

Projektdokumentation 3. semester

E3PRJ3
GRUPPE 8

IKT, ELEKTRO & STÆRKSTRØM
AARHUS SCHOOL OF ENGINEERING

Studienummer - Navn

- 20104209 - ANDREAS LAURSEN
- 201270024 - DANNIE LEHMANN
- 10253 - JENS RIX JØRGENSEN
- 201270725 - MATHIAS JESSEN
- 20062548 - MORTEN MØLLER CHRISTENSEN
- 201303580 - SIMON MOURIDSEN
- 201270943 - STINE SKAARUP HØGSBERG

VEJLEDER - LARS G. JOHANSEN

Deltageres underskrift

Andreas Laursen
20104209

Dannie Lehmann
201270024

Jens Rix Jørgensen
10253

Mathias Jessen
201270725

Morten Møller Christensen
20062548

Simon Mouridsen
201303580

Stine Skaarup Høgsberg
201270943

Indholdsfortegnelse

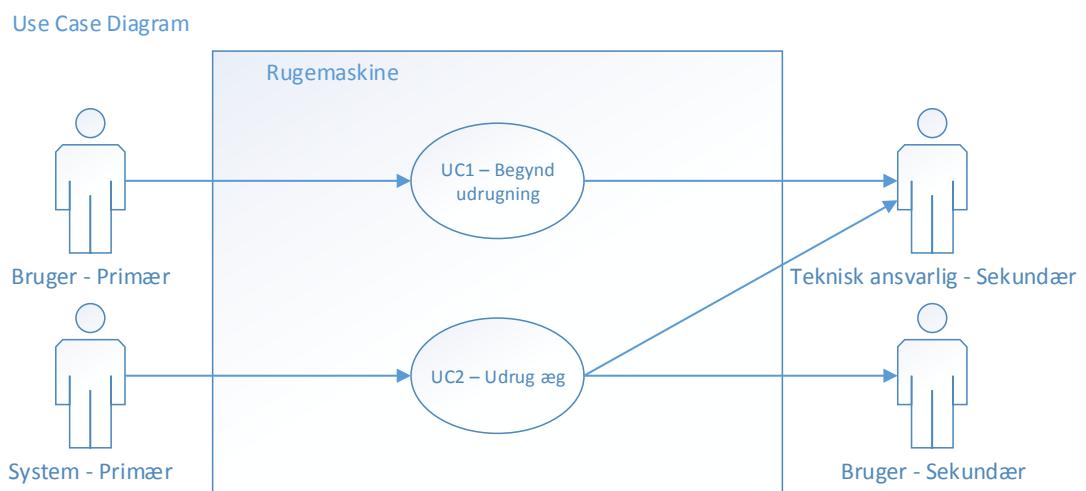
Kapitel 1 Kravspecifikation	1
1.1 Use Case diagram	1
1.2 Aktørbeskrivelse	2
1.2.1 Bruger	2
1.2.2 System	2
1.2.3 Tekniske ansvarlig	2
1.3 Use Cases	3
1.3.1 Begynd udrugning	3
1.3.2 Udrug æg	4
1.4 Funktionelle krav	5
1.5 Ikke-funktionelle krav	5
Kapitel 2 Systemarkitektur	6
2.0.1 Sequence Diagram Use Case 1	7
2.0.2 Sequence Diagram Use Case 2	8
2.0.3 State Machine Diagram	9
2.0.4 Logical bdd	9
2.0.5 Physical bdd	10
2.0.6 Internal Block Diagrams	11
2.0.7 Class Diagram System	12
Kapitel 3 Hardware	13
3.1 Varmelegeme	13
3.1.1 Design	13
3.1.2 Implementation	14
3.1.3 Test	15
3.2 Befugter	16
3.2.1 Design	16
3.2.2 Implementation	17
3.2.3 Test	17
3.3 Ægvender	18
3.3.1 Design	18
3.3.2 Implementation	18
3.3.3 Test	19
3.4 Låge kontakt	19
3.4.1 Design	19
3.4.2 Implementation	20
3.4.3 Test	21
3.5 Aktuator interface	22
3.5.1 Design	22

3.5.2	Implementation	23
3.5.3	Test	24
3.6	Printplade	25
3.6.1	Design	25
3.6.2	Implementation	26
3.6.3	Test	27
Kapitel 4 Software		28
4.1	Sensor	28
4.1.1	Design	28
4.1.2	Implementering	28
4.2	Varmeregulering	31
4.2.1	Design	31
4.2.2	Implementation	35
4.2.3	Test	38
4.3	Befugtning	40
4.3.1	Design	40
4.3.2	Implementation	43
4.3.3	Test	46
4.4	Stepmotor	47
4.4.1	Design	47
4.4.2	Implementering	48
4.4.3	Test	49
4.5	Sekvenstimer	50
4.5.1	Design	50
4.5.2	Implementering	50
4.5.3	Test	51
4.6	PSoC SPI	52
4.6.1	Design	52
4.6.2	Implementering	55
4.7	PSoC Main	57
4.7.1	Design	57
4.7.2	Implementering	59
4.7.3	Test	61
4.8	DevKit8000 Driver	62
4.8.1	Design	62
4.8.2	Implementering	63
4.8.3	Test	66
4.9	DevKit8000 GUI applikation	67
4.9.1	Design	67
4.9.2	Implementering	68
4.9.3	Test	71
Kapitel 5 Accepttest		72
5.1	Accepttest for Use Case 1	72
5.1.1	Hovedscenarie	72

5.2 Accepttest for Use Case 2	72
5.2.1 Hovedscenarie	72
5.2.2 Undtagelser	74
Litteratur	76

Kravspecifikation 1

1.1 Use Case diagram



Figur 1.1. Use Case diagram

1.2 Aktørbeskrivelse

1.2.1 Bruger

Aktørnavn:	Bruger
Alternativt navn:	User
Type:	Primær og sekundær
Beskrivelse:	<ul style="list-style-type: none"> 1. Bruger ønsker at udruge æg

Tabel 1.1. Aktør - Bruger

1.2.2 System

Aktørnavn:	System
Alternativt navn:	Rugemaskine
Type:	Primær
Beskrivelse:	<ul style="list-style-type: none"> 1. Systemet styrer udrugningsprocesser. 2. Systemet har til opgave at overvåge temperatur, luftfugtighed samt tidsprogression.

Tabel 1.2. Aktør - System

1.2.3 Tekniske ansvarlig

Aktørnavn:	Tekniske ansvarlig
Alternativt navn:	Servicetekniker
Type:	Ekstern
Beskrivelse:	<ul style="list-style-type: none"> 1. Tekniske ansvarlig er en uddannet tekniker, der har en faglig forståelse for opbygning af rugemaskinen.

Tabel 1.3. Aktør - Tekniske ansvarlig

1.3 Use Cases

1.3.1 Begynd udrugning

Navn	Begynd udrugning
Usecase ID	#1
Scope	
Primær aktør	Bruger
Interessenter	Tekniske ansvarlig.
Forudsætning	Use Case 2 er ikke aktiv
Resultat	Udrugning af indlagte æg er påbegyndt.
Hovedforløb	<ol style="list-style-type: none"> 1. Bruger åbner låge. 2. Systemet registrerer åbning af låge. 3. Systemet låser for interfacet (skriver "låge åben"). 4. Bruger placerer æg i maskine. 5. Bruger lukker låge. 6. Systemet registrerer lukning af låge. 7. Systemet låser op for interfacet (fjerner "låge åben"). 8. Bruger vælger type af æg til udrugning (sekvens). 9. Systemet spørger bruger om bekræftelse. 10. Bruger bekræfter valg (vælger "OK"). 11. Systemet registrerer valg. 12. Systemet aktiverer sekvensen Udrug Æg (Use Case 2).
Undtagelser	<ol style="list-style-type: none"> 2a. System registrerer ikke åbning af låge. <ul style="list-style-type: none"> • Bruger kontakter teknisk ansvarlig. 6a. System registrerer ikke lukning af låge. <ul style="list-style-type: none"> • Bruger kontakter teknisk ansvarlig. 10a. Bruger bekræfter ikke valg (vælger "Annuler"). <ul style="list-style-type: none"> • Step 8 gentages.

Tabel 1.4. Use Case - Begynd udrugning

1.3.2 Udrug æg

Navn	Udrug æg
Use Case ID	#2
Primær aktør	System
Interessenter	Sekundær aktør: Bruger
Forudsætning	Use Case 1
Resultat	Æggene er udrugt og fjernet fra maskinen
Hovedforløb	<ol style="list-style-type: none"> 1. Systemet indlæser valgte sekvens. 2. Systemet regulerer temperatur og luftfugtighed. 3. Systemet kontrollerer om det er tid til æggevending. hvis (tid = æggevending): Vend æg. ellers: Vend ikke æg. Step 2-3 gentages indtil udrugningstiden er afsluttet. 4. System informerer brugeren om at udrugningssekvensen er færdig. 5. Bruger åbner låge. 6. Systemet registrerer åbning af låge. 7. Systemet afbryder regulering af temperatur og luftfugtighed. 8. Bruger fjerner emne(r). 9. Bruger lukker låge. 10. Systemet registrerer lukning af låge. 11. Systemet genoptager regulering af temperatur og luftfugtighed. Step 5-11 gentages indtil brugeren indikerer overfor systemet at alle emner er fjernet. 12. Systemet stopper med regulering af temperatur og luftfugtighed.
Undtagelser	<ul style="list-style-type: none"> * Bruger afbryder udrugningen. <ul style="list-style-type: none"> • Systemet afbryder Use Case 2 * (2-3) Bruger åbner maskinen. <ul style="list-style-type: none"> • Systemet afbryder trin 2-3. <ul style="list-style-type: none"> – Bruger lukker maskinen. – Systemet fortsætter med trin 2-3. 2a. Systemet kan ikke regulere temperatur eller luftfugtighed. <ul style="list-style-type: none"> • Systemet registrerer fejl. • Systemet informerer bruger om fejl. 6a. System registrerer ikke åbning af maskine. <ul style="list-style-type: none"> • Bruger kontakter tekniske ansvarlig. 10a. System registrerer ikke lukning af maskine. <ul style="list-style-type: none"> • Bruger kontakter tekniske ansvarlig.

Tabel 1.5. Use Case - Udrug æg

1.4 Funktionelle krav

1. Systemet skal kunne regulere temperaturen efter justering af parameter inden for 10 min.
2. Systemet skal kunne regulere luftfugtigheden efter justering af parameter inden for 10 min.

1.5 Ikke-funktionelle krav

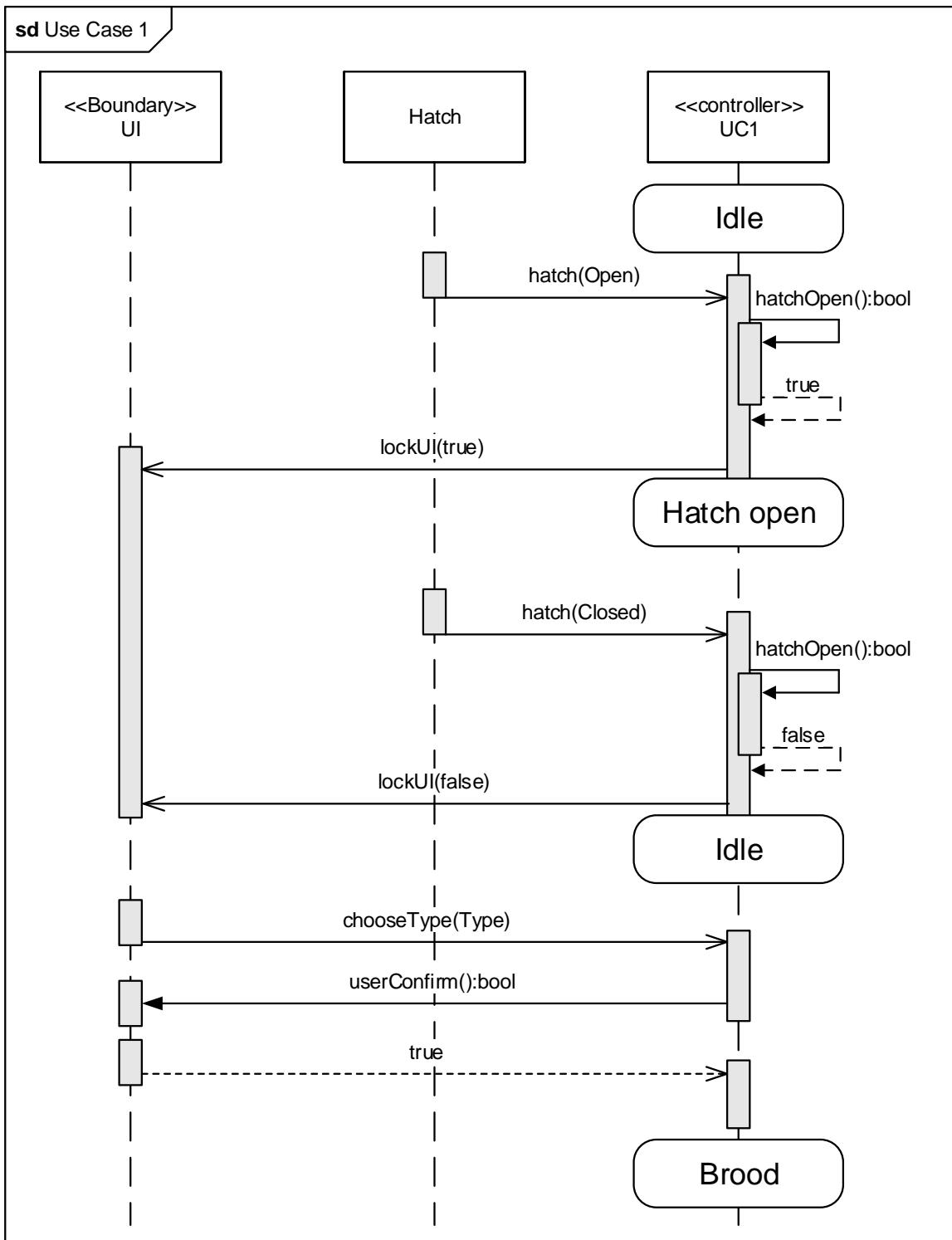
1. Hvis systemet ikke kan overholde de angivne grænser som er beskrevet i de funktionelle krav, skal brugeren informeres vha. alarmer.
2. DevKit8000 8000 anvendes som styrende enhed samt grafisk brugergrænseflade (GUI).
3. PSoC3 og/eller 4 anvendes som grænseflade til føler/sensorer samt aktuatorer.
4. Tilladte afvigelser:
 - Temperatur: $\pm 1^{\circ}\text{C}$.
 - Luftfugtighed : ± 10 procentpoint.
5. Krav til opstillingsmiljø:
 - Temperatur : 15-30°C.
 - Luftfugtighed : <45% procentpoint.
6. krav til GUI:
 - Der skal være mulighed for at navigere via GUI.
 - GUI skal kunne bruges til at starte og stoppe udrugningen.
 - Der skal altid på UI under udrugning fremgå: Temperatur, luftfugtighed, tidsprogression.
 - GUI skal gøre bruger opmærksom på tilstandsændringer.
7. Rugemaskinens dimensioner er: ± 2 cm
 - Brede x cm
 - Højde x cm
 - Dybe x cm
8. Udrugningsprocedurer skal følge anbefalingerne angivet på www.honsehus.dk/opdraet/rugetips/
9. Rotation: Rotation på $180^{\circ} \pm 45^{\circ}$

Systemarkitektur 2

For at skabe overblik over samtlige dele af systemet, udarbejdes der SysML systemarkitektur. SysML diagrammerne danner overblik over hele systemet og hjælper med at danne en nemmere overgang til design og implementeringsfasen. Systemarkitekturen er bygget op således, at jo længere frem i arkitekturen man kommer, desto flere detaljer får man for systemet.

2.0.1 Sequence Diagram Use Case 1

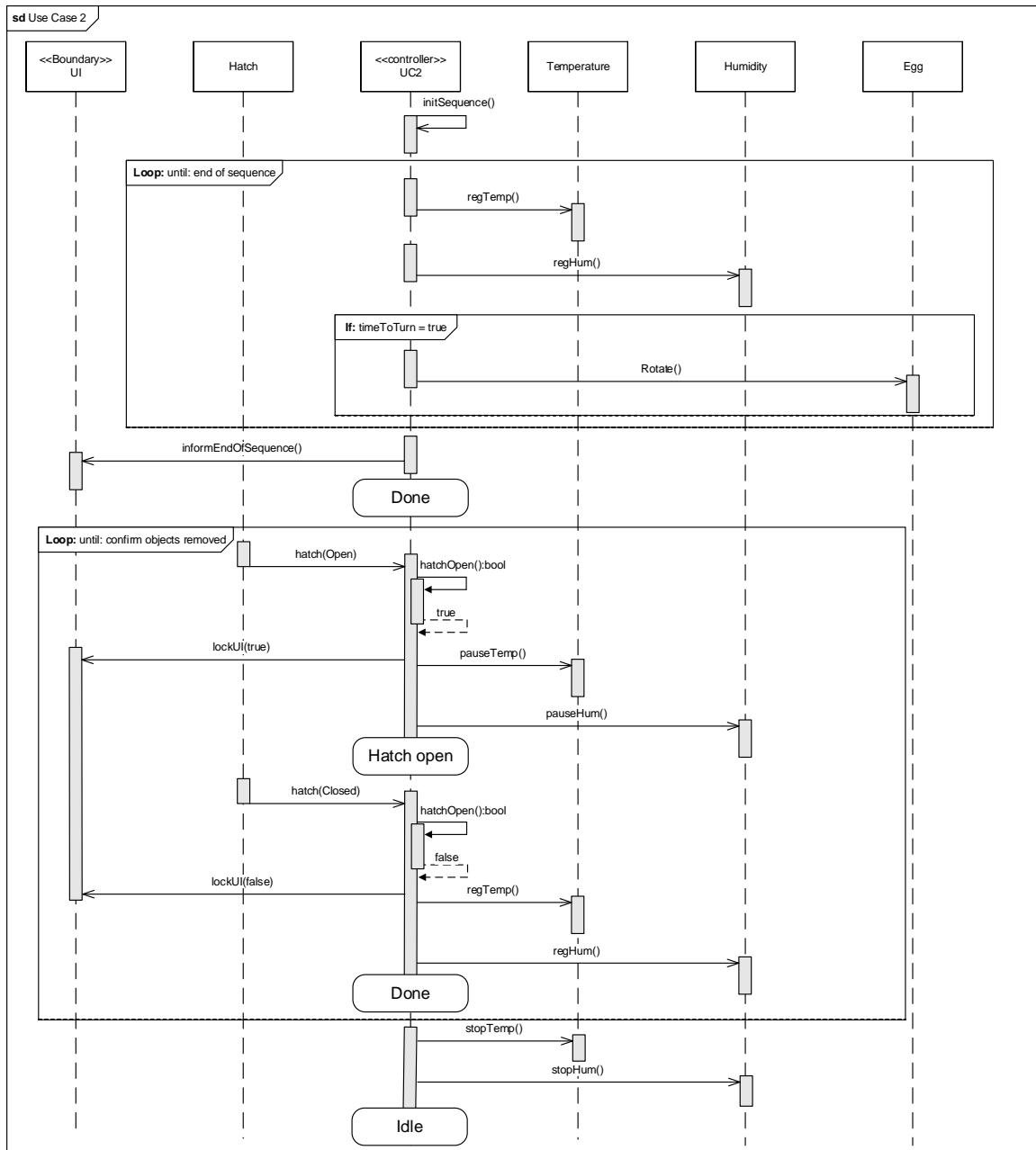
Det første diagram, der udarbejdes er Sekvensdiagrammet for Use Case 1. Sekvensdiagrammet er med til at give overblik over de forskellige funktioner, der skal implementeres i design og implementeringsfasen. På sekvensdiagrammet tilføjes også States, som benyttes i State Machine Diagrammet, som kan ses senere i arkitekturen.



