante, og hvis jordfugtigheden er for lav, sættes boolean til at den gældende plante mangler vand.

Tabel 108: ControlData

6.4.7 Domæinklasse Systemlog

Attributter

SystemMsg	DoublyLinkedList<SystemMsg>	SystemMsg implementeres som en struct, der nedarver fra Node klassen, og udvider dem med en message item af typen string.
-----------	-----------------------------	---

Tabel 109: Attributter for klassen Systemlog

Metoder

Prototype	<code>void AddMessage(string msg)</code>
Parametre	<p><code>string msg</code> <code>string msg</code> er en besked indeholdende den pågældende systemhændelse formateret på følgende måde: "klassenavn": "hændelse" på "tidspunkt".</p>
Returværdi	-
Beskrivelse	Funktionen har til formål at modtage systembeskeder fra andre klasser og indsætte disse beskeder i en datastruktur til senere brug.

Tabel 110: AddMessage

Prototype	<code>void PrintSystemLog()</code>
Parametre	-
Returværdi	-
Beskrivelse	PrintSystemLog bliver kaldt fra QT menuen "Systemlogmenu" og udskriver de sidste 5 systemhændelser med den femte ældste hændelse først og den nyeste hændelse sidst.

Tabel 111: PrintSystemLog

6.4.8 Boundaryklasse UART

Attributter

SystemMsg	DoublyLinkedList<SystemMsg>	SystemMsg implementeres som en struct, der nedarver fra Node klassen, og udvider dem med en message item af typen string.
-----------	-----------------------------	---

Tabel 112: Attributter for klassen UART

Metoder

Prototype	<code>void ScanForSensors()</code>
Parametre	-
Returværdi	-
Beskrivelse	metoden sender en besked til PSoC masteren og beder om antallet af tilsluttede sensorer til systemet. UARTEten sender kommandoen (REF til UART protokol), og venter derefter på svar fra PSoC masteren. Hvis den får et gyldigt svar tilbage afsluttes metoden. Hvis svaret ikke er gyldigt vil metoden kontakte PSoC masteren en gang til for at få antallet af tilsluttede sensorer. Efter 4 fejlforsøg afsluttes metoden, og systemet sender en besked til bruger at tjekke systemet.

Tabel 113: ScanForSensors

Prototype	<code>SensorData GetSensorData()</code>
Parametre	-
Returværdi	SensorData SensorData, som er en struct over temperatur, luftfugtighed, lysintensitet og jordfugtighed returneres med værdierne for målt temperatur, luftfugtighed, lysintensitet og jordfugtighed.
Beskrivelse	metoden sender via UART protokollen en anmodning til PSoC masteren om at få temperaturen i drivhuset. Når korrekt data er modtaget fortsætter metoden til at indsamle data for luftfugtighed, lysintensitet og jordfugtighed for de 6 planter. Hvis en fejl i en af beskederne opstår, forsøges yderlige 3 forsøg for den gældte data, før den springer den gældende data over og fortsætter til næste data.

Tabel 114: GetSensorData

6.5 Trådhåndtering

For at kunne afvikle flere funktionaliteter på samme tid i systemet bruges multithreading. Ved at bruge en thread til brugerfladen, en til at læse data, en til at regulere klimaet, hvis data ligger er for langt uden for tolerancer niveauer og en til at styrer tiden med, kan de forskellige funktionaliteter køre sideløbende.

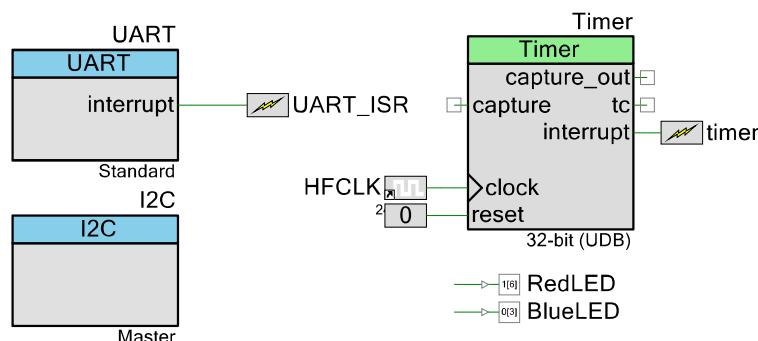
7 Hardware Implementering

7.1 Version

Dato	Version	Initialer	Ændring
31. marts	1	MHG	Implementering af SW i Aktuator.
8. april	2	MHG	Færdiggjort beskrivelse af SW i Aktuator.
27. april	3	HBJ	Opdateret beskrivelse af SW i Aktuator.
27. aril	4	MHG	Skrevet implementering af Mosfet drive til Varmelgeme, Blæsere og Vinduesmotor.
5. maj	5	HBJ	Implementering af Jordfugt tilføjet.
8. maj	6	MHG	Rettelser i Implementering af Jordfugt.
11. maj	7	PKP	PSoC Master implementering rettet og opdateret.

7.2 PSoC Master implementering

Dette afsnit indeholder overvejelser og dokumentation for implementeringen af PSoC Master blokken i systemet.



Figur 43: TopDesign.cysch for PSoC_Master

I Figur 7.2 ses topdesignet for PSoC Master. Ud fra dette ses det at vi overordnet set har at gøre med en UART, som genererer interrupts, en Timer, som genererer interrupts, en I2C blok og to outputs til LED. Selve topdesignet er lavet ud fra de behov der er stillet i Design-fasen. Der blev under implementeringen af UART afprøvet en anden UART komponent grundet problemer med UART kommunikationen, men problemet viste sig at ligge i et tidligere design-valg angående paritet.

7.2.1 Main implementering

Main funktionen, der er vist i Listing 7.1, er ganske simpel da stort set alt funktionaliteten er håndteret vha interrupts. Det er næsten altså kun initialisering af klasserne og efterfølgende konstant kald af to interrupt handler funktioner der finder sted. De to funktioner er nærmere beskrevet i afsnittet om Controller klassen på side 115.

```

20 int main(){
21     // Init
22     initDSP();
```

```
23 initI2C();  
24 initPSoC_Master();  
25 initUART();  
26 CyGlobalIntEnable; // Global interrupt enable  
27  
28 for(;;){  
29     uartIntHandler(); // Check if UART flag has been set  
30     timerIntHandler(); // Check if Timer flag has been set  
31 }  
32 return 0;  
33 }
```

Listing 7.1: PSoC Master'ens main funktion

7.2.2 I²C implementering

I forbindelse med implementeringen af I²C klassen blev prototyperne oprettet jf. I²C protokollen (se side 47). Generelt set formidler klassen kald fra både timer og UART og er i stand til at sende og indhente information hhv. fra og til sensorer og aktuatorer der er tilkoblet I²C -bussen. Timer klassen anvender I²C klassen til at hente data fra sensoren med jævne mellemrum. Dog blev der pga. problemer med de bestilte sensorer ændret i arkitekturen af projektet, hvilket medførte at der anvendes en LM75 i stedet for den tidligere oplyste temperatur/luftfugtsensor. Ligeledes er lyssensorens implementering nedprioriteret, grundet tidsmangel.

```

215 int8 getTemp(int32* temp){
216
217     uint8 dataget[2] = {0,0};
218     uint32 errorStatus[2] = {9,9};           // For debugging and error handling
219
220     I2C_I2CMasterClearReadBuf();
221     errorStatus[0] = I2C_I2CMasterSendStart(TEMP_SENSOR_ADDRESS,
222                                             I2C_I2C_READ_XFER_MODE);
223     if (errorStatus[0] == I2C_I2C_MSTR_NO_ERROR) {
224         dataget[0] = I2C_I2CMasterReadByte(I2C_I2C_ACK_DATA);
225         dataget[1] = I2C_I2CMasterReadByte(I2C_I2C_NAK_DATA);
226         errorStatus[1] = I2C_I2CMasterSendStop();
227     }
228     else{
229         I2C_I2CMasterSendStop();
230         return -1;
231     }
232
233     // The data is converted directly to UART protocol because of the ,5 resolution
234     *temp = (dataget[0]*2)+(dataget[1] >> 7)+40;
235
236     return 0;
237 }
```

Listing 7.2: Implementering af getTemp()

I Listing 7.2, ses der et eksempel på håndtering af en request fra timer interruptet. I dette eksempel er det temperatursensoren LM75, der kommunikeres med. For at LM75 reagerer på masterens kald, sendes der en slaveadresse og en read-request. Denne udskriver to bytes når der læses over I²C . Der sendes herefter en I²C stop-kommando for at frigøre bussen igen. Jf. I²C protokollen på side 47, indeholder de to bytes temperaturdata, hvor MSB angiver om temperaturen er negativ eller ej (værdien er signed) og de 8 efterfølgende bits angiver selve temperaturen med en halv grads opløsning.

