

Projektdokumentation
AutoGreen
Gruppe 9

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

27. 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	iii
Arbejdsopgaver	vi
1 Projektformulering	1
1.1 Beskrivelse	2
1.2 MoSCoW prioritering	3
1.3 Rigt Billede	4
2 Kravspecifikation	5
2.1 Ordforklaring	6
2.2 Systembeskrivelse	7
2.3 Brugerfladen	10
2.4 Aktør Kontekst Diagram	10
2.4.1 Aktørbeskrivelser	11
2.5 Funktionelle Krav	12
2.6 Ikke Funktionelle Krav	13
2.7 Use Case Diagram	14
2.7.1 Use Case beskrivelser - Initiering og Formål	15
2.7.2 Use Case Beskrivelser - Fully Dressed	17
3 Systemarkitektur	27
3.1 Indledning	27
3.2 Hardwarearkitektur	27
3.2.1 BDD for System	28
3.2.2 IBD'er for System	29
3.2.3 IBD for Aktuator	32
3.2.4 IBD for Jordfugt	33
3.2.5 Signalbeskrivelser	34
3.3 Softwarearkitektur	35
3.3.1 Applikationsmodel	35
3.3.2 Controller-Klasser	35
3.3.3 Boundary-Klasser	35
3.3.4 Domain-Klasser	36
3.3.5 Menuoversigt	37
3.3.6 Menubeskrivelse	37
3.4 Protokol for UART	39
3.4.1 UART indstillinger	39
3.4.2 Datavalidering	39
3.4.3 Kommandoer	40
4 Hardware Design	43
4.1 I ² C Protokol	44
4.1.1 Temperatursensor	44
4.1.2 Slave Aktuator	45
4.1.3 Slave Jordfugt	47
4.2 PSoC Master Design	48

4.2.1	Klassebeskrivelser	49
4.2.2	Sekvensdiagrammer	62
4.3	Breakoutboards	64
4.4	Aktuator Design	65
4.4.1	Varmelegeme.....	65
4.4.2	Blæsere	67
4.4.3	Vinduesmotor.....	69
4.4.4	PSoC4	71
4.4.5	Drivers til PSoC4.....	72
4.5	Strømforsyning Design	73
4.6	Jordfugt Design	75
5	Software Design	77
5.1	Indledning	77
5.2	Udvidet Applikationsmodel: Sekvensdiagrammer	78
5.2.1	Usecase 1: Start	78
5.2.2	Usecase 2: Stop	79
5.2.3	Usecase 4: Administrer Drivhus	80
5.2.4	Usecase 5: Vis Historik.....	83
5.2.5	Usecase 6: Admininstrerer Plantedatabase	84
5.2.6	Usecase 7: Konfigurer System	85
5.2.7	Usecase 8: Se Systemlog	89
5.2.8	Usecase 9: Rapportering	90
5.2.9	Usecase 10: Monitorering	91
5.2.10	Usecase 11: Regulering.....	92
5.3	Klassebeskrivelser.....	93
5.3.1	Datalog.....	93
5.3.2	DoublyLinkedList	95
5.3.3	Indstillinger	98
5.3.4	Monitor	103
5.3.5	Plantedatabase	104
5.3.6	Rapport	106
5.3.7	Regulator	107
5.3.8	SystemLog	109
5.3.9	UART	110
5.4	Trådhåndtering	112
6	Hardware Implementering	113
6.1	PSoC Master	114
6.1.1	Main implementering	114
6.1.2	I ² C implementering	115
6.1.3	UART implementering.....	116
6.1.4	DSP implementering	117
6.1.5	Controller implementering	119
6.2	Aktuator	121
6.2.1	HW PSoC4	121
6.2.2	SW PSoC4	122
6.2.3	HW Varmelegeme	129
6.2.4	HW Blæsere	131

6.2.5	HW Vinduesmotor	132
6.3	Strømforsyning.....	133
6.4	Jordfugt	135
6.4.1	HW PSoC4	135
6.4.2	SW PSoC4	136
7	Software Implementering	137
7.1	System structs	138
7.1.1	Sensordata	138
7.1.2	Date	138
7.1.3	Plant	138
7.2	DoublyLinked List	139
7.2.1	Valg af datastruktur	139
7.2.2	Forklaring af Doubly Linked List	139
7.2.3	HeadInsert	140
7.2.4	GetItemInList	140
7.2.5	DeleteAt.....	141
7.2.6	PeekHead	142
7.3	Indstillinger	143
7.3.1	SetDate.....	143
7.3.2	Exec af linux kommandoer	144
7.3.3	GetDate	144
7.3.4	Get- og set-metoder	145
7.4	Datalog	145
7.4.1	Lagering af data i datalogen	145
7.4.2	GetNewestData	146
7.5	SystemLog	146
7.6	UART	148
7.6.1	UART opsætning.....	148
7.6.2	UART Connect	149
7.6.3	UART senddata	149
7.6.4	UART recievedata.....	150
7.6.5	UART getSensor.....	150
7.6.6	UART activateSensor	150
7.6.7	UART getSensorData	151
7.6.8	Fejlsikring i UART	152
7.7	Monitor	153
7.8	Regulator	154
7.8.1	Regulator constructor	155
7.8.2	loadData	155
7.8.3	Run metoden	156
7.8.4	ControlData metoden	156
7.9	GUI - QT	158
7.9.1	QT .ui filer	158
7.9.2	QT event driven programming	158
7.9.3	Menuhåndtering og menublokering	159
7.9.4	Trådhåndtering i QT	160
7.9.5	Timers til opdatering	160
7.9.6	System sammensætning	160

7.10 Build tools	161
8 Accepttest	163
8.1 Funktionelle Krav	164
8.2 Ikke-funktionelle krav	174
Litteraturliste	176

Arbejdsopgaver

Under projektarbejdet har arbejdsopgaver været fordelt efter følgende tabel:

	Kristian Søgaard Sørensen	Kasper Torp Samuelsen	David Erik Jensen	Henrik Bagger Jensen	Lasse Barner Sivertsen	Philip Krogh-Pedersen	Kristian Thomsen	Morten H. Gormsen
Projektformulering	X	X	X	X	X	X	X	X
Kravspecifikation	X	X	X	X	X	X	X	X
Systemarkitektur	X	X	X	X	X	X	X	X
HW Design og impl. - I ² C Protokol				X	X	X	X	X
HW Design og impl. - PSoC Master					X	X	X	
HW Design og impl. - Slave Aktuator				X				X
HW Design og impl. - Slave Jordfugt					X			X
HW Design og impl. - Strømforsyning				X				X
SW Design	X	X	X					
SW Impl. - Build Tools				X				
SW Impl. - Doubly Linked List				X				
SW Impl. - DevKit8000 UART			X					
SW Impl. - Indstillinger				X				
SW Impl. - Monitor	X							
SW Impl. - SystemLog	X							
SW Impl. - Regulator			X					
SW Impl. - Brugerflade	X	X	X					
Accepttest	X	X	X	X	X	X	X	X
Rapport	X	X	X	X	X	X	X	X

1 Projektformulering

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.
16. maj	4	MHG	Mindre rettelser.

1.1 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 erfarte 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]

1.2 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.

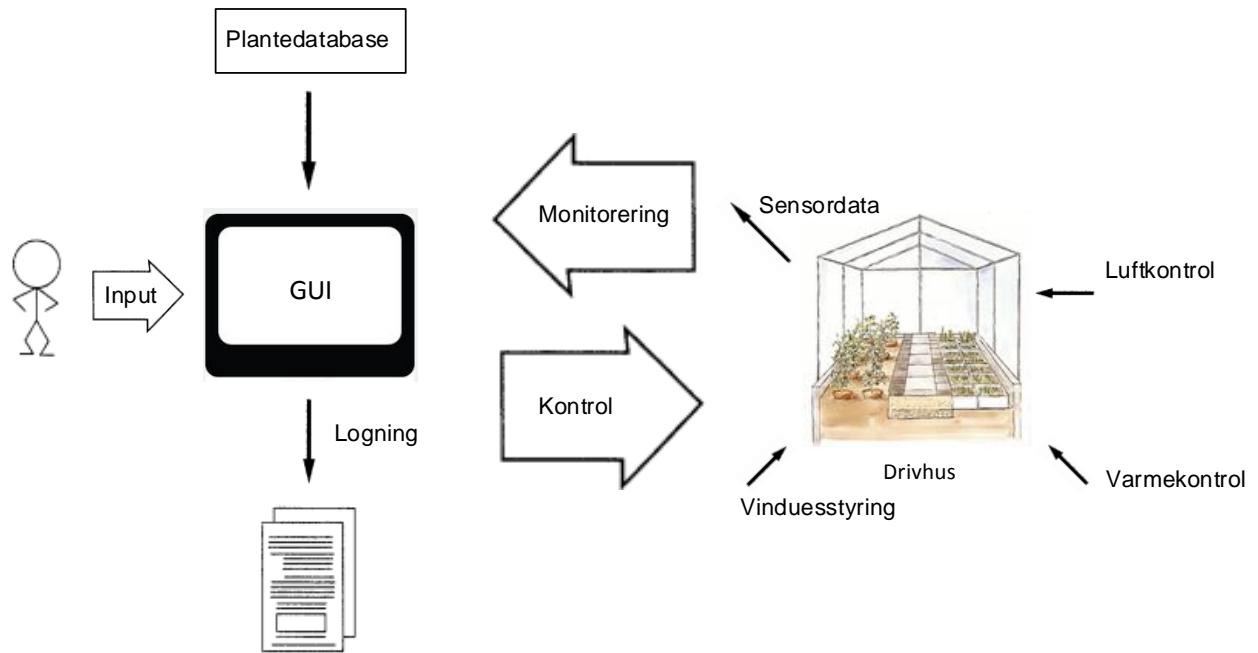
- **Systemet kan:**

- 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.

1.3 Rigt Billede



Figur 1: AutoGreen Automatiseret Drivhus

2 Kravspecifikation

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.
16. maj	4	MHG	Mindre rettelser.

2.1 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

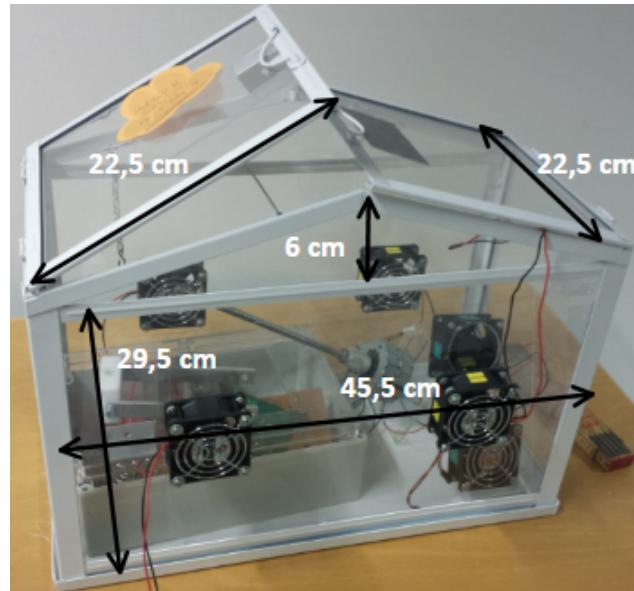
Dette er en E-mail, som brugeren kan vælge at få tilsendt. Den sendes, hvis en parameter i det fysiske drivhus kommer uden for tolerancen af den ønskede værdi.

API

Application Programming Interface.

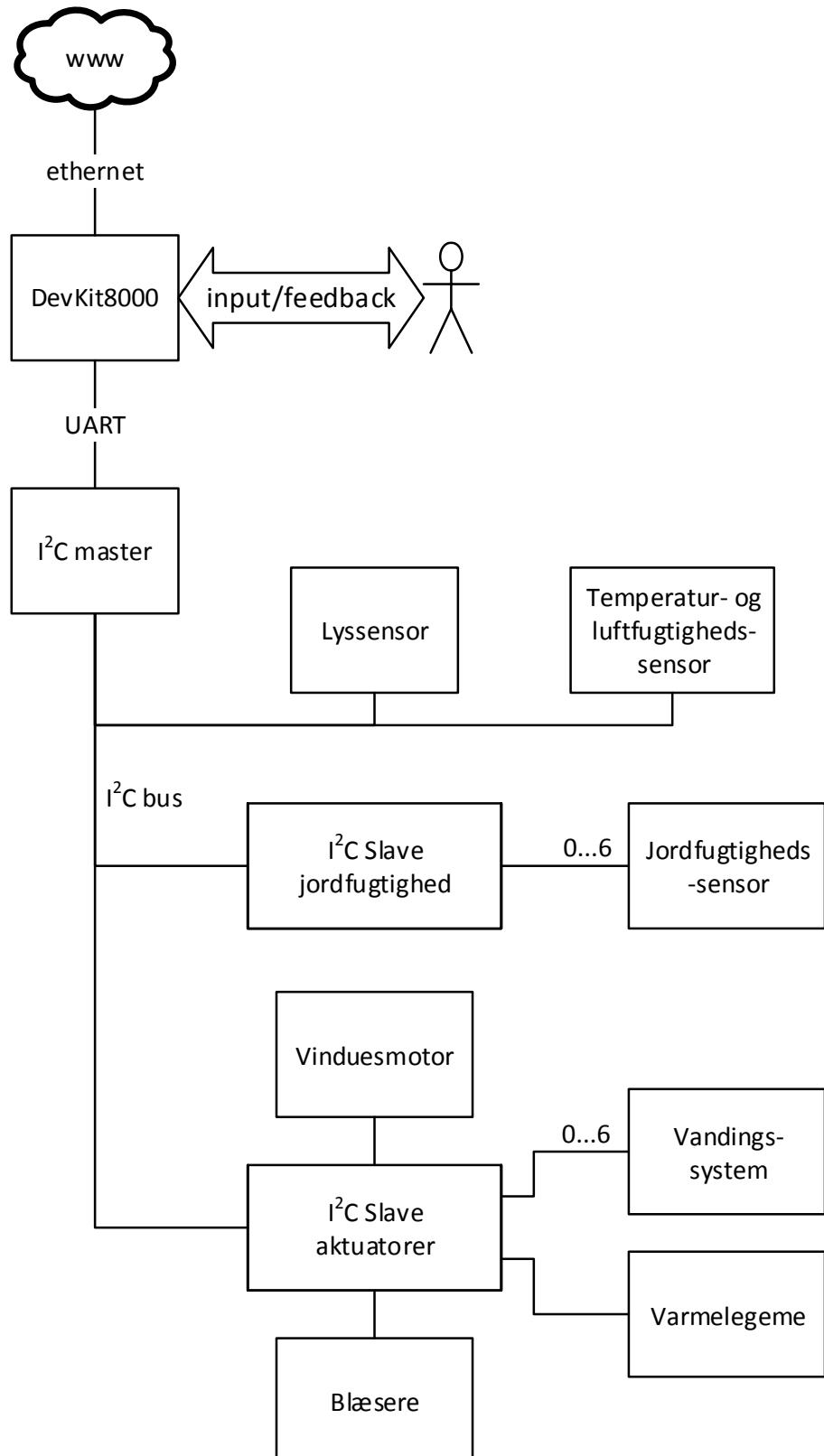
2.2 Systembeskrivelse

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 Slave, hvorefter respons sendes retur til DevKit8000.

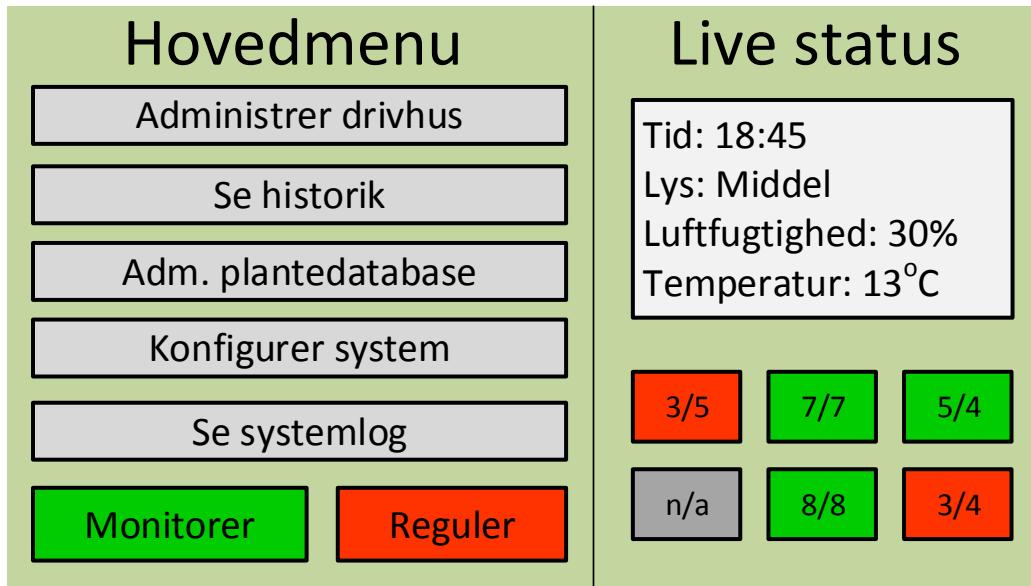
I²C Slave Jordfugtighed

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 Brugerfladen



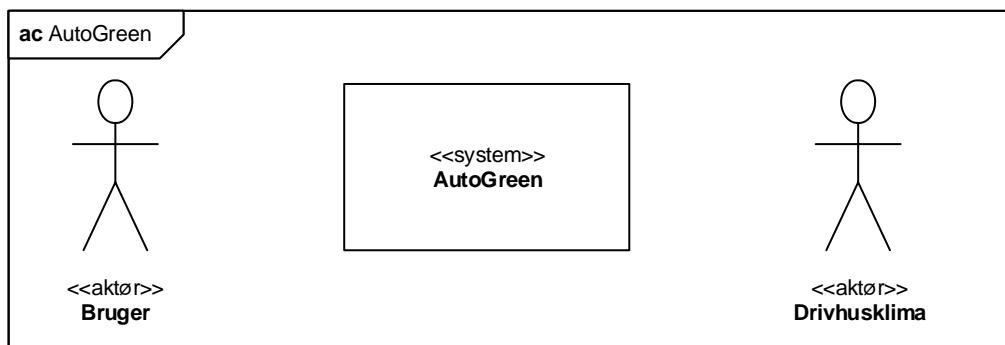
Figur 4: Skitse af hovedmenuen på brugerfladen.

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.

2.4 Aktør Kontekst Diagram



Figur 5: Aktør Kontekst Diagram for AutoGreen

2.4.1 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

2.5 Funktionelle Krav

Systemet...

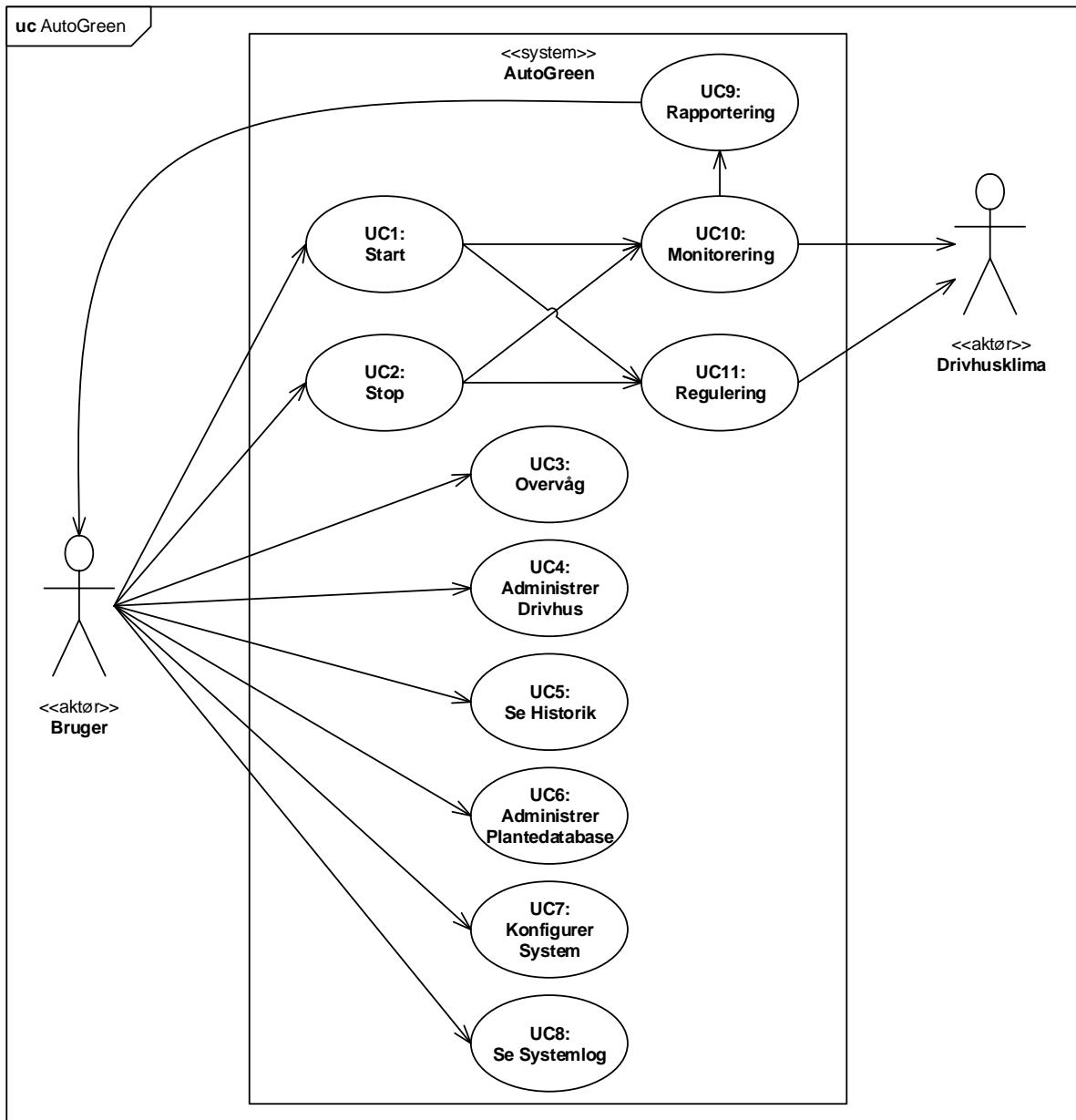
1. ... *Skal* give brugeren mulighed for at monitorere og konfigurere drivhusklimaet vha. en grafisk brugerflade på et touch display.
2. ... *Skal* have mulighed for at starte og stoppe systemet.
3. ... *Skal* måle lufttemperatur i det fysiske drivhus.
4. ... *Skal* kunne regulere temperatur i det fysiske drivhus.
5. ... *Skal* kunne indstilles til brugerdefineret tid og dato.
6. ... *Skal* kunne give brugeren mulighed for at vælge brug af varmelegeme og ventilatorer.
7. ... *Skal* give brugeren mulighed for at tilføje en plante i det virtuelle drivhus.
8. ... *Skal* give brugeren mulighed for at fjerne en plante i det virtuelle drivhus.
9. ... *Skal* give brugeren 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 brugeren mulighed for at se en systemlog over hændelser i systemet.
18. ... *Bør* gemme alt monitorering i en data log.
19. ... *Kan* give brugeren mulighed for at redigere og slette planter i plantedatabasen, som brugeren selv har tilføjet.
20. ... *Kan* give brugeren mulighed for at tilføje/redigere/slette e-mail adresser.
21. ... *Kan* give brugeren mulighed for valg af varslings e-mail omhandlende dårligt klima og daglig e-mail.
22. ... *Kan* sende e-mail til brugeren, på baggrund af brugerindstillinger.

2.6 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 højst 30 minutter ved en starttemperatur der ligger højst 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 højst 1 minut efter et for lavt jordfugtighedsniveau er målt, hvis den er indstillet til dette.

2.7 Use Case Diagram



Figur 6: Use Case Diagram for AutoGreen

2.7.1 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 interface's 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: Administrer 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: Administrer 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 minutten.

UC11: Regulering

Initieres af: UC1 Start.

Denne Use Case regulerer temperaturen i drivhuset, vha. vinduesåbner, varmelegeme og luftcirculation, med mindre brugeren har slævt 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.

2.7.2 Use Case Beskrivelser - Fully Dressed

For alle Use Cases hvor brugeren navigerer 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.
Initiering:	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.
Initiering:	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.
Initiering:	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:	Bruger
Aktører:	Bruger (primær)
Reference:	Ingen
Antal samtidige forekomster:	Én
Forudsætning:	Systemet er operationelt og hovedmenuen vises.
Resultat:	Brugeren har tilføjet, redigeret og/eller fjernet plante i plantedatabasen. Systemet viser Hovedmenuen.
Hovedscenarie:	<ol style="list-style-type: none"> 1. Bruger trykker "Administrer Plantedatabase" i hovedmenu. 2. System viser "Plantedatabasemenu". 3. Bruger trykker på "Tilføj Data". <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. Systemet opretter en plante med standardparametre og præsenterer "Databaseredigeringsmenu". 5. Bruger redigerer ønskede parametre og trykker "OK". 6. Systemet gemmer brugerens valg og viser "Plantedatabasemenu". 7. Bruger trykker "Tilbage". 8. UC6 afsluttes og systemet viser Hovedmenuen.
Alternativ:	<p>[Alt 3.a : Bruger trykker på en plante, der skal fjernes.]</p> <ol style="list-style-type: none"> 4. System viser "Databaseredigeringsmenu". 5. Bruger 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 : Bruger 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. Bruger indtaster op til tre ønskede E-mail adresser og trykker "OK". 6. Systemet gemmer E-mail adresserne i konfigurationsfilen. Systemet viser "Konfigurationsmenu". 7. Bruger 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. Bruger indtaster ønskede indstillinger for notifikationer. 6. Bruger 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. Bruger 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. Bruger vælger blæser on/off og/eller varmelegeme on/off. 6. Bruger 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.
Initering:	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.
Initering:	UC10: Monitorering
Aktører:	Bruger
Reference:	UC10: Monitorering
Antal samtidige forekomster:	Én
Forudsætning:	UC10 er aktiv, systemet er operationelt og notifikations- og advarselesemail 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.
Initering:	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"> Systemet indlæser konfigurationsfilen. Systemet sammenligner nyeste værdier for jordfugtighed fra data loggen med ønskede værdier fra det virtuelle drivhus. Jordfugtværdierne ligger inden for tolerancerne. <ul style="list-style-type: none"> [Ext 3.a : En eller flere jordfugtværdier ligger under tolerancen.] Systemet sammenligner nyeste værdi for temperatur fra data loggen med angiven værdi i det virtuelle drivhus. 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.] 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"> Systemet aktiverer aktuator for vanding for den eller de planter der er under tolerancen. Systemet fortsætter fra pkt. 4 i hovedscenariet. <p>[Ext 5.a : Værdien for temperatur ligger over tolerancen.]</p> <ol style="list-style-type: none"> Systemet regulerer temperaturen nedad jf. konfigurationsfilen. Systemet fortsætter fra pkt. 6 i hovedscenariet. <p>[Ext 5.b : Værdien for temperatur ligger under tolerancen.]</p> <ol style="list-style-type: none"> Systemet regulerer temperaturen opad jf. konfigurationsfilen. Systemet fortsætter fra pkt. 6 i hovedscenariet.

Tabel 11: UC11: Regulering

3 Systemarkitektur

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 PSoC Master.
1. maj	5	HBJ	Rettet signalbeskrivelse for signaltypen Analog. Passer nu til jordfugt blokken.
16. maj	6	MHG	Tilføjet jordfugtsensorer på IBD for forsyning. Mindre rettelser.
18. maj	7	DJE	UML diagrammet manglede en association fra regulator til indstillinger. Samt mindre rettelser i regulator, rapport og indstillinger.

3.1 Indledning

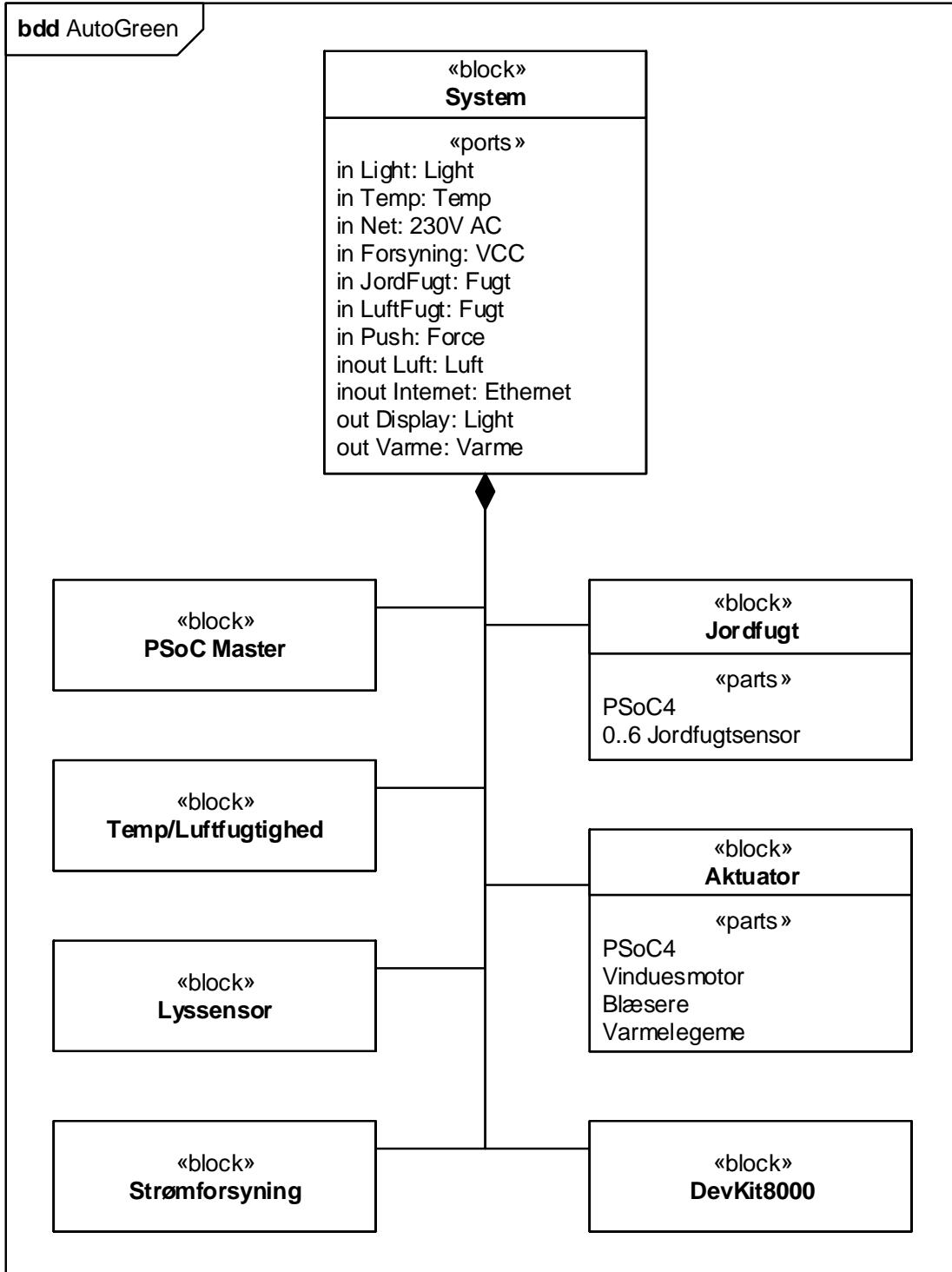
I dette kapitel vil systemarkitekturen for AutoGreen være opdelt i to underdele, hhv. for hardware og for software. Formålet med kapitlet er at gøre systemets grænseflader, både interne og eksterne, klare ift. signaltyper, niveauer og softwaregrænseflader.

3.2 Hardwarearkitektur

Dette afsnit beskriver arkitektur for hardware i AutoGreen.

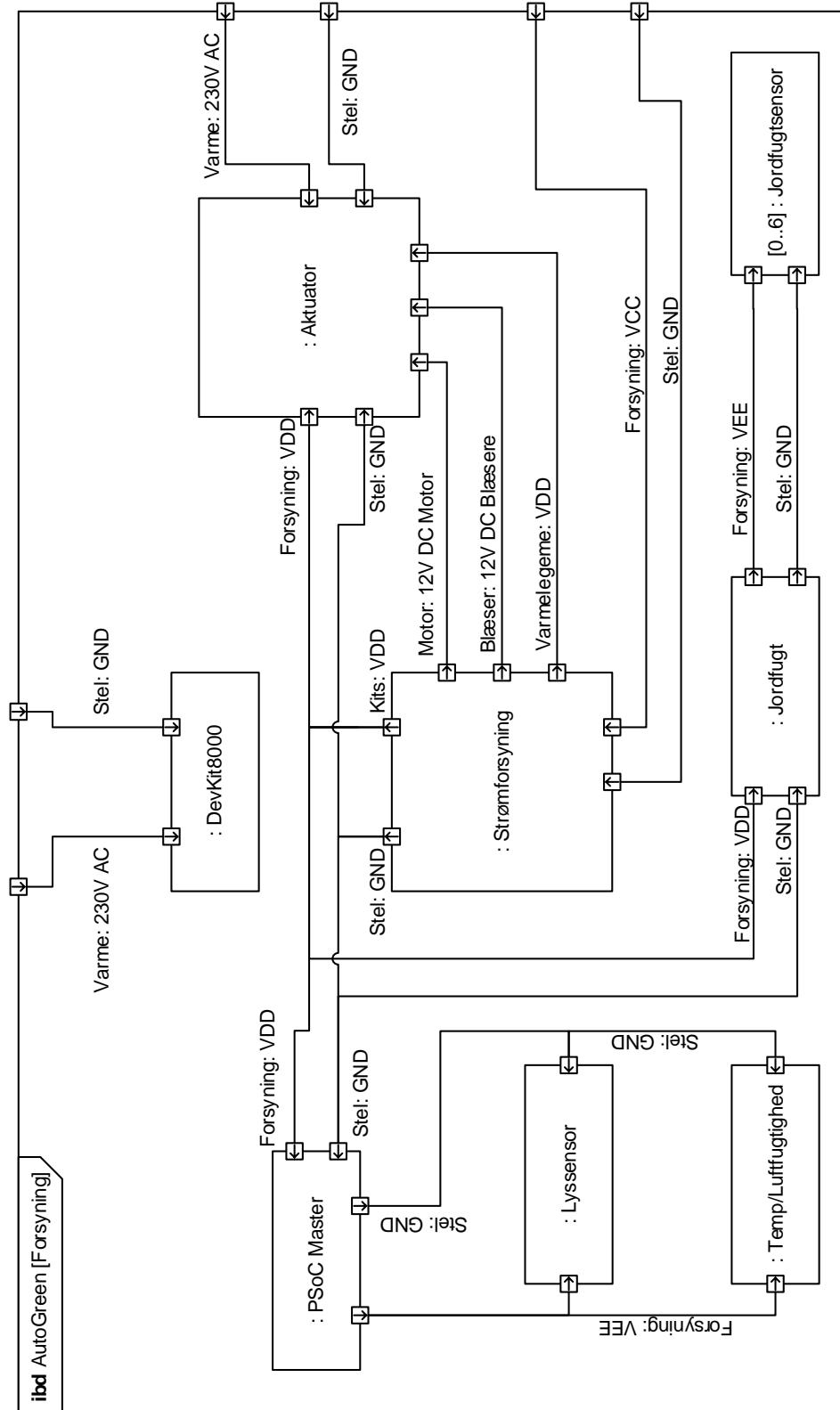
Forsyning til alle blokke er beskrevet på IBD for system, Figur 8. Forsyninger er ikke tegnet ind på øvrige diagrammer for overskuelighedens skyld. Det gælder desuden at alle blokke har fælles reference (GND).