Figur 2.1. Sequence diagram - Use Case 1

2.0.2 Sequence Diagram Use Case 2

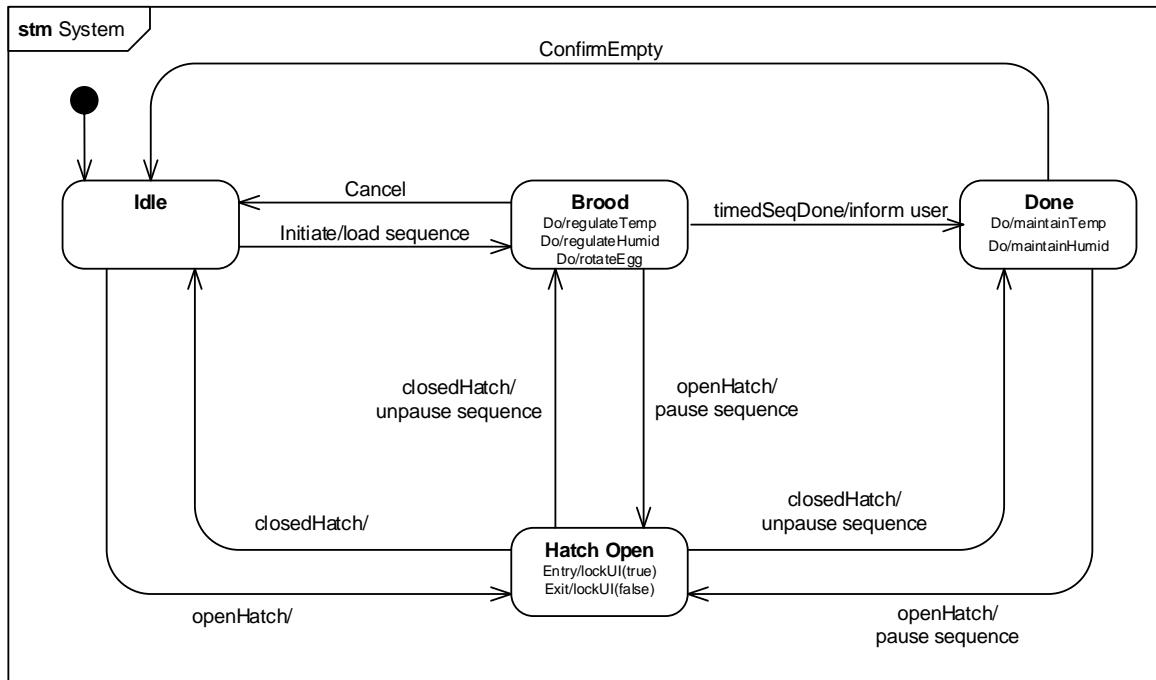
I sekvensdiagrammet nedenfor beskrives den overordnede interaktion mellem systemets dele i konteksten af Use Case 2. Systemet initierer udrugningssekvensen, hvor der kontinuert reguleres på temperatur og luftfugtighed, og hvor der tjekkes om æggene skal vendes. Når denne sekvens er afsluttet, informerer systemet brugeren om dette. Dernæst reguleres temperatur og luftluftighed indtil alle udrugede emner er fjernet fra systemet. Herefter stoppes regulering af temperatur samt luftfugtighed, og systemet går i sin idle-state.



Figur 2.2. Sequence diagram - Use Case 2

2.0.3 State Machine Diagram

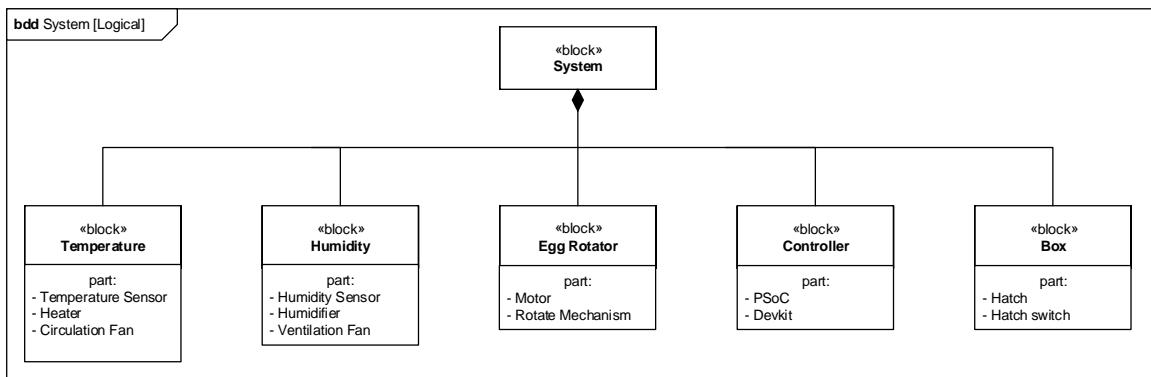
State Maskine diagrammet bruges til at vise de forskellige stadier systemet kan befinde sig i, samt overgang mellem dem. Systemet har fire stadier: "Idle", "Brood", "Done" samt "Hatch Open". Det blev valgt at der er mulighed for at gå til "Idle" fra de tre andre stadier. For at have mulighed for at gå til "Idle" fra "Brood" blev der tilføjet en Cancel som giver brugeren mulighed for at afbryde udrugningen.



Figur 2.3. State Machine diagram

2.0.4 Logical bdd

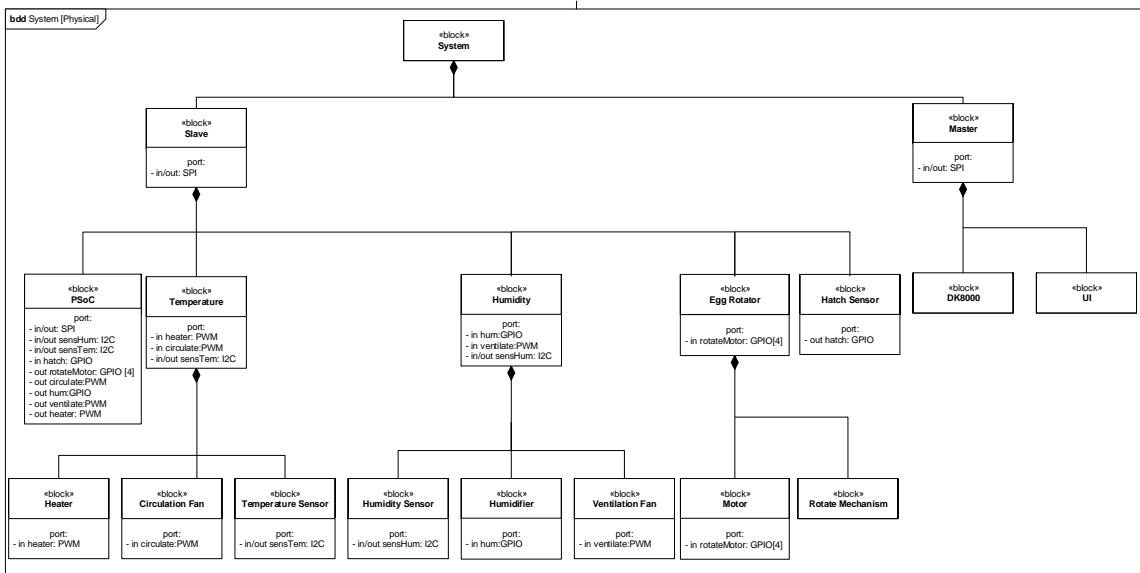
Nedenstående ses et logisk bdd for systemet, dette diagram er med til at give et overblik over de enkelte logiske blokke for systemet. Dette gør implementeringen lettere da systemet er brutt ned i mindre dele.



Figur 2.4. Logical bdd

2.0.5 Physical bdd

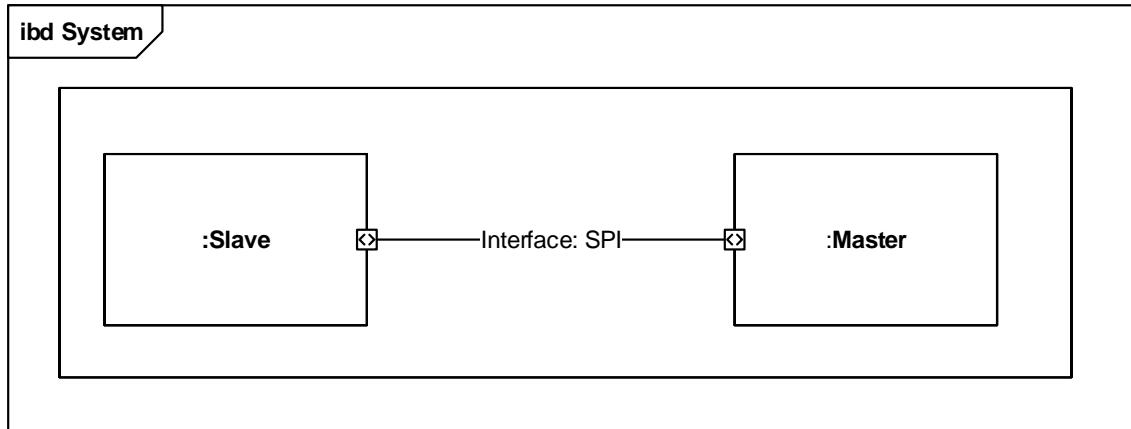
Nedenstående ses et fysisk bdd for systemet, dette diagram er med til at give et overblik over de enkelte fysiske enheder og deres relationer, ligeledes giver dette diagram en hurtig beskrivelse af signalerne som opstår mellem de enkelte fysiske enheder.



Figur 2.5. Physical bdd

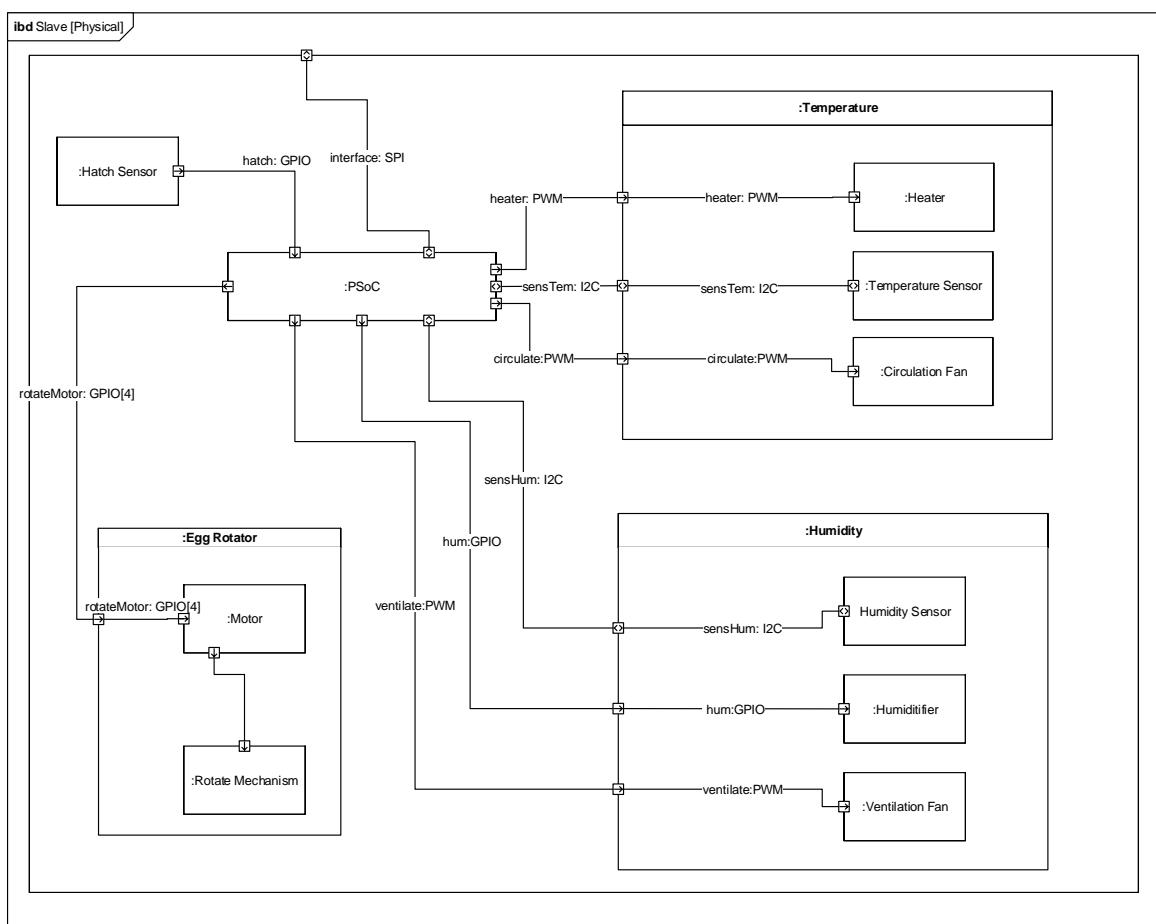
2.0.6 Internal Block Diagrams

Figuren herunder viser IBD for hele systemet.



Figur 2.6. IBD for System

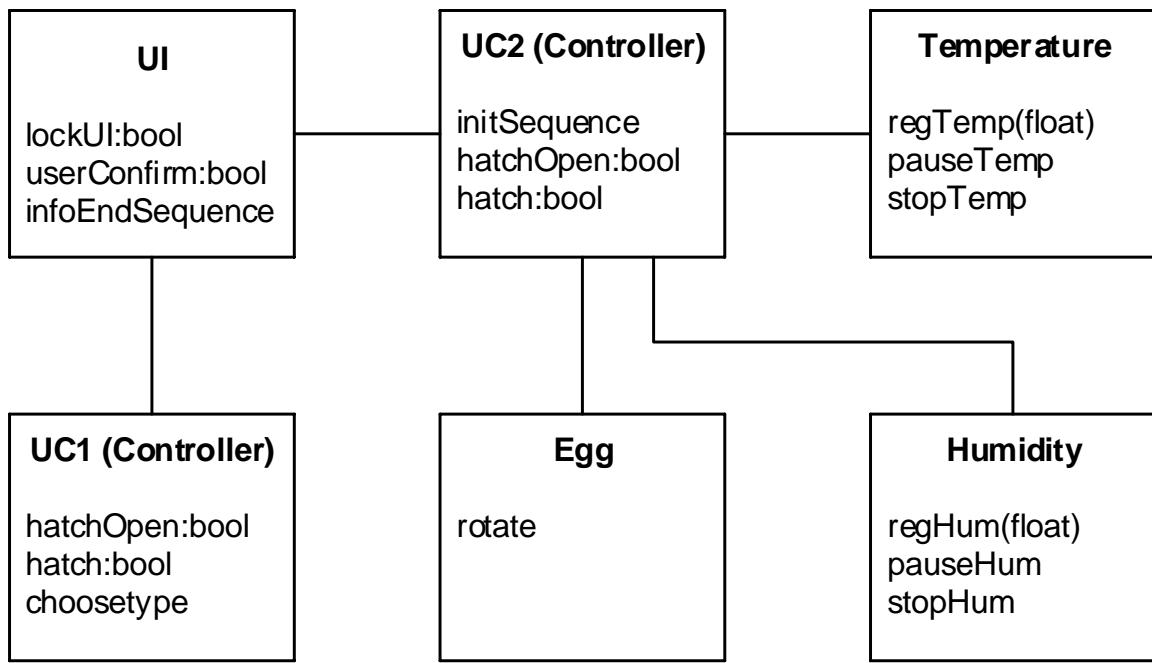
Figuren herunder viser IBD for Slave delen af systemet, som er den mest komplekse del af det overordnede system.



Figur 2.7. IBD for Slave

2.0.7 Class Diagram System

Med udgangspunkt i Usecases blev der sammen med sekvensdiagrammerne udarbejdet et Klasse diagram. Dette giver overblik over og samler alle funktioner brugt i de to sekvensdiagrammer. Flere af funktionerne har ikke fået angivet return- og/eller parametre type. Dette er grundet at der mangler at blive taget stilling til disse.



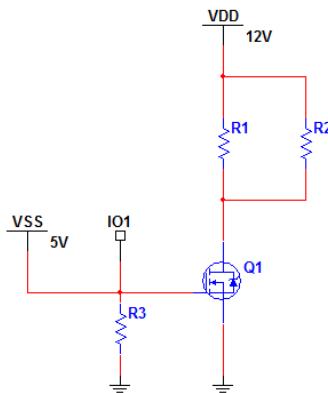
Figur 2.8. Class diagram

Hardware 3

3.1 Varmelegeme

3.1.1 Design

Til varmelegemet blev der hurtigt foreslægt, at den skulle laves med en køleplade med 2 effektmodstande, som styres med en mosfet transistor (figur 3.1). For at opvarme vores maskines miljø blev der uden erfaring antaget, at den nødvendige effekt der skulle afsættes ville være i størrelse på 60-80 Watt (30-40W i hver modstand).



Figur 3.1. Varmelegeme kredsløb med 2 modstande

Da der skal skabes en fælles temperatur for hele maskinen vil der blive sat en blæser til at sætte rotation i luften og derved fordele temperaturen.

Et alternativt design ville være at lave varme genereringen med en varmelampe, men i modsætningen til køleplade og effektmodstande, så er varmelampen ikke umiddelbart til rådighed. Størrelsen og figuren af en varmelampe er heller ikke optimal for placering i en forholdsvis lille kasse.

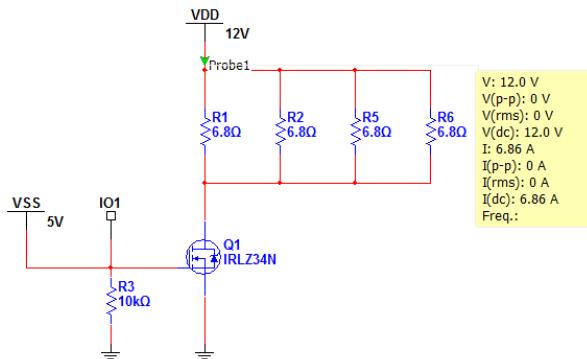
3.1.2 Implementation

Ved implementeringen blev der ved komponentundersøgelsen fastslået at der ikke var nemt tilgængelige effektmodstande som kunne overholde vores krav. Tilgængelig var effektmodstande¹ på 6.8Ω som kunne klare 50W. Med en spændingskilde på 12V, og 4 effektmodstande parallelt forbundet med 6.8Ω ville der blive afsat 84.7W tilsammen i de 4 modstande.

$$P = \frac{U^2}{(4 \cdot R^{-1})^{-1}} = \frac{(12V)^2}{(4 \cdot (6.8\Omega)^{-1})^{-1}} = 84.7W$$

Dette er over de 80W der først var antaget, men da varmen er styret gennem en mosfet kan den mængde af effekt per tid der bliver generet nemt ændres.

Kredsløbet er blevet simuleret i multisim for at se efter overensstemmelse mellem analysen og simulering via følgende kredsløb. (figur 3.2)



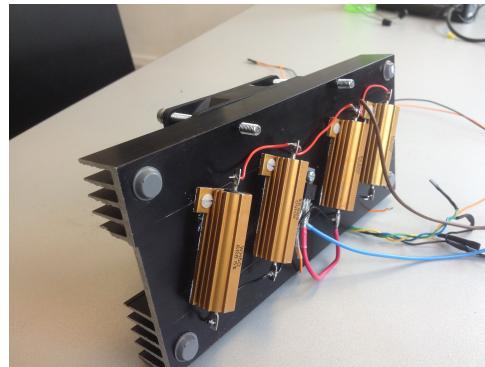
Figur 3.2. Varmelegeme simuleringskredsløb

Hvis mængden bliver for stor kan en modstand nemt frakobles og derved vil der blive afsat en fælles effekt på 63.5W.

$$P = \frac{U^2}{(3 \cdot R^{-1})^{-1}} = \frac{(12V)^2}{(3 \cdot (6.8\Omega)^{-1})^{-1}} = 63.5W$$

¹Se datablad for effektmodstanden [9]

Kølepladen som effektmodstandende skal opvarme, er af en størrelse der i længden kan håndtere 2 blæsere, men der ses ingen grund til bedre fordeling af temperaturen og der fastholdes at en enkelt blæser er nok til fordeling af varmen. Effektmodstandende bliver monteret fast på den ikke rillede side af kølepladen hvorimod blæseren bliver monteret på den rillede side af kølepladen. Det færdige varmelegeme kan ses på figur 3.3



Figur 3.3. Det fysiske varmelegeme

3.1.3 Test

Til første test havde vi en strømforsyning der ikke leverede de 12V som den skulle, men leverede 10V. Med denne forsyning fik vi en fælles strøm gennem modstandene på 5.4A, og derved en effekt på 64.8W. Dette er tilfredsstillende da der ved simuleringen var en strøm på ca. 7A med en spænding på 12V.