Fra aktuatoren side er det valgt, at når der modtages en kommando over I²C , aktiveres der en interrupt der håndterer den pågældende kommando. For I²C -klassen betyder det at uanset tidspunktet, kan der sendes kommandoer hele tiden, men af hensyn til svaret der skal modtages over bussen, holder masteren klokken lav, indtil enten et svar eller en fejl er modtaget. Princippet i read/write funktionen er, at masteren blokerer for andre interrupts og sender en slaveadresse ud på bussen, hvorefter der kommer et acknowledge tilbage fra slaven, som derefter sender et antal bytes ud. Det samme kan overordnet siges om write metoden.

```

163 int8 getActuatorStatus(uint8* window, uint8* heat, uint8* vent, uint8* irrigation){
164     uint8 result = 0;
165     uint8 dataget[2] = {0, 0};
166
167     CyDelay(60);
168     I2C_I2CMasterClearReadBuf();
```

```

169 result = I2C_I2CMasterReadBuf(ACTUATOR_ADDRESS, dataget, 2,
170     I2C_MODE_COMPLETE_XFER);
171
172 while (0u == (I2C_I2CMasterStatus() & I2C_I2C_MSTAT_RD_CMPLT)); //Wait for the
173     dataget array to be updated
174
175 if ((result == I2C_I2C_MSTR_NO_ERROR) && (I2C_I2CMasterGetReadBufSize() != 0)){
176     if (window){                                     // Expecting to receive MSB
177         first
178         *window = (dataget[0] >> 4);           // Shifting out the 4 least significant
179             bits.
180             #ifdef debugging
181                 UART_UartPutChar(*window+CONVERT_TO_ASCII);
182             #endif
183     }
184     if (heat){
185         *heat = ((dataget[0] & 0b00001110) >> 1);      // Ignoring everything
186             but bit 1-3 and shifting 1 right.
187             #ifdef debugging
188                 UART_UartPutChar(*heat+CONVERT_TO_ASCII);
189             #endif
190     }
191     if (vent){
192         if ((dataget[0] & 0b00000001) == 0b00000001){          // Maybe we can find
193             a smarter way to do this?
194             *vent = (dataget[1] >> 6) | 0b00000100;           // The if statements
195                 checks if bit 1 of dataget[0] is 1 or not and then sends it
196                 onwards.
197                 #ifdef debugging
198                     UART_UartPutChar(*vent+CONVERT_TO_ASCII);
199                         // Shifting 5 right so only 2 bits
200                             are left and adding bit 1 of dataget[0] in the 3rd bit.
201                         #endif
202     }
203     else {
204         *vent = (dataget[1] >> 6) | 0b00000000;           // shifting 5 right
205             since only the two most significant bits are relevant.
206             #ifdef debugging
207                 UART_UartPutChar(*vent+CONVERT_TO_ASCII);
208             #endif
209     }
210 }
211 }
```

Listing 7.3: Implementering af getActuatorStatus()

Det kan ses på Listing 7.3 linje 193, at systemet venter på, at bussen bliver ledig. Under debugging af I²C -klassen var der store problemer, netop med at bussen ikke var ledig under fx. en læsning.

Problemet viste sig at ligge i rækkefølgen PSoC4 afvikler de interrupts der kommer. Systemet ventede med at udføre alle andre metodekald, indtil efter UART'en havde fået et svar. Dog udførte systemet stadig en del af funktionaliteten for I²C read/write, hvilket betød at bussen var optaget i den tid der blev forsøgt at læse. Løsningen på problemet viste sig at være en genopbygning af UART- og timer-interruptet, således alt funktionelt blev rykket ud i main og at der bliver sat flag, når det er nødvendigt.

7.2.3 UART implementering

UART klassen har til formål at modtage kommandoer fra DevKit8000, tolke disse og give passende svar.

Generelt set er klassen implementering med udgangspunkt i vores UART protokol, men er udvidet således at der let kan implementeres flere trin i hhv. kontrol af vindue, motor og vanding. UART klassen gør brug af UART komponenten (SCB) vist i Figur 7.2.

```

24 int8 respondTemp( uint8 temp){
25     if(temp){
26         // If temp is between 1 and 200(both inclusive) "T" and temp is sent to
27         // DevKit8000
28         UART_UartPutChar('T');
29         UART_UartPutChar(temp);
30         return 0;
31     }
32     else{
33         // If temp isn't between 1 and 200(both inclusive) "XT" is sent to DevKit8000
34         UART_UartPutChar('X');
35         UART_UartPutChar('T');
36     }
37 }
```

Listing 7.4: Implementering af respondTemp()

I Listing 7.4 vises et eksempel på en af funktionerne der håndterer svar via UART. De øvrige funktioner i klassen fungerer på samme måde. Funktionen modtager den værdi, der skal sendes tilbage til DevKit8000. Hvis parametren er 0 vil det sige at der er sket en fejl. Når funktionen kaldes kaldes den med returnværdi fra DSP klassen, som beskrevet i afsnit 7.2.4.

```

113 uint8 dkRequest( void){
114     // Reads the UART buffer
115     return UART_UartGetChar();
116 }
```

Listing 7.5: Implementering af dkRequest()

Vi har ydermere valgt at indkapsle løsningen fra UART ved hjælp af dkRequest() funktionen vist i Listing 7.5. Grunden til at vi har valgt at implementere denne er for at sikre os at hvis UART protokollen skulle ændre sig i fremtiden, kan disse ændringer tages højde for i denne funktion inden PSoC Master controllerklassen skal håndtere input fra UART.

Klassen er testet ved at koble en PC på med en COM port og via en terminal indlæse forskellige værdier, for at simulere aktivitet fra DevKit8000. Der er herved kontrolleret at alle muligheder for inputs er tjekket og at klassen udfører det den skal ved hvert input.

7.2.4 DSP implementering

DSP klassen agerer både digital signal processor og hukommelse for vores måledata. Hver type af data er gemt i sit eget array, som vist i Listing 7.6. Hvert arrays har ligeledes en pointer til den næste plads i arrayet der skal overskrives.

```

16 // Private data members
17 int32 tempArray [ARRAYSIZE];
18 int32* tempArrayPtr;
19 int16 soilHumArray [NBR_OF_SOILHUM_SENSORS] [ARRAYSIZE];
20 int16* soilHumPtr [NBR_OF_SOILHUM_SENSORS];
21 uint8 temp, soilHum [NBR_OF_SOILHUM_SENSORS];

```

Listing 7.6: Deklaration af arrays og pointers

Arrays'ne bliver brugt til at gemme en række datapunkter i råt format. Disse datapunkter kan herefter konverteres og midles. For jordfugt (**soilHumArray**), er der oprettet et todimensionelt array således at der kan gemmes et array med data for hver af de 6 sensorer.

Når der er målt en ny værdi fra en sensor, via I²C klassen, bliver den indlæst i DSP klassen med input-funktionerne.

```

136 void inputTemp (int32* temp) {
137     *tempArrayPtr = *temp;           // The input value is written to the array
138     tempArrayPtr++;                // The pointer is moved to the next place in array
139     if (tempArrayPtr > &tempArray [ARRAYSIZE-1]) {
140         tempArrayPtr = &tempArray [0]; // If the pointer is pointing past the end of
141                                     // the array it's reset
142     }
143     avgTemp();                   // The average value is calculated and converted into temp (global)
144 }

```

Listing 7.7: Funktion til at indlæse en sensorværdi i DSP klassen

I Listing 7.7 ses **inputTemp**-funktionen der indlæser den målte værdi i **tempArray** vha den tilhørende pointer **tempArrayPtr**. Pointeren flyttes herefter til næste plads i arrayet. Arrayet overskrives på ny når dette er fuldt. Til sidst kaldes den private funktion **avgTemp**.

```

67 void avgTemp (void) {
68     uint8 skip = 0;
69     int64 total = 0;
70     {
71         uint8 i ;
72         for (i = 0 ; i < ARRAYSIZE ; i++){
73             if (tempArray [i]>=0){
74                 total += tempArray [i];
75             }
76             else{
77                 skip++;
78             }
79         }
80     }
81     // Makes sure that enough datapoints are present
82     if (ARRAYSIZE-skip>=NMR_OF_VALID_DATAPOLNTS_NEEDED) {
83         int32 avg = total/(ARRAYSIZE-skip);           // Calculate the average value
84
85         // temp is limited to 1 and 200
86         if (avg>200){
87             temp = 200;
88         }

```

```

89         else if (avg < 1){
90             temp = 1;
91         }
92     else{
93         temp = (uint8)avg;
94     }
95 }
96 else{
97     temp = 0;
98 }
99 }
```

Listing 7.8: Funktion der midler og konverterer tempArray

I Listing 7.8 vises avgTemp funktionen. Denne har flere funktionaliteter. Først midles alle de data der er i `tempArray`, samtidig med at det kontrolleres at der er nok valide datapunkter. Øverst i klassen er der defineret hvor mange valide datapunkter der skal være til stede i et givent array fra en sensor, for at der sendes en værdi videre fra PSoC_Master til DevKit8000. Herefter konverteres data fra den form det kommer i fra sensorerne, til den form de skal sendes til DevKit8000 i. I dette tilfælde med temperaturen, skal temperaturen begrænses en værdi mellem 1 og 200, jf. UART protokollen som kan ses på side 44. Når data'en er blevet begrænset, gemmes den i en privat variabel(`temp`), som herefter kan tilgås med funktionen `getTemp_DSP`, der kan ses i Listing 7.9.

```

55 uint8 getTemp_DSP(void){
56     return temp;
57 }
```

Listing 7.9: Getter-funktion som returnerer den seneste midlede temperatur

De resterende dele af DSP klassen, altså dem der håndterer jordfugt, luftfugt og lysintensitet, er implementeret efter samme principper og fremgangsmåde og der henvises derfor til kommentarerne i kildekoden som findes i Bilag XXX. for mere information om dette.