3.2.1 BDD for System

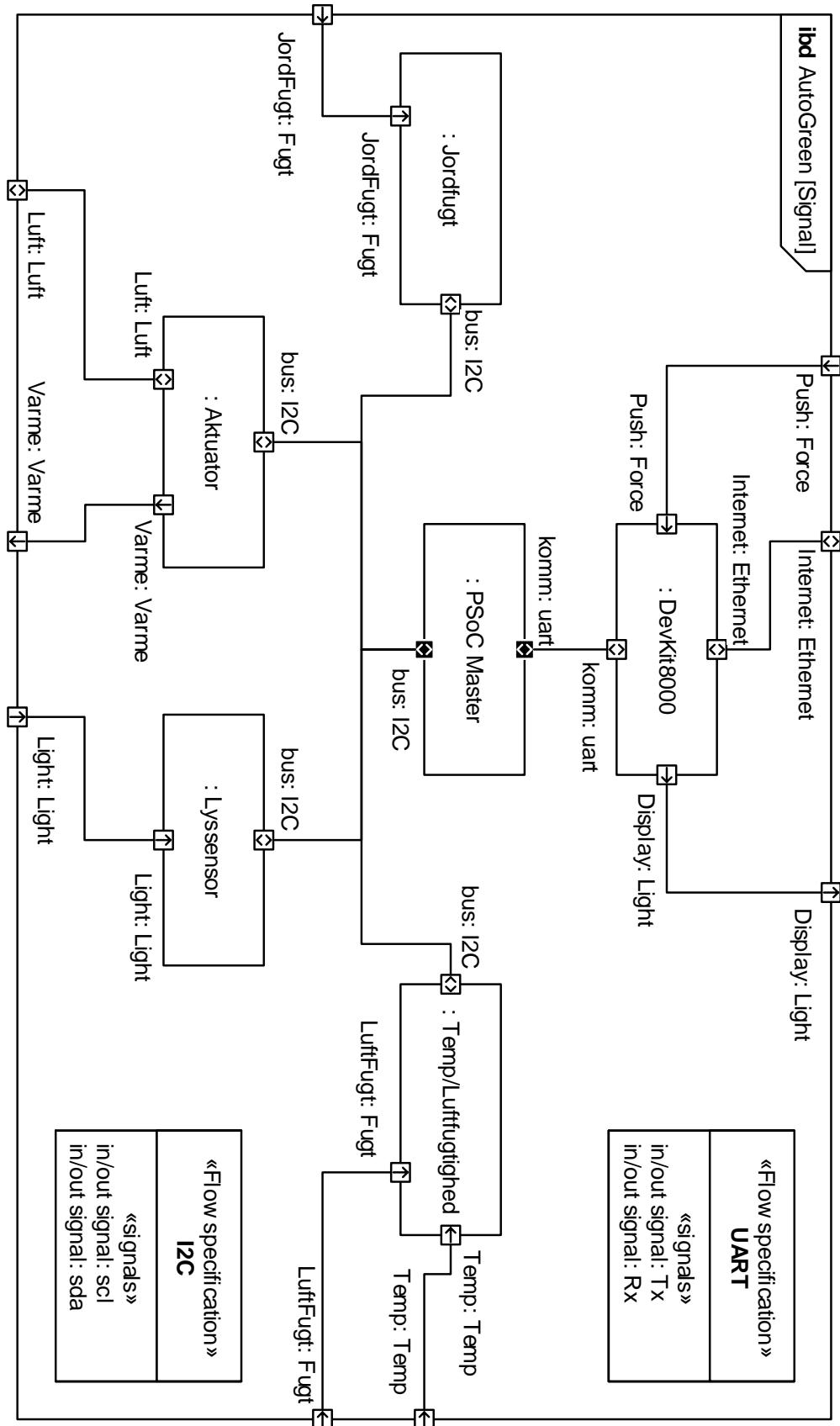


Figur 7: BDD for System.

3.2.2 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 sensorer. Blokken forsynes fra en laboratorieforsyning.

DevKit8000

Systemets brugerflade, er samtidigt controller for systemet.

PSoC Master

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 måler temperatur og luftfugtighed i det fysiske drivhus.

Lyssensor

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

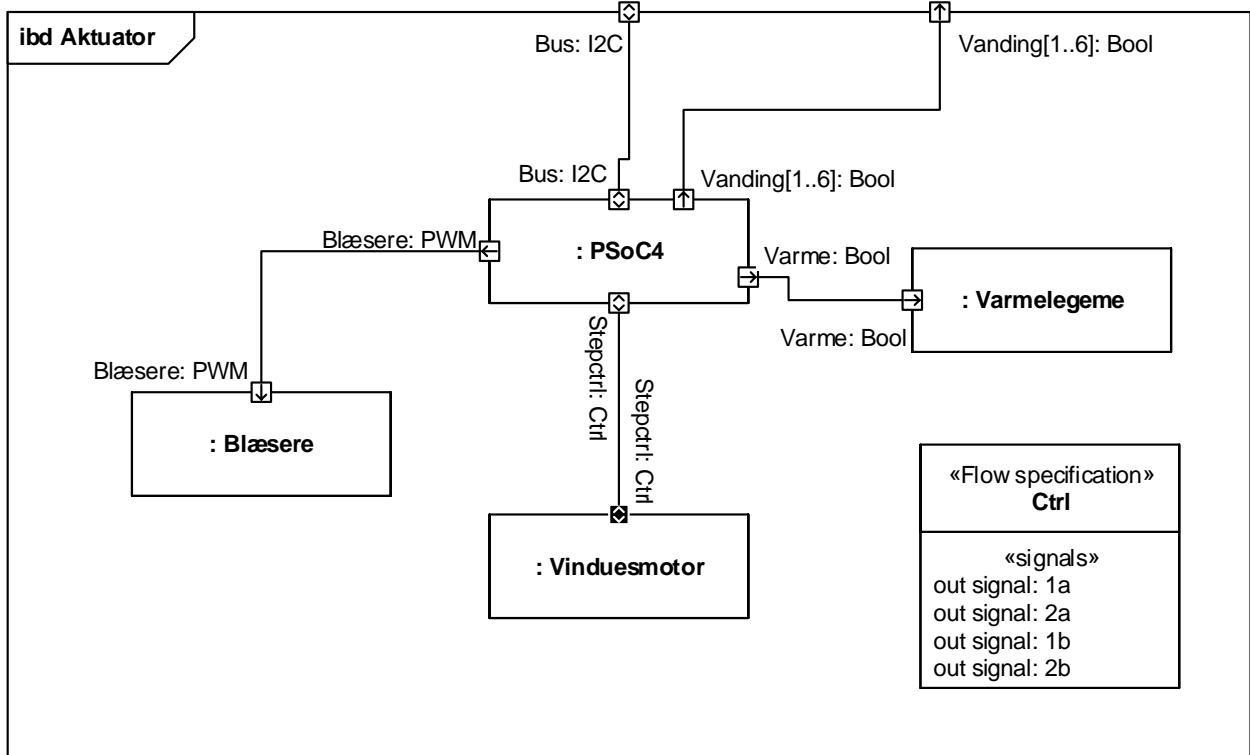
Jordfugt

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.

3.2.3 IBD for Aktuator



Figur 10: 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.

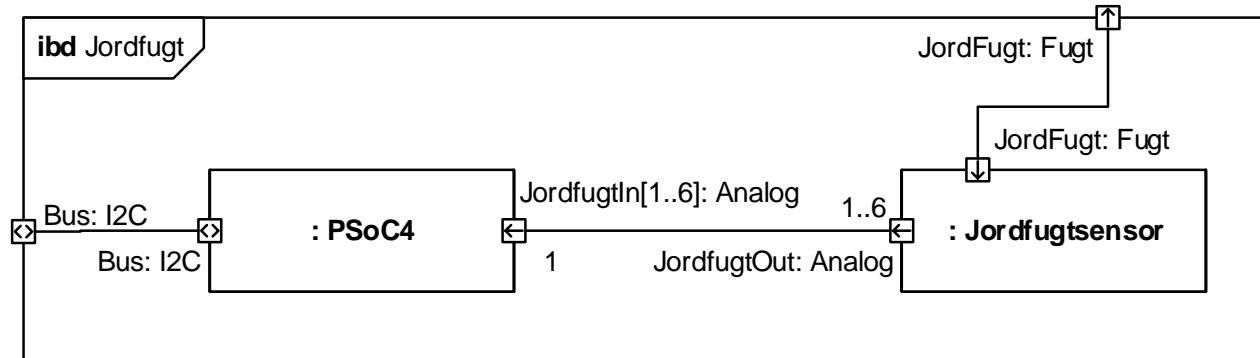
Varmelegeme

Varmelegemet 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 fire blæsere, som kan ventilere luften i det fysiske drivhus. Blæserne styres af PSoC4, og de forsynes fra Strømforsyning.

3.2.4 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 mäter jordfugt ved en plante i det fysiske drivhus. Den kan kobles op til seks af disse til PSoC4.

3.2.5 Signalbeskrivelser

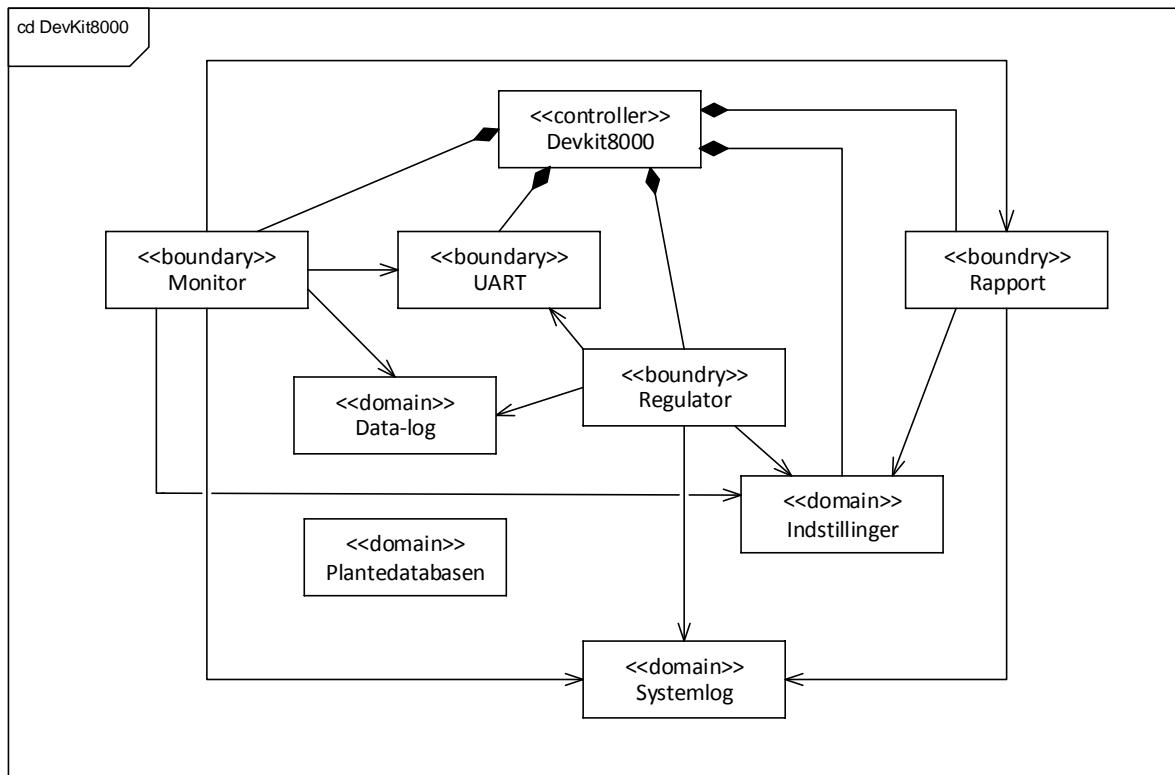
Signaltyp	Funktion	Tolerancer	Kommentar
VCC	Forsyning til strømforsyning	$12V \pm 0,25V$ 3A max.	Lab.forsyning
VDD	Forsyning til alle PSoC4 Pioneer Kits.	$5V DC \pm 0,15V$, 0.5A max	-
VEE	Forsyning til sensorer	$3.3V DC \pm 0.1V$, 0.1A max	-
12V DC Blæsere	Forsyning til blæsere.	$12V DC \pm 0,25V$, 140mA max.	-
12V DC Motor	Forsyning til vinduesmotor.	$12V \pm 0,25V$, 500mA max.	-
230V AC	Forsyning til varmelegeme og DevKit8000.	$230V AC \pm 10\%$, 50 Hz, 0.3A max	-
Analog	Analogt målesignal fra jordfugtmåler.	$0-3.3V \pm 0.1V$	-
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. 0 svarer til 0% og 255 svarer til 100%

Tabel 12: Beskrivelse af signaler.

3.3 Softwarearkitektur

3.3.1 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

3.3.2 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.

3.3.3 Boundary-Klasser

Monitor

Monitorklassens primære opgave er at opsamle sensordata fra UART klassen og skrive dem til datalogen. 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, samt indstillinger som indeholder de virtuelle planter 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 rapporter 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 hvilken slags E-mails, der skal benyttes. Rapportering skal sende E-mail til brugeren dagligt, når der er kritisk klima i drivhuset, eller både dagligt og ved kritisk klima.

3.3.4 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

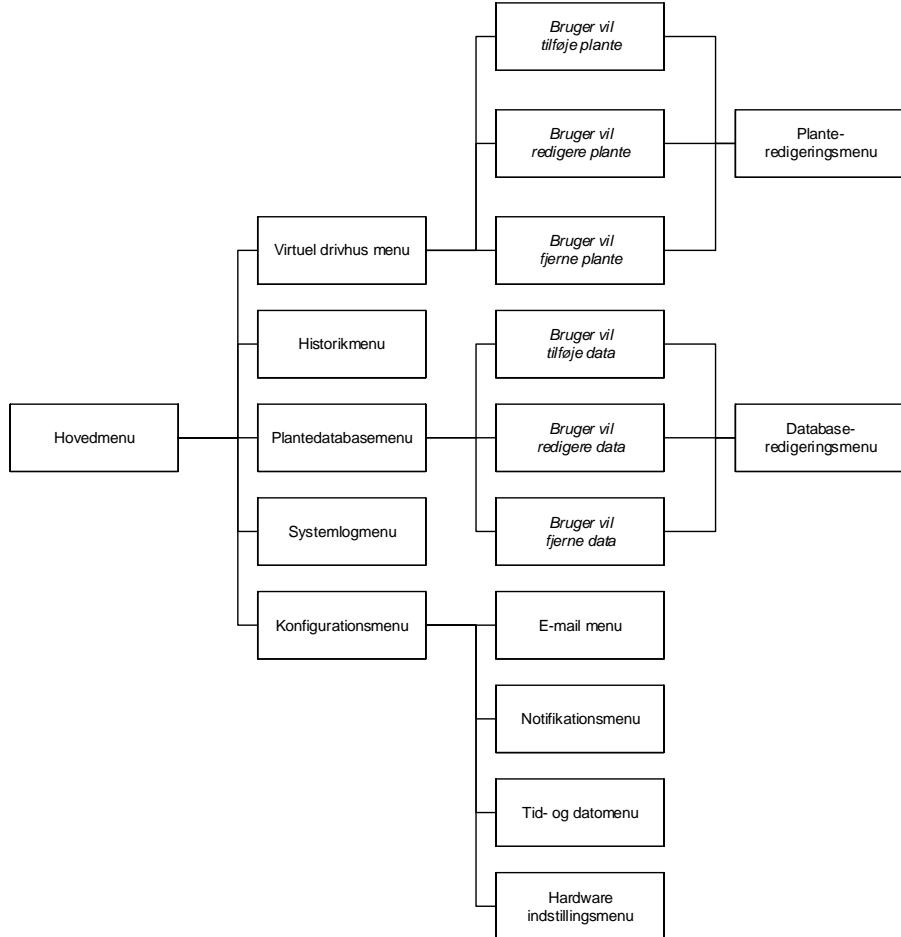
Indstillinger gemmer konfigurationer og indlæser dem i konfigurationsfilen, når regulering eller rapportering startes af brugeren. Den holder gemmer også de virtuelle planter og hvilken hardware der må bruges til at regulere temperaturen med. Der ud over bruges den også til at indstille og hente tiden i systemet.

Plantedatabasen

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

3.3.5 Menuoversigt

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



Figur 13: Oversigt over AutoGreen's menuer

3.3.6 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.

3.4 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.

3.4.1 UART indstillinger

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

3.4.2 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.

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.

3.4.3 Kommandoer

Kommando (DevKit til PSoC Master)	Svar (PSoC Master til DevKit)	Beskrivelse	Bitmønster
RequestTemp		Forespørgsel om data fra temperatursensor.	'T'
	Temp	Temperaturværdi fra sensor.	'T' efterfulgt af char med decimalværdi 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 temperatursensor.	'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.
RequestAir-Hum		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.
RequestSoil-Hum		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.
	WindowStatus	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 vandingssignaler.	'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 13: Kommando liste for UART kommunikation

4 Hardware Design

Version

Dato	Version	Initialer	Ændring
30. marts	1	PKP	PSoC Master use case diagram og klassebeskrivelser tilføjet.
30. marts	2	MHG	I2C Protokol og design af Aktuator.
8. april	3	MHG	Mindre rettelser i Aktuatordesign.
10. april	4	MHG	Skrevet det sidste (varmelegeme) design i aktuator blokken.
23. april	5	MHG	Lavet mindre rettelser i et par diagrammer.
27. april	6	MHG	Skrevet design for Strømforsyning.
27. april	7	PKP	Design for PSoC master rettet.
1. maj	8	MHG	Skrevet design for Slave Jordfugt.
13. maj	9	PKP	I ² C protokol opdateret med ny temperatursensor og PSoC Master funktionsbeskrivelser opdateret.
21. maj	10	LBS	Tilføjet breakoutboard afsnit.

4.1 I²C Protokol

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

4.1.1 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 14: 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

4.1.2 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 PSoC Master. Under dette afsnit vil protokolen mellem Slave Aktuator og PSoC Master 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 15: 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 16: 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 17: 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 18: 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 19: 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 20: I²C Kommando ReadStatus

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

4.1.3 Slave Jordfugt

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

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

Tabel 21: 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 22: I²C Kommando WriteDesiredSensor

PSoC Master 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.

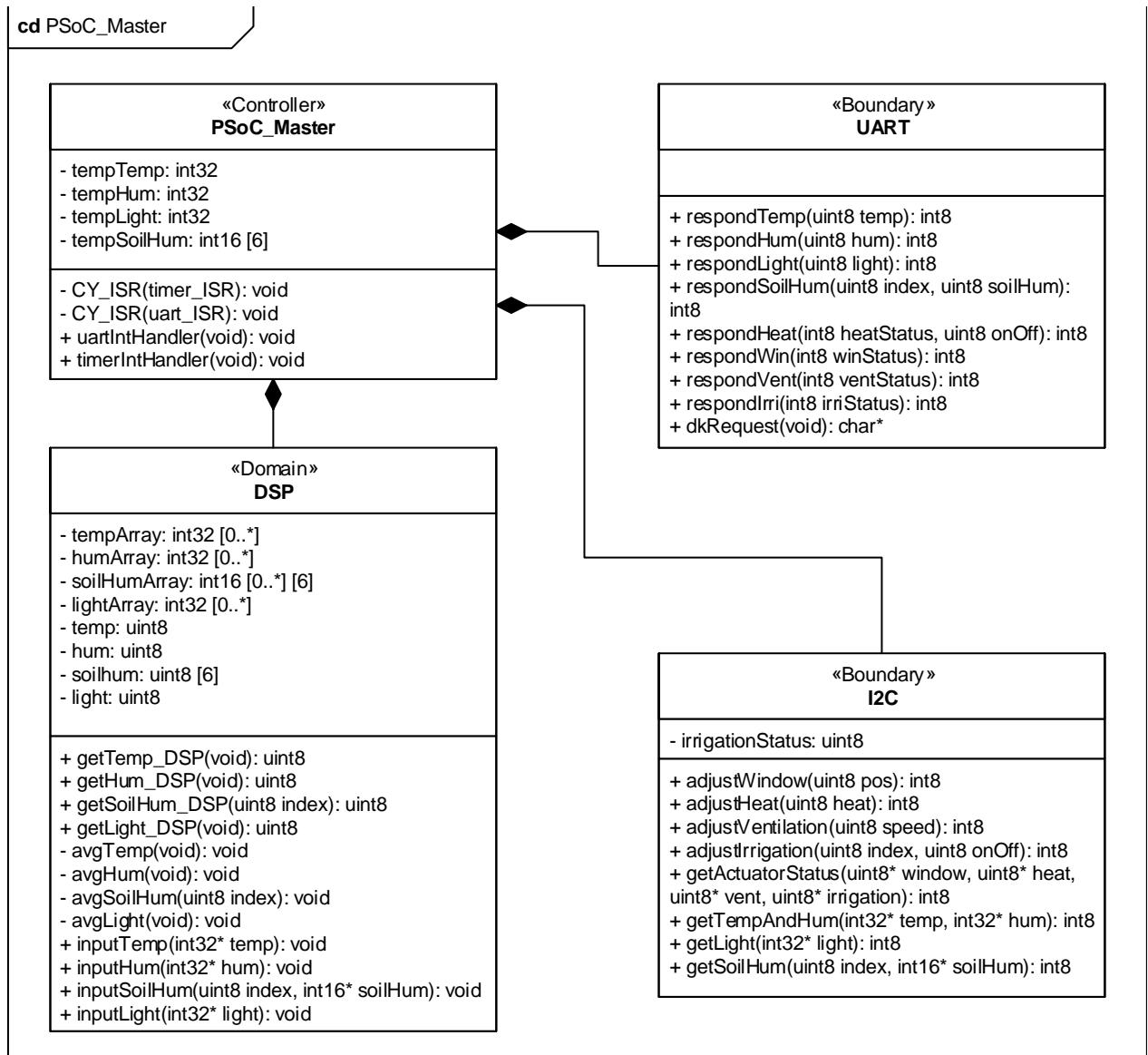
S7	S6	S5	S4	S3	S2	S1	S0
Sensor status, 0x1 = deaktive, 0x0 = aktive							Sensor værdi, 0x1 - 0x64 (1 - 100)

Tabel 23: I²C Kommando ReadDesiredSensor

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

4.2 PSoC Master Design

På Figur 15 ses klassediagrammet for PSoC Master. 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 domænekklasse, som indeholder alle de målte dataværdier, der er modtaget af sensorerne.



Figur 15: Klassediagramm für PSoC Master

4.2.1 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 24: 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 25: 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 26: 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 27: timerIntHandler()

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

Tabel 28: uartIntHandler()

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 29: 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 30: 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 31: 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 svarer 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 32: respondSoilHum

Prototype	<code>int8 respondHeat(uint8 heatStatus, uint8 On)</code>
Parametre	<p><code>uint8 heatStatus</code> Returværdien fra funktionen adjustHeat i boundaryklassen I²C ; fortæller om kommunikationen over I²C er gået godt.</p> <p><code>uint8 On</code> Returværdi til UART afhængig af hvilken kommando, der blev kaldt. 0 = off, 0 != on.</p>
Returværdi	<p><code>int8</code> Er denne værdi nul er kommunikationen over I²C gennemført. Hvis værdien er -1 er der blevet returneret en værdi tilsvarene requesten fra DevKit8000 over UART (se UART protokol side 39) og der er sket en fejl i kommunikationen over I²C .</p>
Beskrivelse	Hvis parameter værdien er lig nul, sendes en char tilsvarene requesten over UART. Hvis værdien er lig -1, sendes sendes en tilsvarene fejlmeldelse.

Tabel 33: respondHeat

Prototype	<code>int8 respondWin(int8 winStatus)</code>
Parametre	<p><code>int8 winStatus</code> Returværdien fra funktionen adjustWin i boundaryklassen I²C . Fortæller om kommunikationen via. I²C til vinduesaktuatoren er forløbet godt.</p>
Returværdi	<p><code>int8</code> Er denne værdi nul er kommunikationen over I²C gennemført. Hvis værdien er -1 er der blevet returneret 'XK' over UART og der er sket en fejl i kommunikationen over I²C .</p>
Beskrivelse	Hvis parameter værdien er lig nul, sendes en char 'K' over UART. Hvis værdien er lig -1, sendes strengen "XK".

Tabel 34: 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 35: 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 36: 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 37: 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 44).

Tabel 38: 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 44).
Beskrivelse	Slukker eller tænder for varmeaktuatoren. Sender komandoen "WriteAdjustHeat" via I ² C bussen (se I ² C protokol på side 44).

Tabel 39: 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 44).
Beskrivelse	Slukker eller tænder for ventilation. Sender kommandoen "WriteAdjustVentilation" via I ² C bussen (se I ² C protokol på side 44).

Tabel 40: 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 44).
Beskrivelse	Slukker eller tænder for individuelle vandingsaktuatorer. Sender komandoen "WriteAdjustIrrigation" via I ² C bussen (se I ² C protokol på side 44).

Tabel 41: 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 44 for yderligere information.

Tabel 42: 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. [4]
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 44 og til sensorens datablad [4] for yderligere information.

Tabel 43: 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. [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 en variabel. Der henvises til I ² C protokollen på side 44 og til sensorens datablad [5] for yderligere information.

Tabel 44: 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 45: getSoilHum

Domainklasse DSP

Attributter

Navn	Type	Beskrivelse
tempArray	int32[0...*]	Array der indeholder måleværdier fra temperaturmåler.
humArray	int32[0...*]	Array der indeholder måleværdier fra fugtighedsmåler.
soilHumArray	int16[0...*][6]	To-dimensionelt array der indeholder målinger fra jordfugtsensorerne.
lightArray	int32[0...*]	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[6]	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 46: Attributter for klassen DSP

Metoder

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 47: 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 48: 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 49: 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 50: 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 39 og databladet for temperatursensoren [4].

Tabel 51: 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 39 og databladet for temperatursensoren [4].

Tabel 52: 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 <code>soilHum</code> arrayet. Se UART protokollen side 39.

Tabel 53: 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 <code>light</code> . Se UART protokollen side 39 og databladet for temperatursensoren [5].

Tabel 54: 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 55: 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 56: 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 tilsvarende 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.

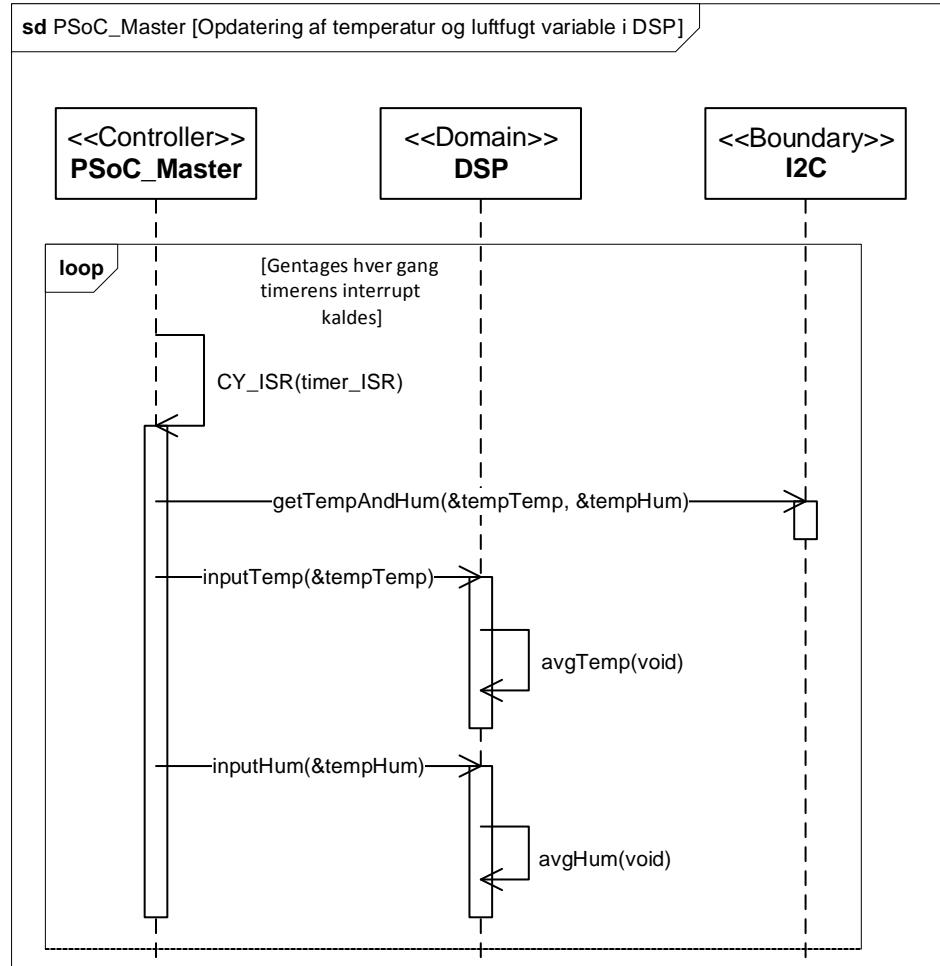
Tabel 57: inputSoilHum

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.

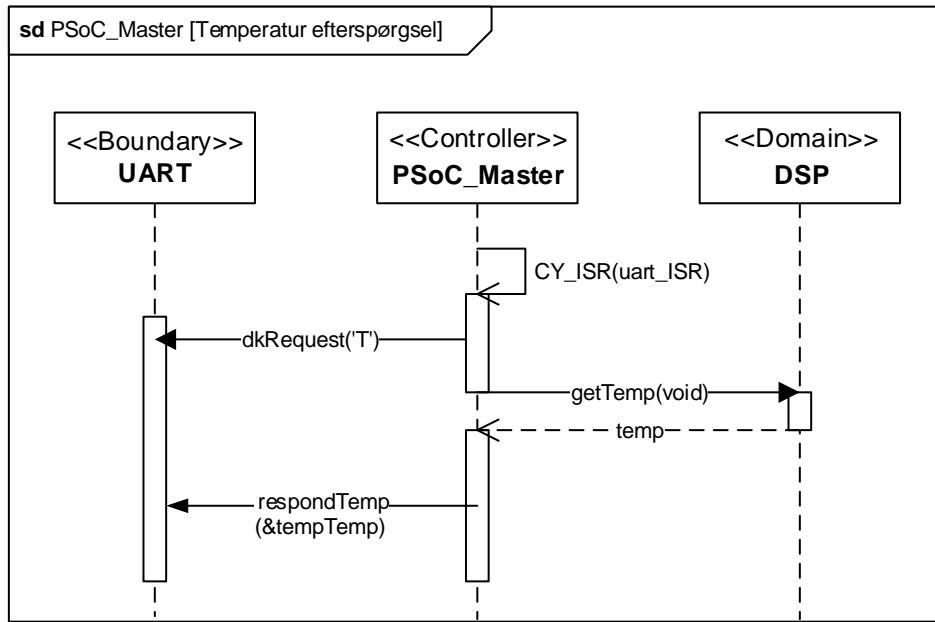
Tabel 58: inputLight

4.2.2 Sekvensdiagrammer

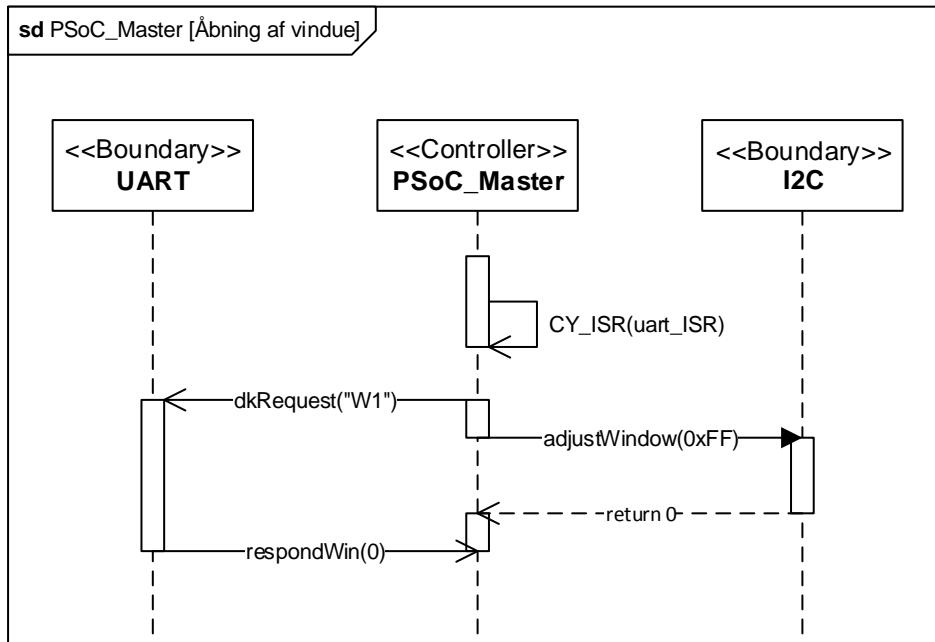
På Figur 16, 17 og 18, ses der forskellige forløb af masterens aktivitet. Sekvensdiagrammerne er eksempler på den funktionalitet der forekommer i masteren.



Figur 16: Sekvensdiagram over opdatering af sensorer.



Figur 17: Sekvensdiagram over forespørgsel af temperatur.



Figur 18: Sekvensdiagram over åbning af vindue.

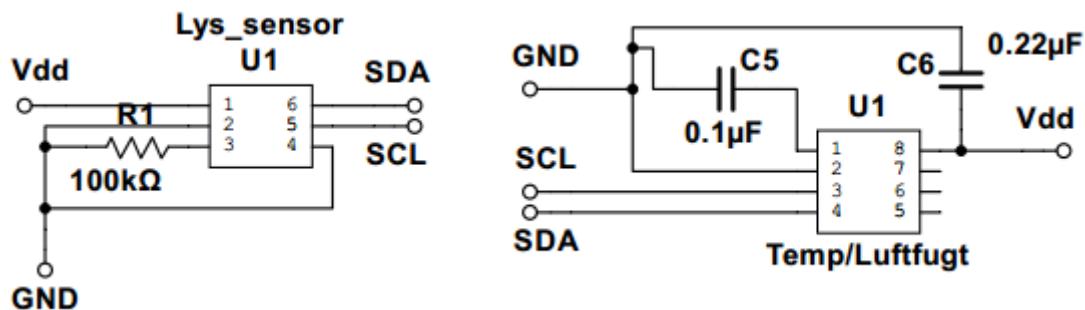
4.3 Breakoutboards

Da AutoGreen er tiltænkt at skulle monitorere temperatur, luftfugtighed og lysintensitet, blev der valgt nogle sensorer der kunne foretage målinger af disse variable.