3.2 Befugter

For at kunne opretholde et optimalt udrygningsmiljø, er det nødvendig med befugtning af luften i rugemaskinen. Følgende afsnit vil omhandle dette emne.

3.2.1 Design

Der har været følgende design-idéer i spil.

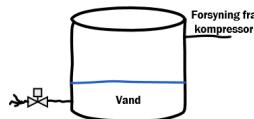
- Dryppe vand direkte på varmelegeme, for derefter at lade varmelegeme fordamp vandet.
- Forstøvet vandet fint via en dysse.

Det blev valgt at gå videre med idéen om at forstøve vandet via en dysse. Første udkaste bestod i:

- En beholder fyldt med vand under et konstant tryk via en kompressor. Selve styringen af forstøvningen af vandet skulle forgå ved at åbne/lukke en magnetventil (figur 3.4).

Denne idé blev droppet da det ville kræve at brugerne af rugemaskinen skulle havde adgang til en kompressor, samt der skulle bestilles en typegodkendt trykluftbeholder hjem, hvilket ville fordyre projektet væsentligt.

Andet udkast bestod i en ombygget havesprøjte, af nedenstående type (figur 3.5).



Figur 3.4. Princip tegning af første udkast



Figur 3.5. Havesprøjte

3.2.2 Implementation

Pga. en misforståelse med mekanikværkstedet, blev befugteren ikke konstrueret som tiltænkt. Dyssen og magnetventilen endte med at blive fastmonteret på selve vandbeholderen og ikke med en fleksibel slange, som der ellers var idéen.

På figur 3.6 se det færdig resultat af befugteren.



Figur 3.6. Havesprøjte med på monteret magnetventil

På magnetventilen er der monteret en diode², denne diode har til formål at beskytte vores elektronik mod de transienter, som der opstår når forsyningsspændingen fjernes til magnetventilen. Dioden er monteret som en ekstra sikkerhed, da der ligeledes er indbygget transient beskyttelse i vores driver-print.

3.2.3 Test

Følgende test er udført på befugteren.

- Beholder blev tryksat.
 - Det kunne konstateres at systemet var tæt i lukket tilstand.
- Magnetventil blev forsynet med 12VDC.
 - Det kunne konstateres at vandet blev forstøvet som ønsket, dog kunne en finere forstøvning af vandet have været optimalt.
- Det blev gradvis forsøgt at nedsætte længden af impulsen for åbningen af magnetventilen. Denne test blev fortaget for at finde den korteste puls hvorved magnetventilen kunne åbnes og lukkes tilfredsstillende.
 - Efter denne test kunne det konstateres, at en puls på 10ms var den korteste tid hvorved magnetventilen kunne åbnes.
 - * En puls på 5ms var ikke tilstræklig til at åbne magnetventilen.
 - * En puls på 7ms medførte et ustabilt resultat

²Af typen 1N5819[4]

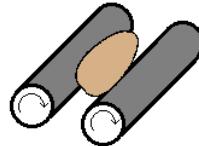
3.3 Ægvender

For at muliggøre vendingen af æggene, er det nødvendig med en mekanisme til at fortage rotation af æggene. Det følgende afsnit vil omhandle dette.

3.3.1 Design

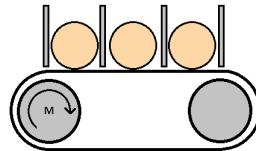
Under designprocessen var der to forskellige typer af mekanismer i spil.

- Placere æggene mellem to valser, der vender æggene ved at fortage rotation i samme retning.



Figur 3.7. Første udkast til æggevender

- Placere æggene på et underlag, hvor selve underlaget bevæger sig.



Figur 3.8. Andet udkast til æggevender

Det blev besluttet at gå videre med løsningen på figur 3.8.

3.3.2 Implementation

Pga. tidspres, samt da hovedfokus i projektet ikke ligger på implementering af mekaniske elementer, er det blevet valgt, at fortage implementering kun med selve aktuatoren som en illustrativ visning af funktionaliteten.

Som aktuator type til udførelsen af æggevendingen er der valgt en stepmotor, da denne type motor har egenskaber der egner sig godt til positionering. Da der ikke stilles nogle særlige krav til stepermotoren (opløsning, moment) i forbindelsen med æggevendingsmekanismen, er det valgt at bruge en stepermotor af typen 28BYJ-48³, fordi skolen havde denne stepermotor på lager. Dette er en gearet unipolær stepermotor, hvilket betyder at den har et højt moment pga. gearingen på 64 gange. Denne gearing medfører en forholdsvis lav max-rotationshastighed. Da det ønskes at æggene bliver vendt 180 grader i et stille tempo, vil dette ikke medføre problemer. Der henvises til afsnittet om 3.5 Aktuator interface for beskrivelse af hvorledes den elektroniske styring af stepermotoren fortages, samt til afsnittet om 4.4 Stepmotor.

³Se datablad for stepermotor 28BYJ-48 [7]

3.3.3 Test

Da den mekaniske del af æggevenderen ikke er blevet implementeret, er der ikke fortaget test af dette. For test af stepermotor henvise der til afsnittet om 4.4 stepermotor-softwaren, samt for test af stepermotor-driveren henvises der til afsnittet om aktuator-interface 3.5.

3.4 Låge kontakt

3.4.1 Design

For at muliggøre registreringen af lågens status, er det nødvendig med en kontakt, som har følgende egenskaber:

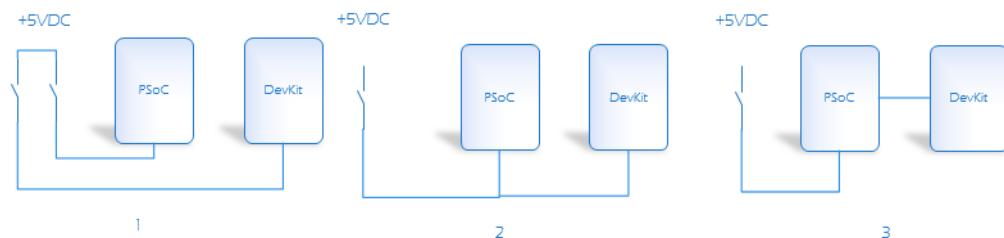
- Et GPIO i form af 5VDC.
- Være udført som en normal closed kontakt, således at evt. ledningsbrud er overvåget.

Lågekontakten skal både forbindes med PSoC3 og DevKit8000. Da begge enheder har brug for at kende lågens status.

Dette kan udføres på flere måder:

1. To separate kontakter; en til PSoC3 samt en til DevKit8000.
2. En kontakt med et signal til PSoC3, samt et signal til DevKit8000.
3. En kontakt med et signal til PSoC3, som herfra router signalet videre til DevKit8000'et.

Nedenstående figur 3.9 illustrerer de tre forskellige løsninger.



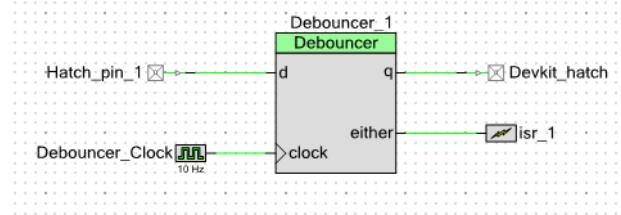
Figur 3.9. Designnudkast til lågekontakten

Løsningen blev af route signalet fra PSoC3 til DevKit8000'et. Dette har følgende fordele fremfor de to andre nævnte mulige løsninger:

- Det holder antallet af hardware-komponenter på et absolut minimum.
- Gør at der opstår en klar, samt naturlige grænse mellem alle sensorer og resten af systemet.
- Har den fordel, at der kun skal implementeres et debouncefilter, for at fjerne prel fra kontakten på PSoC3.

3.4.2 Implementation

For at fjerne kontaktprel fra lågekontakten, er der implementeret et prefilter af nedenstående type i PSoC3 (figur 3.10).

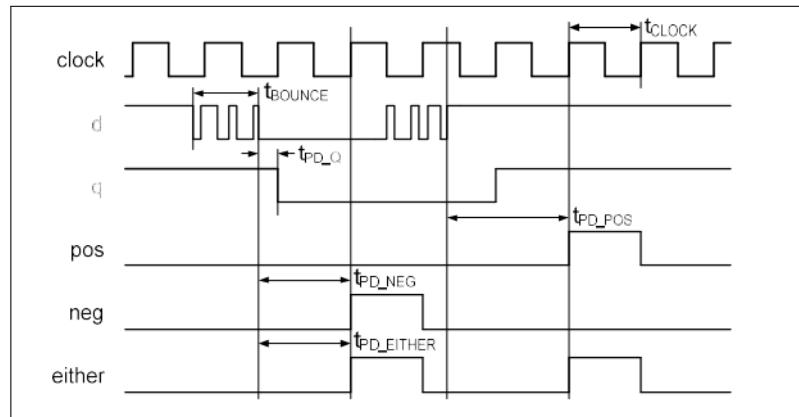


Figur 3.10. Debounce filter PSoC3

Det er valgt at køre med en relativ lav clock på 10Hz til debouncerne, da det er begrænset hvor hurtigt det er nødvendigt at registrere skift i lågens status.

q-udgangen på debouncefiltret, er ført vider til DevKit8000'et. Denne udgang er filtreret for kontaktprel.

Nedenstående figur 3.11 viser timing-diagrammet over debounce'erne.

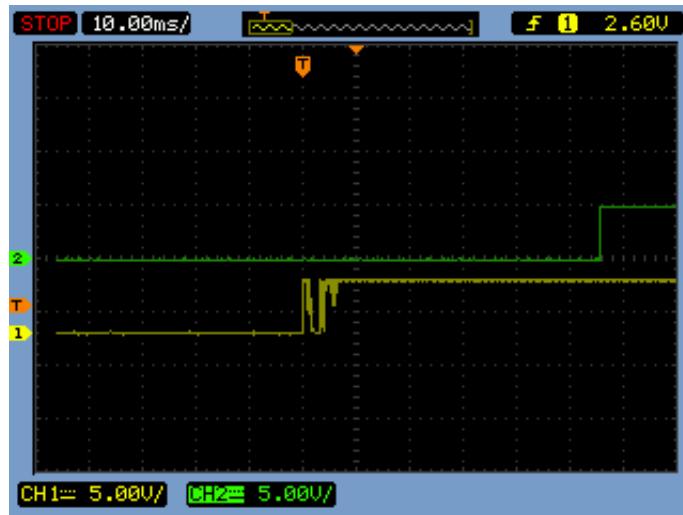


Figur 3.11. Debounce filter timing

Der bliver kaldt en ISR på både den positive og den negative flanke, da debouncefiltret er sat i "either"-mode. ISR toggler et flag, som bliver brugt i "Main" til at overvåge lågens status.

3.4.3 Test

Der blev skrevet et testprogram til PSoC3. Dette testprogram bestod i en ISR-del, som toggler en lysdiode på PSoC3. Der blev også målt på udgangen til DevKit8000'et med et oscilloskop. Nedenstående figur 3.12 viser dette udgangssignal (grøn) i forhold til indgangssignalet (gul). Det kan ses at udgangssignalet til DevKit8000'et er fri for prel.



Figur 3.12. Kontakt prel

3.5 Aktuator interface

For at muliggøre styringen af aktuatorerne i systemet (stepermotoren, magnetventilen, samt de to blæsere) er det nødvendig med en form for bindeled mellem PSoC'en og de fysiske komponenter. Det følgende afsnit vil omhandle dette.

3.5.1 Design

Der er flere måder at fortage styringen af disse aktuatorer på, bl.a. via:

- Relæ
- Solid-state relæ
- Effekttransistor
- Mosfet
- Darlington-array.

Til at trække stepermotoren, magnetventilen, samt de to blæsere faldt designvalget på et darlington-array.

Dette designvalg blev truffet på baggrund af følgende overvejelser.

- Relæet ville ikke være optimalt i PWM-sammenhæng pga. det store mekaniske slid dette ville medføre kontakter. Det er heller ikke muligt at køre med en ret høj PWM-frekvens pga. spolen i relæet.
- Solid-state relæet kunne have været en mulighed, da der ikke er noget mekanisk slid i denne form for relæ. Solid-state relæet blev valgt fra pga. prisen og den fysiske byggestørrelse på komponenten.
- Effekttransistoren har en lille forstærkning, hvilket ville kræve en større strøm end de 4mA PSoC3 er i stand til at levere på dens udgange. Derfor blev denne fravalgt.
- Mosfet'en kunne have været et oplagt valg, men blev fravalgt da det blev ønsket at holde antallet af elektriske komponenter på et minimum, hvilket et darlington-array muliggjorde.

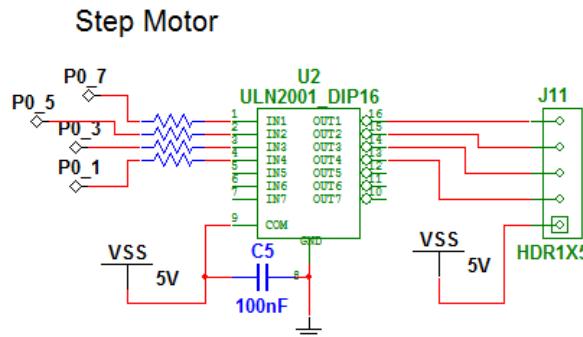
3.5.2 Implementation

Aktuatorerne blev fordelt på to forskellige darlington-arrays.

- Et darlington-array til stepmotoren.
- Et darlington-array til de 2 blæser, samt magnetventilen.

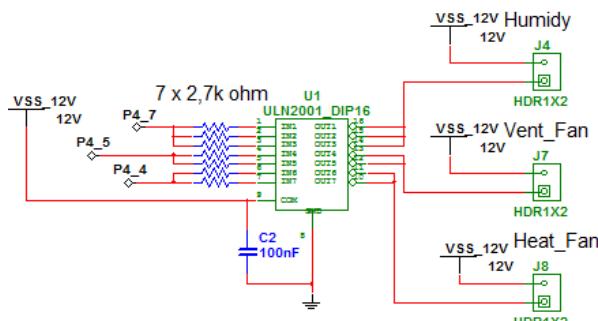
Dette skyldes de to forskellige spændingsniveauer på hhv. 5VDC og 12VDC.

Nedenstående figur 3.13 viser implementeringen af darlington-array'et til step-motoren.



Figur 3.13. Driver kredsløb Step motor

Nedenstående figur 3.14 viser implementeringen af darlington-array'et til de to blæsere samt til magnetventilen.



Figur 3.14. Driver kredsløb

Det meste optimale komponentvalg ville have været at anvende en ULN2003⁴. Men da skolen lagerfører en ULN2001⁴, er det valgt at bruge denne. Forskellene på disse to IC'er er, at ULN2003 har indbygget $2.7\text{k}\Omega$ formodstande, således at den er tilpasset til 5VDC CMOS kredse. Hvorimod ULN2001 er et darlington-array til generel brug, og har derfor ikke indbyggede formodstande. Dette er baggrunden for de 7 stk $2.7\text{k}\Omega$ modstande placeret foran darlington-array'et. Der er kun anvendt "flyback-diode på magnetventilen. Dette er udelukket som en ekstra sikkerhed, da der er indbygget flyback-dioder i dalington-array'et.

⁴Se datablad for ULN2001[8]

Hver udgang på darlington-array'et er i stand til at trække en strøm på 500mA⁴, dog er det muligt at parallelkoble flere udgange for at opnå en større strøm. Denne mulighed er udnyttet, da magnetventilen trækker en strøm på 1A. For at have en sikkerhedsmargin, er der anvendt 3 udgange. Belastningen af de to blæsere er fordelt på de resterende udgange. Der er valgt ikke at montere en køleplade på IC'en, da magnetventilen, som er den største strømforbruger, kun er aktiv i 10ms af gange.

3.5.3 Test

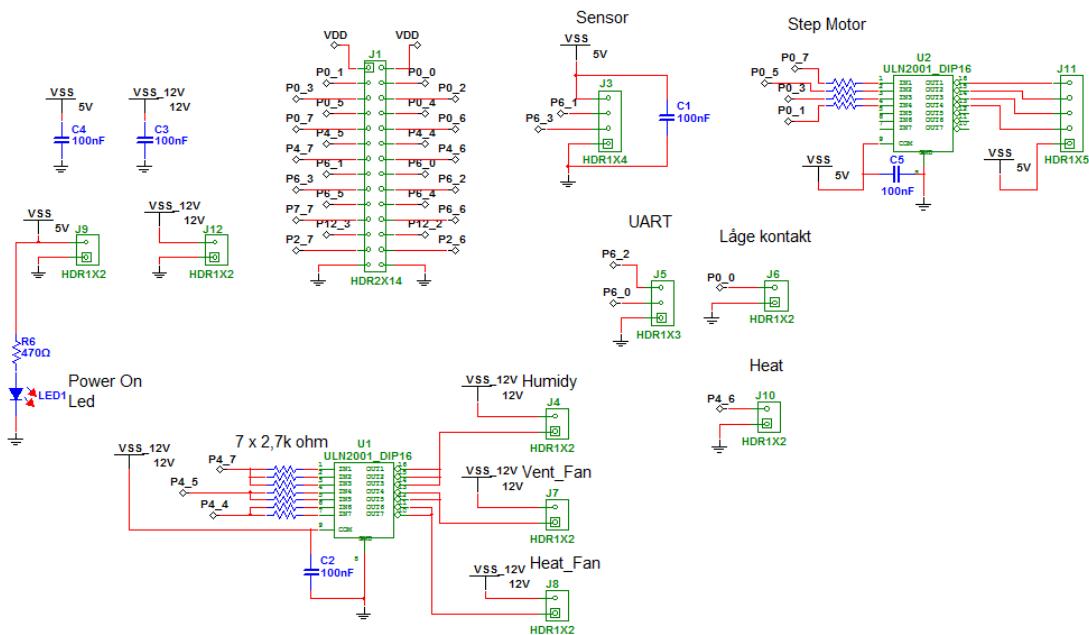
De respektive indgange på darlington-array'et blev forsynet med 5VDC, da udgangsspændingen blev målt. Ved denne test kunne det konstateres, at spændingsniveauet på udgangene lå på hhv. 5VDC ved darlington-array'et til step-motoren, samt 12VDC ved darlington-array'et til de to blæsere og magnetventilen.

3.6 Printplade

Følgende afsnit omhandler driver-printet. Dette print har til formål at forbinde sensorerne og aktuatorene til PSoC3.