7.2.5 Controller implementering

PSoC_Master controller-klassen er som udgangspunkt designet ud fra at blive styret af hvilke kommandoer der er modtaget på UART'en. På den måde agerer vores 'master' slave for DevKit8000. For at huske den nuværende status er der oprettet en `enum` med den nuværende status samt ekstra buffer til at holde styr på hvilken vandingsaktuator der modtages data omkring.

```

33 // Buffers / flags
34 typedef enum {IDLE, ADJW, ADJH, ADJV, ADJI, RESP_SOIL_HUM} state;
35 volatile state theState = IDLE;
36 volatile int8 irrigationIndex = 0;
37 uint8 buff;
38 uint8 uartInt = 0;
39 uint8 timerInt = 0;

```

Listing 7.10: Deklaration af buffers og flag.

Ydermere er der lavet en form for debugging ved hjælp af de tre farvede LED'er på PSoC4 Pioneer Kit. Der tændes fx for den røde LED når et interrupt sker på timeren og for den blå når et interrupt sker på UART'en.

Når der sker et interrupt på UART'en, sættes flaget `uartInt` til 1 og der fyldes data i bufferen. Dette ses i Listing 7.11.

```

80 // UART ISR
81 CY_ISR(UART_ISR){
82     uartInt = 1;
83     buff = dkRequest();
84     UART_ClearRxInterruptSource(UART_GetRxInterruptSourceMasked()); // Clear
85         interrupt flag
}

```

Listing 7.11: ISR for UART.

Årsagen til at vi har valgt at sætte et flag er at vi hurtigst muligt vil ud af interrupt service rutinen samt at det gav os problemer at have al funktionaliteten som kald fra UART ISR.

Der er derfor udarbejdet en ny privat funktion kaldet `uartIntHandler()`, som sørger for at håndtere selve arbejdet mht. det input UART'en giver. Denne bliver kaldt med jævne mellemrum fra en `while(1)` løkke i main.c, se Listing 7.1 på side 108.

```

89 void uartIntHandler(void){
90     if (uartInt){
91         uartInt = 0;
92         BlueLED_Write(LED_ON); // Turn on blue LED
93
94         if (theState == IDLE){
95             switch (buff){
96                 case 'T':{ //RequestTemp
97                     respondTemp(getTemp_DSP());
98                     break;
99                 }
100                case 'H':{ //TurnHeatOn
101                    // 0x7 is the maximum value.
102                    respondHeat(adjustHeat(0x7), 'H');
103                    break;
104                }
105                case 'K':{ //TurnHeatOff
106                    // 0x0 is the minimum value.
107                    respondHeat(adjustHeat(0x0), 'K');
108                    break;
}

```

```

109 }
110 case 'W':{ //AdjustWindow
111     theState = ADJW;
112     break;
113 }
114 case 'V':{ //Ventilation
115     theState = ADJV;
116     break;
117 }
118 case 'F':{ //Watering
119     theState = ADJI;
120     break;
121 }
122 case 'S':{
123     theState = RESP_SOIL_HUM;
124     break;
125 }
126 default:{
127     // Do nothing – let the DevKit8000 timeout
128     break;
129 }
130 }
131 }
132 else if(theState == ADJW){
133     if(buff-CONVERT_TO_ASCII == 1){
134         respondWin(adjustWindow(0xFF));
135     }
136     else{
137         respondWin(adjustWindow(0x00));
138     }
139     theState = IDLE;
140 }
141 else if(theState == ADJV){
142     if(buff-CONVERT_TO_ASCII == 1){
143         respondVent(adjustVentilation(0xFF));
144     }
145     else{
146         respondVent(adjustVentilation(0x00));
147     }
148     theState = IDLE;
149 }
150 else if(theState == ADJI){
151     if(!irrigationIndex){
152         irrigationIndex = buff - CONVERT_TO_ASCII;
153     }
154     else{
155         if(buff-CONVERT_TO_ASCII == 1){
156             respondIrri(adjustIrrigation(irrigationIndex - 1, 0xFF));
157         }
158         else if(buff - CONVERT_TO_ASCII == 0){
159             respondIrri(adjustIrrigation(irrigationIndex - 1, 0x00));
160         }
161         else{
162             respondIrri(-1); //Bad argument
163         }
164         irrigationIndex = 0;
165         theState = IDLE;
166     }
167 }
168 else if(theState == RESP_SOIL_HUM){

```

```

169     uint8 temp = buff - CONVERT_TO_ASCII - 1; //incoming number is in ASCII
170     1-6, DSP works in 0-5.
171     if (temp <= 5 && temp >= 0){
172         respondSoilHum(temp, getSoilHum_DSP(temp));
173         theState = IDLE;
174     }
175     buff = 0;
176     BlueLED_Write(LED_OFF);           // Turn off blue LED
177 }
178 }
```

Listing 7.12: Interrupt handler for UART.

I Listing 7.12 ses implementeringen af selve interrupthandleren. Det ses hvordan der først tjekkes for om der er kommet ny data via `uartInt` og der herefter kontrolleres hvilket stadie, systemet er i. Hvis det er i IDLE tjekkes der på hvad der står i bufferen og der udfærdes evt et svar eller ventes på næste databyte. Til at starte med tændes den blå LED og når kaldet er slut slukkes denne.

Til at indsamle data fra sensorerne med jævne mellemrum er der implementeret en timer med et tilhørende interrupt. Denne interrupt service rutine sætter et flag, på samme måde som UART-klassen, som herefter udløser en privat funktion `timerIntHandler`.

```

182 void timerIntHandler(void){
183     if(timerInt){
184         timerInt = 0;           // Reset flag
185         RedLED_Write(LED_ON); // Turn on red LED
186
187         // Measure temp and input to DSP class
188         getTemp(&tempTemp);
189         inputTemp(&tempTemp);
190
191         // Measure soilhum and input to DSP class
192         {
193             uint8 i;
194             for(i = 0; i<6 ; i++){
195                 getSoilHum(i, &tempSoilHum[i]);
196                 inputSoilHum(i, &tempSoilHum[i]);
197             }
198         }
199
200         RedLED_Write(LED_OFF); // Turn off red LED
201     }
202 }
```

Listing 7.13: Interrupt handler for timer.

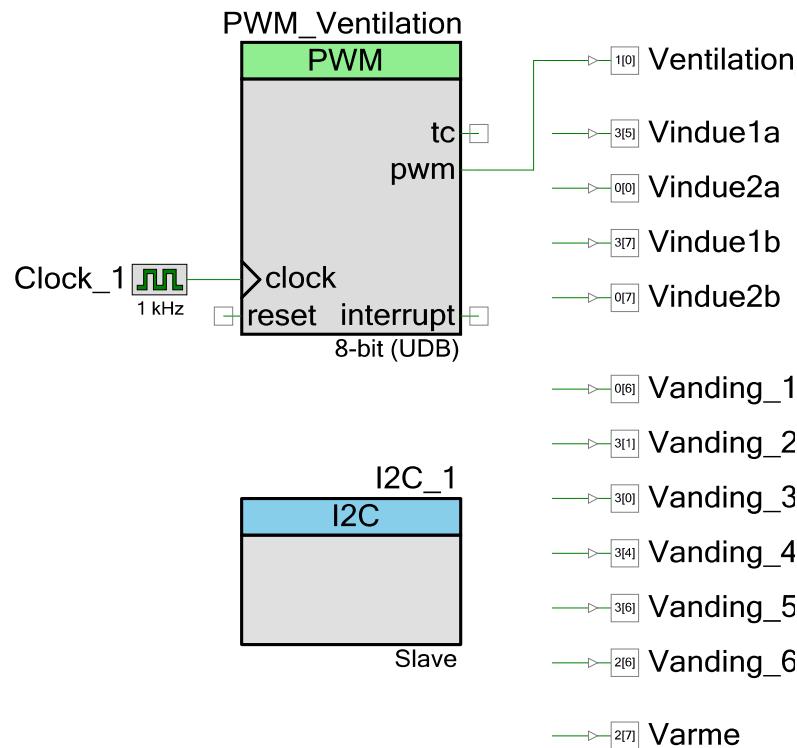
I Listing 7.13 ses selve implementeringen af `timerIntHandler`. Den har, ligesom `uartIntHandler`, en LED der tændes og der udføres herefter selve arbejdet, som består i at fodre data ind i vores digitale signalprocessor (DPS-klassen). Data'en kommer fra funktioner i I2C-klassen, der henter selve dataen fra vores sensorer på I²C bussen.

Værd at notere at for data fra jordfugtsensorerne tjekkes hver sensor for sig, selvom de alle sidder på samme I2C slave reelt set.

7.3 Aktuator

Dette afsnit beskriver implementering af SW og HW i blokken Aktuator. Afsnittet beskriver først hhv. HW og SW i underblokken PSoC4, herefter HW i underblokkene Varmelegeme, Blæsere og Vinduesmotor.

7.3.1 HW PSoC4



Figur 44: TopDesign.cysch for PSoC4 i Aktuator

Den HW, der syntetiseres i PSoC4 Aktuator er vist på figur44.