Temperatur- og luftfugtighedssensoren som blev valgt var en *HONEYWELL S&C HIH6030-021-001* [3]. Sensoren følger en standard I²C -protokol og det er nødvendigt at tage hensyn til I²C -klokvens hastighed samt spændingen på I²C -bussen, så det er muligt for alle enheder på bussen at kommunikere. Fra producenten har sensoren fået en standard adresse, som skal bruges til kommunikation med enheden.

Lyssensoren der blev valgt var en *Intersil ISL29010IROZ* [5]. Lysfølsomheden kan ændres i sensoren ved at ændre en reference vha. forskellige modstandsstørrelser. Intensiteten af lyset måles i lumen, og sensoren kan måle fra 0 til 128.000 lumen, hvilket passer fint til drivhuset, som kan stå både i mørke og i stærk sollys. Som udgangspunkt oplyser databladet at sensoren fungerer bedst som vist i multisimdiagrammet i Figur 19.

Størrelsen på sensorerne gør dem meget svære at arbejde med, så der blev designet et breakoutboard til hver sensor. Både Temp/Luftfugt og Lyssensoren er af SOIC typen og meget små, så der er anvendt Multisim og Ultiboard til design af breakoutboards, så interfacing med sensorerne blev gjort væsentligt nemmere. Kredsløbene for breakout boards kan ses i Figur 19.



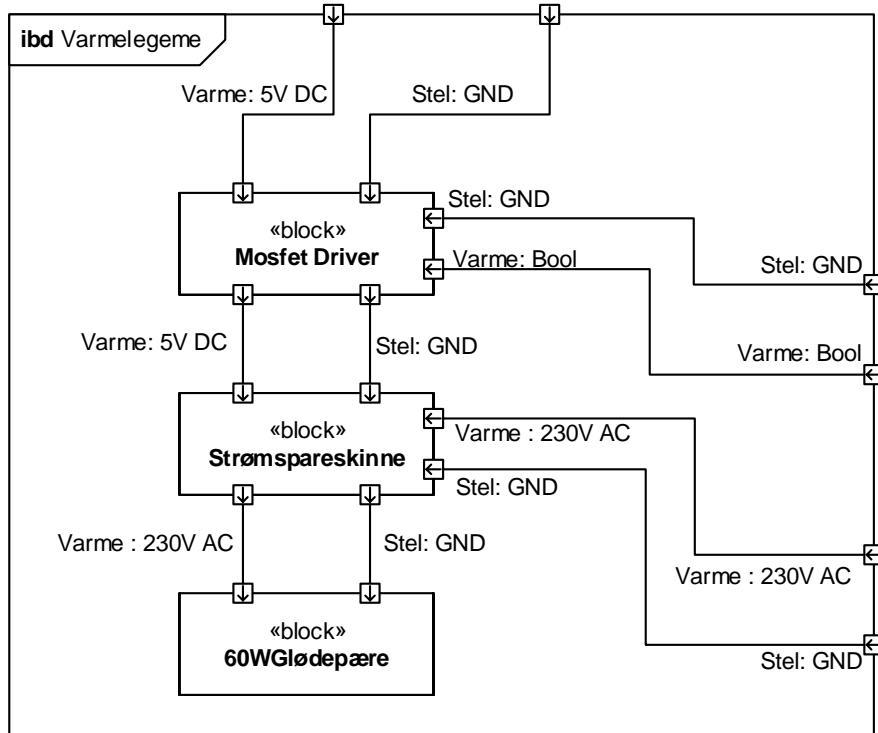
Figur 19: Design af temp/luftfugtsensor og lyssensor breakoutboards.

Alle komponenters værdier fra billederne er taget fra sensorernes datablade.

4.4 Aktuator Design

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

4.4.1 Varmelegeme

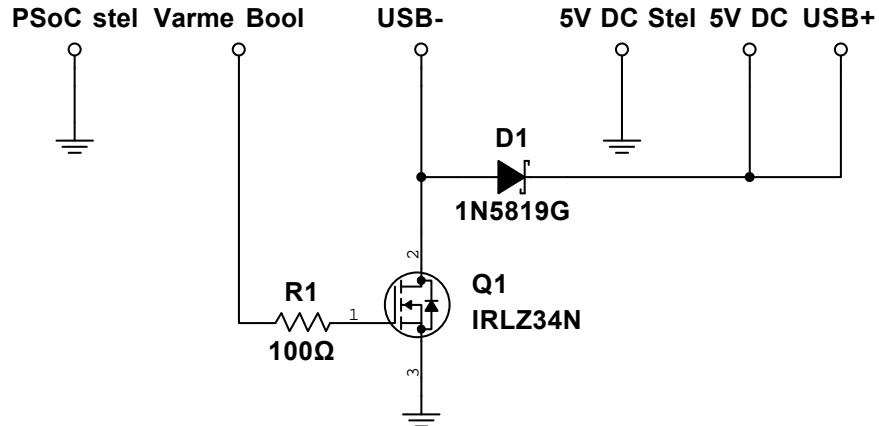


Figur 20: IBD for underblokken Varmelegeme i Aktuator

Ovenstående diagram (Figur 20) 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.

Aktuatorenens 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.

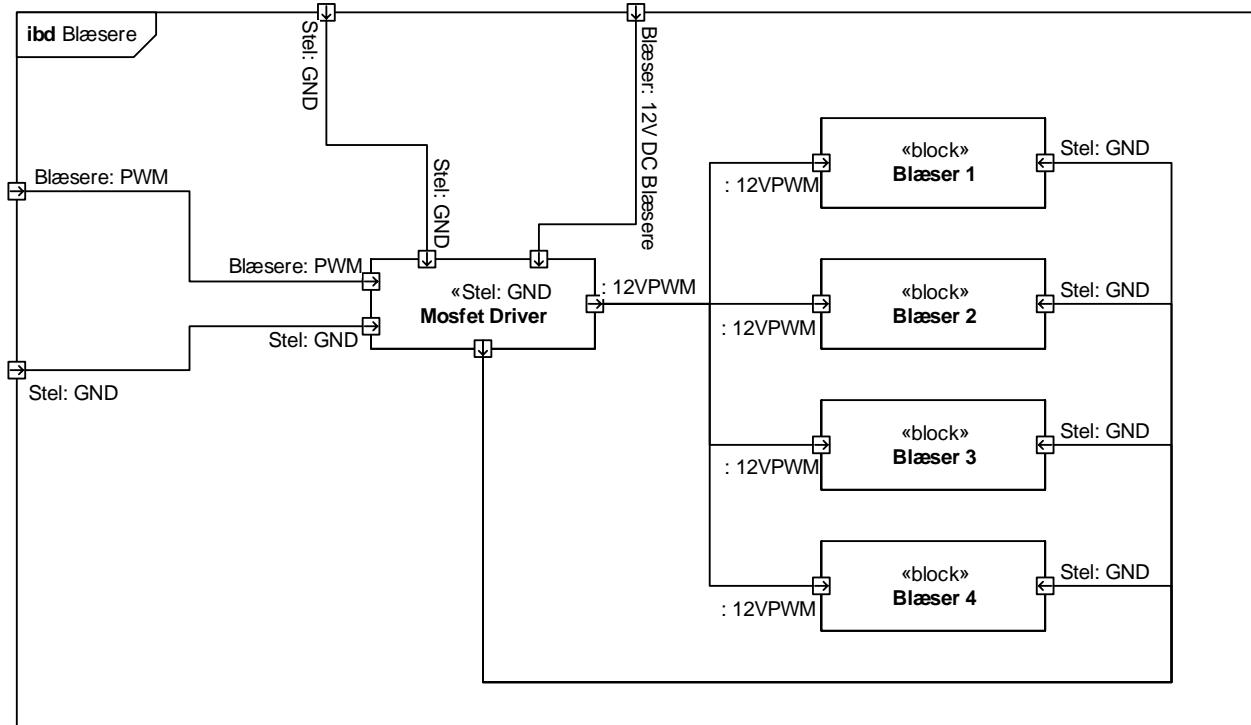
Såfremt det senere vælges at opgradere til PWM styring, skal man tage højde for - eller se bort fra - at effekten ikke er lineært sammenhængende med dutycyclen. Dette skyldes dels at effekt har en sammenhæng med kvadratet af strømmen, dels at modstanden i glødetråden afhænger af temperaturen.



Figur 21: Kredsløb for Mosfet Driver i underblokken Varmelegeme

Når Varme Bool på Figur 20 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.

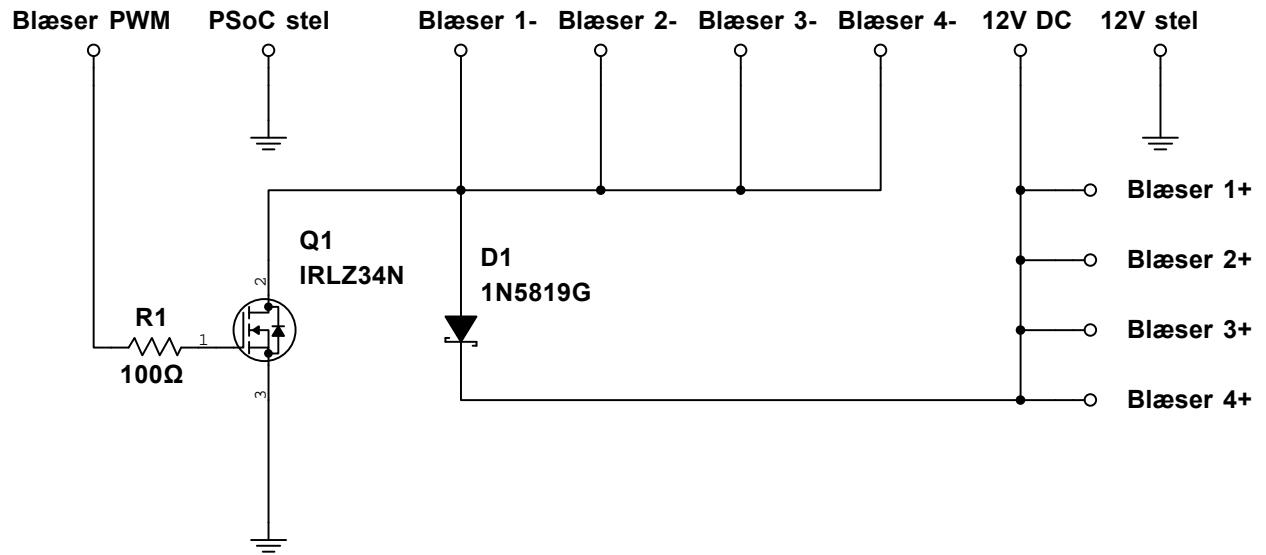
4.4.2 Blæsere



Figur 22: IBD for underblokken Blæsere i Aktuator

Figur 22 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.

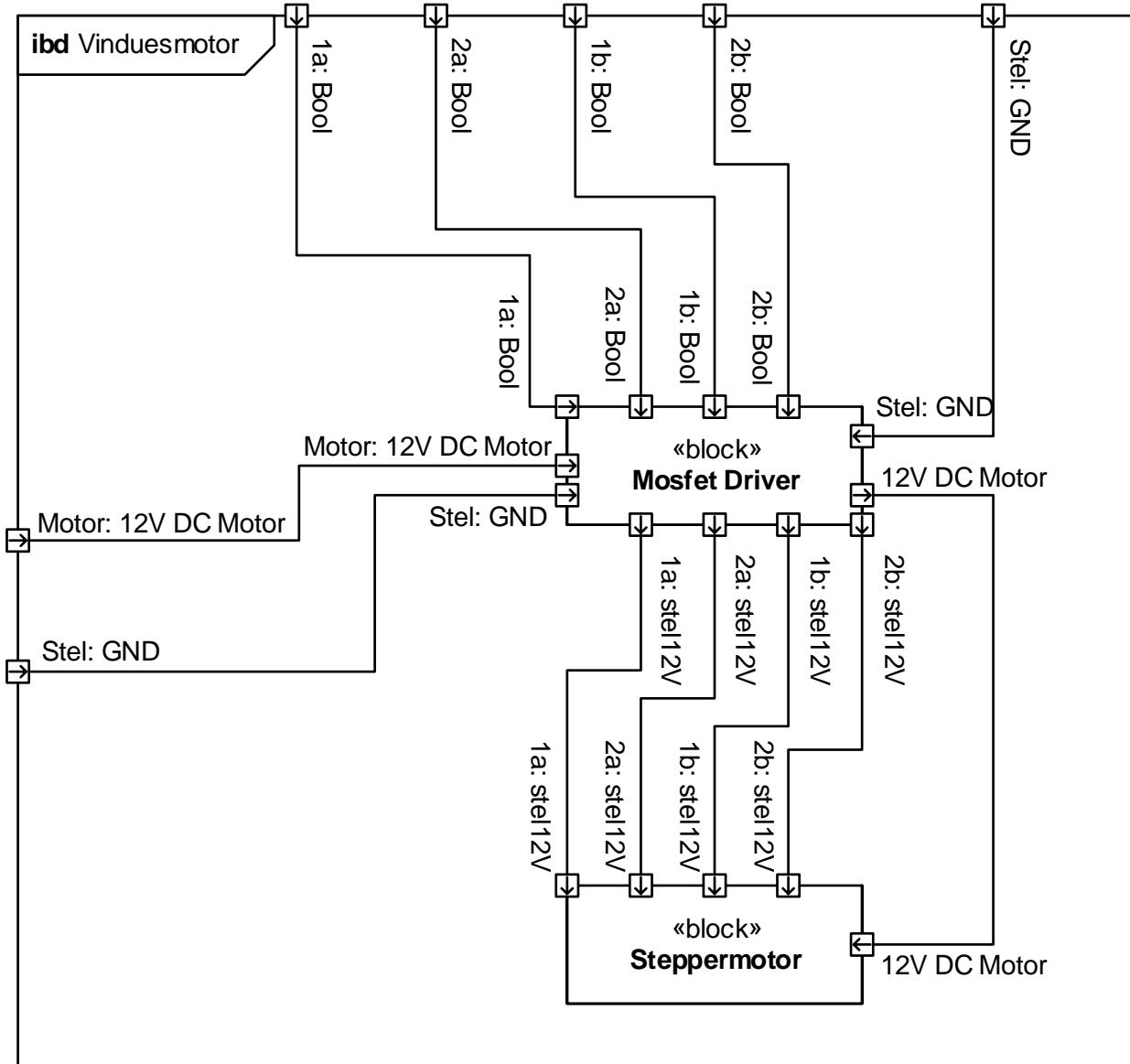
Ved praktiske forsøg, konstateredes det, at en dutycycle på 50% er et fornuftigt maximum. Det konstateredes desuden, at blæserne skal startes på maximum (dutycycle 50%) for at komme i gang. Hvis der startes med en mindre dutycycle, opnår motoren ikke inertie nok til at begynde dreje. Begge dele implementeres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



Figur 23: Kredsløb for Mosfet Driver i underblokken Blæsere

Mosfet Driveren til Blæsere på Figur 23 fungerer i principippet på samme måde som Mosfet Driver for Varmelegeme (Figur 21). Der er blot tilsluttet fire blæsere, der alle styres vha. den samme transistor.

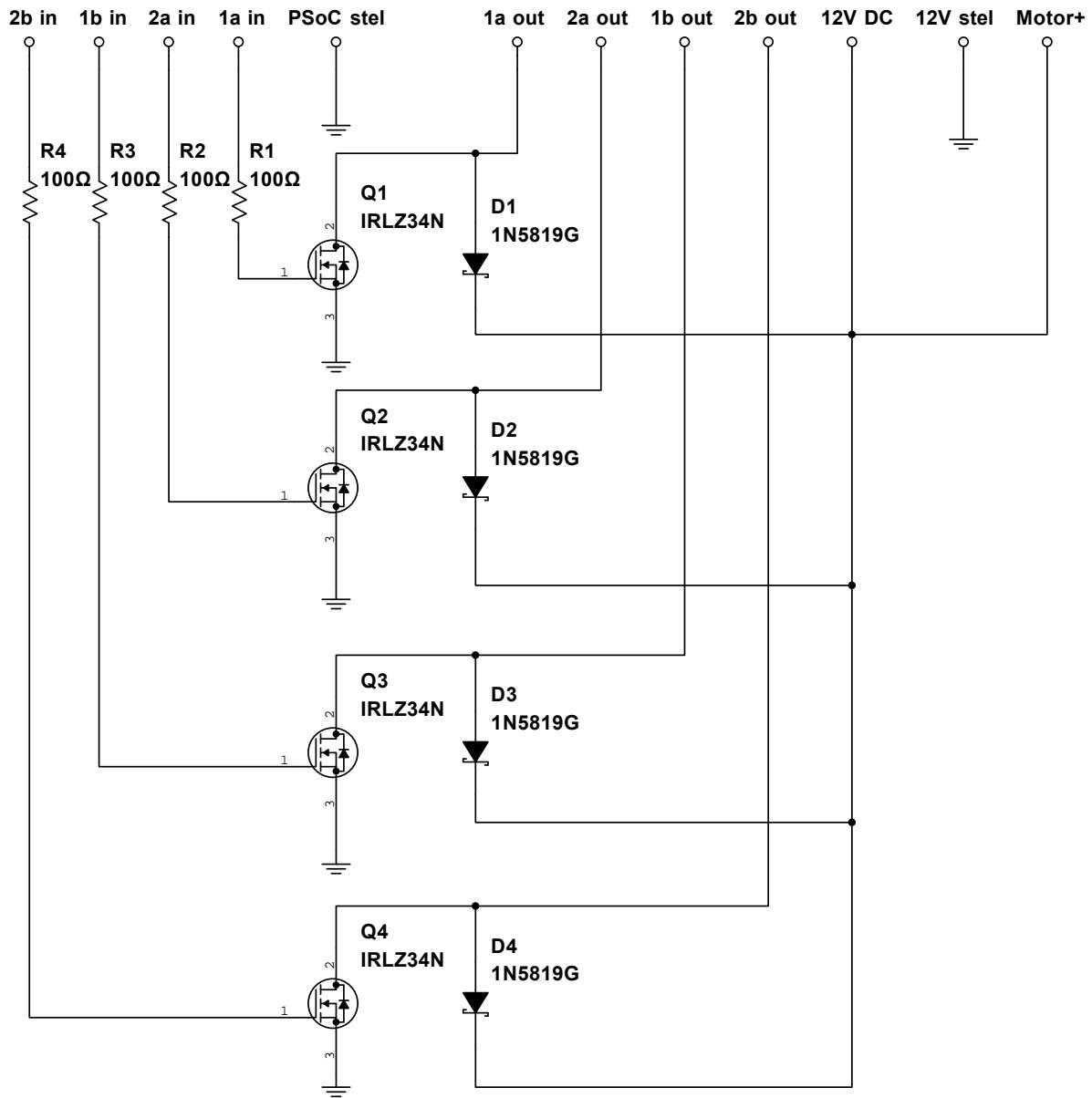
4.4.3 Vinduesmotor



Figur 24: IBD for underblokken Winduesmotor i Aktuator

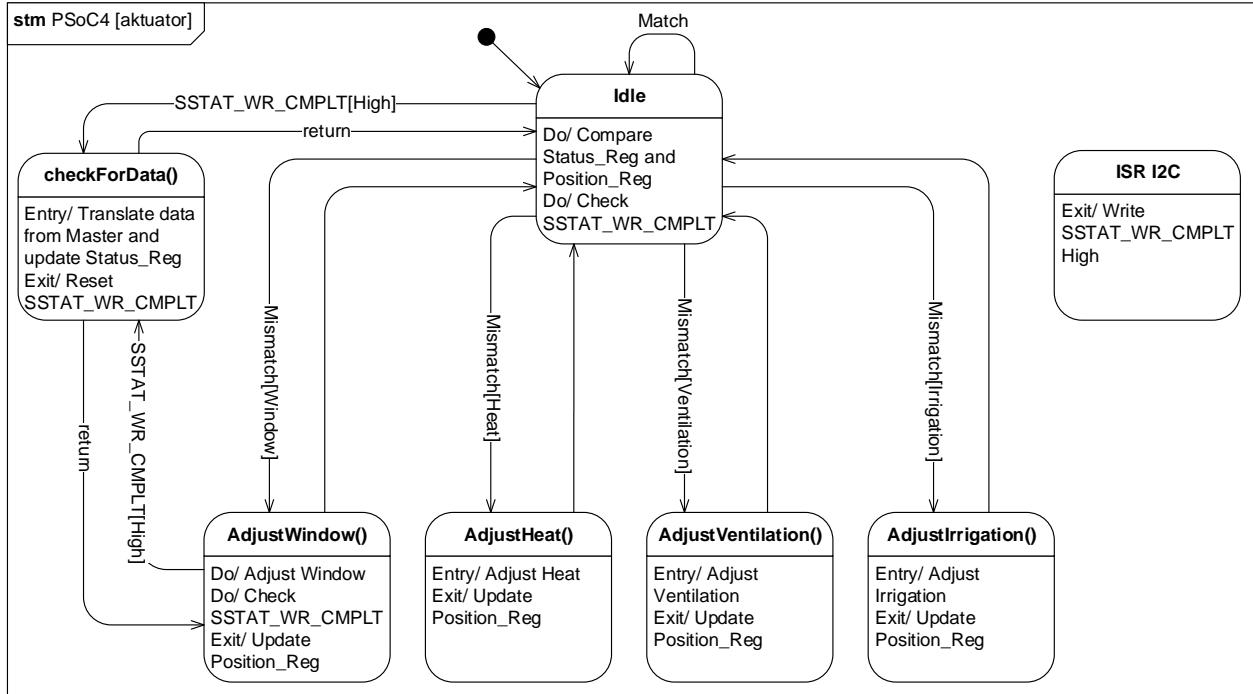
Ovenstående diagram (Figur 24) viser interne forbindelser i underblokken Winduesmotor, der består af en Steppermotor og en Mosfetdriver. Der åbnes og lukkes for mosfet transistorer i Mosfet Driveren vha. 3,3V signaler fra PSoC4, og derved forsyner Steppermotor med 12V DC.

Mosfet Driveren til Winduesmotor på Figur 25 side 70 fungerer i principippet på samme måde som Mosfet Driver for Blæsere (Figur 23). Der er dog den forskel, at de fire signaler styres af hver deres transistor, da de ikke alle skal åbne og lukke samtidigt.



Figur 25: Kredsløb for Mosfet Driver i underblokken Vinduesmotor

4.4.4 PSoC4



Figur 26: State Machine for software på underblokken PSoC4 i Aktuator

Ovenstående diagram (Figur 26) 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 opdatering 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.

4.4.5 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 Vanding

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

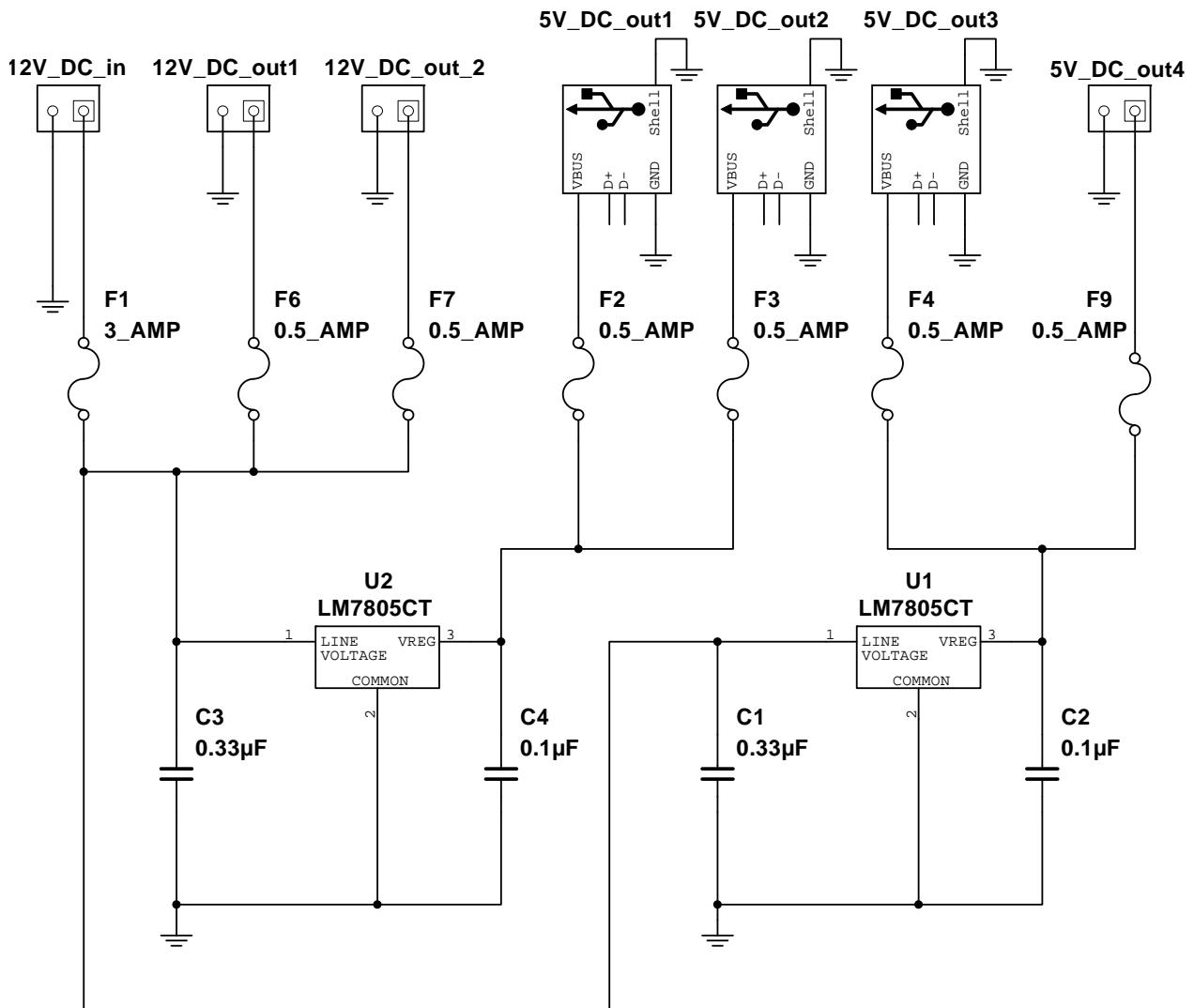
Denne driver har til formål at opdatere Status_Reg. Der er mulighed for at kalde denne fra Idle tilstand og under AdjustWindow. Driveren kaldes kun i det tilfælde, at I2C bussen har kaldt et interrupt, hvilket medfører opdatering af slave write complete registeret fra lav til høj.

4.5 Strømforsyning Design

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 12 på side 34. 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 27: Diagram for blokken Strømforsyning

Figur 27 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.

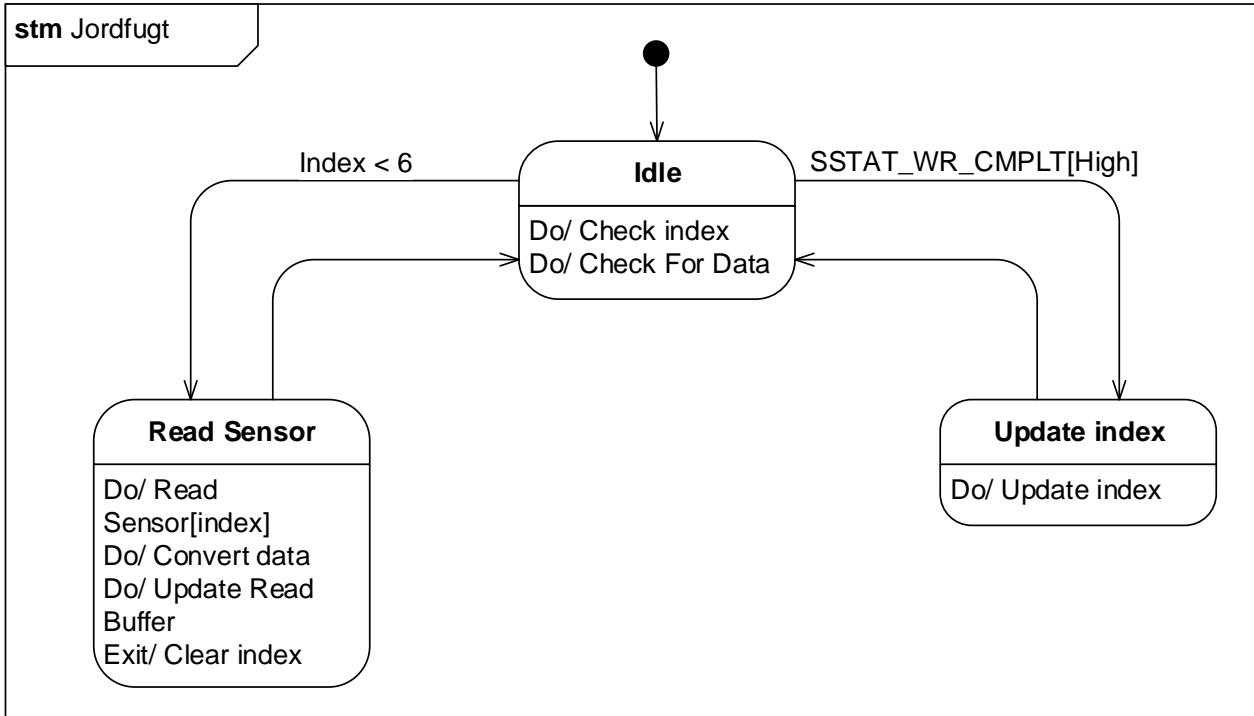
12V DC out1 udgangen til motor trækker max. 500 mA jf. databladet [7] side 38, derfor monteres den med en sikring på 500 mA.

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 maksimalt 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 slæbet 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 levere 1A, derfor anvendes to stk. De monteres med afkoblinger til stel på ben 1 og 3 jf. standardapplikationen på side 1 i databladet [6].

4.6 Jordfugt Design

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



Figur 28: State Machine for SW på PSoC4 i blokken Jordfugt

Ovenstående figur (Figur 28) 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 komunikationen, 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 [9] 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.

5 Software Design

Version

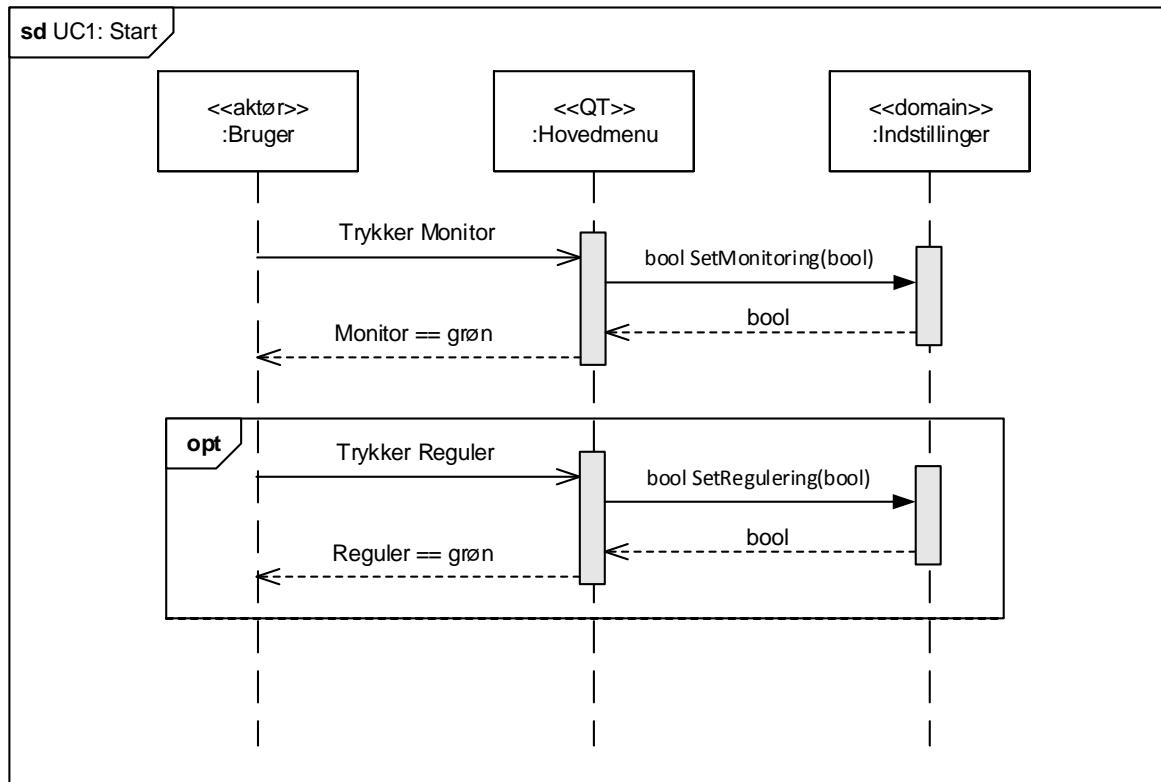
Dato	Version	Initialer	Ændring
29. marts	1	KS	Første udkast.
15. maj	2	HBJ	Klassebeskrivelser indskrevet i LaTeX samt andre mindre rettelser.
18. maj	3	KTS	Opdateret figurer og trådhåndtering.
20. maj	4	KTS	Tilføjet forklarende tekst til figurer

5.1 Indledning

Under dette afsnit vil design af software blive beskrevet gennem sekvensdiagrammer og dertil tilhørende klassebeskrivelser. Herudover vil der kort blive beskrevet hvorledes tråde håndteres i systemet.