3.6.1 Design

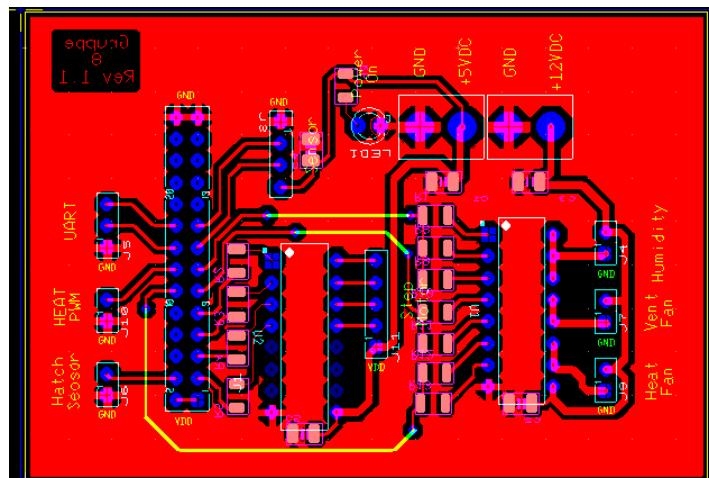
Figur 3.14 viser det komplette kredsløb til driver-printet. Der er lagt vægt på et forholdsvis kompakt design samt valg af SMD-komponenter, hvis disse var tilrådighed. Der er tilføjet 4 kondensatorer som spændingsstabilisering. Disse er placeret så tæt på komponenterne som mulig. Via harwin-stikket J1 forbindes printet til PSoC3.



Figur 3.15. Printplade

3.6.2 Implementation

Printet er implementeret på et 2-sidet PCB, da dette giver mulighed for at have et stelplan til mest mulig at begrænse støj både fra og til printet. Endvidere giver et 2-sidet PCB også mulighed for at tilføje tekst på forsiden af printet, der tydeliggør hvor de forskellige sensorer og akutatorer skal tilsluttes. Figur 3.16 viser det færdige print.



Figur 3.16. Printplade

Da der sent i projektet blev skiftet sensortype, mangler der 2 pull-up modstande til sensoren. Disse er monteret på et særskilt print.

3.6.3 Test

Efter montering af komponenterne blev følgende test fortaget.

- Modstanden mellem 5VDC, 12VDC og GND blev målt for at være sikker på, at der ikke var en kortslutning.
- Printet blev spændingssat, og det blev observeret at *Power On* lyste.
- Kablet til PSoC3 blev forbundet, hvorefter der blev fortaget en IO-test af printet. Under denne test blev det konstateret, at stikket *J1* var blevet spejlvendt. Dette problem blev løst ved at ændre pins'ene i PSoC Creator. Figur 3.17 viser det oprindelige samt det nye PinOut.

PSoC 3	Pin Out	
PWM	Oprindeligt	Ny
PWM_Heat_Fan	P4.4	P4.5
PWM_Heat	P4.6	P4.7
PWM_Vent_Fan	P4.5	P4.4
Humidy	P4.7	P4.6
Step Motor		
Step Motor Pin 1	P0.1	P0.0
Step Motor Pin 2	P0.3	P0.2
Step Motor Pin 3	P0.5	P0.4
Step Motor Pin 4	P0.7	P0.6
Sensor		
SCL	P6.1	P6.0
Data	P6.3	P6.2
Låge kontakt	P0.0	P0.1
UART		
Tx	P6.0	P6.1
Rx	P6.2	P6.3

Figur 3.17. Pin Out

Software 4

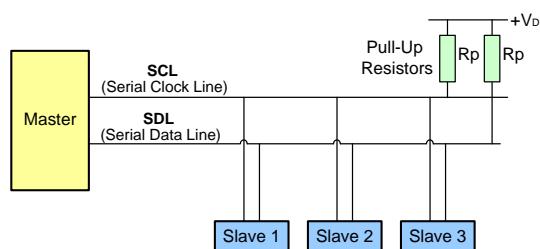
4.1 Sensor

4.1.1 Design

Til udrugningen skal luftfugtigheden og temperaturen holdes på et fast niveau. Ved lidt hurtigt søgering i lokal varekakaori, er der fundet en sensor som kunne måle begge dele. Sensoren har dog en kompliceret protokol. Efter noget research og forsøg med at få lavet et program, viser det sig dog, at for at få sensoren til at virke, vil det kræve en ekstra microprocessor kun til at arbejde med sensoren. Derfor vælges sensoren HIH6121-021, som har samme nødvendige funktionalitet som sensoren SHT71 dog knapt så fintfølende. Sensoren overholder stadigvæk kravspecifikationerne og kommunikerer med en standard I2C protokol, hvilket vil være forholdsvis nemt at implementere. HIH6121-021 sensoren har et spændingsniveau mellem 2,5 V og 5,5 V, da PSoC3'ens spændingsniveau ligger på 3,3 V eller 5 V vil det ikke være et problem uanset PSoC3'ens spændingsniveau. Da sensoren følger en I2C protokol, skal der ikke tages hensyn til støj.

4.1.2 Implementering

Sensoren HIH6121-021 skal have 2 pull-up modstande på $2,2\text{ k}\Omega$, en til hver af kommunikationslinjerne. Yderligere skal der sættes en kondensator på 22 nF mellem forsyningens spændingen V_{DD} og GND . Sensoren bliver forsynet af PSoC'en da dens forbrugsstrøm er så lille, at den uden problemer kan forsynes direkte derfra. Opstillingen kan ses på nedenstående figur 4.1. Sensorens spændingsniveau skal ligge mellem $2,3 - 5,5\text{ V}$, hvilket gør at det ikke er kritisk hvilket spændingsniveau PSoC'en kører med.¹ [6]



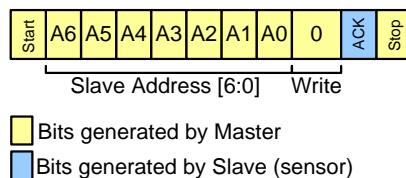
Figur 4.1. Generel opstilling til I2C kommunikation

¹PSoC'en kan enten kører med 3,3 V eller 5 V hvilket ligger inden for sensorens spændingskrav

Sensoren fungerer ved at man først skal bede om at få målt luftfugtighed og temperatur, og derefter bede om resultatet fra målingerne. Der er 2 return-datatyper. Den ene returnerer kun luftfugtigheden, den anden returnerer både luftfugtigheden og temperaturen. Det er valgt, at sensoren altid returnerer begge værdier. PSoC Creator har indbygget et modul til I2C kommunikation, som der bliver brugt til kommunikation med sensoren. Til at starte en måling bliver der først sendt et startsignal efterfyldt af 7 bit, som repræsenterer adressen på den slave der kommunikeres med.² Den 8. bit, der sendes, indikerer til sensor at den enten skal måle eller returnerer målte værdier. Hvis det 8. bit er et nul, skal den måle værdierne af fugtighed og temperatur, hvis det er "1" skal slaven returnerer de målte værdier. På figur 4.2 sættes protokollen til at måle luftfugtigheden og temperaturen. Det udføres på PSoC'en vha. følgende kode.

```
err = I2C_HIH61xx_MasterSendStart(Address,0); // Read
err = I2C_HIH61xx_MasterSendStop(); //Send Stop
CyDelayUs(500);
```

CyDelayUs(500); bliver brugt til at sikre sensor tid til at få målt.



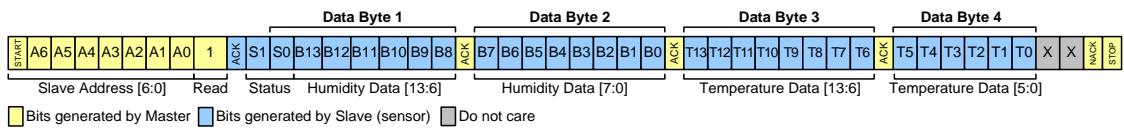
Figur 4.2. Send Measure kommando

På figur 4.3 vises protokollen[5] for aflæsning af værdierne på sensoren. Der er 4 data bytes, hvilket viser at både fugtighed og temperatur bliver målt. Status bits'ne (S0 og S1) bliver ikke kontrolleret. Følgende kode er anvendt til aflæsning af sensorens værdier.

```
err = I2C_HIH61xx_MasterSendStart(Address,1); // Read
for(i = 0; i < 4; i++)
{
    if(err == I2C_HIH61xx_MSTR_NO_ERROR && i < 3)
        buffer[i] = I2C_HIH61xx_MasterReadByte(I2C_HIH61xx_ACK_DATA);

    if(err == I2C_HIH61xx_MSTR_NO_ERROR && i == 3)
        buffer[i] = I2C_HIH61xx_MasterReadByte(I2C_HIH61xx_NAK_DATA);
}
err = I2C_HIH61xx_MasterSendStop(); //Send Stop
```

²HIH6121-021's default adresse er 0x27

**Figur 4.3.** Aflæsning af målte værdier.

Efter modtagelse af data bytes, bliver dataerne behandle og allokeret i lokale værdier (*static float humidity* og *static float temperatur*), ved at gøre dette, undgår main programmet at skulle udfører kommunikationen med sensoren, når den kan bruge værdierne, men derimod kan kører funktioner når det passer ind i dens sequence. Til at hente værdierne er funktionerne *getTemp()* og *getHumid()* lavet, de returner de sidste målte værdier. Dette gøre det også nemt for andre funktioner, at hurtigt indhente værdier på enten luftfugtighed eller temperatur.

void Init_HIH61xx(void)

Description	Initialiserer I2C komponenten til kommunikation med sensor HIH61xx
Parameters	(void)
Return Value	(void)

uint8 Measure_HIH61xx(void)

Description:	Beder sensoren om at måle luftfugtigheden og temperaturen
Parameters:	(void)
Return Value:	(uint8) returner errorstatus værdi fra I2C komponenten

uint8 Read_HIH61xx(void)

Description:	Beder sensoren om målte værdier af fugtighed og temperatur, derefter behandler og allokerer værdierne lokalt
Parameters:	(void)
Return Value:	(uint8) returnerer errorstatus værdi fra I2C komponenten

float getTemp(void)

Description:	Returnerer værdi af lokalt gemt værdi <i>Temperatur</i>
Parameters:	(void)
Return Value:	(float) Returnerer lokal værdi <i>Temperatur</i>

float getHumid(void)

Description:	Returnerer værdi af lokalt gemt værdi <i>Humidity</i>
Parameters:	(void)
Return Value:	(float) Returnerer lokal værdi <i>Humidity</i>

Der er ikke blevet valgt at håndtere statusmeddelelser fra sensoren og heller ikke errorstatus fra I2C komponenten med nogle undtagelser. *measure_HIH61xx()* kører i loop indtil den får sendt et rigtigt signal. Ved fejl i læsningen af målte værdier stopper den processen.

4.2 Varmeregulering

4.2.1 Design

Formålet med dette software er at styre det fysiske varmelegeme så systemet opnår en ønsket temperatur.

1. indenfor den i kravspecifikationen angivne tidsramme,
2. som holdes indenfor de i kravspecifikationen erklærede rammer,
3. fejlmeddeler hvis punkt 1) og 2) ikke kan overholdes.

Til dette designes følgende funktioner, som beskrevet i klassediagrammet under "Varmeregulering":

Funktionsnavn:	Beskrivelse:
<code>void initTemp(void)</code>	Skal initialisere de enheder, på PSoC3, som skal benyttes til regulering af temperaturen.
<code>void regTemp(float temp)</code>	Skal igangsætte at systemet indstiller sig på og holder <code>temp</code> parameterens temperatur.
<code>void pauseTemp(void)</code>	Skal kunne afbryde og igangsætte reguleringen igangsat af <code>regTemp</code> .
<code>void stopTemp(void)</code>	Skal afbryde reguleringen igangsat af <code>regTemp</code>

Bemærk at `initTemp()` ikke fremgår af klassediagrammet.

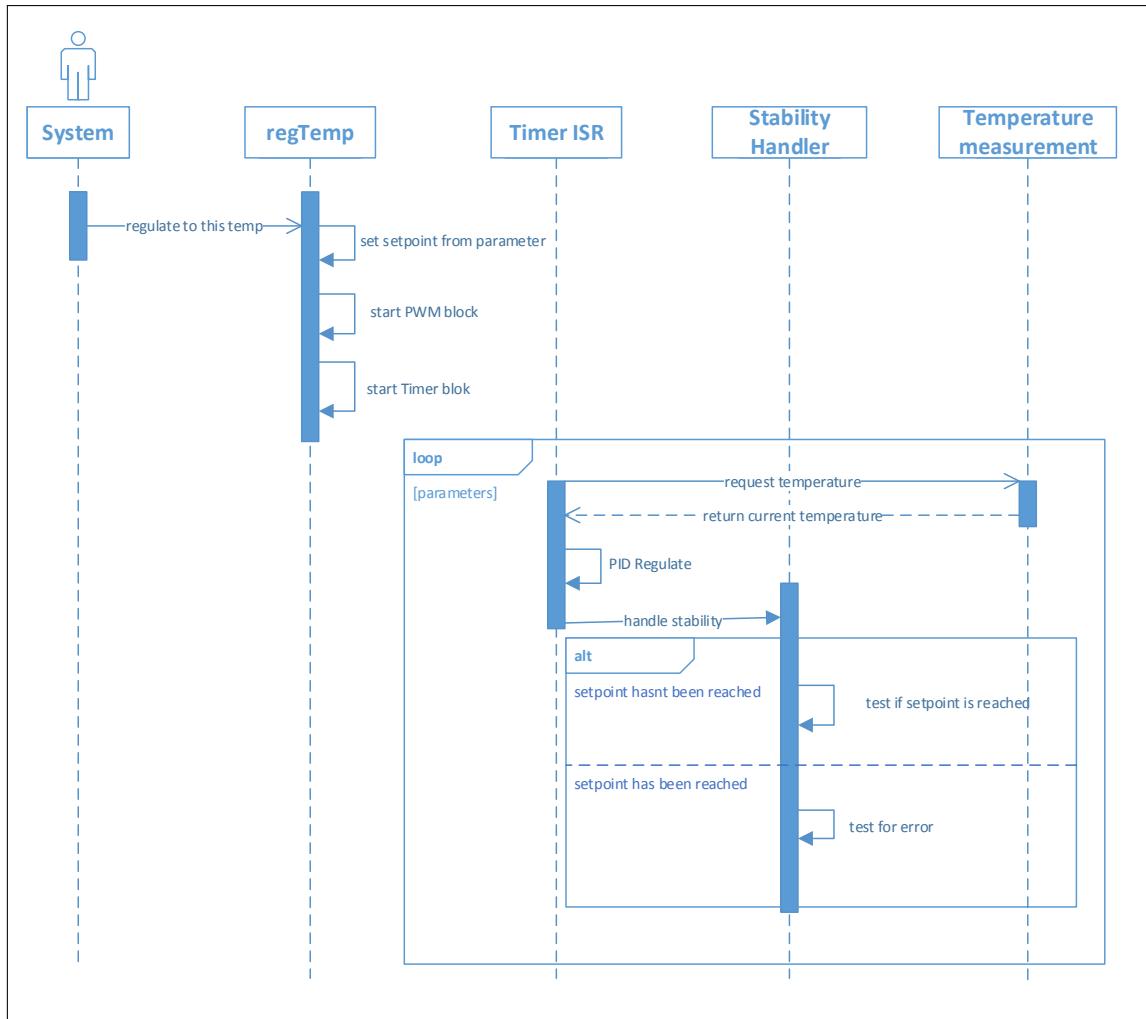
Desuden ønskes softwaren at styre den cirkulationsventilator, der er opgivet på BDD'et.

Softwaren designes således, at en PID regulering udføres via ISR, der initieres ved brug af en Timer. Dette gøres for at gøre denne del af systemet autonom. Da denne del af systemet ikke er tidskritisk vil denne ISR have lav prioritet.

Da denne del implementeres på PSoC3 benyttes en PWM blok med to udgange, en til reguleringen af varmelegemets PWM signal og en til at styre cirkulationsventilatoren, samt en Timer blok. Desuden benyttes to Digital Out pins, en til hver PWM udgang.

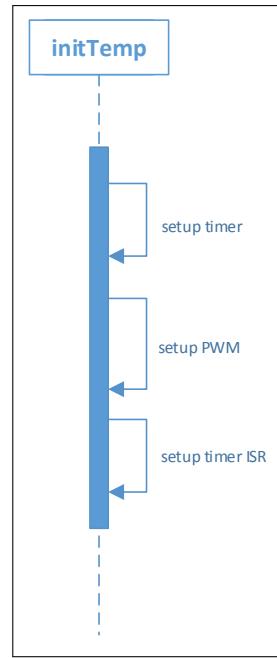
Selve softwaren til styring af varmereguleringen bygges op omkring de standard funktioner, der hører til blokke i PSoC3 Creator.

De fire funktioner implementeres efter følgende sekvensdiagrammer:



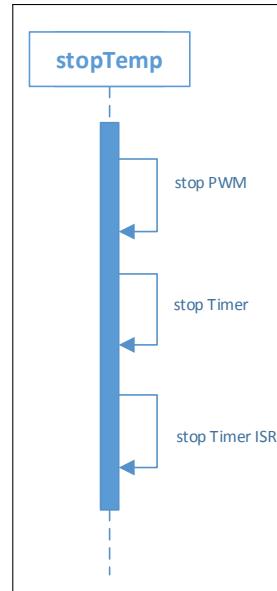
Figur 4.4. Sekvensdiagrammet for `void regTemp(float)`

Figur 4.4 viser sekvensdiagrammet for `regTemp(float)` funktionen. Funktionaliteten af "Temperature measure" sjæljen bliver ikke designet i dette software; funktionaliteten af denne er at returnere den aktuelle temperatur, som reguleringen skal foretages efter. "Stability Handler"-funktionen skal holde øje med om systemet overholder de i kravspecifikationen opgivne krav mht. tilladte afvigelser og indstillings-hastighed, samt give systemet besked om hvis dette ikke sker.



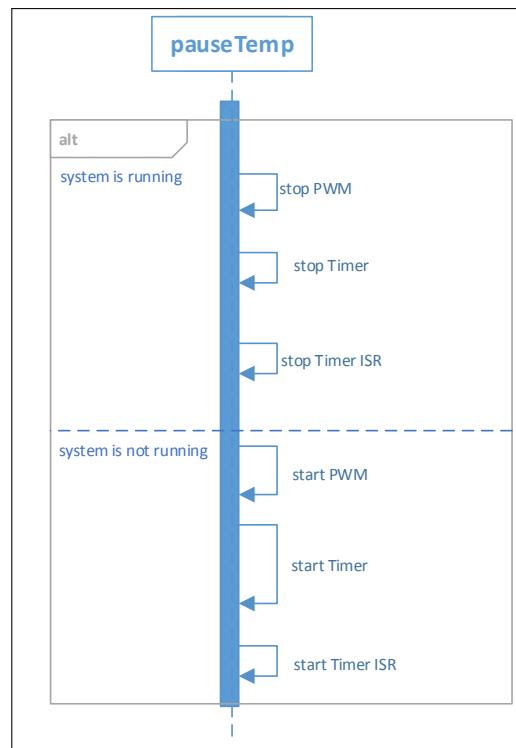
Figur 4.5. Sekvensdiagram for void initTemp(void)

Figur 4.5 viser sekvensdiagrammet for `initTemp()`.



Figur 4.6. Sekvensdiagram for void stopTemp(void)

Figur 4.6 viser sekvensdiagrammet for `stopTemp()`.

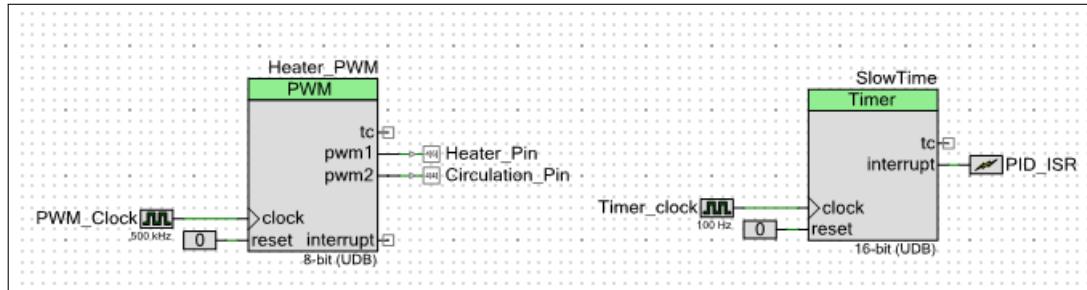


Figur 4.7. Sekvensdiagram for `void pauseTemp(void)`

Figur 4.7 viser sekvensdiagrammet for `pauseTemp()`.