Clock_1 forsyner PWM komponenten med en grundfrekvens; der er valgt 1 kHz.

PWM_Ventilation genererer et PWM signal til styring af drifthusets fire ventilatorer. Timeren er indstillet til 8 bit.

I2C_1 komponenten styrer kommunikation med MasterPSoC via I²C . Den konfigureret til at være slave med adressen 0x42, datarate er indstillet til 100 kbps.

Alle pins i topdeignet er konfigureret til strong drive. Det vil sige at de altd er defineret enten high eller low.

7.3.2 SW PSoC4

```

1   for (;;) //Evigt loop
2   {
3       checkForData(); //Opdater Status_Reg, hvis der er modtaget data
4
5       if (!(Status_Reg == Position_Reg))//Hvis der er et mismatch
6       {
7           //Tjek for mismatch for Varme
8           if (!((Status_Reg & 0b0000111000000000) == (Position_Reg & 0
9               b0000111000000000)))
10          {
11              AdjustHeat();
12          }
13
14          //Tjek for mismatch for Ventilation
15          if (!((Status_Reg & 0b0000000111000000) == (Position_Reg & 0
16              b0000000111000000)))
17          {
18              AdjustVentilation();
19          }
20
21          //Tjek for mismatch for Vandning
22          if (!((Status_Reg & 0b0000000000111111) == (Position_Reg & 0
23              b0000000000111111)))
24          {
25              AdjustIrrigation();
26          }
27
28          //Tjek for mismatch for vindue
29          if (!((Status_Reg & 0b1111000000000000) == (Position_Reg & 0
30              b1111000000000000)))
31          {
32              AdjustWindow();
33          }
34      }
35  }

```

Listing 7.14: Udsnit af main.c for PSoC4 i Aktuator

Filen main.c, hvoraf den vigtigste del er vist på Listing 7.14, fungerer jf. State Machinen på Figur 25 side 72.

Programmet tjekker om der er modtaget data på I²C , og opdaterer evt. ønskede indstillinger for aktuatorer i Status_Reg.

Herefter sammenlignes Status_Reg med nuværende indstillinger af aktuatorer i Position_Reg.

Såfremt der er uoverensstemmelse, opdateres den pågældende aktuator.

Sammenligningen af de to registre sker i en prioriteret rækkefølge.

Vinduet er sidste i denne proces, da det tager temmelig lang tid (flere sekunder) at åbne eller lukke vindet.

```

1 void checkForData()
2 {
3     uint16 temp = 0;
4     //Check for om der er modtaget data
5     if(I2C_1_I2CSlaveStatus() & I2C_1_I2C_SSTAT_WR_CMPLT)
6     {
7         //Put data i Status_Reg
8         if((writeBuffer[0] >> 6) == 0x0) //Check for Vindue
9         {
10            //Put data fra buffer i uint16 og skift til rigtig position
11            temp = (writeBuffer[0] & 0b00001111) << 12;
12            //Overskriv relevante pladser med 0'er
13            Status_Reg = Status_Reg & 0b0000111111111111;
14            //Put nye data ind i Status_Reg
15            Status_Reg = Status_Reg | temp;
16        }
17        if((writeBuffer[0] >> 6) == 0x1) //Check for Varme
18        {
19            //Put data fra buffer i uint16 og skift til rigtig position
20            temp = (writeBuffer[0] & 0b00000111) << 9;
21            //Overskriv relevante pladser med 0'er
22            Status_Reg = Status_Reg & 0b1111000111111111;
23            //Put nye data ind i Status_Reg
24            Status_Reg = Status_Reg | temp;
25        }
26        if((writeBuffer[0] >> 6) == 0x2) //Check for Ventilation
27        {
28            //Put data fra buffer i uint16 og skift til rigtig position
29            temp = (writeBuffer[0] & 0b00000111) << 6;
30            //Overskriv relevante pladser med 0'er
31            Status_Reg = Status_Reg & 0b1111110001111111;
32            //Put nye data ind i Status_Reg
33            Status_Reg = Status_Reg | temp;
34        }
35        if((writeBuffer[0] >> 6) == 0x3) //Check for Vandning
36        {
37            //Put data fra buffer i uint16 og skift til rigtig position
38            temp = (writeBuffer[0] & 0b00111111);
39            //Overskriv relevante pladser med 0'er
40            Status_Reg = Status_Reg & 0b11111111000000;
41            //Put nye data ind i Status_Reg
42            Status_Reg = Status_Reg | temp;
43        }
44        I2C_1_I2CSlaveClearWriteBuf(); //Clear buffer pointer
45        I2C_1_I2CSlaveClearWriteStatus(); //Clear status
46        //Opdater Read buffer
47        readBuffer[0] = Status_Reg >> 8;
48        readBuffer[1] = Status_Reg;
49        I2C_1_I2CSlaveClearReadBuf();
50    }
51 }

```

Listing 7.15: Udsnit af checkForData.c for PSoC4 i Aktuator

Funktionen checkForData() på Listing 7.15 checker om slavens status er, at den har modtaget data. I så fald checker den for hvilken aktuator, der modtages data til. Herefter behandles data, og Status_Reg opdateres.

Efter dette klargøres systemet til at modtage nye data, ved at write buffer og status for slaven nulstilles.

Til slut opdateres read buffer, i tilfælde af at MasterPSoC beder om information om aktuel status.

```

1 void InitHeat()
2 {
3     //Slukker for Varme
4     Varme_Write(0);
5
6     //Opdater nuværende indstilling
7     Position_Reg = Position_Reg & 0b11100011111111;
8 }
9
10 void AdjustHeat()
11 {
12     //Opdater aktuator for varmelegeme
13     Varme_Write((Status_Reg & 0b0000111000000000) >> 9);
14
15     //Opdater nuværende indstilling
16     Position_Reg = Position_Reg & 0b11100011111111;
17     Position_Reg = Position_Reg | (Status_Reg & 0b0000111000000000);
18 }
```

Listing 7.16: Udsnit af heat.c for PSoC4 i Aktuator

Koden i heat.c i Listing 7.16 består af to funktioner.

InitHeat() trækker pin for varme lav og initialiserer indstilling af Position_Reg for varme.

AdjustHeat() opdaterer pin for varme og Position_Reg opdateres med de nuværende indstillinger for aktuator.

```

1 void InitVentilation()
2 {
3     //Start komponent
4     PWM_Ventilation_Start();
5     //Sluk ventilatorer
6     PWM_Ventilation_WriteCompare(0);
7     //Opdater nuvaerende indstillinger
8     Position_Reg = Position_Reg & 0b1111111000111111;
9 }
10
11 void AdjustVentilation()
12 {
13     //Start med fuld styrke for at blaeserne kommer i gang.
14     PWM_Ventilation_WriteCompare((MAXIMUM_VENT*255)/100);
15     CyDelay(100);
16     //Omregn bits fra Status_Reg og start PWM med oensket dutycycle
17     PWM_Ventilation_WriteCompare(((Status_Reg & 0b0000000111000000) >> 6)*
18         MAXIMUM_VENT*255)/(7*100));
19
20     //Opdater nuvaerende indstillinger
21     Position_Reg = Position_Reg & 0b1111111000111111;
22     Position_Reg = Position_Reg | (Status_Reg & 0b0000000111000000);
}

```

Listing 7.17: Udsnit af ventilation.c for PSoC4 i Aktuator

Koden i ventilation.c i Listing 7.17 består af to funktioner.

InitVentilation() starter PWM komponenten, slukker for blæsere (dutycycle = 0%) og initialiserer indstilling af Position_Reg for ventilation.

AdjustVentilation() indstiller ønsket dutycycle i PWM komponenten.

MAXIMUM_VENT er en global definition af den dutycycle, der maximalt ønskes. Ved praktiske forsøg er det konstateret at 50% er passende.

For at sikre at blæserne rent faktisk kommer i gang, startes PWM komponenten først for fuld styrke i 100 ms, hvorefter den ønskede dutycycle indstilles.

Der er som udgangspunkt kun mulighed for at tænde eller slukker for ventilationen, men ved at lave koden på denne måde, kan systemet meget nemt opgraderes, hvis PWM styring af varmelegemet ønskes.

Efter start at PWM komponenten opdateres Position_Reg med de nuværende indstillinger for aktuatorer.

```

1 void InitIrrigation()
2 {
3     //Sluk for al vanding
4     Vanding_1_Write(0);
5     Vanding_2_Write(0);
6     Vanding_3_Write(0);
7     Vanding_4_Write(0);
8     Vanding_5_Write(0);
9     Vanding_6_Write(0);
10
11    //Opdater nuvaerende indstillinger
12    Position_Reg = Position_Reg & 0b1111111111000000;
13 }
14
15 void AdjustIrrigation()
16 {
17     //Opdater alle aktuatorer for vanding
18     Vanding_1_Write(Status_Reg & 0b0000000000000001);
19     Vanding_2_Write((Status_Reg & 0b00000000000000010) >> 1);
20     Vanding_3_Write((Status_Reg & 0b000000000000000100) >> 2);
21     Vanding_4_Write((Status_Reg & 0b0000000000000001000) >> 3);
22     Vanding_5_Write((Status_Reg & 0b00000000000000010000) >> 4);
23     Vanding_6_Write((Status_Reg & 0b000000000000000100000) >> 5);
24
25    //Opdater nuvaerende indstilling
26    Position_Reg = Position_Reg & 0b1111111111000000;
27    Position_Reg = Position_Reg | (Status_Reg & 0b0000000000011111);
28 }
```

Listing 7.18: Udsnit af irrigation.c for PSoC4 i Aktuator

Koden i irrigation.c i Listing 7.18 består af to funktioner.