5.2 Udvidet Applikationsmodel: Sekvensdiagrammer

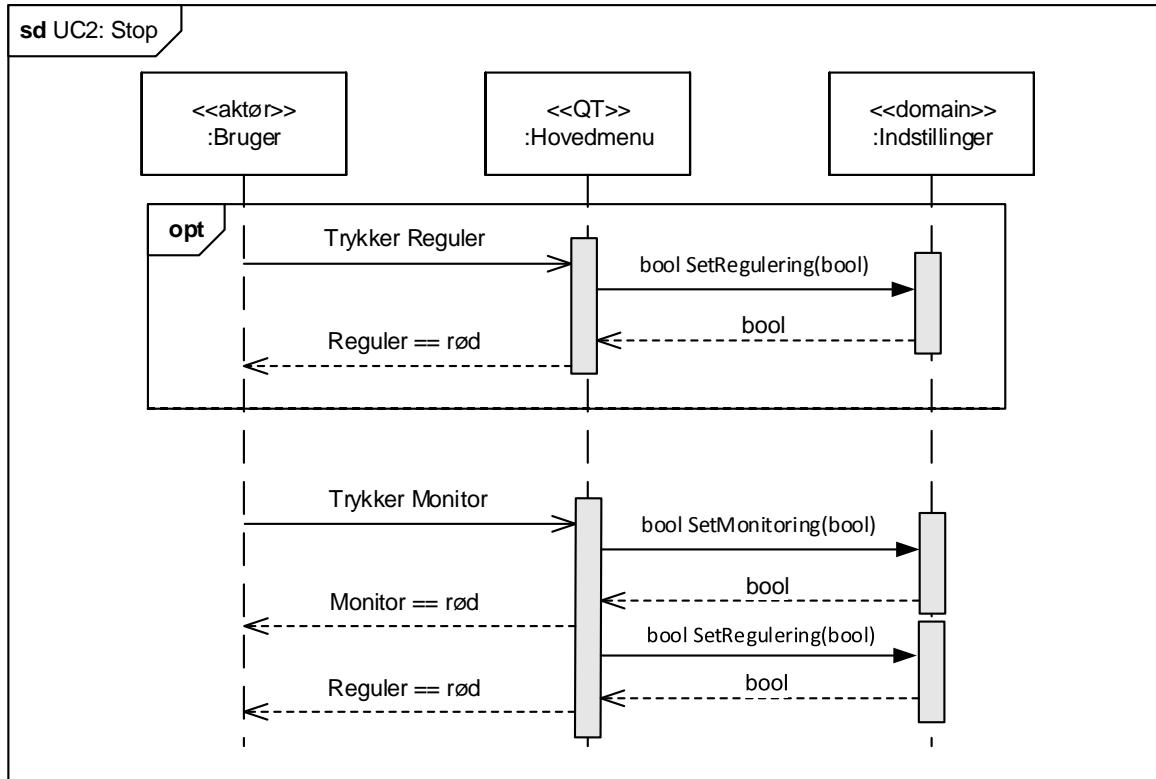
5.2.1 Usecase 1: Start



Figur 29: Application model for UC1: Start

Sekvensdiagrammet for start er funktionaliteten der skal forgå når der startes for monitorering, og også funktionaliteten for regulator hvis den også tændes.

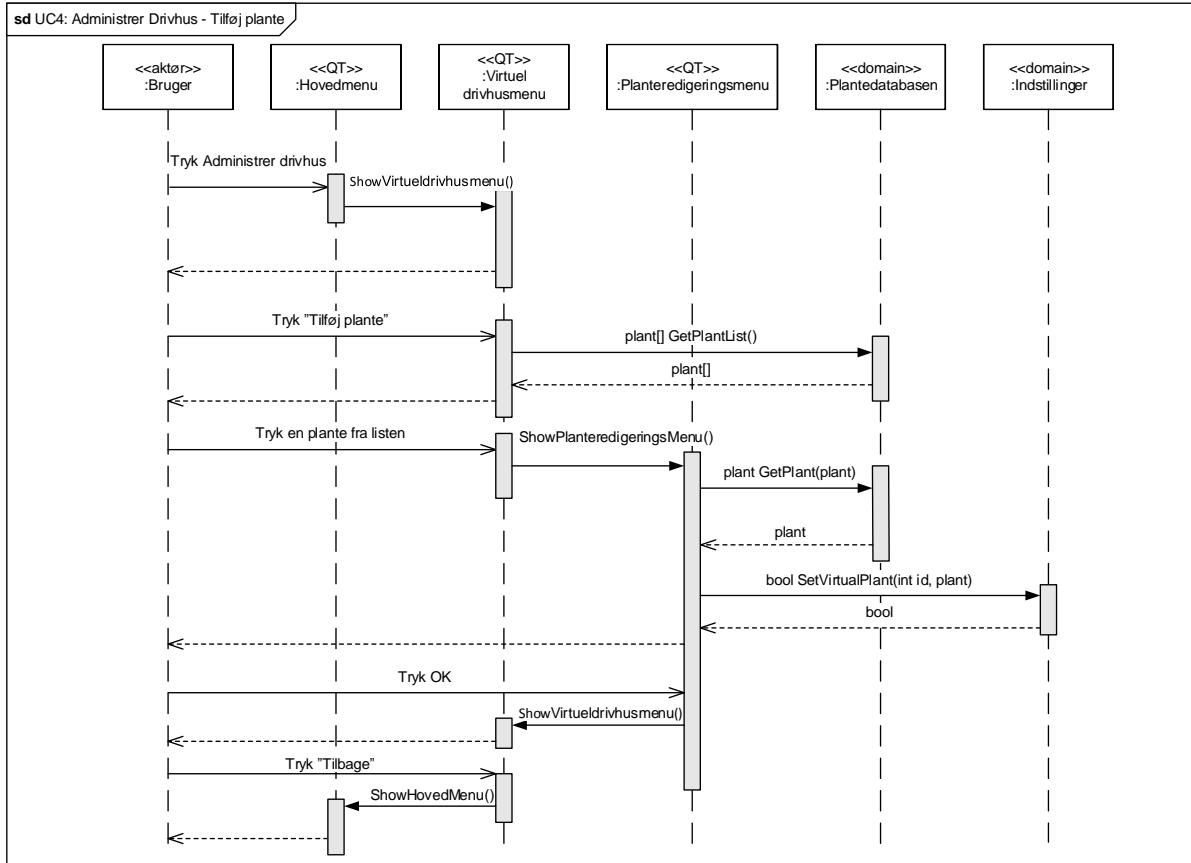
5.2.2 Usecase 2: Stop



Figur 30: Application model for UC2: Stop

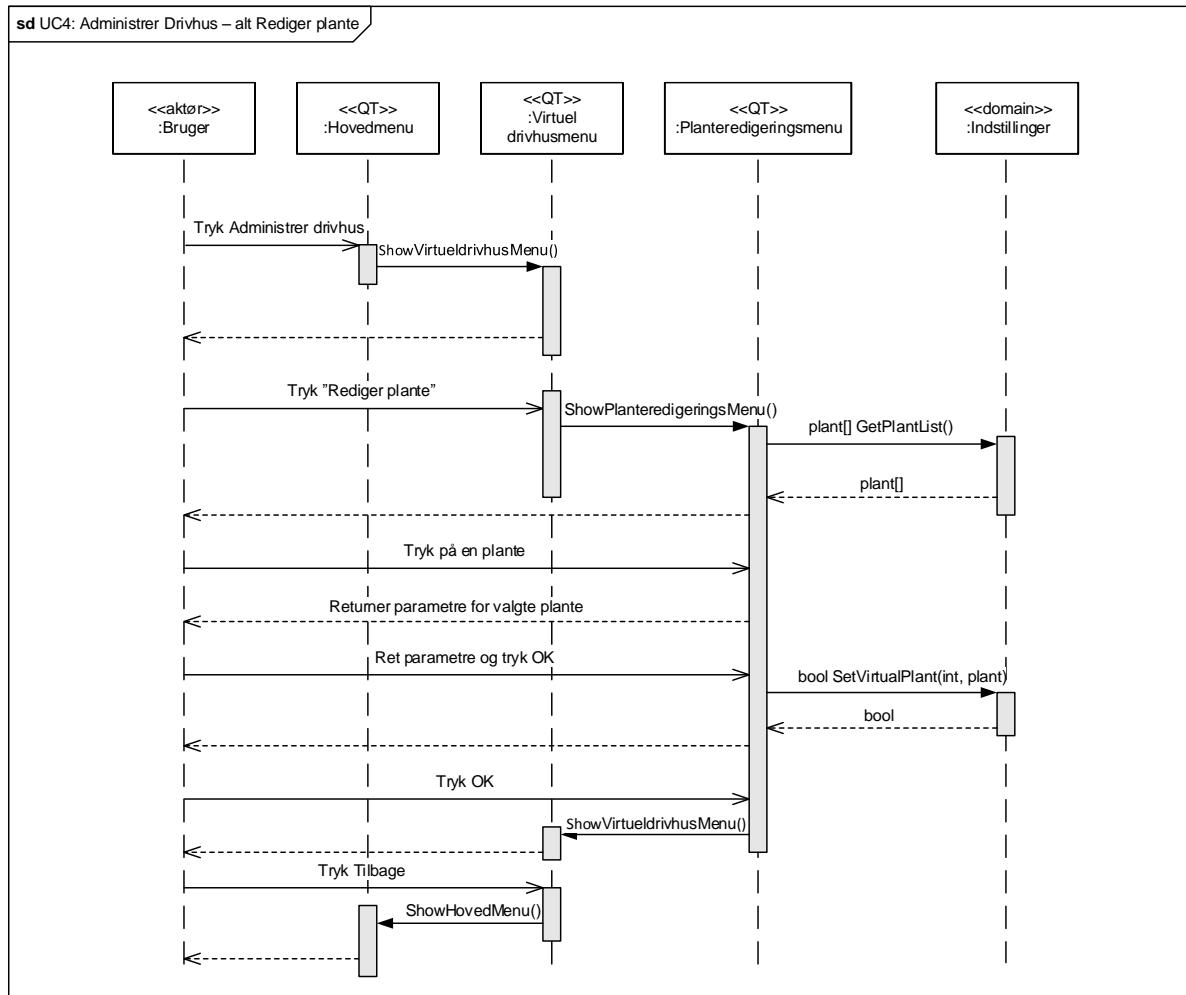
Sekvensdiagrammet for stop, når der ønskes at stoppe for monitorering og/eller regulering. Tryk på monitorering på begge er tændt, resultere i at både regulator og monitor slukkes.

5.2.3 Usecase 4: Administrer Drivhus



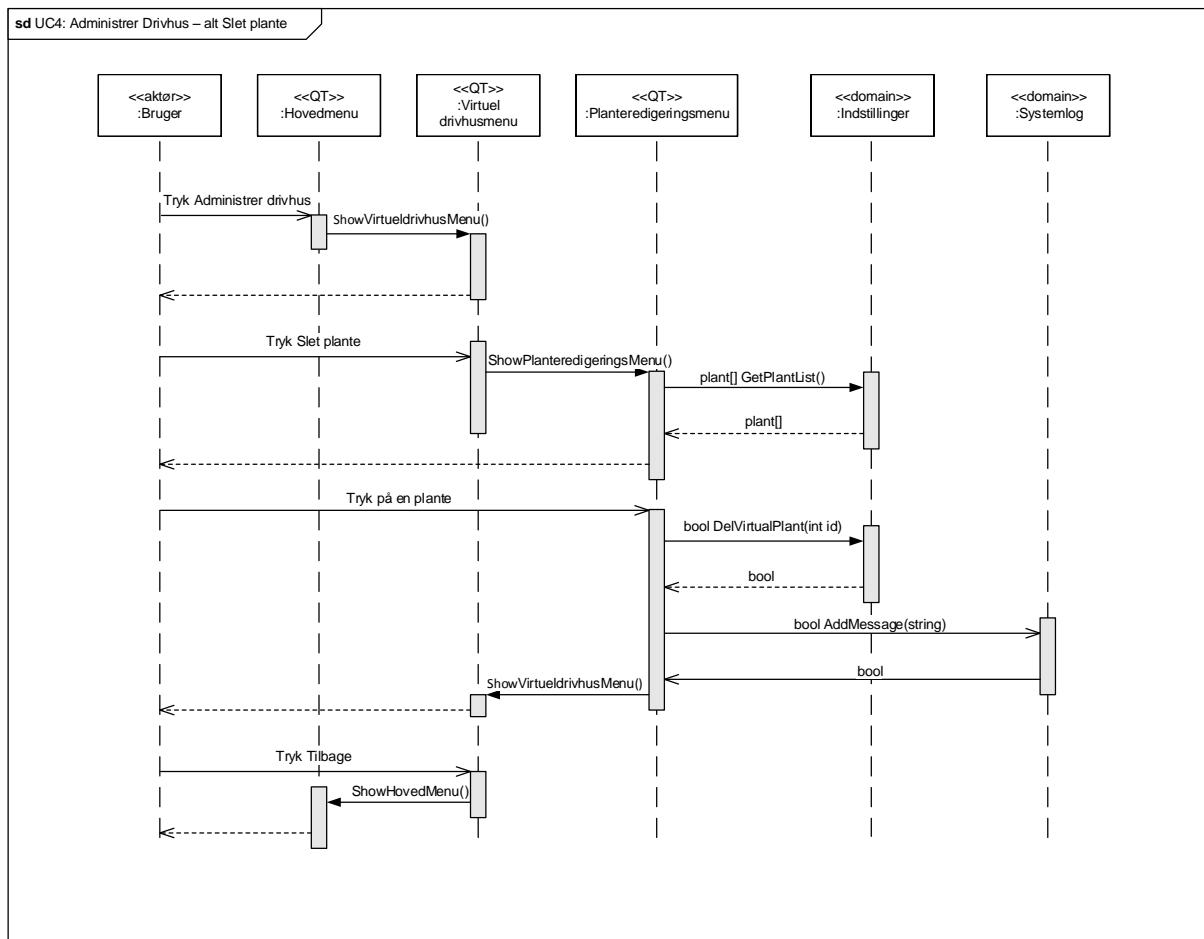
Figur 31: Application model for UC4: Administrer Drivhus - Tilføj Plante

Administrere drivhus sekvensdiagrammet giver overblik over hvad der fortages når der ønskes at tilføje en plante igennem det virtuelle drivhus.



Figur 32: Application model for UC4: Administrer Drivhus - [alt] Rediger Plante

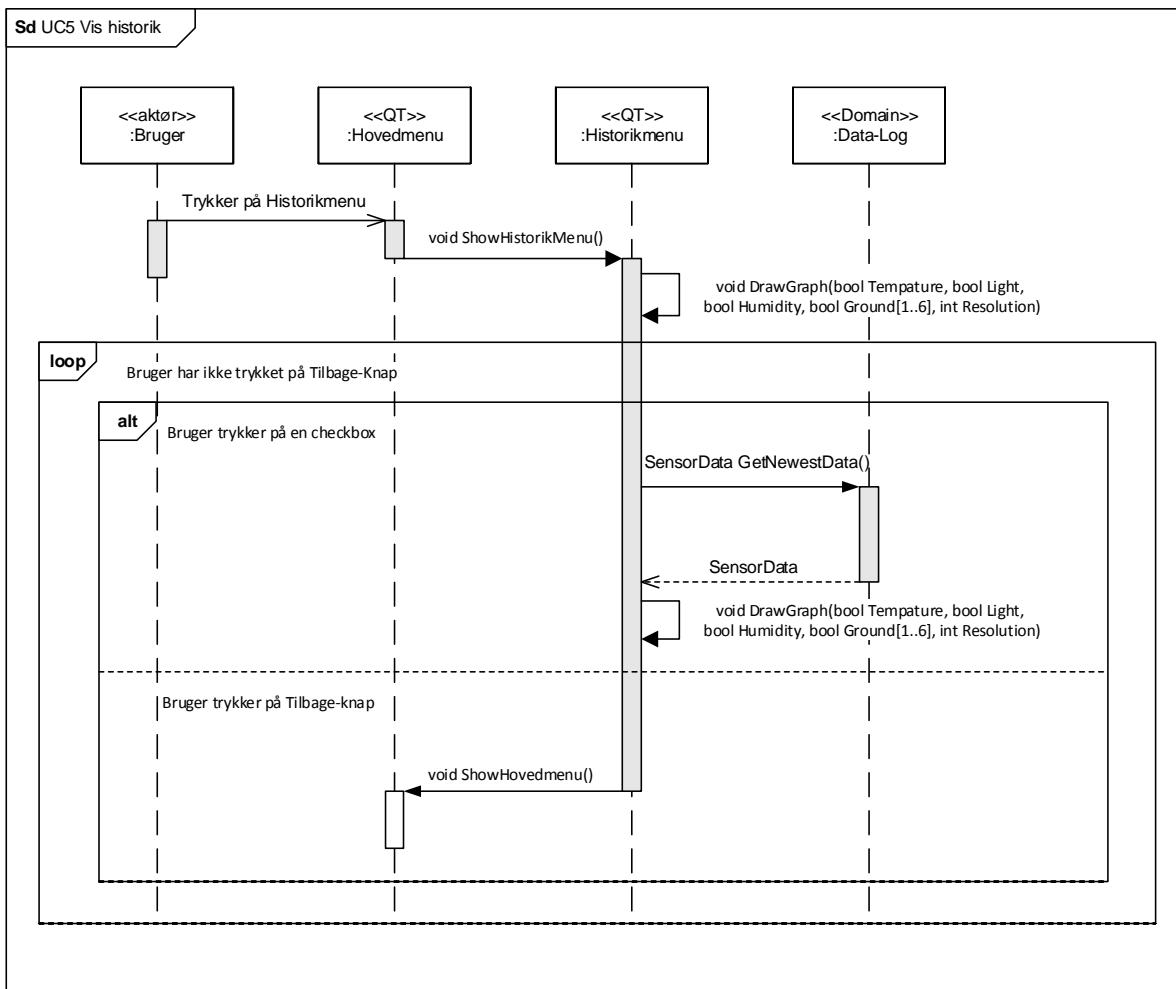
Sekvensdiagrammet giver overblik over hvad der fortages i softwaren, når det ønskes at redigere en af de allerede tilstedevarende planter i det virtuelle drivhus.



Figur 33: Application model for UC4: Administrer Drivhus - [alt] Slet Plante

Sekvensdiagrammet giver overblik over hvad der fortages i softwaren, når det ønskes at fjerne en tilstedeværende plante fra det virtuelle drivhus.

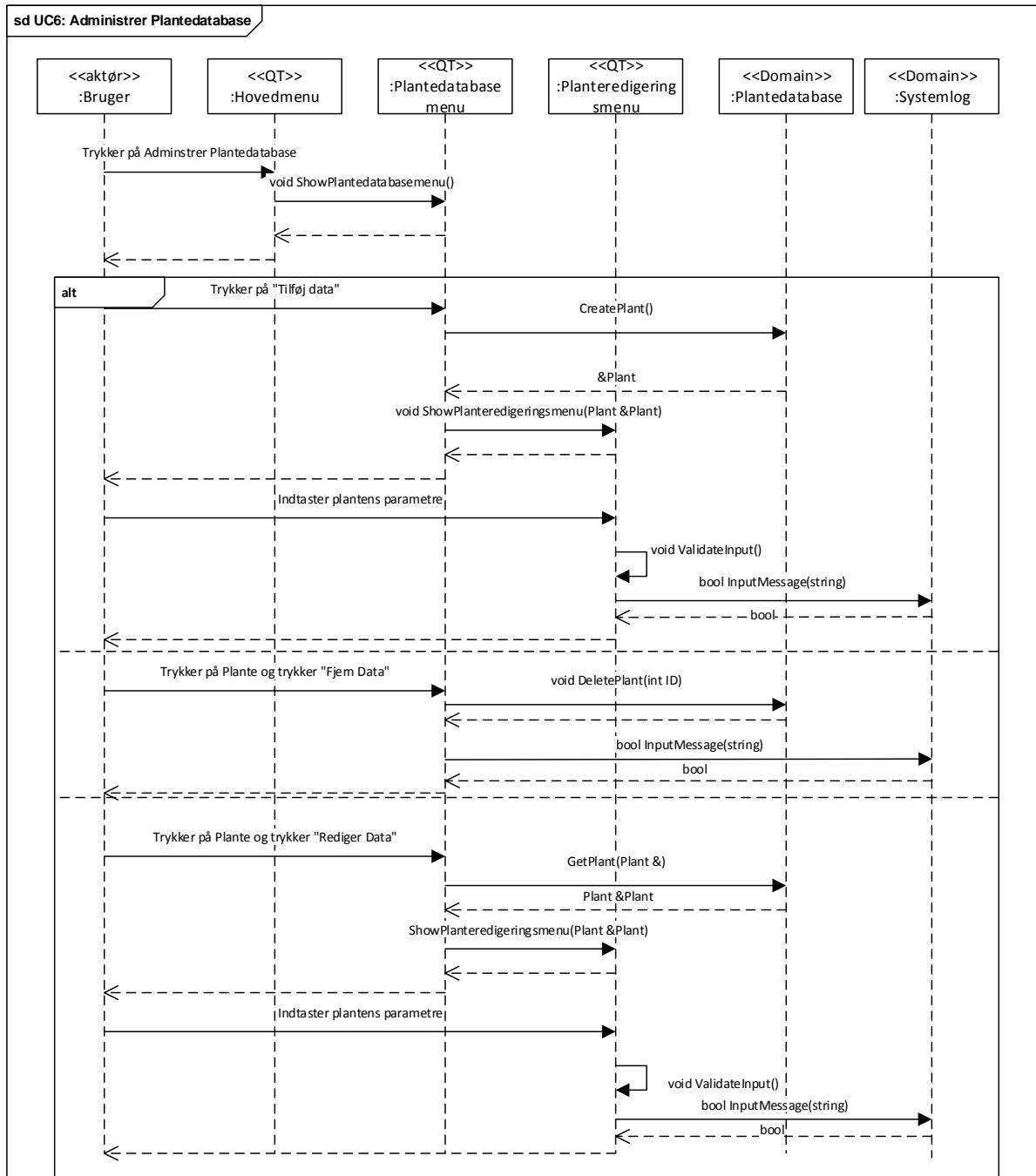
5.2.4 Usecase 5: Vis Historik



Figur 34: Application model for UC5: Vis Historik

Sekvensdiagrammet giver overblik over hvordan historiken fungerer, når brugeren trykker ind på 'se historik' igennem hovedmenuen.

5.2.5 Usecase 6: Adminstrerer Plantedatabase

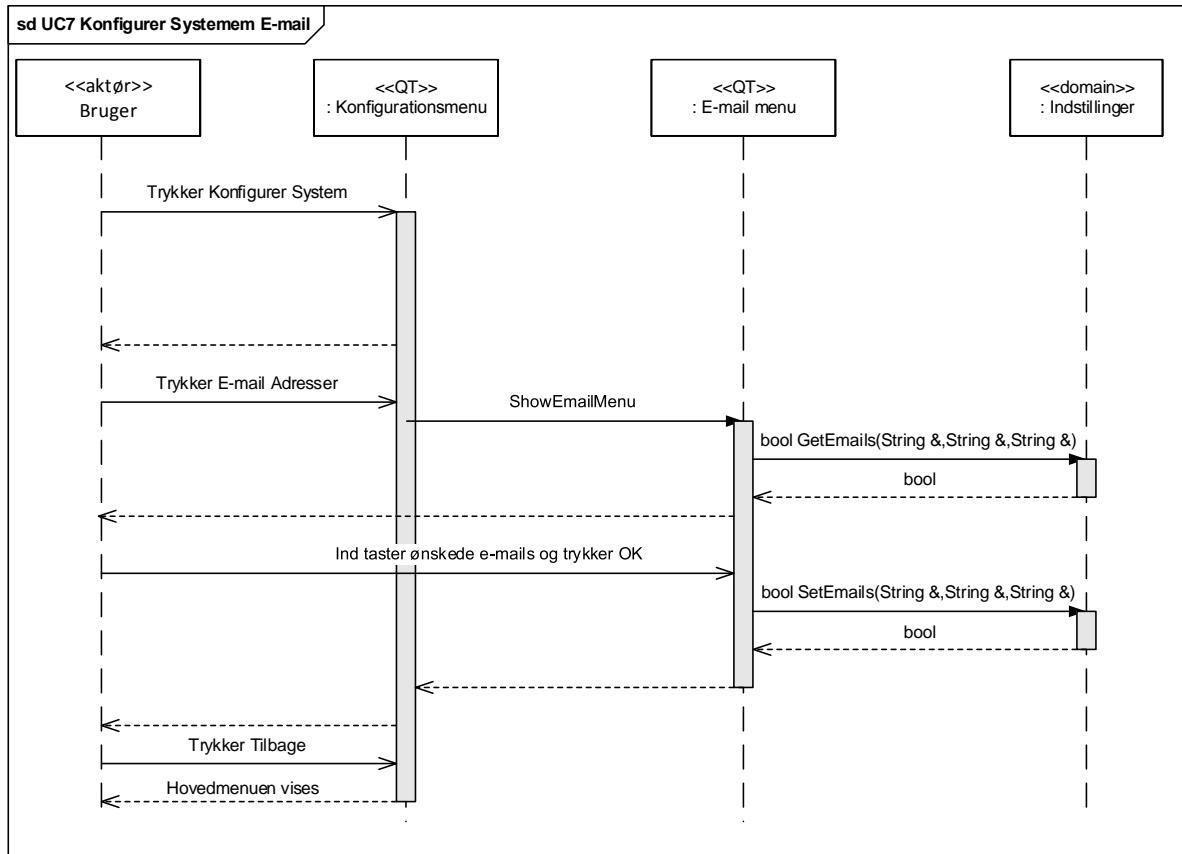


Figur 35: Application model for UC6: Administerer Plantedatabase

Sekvensdiagrammet giver overblik over de forskellige muligheder brugeren skal have inde fra plantedatabasen, hvorvidt en plante skal tilføjes, redigeres eller slettes.

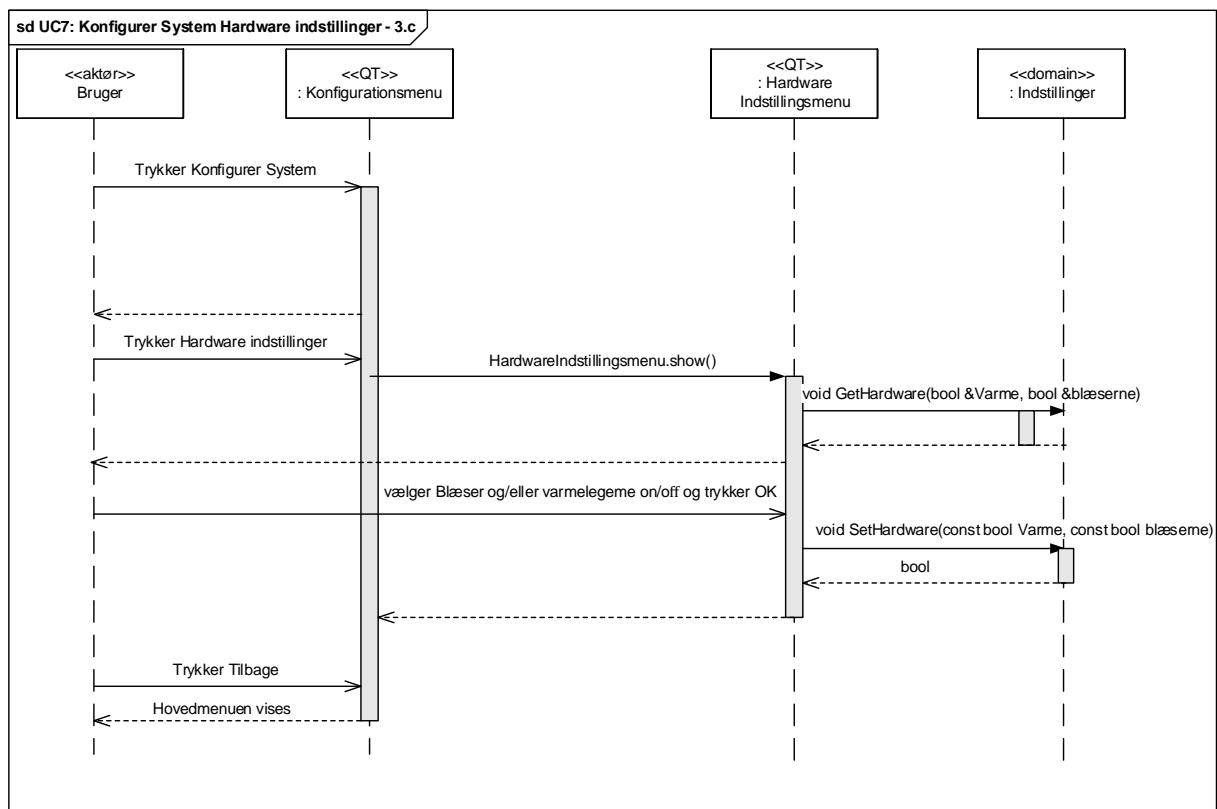
5.2.6 Usecase 7: Konfigurer System

Når brugeren går ind på indstillingsmenuen bliver brugeren mødt med 4 undermenuer som kan tilgåes, hertil er sekvensdiagrammerne for disse menuer.



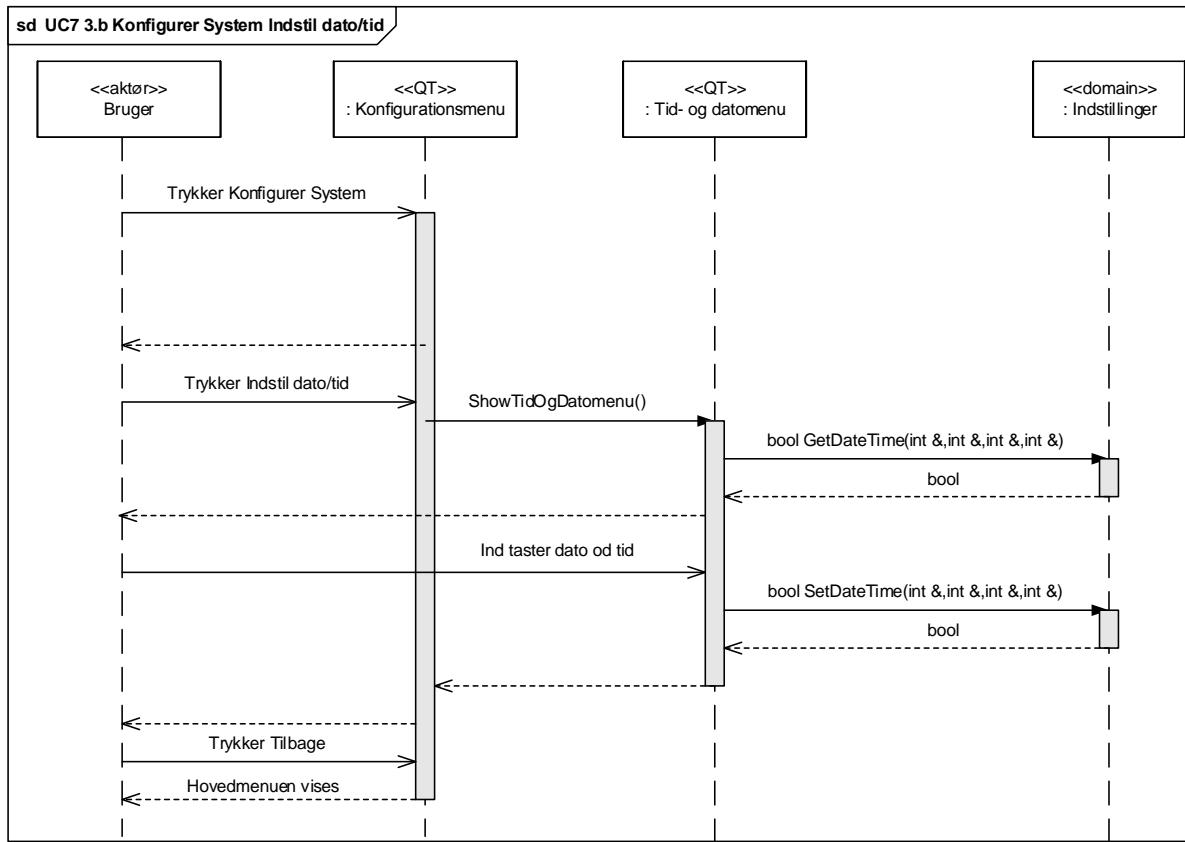
Figur 36: Application model for UC7: Konfigurer System - E-mail

sekvensdiagrammet giver overblik over, hvordan brugeren kan skifte E-mails igennem E-mail menuen.



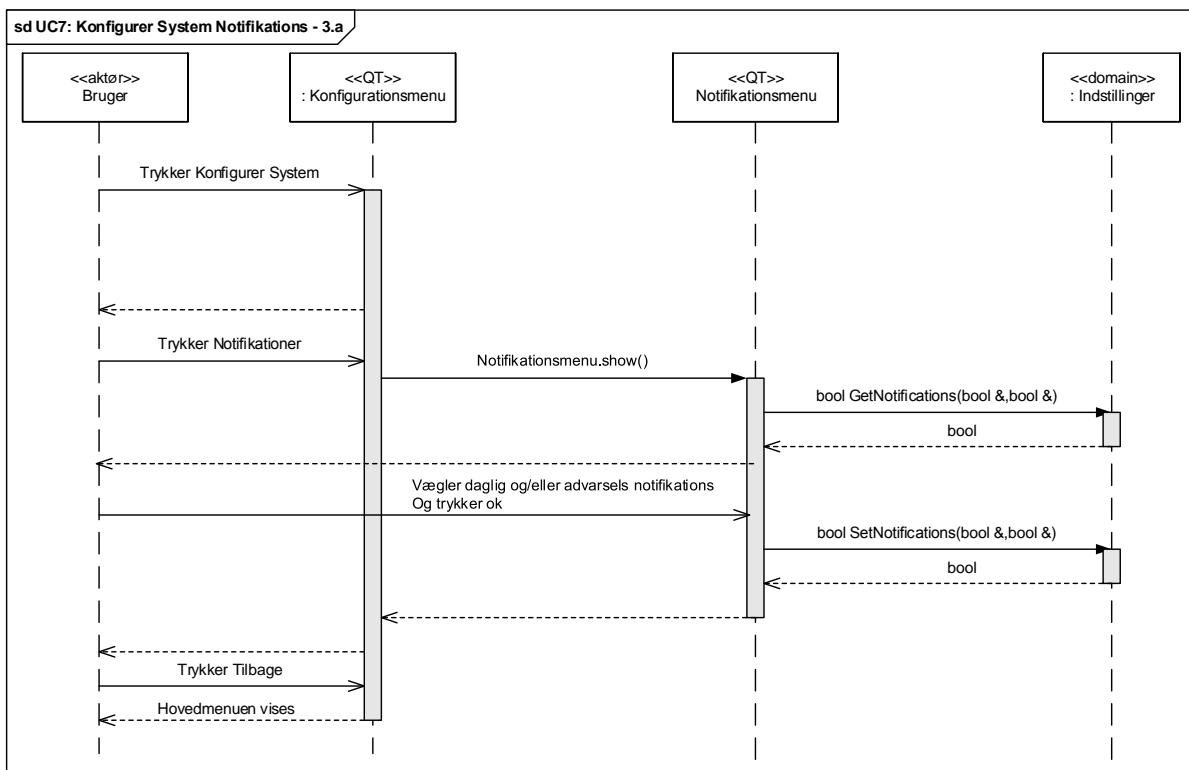
Figur 37: Application model for UC7: Konfigurer System - Hardware Indstillinger

sekvensdiagrammet giver overblik over, hvordan brugeren kan skifte hardware indstillinger. Brugeren kan her vælge om blæsere og varmelegeme skal anvendes under regulering.