4.2.2 Implementation

Herunder vises PSoC3 Creator topdesign for implementeringen:



Figur 4.8. PWM blok PSoC3

Heater_PWM Blok: Denne blok driver varmelegemets PWM; PWM1 indstilles i Timer ISR, PWM2 indstilles til 50% duty cycle til at drive cirkulationsventilatoren. Implementeret med en 8 bit opløsning og 300 kHz clock signal.

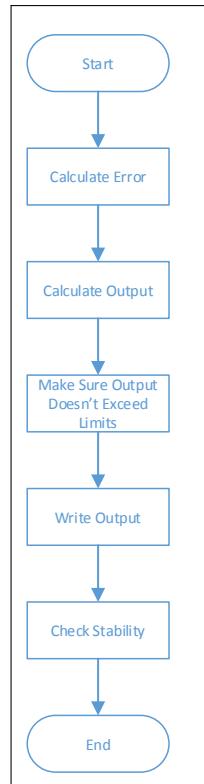
SlowTime blok: Driver den automatiske regulering af varmelegemets PWM styringen. Implementeret med en 100 Hz clock og et 3000 step, hvilket effektivt giver en periode på 30 sekunder.

Der er implementeret et flag til detektering af fejl, ved navn **errorOccured**, der er 1 hvis der er opstået en fejl, og 0 ellers.

Der er oprettet et styreflag, kaldet **isRunning**, der sættes til 1 når en regulering igangsættes, og 0 når reguleringen stoppes. Det benyttes når

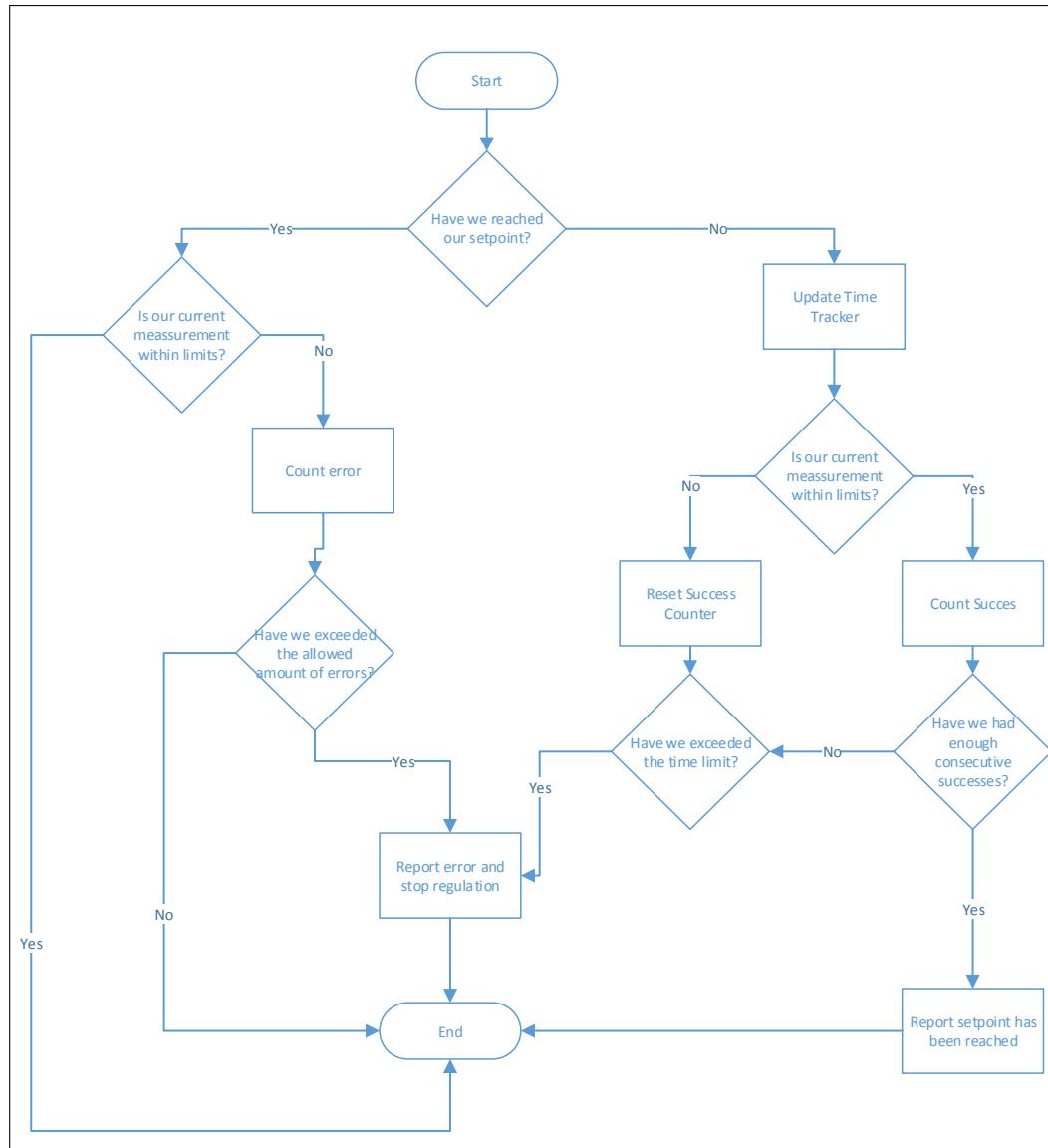
1. **regTemp(float)** kaldes: hvis flaget er 0 tændes komponenterne og flaget sættes til 1
2. **pauseTemp()** kaldes: hvis flaget er 1 kan udrugningen sættes på pause.
3. **stopTemp()** kaldes: hvis flaget er 1 stoppes komponenterne og flaget sættes til 0.

Selve ISR rutinen følger dette diagram:



Figur 4.9. Flowdiagram for PID_ISR

"Stability handler"-funktionen fungerer som beskrevet i figur 4.10:



Figur 4.10. Flowdiagram for Stability Handler funktionaliteten

Som standard betragtes systemet værende "stabilit" hvis temperaturen holdes indenfor de tilladte grænser i en periode af 1 minut. Når systemet har opnået stabilitet holdes øje med om målingerne begynder at ligge udenfor de tilladte grænser. Som standard er det ikke tilladt at værdierne falder udenfor den angivne grænse.

Til evt. at ændre disse regler er følgende defines lavet:

```

#define ERR_MARGIN - Den tilladte afvigelse, +/- 
#define SUCCES_LIMIT - Antallet af på hinanden efterfølgende succeser, der skal til før systemet betragtes stabilt
#define ERROR_LIMIT - Antallet af på hinanden efterfølgende fejl, der skal til før systemet betragtes som ustabil

```

Fælles for `#define SUCCES_LIMIT` og `#define ERROR_LIMIT` er, at den tidsmæssige perioden, de dækker over, beregnes ud fra Timerens periodetid multipliceret med disse tal.

4.2.3 Test

Til test af softwarens egenskaber laves en driver til at generere et temperatur output som PID reguleringen kan reagere på. Til monitorering benyttes UART output. Testen gør følgende:

1. Systemet initialiseras.
2. Testprogrammet får besked på at holde en temperatur på 37 celcius; tidsmonitorering påbegyndes.
3. Testprogrammet får i perioder af 5 sekunder at vide at temperaturen nærmer sig 37 grader; når testprogrammet når indenfor de tilladte grænser, tælles succes-counteren op
4. Testprogrammet når at holde temperaturen i 30 sekunder indenfor den tilladte tidsgrænse, succes flaget sættes.
5. Herefter bliver programmet sat til at skulle holde en temperatur på 34 grader celcius; tidsmonitorering påbegyndes.
6. Testprogrammet får i perioder af 5 sekunder at vide at temperaturen falder med 0.5 grader; når testprogrammet når indenfor de tilladte grænser, tælles succes-counteren op.
7. Når temperaturen falder udenfor de tilladte grænser nulstilles succes-counteren; når tidsgrænsen overskrides sættes error flaget og reguleringen stoppes.

Resultatet aflæst på UART'en:

```

SP: 37.00 - Err: 22.00 - PHM: 116 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 0
SP: 37.00 - Err: 11.00 - PHM: 168 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 1
SP: 37.00 - Err: 5.50 - PHM: 194 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 2
SP: 37.00 - Err: 2.75 - PHM: 207 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 3
SP: 37.00 - Err: 1.38 - PHM: 213 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 4
SP: 37.00 - Err: 0.69 - PHM: 216 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 5
SP: 37.00 - Err: 0.34 - PHM: 218 - SPR: 0 - ErrTrack: 1 - ErrOcc: 0 - Time: 6
SP: 37.00 - Err: 0.17 - PHM: 219 - SPR: 0 - ErrTrack: 2 - ErrOcc: 0 - Time: 7
SP: 37.00 - Err: 0.09 - PHM: 219 - SPR: 0 - ErrTrack: 3 - ErrOcc: 0 - Time: 8
SP: 37.00 - Err: 0.04 - PHM: 219 - SPR: 0 - ErrTrack: 4 - ErrOcc: 0 - Time: 9
SP: 37.00 - Err: 0.02 - PHM: 219 - SPR: 0 - ErrTrack: 5 - ErrOcc: 0 - Time: 10
SP: 37.00 - Err: 0.01 - PHM: 219 - SPR: 0 - ErrTrack: 6 - ErrOcc: 0 - Time: 11
SP: 37.00 - Err: 0.01 - PHM: 219 - SPR: 1 - ErrTrack: 0 - ErrOcc: 0 - Time: 0
SP: 37.00 - Err: 0.00 - PHM: 219 - SPR: 1 - ErrTrack: 0 - ErrOcc: 0 - Time: 0
SP: 37.00 - Err: 0.00 - PHM: 219 - SPR: 1 - ErrTrack: 0 - ErrOcc: 0 - Time: 0
SP: 37.00 - Err: 0.50 - PHM: 222 - SPR: 1 - ErrTrack: 0 - ErrOcc: 0 - Time: 0
SP: 34.00 - Err: -2.00 - PHM: 211 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 0
SP: 34.00 - Err: -1.50 - PHM: 204 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 1
SP: 34.00 - Err: -1.00 - PHM: 199 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 2
SP: 34.00 - Err: -0.50 - PHM: 197 - SPR: 0 - ErrTrack: 1 - ErrOcc: 0 - Time: 3
SP: 34.00 - Err: 0.00 - PHM: 197 - SPR: 0 - ErrTrack: 2 - ErrOcc: 0 - Time: 4
SP: 34.00 - Err: 0.50 - PHM: 200 - SPR: 0 - ErrTrack: 3 - ErrOcc: 0 - Time: 5
SP: 34.00 - Err: 1.00 - PHM: 205 - SPR: 0 - ErrTrack: 4 - ErrOcc: 0 - Time: 6
SP: 34.00 - Err: 1.50 - PHM: 212 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 7
SP: 34.00 - Err: 2.00 - PHM: 223 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 8
SP: 34.00 - Err: 2.50 - PHM: 235 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 9
SP: 34.00 - Err: 3.00 - PHM: 250 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 10
SP: 34.00 - Err: 3.50 - PHM: 255 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 11
SP: 34.00 - Err: 4.00 - PHM: 255 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 12
SP: 34.00 - Err: 4.50 - PHM: 255 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 13
SP: 34.00 - Err: 5.00 - PHM: 255 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 14
SP: 34.00 - Err: 5.50 - PHM: 255 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 15
SP: 34.00 - Err: 6.00 - PHM: 255 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 16
SP: 34.00 - Err: 6.50 - PHM: 255 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 17
SP: 34.00 - Err: 7.00 - PHM: 255 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 18
SP: 34.00 - Err: 7.50 - PHM: 255 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 19
SP: 34.00 - Err: 8.00 - PHM: 255 - SPR: 0 - ErrTrack: 0 - ErrOcc: 0 - Time: 20

```

Figur 4.11. Resultat af varmereguleringstest

Figur 4.11 visser test af varmeregulering på formatet: Setpoint - Error - PWM - Om setpointet er nået - Succeser - Om der er sket en fejl - Hvor lang tid derer gået siden der sidst blev sat et setpoint.

Som det ses sættes setpointet til 37 og Time tælles op. Når Err kommer under 1, tælles ErrTrack op. PWM outputtet reguleres også løbende. Da ErrTrack når op på 6 bliver SPR sat. Så bliver setpointet sat til 34, og temperaturen bliver ved med at falde. Da Time når 20 er setpoint ikke nået og systemet stopper.

Så testen viser, at systemet virker efter hensigten.

4.3 Befugtning

4.3.1 Design

Formålet med dette software er at styre den fysiske luftfugtighedsreguleringsmekanisme så systemet opnår en ønsket luftfugtighed.

1. Indenfor den i kravspecifikationen angivne tidsramme,
2. Som holdes indenfor de i kravspecifikationen erklærede rammer,
3. Fejlmeddeler hvis punkt 1) og 2) ikke kan overholdes.

Til dette designes følgende funktioner, som beskrevet i klassediagrammet under "Befugtning":

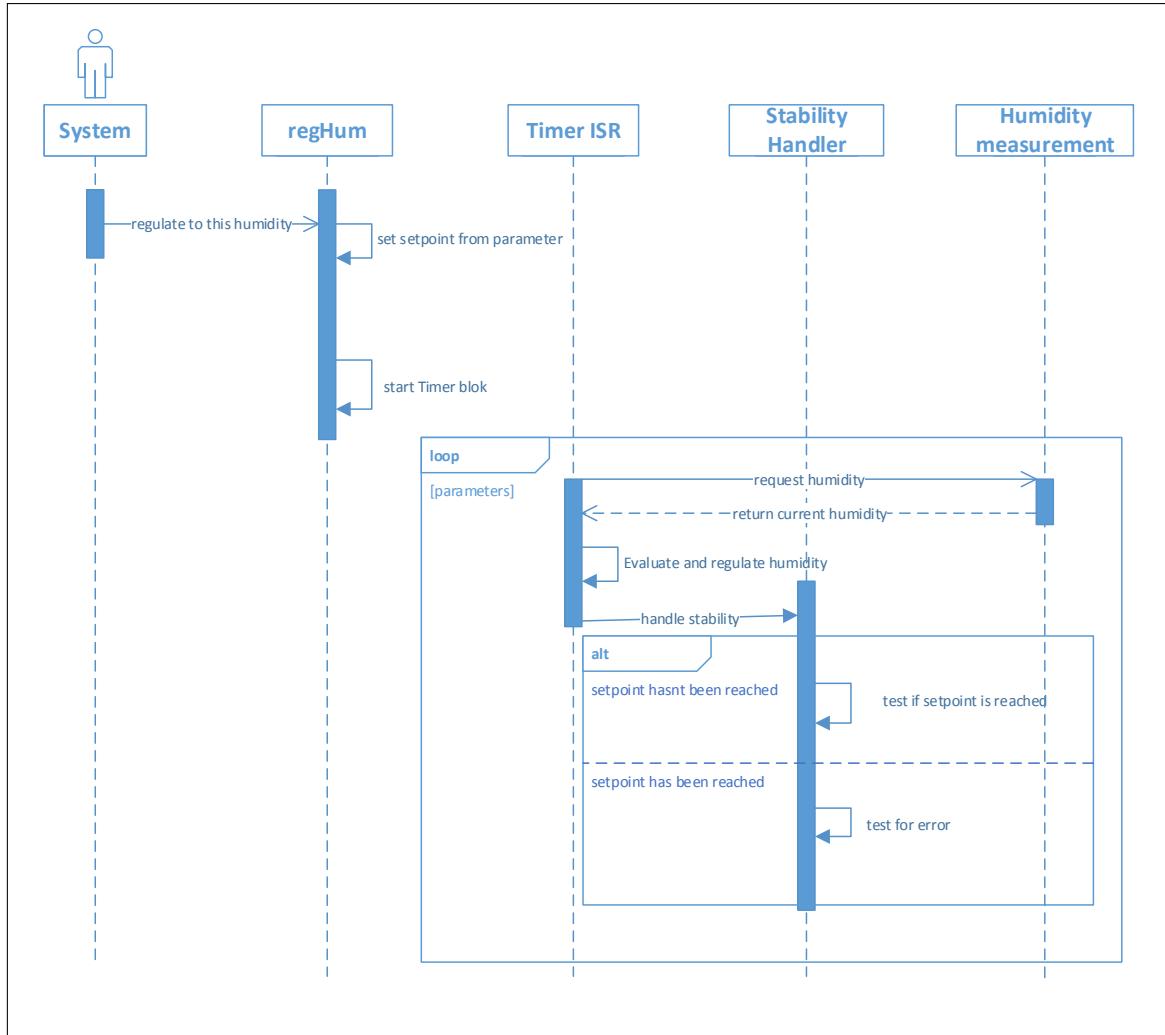
Funktionsnavn:	Beskrivelse:
<code>void initHum(void)</code>	Skal initialisere de enheder, på PSoC3, som skal benyttes til regulering af luftfugtigheden.
<code>void regHum(float hum)</code>	Skal igangsætte at systemet indstiller sig på og holder <code>hum</code> parameterens temperatur.
<code>void pauseHum(void)</code>	Skal kunne afbryde og igangsætte reguleringen igangsat af <code>regHum</code> .
<code>void stopHum(void)</code>	Skal afbryde reguleringen igangsat af <code>regHum</code>

Softwareen designes således, at regulering udføres via ISR, der initieres ved brug af en Timer, for at gøre denne del af systemet autonom. Siden denne del af systemet ikke er tidskritisk vil denne ISR være af lav prioritet.

Da denne del implementeres på PSoC3 benyttes en Timer blok samt en Digital Out pin. Pin'en benyttes til at styre hvornår og i hvor lang tid der befugtes.

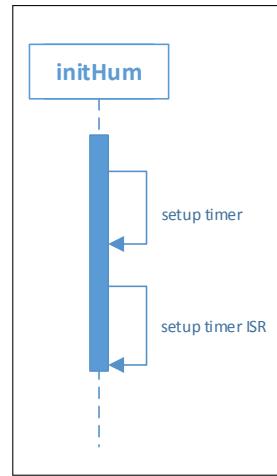
Selve softwaren til styring af varmereguleringen bygges op omkring de standard funktioner, der hører til blokke i PSoC3 Creator.

De fire funktioner laves efter følgende sekvensdiagrammer:



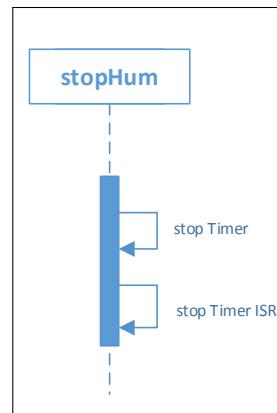
Figur 4.12. Sekvensdiagram for `void regHum(float)`

Figur 4.12 viser sekvensdiagrammet for `regHum(float)` funktionen. Funktionaliteten af "Humidity Measurement"-søjlen bliver ikke designet i dette software; funktionaliteten af denne er at returnere den aktuelle luftfugtighed, som reguleringen skal foretages efter. "Stability Handler"-funktionen skal holde øje med om systemet overholder de i kravspecifikationen opgivne krav mht. tilladte afvigelser og indstillings-hastighed, samt give systemet besked om hvis dette ikke sker.



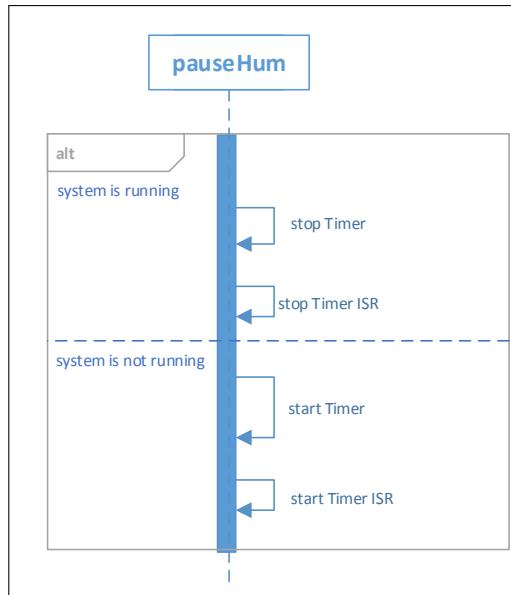
Figur 4.13. Sekvensdiagram for `void initHum(void)`

Figur 4.13 viser sekvensdiagrammet for `initHum()` funktionen.