InitIrrigation() trækker alle pins for vanding lav og initialiserer Position_Reg.
 AdjustIrrigation() opdaterer alle pins for vanding og indstiller Position_Reg.

```

1 void InitWindow()
2 {
3     // Initialisering af pins til startposition
4     Vinde1a_Write(1);
5     Vinde2a_Write(1);
6     Vinde1b_Write(0);
7     Vinde2b_Write(0);
8     // Initialisering af nuvaerende position
9     currentTurn = 0;
10    // Opdatering af nuvaerende indstillinger
11    Position_Reg = Position_Reg & 0b0000111111111111;
12 }
13
14 void AdjustWindow()
15 {
16     // Konvertering af data fra Status_Reg og indsættelse i desiredTurn.
17     desiredTurn = ((MAX_WINDOW)*(((Status_Reg >> 12)*100)/15))/100;
18
19     while(desiredTurn != currentTurn) // Saa længe vinduet ikke er i ønsket position
20     {
21         if (currentTurn > desiredTurn) // Luk 1 omgang hvis vinduet er for aabent
22         {
23             CloseOneTurn();
24         }
25
26         if (currentTurn < desiredTurn) // aaben 1 omgang hvis vinduet er for lukket
27         {
28             OpenOneTurn();
29         }
30     }
31     // Opdatering af nuvaerende indstillinger
32     Position_Reg = Position_Reg & 0b0000111111111111;
33     Position_Reg = Position_Reg | (Status_Reg & 0b1111000000000000);
34 }

```

Listing 7.19: Udsnit A af window.c for PSoC4 i Aktuator

Kodeudsnittet fra window.c på Listing 7.19 viser de to funktioner InitWindow() og AdjustWindow().

MAX_WINDOW er en global definition, som angiver antallet af steps motoren skal køre, for at åbne vinduet helt. TIME_BETWEEN_STEPS er en global definition som angiver hvor mange milisekunder, der går mellem hvert af motorens steps. Ved praktiske forsøg er hhv. 420 steps og 10 ms fundet hensigtsmæssige.

InitWindow() initialiserer de fire vindues pins til startposition og initialiserer Position_Reg.

Den initialiserer desuden variablen currentTurn til 0. Denne variabel holder styr på hvor vinduet befinner sig.

BEMÆRK! Vinduet skal være lukket, når systemet startes!

AdjustWindow() kontrollerer om currentTurn er større, mindre eller lig desiredTurn. Hvis de er forskellige kaldes enten OpenOneTurn() eller CloseOneTurn().

Herefter opdateres Position_Reg.

```

1 void CloseOneTurn() //48 steps paa en omgang, 12*4=48, funktionen lukker 1/12 af en
2   omgang.
3 {
4   //Efter hvert fjerde step checkes der for ny data og stilling samt oenskede
5   //stilling opdateres
6   Vindue1a_Write(0);
7   Vindue2a_Write(1);
8   Vindue1b_Write(1);
9   Vindue2b_Write(0);
10  CyDelay(TIME_BETWEEN_STEPS);
11  Vindue1a_Write(0);
12  Vindue2a_Write(0);
13  Vindue1b_Write(1);
14  Vindue2b_Write(1);
15  CyDelay(TIME_BETWEEN_STEPS);
16  Vindue1a_Write(1);
17  Vindue2a_Write(0);
18  Vindue1b_Write(0);
19  Vindue2b_Write(1);
20  CyDelay(TIME_BETWEEN_STEPS);
21  Vindue1a_Write(1);
22  Vindue2a_Write(1);
23  Vindue1b_Write(0);
24  Vindue2b_Write(0);
25  CyDelay(TIME_BETWEEN_STEPS);
26  checkForData();
27  desiredTurn = ((MAX_WINDOW)*(((Status_Reg >> 12)*100)/15))/100;
28  currentTurn--;
29 }
30
31 void OpenOneTurn() //48 steps paa en omgang, 12*4=48, funktionenaabner 1/12 af en
32   omgang.
33 {
34   ...

```

Listing 7.20: Udsnit B af window.c for PSoC4 i Aktuator

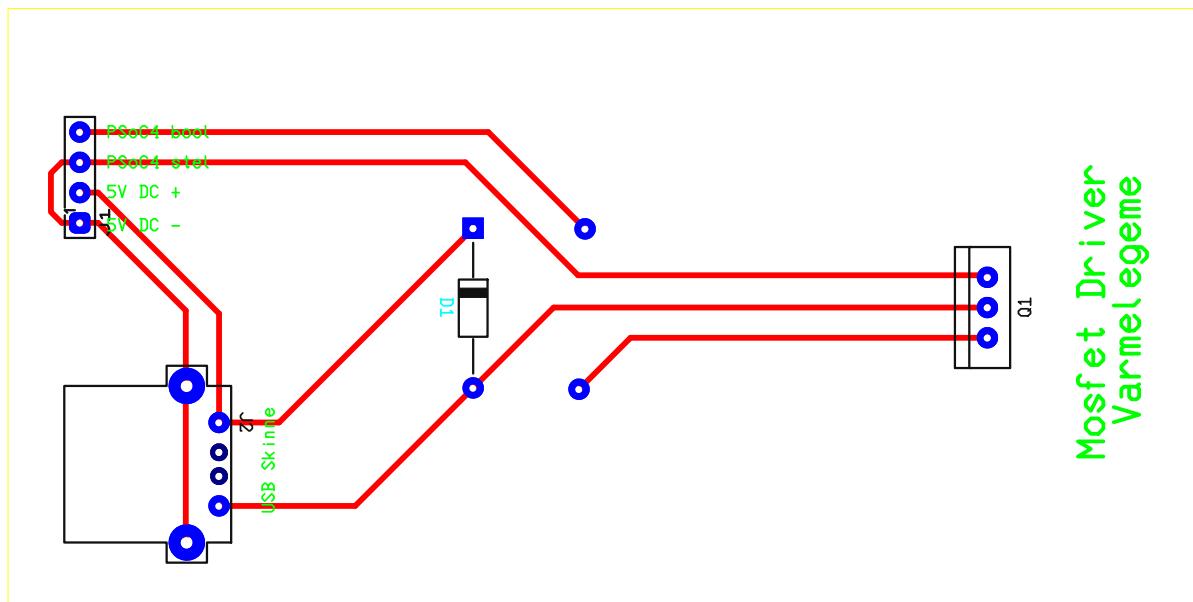
Kodeudsnittet fra window.c på Listing 7.20 viser de to funktioner CloseOneTurn() (og OpenOneTurn()).

CloseOneTurn() gennemløber sekvensen for at motoren kører et step mod urets retning.

Sekvensen i OpenOneTurn() er modsat, så motoren kører med urets retning.

For hvert step motoren kører, tjekkes der for nye data, og desiredTurn opdateres. Dette sker før at en ny kommando til vinduet eksekveres inden en igangværende kommando afsluttes. Herefter opdateres currentTurn. Denne funktionalitet er implementeret i begge funktioner.

7.3.3 HW Varmelegeme



Figur 45: Printudlæg for Mosfet driver til Varmelegeme i Ultiboard

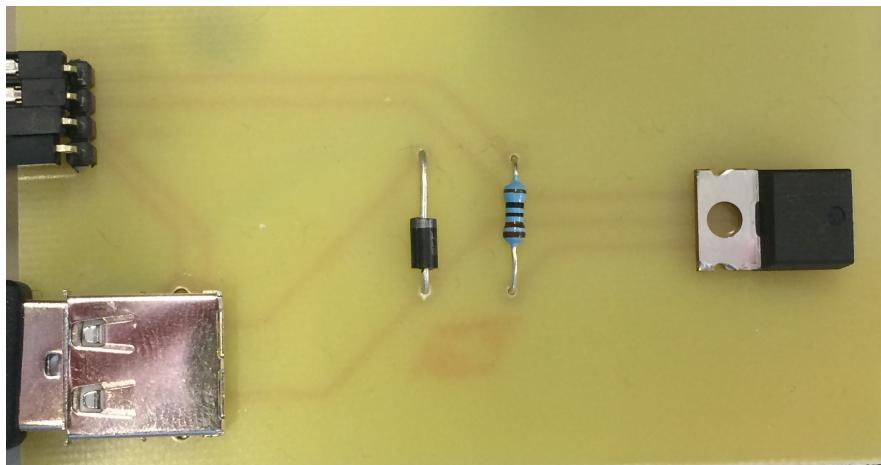
Implementering af mosfet driveren til varmelegemet foretages ved at Multisim diagrammet eksporteres til Ultiboard, hvorefter printet udlægges. Det forsøges gjort således at printet er overskueligt fremfor at printet fylder så lidt som muligt. Som udgangspunkt er alle forbindelser på printet lagt på bagsiden. Det designede print er vist på Figur 45.