Figur 38: Application model for UC7: Konfigurer System - Dato/Tid

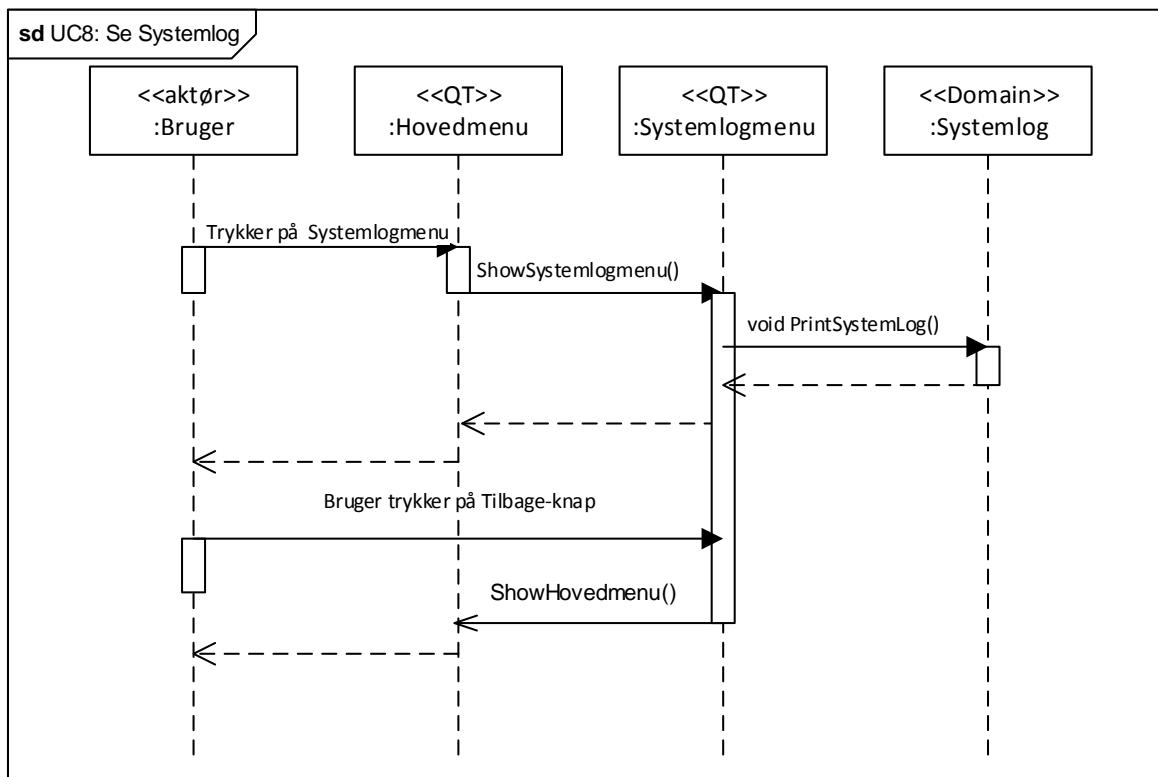
Sekvensdiagrammet giver overblik over, hvordan brugeren kan indstille tiden på systemet.



Figur 39: Application model for UC7: Konfigurer System - Notifikation

Sekvensdiagrammet giver overblik over, hvordan brugeren kan aktivere og deaktivere brugen af daglige og vigtige notifikations emails fra systemet.

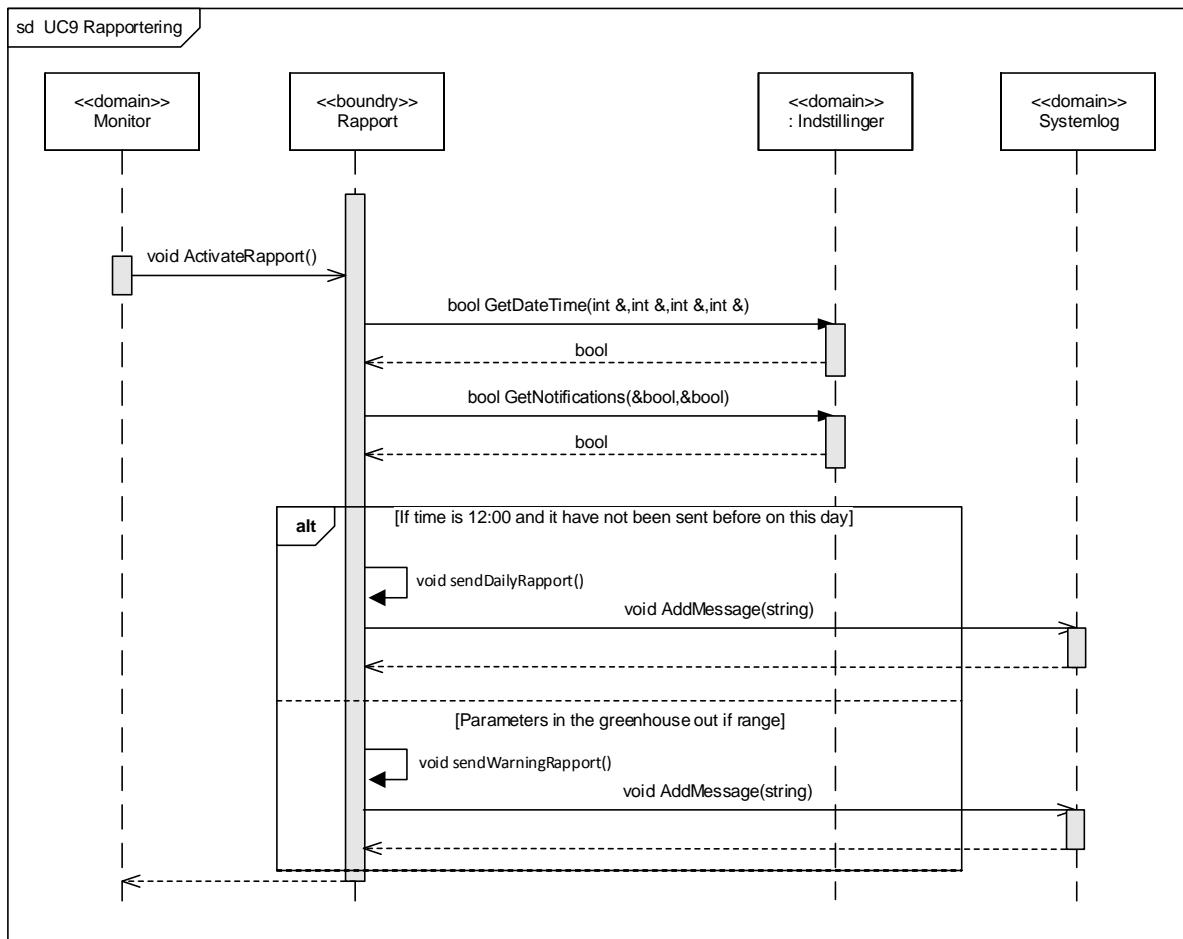
5.2.7 Usecase 8: Se Systemlog



Figur 40: Application model for UC8: Se Systemlog

Sekvensdiagrammet giver overblik over hvordan brugeren kan tilgå systemloggen igennem dens egen menu. Når brugeren har tilgået menuen, udskrives de seneste systemhændelser på skærmen.

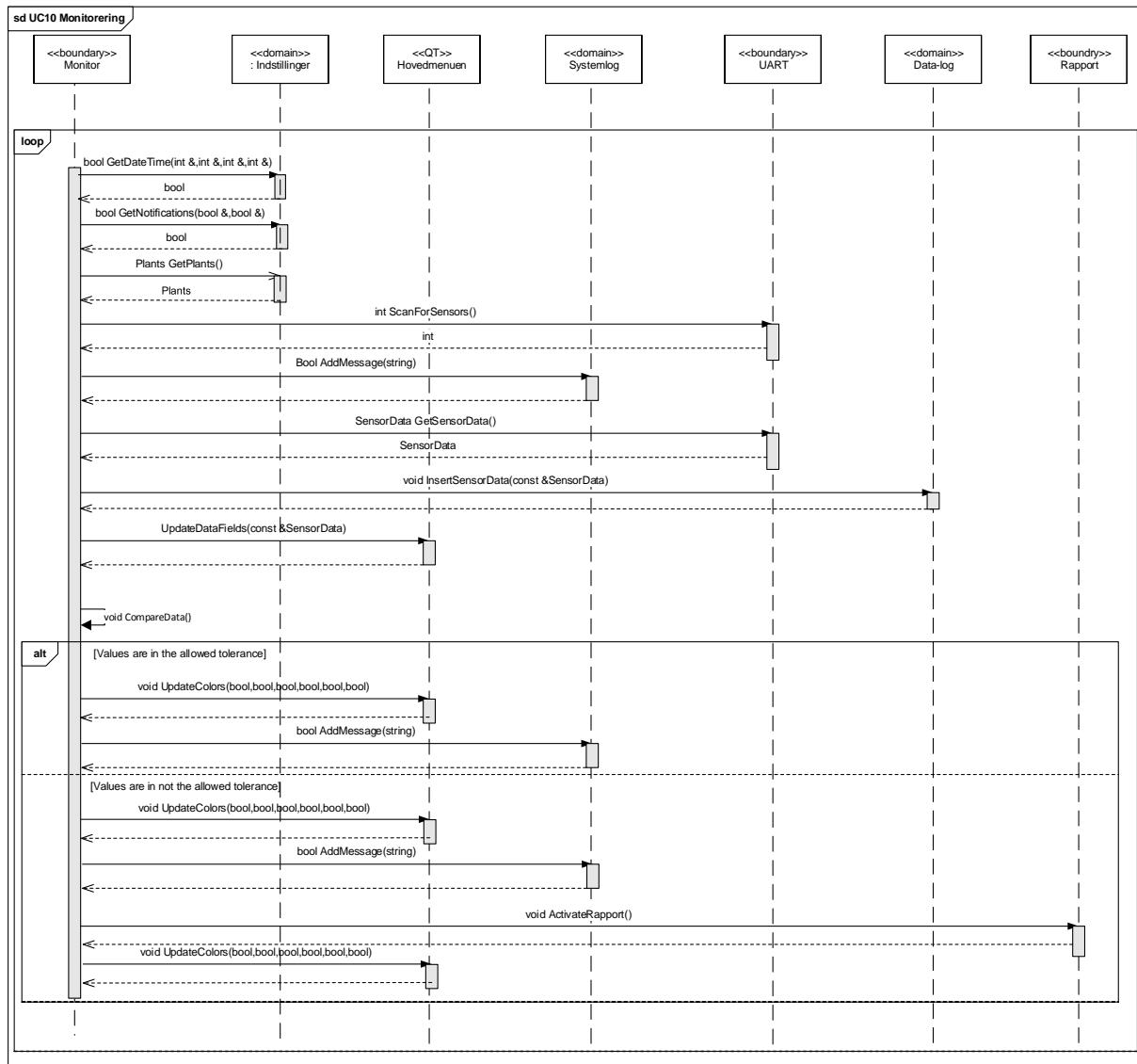
5.2.8 Usecase 9: Rapportering



Figur 41: Application model for UC9: Rapportering

Sekvensdiagrammet giver overblik over, hvordan systemet kan bruger rapportering til at udsende emails til bruger, om den skal sende daglige, kun vigtige eller ingen email hentes ind fra indstillinger.

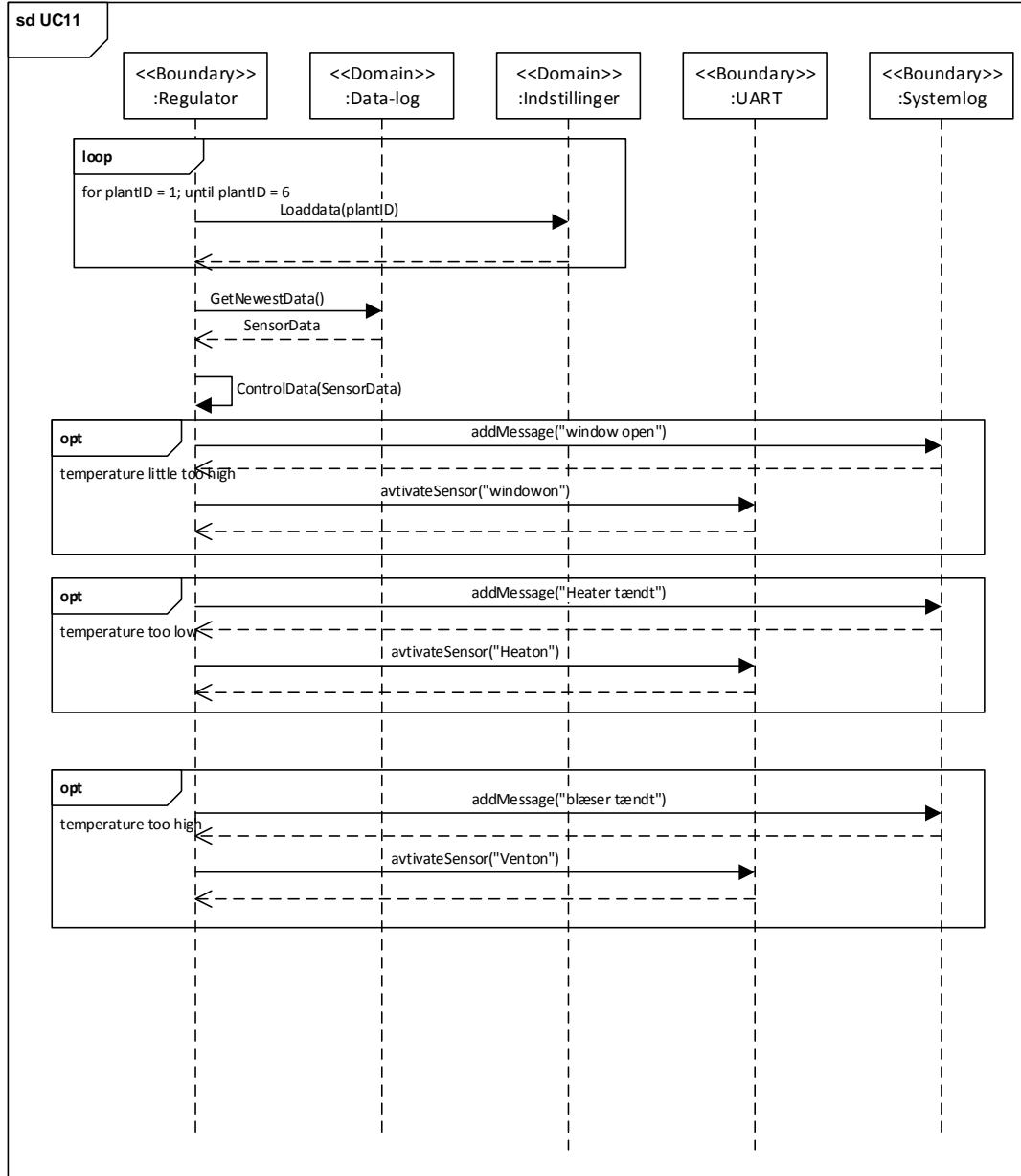
5.2.9 Usecase 10: Monitorering



Figur 42: Application model for UC10: Monitorering

Sekvensdiagrammet giver et overblik over hvordan monitoreringstråden skal fungere, og hvilke funktionskald den skal fortage til andre klasser.

5.2.10 Usecase 11: Regulering



Figur 43: Application model for UC11: Regulering

Sekvensdiagrammer giver overblik over hvordan reguleringstråden skal fungere, og hvilke funktionskald den skal foruge til andre klasser. f.eks. kald til UART omkring åbning eller lukning af vinduet.

5.3 Klassebeskrivelser

5.3.1 Datalog

Dataloggen er en klasse som anvendes til at gemme data omkring klimaet i drivhuset i. Dataloggen består af en linked DoublyLinkedList 5.3.2 , som indeholder data over hver parameter i drivhuset, der bliver logget hvert minut.

Attributter

list	DoublyLinkedList- <SensorData>	En DoublyLinkedList der bruges til at gemme data i SensorData.
------	--------------------------------	--

Tabel 59: Attributter for klassen Datalog

Metoder

Prototype	<code>void InsertSensorData(SensorData sensorData)</code>
Parametre	<code>SensorData SensorData</code> Indeholder det sensordata som du gerne vil indsætte i dataloggen.
Returværdi	-
Beskrivelse	Bruges til at indsætte sensorData i dataloggen.

Tabel 60: InsertSensorData

Prototype	<code>SensorData GetNewestData()</code>
Parametre	-
Returværdi	Det nyste SensorData som ligger gemt indeholder.
Beskrivelse	Metoden har til formål at hente de seneste indsatte data i linked listen og returnere dem.

Tabel 61: GetNewestData

Prototype	<code>void Sortday()</code>
Parametre	-
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 62: Sortday

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.

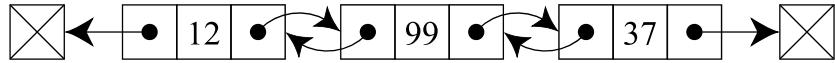
Tabel 63: Sortweek

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 64: Sortmonth

5.3.2 DoublyLinkedList

En doublylistedlist er blevet valgt til projektet da det er blevet vurderet at den har de fleste fordele i forhold til andre datastrukturer. Som der er erfaring med i gruppen, en fordel ved at holde sig til en datastruktur er mere tid til at udvikle den og lave den rigtig god i forhold til at lave mange forskellige strukturere. Den er Implementeret som en template klasse.



Figur 44: A doubly linked list whose nodes contain three fields: an integer value, the link forward to the next node, and the link backward to the previous node.

Denne kommer til at have samme struktur som Figur 44 i stedet for kun at gemme ints laves den som en template klasse så vi kan gemme alle data typer.

Attributter

<code>headPtr</code>	<code>Node<A_Type>*</code>	Head ponteren som bruges internt i klassen, og peger på det første element i listen.
<code>tailPtr</code>	<code>Node<A_Type>*</code>	Tail ponteren som bruges internt i klassen, og peger på det bagerste element i listen.
<code>itemsInList</code>	<code>int</code>	En int som holder styr på hvor mange elemeter som den doubly listed list indeholder.

Tabel 65: Attributter for klassen DoublyLinkedList

Metoder

Prototype	<code>DoublyLinkedList()</code>
Parametre	-
Returværdi	-
Beskrivelse	headPtr og tailPtr sættes til at peger på NULL og itemsInList sættes til 0.

Tabel 66: DoublyLinkedList

Prototype	<code>~DoublyLinkedList()</code>
Parametre	-
Returværdi	-
Beskrivelse	Når klassen nedlægges fjernes alt det data som den havde gemt.

Tabel 67: ~DoublyLinkedList

Prototype	<code>void tailInsert(A_Type info)</code>
Parametre	<code>A_Type info</code> Datatypen som skal gemmes i listen.
Returværdi	-
Beskrivelse	Bruges til at gemme datatypen bagerst i listen.

Tabel 68: tailInsert

Prototype	<code>void headInsert(A_Type info)</code>
Parametre	<code>A_Type info</code> Datatypen som skal gemmes i listen.
Returværdi	-
Beskrivelse	Bruges til at gemme datatypen fronten i listen.

Tabel 69: tailInsert

Prototype	<code>void headDelete()</code>
Parametre	-
Returværdi	-
Beskrivelse	Bruges til at fjerne element i fronten listen.

Tabel 70: headDelete

Prototype	<code>void tailDelete()</code>
Parametre	-
Returværdi	-
Beskrivelse	Bruges til at fjerne det bagerste element i listen.

Tabel 71: tailDelete

Prototype	<code>void forwardTraversing()</code>
Parametre	-
Returværdi	-
Beskrivelse	Løber gennem hele den doubly linked list fra headPtr til tailPtr og udskriver alle elementer som den har glemt.

Tabel 72: forwardTraversing

Prototype	<code>void backwardTraversing()</code>
Parametre	-
Returværdi	-
Beskrivelse	Løber gennem hele den doubly linked list fra tailPtr til headPtr og udskriver alle elementer som den har glemt.

Tabel 73: backwardTraversing

Prototype	<code>int Find(A_Type valueToFind)</code>
Parametre	<code>A_Type valueToFind</code> En type af A_Type som den skal prøve at finde i listen.
Returværdi	Hvis den findes i listen returnere den et tal over 0 som angiver dens placering i listen. Hvis -1 findes den ikke i listen.
Beskrivelse	Bruges til at finde en type af A_Type i den linked list.

Tabel 74: Find

Prototype	<code>int PeekTail(A_Type &PeekTail)</code>
Parametre	<code>A_Type &PeekTail</code> En reference til en A_Type, som ændres til det element som tailPtr peger på.
Returværdi	Hvis der er noget indhold i listen får man 1 tilbage ellers -1.
Beskrivelse	Anvendes til at hente det element som ligger bagerst i listen.

Tabel 75: PeekTail

Prototype	<code>int PeekHead(A_Type &PeekHead)</code>
Parametre	<code>A_Type &PeekHead</code> En reference til en A_Type, som ændres til det element som headPtr peger på.
Returværdi	Hvis der er noget indhold i listen får man 1 tilbage ellers -1.
Beskrivelse	Anvendes til at hente det element som ligger fronten i listen.

Tabel 76: PeekHead

Prototype	<code>void deleteAt(int place)</code>
Parametre	<code>int place</code> En placering i listen, som angiver det sted, hvor et element som ønskes slettet.
Returværdi	-
Beskrivelse	Bruges et at slette et element på den plads som parlamenteeren angiver.

Tabel 77: deleteAt

Prototype	<code>int GetItemsInList()</code>
Parametre	-
Returværdi	Et tal som beskriver hvor mange elementer som er gemt i listen.
Beskrivelse	Bruges til at se hvor mange elementer der er gemt i listen.

Tabel 78: GetItemsInList

5.3.3 Indstillinger

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 79: Attributter for klassen Indstillinger

Metoder

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

Tabel 80: exec

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

Tabel 81: SetMonitorering

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

Tabel 82: SetRegulering

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

Tabel 83: getRegulering

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

Tabel 84: getMonitorering

Prototype	<code>void SetVirtualPlant(int id, Plant plantToPlace)</code>
Parametre	<code>int id</code> et id som er mellem 1 og 6 som svare til hvor planeten er i det Virtual drivhus. <code>Plant plantToPlace</code> Planten som skal sættes ind i det virtual drivhus.
Returværdi	-
Beskrivelse	Bruges til at indsætte en plante ind i det virtuelle drivhus. På den ønskede placering

Tabel 85: 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	-
Beskrivelse	Anvendes til at slette en virtuel plante i det virtuel drivhus.

Tabel 86: DelVirtualPlant

Prototype	<code>void GetEmails(string &mail1, string &mail2, string &mail3)</code>
Parametre	<code>string &mail1</code> En reference til en string, son ændres til E-mail et som er gemt i system. <code>string &mail2</code> En reference til en string, son ændres til E-mail to som er gemt i system. <code>string &mail3</code> En reference til en string, son ændres til E-mail tre som er gemt i system.
Returværdi	-
Beskrivelse	Bruges til at hente de tre e-mail adresser som systemet har gemt.

Tabel 87: GetEmails

Prototype	<code>void SetEmails(const string mail1, const string mail2, const string mail3)</code>
Parametre	<code>string &mail1</code> E-mail et som der kan ønskes gemt. <code>string &mail2</code> E-mail to som der kan ønskes gemt. <code>string &mail3</code> E-mail et som der kan ønskes gemt.
Returværdi	-
Beskrivelse	Bruges til at gemme tre e-mail adresser i systemet.

Tabel 88: 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 må bruges til at regulere. Hvis den er true må hardwaren bruges hvis false må det ikke bruges til at regulere med.

Tabel 89: 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 regulerig i drivhuset.

Tabel 90: SetHardware

Prototype	<code>void setDate(Date time)</code>
Parametre	Date time En struct af type Date som beskriver den tid som systemet skal indstilles til.
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 91: 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 92: getDate

Prototype	<code>void GetNotifications(bool &daily, bool &Warning)</code>
Parametre	bool &daily 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. bool &Warning 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.
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 93: GetNotifications

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

Tabel 94: SetNotifications

Prototype	<code>Plant Getplant(int plantId)</code>
Parametre	<p><code>int plantId</code> Et id som beskriver hvilken plante der skal hentes ud. Id'et skal være et tal mellem 1 og 6.</p>
Returværdi	Plant
Beskrivelse	Anvendes hente en plante ud af det virtuelle drivhus.

Tabel 95: Getplant

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

Tabel 96: GetAll

5.3.4 Monitor

Monitor er tråd styret, hvilket betyder den kan arbejde samtidigt med fra andre software dele i AutoGreen systemet. Tråden virker som en grænseflade mellem DevKit8000 og sensorer fra drivhus klimaet. Dens primære opgave er at finde ud af, hvilke sensorer er tilkoblet og udvinde data fra dem mindst 1 gang i minuttet og derefter sende det indhentede data til dataloggen. På grund af Monitor konstant anmoder om data, skal Monitor loope uendeligt fra DevKit8000 starter tråden indtil DevKit8000 afslutter tråden, dog kan monitor klassen sættes til at sove, ved styring af hovedmenuen.

Attributter

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

Tabel 97: Attributter for klassen Monitor

Metoder

Prototype	<code>void CompareData()</code>
Parametre	-
Returværdi	-
Beskrivelse	CompareDatas opgave er en live opdatering af plantesundhedstegn 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 98: CompareData

5.3.5 Plantedatabase

Plantedatabasen er en klasse som anvendes til at gemme planter i. Den er indeholder en Doublylinkedlist. Ud over dette tilbyder den metoder så man kan interagere med den.

Attributter

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

Tabel 99: Attributter for klassen Plantedatabase

Metoder

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

Tabel 100: GetPlantList

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

Tabel 101: GetPlant

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

Tabel 102: InsertPlant

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

Tabel 103: DeletePlant

Prototype	<code>plant CreatePlant()</code>
Parametre	-
Returværdi	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 104: CreatePlant

5.3.6 Rapport

Rapport sender beskeder til bruger gennem E-mail, så frem brugeren har aktiveret dette i Indstillinger. Klassen kan sende 2 forskellige beskeder til brugeren: En dagligt status af alle systemhændelser sket siden sidste status E-mail, samt en kritisk E-mail med en systemhændelse, der kræver brugerens indgreb. Den daglige status sendes på et brugerdefineret tidspunkt, mens den kritiske status-besked sendes ved første mulighed.

Private attributter

Navn	Type	Beskrivelse
CurrentTime	Date	Indholder sidste tidpunkt modtaget fra Indstillinger.
Email []	String	Indholder mailingsliste for status emails.

Tabel 105: 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 106: 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 107: SendDailyRapport()

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

Tabel 108: SendWarningRapport()

5.3.7 Regulator

Regulatoren er en klasse der køres i sin egen tråd på systemet. Regulatoren går ind og læser data-loggen en gang i minuttet, og tjekker at alle data er inden for tolerance niveauerne, i forhold til de ønskede som aflæses i Indstillings-klassen. Hvis noget data ligger for langt uden for tolerance niveaueret har regulatoren mulighed for at sende besked til PSoC masteren ved brug a UARTen, om at åbne eller lukke for drivhusvinduet, tænde eller slukke for en blæser, og tænde og slukke for et varmelegeme. Regulatoren er en evighedsløkke der kun stoppes ved at afslutte den på QT-hovedmenuen.

Når regulator kører, starter den ud med at hente data fra data-loggen og gemmer dem i interne variabler, hvorefter der bliver hentet data ind fra indstillingsklassen. Herefter køres ControlData().Regulatoren bruger derefter UARTen til at tilrette klimaet i drivhuset efter behov. Herefter lægger Regulatoren sig til at sove i et minut, hvorefter den kører starter forfra.

Attributter

<code>plante1..6</code>	<code>Plant</code>	plante 1..6 er de virtuelle planter der lagres her, de hentes ind fra indstillinger.
<code>uart</code>	<code>UART *</code>	En pointer til UARTen, så den kan bruges til at sende beskeder til pSoC masteren.
<code>systemlog</code>	<code>SystemLog *</code>	En pointer til systemloggen, så der kan skrives beskeder til systemloggen om evt. fejl eller handlinger fra regulatoren.
<code>datalog</code>	<code>DataLog *</code>	En pointer til dataloggen, så regulatoren har mulighed for at tilgå de nyeste data fra den og bruge til at tilpasse klimaet.
<code>settings</code>	<code>Indstillinger *</code>	En pointer til Indstillingsklassen, så information om systemet og de virtueller kan tilgåes.

Tabel 109: Attributter for klassen Regulator

Metoder

<code>Prototype</code>	<code>Void Run()</code>
<code>Parametre</code>	-
<code>Returværdi</code>	-
<code>Beskrivelse</code>	Run har et formål at fungere som regulatoren og kører evigt. Den henter data ind om de virtuelle planter fra drivhuset, og bruger herefter ControlData funktionen til at til.

Tabel 110: Run

Prototype	<code>void ControlData(SensorData drivhus_data)</code>
Parametre	<code>drivhus_data</code> Sensor data omkring det faktiske klima i drivhuset.
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. Hvis nogle af parametrene ligger for langt fra den ønskede kalder ControlData via UART pointeren, til UARTEn om at ændre klimaet i drivhuset.

Tabel 111: ControlData

Prototype	<code>void Loaddata()</code>
Parametre	-
Returværdi	-
Beskrivelse	LoadData er en hjælpefunktion, som bruges til at kunne loade de virtuelle planter fra indstillinger ind i plante-objekterne der befinner sig i regulatoren.

Tabel 112: LoadData

5.3.8 SystemLog

Systemloggen anvendes til at gemme data omkring systemhændelser angående: ændringer i den virtuelle og normale plantedatabase, aktivering af aktuatorerne, samt sendte beskeder til brugeren. Systemloggen kan tilgås af brugeren i Systemlogmenuen.

Attributter

SystemMsg	DoublyLinkedList <string>	SystemMsg implementeres med en DoublyLinkedList, som gemmer i alle systemhændelser i form af strings.
-----------	---------------------------	---

Tabel 113: Attributter for klassen SystemLog

Metoder

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

Tabel 114: 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 115: PrintSystemLog

5.3.9 UART

UART er en klasse til at sende og modtage data fra PSoC masteren. Til kommunikation bruges UART protokollen (REF til UART protokol).

UARTen opbygges ved hjælp af et open source bibliotek (ref), som allerede har funktioner til sending og modtagelse af data.

Attributter

-	<code>SystemLog *</code>	En pointer til udskrivelse til systemloggen.
---	--------------------------	--

Tabel 116: 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 fejl forsøg afsluttes metoden, og systemet sender en besked til bruger at tjekke systemet.

Tabel 117: ScanForSensors

Prototype	<code>SensorData GetSensorData()</code>
Parametre	-
Returværdi	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 118: GetSensorData

Prototype	<code>void ActivateSensor(String Command_)</code>
Parametre	Command_ commando er enten heaton/heatoff for at tænde og slukke for heater, windowon/windowoff for at åbne og lukke for vinduet, venton/ventoff for at tænde og slukke for blæseren.
Returværdi	-
Beskrivelse	-

Tabel 119: ActivateSensor

Prototype	<code>int RecieveData()</code>
Parametre	-
Returværdi	Den returnerede værdi er svarende til en værdi udreget ud fra hvilken char der modtages, f.eks. hvis T modtages som første byte og derefter modtages en char svarende til 'A' (ascii) returnes 65, da det er værdien for 'A'.
Beskrivelse	RecieveData kaldes og står derefter og læser på UARTEn, ud fra UART protokollen, er det bestemt hvilke sammensætninger af bogstaver der ledes efter, og når 2 bytes der passer til protokollen er fundet, returne funktionen.

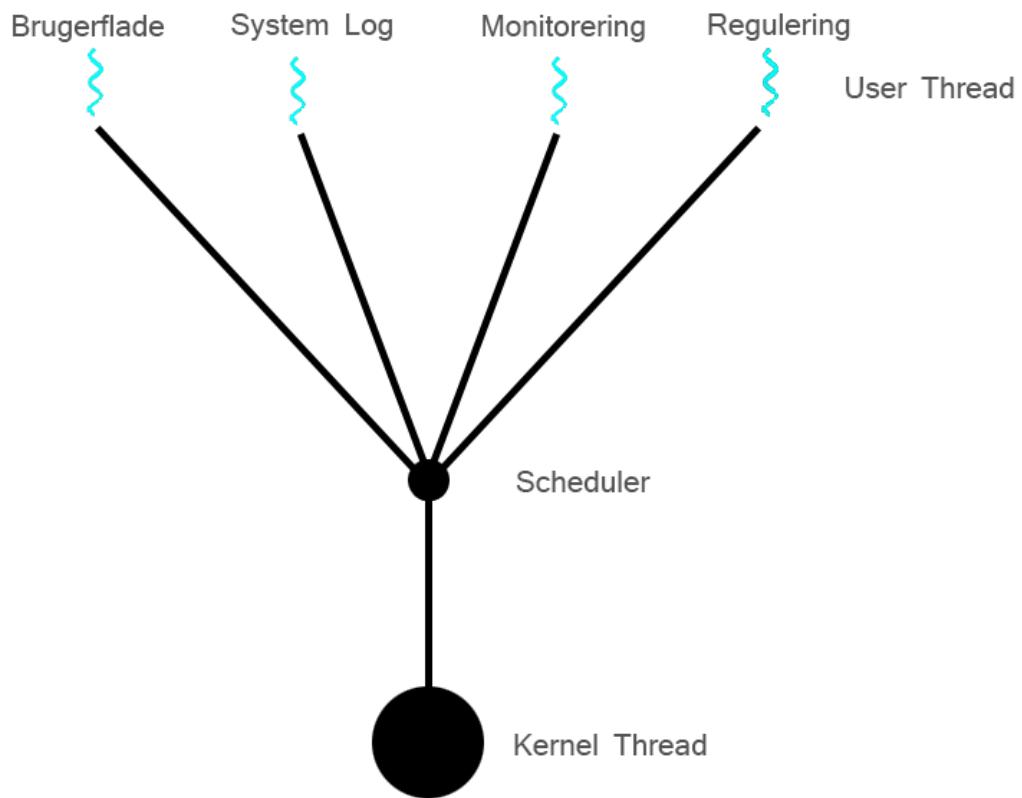
Tabel 120: RecieveData

Prototype	<code>Void SendData(string command_)</code>
Parametre	command_ Er kommandoen der skal sendes over til pSoC masteren via UART protokollen.
Returværdi	-
Beskrivelse	funktionen har til formål at oversætte sætninger. F.eks. 'heaton' til det der skal sendes via UART protokollen. Den oversætter simple sætninger/ord til et enkelt Byte eller to, som er passer til hvad UART protokollen, og sender derefter den forkortede besked over UART.

Tabel 121: SendData

5.4 Trådhåndtering

For at effektivisere systemet anvendes tråde til håndtering af forskellige opgaver. Der anvendes en tråd til at styre den overordnede brugerflade, en til at styre regulering af systemet, en til monitorering af klimaet i drivhuset samt en til at håndtere system loggen. Ved at bruge tråde kan alle disse funktionaliteter køres parallelt. Dette giver en højere hastighed i systemet da funktionerne ikke skal vente på hinandens eksekvering. På Figur 45 er der en visuel beskrivelse af disse tråde. Yderligere beskrivelse af tråde kan findes under literaturliste [14].