Figur 4.14. Sekvensdiagram for `void stopHum(void)`

Figur 4.14 viser sekvensdiagrammet for `stopHum()` funktionen.

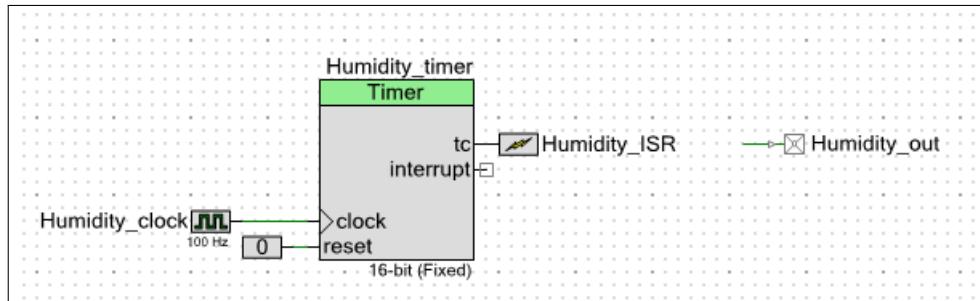


Figur 4.15. Sekvensdiagram for `void pauseHum(void)`

Figur 4.15 viser sekvensdiagrammet for `pauseHum()` funktionen.

4.3.2 Implementation

Herunder vises PSoC3 Creator topdesign for implementeringen:



Figur 4.16. Befugt PSoC3 Topdesign

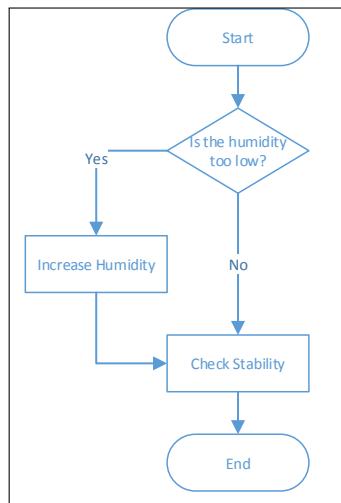
Humidity_timer blok: Driver den automatiske regulering af luftfugtigheden. Implementeret med en 100 Hz clock og et 3000 step, hvilket effektivt giver en periode på 30 sekunder.

Der er implementeret et flag til detektering af fejl, ved navn `errorOccured`, der er 1 hvis der er opstået en fejl, og 0 ellers.

Der er oprettet et styreflag, kaldet `isRunning`, der som default er 0. Det ændres når

1. `regHum(float)` kaldes: hvis flaget er 0 tændes komponenterne og flaget sættes til 1
2. `pauseHum()` kaldes: hvis flaget er 1 kan reguleringen pauses.
3. `stopHum()` kaldes: hvis flaget er 1 stoppes komponenterne og flaget sættes til 0.

Selve ISR følger dette diagram:



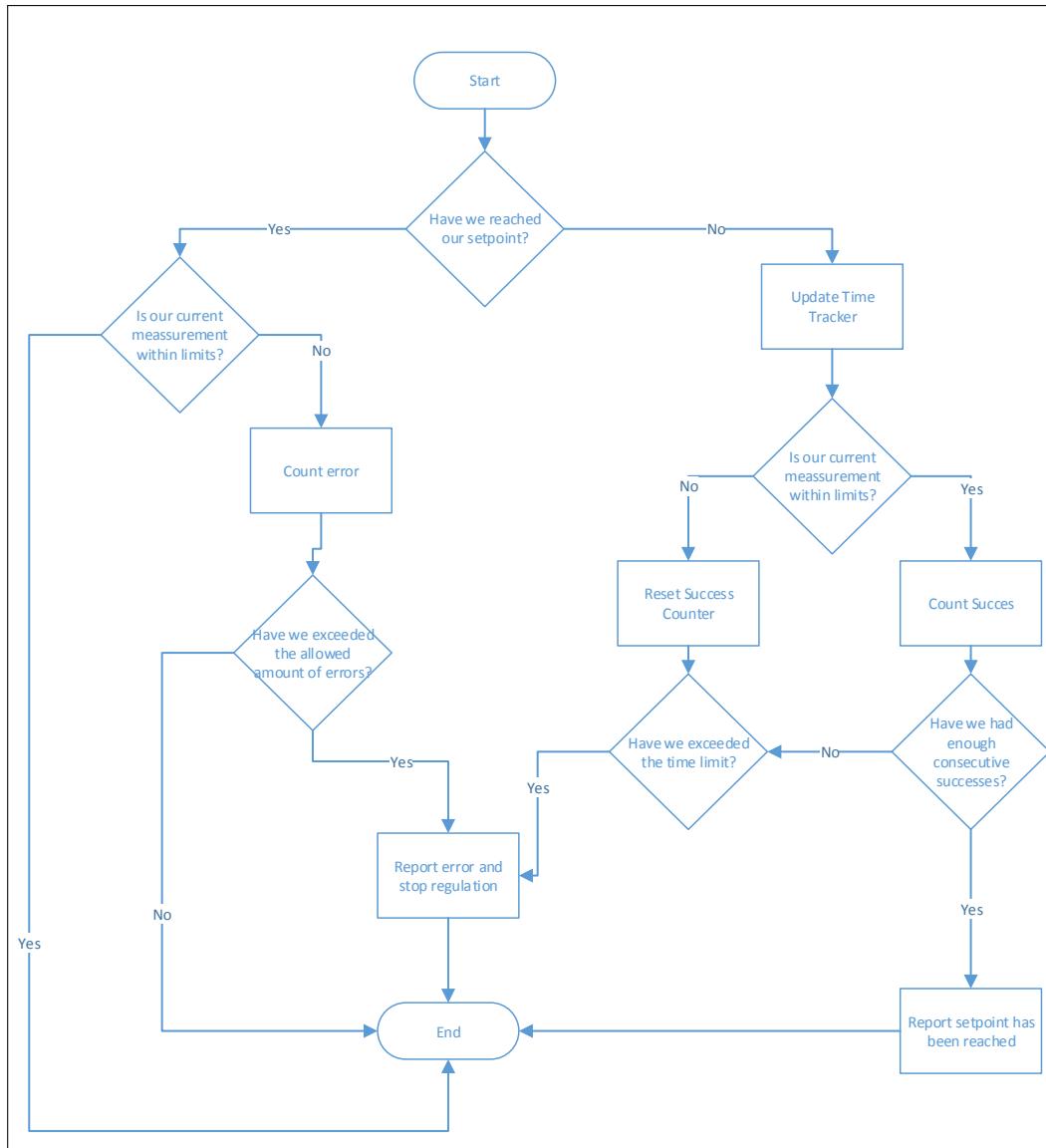
Figur 4.17. Flowdiagram for Humidity_ISR

Da systemet ikke har en aktiv mulighed for at affugte luften, implementeres rutinen ved at evaluere om **luftfugtigheden er større end setpunkt minus tilladte negativ afvigelse**. Denne afvigelse, angivet ved `#define LOWER_VAR`, gør det muligt for luftfugtigheden at falde og blive forøget inden den falder uden for de tilladte rammer. Dette kræver selvfølgelig at `LOWER_VAR` ikke bliver større end den tilladte afvigelse.

Stabiliteten evalueres ud fra de i kravspecifikationen angivne grænser.

Mængden af fugtighed, der afgives, reguleres ud fra toggling af `Humid_out` pinnen. Periodelængden for afgivelse af fugt bestemmes ud fra `#define PERIOD`, og angives i milisekunder. Størrelsen af denne afhænger af den fysiske enheds aktiveringstid, og skal kalibreres.

"Stability handler" funktionen fungerer som beskrevet i figur 4.18:



Figur 4.18. Flowdiagram for Stability Handler funktionaliteten

Som standard betragtes systemet værende "stabilit" hvis det temperaturen holdes indenfor de tilladte grænser i en periode af 1 minut. Når systemet har opnået stabilitet holdes øje med om målingerne begynder at ligge udenfor de tilladte grænser. Som standard er det ikke tilladt at værdierne falder udenfor den angivne grænse.

Til evt. at ændre disse regler er følgende defines lavet:

```

#define ERR_MARGIN - Den tilladte afvigelse, +/- 
#define SUCCES_LIMIT - Antallet af på hinanden efterfølgende succeser, der skal til før systemet betragtes stabilt
#define ERROR_LIMIT - Antallet af på hinanden efterfølgende fejl, der skal til før systemet betragtes som ustabil
  
```

Fælles for `#define SUCCES_LIMIT` og `#define ERROR_LIMIT` er, at den tidsmæssige perioden, de dækker over, beregnes ud fra Timerens periodetid multipliceret med disse tal.

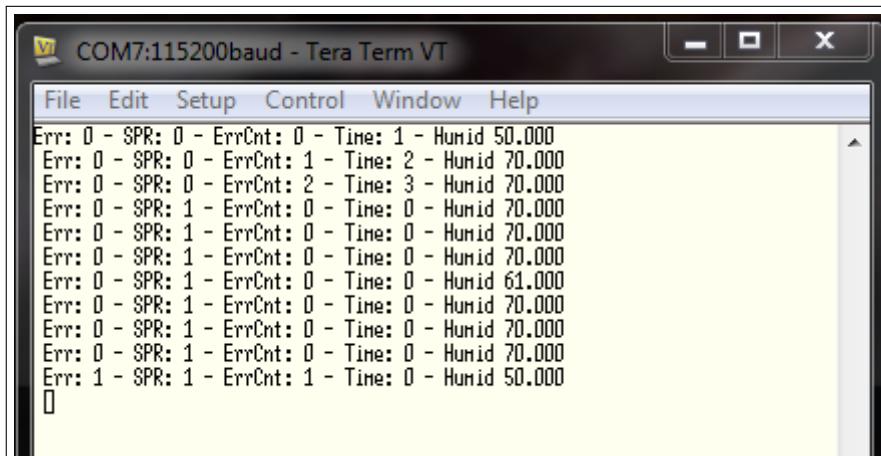
4.3.3 Test

Til test af softwareens egenskaber benyttes en driver til at give rutinen feedback mht. luftfugtighed. Til monitorering benyttes UART output. Test softwaren gør følgende:

1. Systemet initialiseres.
2. Systemet sættes til at holde en luftfugtighed på 70%; tidtagning starter.
3. Systemet får at vide at luftfugtigheden er 50%; systemet vil øge luftfugtigheden
4. Systemet får at vide, at luftfugtigheden er indenfor de tilladte afvigelser; dette gentages et antal gange
5. Systemet får at vide, at temperaturen er faldet tæt på grænsen; systemet vil øge luftfugtigheden.
6. Systemet får at vide at luftfugtigheden efter er inden for de tilladte afvigelser; dette gentages et antal gange.
7. Systemet får at vide at luftfugtigheden er uden for de tilladte rammer; systemet hejser error-flaget; reguleringen stopper.

Softwareen er komplet til at systemet skal holde luftfugtigheden 2 gange i træk for at være stabil, og systemet må ikke falde udenfor grænserne.

Resultat aflæst på UART'en:



The screenshot shows a terminal window titled "COM7:115200baud - Tera Term VT". The window has a menu bar with File, Edit, Setup, Control, Window, and Help. The main text area displays a series of log entries. Each entry consists of "Err: 0 - SPR: 0 - ErrCnt: 0 - Time: 1 - Humid 50.000", followed by a blank line. This pattern repeats 10 times. After the tenth entry, there is a single blank line, then a square bracket character "[".

```
Err: 0 - SPR: 0 - ErrCnt: 0 - Time: 1 - Humid 50.000
Err: 0 - SPR: 0 - ErrCnt: 1 - Time: 2 - Humid 70.000
Err: 0 - SPR: 0 - ErrCnt: 2 - Time: 3 - Humid 70.000
Err: 0 - SPR: 1 - ErrCnt: 0 - Time: 0 - Humid 70.000
Err: 0 - SPR: 1 - ErrCnt: 0 - Time: 0 - Humid 70.000
Err: 0 - SPR: 1 - ErrCnt: 0 - Time: 0 - Humid 70.000
Err: 0 - SPR: 1 - ErrCnt: 0 - Time: 0 - Humid 61.000
Err: 0 - SPR: 1 - ErrCnt: 0 - Time: 0 - Humid 70.000
Err: 0 - SPR: 1 - ErrCnt: 0 - Time: 0 - Humid 70.000
Err: 0 - SPR: 1 - ErrCnt: 0 - Time: 0 - Humid 70.000
Err: 0 - SPR: 1 - ErrCnt: 1 - Time: 0 - Humid 50.000
[
```

Figur 4.19. Resultat af befugtertest

Figur 4.19 viser testoutputet på formatet:

Om der er sket en fejl - Om Setpoint er nået - antallet af successer - tid - luftfugtigheden.

Det ses af testen, at luftfugtigheden starter på 50. Tidtagning startet. Luftfugtigheden stiger til 70. Efter 2 successer sættes SPR 1. Temperaturen falder til tæt på grænsen og systemet øger luftfugtigheden. Til sidst falder luftfugtigheden til under de tilladte grænser, Err flaget bliver hejst, og reguleringen stopper.

4.4 Stepmotor

4.4.1 Design

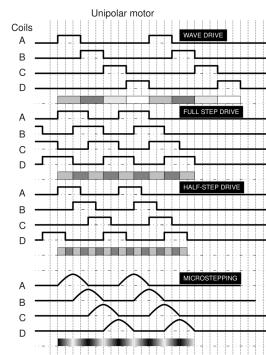
Stepmotorstyringen kan foregå på mange måder, den kan oprettes med "full-step", "half-step" og "micro-stepping". Der er også forskellige måder at styre hastigheden af stepmotoren på.

Ved full-stepping vil der være ét eller 2 ben aktivt af gangen, dette vil give den mindste præcision. Ved enkelt fases styring (1 aktivt ben af gangen) får man laveste strømforbrug til styring af motoren. Ved 2 fases styring (2 aktive ben af gangen) får man det største mulige moment med større strømforbrug. Dette er den foretrukne løsning til systemer der ikke har store krav til præcision. I et generelt tilfælde vil der være ca. 70% moment når der bruges enkelt fases styring i forhold til 2 fases styrings moment.

Half-stepping er hvor der skiftevis er ét og 2 ben aktivt af gangen, dette vil give det dobbelte antal intervaller og derfor vil hvert step være den halve vinkel. Dette giver en præcision der langt overgår vores behov til dette system.

Micro-stepping er hvor hvert af de 4 ben til stepmotoren bliver styret med en varierende RMS spænding, dette vil give en ekstrem høj stepinddeling og kan bruges til høj præcisionsstyring. Ved aeggevending er der ikke brug for sådan en præcision da kravet er et meget stort interval.

De 4 typer af step-styring kan ses illustreret på nedenstående billede (figur 4.20).



Figur 4.20. Illustration af de 4 step-styringsmetoder.

Tidsforsinkelsen mellem hvert step kan laves med en simpel wait funktion, sleep funktion eller steppet kan tages i et interrupt som bliver styret af en timer.

Wait eller Sleep kan bruges hvis der ikke er andre tidskritiske elementer i systemet. Da der er usikkerhed omkring antallet af tidskritiske elementer i projektet er denne idé fravalgt.

Efter disse overvejelser blev der besluttet at stepmotor controlleren skulle være en uni/bipolær stepmotor som styres med full stepping der bruger en timer til at lave delayet mellem hvert step, altså hastigheden på motoren.

4.4.2 Implementering

Der blev først oprettet et program til at teste step følgen, samt hastigheden af motorens rotation. Dette program blev oprettet med delays mellem hvert step. Først blev Funktionerne Step_Forward og Step_Backward oprettet.

```
Step_Forward(){
    if(register_read() & 0x1)
        register_write(0x2); //writing 0b0010
    else if(register_read() & 0x2)
        register_write(0x4); //writing 0b0100
    else if(register_read() & 0x4)
        register_write(0x8); //writing 0b1000
    else if(register_read() & 0x8)
        register_write(0x8); //writing 0b0001
}
```

```
Step_Backward(){
    if(register_read() & 0x1)
        register_write(0x8); //writing 0b0010
    else if(register_read() & 0x8)
        register_write(0x4); //writing 0b0100
    else if(register_read() & 0x4)
        register_write(0x2); //writing 0b1000
    else if(register_read() & 0x2)
        register_write(0x1); //writing 0b0001
}
```

Derefter blev der lavet en main til at teste disse følgen, programmet blev indstillet med en stepvinkel på 5.625° . Dette svarer til at der skal bruges 64 steps til en omgang af stepmotoren, start delayet blev sat til en tiendedel sekund så rækkefølgen kunne ses på de 4 leds der var til det tilhørende print ved motoren.

```
main(){
    while(){
        for(i=0; i<64; i++){
            Step_Forward();
            wait(100ms);
        }
        //Breakpoint here
        for(i=0; i<64; i++){
            Step_Forward();
            wait(100ms);
        }
        //Breakpoint here
    }
}
```

Denne test viste at rækkefølgen af steps var korrekt, men rotationen af motoren var alt for lav. Derfor blev antallet af steps sat op, og der blev fundet at der skulle tages 1024 steps

til en halv omgang, altså 16 gange så mange steps som tidligere antaget. Hvert step har derfor en vinkel på:

$$\frac{180^\circ}{1024\text{step}} \approx \frac{0.176^\circ}{\text{step}}$$

Tidsintervallet mellem steps blev testet med 5, 2 og 1 millisekund. Ved 1 millisekund kom der nogle sjældne afvigelser fra følgen, men 2 millisekunder gav ingen problemer, derfor vil der blive brugt ca. 2 millisekunder i stepmotor programmet.

Derefter skulle der opsættes en timer som kalder en ISR ved timecompare(tc) værdien, tc værdien kan nemt indstilles i PSoC3 C reator 3, der vises nemlig periodetiden på interruptbenet. Timeren bliver sat op til at bruge bus_clock som er på 24MHz, og derfor skal timeren være 16bit for at få en periode på de 2 millisekunder, og tc blev valgt til 50000 som giver perioden på 2.083 millisekunder.

Funktionerne til steps blev genbrugt med additionen at den nulstiller timerflaget til interrupt. ISR vil kalde den ene eller den anden step funktion efter et tjek på en variabel kaldet "direction". Derefter vil der tælles en variabel op som holder styr på antallet af steps, som også bliver tjekket for om der er foretaget en halv omgang med motoren, hvorefter den så vil skifte retning, genstarte antallet af steps taget og disable vores ISR. Dette vil så ende ud med at ved aktivering af ISR vil den være aktiv til en halv omgang er foretaget og næste gang den så bliver kaldt vil den køre en halv omgang den modsatte vej.

Det eneste styringen der skal til for at styre motoren i vores system, er altså an man skal aktivere en ISR for hver rotationstid, og initiering af timeren ved opstarten.

Ved den sidste test skete der en fejl ved rotationen hvor op til en tredjedel af steps gav fejl ved forward call, og halvdelen ved reverse. Disse fejl kunne være forekommel ved en ødelagt IC, men da der tidsmæssigt ikke var tid til implementering af ændringer, blev programmet revideret til en tidligere version, som var lavet med en enkelt faset styring, samt en tidsforsinkelse på 2.73ms som er den maksimale tc i den anvendte 16bit timer på bus_clock.

4.4.3 Test

Den afsluttende test af programmet blev udført med en simpel main funktion, denne initierede timeren og enablede globale interrupts. Derefter kørte en uendelig løkke der først enablede MotorISR og bagefter kørte et delay på 10 sekunder, dette var en stor buffer for at sikre der var tid til en halv rotation af stepermotoren før interruptet blev enablet igen. Programmet blev debugged med PSoC3 creatoren med et breakpoint efter hvert delay.