Kobber på undersiden af printet er vist med rød, mens kobber på oversiden af printet er vist med grøn. Kobberører er vist med blå. Der er en lille hage ved kobberørerne; der er ikke forbindelse mellem oversiden og undersiden. Dette har dog ingen betydning, da der loddes komponentben igennem dem alle.

Omkredserne af komponenterne (sort) printes ikke, de vises kun som en slags hjælpelag. Databasen i Ultiboard indeholder ikke komponenter til modstande, derfor er disse omkredse ikke med på figuren.

Der skrives lidt forklarende tekst på oversiden af printet, så man kan se hvad der skal kobles til hvor; dette er lidt svært at se på figuren.

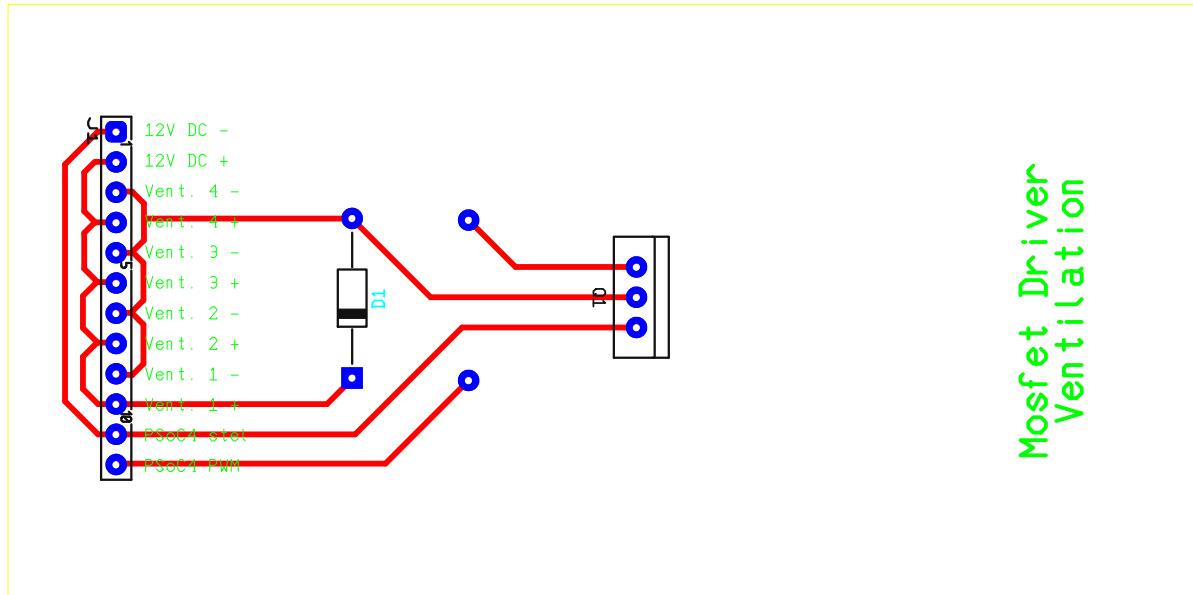
Printudlægget bestilles ved E-LAB på IHA, hvorefter komponenter loddes på. Det færdige print er vist på Figur 46.



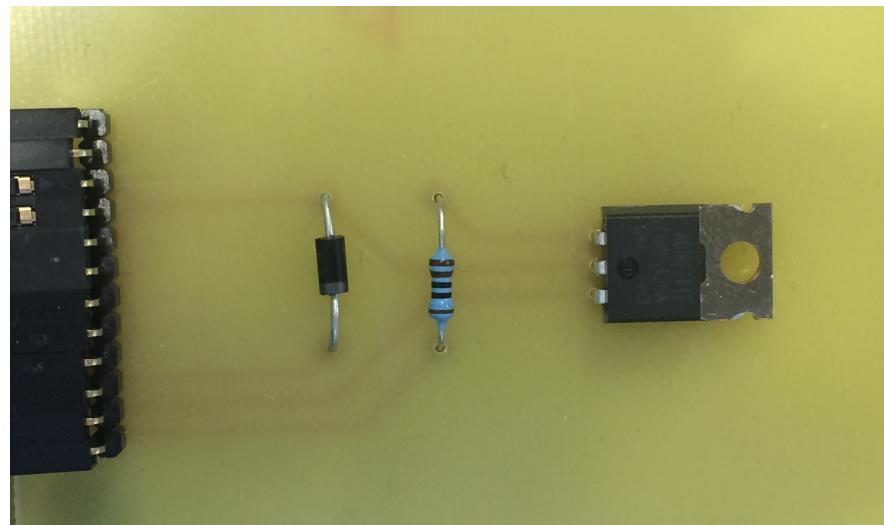
Figur 46: Den færdige Mosfet driver til Varmelegeme

7.3.4 HW Blæsere

Implementering af Mosfetdriver til Blæsere foretages på samme måde som til Varmelegeme. Printudlægget i Ultiboard er vist på Figur 47, og det færdige print er vist på Figur 48.



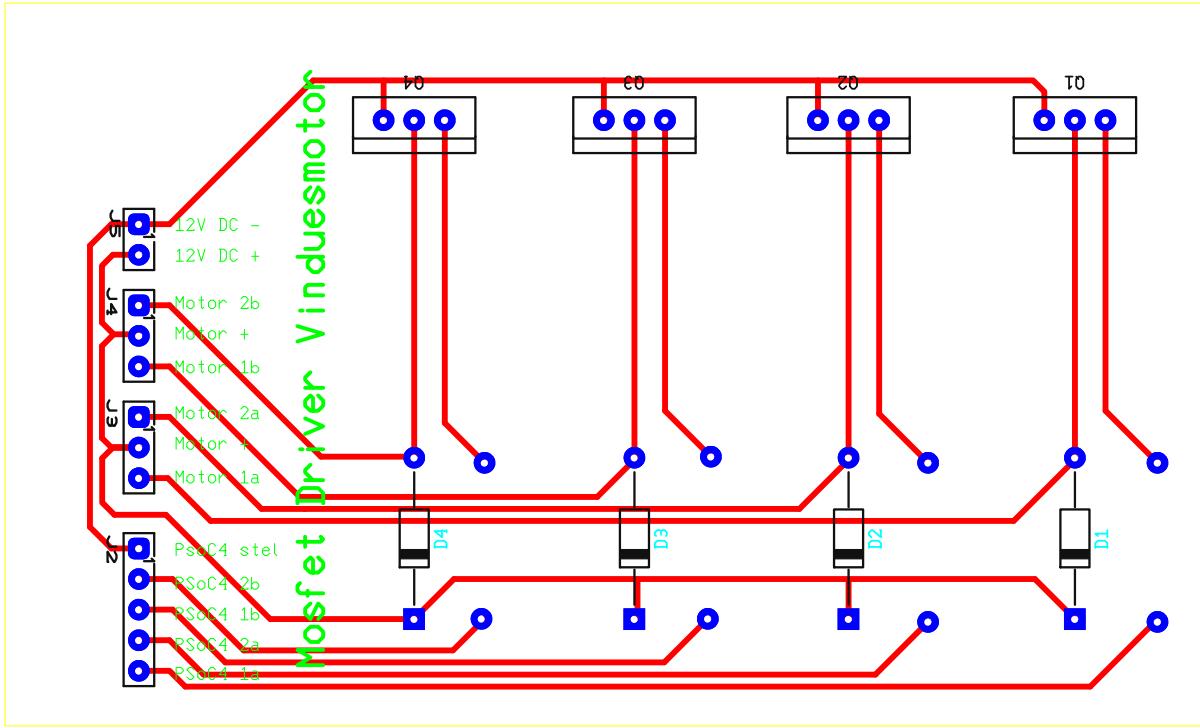
Figur 47: Printudlæg for Mosfet driver til Blæsere i Ultiboard



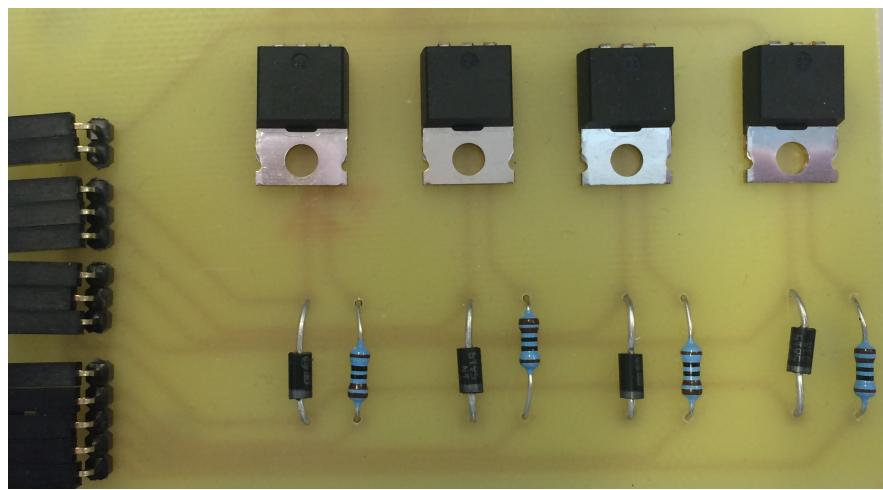
Figur 48: Den færdige Mosfet driver til Blæsere

7.3.5 HW Vinduesmotor

Implementering af Mosfetdriver til Vinduesmotor foretages på samme måde som til Varmelegeme og Blæsere. Printudlægget i Ultiboard er vist på Figur 49, og det færdige print er vist på Figur 50.