Figur 45: Thread opsætning af systemet

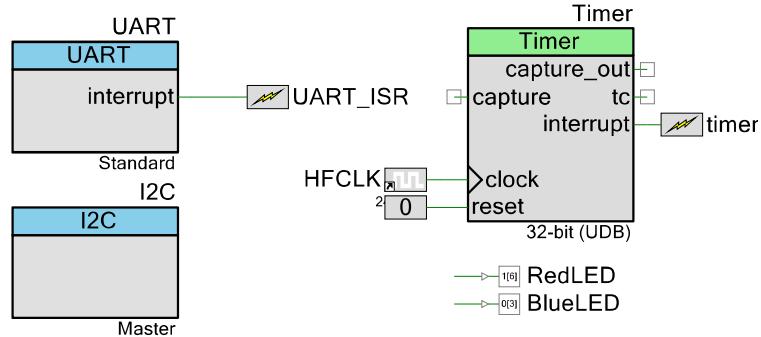
6 Hardware Implementering

Version

Dato	Version	Initialer	Ændring
31. marts	1	MHG	Implementering af SW i Aktuator.
8. april	2	MHG	Færdiggjort beskrivelse af SW i Aktuator.
17. april	3	PKP	PSoC Master implementering tilføjet.
27. april	4	HBJ	Opdateret beskrivelse af SW i Aktuator.
27. aril	5	MHG	Skrevet implementering af Mosfet drivere til Varmel- geme, Blæsere og Vinduesmotor.
5. maj	6	HBJ	Implementering af Jordfugt tilføjet.
8. maj	7	MHG	Rettelser i Implementering af Jordfugt.
11. maj	8	PKP	PSoC Master implementering rettet og opdateret.
21. maj	9	KT	Rettelse af note omkring UART.

6.1 PSoC Master

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



Figur 46: TopDesign.cysch for PSoC_Master

I Figur 6.1 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 med tilhørende clock, 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.

6.1.1 Main implementering

Main funktionen, der er vist i Listing 6.1, 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 119.

```

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 6.1: PSoC Master'ens main funktion

6.1.2 I²C implementering

I forbindelse med implementeringen af I²C klassen blev prototyperne oprettet jf. I²C protokollen (se side 44). 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.

```

216 int8 getTemp( int32* temp) {
217
218     uint8 dataget[2] = {0,0};
219     uint32 errorStatus[2] = {9,9};           // For debugging and error handling
220
221     I2C_I2CMasterClearReadBuf();
222     errorStatus[0] = I2C_I2CMasterSendStart(TEMP_SENSOR_ADDRESS,
223         I2C_I2C_READ_XFER_MODE);
224     if (errorStatus[0] == I2C_I2C_MSTR_NO_ERROR) {
225         dataget[0] = I2C_I2CMasterReadByte(I2C_I2C_ACK_DATA);
226         dataget[1] = I2C_I2CMasterReadByte(I2C_I2C_NAK_DATA);
227         errorStatus[1] = I2C_I2CMasterSendStop();
228     }
229     else {
230         I2C_I2CMasterSendStop();
231         *temp = -1;
232         return -1;
233     }
234     // The data is converted directly to UART protocol because of the ,5 resolution
235     *temp = (dataget[0]*2)+(dataget[1] >> 7)+40;
236
237     return 0;
238 }
```

Listing 6.2: Implementering af getTemp()

I Listing 6.2, 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 44, 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 oplosning.

Fra aktuatorens 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_ADRESS, dataget, 2,
170                                     I2C_MODE_COMPLETE_XFER);
171
172     while (0u == (I2C_I2CMasterStatus() & 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)) {

```

Listing 6.3: Udsnit af getActuatorStatus()

Det kan ses på Listing 6.3 linje 171, 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.

6.1.3 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 6.1.

```

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         return -1;
37     }

```

Listing 6.4: Implementering af respondTemp()

I Listing 6.4 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 6.1.4.

```

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

Listing 6.5: Implementering af dkRequest()

Vi har ydermere valgt at indkapsle læsningen fra UART ved hjælp af dkRequest() funktionen vist i Listing 6.5. 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.

6.1.4 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 6.6. 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 6.6: 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(globel)
144 }
```

Listing 6.7: Funktion til at indlæse en sensorværdi i DSP klassen

I Listing 6.7 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_DATAPOINTS_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 6.8: Funktion der midler og konverterer tempArray

I Listing 6.8 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 39. 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 6.9.

```

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

Listing 6.9: 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 [11] for mere information om dette.

6.1.5 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 6.10: 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 6.11.

```
81 // UART ISR
82 CY_ISR(UART_ISR){
83     uartInt = 1;
84     BlueLED_Write(LED_ON);           // Turn on blue LED
85     buff = dkRequest();
86     UART_ClearRxInterruptSource(UART_GetRxInterruptSourceMasked());      // Clear
87     interrupt flag
}
```

Listing 6.11: 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 6.1 på side 114.

```
91 void uartIntHandler(void){
92     if (uartInt){
93         uartInt = 0;
94
95         if (theState == IDLE){
96             switch (buff){
97                 case 'T':{ //RequestTemp
98                     respondTemp(getTemp_DSP());
99                     break;
100                }
101            }
102        }
103    }
104 }
```

Listing 6.12: Interrupt handler for UART.

I Listing 6.12 ses den indledende del af 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.

```

135     else if(theState == ADJW){
136         if(buff-CONVERT_TO_ASCII == 1){
137             respondWin(adjustWindow(0xFF));
138         }
139         else{
140             respondWin(adjustWindow(0x00));
141         }
142         theState = IDLE;
143     }

```

Listing 6.13: Interrupt handler for UART.

I Listing 6.13 ses endnu et udsnit af `uartIntHandler`. Her er metoden kaldt i et stadie, hvor systemet venter på en parameter til kommandoen 'W', der oversættes til åbning/lukning af vinduet.

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`.

```

186 void timerIntHandler(void){
187     if(timerInt){
188         timerInt = 0;           // Reset flag
189
190         // Measure temp and input to DSP class
191         getTemp(&tempTemp);
192         inputTemp(&tempTemp);
193
194         // Measure soilhum and input to DSP class
195     {
196         uint8 i;
197         for(i = 0; i<6 ; i++){
198             getSoilHum(i , &tempSoilHum[ i ]);
199             inputSoilHum(i , &tempSoilHum[ i ]);
200         }
201     }
202
203     RedLED_Write(LED_OFF);      // Turn off red LED
204 }
205 }

```

Listing 6.14: Interrupt handler for timer.

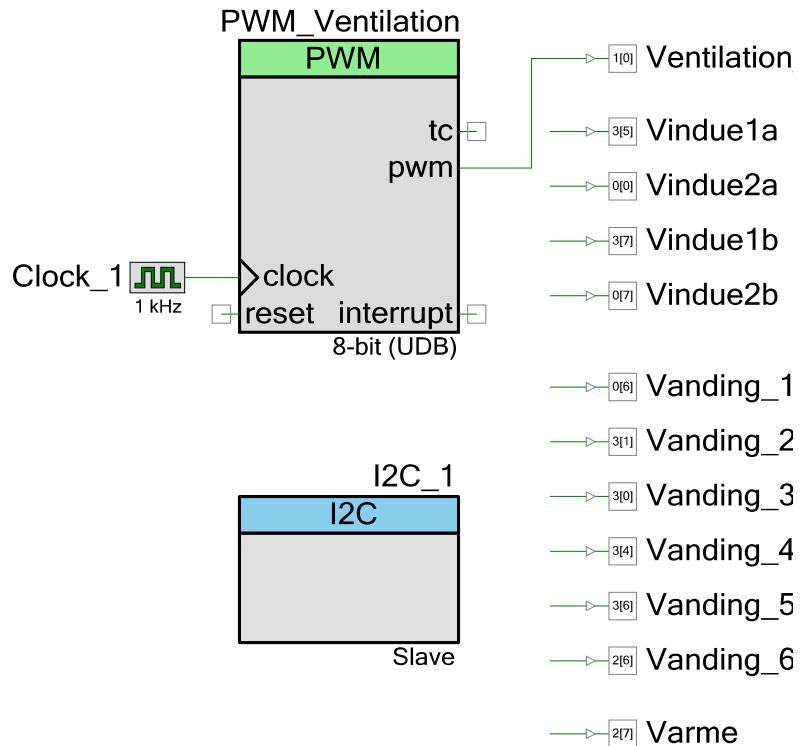
I Listing 6.14 ses selve implementeringen af `timerIntHandler`. Den har, ligesom `uartIntHandler`, en LED der tændes og der udføres herefter selve arbejdet, som består i at fodre data ind i vores digitale signalprocessor (DPS-klassen). Data'en kommer fra funktioner i I2C-klassen, der henter selve dataen fra vores sensorer på I²C bussen.

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

6.2 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.

6.2.1 HW PSoC4



Figur 47: TopDesign.cysch for PSoC4 i Aktuator

Den HW, der syntetiseres i PSoC4 Aktuator er vist på figur47.

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

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

I2C_1 komponenten styrer kommunikation med PSoC Master 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.

6.2.2 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 & 0b1110000000000000) == (Position_Reg & 0
30           b1110000000000000)))
31      {
32        AdjustWindow();
33      }
34    }
35  }

```

Listing 6.15: Udsnit af main.c for PSoC4 i Aktuator

Filen main.c, hvoraf den vigtigste del er vist på Listing 6.15, fungerer jf. State Machinen på Figur 26 side 71.

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 vinduet.

```

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            /* ... */
20        }
21        if((writeBuffer[0] >> 6) == 0x2) //Check for Ventilation
22        {
23            /* ... */
24        }
25        if((writeBuffer[0] >> 6) == 0x3) //Check for Vandning
26        {
27            /* ... */
28        }
29        I2C_1_I2CSlaveClearWriteBuf(); //Clear buffer pointer
30        I2C_1_I2CSlaveClearWriteStatus(); //Clear status
31        //Opdater Read buffer
32        readBuffer[0] = Status_Reg >> 8;
33        readBuffer[1] = Status_Reg;
34        I2C_1_I2CSlaveClearReadBuf();
35    }
36 }
```

Listing 6.16: Udsnit af checkForData.c for PSoC4 i Aktuator

Funktionen checkForData() på Listing 6.16 checker om slavens status er, at den har modtaget data. Halvdelen af koden er ikke vist, da det stort set er den samme kode, der bliver brugt under check for vindue som for check for varme, ventilation og vanding.

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 PSoC Master beder om information om aktuel status.

```
1 void InitHeat()
2 {
3     //Slukker for Varme
4     Varme_Write(0);
5
6     //Opdater nuvaerende indstillinger
7     Position_Reg = Position_Reg & 0b1111000111111111;
8 }
9
10 void AdjustHeat()
11 {
12     //Opdater aktuator for varmelegeme
13     Varme_Write((Status_Reg & 0b0000111000000000) >> 9);
14
15     //Opdater nuvaerende indstillinger
16     Position_Reg = Position_Reg & 0b1111000111111111;
17     Position_Reg = Position_Reg | (Status_Reg & 0b0000111000000000);
18 }
```

Listing 6.17: Udsnit af heat.c for PSoC4 i Aktuator

Koden i heat.c i Listing 6.17 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 & 0b11111100011111;
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 & 0b000000011100000) >> 6)*
18         MAXIMUM_VENT*255)/(7*100));
19     //Opdater nuvaerende indstillinger
20     Position_Reg = Position_Reg & 0b11111100011111;
21     Position_Reg = Position_Reg | (Status_Reg & 0b000000011100000);
22 }
```

Listing 6.18: Udsnit af ventilation.c for PSoC4 i Aktuator

Koden i ventilation.c i Listing 6.18 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 & 0b0000000000000010) >> 1);
20     Vanding_3_Write((Status_Reg & 0b00000000000000100) >> 2);
21     Vanding_4_Write((Status_Reg & 0b000000000000001000) >> 3);
22     Vanding_5_Write((Status_Reg & 0b00000000000010000) >> 4);
23     Vanding_6_Write((Status_Reg & 0b0000000000100000) >> 5);
24
25    //Opdater nuvaerende indstillinger
26    Position_Reg = Position_Reg & 0b1111111111000000;
27    Position_Reg = Position_Reg | (Status_Reg & 0b0000000000111111);
28 }
```

Listing 6.19: Udsnit af irrigation.c for PSoC4 i Aktuator

Koden i irrigation.c i Listing 6.19 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     Vindue1a_Write(1);
5     Vindue2a_Write(1);
6     Vindue1b_Write(0);
7     Vindue2b_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 indsaettelse i desiredTurn .
17     desiredTurn = ((MAX_WINDOW)*(((Status_Reg >> 12)*100)/15))/100;
18
19     while(desiredTurn != currentTurn) //Saa laenge vinduet ikke er i oensket position
20     {
21         if (currentTurn > desiredTurn) //Luk 1 omgang hvis vinduet er for aabent
22         {
23             CloseOneTurn();
24         }
25
26         if (currentTurn < desiredTurn) //aabnen 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 6.20: Udsnit A af window.c for PSoC4 i Aktuator

Kodeudsnittet fra window.c på Listing 6.20 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 befinder 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().

Herved opdateres Position_Reg.

```

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

```

Listing 6.21: Udsnit B af window.c for PSoC4 i Aktuator

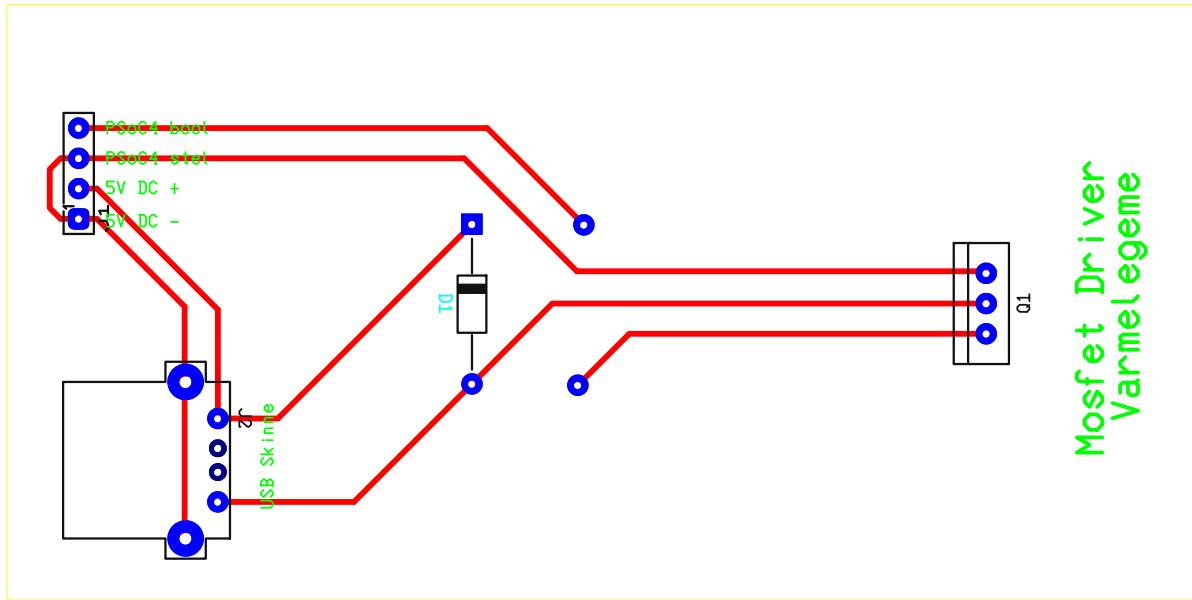
Kodeudsnittet fra window.c på Listing 6.21 viser de to funktioner CloseOneTurn() (og OpenOneTurn()).

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 for at en ny kommando til vinduet eksekveres inden en igangværende kommando afsluttes. Herefter opdateres currentTurn. Denne funktionalitet er implementeret i begge funktioner.

6.2.3 HW Varmelegeme



Figur 48: Printudlæg for Mosfet driver til Varmelegeme i Ultiboard

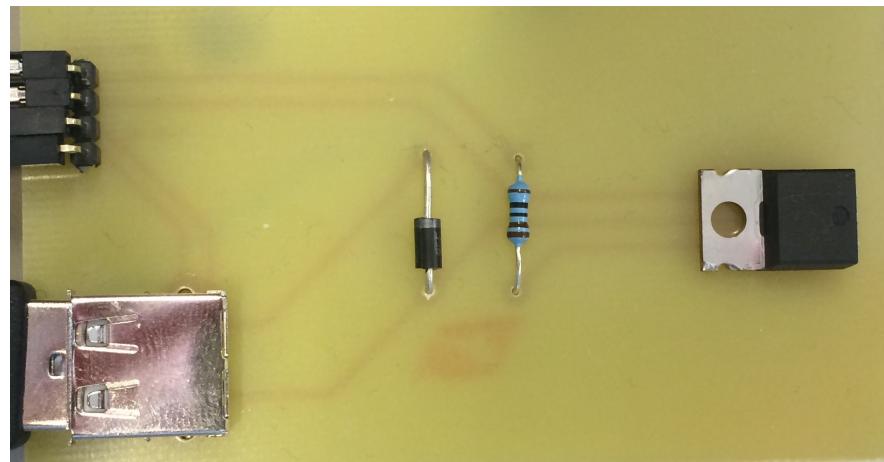
Implementering af mosfet driveren til varmelegemet foretages ved at Multisim diagrammet eksporteres til Ultiboard, hvorefter printet udlægges. Det forsøges gjort således at printet er overskueligt fremfor at printet fylder så lidt som muligt. Som udgangspunkt er alle forbindelser på printet lagt på bagsiden. Det designede print er vist på Figur 48.

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 loddedes 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.

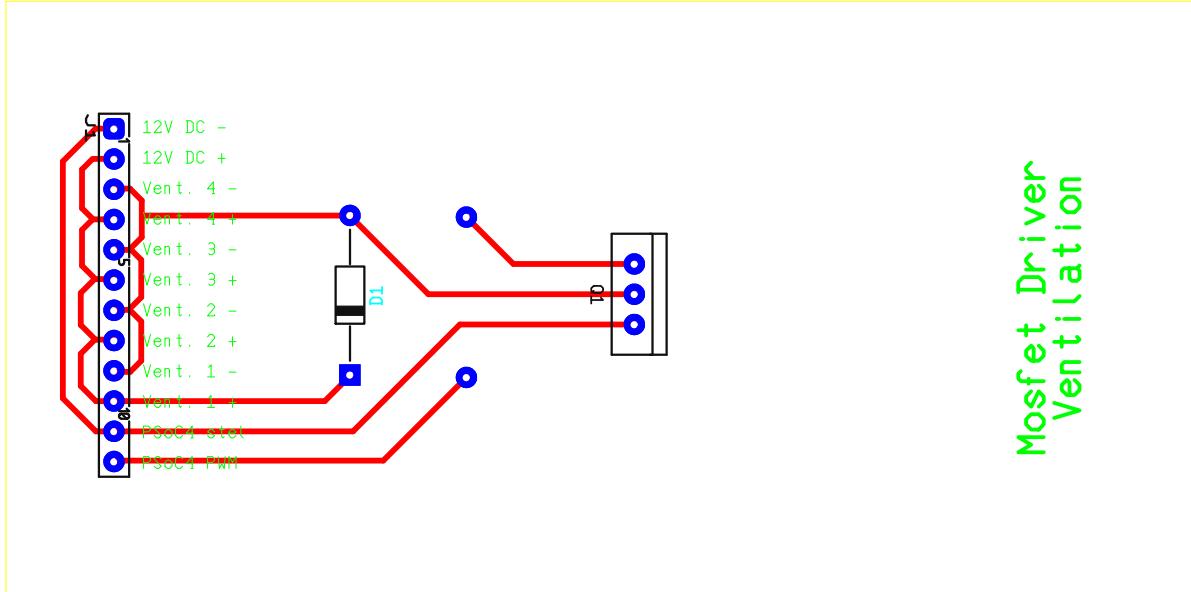
Printudlægget bestilles ved E-LAB på IHA, hvorefter komponenter loddes på. Det færdige print er vist på Figur 49.



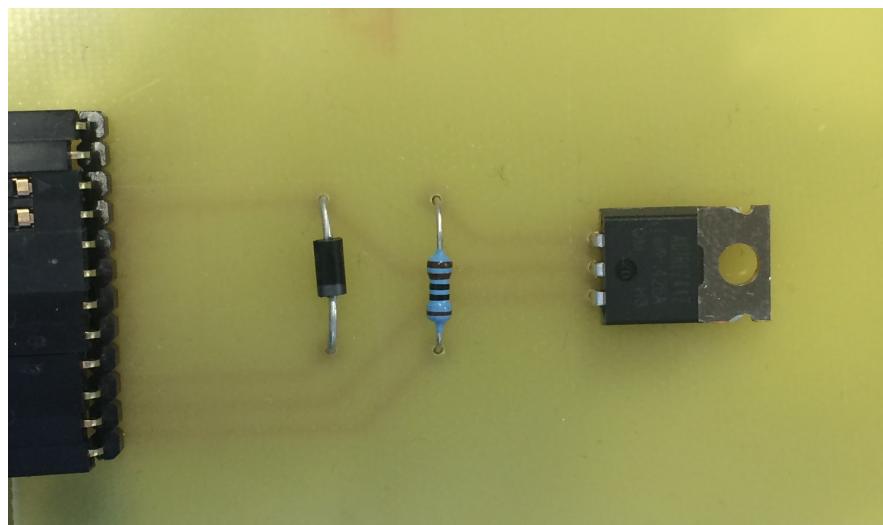
Figur 49: Den færdige Mosfet driver til Varmelegeme

6.2.4 HW Blæsere

Implementering af Mosfetdriver til Blæsere foretages på samme måde som til Varmelegeme. Printudlægget i Ultiboard er vist på Figur 50, og det færdige print er vist på Figur 51.



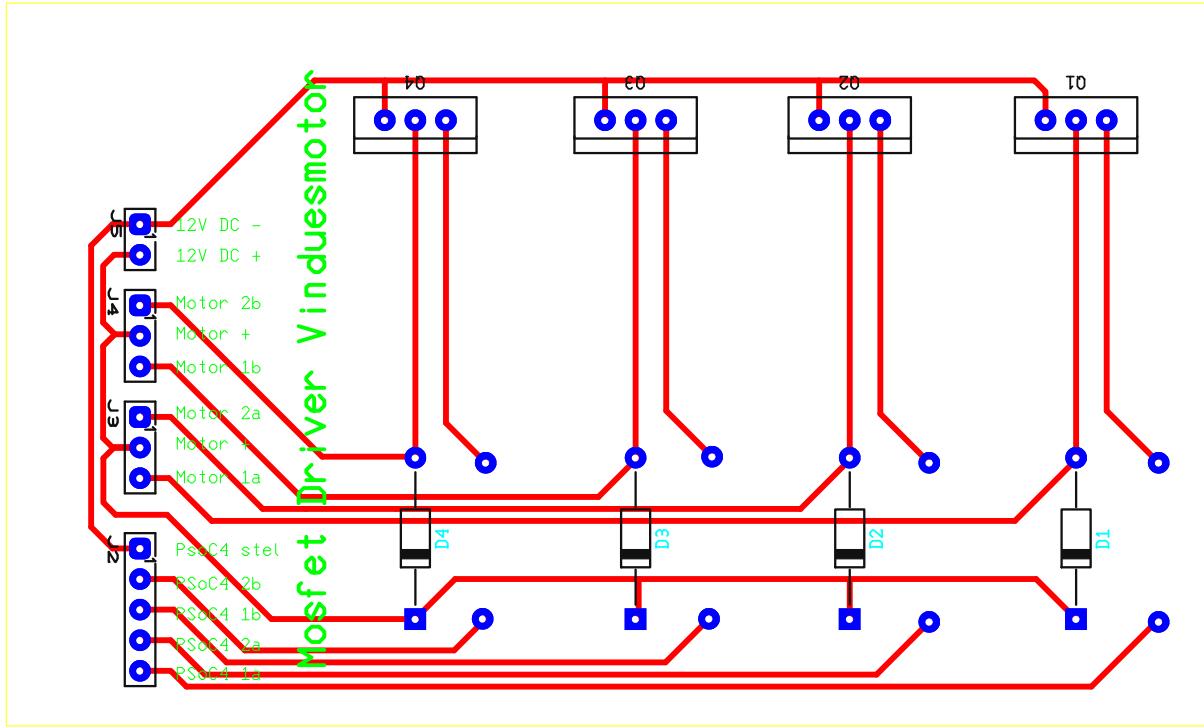
Figur 50: Printudlæg for Mosfet driver til Blæsere i Ultiboard



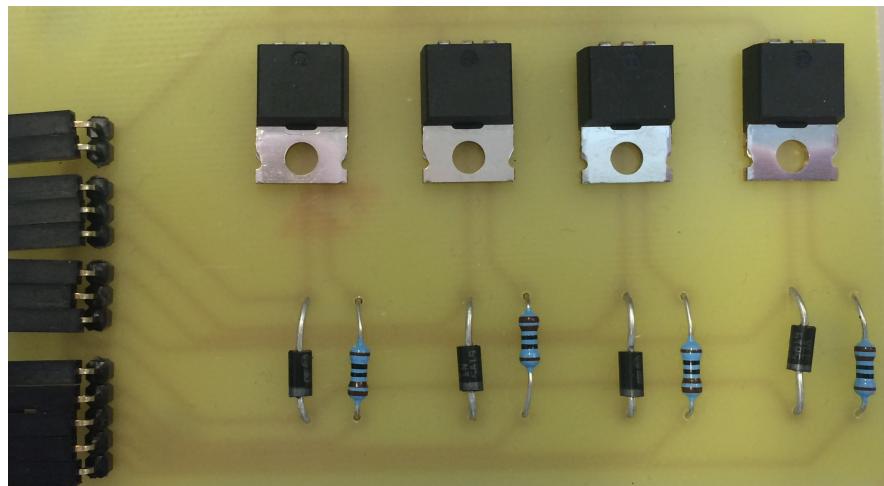
Figur 51: Den færdige Mosfet driver til Blæsere

6.2.5 HW Vinduesmotor

Implementering af Mosfetdriver til Vinduesmotor foretages på samme måde som til Varmelegeme og Blæsere. Printudlægget i Ultiboard er vist på Figur 52, og det færdige print er vist på Figur 53.

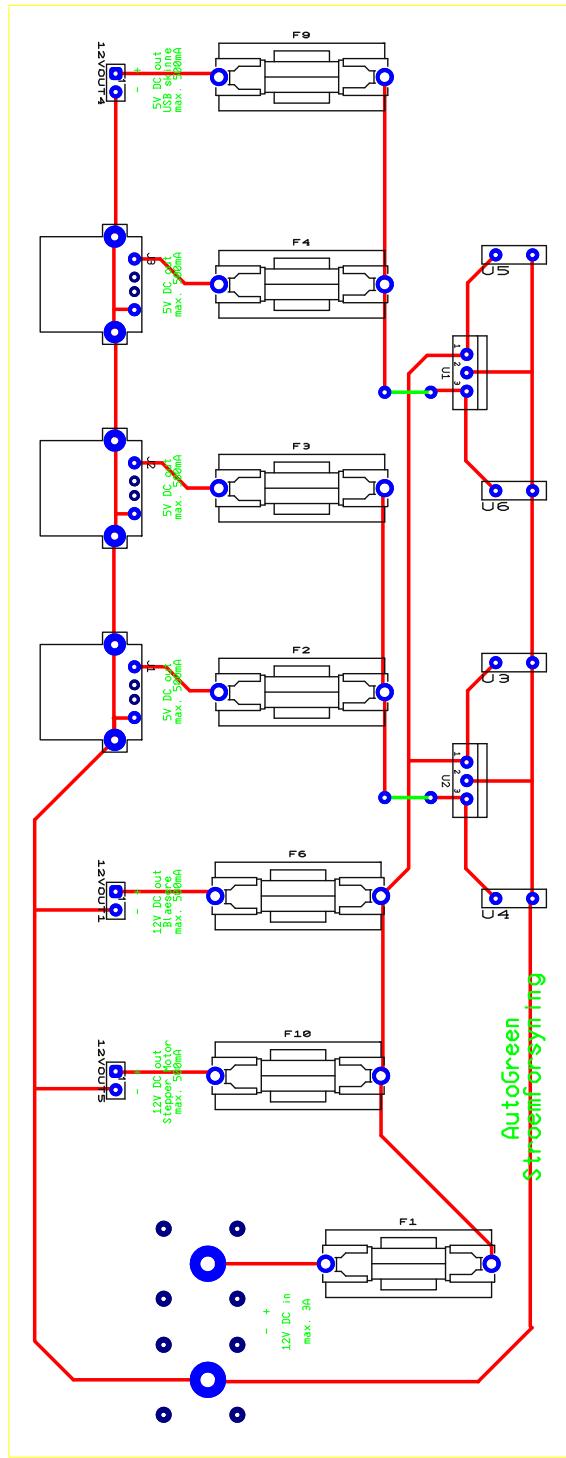


Figur 52: Printudlæg for Mosfet driver til Vinduesmotor i Ultiboard



Figur 53: Den færdige Mosfet driver til Vinduesmotor

6.3 Strømforsyning



Figur 54: 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 54.

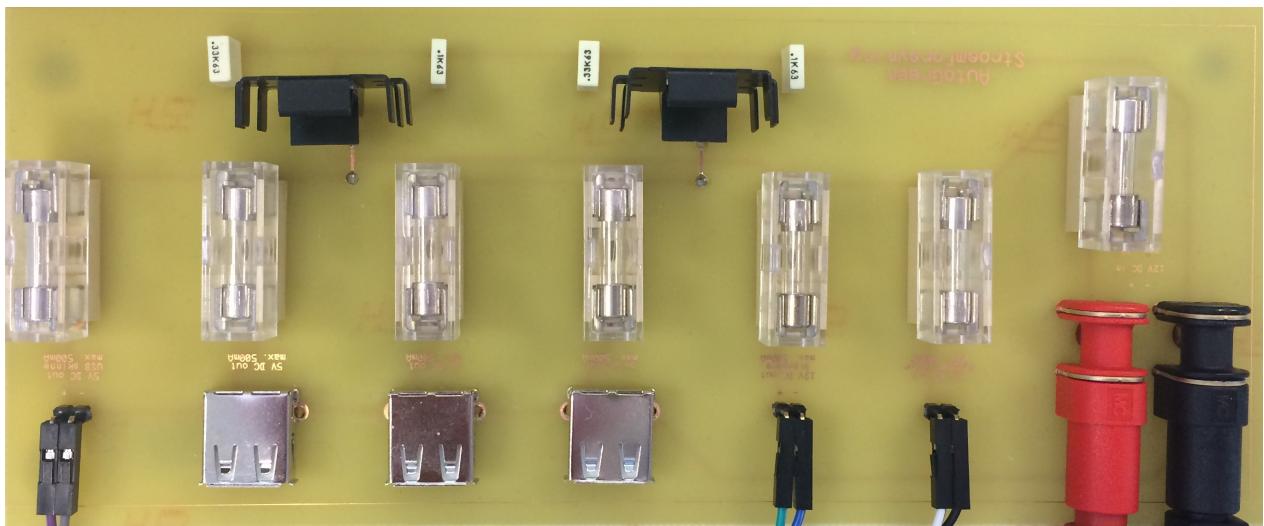
Kobber på undersiden af printet er vist med rød, mens kobber på oversiden af printet er vist med grøn. Kobberører er vist med blå. Der er en lille hage ved kobberørerne; der er ikke forbindelse mellem oversiden og undersiden. Derfor må man lodde et lille stykke monteringstråd igennem printet for at skabe 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 55.

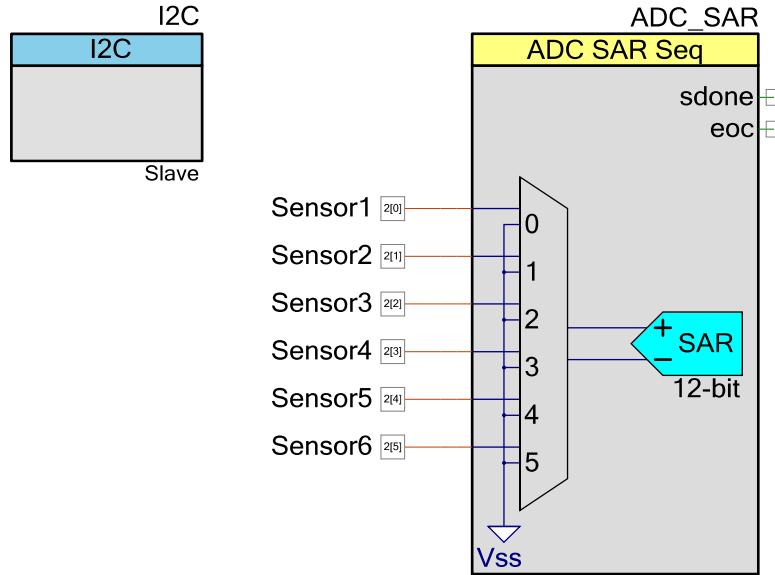


Figur 55: Den færdige Strømforsyning

6.4 Jordfugt

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

6.4.1 HW PSoC4



Figur 56: TopDesign.cysch for PSoC4 i Jordfugt

Den HW, der syntetiseres i PSoC4 Jordfugt er vist på Figur 56.

Komponenten I2C styrer kommunikation med PSoC Master 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. Det er overvejet hvorvidt det har kunne betale sig at gøre indgangene på ADC'en differentielle for at undertrykke støj. Dog har det gennem MSE øvelse 6 [9] vist sig, at den støj der kommer fra jordfugtsensoren er så ubetydelig, at det ikke har noget at sige, og derfor er der valgt single-ended for at undgå længere konverteringer.

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

6.4.2 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; //Goer data klar til aflæsning
18            I2C_I2CSlaveClearReadBuf(); //Reset read buffer pointer
19            i = 0xFFFFFFFF; //Variabel i sættes til en værdi uden for sensor numre
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 6.22: Udsnit af main.c for PSoC4 i Jordfugt

Filen main.c, hvoraf den vigtigste del er vist på Listing 6.22, fungerer jf. State Machinen på Figur 28 side 75.

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.

7 Software Implementering

Version

Dato	Version	Initialer	Ændring
7. maj	1	KT	Første udkast.
18. maj	2	PKP	Tilføjet DoublyLinked List og Grundsystem afsnit

7.1 System structs

7.1.1 Sensordata

Sensor data er en simpel struct der bruges til at lagre, temperatur, lysintensitet, luftfugtighed og jordfugtighederne for de 6 planter. Den indeholder også en Date struct, som indeholder tid og dato.

```

1 struct SensorData
2 {
3     Date time;
4     double temp;
5     int light;
6     int humidity;
7     int grund[6];
8 };

```

Listing 7.1: Sensordata-structen.

Ideen bag SensorData structen er at kunne returnere mere end en parameter adgangen fra UARTEn, ved at returnere et SensorData object kan alle data om drivhuset returneres, uden brug af referencer.

7.1.2 Date

Date structen er en struct, brugt til at have samling på data der har med tid og dato at gøre.

```

1 struct Date
2 {
3     int Min;
4     int Hour;
5     int Day;
6     int Month;
7     int Year;
8 };

```

Listing 7.2: Date-structen.

Date indeholder, minut-tal, time-tal, dag på måned(mellem 1-31), og hvilken måned det er(1-12), samt årstal. Formålet er at kunne bruge til lagering af Sensordata, så det kan ses hvornår en given data er fra.