Resultatet af testen var at motoren gennemførte en halv omgang, og derefter ventede på det resterende delay på at blive færdigt. Den næste gang programmet blev sat til at køre, efter breakpointet, roterede motoren en halv omgang i den modsatte retning. Dette blev gjort 10 gange i træk uden problemer.

En sidste test inden samlingen til den fulde systemstyring gav afvigelser og en tidligere version blev genbrugt, denne blev testet 4 gange frem og tilbage hvor der ikke opstod afvigelser. afvigelsen kan skyldes at der skal længere tid til at lave hvert step ved 2 fases

styring, dette var ikke antaget et problem med ca. 40% ekstra tidsforsinkelse, men er taget op til overvejelse som en mulig fejlkilde.

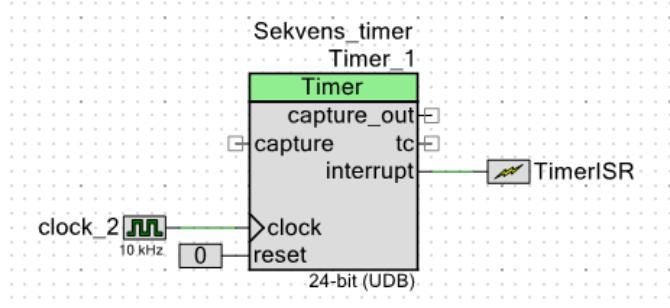
4.5 Sekvenstimer

4.5.1 Design

I figur 4.21 ses de tre komponenter i PSoC3 Creator, som skal bruges i sekvenstimeren. Vi har brug for en clock, en timer og en ISR.

For at kunne genere et interrupt hvert minut, skal timerens periode samt clock'ens frekvens passe sammen. Vi kan bruge 10 kHz i samspil med en periode på 600.000. Udregningen ses nedenfor.

$$600.000 / 10^3 = 60$$

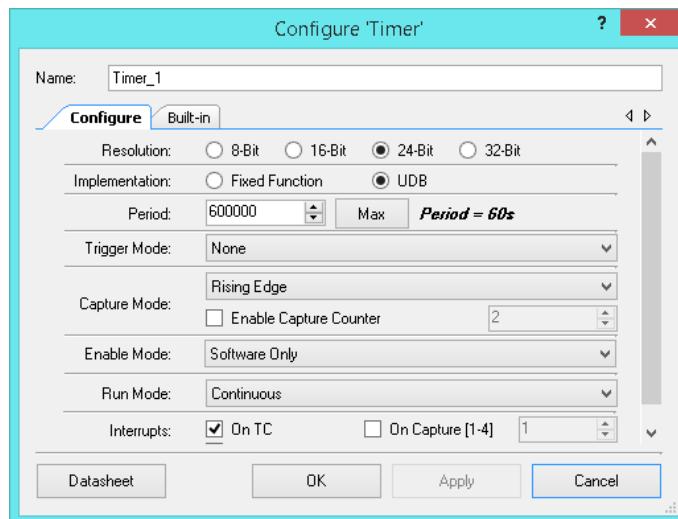


Figur 4.21. Sekvenstimer TopDesign

4.5.2 Implementering

Timeren sættes op til at være 24-bit, og perioden sættes til 600.000. Clocken sættes til 10 kHz. Derved vil vi få generet et interrupt hvert minut. Opsætningen fra PSoC3 Creator ses i figur 4.22.

Clock'en vi bruger har en nøjagtighed på $\pm 5\%$. Dette betyder at vi kan risikerer en afvigelse på over 24 timer i løbet af 21 dage, hvilket er uacceptabel i den virkelige verden. Men til vores prototype er nøjagtigheden acceptabel.



Figur 4.22. Sekvenstimer Indstillinger

I ISR-funktionen sættes en char (countFlag) til værdien 1, hver gang timeren genererer et interrupt. Man kunne også have valgt at optælle tællervariablen her direkte, men pga. den valgte håndtering af udrugningssekvensen (implementering i main) sætter vi blot et flag.

```
CY_ISR(isr_timer_sequence)
{
    countFlag = 1;
}
```

4.5.3 Test

Da vores formål med timeren i denne prototype ikke handler om nøjagtighed så meget som funktionalitet, blev det valgt at en stopurs-test til formålet. Timeren sattes til at toggle en LED hvert 60. sekund. Tiden fra toggle til toggle blev herefter målt fem gange. Testresultaterne ses i tabel 4.1.

Nr.	Resultat i sek.
1	60,2
2	60,1
3	60,1
4	60,2
5	60,1

Tabel 4.1. Test af sekvenstimer

Taget den menneskelige reaktionstid i betragtning, er ovenstående test en indikation på at vores timer virker som forventet.

4.6 PSoC SPI

4.6.1 Design

Der ønskes en kommunikation imellem PSoC3 og DevKit8000. Denne kommunikation skal foregå over SPI. Formålet med kommunikationen er, at brugeren af rugemaskinen skal kunne bruge et User Interface på DevKittet til at kommunikere med PSoC3. Brugeren skal kunne starte og stoppe udrugningen samt kunne se en status af temperatur, luftfugtighed og tid på skærmen.

Der ønskes implementeret en interrupt service rutine, der sættes til at vente på besked fra DevKittet. En besked fra DevKittet kan være en af følgende:

- En start besked.
- En slut besked.
- En status besked for temperatur.
- En status besked for luftfugtighed.
- En status besked for tid.

Nedenfor er specificering af, hvad de enkelte kommandoer indebærer:

Start Kommando: Start kommandoen skal udføres idet brugeren indikerer at udrugningssekvensen skal begynde. Når udrugningssekvensen sættes i gang, skal timeren sættes til at starte. Derefter skal *Temperaturen* og *Luftfugtigheden* sættes.

Slut Kommando: Slut kommandoen skal udføres idet brugeren indikerer, at alle emner er taget ud af rugemaskinen og at udrugningssekvensen dermed er færdig. Når udrugningssekvensen er færdig skal reguleringen af temperatur og luftfugtighed stoppes.

Status Kommando Temperatur: Sender status på temperatur til DevKit8000.

Status Kommando Luftfugtighed: Sender status på luftfugtighed til DevKit8000.

Status Kommando Tid: Sender status på tid til DevKit8000.

Protokol

Før implementeringen kan begynde, skal der udarbejdes en protokol, så DevKit8000 og PSoC3 er enige om, hvad der sendes imellem dem og hvordan det sendes.

Nedenstående tabeller, Tabel 4.2 og 4.3, viser en beskrivelse af hhv. adressen og kommandoen for beskederne samt det ønskede bitmønster.

Adresser (jvf. Minor nummer på DevKit8000):

Nr.	Beskrivelse	Bitmønster
1	Start eller Slut	0001
2	Temperatur	0010
3	Luftfugtighed	0011
4	Tid	0100

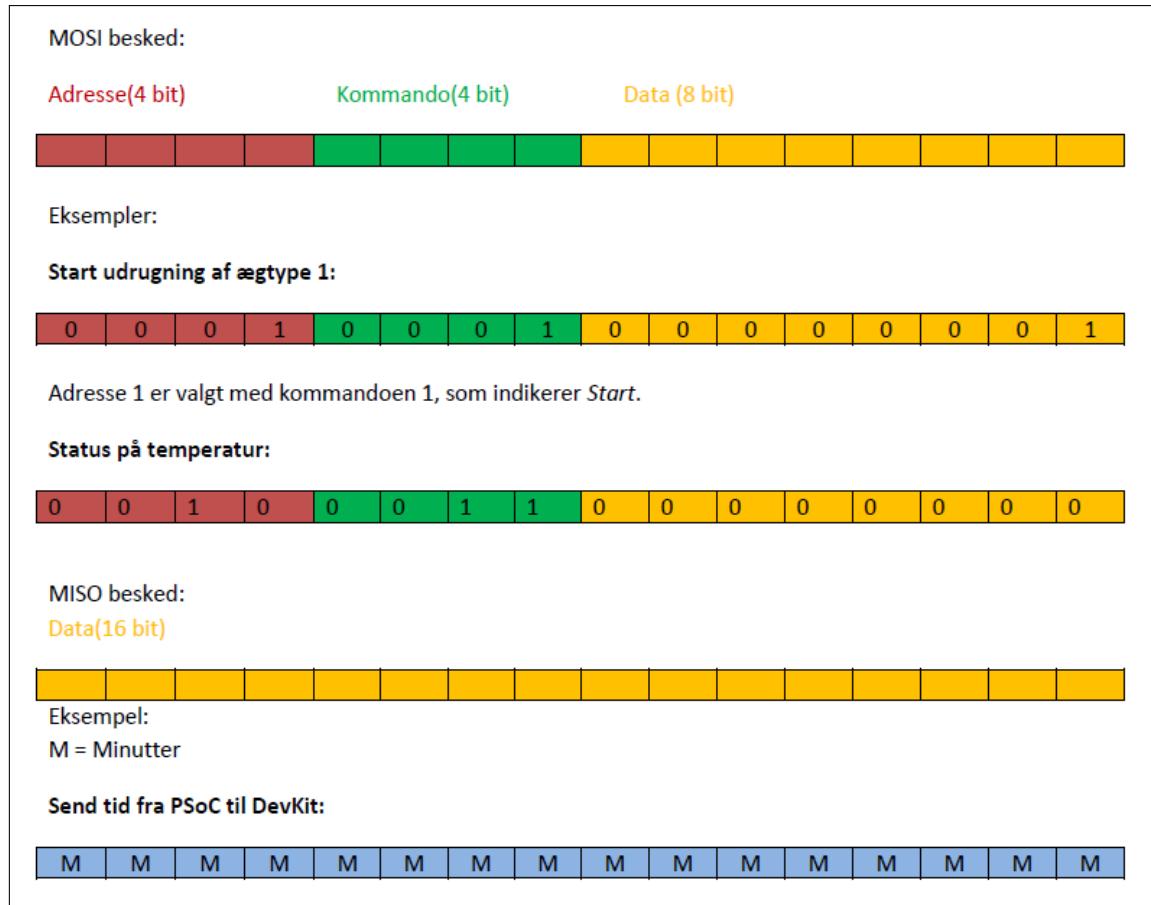
Tabel 4.2. Adresser

Kommando:

Nr.	Beskrivelse	Bitmønster
1	Start udrugningssekvens	0001
2	Slut udrugningssekvens	0010
3	Få status på temperatur	0011
4	Få status på luftfugtighed	0011
5	Få tid	0011

Tabel 4.3. Kommando

Figur 4.23 viser det ønskede bitmønster for beskederne. Som det kan ses, er der afsat 4 bit til adressen, 4 bit til kommandoen og 8 bit til dataen. Denne protokol gælder for MOSI (Master Out Slave In) beskederne. MISO (Master In Slave Out) beskederne indeholder 16 bit data. F.eks. vil dataen omhandlende tiden blive sendt i minutter. En konvertering til anden visningsformat af tid kan blive implementeret andetsteds.

*Figur 4.23.* Protokol

Følgende pseudokode bliver udarbejdet for kommunikationen mht. adresse og kommando.

```
/*SPI interrupt rutine
Wait for message from DevKit8000
switch(Message)
    case(Start)
        setTemp(normalTemp)
        setHumid(normalHumid)
        Start timer
    case(Stop)
        stopTemp()
        stopHumid()
        Stop timer
    case(Temperature status)
        Send temperature
    case(Humidity status)
        Send humidity
    case(Time status)
        Send time */
```

4.6.2 Implementering

Håndteringen blev implementeret i en interrupt service rutine, der står for dekodningen af data fra SPI.

Ifølge protokollen beskrevet i design afsnittet, bliver der sendt 16 bit fra devkit til PSoC3 via SPI. Adressen fylder de første 4 bit, derefter kommer kommandoen med 4 bit og til sidst kommer dataen med 8 bit.

Første skridt i implementeringen af SPI kommunikationen var at opsætte SPI i PSoC3 projektet. PSoC3 skulle sættes til at være en SPI slave, og den skulle sættes op til at kunne modtage beskeder fra DevKittet. Koden for opsætning af SPI:

```
/* Init SPI Slave */
SPIS_1_Start();
SPIS_1_EnableRxInt();
rx_isr_StartEx(isr_spi_rx);
SPIS_1_ClearFIFO();
SPIS_1_ClearTxBuffer();
```

Der blev oprettet en 16 bit variable 'rxvalue' hvori dataen fra DevKittet blev overført. Derefter blev der oprettet hhv. adresse variablen 'addr', kommando variablen 'cmd' og data variablen 'rddata'. Disse tre fik vha. bit-shifting overført det ønskede fra rxvalue. Koden for dette kan ses nedenfor.

```
/*Handle data received from DevKit8000*/
rxvalue = SPIS_1_ReadRxData();

addr = (rxvalue >> 12) & 0xf; // cmd = rdvalue[15:12]
cmd = (rxvalue >> 8) & 0xf; // addr = rdvalue[11:8]
rddata = rxvalue & 0xff; // data = rdvalue[7:0]
```

Gennemgang af switch-case:

Case 0x1:

Denne adresse er reserveret til Start/Stop kommandoen. Kommanden (cmd) bliver brugt til at skelne imellem Start eller Slut. Ifølge protokollen er kommandoen 0001 reservert til Start og kommandoen 0010 til Stop.

Denne håndertering af Start/Stop blev implementeret vha. en if-sætning. Hvis kommandoen Start registreres skal der ske følgende:

- Sæt temperatur til den ønskede temperatur på 37 grader.
- Sæt luftfugtighed til den ønskede luftfugtighed på 45.
- Nulstil tiden, countInMinutes.
- Start timeren.

Hvis Slut kommandoen derimod registeres skal følgende ske:

- Stop regulering af temperatur.
- Stop regulering af luftfugtighed.

- Stop timeren.

```

/*Switch case (addr)*/
/*Start and Stop case in switch case*/
case 0x1: //start or stop sequence
    if (cmd == 0x1) //command start
    {
        countInMinutes = 0u;
        Timer_1_Start();

        regTemp(CHICK_TEMP_NORMAL);
        regHum(CHICK_HUM_NORMAL);
    }

    else if (cmd == 0x2) //command stop
    {
        Timer_1_Stop();
        stopTemp();
        stopHum();
    }
    break;

```

Case 0x2-0x4:

Disse adresser er reservert til hhv. status på temperatur, på luftfugtighed og på tid. Hvis en af disse adresser registeres, skal der sendes data fra PSoC3 til DevKit8000. Denne data indeholder information om temperaturen, luftfugtigheden og tiden i det øjeblik kommandoer registeres. I denne del af koden blev der brugt de SPI specifiseret kommandoer i PSoC3 creator. Der ønskes en skrivning over SPI fra PSoC3 til DevKit8000. Til dette bruges kommandoen **SPIS_1_WriteTxData**.

Nedenfor ses et eksempel på en sending af temperatur status:

```

/*Switch case (addr)*/
/*Send status case in switch case*/
case 0x2: //status temperature
    if (cmd == 0x3) //command status
    {
        //clear buffer before sending
        SPIS_1_ClearTxBuffer();
        //send data temp from sensor to devkit
        SPIS_1_WriteTxData(getTemp());
    }
    break;

```

De efterfølgende statuser for luftfugtighed og tid er stort set ens med denne. Forskellen imellem dem er at der for at sende luftfugtighed bliver sendt data fra `getHumid()` og for tiden bliver variablen `countInMinutes` sendt.

`getTemp()` og `getHumid()` er begge metoder der kalder hhv. temperaturen og luftfugtigheden fra sensoren.

4.7 PSoC Main

4.7.1 Design

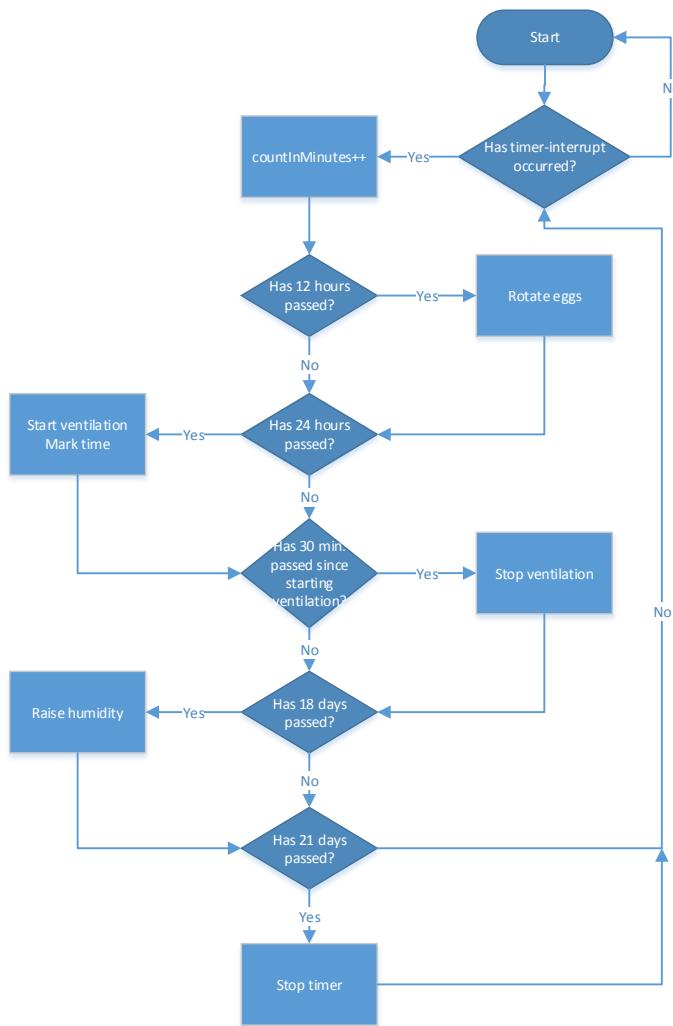
Der skal oprettes en PSoC applikation som fungerer som en main, der styrer alle funktionaliteter til rugemaskinen. Denne main skal stå for at kommunikationen imellem DevKit8000 og de forskellige enheder i rugemaskinen fungerer. I mainen bruges der en timer, der sættes i gang, når udrugningssekvensen sættes i gang. Denne timer er blevet beskrevet ovenfor i Timer - Udrugningssekvens. Selve håndteringen af tidsbegrænsningerne foregår i mainen.

Udrugningssekvensen er inspireret fra Hønsehus.dk³.

- Tiden nulstilles når udrugningssekvensen påbegyndes - Udrugningen tager 21 dage.
- Temperaturen sættes til 37 grader og luftfugtigheden sættes til 45.
- Hver 12 time skal æggene vendes.
- Hvert døgn skal det simuleres at hønen forlader æggene i 30 minutter. Dette gøres ved at sætte temperaturen ned til 20 i 30 minutter.
- Efter 18 dage skal luftfugtigheden sættes op til 75.
- Efter 21 dage er udrugningssekvensen slut. Timeren skal stoppes og devkittet informeres.

Forløbets flow er skitseret i nedenstående diagram, Figur 4.24:

³Fra Hønsehuset.dk[2]



Figur 4.24. Flowchart over Main forløb

Nedenfor ses det første udkast på mainen i form af pseudo kode. Der er taget udgangspunkt i ovenstående flowchart. Idéen med mainen er, at der er en overordnet håndtering af, om lågen på ruge maskinen er åben eller lukket. Det er meningen at udrugningssekvensen skal pauses, hvis lågen bliver åbnet.

```

/* Main */
/*
    If HATCH is NOT open
        If 12 hours have passed
            rotate
        If 24 hours have passed
            setTemp (lowTemp)
            set FLAG og current time
        If FLAG is set & time since = 30 minutes
            setTemp(normalTemp)
        If 18 days have passed
            setHumid(highHumid)
        If 21 days have passed
            Stop Timer
*/

```

Senere i forløbet blev der fundet en brist i ovenstående design. Fejlen bestod i, at der ikke var taget højde for, hvor mange gange i minuttet sekvensen bliver kørt igennem. F.eks. blev rotering kaldt flere gange i minuttet, når 12 timer var gået.