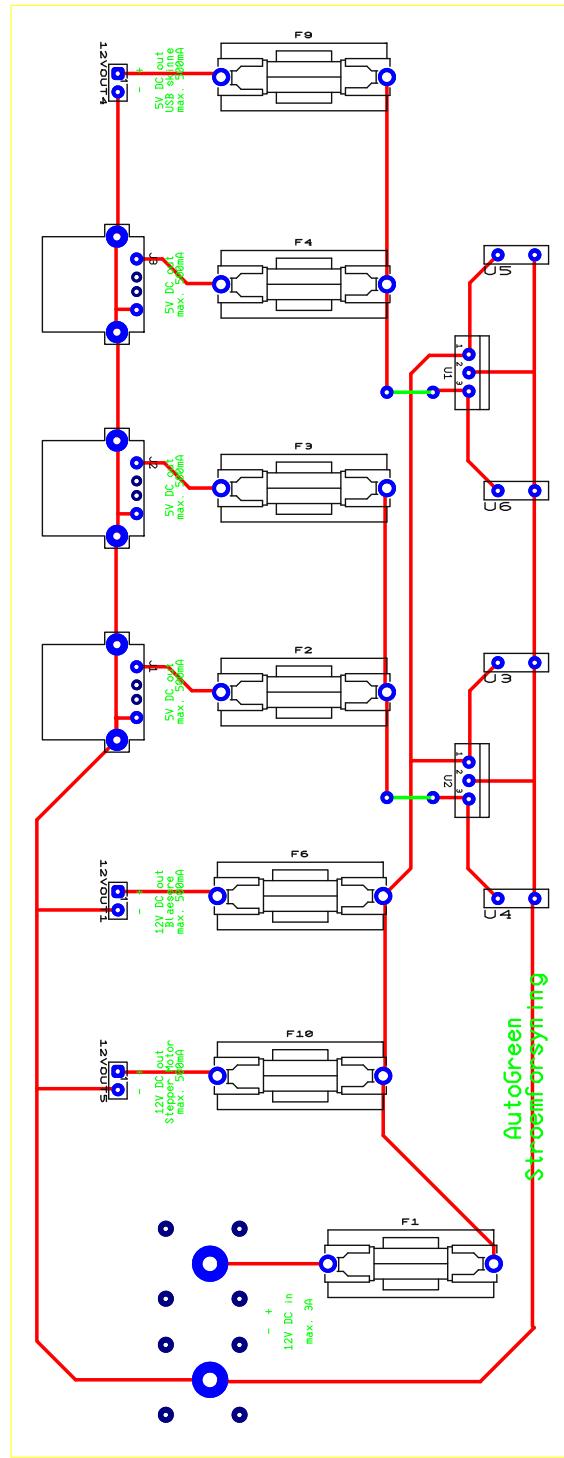


Figur 49: Printudlæg for Mosfet driver til Vinduesmotor i Ultiboard



Figur 50: Den færdige Mosfet driver til Vinduesmotor

7.4 Strømforsyning



Figur 51: Printudlæg for Strømforsyning i Ultiboard

Implementering af strømforsyning foretages ved at Multisim diagrammet eksporteres til Ultiboard, hvorefter printet udlægges. Det forsøges gjort således at printet er overskueligt, og med så få lus som overhovedet muligt. Som udgangspunkt er alle forbindelser på printet lagt på bagsiden. Det designede print er vist på Figur 51.

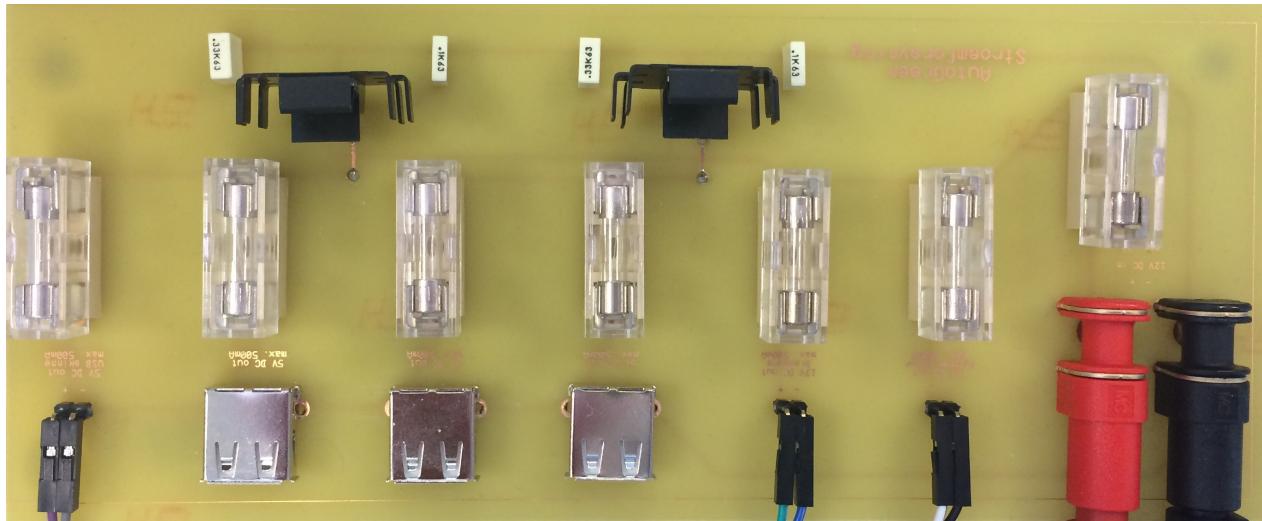
Kobber på undersiden af printet er vist med rød, mens kobber på oversiden af printet er vist med grøn. Kobberører er vist med blå. Der er en lille hage ved kobberørerne; der er ikke forbindelse mellem oversiden og undersiden. Derfor må man lodde et lille stykke monteringstråd igennem printet for at skabe forbindelse de stder hvor det er nødvendigt.

Omkredserne af komponenterne (sort) printes ikke, de vises kun som en slags hjælpelag.

Databasen i Ultiboard indeholder ikke komponenter til bananbøsninger, derfor er disse omkredse ikke med på figuren.

Der skrives lidt forklarende tekst på oversiden af printet, så man kan se hvad der skal kobles til hvor; dette er lidt svært at se på figuren.

Printudlægget bestilles ved E-LAB på IHA, hvorefter komponenter loddes på. Det færdige print er vist på Figur 52.

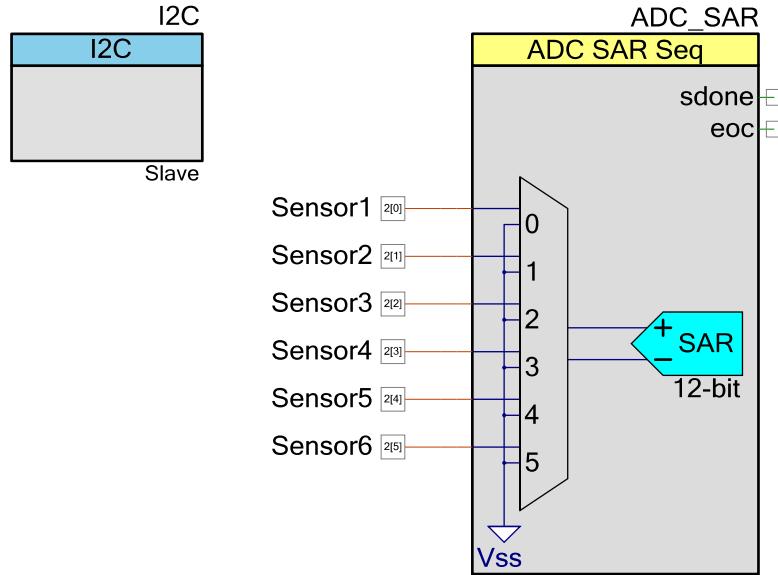


Figur 52: Den færdige Strømforsyning

7.5 Jordfugt

Dette afsnit beskriver implementering af SW og HW på PSoC4 i blokken Jordfugt.

7.5.1 HW PSoC4



Figur 53: TopDesign.cysch for PSoC4 i Jordfugt

Den HW, der syntetiseres i PSoC4 Jordfugt er vist på Figur 53.

Komponenten I2C styrer kommunikation med MasterPSoC via I²C . Den er konfigureret til at være slave med adressen 0x32, datarate er indstillet til 100 kbps.

ADC_SAR komponenten er en Analog to Digital converter med seks forskellige inputs. Den indeholder seks registre, som opdateres løbende, når den sættes til at køre. Referencespænding er konfigureret til VDDA, dvs. måleområdet er 0V - 3.3V. Den er desuden indstillet til 8 bit opløsning og en clock frekvens på 1 MHz, hvilket resulterer i en samplerate på 11904 SPS på hver sensor.

Alle pins i topdeignet er konfigureret til high impedans analog, for at undgå spændingsdeling med det øvrige kredsløb.