7.1.3 Plant

Plant structen er en simpel struct der indeholder data over en plante. den indeholder plantens navn, plantes ønskede temperatur, lysintensitet, luftfugtighed og jordfugtighed.

```

1 struct Plant
2 {
3     string name;
4     int temp;
5     int light;
6     int hum;
7     int water;
8 };

```

Listing 7.3: Plant-struct.

Ideen med Plante structen er at kunne sende structen rundt i systemet uden at skulle give alle de individuelle parametre med. Den er valgt at blive lavet som en struct, frem for en klasse, da der ikke er yderligere mening med den, end at lagre data samlet.

7.2 DoublyLinked List

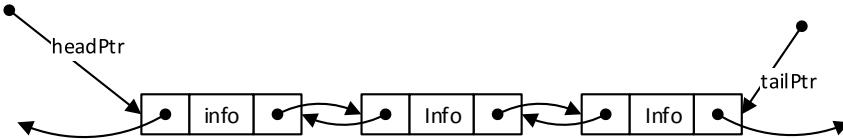
7.2.1 Valg af datastruktur

Faget *Datastrukturer og Algoritmer* skulle indgå i projektet og derfor var det nødvendigt at vælge en datastruktur, som passede godt det til det vi lavede. I stedet for blot at anvende en præfabrikeret i *STL*, ønskede vi at lave en datastruktur selv. Standard strukturen *vektor* var en mulighed, men den blev valgt fra da en *Doubly Linked List* (fremover benævnt DLL) er velegnet til at indsætte, fjerne og hente data fra front eller bag. Da der i projektet var brug for at indsætte og hente data ofte, faldt valget på denne datastruktur.

7.2.2 Forklaring af Doubly Linked List

En DLL er en udbygning af datastrukturen *Linked List* og denne udbygning, har stedet for kun at have ét info felt til data og én next-pointer til det næste element i listen, har den også en previous-pointer. Med denne ekstra pointer, er det således muligt at gå frem og tilbage i listen, hvorimod en normal linked list kun kan gå frem. Klassen er implementeret som en template klasse, så alle simple typer kan bruges. I tilfældet at typen er kompleks, kan der implementeres overloads på equal-operatoren og ». Konceptet kan være lidt svært at forstå, men tænk på det som en analogi: Et tog. Programmøren gemmer altid det første knudepunkt på listen. Dette svarer til at være motoren i toget. Pointeren er koblingen, der sidder mellem vognene på toget. Hver gang toget tilføjer en vogn, bruger vi koblingen til at tilføje en ny vogn.

I Figur 57 er der vist et billede som viser hvordan tre noder er linket sammen.



Figur 57: Sammenhængen mellem nodes.

7.2.3 HeadInsert

I Listing 7.4 ses metoden til at indsætte data i DLL'en.

```

44 void headInsert(A_Type info) {
45     Node<A_Type> *NewNode = new Node<A_Type>(info);
46     if (headPtr == NULL) //Hvis ikke er nogen liste.
47     {
48         headPtr = NewNode;
49         tailPtr = NewNode;
50     }
51     else //Hvis der er en headptr altså der er noget.
52     {
53         headPtr->prev = NewNode;
54         NewNode->next = headPtr;
55
56         headPtr = NewNode;
57     }
58     ++this->itemsInList;
59 }
```

Listing 7.4: Implementering af headInsert()

Som det fremgår af Listing 7.4, bliver headPtr og tailPtr sat til at pege på den nye node som er indsæt, hvis der ikke er lavet nogen liste. I det tilfælde at en liste allerede er konstrueret, sætter metoden den nye node headPtr lig med noden lige før den. En DLL har også en tailInsert, som kan indsætte data i enden af listen efter næsten samme princip.

7.2.4 GetItemInList

DLL'en holder også styr på, hvor mange elementer der er indsæt. Således er der kontrol over hvor lang listen er, hvilket bruges internt i klassen, så det ikke kan lade sig gøre at slette en node på plads 7 hvis der kun er 4 noder i hele listen. Metoden for at få vist hvor mange elementer der findes i listen, ses i Listing 7.5

```

192 int GetItemsInList()
193 {
194     return this->itemsInList;
195 }
```

Listing 7.5: Implementering af getItemInList()

For at denne metode virker korrekt, er det meget vigtigt at alle metoder, der laver en mutation i strukturen, opdaterer den private variabel: itemsInList som beskrives i tabel 65, så den hele tiden passer med det, der bliver udført. Dvs. hvis noget tilføjes, skal der adderes én til variablen, modsat hvis der fjernes der noget skal der subtraheres én. Det er også muligt at slette en node i en DLL via. metoden deleteAt(int place). Metoden kan ses i Listing 7.6.

7.2.5 DeleteAt

Det er også muligt at slette en node i Doubly linked listen via. Metoden: deleteAt(int place). Metoden kan ses i tabel 77

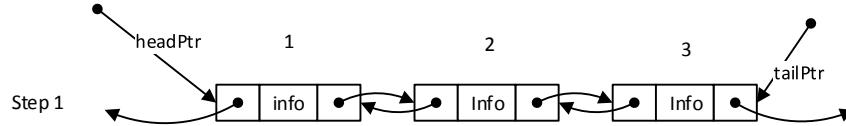
```

152     void deleteAt(int place) {
153         Node<A_Type>* cursor = headPtr;
154
155         for (int i = 1; i <= this->itemsInList; ++i) {
156
157             if (i == place) {
158                 if (cursor == headPtr) {
159                     headDelete();
160                     return;
161                 } else if (cursor == tailPtr) {
162                     tailDelete();
163                     return;
164                 } else {
165
166                     Node<A_Type>* cond = cursor; //pointer til den cond node.
167                     Node<A_Type>* prevPtr = cursor->prev; //pointer til den lige før con node.
168
169                     prevPtr->next = cond->next; //prevPtr next sættes til cons next.
170                     cond->prev = prevPtr->prev; //prevPtr next sættes til cons next.
171                     delete cond; //Sletter cond.
172                     --this->itemsInList;
173                     return;
174                 }
175                 cursor = cursor->next;
176             }
177         }
178
179         virtual void findAndDelete(A_Type valueToFind)
180     {
181         int found = Find(valueToFind);
182         if (found != -1) {
183             deleteAt(found);
184         }
185     }
186 }
```

Listing 7.6: Implementering af deleteAt(int place)

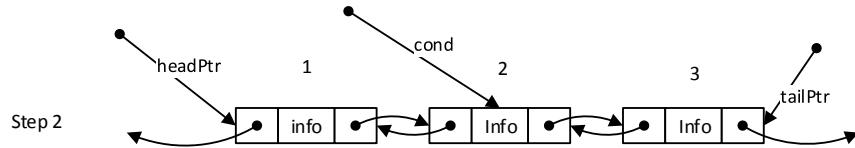
For at give et bedre overblik over hvad der sker i koden ses i Figur 58 og frem til 61 en grafik, som beskriver hvordan det foregår i 4 steps.

Dette er et eksempel på en DLL, hvor 3 noder er sammensat. Node 2 ønskes fjernet fra DLL'en.



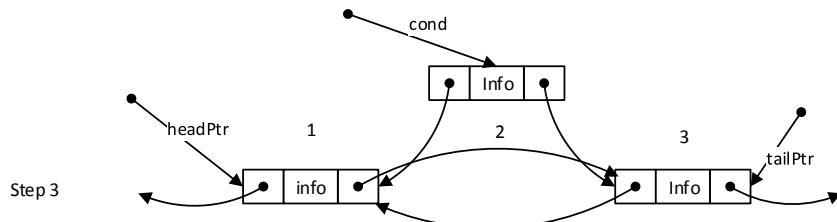
Figur 58: Tre noder, step 1.

Som det ses, er next-pointeren til node 1 flyttet, så den nu peger på node 3 og previous-pointeren fra node 3 peger nu på node 1.



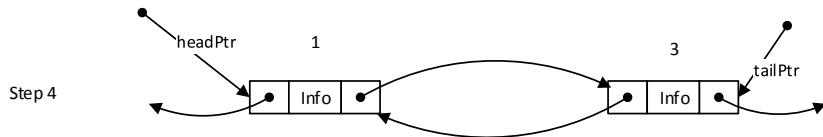
Figur 59: Ny pointer, step 2.

Der laves en ekstra pointer, som benævnes cond, og som sættes til at pege på node 2, således noden ikke mistes når step 3 flytter på pointere til node 1 og 2.



Figur 60: Flytning af pointere, step 3.

Dette betyder at node 2 nu er ude af systemet og kan slettes sikkert og uden at det ødelægger den listen. Efterfølgende er der kun node 1 og 3 tilbage og listen har nu fået fjernet node 2.



Figur 61: Sletning af node, step 3.

7.2.6 PeekHead

Metoden spørger efter det element som ligger gemt i fronten af listen. Metoden er særdeles vigtig, da det sikres det gemte materiale i listen kan hentes ud igen. Som der ses i metodesignaturen, returnerer metoden 1, hvis operationen gik godt og -1, hvis der var fejl. Metoden hvorpå der kan hentes elementer fra DLL'en kan ses i Figur 7.7.

```

138 int PeekHead(A_Type &PeekHead)
139 {
140     if (headPtr != NULL) {
141         PeekHead = headPtr->info;
142         return 1;
143     }
144     return -1;
145 }
```

Listing 7.7: Implementering af PeekHead.

7.3 Indstillinger

Indstillinger er en klasse, som holder styr på systemets indstillinger. Klassen kan også blive refereret til, som konfigurationsfil, enkelte steder. Klassens primære opgave er, at holde styr på tiden i systemet, de virtuelle planter i drivhuset samt hvilken hardware der må bruges til at regulere med. Tiden kunne implementeres på et par måder. Der kunne fx laves en counter, hvor der tælles op hvert sekund. Denne løsning blev dog ikke valgt, da vi anvender en embedded linux platform, som selv kan holde styr på tiden, hvis den indstilles korrekt. Der er dog nogle krav til at indstille tiden på den embeddede linux platform. Når tiden skal indstilles, skal der bruges ”super brugerrettigheder”, altså tiden skal sættes mens root anvendes. I Listing 7.8, ses noget af koden der indstiller tiden.

7.3.1 SetDate

Koden i Listing 7.8 opretter en pipe til linux’s commandline, og kører kommandoen `pwd` på linux platformen, som returnerer stien programmet køres i. Denne sti bruges til at eksekvere et script med de ønskede tidsindstillinger, således tiden sættes til det rigtige på systemet.

```

91 string prepCommand;
92 std::string pwd = exec("pwd"); // get current path
93
94 //looks for the return char and removes it
95 if (!pwd.empty() && pwd[pwd.length()-1] == '\n') {
96     pwd.erase(pwd.length()-1);
97 }
98 //concatenate string.
99 prepCommand += pwd + string("/setTimeBeagleBoard.sh ") + to_string(time.Year) +
100    string(" ") +
101    to_string(time.Month) + string(" ") + to_string(time.Day) + string(" ") +
102    to_string(time.Hour) +
103    string(" ") + to_string(time.Min);
104
105 const char * command = prepCommand.c_str();
106 std::string res = exec(command);
```

Listing 7.8: Implementering af setDate

Scriptet som koden afvikler, bruger kommandoen `Date`, som enten kan sætte tiden eller vise den. Kommandoen `Date` findes i en minimal version på DevKit8000, da linux distributionen angstrom, har brugt busybox. BusyBox kombinerer bittesmå versioner af mange almindelige UNIX utilities, i en enkelt lille eksekverbar fil. Det betyder at de ikke tager hele programmet med, kun det mest nødvendige af det. Koden kan ses i Listing 7.9.

```

2 #!/bin/bash
3 echo $1
4 echo $2
5 echo $3
6 echo $4
7 # for goldenImage
8 #date -s "2 OCT 2006 18:00:00"
9 #date -s "$1 $2 $3 $4:$5:00"
10 #For devkit date 2011.01.11-20:00
11 date $1.$2.$3-$4:$5

```

Listing 7.9: Implementering af setTimeBeagleBoard scriptet

7.3.2 Exec af linux kommandoer

I metoden exec, er det kode som får platformen til at køre kommandoer i linux's shell. Metoden tager et char array, der indeholder den kommando der ønskes afviklet på systemet, som parameter. Popen-metoden laver en proces ved at kalde pipe i read-mode, så retursvaret fra kommandoen fås i en string. Metoden kan ses i Listing 7.10.

```

187     string exec(const char* cmd) {
188         //Make a linux shell with read
189         FILE* pipe = popen(cmd, "r");
190         if (!pipe) return "ERROR"; //Check if i got pipe
191         char buffer[128];
192         string result = "";
193         while (!feof(pipe)) { //While output read it.
194             if (fgets(buffer, 128, pipe) != NULL)
195                 result += buffer;
196         }
197         pclose(pipe); //Close the shell
198         return result;
199     }

```

Listing 7.10: Implementering af exec

7.3.3 GetDate

Nu da tiden kan sættes til den tid det ønskede på platformen, er det meget nemmere at læse den ud fra systemet. Fordi vi bare kan bruge et almindeligt API kald som er bygget ind i c++. I Listing 7.11 vises den kode, der henter tiden ud.

```

113     time_t mytime = time(0);
114     struct tm* tm_ptr = localtime(&mytime);
115     struct Date date;
116     if (tm_ptr)
117     {
118         date.Min = tm_ptr->tm_min;
119         date.Hour = tm_ptr->tm_hour;
120         date.Day = tm_ptr->tm_mday;
121         date.Month = tm_ptr->tm_mon+1;
122         date.Year = tm_ptr->tm_year+1900;
123         return date;
124     }
125     return date;

```

Listing 7.11: Implementering af getDate.

I projektet er det valgt at repræsentere tid med en struct, der indeholder år, måneder, dage, timer og minutter. Dog skal structen først udfyldes med tiden fra systemet, hvilket gøres ved at oprette et objekt af typen `time_t` med navnet "tm". Objektet fyldes så med systemets tid med funktionen `time(0)`. Efter dette fylder metoden `tids-structen` ud og returnerer denne.

7.3.4 Get- og set-metoder

Ud over disse metoder indeholder indstillinger en masse get- og set-metoder, så de andre klasser kan hente information om drivhuset. Eksempler kan ses i Listing 7.12 og 7.13.

Her tages en reference til

```
70 void GetHardware( bool &Varmelegeme , bool &bloeserne )
71 {
72     Varmelegeme = this->Varmelegeme ;
73     bloeserne = this->bloeserne ;
74 }
```

Listing 7.12: Implementering af `getHardware`.

```
78 void SetHardware( const bool Varmelegeme , const bool bloeserne )
79 {
80     this->Varmelegeme = Varmelegeme ;
81     this->bloeserne = bloeserne ;
82 }
```

Listing 7.13: Implementering af `setHardware`.

7.4 Datalog

Dataloggen gør internt brug af template-klassen `DoublyLinkedList`, afsnit 7.2 på side 139, som er en datastruktur. Denne datastruktur avendes til at gemme sensordata (se afsnit 7.1.1, side 138). Både `Regulator` og `Monitor` bruger den funktionalitet i klassen tilbyder, så der kan hentes og gemmes det nyeste data, som er blevet indsatt. `DoublyLinkedList` er en template-klasse, dvs. den kan rumme mange forskellige datatyper. Der kan dog kun vælges en type, som den så vil indeholde resten af dens livscyklus.

7.4.1 Lagering af data i datalogen

```
17 private :
18     DoublyLinkedList<SensorData> list ;
```

Listing 7.14: Template af `DoublyLinkedList`.

For at hente data ud gøres der brug af metoden `GetNewestData`, som bruger den førnævnte variabel "list" af typen `DoublyLinkedList` og bruger dens metode `PeekHead`. Efter den har hentet informationerne ud, returnerer `GetNewestData` det nyeste sensordata-struct som beskrevet i metodens signatur.

7.4.2 GetNewestData

```

31 SensorData GetNewestData()
32 {
33     SensorData dataToReturn;
34     list.PeekHead(dataToReturn);
35     return dataToReturn;
36 }
```

Listing 7.15: Implementering af GetNewestData.

Ønskes det at indsætte SensorData, gøres dette ved at bruge "list"igen, men hvor metoden headInsert bruges.

```

24 void InsertSensorData(SensorData SensorData)
25 {
26     list.headInsert(SensorData);
27 }
```

Listing 7.16: Implementering af InsertSensorData.

7.5 SystemLog

Bare for iteration, så bruges SystemLog klassen til at gemme systemhændelser fra monitor, regulator, uart, samt rapport klassen, hvis den var blevet implementeret. Klassen bygger videre på en ISU øvelse angående inter-thread communication, og kan håndterer beskeder fra forskellige tråde uden der sker segmentation faults. Centralt i implementation er to klasser Message og MsgQueue. De har selve ansvaret for at håndterer beskeder mellem klassen, der sender beskeden og SystemLog, som gemmer beskederne.

```

26 void send(unsigned long id, Message* msg = NULL) {
27
28     pthread_mutex_lock(&mtx);
29
30     Item newItem;
31
32     newItem.id_ = id;
33     newItem.msg_ = msg;
34     while(itemQueue_.size() >= size_)
35         pthread_cond_wait(&condi, &mtx);
36
37     itemQueue_.push(newItem);
38     pthread_cond_broadcast(&condi);
39     pthread_mutex_unlock(&mtx);
40 }
```

Listing 7.17: MsgQueues send metode

MsgQueue indeholder to interøstante metoder **send** og **receive**. Send metoden indsætter message objekter sammen med dens id ind i en stl vector. Den anvender en mutex, som sørger for at kun en tråd kan indsætte/udtage et message objekt af gangen. Den bruger desuden en conditional til at danne en kø, hvis MsgQueue'en er fyldt op.

```

43 Message* receive( unsigned long& id ){
44
45     pthread_mutex_lock(&mtx);
46
47     while(itemQueue_.size() == 0)
48         pthread_cond_wait(&condi, &mtx);
49
50     Item oldItem;
51     oldItem = itemQueue_.front();
52     id = oldItem.id_;
53
54     itemQueue_.pop();
55     pthread_cond_broadcast(&condi);
56     pthread_mutex_unlock(&mtx);
57     return oldItem.msg_;
58 }
```

Listing 7.18: MsgQueues receive metode

Receive virker på samme måde, dog pop den items på stakken. Den bruger samme mutex som send, da tråd procceser helst ikke må til gå vectoren på samme tid.

```

4 class Message
5 {
6     public:
7     virtual ~Message(){}
8 };
9 //-----Message1-----
10 #endif
```

Listing 7.19: Message klassen

Alle beskeder sendt til SystemLog, skal nedarve fra Message klassen, som er tom udover en virtuel destructor, således den nedarve klasse selv kan styre sig nedlæggelse. I AutoGreen systemet er oprettet en struct kladet SysMsg som nedarver fra Message og udvider den med en member msg af typen string. Dette giver klasserne, der skal sende beskeder til SystemLog, mulighed for at sende en string sammen med.

```

18 void checkMsg(){
19     while(true){
20         unsigned long id;
21         SysMsg* logentry = static_cast<SysMsg*>(msgs_->receive(id));
22         addMessage(logentry->msg_);
23         delete logentry;
24         usleep(10000000);
25     }
26 }
```

Listing 7.20: SystemLog's checkMsg metode

SystemLog's checkMsg metode virker som en event handler for alle beskeder, der bliver tilsendt systemLog's MsgQueue og videre sender den til en handler metode til at indsætte beskeden i en doublylinkedlist. SystemLog indeholder desuden en get metode, der henter head elementet af listen for visning i gui'en. checkMsg skal desuden lave en staticcast til SysMsg, da systemlog giver mulighed for flere typer beskeder(structs nedarvet fra message). Men siden der på nuværende tidspunkt kun er en type message, kan der med sikkerhed casts til den, ellers skal castningen ske på baggrund af det med sendte id. Siden beskeder bliver dynamisk opret dvs allokeret på heapen stedet for stacken. Det betyder også den skal deallokeres på heapen, ellers sker der et memoryleak.

```

87 int compareSet(SensorData data, Plant plant, int offset, int i)
88 {
89     if (plant.hum == 0 || data.grund[i] == -99)
90         return 1;
91
92     if (data.grund[i] < plant.hum + offset){
93
94         SysMsg* monmsg = new SysMsg;
95         monmsg->msg_ = "Plante " + i + " Mangler vand";
96         syslog_->send(1, monmsg);
97
98         return 2;
99     }
100    return 3;
101 }
102 //-----MonitorSet1-----
103
104 GUIStruct gui;
105 Plant virtuel_[6];
106 Date date_;
107 bool daily_;
108 bool warning_;
109 Indstillinger* ind_;
110 MsgQueue* syslog_;
111 DataLog* datalog_;
112 UART* uart_;
113 SensorData sensordata_;
114 };
115
116 #endif

```

Listing 7.21: Eksempel på indsættelse i SystemLog

For at indsætte en besked i SystemLog, skal der først oprettes en ny SysMsg via **new** c++ kommandoen. Derefter kan der indsættes en string i SysMsg'en med de relavante data. SysMsg lægges ind SystemLog's MsgQueue med send sammen med et id.

7.6 UART

UART'en som bruges til at transmittere data imellem DevKit8000 og PSoC Masteren, har til formål at oversætte sætninger, der har en given betydning (UART Protokol, side 39), op i chars. Disse chars sendes via DevKit8000's add-on Board. [12] UART'en er lavet ud fra et open-source bibliotek [12], har giver en række funktionaliteter til at skrive UARTEn, uden at skulle lave egne kernemoduler til devkittet.

7.6.1 UART opsætning

Da open-source biblioteket til UART'en er skrevet til brug på en seriell(RS232) udgang, har det betydet at en opsætning på DevKit8000 har været nødvendig, og uddover dette, en lille ændring i source-koden til biblioteket. I biblioteket er der ændret i rs232.c filen under "comports". Her er standarden for pc'er at comportene hedder henholdsvis /dev/ttyS0, /dev/ttysS1 osv. Dette er ændret til at passe til de comporter der befinner sig på DevKit8000.

```
1 char comports[38][16]={ "/dev/ttyO1", "/dev/ttyO0", "/dev/ttyO2", "/dev/ttyS3", "/dev/
  ttyS4", "/dev/ttyS5", "/dev/ttyS6", "/dev/ttyS7", "/dev/ttyS8", "/dev/ttyS9", "/dev/
  ttyS10", "/dev/ttyS11",
```

Listing 7.22: Porte connect metoden vil forsøge at lave forbindelse på

Den port der sendes med UART til, hedder /dev/ttyO1, og som det ses på Listing 7.22, er denne sat til dette. Denne ændring resulterer i, at data bliver sendt ud på J1(RS-232). Men da det ønskes at bruge pins til dette istedet for det fulde RS-232 kabel, bruges en mapping til J9. Denne mapping fortages i et script som indeholder hvad der kan ses i Listing 7.23.

```
1 echo 0x2 > /sys/class/cplddrv/cpld/ext_serial_if_route_reg
```

Listing 7.23: kommando til at mappe forbindelse fra J1 til J9 på addonboard

Dette danner mapningen over til J9, og det er nu muligt at forbinde UART'ens TX til pin 4, RX til pin 6 og en ground til pin 7.

7.6.2 UART Connect

Metoden Connect bruges til at forbinde til en aktiv port på DevKit8000. Ved brug af en while-løkke, løbes programkoden igennem samtlige porte på systemet, for at tjekke for om de er tilsluttet et kabel.

```
1 while (RS232_OpenComport(cport_nr, bdrate, mode))
2 {
3     cout << "cannot connect to port number " << cport_nr << endl;
4     cport_nr++;
5 }
6 cout << " connection made on port: " << cport_nr << endl;
7
8 }
```

Listing 7.24: connect metodens funktionalitet

Set i Listing 7.24, køres while-løkkken indtil der er oprettet en forbindelse på en port. Når denne port er fundet, tælles cport_nr ikke længere op, og denne port kan bruges til at sende data over. Det vil ikke forekomme at systemet vil tjekke på mere end den første port. Det er skrevet i koden, at scriptet skal tjekke på den port, som vi ønsker at anvende, som den første.

7.6.3 UART senddata

Metoden senddata er en hjælpe-metode, som har til formål at tage et kommando-parameter ind- og ud fra den kommando, og kan ved brug af if-statements bestemme hvilke chars der skal sendes over UART.

```
1 if (command_ == "temp")
2     strcpy(command[1], "T");
3 else if (command_ == "light")
4     strcpy(command[1], "L");
5 else if (command_ == "airhum")
6     strcpy(command[1], "A");
7 ....
8
9     RS232_cputs(cport_nr, command[1]);
```

Listing 7.25: logikken for at sende data ud på uartens tx

Listing 7.25, viser et udsnit af den samlede kode, hvor der tjekkes på command_ parameteren. Hvis if-sætningen er sand, kopieres en reduceret string - fx T til command-arrayet. Derefter sendes command-arrayet ud på UART'en, ved brug af RS232_cputs funktionen.

7.6.4 UART receivedata

Metoden receivedata er ligesom senddata en hjælpefunktion. Dog hvor senddata bruges til at sende data over UART'en, bruges receivedata til at læse på UART'en. Modtages der data der overholder UART-protokollen, bliver der returneret en værdi, afhængigt af, om der modtages data for temperatur, luftfugtighed, jordfugtighed eller lysintensitet

```

1  RS232_PollComport(cport_nr, buf, 4);
2
3  if (buf[0] == 'T')
4  {
5
6      return (int)buf[1];
7  }
8  else if (buf[0] == 'X' && buf[1] == 'T')
9  {
10     return -99;
11 }
12 ...

```

Listing 7.26: kodeafsnit af indsamling af data på rx

Listing 7.26, viser hvor i koden der bliver hentet 2 byte data ind i buf-parameteren, ved hjælp af RS232_PollComport funktionen. Der køres herefter et tjek på buf (som er en buffer til at indeholde modtagne data), om UART-protokollen overholdes. Fx hvis der modtages et T, betyder dette, at der er blevet returneret en temperatur, og der returneres værdien af buf[1] konverteret fra en char til int. Hvis indholdet af buf[0] er et X, er det en standard i UART-protokollen, at der er sket en fejl. Herefter tjekkes der på buf[1], for at se hvilken sensor der er sket en fejl på. Hvis det som i Listing 7.26 er et 'T', betyder det, at der er sket en fejl i aflæsning af temperatursensoren. Dette returnerer et -1. Ved fejl returneres -99, som er en overordnet fejlkode brugt i UART og regulator.

7.6.5 UART getSensor

Metoden getSensor er ikke implementeret i systemet, men idéen bag funktionen er, at sende besked til PSoC Masteren og derefter returnere hvor mange sensorer der er tilsluttet systemet.

7.6.6 UART activateSensor

Metoden activateSensor er brugt til at aktivere de 3 forskellige aktuatorer i systemet. Den kan tænde og slukke for aktuatorene og sender samtidig en besked til Systemloggen om hvilken hændelse der er sidst er foretaget. Der kan ses et eksempel på activateSensor i Listing 7.27, som viser et af de mange if-statements. Hjælpefunktionen senddata bruges til at transmitere kommandoen til PSoC masteren, og der sendes en besked til systemloggen om, at varmelegemet er blevet tændt.

```

1  if( command_ == "heaton")
2  {
3      senddata(command_);
4      systemlog->addMessage("Heat Turned on");
5  }
6  ....

```

Listing 7.27: activateSensor()

Fejlhåndtigering på activateSensor forgår ved, at der efter sendt kommando ventes på et svar fra PSoC-masteren, om aktuatoren er blevet tændt eller ej.

```

1 int response = receivedata();
2 if(response == 999)
3 {
4     //response OK
5 }
6 else{
7     int count;
8     for(count = 0; count < 2; count++)
9     {
10         senddata(command_);
11         response = recievedata();
12         if(response == 999)
13             break;
14     }
15 }
16 }
```

Listing 7.28: ReceiveData()

Hvis den returnede værdi i Listing 7.28 er 999, betyder det at operationen var vellykket. Hvis ikke operationen er vellykket, sendes kommandoen til PSoC Masteren 2 gange mere, og hvis et godkendt svar kommer, stoppes metoden. Hvis ikke aktuatoren får et godkendt svar inden for de 3 forsøg, springer metoden over den givne aktuator indtil videre.

7.6.7 UART getSensorData

Metoden getSensorData anvendes til at returnere et SensorData-objekt, med indhold der er hentet fra PSoC Masteren.

```
1 SensorData newdata;
```

Listing 7.29: sensorData()

Der oprettes først et nyt objekt af SensorData-typen til at gemme data i. Herefter hentes data for de forskellige parametre.

```

1 senddata("temp");
2 int data = recievedata();
3 cout << data << endl;
4 if (data != -99)
5 {
6
7     newdata.temp = (double(data)/2)-20;
8 }
9 else
10 {
11     newdata.temp = -99;
12 }
```

Listing 7.30: indsamling af temperatur og fejlsikring

I Listing 7.30, vises hvordan der hentes data ned fra temperatursensoren. Der sendes besked til PSoc Masteren om, at få temperaturen ved brug af hjælpefunktionen senddata(); Herefter indhentes svar ved brug af hjælpefunktionen recievedata(). De modtagne data bliver valideret og så længe der ikke er modtaget en fejlkode, udregnes temperaturen og indsættes i sensordata-structen. Hvis der sker en fejl, indsættes -99 i sensordata-structen, så monitoren ved at den skal skrive "ugyldig data" ud i

hovedmenuen. Efter temperaturen er indsatt, fortsætter metoden med at hente data ind for de seks jordfugtsensorer. Dog er der foretaget en lille ændring i forhold til indsamling, hvilket vises i Listing 7.31.

```

1 senddata("ground1");
2     data = recievedata();
3     if (data != -99)
4     {
5         newdata.grund[0] = data;
6     }
7     else {
8         senddata("ground1");
9         data = recievedata();
10        if (data != -99)
11        {
12            newdata.grund[0] = data;
13        }
14        else {
15            newdata.grund[0] = -99;
16        }
17    }

```

Listing 7.31: indsamling af jordfugtighedsdata og fejsikring

I indsamling af temperatur spørges der kun efter data én gang, hvilket der af sikkerhedsmæssige grunde er ændret, så der kan spørges efter jordsensordata en ekstra gang, hvis ikke der kommer noget svar fra PSoC Masteren. Returneres der en fejlmeddelse fra recievedata(), sendes en efterspørgsel om data én gang til. Hvis det andet forsøg igen fejler, indsættes -99 i sensordata-structen som fejlbesked. Når Der er indsamlet data fra alle jordfugtsensorer, inklusive fejlmeddelelser, returnes sensordata-structen.

7.6.8 Fejsikring i UART

UART'ens måde at indhente data på foregår ved, at UART'en anmoder om data. Herefter afvikles recievedata(), som startes med et sleep wait i et sekund, hvilket giver PSoC Masteren tid til at sende data over. Herefter validerer recievedata() de sendte data og hvis den har modtaget ugyldige eller ingen data, returnes -99. Med valideringen sikres systemet mod fejl i forbindelse med udtagede stik og bruger ikke tid på at vente på data der aldrig kommer. Systemet er altså fejsikret, da det fylder structet op med fejlmeddelelser, indtil stikket igen er sat korrekt i.

7.7 Monitor

Monitor klassen bliver oprettet i main og tager i konstruktoren referencer til uart, datalog, indstillinge og en MsgQueue. Desuden har konstruktoren til opgave at starte uart forbindelsen, og sætte startværdier for gui elementer i hovedmenuen.

```

21 Monitor(UART &uart , DataLog &data , Indstillinger &ind ,
22           MsgQueue &syslog)
23 {
24     uart_ = &uart;
25     datalog_ = &data;
26     ind_ = &ind;
27     syslog_ = &syslog;
28     uart_->connect();
29
30     gui.temp = 25;
31     gui.avgtemp = 25;
32
33     for( int i = 0; i < 6; i++)
34     {
35         gui.status[ i ] = 1;
36         gui.realHum[ i ] = 0;
37         gui.virtualHum[ i ] = 0;
38     }
39 }
```

Listing 7.32: Monitors konstruktor

Gui er en private member af Monitor og er af typen GuiStruct. GuiStruct bruges til at gemme de fleste grafiske data om hovedmenu, derefter kan hentes fra mainwindow klassen via monitors getGuiData metode.

```

46 void compareData()
47 {
48     if(ind_->getMonitorering()){
49
50     // update local values to newest settings.
51     date_ = ind_->getDate();
52     ind_->GetNotifications(daily_ , warning_);
53     ind_->GetPlants(virtuel_);
54
55
56     // See if sensor data is aviliable
57     //int sensor = uart_->getSensor();
58     sensordata_ = uart_->getSensorData();
59     datalog_->InsertSensorData(sensordata_);
60
61
62     // update live Gui temperature
63     gui.temp = sensordata_.temp;
64     gui.avgtemp = virtuel_[ 0 ].temp;
65
66     for( int i = 0; i < 6; i++)
67     {
68         gui.status[ i ] = compareSet(sensordata_ , virtuel_[ i ] , 0 , i );
69         gui.realHum[ i ] = sensordata_.grund[ i ];
70         gui.virtualHum[ i ] = virtuel_[ i ].hum;
71     }
72
73     usleep(100000);
```

```

74 }
75
76 usleep(100000);
77
78 }
```

Listing 7.33: Monitors compareData

Den vigtigste metode i monitor er compareData, som køres fra en tråd i main sammen med Regulator. Den har til opgave at: Hente sensordata fra PSoC masteren, indsætte disse data i datalogen, sammenligne fugtighedsdata mellem de virtuelle planter og de indhentede sensordata fra uart'en, opdater GuiStruct'en udfra disse sammen ligningerne.

```

87 int compareSet(SensorData data, Plant plant, int offset, int i)
88 {
89     if (plant.hum == 0 || data.grund[i] == -99)
90         return 1;
91
92     if (data.grund[i] < plant.hum + offset) {
93
94         SysMsg* monmsg = new SysMsg;
95         monmsg->msg_ = "Plante " + i + " Mangler vand";
96         syslog_->send(1, monmsg);
97
98         return 2;
99     }
100    return 3;
101 }
```

Listing 7.34: Monitors compareSet

CompareSet er en privat hjælpefunktion til sammenligne fugtighedsdata og opdater Gui'en for status knapperne i hovedmenuen. Den tager som parametre: Sensordata fra uart, virtuel plantedata fra indstillinger, offset for fugtighedstolerance. Den sender en besked til systemloggen hvis en plante falder uden for dens tolerance.

7.8 Regulator

Regulatoren står for håndtering af ændringer i drivhuset. Hvis temperaturen er for høj eller for lav, har regulatoren mulighed for at ændre klimaet ved brug af aktuatorerne. Regulatoren kører i samme tråd som monitoren. Regulatortråden bliver ikke stoppet, men et tjek i regulatoren sørger for at regulatoren ikke ændrer noget ved drivhuset.

```

1 if (!settings->getRegulering())
2 {
3
4     usleep(100000);
5 }
6 else
7 {
8     ....
```

Listing 7.35: Regulatorens tjek af settings

Set i Listing 7.35, tilgår regulatoren settings og tjekker om regulering er slået til i indstillinger. Hvis ikke regulatoren er slået til, ventes der i et sekund, hvorefter run-metoden venter på at blive aktiveret igen.

7.8.1 Regulator constructor

Da Regulatoren skal have forbindelse til UART'en, Indstillinger, Systemloggen og Dataloggen, opstættes dette i constructoren. Der er ingen grund til at kunne ændre pointerne til forbindelserne, hvorfor placeringen blev valgt til constructoren.

```

1 Regulator(UART * uart_, Indstillinger * settings_, SystemLog *systemlog_, DataLog *
2   datalog_)
3 {
4   /* set pointers
5   uart = uart_;
6   settings = settings_;
7   systemlog = systemlog_;
8   datalog = datalog_;
9   ...

```

Listing 7.36: Regulator()

Når Regulator-klassen oprettes, skal der opgives pointere til UART, Indstillinger, SystemLog og Datalog som set i Listing 7.36. Systemet ville ikke fungere uden disse pointers, da regulator-klassen skal bruge dataloggen til at læse fra drivhusets klima. UART'en er vigtig, da reguleringen skal kunne snakke til PSoC Masteren om hvad aktuatorerne, der skal køre i drivhuset.

På opstart af systemet sættes der også en række booleans som set i Listing 7.37.

```

1 heatON = false;
2 ventsON = false;
3 windowON = false;
4 water1ON = false;
5 water2ON = false;
6 water3ON = false;
7 water4ON = false;
8 water5ON = false;
9 water6ON = false;

```

Listing 7.37: Oprettelse af booleans ved system opstart

Disse booleans sættes så regulatoren internt ved at blæser, varmelegeme, vindue og vanding på alle planterne er slukket. Herefter sender regulatoren besked over UART'en om, at slukke for alle aktuatorer i system ved brug af activateSensor fra UART-klassen.

```

1 uart->activateSensor("ventoff");
2 uart->activateSensor("heatoff");
3 uart->activateSensor("water1off");
4 uart->activateSensor("water2off");
5 ....

```

Listing 7.38: Anvendelse af UART-klassens activateSensor

Da der sendes 9 kommandoer til UART'en ved opstart, har systemet derved en boot-time på omkring 9-10 sekunder, da den skal have slukket alle aktuatorer før hovedmenuen vise, og systemet kan aktiveres.

7.8.2 loadData

Metoden loadData er en hjælpe metode, hvis eneste formål er at gøre run-metoden mere overskuelig. Metoden har til formål at hente data ind fra indstillinger og derefter gemme informationerne om de virtuelle planter.

```

1 plant1 = settings->getPlant(1);
2 plant2 = settings->getPlant(2);
3 ...

```

Listing 7.39: Implementering af loadData.

Data bliver hentet ind ved brug af settings-pointeren, som kalder getPlant-metoden fra Indstillingsklassen, hvilket returnerer et plante-objekt. Objektet kopieres ind i planten metoden bliver kaldt for. Der laves et getPlant kald for alle 6 planter.

7.8.3 Run metoden

Run-metoden i Regulatoren kaldes fra main. Formålet med metoden er at kalde alle andre metoder direkte fra main. Fra design er det meningen at run skal køre i en evighedsløkke, men den blev kun implementeret til at køre igennem en enkelt gang. I stedet har QT en tråd der indeholder en evighedsløkke, som kalder run, når der skal laves reguleringer.

```

1 //load data into plants from Settings (indstillinger)
2 loadData();
3 SensorData loadeddata = datalog->GetNewestData();
4 ControlData(loadeddata);
5 //linux
6 usleep(100000);

```

Listing 7.40: Implementering af run metoden.

Hvis reguleringen er aktiveret i hovedmenuen, køres koden i Listing 7.40. Den hentes først det nyeste data ind fra indstillings-klassen ved brug af loadData-metoden. herefter oprettes der et SensorData-objekt, hvor de nyeste data fra datalogen indhentes ved brug af pointeren til dataloggen. SensorData-objektet gives til ControlData-metoden, som sørger for at sammenligne de interne plante-objekters klimapræferencer med de faktiske data fra drivhuset. Når ControlData er afviklet, lægger tråden sig til at sove i et minut, før run starter fra begyndelsen af while-løkken igen.

7.8.4 ControlData metoden

ControlData-metoden er står for sammenligningen af det ønskede klima og det faktiske klima i drivhuset. Hvis en parameter, f.eks. temperatur, er meget afhængig fra den ønskede temperatur, kan controlData regulere temperaturen op og ned ved brug af aktuatorerne i systemet.

ControlData fungerer ved først at hente den faktiske temperatur ud af SensorData structen. Den finder herefter en gennemsnitlig temperatur for de virtuelle planter i systemet.

```

1 double temp_drivhus = drivhus_data.temp;
2 double avg_temp_drivhus = (plant1.temp + plant2.temp + plant3.temp + plant4.temp
+ plant5.temp + plant6.temp) / 6;

```

Listing 7.41: Implementering af ControlData metoden.

Metoden henter herefter information ind fra indstillingsklassen om brugen af varmelegeme og blæserne, hvilket sker ved brug af settings-pointeren, hvor GetHardware metoden kaldes med referencer til use_heater og use_vents bools, som set i 7.42.

```

1 bool use_heater = false;
2 bool use_vents = false;
3 settings->GetHardware(use_heater, use_vents);

```

Listing 7.42: Implementering af getHardware metoden.

ControlData kører, efter indhentning af data, en lang række tjek på temperaturen ved brug af if-statements. I tilfælde af temperaturforskelle på aktuel temperatur og ønsket temperatur, bruges aktuatorer til at regulere temperaturen.

```

1 else if (temp_drivhus >= avg_temp_drivhus + 0.5)
2 {
3
4     cout << "temp lidt varm\n";
5 //OPEN WinDOW
6
7     if (!windowON)
8     {
9         uart->activateSensor("windowon");
10        usleep(100);
11        windowON = true;
12    }
13    if (ventsON)
14    {
15        uart->activateSensor("ventoff");
16        usleep(100);
17        ventsON = false;
18    }
19 }
```

Listing 7.43: Eksempel på if-statements i regulator.

Hvis temperaturen er en halv grad for høj, bruges vinduet til at regulere temperaturen. Der tjekkes på den interne boolean `windowON` om vinduet allerede er åbent. Hvis vinduet allerede er åbnet fortages ingenting, men hvis det er lukket åbnes det. Der tjekkes også på, om blæserne er tændt. Hvis de er tændte, slukkes der for dem, da det ikke er ønsket at bruge dem når temperaturen er så tæt på det ønskede. Temperaturreguleringen kan ses i 7.44.

Efter temperaturreguleringen, bliver der kørt et tjek på de 6 planter i drivhuset igen og igen.

```

1 if (drivhus_data.grund[0] < plant1.water)
2 {
3
4
5     if (drivhus_data.grund[0] != -99){
6         if (!water1ON){
7             uart->activateSensor("water1on");
8             water1ON = true;
9         }
10    }
11    else {
12        if (water1ON){
13            uart->activateSensor("water1off");
14            water1ON = false;
15        }
16    }
17
18
19 } else
20 {
21     if (water1ON){
22         uart->activateSensor("water1off");
23         water1ON = false;
24     }
}
```

Listing 7.44: Temperaturregulering.

Funktionaliteten for plantetjekket fungerer ved sammenligning af jordfugtigheden fra SensorData-objektet og den givne plantes ideelle jordfugtighed kaldet **water**. Er den ønskede jordfugtighed højere end den faktiske jordfugtighed, antages det at en vanding vil være aktuel. Hvis en vanding er aktuel tjekkes der først på at data fra plantestruktten ikke er -99, da denne værdi betyder fejl. Er værdien ikke -99, tjekkes der for om vanding allerede er aktiv på den interne boolean **water1ON**. Er **water1ON** false, sendes der besked på UART'en om at tænde for vandingen og den interne boolean sættes til true. Else-sætningen der er gældende, hvis der mangler vand og fejlkoden -99 er indskrevet i plantestruktten, sørger for at slukke for vandingsbooleanen og vise et N/A i hovedmenuen. På denne måde undgås det at vandingsbools altid er tændt, hvis der ikke er modtaget data, eller jordfugtsensoren ikke er tilsluttet systemet.

7.9 GUI - QT

Til at lave den grafiske brugerflade på DevKit8000 er udviklingsmiljøet QT brugt. Ved at bruge QT, gøres det meget letttere at skabe brugerflader og dermed forbindelser mellem brugerfladen og det bagvedliggende system.

7.9.1 QT .ui filer

Til at designe menuer i softwaren, giver QTcreator [15] mulighed for at designe menuer, blot ved brug af drag and drop teknikken. Til design af vores menuer, tilpasser vi et canvas til størrelsen 480x256 pixels, som passer til til at dække hele skærmen på DevKit8000. De mest anvendte funktionaliteter, der er brugt til at designe den grafiske brugerflade, er knapper. Knapperne kan trækkes ind på det ønskede sted på canvas og teksten, samt farven, på knappen kan ændres. Ændring af farve på knapper kan f.eks. ses i hovedmenuen ved tryk på Monitorer og reguler. Knappen kan navngives, hvilket så kan bruges til at referere og lave signaler med, til intern kommunikation i systemet.

7.9.2 QT event driven programming

QT standarden og den metode vi også bruger til at kode og designe brugerfladen i, er "event driven programming" [16]. Når der trykkes på en knap, foretages der en række funktionaliteter, som kan have indflydelse på brugerfladen, men i de fleste tilfælde vil ændre noget i grundsystemet bag brugerfladen. En af disse events, som påvirker både brugerfladen og grundsystemet er 'monitorer'. Et tryk på monitorknappen leder til et kald i indstillinger, hvor monitor booleanen bliver sat til true. Dette sætter systemet igang med at monitorere drivhuset og samtidig skifte knappens farve til grøn, svarende til at den er aktiv.

```

269 void MainWindow :: on_btn_monitor_clicked()
270 {
271
272     if( !refs_.indstillinger->getRegulering() )
273     {
274         if( refs_.indstillinger->getMonitorering() ) //Toggle
275         {
276             //red
277             ui->btn_monitor->setStyleSheet("background-color: rgb(255, 0, 0)");
278             refs_.indstillinger->SetMonitorering(false);
279
280         } else
281         {
282             //green
283             ui->btn_monitor->setStyleSheet("background-color: rgb(0, 255, 0)");

```

```

284     refs_.indstillinger->SetMonitorering( true );
285 }
286 }
287 else
288 {
289     if( refs_.indstillinger->getMonitorering() ) //Toggle
290     {
291         //red
292         ui->btn_monitor->setStyleSheet("background-color: rgb(255, 0, 0)");
293         ui->btn_reguler->setStyleSheet("background-color: rgb(255, 0, 0)");
294         refs_.indstillinger->SetMonitorering( false );
295         refs_.indstillinger->SetRegulering( false );
296     }
297 }
298 }
299 }
```

Listing 7.45: Mainwindow monitor knap.

Koden i Listing 7.45 viser brugen af monitorer knappen på hovedmenuen. Når der trykkes på knappen, køres den viste koden.

Der laves tjek med if-statements på regulering fra indstillings-klassen og hvis reguleringen er tændt, tjekkes der på monitorering fra indstillings-klassen. Hvis monitorering er tændt, sættes både regulering og monitorering til false i indstillings-klassen og farven på knapperne skiftes til rød, svarende til at de er slukket. Hvis regulering er slukket, tjekkes der også på monitorering. Monitoreringen toggles ved at tjekke, om den er true eller false, hvis den er true, slukkes den ved brug af setMonitorering-metoden, fra indstillings-klassen, og farven på knappen skiftes til rød. Omvendt hvis den er slukket, sættes monitorering til true, og farven på knappen sættes til grøn.

I Listing 7.46 ses funktionaliteten i QT, der skifter farve på en knap. I koden tilgås UI'en som tilgår knappen btn_monitor og kalder metoden setStyleSheet på den, for at skifte farve.

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

Listing 7.46: Farveskifte i QT.

7.9.3 Menuhåndtering og menublokering

Det ønskes i brugerfladen, at brugeren kan skifte imellem flere menuer, i stedet for at skulle gemme knapper væk i samme menu. Dette gøres ved at lave flere .ui filer, som også har tilsvarende source- og headerfiler og eksekvere dem som en menu, ovenpå den forhenværende menu.

```

259 void MainWindow::on_btn_systemlog_clicked()
260 {
261     dialoge_systemlog systemlog( refs_ );
262     systemlog.setModal( true );
263     systemlog.exec();
264 }
```

Listing 7.47: Klik på systemlog

I Listing 7.47, ses hvordan der oprettes et object af dialoge_systemlog ved et klik på systemlogknappen i hovedmenuen. Det sikres herefter, at de forhenværende menuer ikke kan bruges, før systemlogmenuen igen er lukket ned. Dette gøres ved brug af setModal. Metoden setModal blokerer alt input til andre menuer, end den aktive menu. Herefter eksekveres menuen vha. exec metoden.

7.9.4 Trådhåndtering i QT

Der gøres brug af POSIX pthreads [17], til at oprette tråde på vores linux platform. Ud over at oprette tråde, anvendes "mutual exclusion" eller bedre kendt som "mutexes", hvilket vil sige at der kun kan køre en tråd i et angivet stykke kode. For at oprette en tråd bruges pthread_create som ses i Listing 7.48.

```

78
79     // Start af tråede
80     pthread_t monAndRegTrd, sysTrd;
81     pthread_create(&monAndRegTrd, NULL, &MonitorAndRegTrd, &referenceStruct);
82     pthread_create(&sysTrd, NULL, &SystemLogTrd, &systemlog);
```

Listing 7.48: Pthread_create.

7.9.5 Timers til opdatering

Opdatering af hovedmenuen gøres via en QTimer fra QT [15].

```

18     QTimer *timer = new QTimer(this);
19     connect(timer, SIGNAL(timeout()), this, SLOT(updateBtn()));
20     timer->start(6900);
```

Listing 7.49: QT timer.

Der oprettes en timer i Listing 7.49, med formålet at køre en funktion hver 6,9. sekunder. Timeren forbindes herefter via connect til updateBtn-metoden. Der sendes så et signal til updateBtn med information om, at den skal køres, når timeren når enden. Tiden på timeren kan indstilles ved brug af timer->start metoden.

GUI struct

Gui-structen er lavet til at forenkle dataoverførelsen mellem Monitor-klassen og QT-klassen, mainwindow. Den indeholder en temperaturværdi for drivhuset, en middelværdi over alle virtuelle planters ideelle temperatur, tre arrays som bestemmer farve og fugtighedsværdier for statusknapperne.

```

4 struct GUIStruct
5 {
6     double temp;
7     double avgtemp;
8     int status[6];
9     int realHum[6];
10    int virtualHum[6];
11};
```

Listing 7.50: GUIStruct

7.9.6 System sammensætning

Der gøres brug af en struct kaldet referencestruct, som indeholder pointere til klasserne Indstillinger, DataLog, Monitor, Systemlog og Regulator. Den sendes med til QT menuer og til tråde, hvis de har brug for at kende en eller flere af klasserne, for at kunne interagere med hinanden. Et eksempel er menuen "Hardware Indstillinger", hvor der gøres brug af indstillinger, for at sætte og hente hvilken hardware, der må bruges til at regulere med. Ud over dette sendes den også til de tråde, som findes i systemet via en void pointer, som derefter static castes til refference structen, som derefter kan bruges til at tilgå de enkelte klasser (se afsnit 7.9.4 - Trådhåndtering i QT).

7.10 Build tools

Under dette projekt er der udviklet scripts til at gøre arbejdet med linux platformen lettere. Scriptene blev brugt til at oprette forbindelse til DevKit8000, kopiere filer fra host til platformen, bygge hele projektet og automatisk flytte det over til DevKit8000, uden at skulle skrive alle kommandoer på ny hver gang. Scriptet **conn2tgt** bruges til at oprette en ssh (Secure Shell) forbindelse til Devkit8000, så det kan undgås at indtaste password ved login. Dette kan lade sig gøre ved at bruge **sshpas**, der sender et password videre til programmer, som kræver et superuser password. I dette tilfælde bruges kommandoen **ssh** sammen med parameteren **root@10.9.8.2**, som betyder at der laves et login med brugernavnet root på IP'en 10.9.8.2. Scriptet kan ses i Listing 7.51.

```
3 #! bin/bash
4 sshpass -p 'stud' ssh root@10.9.8.2
```

Listing 7.51: Implementering af conn2tgt scriptet.

Scriptet **cp2tgt** set i Listing 7.51 bruges kopiere en fil mellem en computer og DevKit8000. Dette sker ved at bruge kommandoen **scp** (Secure copy), til at overføre filerne. I dette script er der anvendt et parameter som ses i linje 3, hvor der står "**\$1**". Det betyder at man kan kalde scriptet med en fil, som vil blive overført. Det kan ses i praksis i Listing 7.53.

```
2 #! bin/bash
3 sshpass -p 'stud' scp $1 root@10.9.8.2:
```

Listing 7.52: Implementering af cp2tgt scriptet.

Dette script gør brug af den makefile som QT har genereret og bruger den ved at anvende linux kommandoen **make**. Efter kommandoen er kørt, kopieres den eksekverbare fil, autogreenbuild3, over til linux platformen via scriptet **cp2tgt**. Til sidst køres **conn2tgt**, som laver en ssh forbindelse til DevKit8000, så det er nemt køre programmet og teste det.

```
2 #!/bin/bash
3
4 #make -clean
5 #qmake -project
6 #qmake
7 make
8
9 cp2tgt autogreenbuild3
10 conn2tgt
```

Listing 7.53: Implementering af buildandrun scriptet.

Et andet script der bruges hedder **setupTarget**, som får host computeren til at forbinde til linux platformen via et usb kabel og kan ses i Listing 7.54.

```
3 #! bin/bash
4 echo setting up Devkit8000
5 sshpass -p 'stud' ifdown usb0
6 sshpass -p 'stud' ifup usb0
7 #
```

Listing 7.54: Implementering af buildandrun scriptet.

Det første der sker er at der bruges kommandoen **echo** til at udskrive en text på skærmen, så brugeren ved at den er ved lave så der kan laves en forbindelse til Devkittet via et usb kabel. Kommandoerne **ifdown usb0** og **ifup usb0** får usb interface på hosten til at lukke en og derefter åbne igen bagefter, det skal gøres for at den ser forbindelsen rigtigt.

8 Accepttest

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.
15. Maj	4	LBS	Udført accepttest, samt lavet mindre rettelser med tidsmæssige tests.

8.1 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.	Monitoreringsknappen skifter farve fra rød til grøn	Godkendt
1.2	Bruger trykker på "Monitorering".	Visuel test: Bruger observerer at "Monitorering" skifter farve fra grøn til rød.	Monitoreringsknappen skifter farve fra grøn til rød	Godkendt
1.3	Bruger trykker på "Regulering".	Visuel test: Bruger observerer at "Monitorering" og "Regulering" skifter farve fra rød til grøn.	Der sker ingenting.	Ikke godkendt. Ikke implementeret.
1.4	Bruger trykker på "Regulering".	Visuel test: Bruger observerer at "Regulering" skifter farve fra grøn til rød.	Der sker ingenting.	Ikke godkendt. Ikke implementeret.
1.5	Bruger trykker på "Monitorering".	Visuel test: Bruger observerer at "Monitorering" skifter farve fra rød til grøn.	Monitoreringsknappen skifter farve fra rød til grøn	Godkendt.
1.6	Bruger trykker på "Regulering".	Visuel test: Bruger observerer at "Regulering" skifter farve fra rød til grøn.	"Regulering" skifter farve fra rød til grøn.	Godkendt.
1.7	Bruger trykker på "Monitorering".	Visuel test: Bruger observerer at "Monitorering" og "Regulering" skifter farve fra grøn til rød.	"Monitorering" og "Regulering" skifter farve fra grøn til rød.	Godkendt.

Tabel 122: 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.	Der observeres en temperatur.	Godkendt.
3.2	Bruger ser på brugergrænsefladen.	Visuel test: Der observeres tilstedeværelse af oplysninger om luftfugtighed.	Der observeres ikke nogen luftfugtighed.	Ikke godkendt. Ikke implementeret.
3.3	Bruger ser på brugergrænsefladen.	Visuel test: Der observeres tilstedeværelse af oplysninger om lysintensitet.	Der observeres ikke nogen lysintensitet	Ikke godkendt. Ikke implementeret.
3.4	Bruger ser på brugergrænsefladen.	Visuel test: Der observeres tilstedeværelse af oplysninger om jordfugtighed for jordfugtmåler 1-6.	Der observeres 6 forskellige jordfugtigheder.	Godkendt. Pga. fejlkomunikation på UART, er der af og til udfald.

Tabel 123: 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.	Der vises en virtel drivhus menu.	Godkendt. Menuen er ikke færdigimplementeret.
4.2	Bruger trykker "Tilføj plante".	Visuel test: Bruger observerer en liste over planterne i Plantedatabasen på brugerfladen.	Findes ikke.	
4.3	Bruger vælger den øverste plante på listen.	Visuel test: Systemet viser undermenuen "Planteredigerings-menu".	Plantedatabase ikke implementeret.	Ikke godkendt. Ikke implementeret.
4.4	Bruger indtaster parametre for planten, temperatur: 25 grader, fugtighed: 10, og trykker "OK".	Visuel test: Undermenuen "Virtuelt Drivhus Menu" vises. Den redigerede plante vises.	Plantedatabase ikke implementeret.	Ikke godkendt. Ikke implementeret.
4.5	Bruger trykker på den plante, der blev tilføjet under pkt. 4.4.	Visuel test: Systemet viser undermenuen "Planteredigerings-menu".	Plantedatabase ikke implementeret.	Ikke godkendt. Ikke implementeret.
4.6	Bruger trykker "Fjern" og trykker "OK".	Visuel test: "Virtuelt Drivhus Menu" vises. Den fjernede plante er ikke længere i det virtuelle drivhus.	Plantedatabase ikke implementeret.	Ikke godkendt. Ikke implementeret.
4.9	Bruger trykker "Tilbage".	Visuel test: Hovedmenuen vises efter der trykkes tilbage	Hovedmenuen vises efter der trykkes tilbage	Godkendt.

Tabel 124: 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."	Systemet viser en historikmenu.	Godkendt.
5.2	Bruger trykker "Uge".	Visuel test: Systemet viser en graf med historik for en uge.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.3	Bruger trykker "Måned".	Visuel test: Systemet viser en graf med historik for en måned.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.4	Bruger trykker "År".	Visuel test: Systemet viser en graf med historik for et år.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.5	Bruger trykker "Temperatur".	Visuel test: Historik for temperatur vises ikke på grafen.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.6	Bruger trykker "Luftfugtighed".	Visuel test: Historik for luftfugtighed vises ikke på grafen.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.7	Bruger trykker "Lys".	Visuel test: Historik for lys vises ikke på grafen.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.8	Bruger trykker "Temperatur".	Visuel test: Historik for temperatur vises på grafen.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.9	Bruger trykker "Luftfugtighed".	Visuel test: Historik for luftfugtighed vises på grafen.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.10	Bruger trykker "Lys".	Visuel test: Historik for lys vises på grafen.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.11	Bruger trykker "Jordfugtighed" for planete nr 1.	Visuel test: Jordfugtigheden for planete nr. 1 vises på grafen.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.

5.12	Pkt. 5.11 gentages for plante 2-6.	Visuel test: Jordfugtigheden for planterne vises på grafen.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.13	Bruger trykker "Jordfugtighed" for plante nr 1.	Visuel test: Jordfugtigheden for plante nr. 1 vises ikke på grafen.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.14	Pkt. 5.12 gentages for plante 2-6.	Visuel test: Jordfugtigheden for planterne vises ikke på grafen.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.
5.15	Bruger trykker på "Tilbage"	Visuel test: Hovedmenuen vises.	Hovedmenuen vises.	Godkendt.

Tabel 125: 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".	Plantedatabase er ikke implementeret.	Ikke godkendt. Ikke implementeret.
6.2	Bruger trykker på "Tilføj Data".	Visuel test: Systemet viser "Databaseredigeringsmenu".	Plantedatabase er ikke implementeret.	Ikke godkendt. Ikke implementeret.
6.3	Bruger vælger ønskede parametre for planten, temperatur: 22 grader, fugtighed: 8, og trykker "OK".	Visuel test: Systemet viser "Plantedatabasemenu". Planten er tilføjet.	Plantedatabase er ikke implementeret.	Ikke godkendt. Ikke implementeret.
6.4	Bruger trykker på den plante, der blev tilføjet under pkt. 6.3.	Visuel test: Systemet viser "Databaseredigeringsmenu".	Plantedatabase er ikke implementeret.	Ikke godkendt. Ikke implementeret.
6.5	Bruger trykker "Fjern".	Visuel test: Systemet viser en dialogboks.	Plantedatabase er ikke implementeret.	Ikke godkendt. Ikke implementeret.
6.6	Bruger trykker "OK".	Visuel test: Systemet viser "Plantedatabasemenu". Planten er fjernet.	Plantedatabase er ikke implementeret.	Ikke godkendt. Ikke implementeret.
6.7	Bruger trykker "Tilbage".	Visuel test: Hovedmenuen vises.	Plantedatabase er ikke implementeret.	Ikke godkendt. Ikke implementeret.

Tabel 126: 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”.	Systemet viser konfigurationsmenu.	Godkendt.
7.2	Bruger trykker ”E-mail Adresser”.	Visuel test: System viser ”E-mail Menu”.	Systemet viser e-mailmenu.	Godkendt.
7.3	Bruger indtaster tre E-mail adresser og trykker ”OK”.	Visuel test: System viser ”Konfigurationsmenu”.	Der kan ikke indtastes e-mailadresser.	Ikke godkendt. Manglende tastatur.
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.	Der vises ingen e-mailadresser.	Ikke godkendt. Refererer til pkt. 7.3.
7.5	Bruger trykker ”Tilbage”.	Visuel test: Systemet viser ”Konfigurationsmenu”.	Konfigurationsmenuen vises af systemet.	Godkendt.
7.6	Bruger trykker ”Notifikationer”	Visuel test: ”Notifikationsmenu” vises.	Systemet viser notifikationsmenu.	Godkendt.
7.7	Bruger trykker på ”Notifikations E-mail”.	Visuel test: ”Notifikations E-mail” skifter farve fra rød til grøn.	”Notifikations E-mail” knappen skifter farve fra rød til grøn.	Godkendt.
7.8	Bruger trykker på ”Notifikations E-mail”.	Visuel test: ”Notifikations E-mail” skifter farve fra grøn til rød.	”Notifikations E-mail” knappen skifter farve fra grøn til rød.	Godkendt.
7.9	Bruger trykker på ”Advarsels E-mail”.	Visuel test: ”Advarsels E-mail” skifter farve fra rød til grøn.	”Advarsels E-mail” skifter farve fra rød til grøn.	Godkendt.
7.10	Bruger trykker på ”Advarsels E-mail”.	Visuel test: ”Advarsels E-mail” skifter farve fra grøn til rød.	”Advarsels E-mail” skifter farve fra grøn til rød.	Godkendt.
7.11	Bruger trykker ”Tilbage”.	Visuel test: Systemet viser ”Konfigurationsmenu”.	Systemet viser konfigurationsmenuen.	Godkendt.
7.12	Bruger trykker ”Indstil dato/tid”.	Visuel test: ”Tid- og Datomenu” vises.	Menuen vises.	Godkendt.

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.	Det er ikke muligt at indtaste noget.	Ikke godkendt. Ikke implementeret.
7.14	Bruger trykker "Hardware Indstillinger".	Visuel test: Systemet viser "Hardware indstillingsmenu".	Menuen vises.	Godkendt.
7.15	Bruger trykker på "Blæsere"	Visuel test: "Blæsere" skifter farve fra rød til grøn.	"Blæsere" skifter farve fra rød til grøn.	Godkendt.
7.16	Bruger trykker på "Blæsere"	Visuel test: "Blæsere" skifter farve fra grøn til rød.	"Blæsere" skifter farve fra grøn til rød.	Godkendt.
7.17	Bruger trykker på "Varmelegeme".	Visuel test: "Varmelegeme" skifter farve fra rød til grøn.	"Varmelegeme" skifter farve fra rød til grøn.	Godkendt.
7.18	Bruger trykker på "Varmelegeme".	Visuel test: "Varmelegeme" skifter farve fra grøn til rød.	Visuel test: "Varmelegeme" skifter farve fra grøn til rød.	Godkendt.
7.19	Bruger trykker på "Tilbage".	Visuel test: "Konfigurationsmenu" vises.	Konfigurationsmenuen vises.	Godkendt.
7.20	Bruger trykker på "Tilbage".	Visuel test: "Hovedmenu" vises.	Hovedmenuen vises.	Godkendt.

Tabel 127: 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".	Systemet viser "systemlogmenu".	Godkendt. Der vises kun seneste event i loggen.
8.2	Bruger trykker "Tilbage".	Visuel test: Systemet viser Hovedmenuen.	Hovedmenuen vises	Godkendt.

Tabel 128: 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. Jordfugtighedssensor 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.	E-mail er ikke implementeret.	Ikke godkendt. Ikke implementeret.
9.2	Bruger tager jordfugtighedssensor 1 op af jorden.	Visuel test: Bruger modtager advarsels E-mail.	E-mail er ikke implementeret.	Ikke godkendt. Ikke implementeret.

Tabel 129: 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 køler temperatursensoren og venter mindst 30 sekunder.	Visuel test: Værdien i feltet "Temperatur" i Hovedmenuen falder.	Temperaturen falder efter ca. 26 sekunder.	Godkendt.
10.2	Bruger deaktiverer monitorering og tilgår systemloggen.	Visuel test: Systemloggen er blevet opdateret med nye data og korrekt tidsstempeling.	Systemloggen viser ikke ny data.	Ikke godkendt. Ikke implementeret.

Tabel 130: 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.	Spændingen på aktuatoren svarer til true.	Godkendt. Målt med LED.
11.2	Bruger køler temperatursensor og venter mindst 20 sekunder.	Visuel test: Varmelegemet aktiveres.	Varmelegemet startes efter ca. 5 sekunder.	Godkendt.
11.3	Det fysiske drivhus opvarmes vha. et varmelegeme.	Visuel test: Blæserne aktiveres og vinduet åbnes.	Vinduet åbnes og blæserne startes.	Godkendt.

Tabel 131: Accepttest for UC11: Regulering

8.2 Ikke-funktionelle krav

Krav	Test	Forventet Resultat	Resultat	Godkendt/ kommentar
1.	Start drivhuset med monitorering, noter hvornår værdier bliver tilføjet til systemloggen. Varighed 2 min.	Data loggen genererer 1 datapunkt hver 10. sekund.	Dataloggen viser kun et enkelt datapunkt.	Ikke godkendt. Ikke færdigimplementeret.
2.	Det fysiske drivhus placeres i et rum ved 25 +/- 1 grader celcius, og opvarmes vha. et varmelegeme 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.	Temperaturen er reguleret ned inden for temperaturtolerancen efter 4 min og 14 sekunder.	Godkendt.
3.	En potte med tør muld indsættes i det fysiske drivhus. En fugtighedssensor 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.	Der vises ikke måleværdier.	Delvis godkendt. Systemet kan vise jordfugt i 10 trin, men det er ikke muligt at gennemføre testen som beskrevet, da mange forskellige faktorer påvirker dem.
4.	Seks fugtighedsmålere tilsluttes systemet, hvorefter systemet startes.	Systemloggen indeholder måleværdier for 6 forskellige sensorer efter 1 min.	Der vises ikke måleværdier.	Delvist godkendt. Sensorerne giver korrekte værdier, men systemloggen er ikke færdigimplementeret.
5.	100 planter indsættes i plantedatabasen.	Databasen kontrolleres for alle 100 planters eksistens.	Plantedatabasen er ikke implementeret.	Ikke godkendt. Ikke implementeret.
6.	Intervallet for datalogging sættes ned for at simulere et års data.	Historik kan ses med et års data.	Historik er ikke implementeret.	Ikke godkendt. Ikke implementeret.

7.	Det fysiske drivhus placeres i et rum ved 25 +/- 1 grader celcius. Systemet sættes til at regulere temperaturen i det fysiske drivhus til 30 grader celcius. Et eksternt termometer med en usikkerhed på højst 0.1 grader celcius placeres ved siden af temperatursensoren.	Det eksterne termometer mæler 30 +/- 1 grad celcius inden for 30 min.	Systemet opvarmer drivhuset til 30 grader efter 2 minutter og 30 sekunder, kommer ikke over 31 grader og tænder igen for varmelegemet efter 7 minutter og 2 sekunder. Derfor betragtes temperaturen som stabil.	Godkendt. Koden tager ikke højde for offset på temperatursensor, derfor anvendes ekstern termometer ikke i testen.
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.	E-mail er ikke implementeret.	Ikke godkendt. Ikke implementeret.
9.	En potte med tør muld indsættes i det fysiske drivhus. En fugtighedssensor 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.	E-mail er ikke implementeret.	Ikke godkendt. Ikke implementeret.

Tabel 132: Ikke funktionelle krav

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] Honeywell International Inc.: *Datablad for Temperatur-/Luftfugtighedssensor*.
"Bilag 003 - Datablad for Temp og Luftfugtsensor". August 2013.
- [4] Honeywell International Inc.: *I²C Communication with the Honeywell HumidIcon*.
"Bilag 004 - I2C Communication with the Honeywell HumidIcon". June 2012.
- [5] Intersil: *Datablad for lyssensor*.
"Bilag 005 - Datablad for lyssensor". Februar 2008.
- [6] Motorola, Inc.: *Three-Terminal Positive Voltage Regulators*.
"Bilag 006 - Datablad for LM7805". Maj 1996.
- [7] SAIA-Burgess Electronics: *Motor Products*.
"Bilag 007 - Motor Products". 2000.
- [8] Maxim: *Digital Temperature Sensor and Thermal Watchdog with 2-Wire Interface*.
"Bilag 008 - Datablad for LM75". 2009.
- [9] PRJ3 F15 Gruppe 9: *MSE Øvelse 6*.
"Bilag 009 - MSE Øvelse 6". Maj 2015.
- [10] USB Implementers Forum, Inc.: *USB 2.0 Specification*.
http://www.usb.org/developers/docs/usb20_docs/#usb20spec. Marts 2015.
- [11] PRJ3 F15 Gruppe 9: *PSoC Master source code*.
https://github.com/PhKP/PSoC_Master. 2015.
- [12] Devkit8000 wiki, IHA: *DevKit8000 wiki*.
https://redmine.ih.a.dk/devs/projects/devkit8000/wiki/Addon_board. Dec 2013.
- [13] Beelen, Teunis van: *RS232 UART-protokol*.
<http://www.teuniz.net/RS-232/>. Jan 2015.
- [14] Codebase.eu: *Multithreading*.
<http://codebase.eu/tutorial posix-threads-c/>. Sep 2011
- [15] The QT company: *QT info*.
<http://doc.qt.io/qt-4.8/timers.html>. Feb 2015.
- [16] Wikipedia: *Event Driven Programming*.
http://en.wikipedia.org/wiki/Event-driven_programming. Maj 2015.
- [17] The Open Group. *Multi threading*.
<http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/pthread.h.html>. 2013.