7.5.2 SW PSoC4

```

1 ...
2     for (;;)
3     {
4         checkForData();
5         if (i < 6)
6         {
7             data = ADC_SAR_GetResult16(i); //Hent de ønskede data
8             //Tjek om ønskede data er inden for grænser
9             if ((data & 0b10000000) || (data >= MAXIMUM) || (data < MINIMUM))
10            {
11                convertedData = 0b10000000; //Skriv fejl til konverteerde data
12            }
13            else //Konverter data til tal med værdi 1 – 100
14            {
15                convertedData = (100 - (((data - MINIMUM)*100)/(MAXIMUM-MINIMUM)));
16            }
17            readBuffer[0] = convertedData; //Gør data klar til aflæsning
18            I2C_I2CSlaveClearReadBuf(); //Reset read buffer pointer
19            i = 0xFFFFFFF;
20        }
21    }
22 }
23 void checkForData()
24 {
25     //Check for om der er modtaget data
26     if (I2C_I2CSlaveStatus() & I2C_I2C_SSTAT_WR_CMPLT)
27     {
28         //Put data i variabel i
29         i = writeBuffer[0] & 0b00000111;
30
31         I2C_I2CSlaveClearWriteBuf(); //Clear buffer pointer
32         I2C_I2CSlaveClearWriteStatus(); //Clear status
33     }
34 }
```

Listing 7.21: Udsnit af main.c for PSoC4 i Jordfugt

Filen main.c, hvoraf den vigtigste del er vist på Listing 7.21, fungerer jf. State Machinen på Figur 27 side 76.

Programmet tjekker om der er modtaget data på I²C , og opdaterer evt. indexvariablen i til den ønskede jordfugt index. Dette sker vha. funktionen checkForData().

Ud over at opdatere indexvariablen i, cleares pointeren i write bufferen ligeledes i checkForData(), så bufferen er klar til at modtage nye data.

I tilfælde af at indexvariablen i er blevet opdateret til et sensornummer (0-5), indlæses data fra jordfugtsensoren med det pågældende nummer. Såfremt at dataene ligger inden for grænseværdierne, vil disse data blive konverteret til et tal mellem 1 og 100. Dette tal vil herefter blive skrevet til read bufferen. I tilfælde af at dataene er uden for grænseværdierne, vil der blive skrevet en fejlværdi til read bufferen. Grænseværdierne er fastsat ud fra praktiske forsøg, således at helt tør jord giver en lav værdi, og gennemvædet jord giver en værdi tæt på 100. Dette er med til at fejsikre systemet; hvis en sensor ved en fejl er koblet fra, vil SAR'en måle en værdi højere end maximum, og der bliver skrevet en fejlværdi til read bufferen. Hvis en sensor er kortsluttet, vil SAR'en måle en værdi under minimum, og det samme vil ske.

Til slut opdateres indexvariablen i til en værdi, der ikke svarer til et sensornummer.

8 Software Implementering

8.1 Version

Dato	Version	Initialer	Ændring
7. maj	1	KT	Første udkast.

8.2 GUI - QT

Til at lave den grafiske brugerflade på Devkit8000 er brugt QT. QT bliver en let måde at skabe brugerflader på, og skabe forbindelser mellem denne brugerflade og det baglæggende system.

QT .ui filer

Til at designe menuer i softwaren, giver QTcreator(1) mulighed for at designe menuer, blot ved brug af drag and drop teknikker. Til design af vores menuer, tilpasser vi et canvas til størrelsen 480x256 pixels, som passer til til at dække hele skærmen på Devkittet. De mest anvendte funktionaliteter der er brugt til at designe den grafiske brugerflade, er knapper. Knapperne kan trækkes ind på det ønskede sted på canvaset, og teksten samt farven på knappen kan ændres. Ændring af farve på knapper kan f.eks. ses på hovedmenuen ved tryk på Monitorer og reguler. Knapper og objekter der bliver sat ind navngives. Dette navn er det der kan bruges til at lave signaler med, til at kommunikere rundt i systemet.

QT event driven programming

QT standarden, og den metode vi også bruger til at kode og designe brugerfladen i er event driven programming.(2) Når en knap trykkes, fortages der en række funktionaliter, som både kan have indflydelse på brugerfladen, men i de fleste tilfælde vil ændre noget bag facaden i grundsystemet. En af disse event der påvirker både brugerfladen og grundsystemet er når 'monitorer' tændes på hovedmenuen. Dette kald leder til et kald til indstillinger hvor monitor booleanen bliver sat til true. Dette vil sætte system igang med at monitorer drivhuset, og samtidig skifte knappens farve til grøn, svarende til at den er aktiv.

```

1 void MainWindow::on_btn_monitor_clicked()
2 {
3
4     if( !indstillinger.getRegulering() )
5     {
6         if( indstillinger.getMonitorering() ) //Toggle
7         {
8             //red
9             ui->btn_monitor->setStyleSheet("background-color: rgb(255, 0, 0)");
10            indstillinger.SetMonitorering(false);
11
12     } else
13     {
14         //green
15         ui->btn_monitor->setStyleSheet("background-color: rgb(0, 255, 0)");
16         indstillinger.SetMonitorering(true);
17     }
18 }
19 else
20 {
21     if( indstillinger.getMonitorering() ) //Toggle
22     {
23         //red
24         ui->btn_monitor->setStyleSheet("background-color: rgb(255, 0, 0)");
25         ui->btn_reguler->setStyleSheet("background-color: rgb(255, 0, 0)");
26         indstillinger.SetMonitorering(false);
27         indstillinger.SetRegulering(false);
28     }
29 }
30 }
31 }
```

Koden viser brugen af monitorer knappen på hovedmenuen. Når knappen trykkes køres koden vist. Der laves tjek med if statement på regulering fra indstillings-klassen, og hvis reguleringen er tændt, tjekkes der på monitorering fra indstillings-klassen. Hvis monitorering er tændt, sættes måde regulering og monitorering til false i indstillings-klassen, og farven på knapperne sættes til rød, svarende til værende slukket. Hvis regulering er slukket, tjekkes der også på monitorering. monitoereringen toggles, ved at tjekke om den er true eller false, hvis den er true (tændt), slukkes den ved brug af setMonitorering metoden fra indstillingsklassen, og farven på knappen skiftes til rød. Omvendt hvis den er slukket, sættes monitorering til true, og farven på knappen sættes til grøn. Farveskift fortages ved brug af QT-funktionalitet.

```
1 ui->btn_monitor->setStyleSheet("background-color: rgb(255, 0, 0);")
```

Der tilgåes UI'en og tilgår knappen btn_monitor og kalder methode setStyleSheet() på den for at skifte farve.

Menuhåndtering og menublokering

Der ønskes i brugerfladen at brugeren kan skift imellem flere menu, istedet for at skulle vise og gemme knapper væk. Dette gøres ved at lave flere .ui filer, som også har tilsvarende .cpp og .h filer, og kører eksekvere dem, som en menu ovenpå den forhenværende menu.

```
1 void MainWindow::on_btn_systemlog_clicked()
2 {
3     dialoge_systemlog systemlog;
4     systemlog.setModal(true);
5     systemlog.exec();
6 }
```

I kode-snippet ovenfor ses hvordan der oprettes et object af dialoge_systemlog, ved klik på systemlog knappen på hovedmenuen. Der sikres herefter af de forhenværende menuer ikke kan bruges før systemlog menuen igen er lukket ned. Dette gøres ved brug af setModal. setModal blokkere alt input til andre menuer end den aktive menu. Herefter eksekveres menues ved brug af exec metoden.

Trådhåndtering i QT

Message handling

Litteraturliste

- [1] Timll Technic Inc: *DevKit8000 brugermanual*. "Bilag 001 - DevKit8000 user manual_en". 2009.
- [2] Cypress Semiconductor: *PSoC 4 Pioneer Kit Guide*. "Bilag 002 - CY8CKIT-042 PSoC 4 Pioneer Kit Guide". 2013.
- [3] USB Implementers Forum, Inc.: *USB 2.0 Specification*. http://www.usb.org/developers/docs/usb20_docs/#usb20spec. 2015-03-17.
- [4] Honeywell International Inc.: *Datablad for Temperatur-/Luftfugtighedssensor*. "Bilag 003 - Datablad for temp_luftfugtsensor". August 2013.
- [5] Honeywell International Inc.: *I²C Communication with the Honeywell HumidIcon*. "Bilag 004 - I2C Communication with the Honeywell HumidIcon". June 2012.
- [6] Intersil: *Datablad for lyssensor*. "Bilag 005 - Datablad for lyssensor". Februar 2008.
- [7] Motorola, Inc.: *Three-Terminal Positive Voltage Regulators*. "Bilag 006 - Datablad for LM7805". Maj 1996.
- [8] Maxim: *Digital Temperature Sensor and Thermal Watchdog with 2-Wire Interface*. "Bilag 008 - Datablad for LM75". 2009.
- [9] SAIA-Burgess Electronics: *Motor Products*. "Bilag 007 - Motor Products". 2000.
- [10] PRJ3 F15 Gruppe 9: *PSoC Master source code*. https://github.com/PhKP/PSoC_Master. 2015.
- [11] PRJ3 F15 Gruppe 9: *MSE Øvelse 6*. "Bilag 009 - MSE Øvelse 6". Maj 2015.