Denne brist gjorde at der ønskes tilføjet endnu et flag til koden, som indkapsler hele det ovenstående kode. Denne tilføjelse medfører, at man kun kommer ind i sekvensen en gang i minuttet.

Til selve koden er tidsbegrænsningerne omregnet til minutter:

- 12 timer = 720 minutter.
- 24 timer = 1440 minutter.
- 18 dage = 25920 minutter.
- 21 dage = 30240 minutter.

4.7.2 Implementering

Implementeringen af udrugningssekvensen foregik i en løkke i PSoC3 projektet, main.c. Da der blev taget udgangspunkt i pseudokoden, beskrevet i afsnittet ovenfor, ønskede mainen implementeret som en række betingelser.

Det første der skal håndteres i mainen er, at det skal sikres, at man kun går ind i sekvensen én gang i minuttet. Dette blev implementeret i form af et flag, countFlag. Dette flag bliver i Timeren sat til 1, når denne er talt én op. I selve mainen blev der derfor implementeret en betingelse i form af en if-sætning, der checker på om countFlag er 1. Hvis dette er opfyldt kommer man ind i sekvensen og countFlag sættes til 0. Samtidig bliver en variable, countInMinutes, tilføjet. Denne variable repræsenterer antal minutter gået, og den bliver talt op, når den ovenstående betingelse om countFlag er opfyldt.

For at håndtere tidsbegrensningerne, der også står beskrevet ovenfor, skulle der udtænkes en metode, hvorpå man kunne checke om et bestemt antal minutter var gået. Det blev valg at bruge regnemetoden 'modulus' til dette. Da countInMinutes holder indeholder tiden der er gået og da der findes faste tidsintervaller for sekvensen, kan mouduls bruges.

Eksempel:

```
if(countInMinutes % 720 == 0)
```

Denne if-sætning checker på, om der er gået 720 minutter.

Til implementeringen af simuleringen af at hønen forlader reden blev der tilføjet et flag kalden henFlag. I if-sætningen der håndteres hvert døgn (if(countInMinutes % 1440 == 0)) bliver der dette flag sat til 1 samtidig med at en variable, 'timeStamp' sættes til den aktuelle værdi af countInMinutes. Samtidig sættes temperaturen til en lavere temperatur:

```
if ((countInMinutes % 1440) == 0)
{
    regTemp(CHICK_TEMP_LOW);

    henFlag = 1;
    timeStamp = countInMinutes;

    //UART print used for testing
}
```

Flaget og timeStamp variablen bruges til at håndtere nedkølingen når hønen forlader reden. For at regulere temperaturen til den normale temperatur igen, efter 30 minutter blev følgende if-sætning implementeret:

```
if (henFlag == 1 && (countInMinutes - timeStamp) == 30)
{
    henFlag = 0;
    regTemp(CHICK_TEMP_NORMAL);

    //UART print used for testing
}
```

Denne if-sætning holder øje med om der er gået 30 minutter siden flaget henFlag er sat. Dette gøres vha. timeStamp.

I mainen skal der også modtages data fra sensoren angående temperatur og luftfugtighed. For at loade dataen fra sensoren blev følgende kode tilføjet til mainen. Denne kode blev lagt ind *før* selve udrugningssekvensen.

```
Measure_HIH61xx(); //Measure temperature and humidity
CyDelay(500);
Read_HIH61xx(); //Read temp and humid
CyDelay(500);
```

4.7.3 Test

For at teste mainen for at se om forløbet blev løbet igennem korrekt, blev der internaliseret en UART. Ved at lave en sprintf i for hver ønskede handling og udskrive denne på UART'en kunne forløbbet simuleres. Mainen blev testet sammen med timeren, hvor timerens periodetid blev sat ned. Dette muliggjorde at forløbet kunne løbes igennem over en kort periode.

Figur 4.25 viser **Test af main**.

```
Rotate. Count == 720
Rotate. Count == 1440
Udluftning. Count == 1440
Udluftning slut. Count == 1470
Rotate. Count == 2160
Rotate. Count == 2880
Udluftning. Count == 2880
Udluftning slut. Count == 2910
Rotate. Count == 3600
Rotate. Count == 4320
Udluftning. Count == 4320
Udluftning slut. Count == 4350
Rotate. Count == 5040
Rotate. Count == 5760
Udluftning. Count == 5760
Udluftning slut. Count == 5790
Rotate. Count == 6480
Rotate. Count == 7200
Udluftning. Count == 7200
Udluftning slut. Count == 7230
```

Figur 4.25. Test af Main

Som det ses på billedet er første begivenhed i forløbet en rotering af æggene. Denne rotering forekommer efter 720 minutter (12 timer) og igen efter 1440 minutter (24 timer). Samtidig med at æggene roteres anden gang, simuleres udluftning og 30 minutter senere (1470 minutter) afsluttes udluftningen. Denne sekvens gentages indtil 21 dage er gået.

4.8 DevKit8000 Driver

4.8.1 Design

At lave software til devkit 8000 afviger i stor grad fra det software, der er skrevet på tidligere semestre til ATmega32 microcontrolleren. Dette bunder i at DevKit8000 er et embeddet system, men med et OS. Dette giver et højere abstraktionsniveau, når der skal tilgå eksterne enheder(SPI, interrupt, GPIO osv.). Dette gøres igennem et driverlag og ikke ved at tilgå pins direkte. Driveren står så for at fx gå ned og læse en pin, når en applikation henvender sig til drivere. Applikationerne tilgår så driveren igennem driverens node filer.

Driveren der skrives til DevKit8000 har brug for følgende funktionalitet:

- Skal kunne kommunikere over SPI protocol til Psoc'en.
- Skal kunne modtage interrupt fra lågen.

Til SPI Driveren er der allerede udarbejdet et skelet i faget I3HAL[1], hvor der er lavet en driver, som kan kommunikere over SPI. Driveren mangler dog noget funktionalitet, som ønskes tilføjet. Til SPI delen ønskes der tilføjet:

- Tilpasning til den tidlige definerede protocol i projektet.
- Automatisk oprettelse af node filer ved at tilføje oprettelse af driver class og driver device.

Dette kunne også gøres via et script. Det anses dog som en langt mere funktionel løsning at driveren selv står for oprettelse.

Envidre ønskes der at laves en interrupt linie i driveren. Interrupt linien skal registrere ændringer fra lågen til rugemaskinen. Derved bruges interruptet til at kunne give data til en applikation, men kun når der registreres en ændring i interruptet.

For at dette kan realiseres skal der i driveren tilføjes:

- Tilføjes en interrupt line, GPIO + interrupt.
- Der skal ændres i read funktionen (funktionen der bruges til at tilgå driveren) så den kan bruges til både spi, samt interrupt.

Designmæssigt vælges det at begge funktionaliteter samles i en driver frem for at splitte det op i to forskellige drivere. Dette vælges, da det ikke anses for nødvendigt, at de to drivere skal kunne benyttes uafhængigt af hinanden.

4.8.2 Implementering

For at lave driveren til devkit8000 blev er taget udgangspunkt i driveren, som er udarbejdet i forbindelse med I3HAL. Driveren der er skrevet der har meget basale funktionaliteter til at kunne kommunikere til PSoC3 over SPI kommunikation. Der er dog flere ting der skal tilføjes og ændres for at få den til at virke som ønsket i projektet.

En af de store forskelle på at til gå eksterne enheder på et embeddet system er at der oftest hentes data igennem drivere istedet for at til gå GPIO pins direkte. Dette gøres i et linux baseret system som det der ligger på devkittet igemmen filer. I den inhentede driver er det brugeren selv der står for at oprette disse efter driveren er indsatt i kernen. Dette ønskes til projektet gjort automatisk, ved indsættelse af driveren.

For at gøre dette i linux, opretter man først et objekt af typen "class", og efterfølgende for hver af de filer der ønskes oprettes et objekt af typen "device". For at tilpasse driveren til den ønskede protocol, forenkles driveren tilgangen ved at lave tre filer til spi, som bliver "read only". Hver af disse filer skal kunne bruges som en "getTemp", "getHumid" og "getTime", hvor den nødvendige protocol for at forspørge data fra PSoC3 allerede er kodet ind i driveren. Derud over bliver der en fil som er "write only" som bruges til at sende information til PSoC3 som ikke involvere at hente temperatur, luftfugtighed eller tid tilbage. Endvidre blev der tilføjet en enkelt fil til at kunne indlæse interrupts fra lågen.

Følgende kode blev derfor tilføjet til driveren:

```
//Declare device and class
static struct device* psoc_device[MINOR_NUM];
static struct class* psoc_class;

//In init function
//Create class
psoc_class = class_create(THIS_MODULE, "PSoC3");

//create devices (acces files)
for ( i = 0; i < MINOR_NUM-1; i++)
    psoc_device[i] = device_create(psoc_class, NULL, MK_DEVNO,
                                   NULL, "psoc%d", i);
interrupt_device = device_create(psoc_class, NULL, MKDEV(MAJOR(devno),4),
                                 NULL, "psoc_interrupt%d",0);
```

Ligeledes blev der i exit funktionen tilføjet kode til at rydde op igen.

For at kunne styre hvilke filer, der blev brugt til at tilgå hvad, blev der i read og write funktionerne lævet en "Switch case", som kigger på driverens minor nummer(filen der bliver tilgået), som man kan se i koden for read funktionen:

```
//Read funktion
minor = MINOR(filep->f_dentry->d_inode->i_rdev);
switch (minor)
{
    case 1:
        if(psoc_spi_read_reg16(0x02,&result) < 0)
            return -1;
        break;
    case 2:
        if(psoc_spi_read_reg16(0x03,&result) < 0)
            return -1;
        break;
    case 3:
        if(psoc_spi_read_reg16(0x04,&result) < 0)
            return -1;
        break;
    case 4:
        wait_event_interruptible(interruptlineQ, (flag == 1));
        result = gpio_get_value(GPIO_KEY);
        flag = 0;
        break;
    default: return -1;
}
```

Dette sikre, at hvis man henvender sig til filen, som giver minor nummer 1, går programmet ud og læser på SPI kommunikationen ved at henvende sig til adresse 2(0x02). Det samme er tilfældet ved case 2 og case 3, som begge også håndtere læsning via SPI protokollen. Derimod står case 4 for at håndtere læsning på interruptet. Interruptet er sat op til at læse på en GPIO og køre interrupt servicerutinen ved både rising og falling edge. Funktionen wait_event_interruptible sikre, at der ved læsning kun bliver retuneret en værdi, når der bliver registreret et interrupt. Dette sikre mod at et program står og poller for at få adgang til data ved ændringer på interrupt linien.

Samme princip blev benyttet i write funktionen. Det er dog kun minor nr. 0, som kan tilgå write funktionen, da det er den, der står for at sende data fra DevKit8000 til PSOC3.

I både read og write funktionen anvendes en SPI funktion til at sætte beskederne, der skal sendes og modtages, op. Den data, der sendes, bliver sat op efter protokollen, der er blevet fastlagt til projektet. SPI read og write funktionen står så for at pakke den data ind i en SPI besked, som sendes ud via SPI kommunikationen. Når der skal sendes data, gøres dette ved 8 bit data uddover adressen og kommandoen. Fra psoc_spi_write_8bit:

```
//addr er adressen.
//cmd er kommandoen
//data er den data der skal sendes.
// Det hele bliver samlet til en besked, som stemmer overens
//med protokollen.
msg = (addr << 12) | ((cmd & 0xF) << 8) | (data & 0xFF) ;

/* Init Message */
memset(t, 0, sizeof(t));
spi_message_init(&m);
m.spi = psoc_spi_device;

//Sender beskeden via transmit bufferen.
t[0].tx_buf = &msg;
t[0].rx_buf = NULL;
t[0].len = 2; //antal bytes der skal sendes.
spi_message_add_tail(&t[0], &m);
```

Det eneste, der skal tages hånd om, når driveren anvendes, er, at driver modulet skal indsættes i kernen, hotplug modulet skal indsættes og der skal ændres i den allerede eksisterende cpld driver. Til dette blev der oprettet et script:

```
insmod psocmod.ko
insmod hotplug\psoc\spi\_device.ko

echo 0x3 > /sys/class/cplddrv/cpld/spi\_route\_reg
echo 0x1 > /sys/class/cplddrv/cpld/ext\serial\_if\route\_reg
```

Dette gav mulighed for nemt at kunne indsætte driver modulerne og indstille cpld driveren, så DevKit8000 var klar til at benytte SPI kommunikationen.

4.8.3 Test

For at teste DevKit8000 driveren, blev der skrevet et lille testprogram til PSoC3. Programmet modtog beskeder over SPI kommunikationen. Programmet blev tilpasset til protokollen, så der var en adressen, hvor der kunne sendes data til PSoC3 og tre adresser, hvor PSoC3 skulle returnere værdier. PSoC3 returnerede blot hard codede værdier, som var forskellige for hver adresse. Når der blev sendt data til PSoC3, blev de otte LED'er på PSoC3 anvendt til at se hvilken 8bit data, der blev modtaget.

På DevKit8000 siden blev programmet cat så brugt til at se, om der kunne læses data på de forskellige adresser samt echo'et værdier ind på sendedelen.

Ud fra testen kunne der konstateres, at SPI driveren virkede som ønsket. Der blev modtaget de rigtige data, og PSoC3 modtog de data, DevKit8000 sendte.

Interrupt linien blev testet med GUI'et.

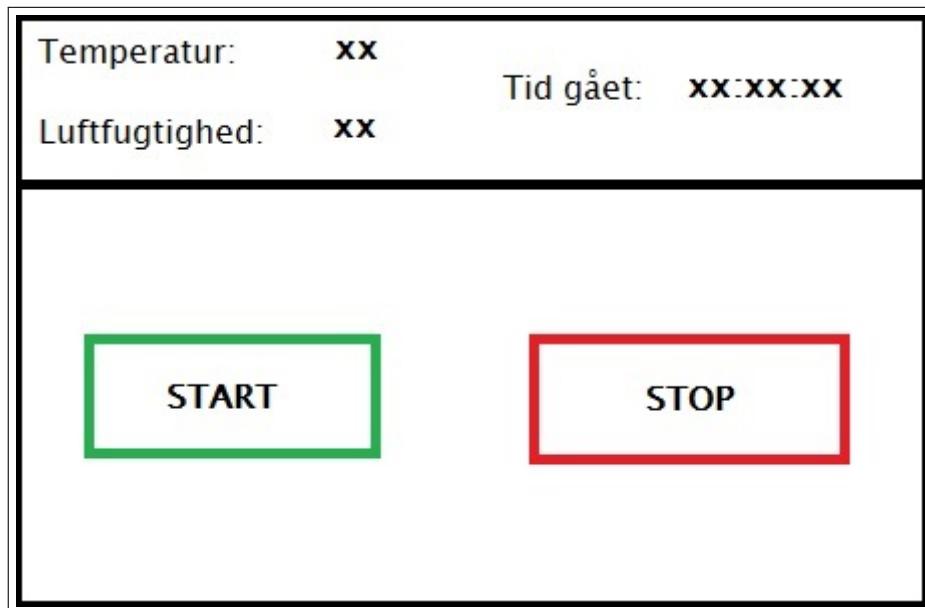
4.9 DevKit8000 GUI applikation

4.9.1 Design

Til den grafiske burgerflade ønskes der et meget simpelt design. Dette skyldes, at systemet, der laves, er en rugemaskine. Derved ses det ikke som væsentligt at have en flot og fancy brugerflade. Derimod lægges der vægt på at have en simpel og funktionel grafisk brugerflade. Den grafiske brugerflade skal have følgende funktionaliteter:

- Skal kunne tænde og slukke for udrugnings maskinen,
- Skal kunne vise tiden der er passeret, temperatur og luftfugtighed.
- Skal give besked til brugerne, når lågen er åben.

Der blev i første omgang lavet et designudkast til, hvordan den grafiske brugerfladen skulle se ud:



Figur 4.26. Design layout af GUI

Dette gav et basisudkast at arbejde ud fra til implementationsfacen og på trods af et relativt simpelt design, er der flere ting, der skal kunne sættes sammen, for at den grafiske brugerflade kan fungere som ønsket.

4.9.2 Implementering

Til at lave GUI applikationen blev der anvendt Qt frameworket. Dette blev valgt, da det er et relativt simpelt framework at benytte, det er baseret på c++, som vi har kendskab til, og DevKit8000, hvor applikationen skal køre, har Qt biblioteket installeret som standart i opsætningen, vi får. Det blev derfor anset som det nemmeste at anvende.

Idet implementeringen af GUI applikationen samtidigt var en del af læringsprocessen i Qt, blev implementeringen meget metodisk.

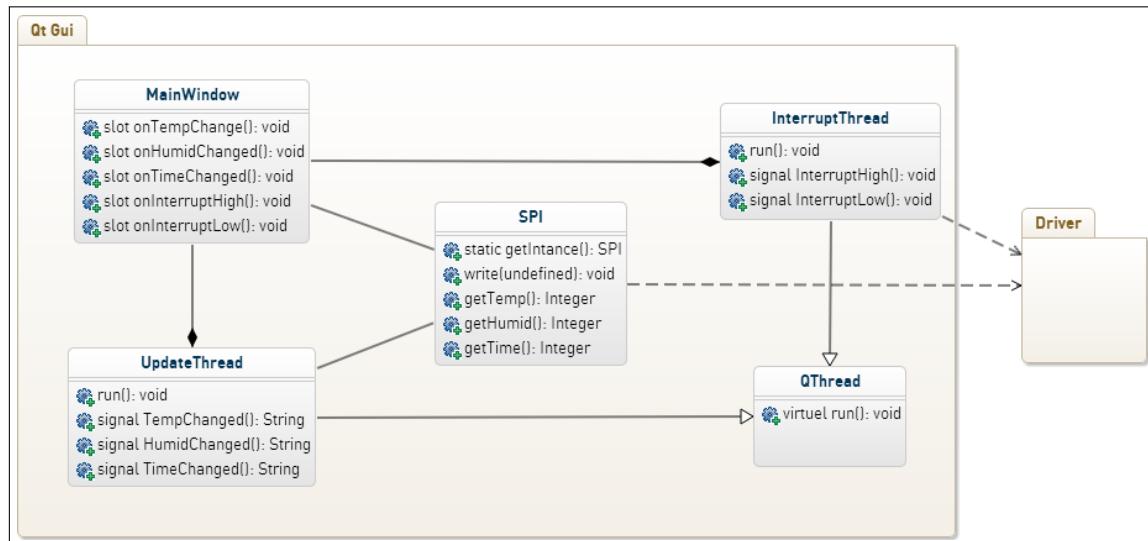
For at give et godt udgangspunkt at arbejde ud fra, blev den grafiske del af programmet først implementeret. Qt brygger selve den grafiske del ud fra et simpelt HTML lag. Selv med mindre kendskab er dette dog simpelt at anvende, da det giver mulighed for at benytte simple ting som "drag and drop" med knapper, lables osv. Det første grafiske design, der forekom, var meget tæt på det endelige. Eneste ændringer, der blev lavet, var at tilføje farver til de to knapper, som det ses på figur 4.27, som er det endelige grafiske design.



Figur 4.27. GUI designet, implementeret i Qt.

I designfasen blev den grafiske del holdt meget simpelt for at gøre det simpelt at implementere. Der blev derfor hurtigt lagt vægt på at få implementeret den underlæggende software til applikationen.

Der blev i første omgang forsøgt med bare at starte fra en ende og begynde at implementere software. Der blev dog hurtigt konstateret, at med de mange elementer, der skulle laves i softwaren, var det nødvendigt med noget overlæggende design for at give overblik. Der blev derfor udarbejdet et UML klassediagram for at give et overblik over de forskellige klasser, der skulle laves, deres relationer, samt nedarvningshierarki. Ligeledes blev det brugt til at give overblik over de forskellige metoder, der skulle laves under de forskellige klasser:



Figur 4.28. Klasse diagram over GUI applikationen.

For at lave en grafisk applikation, som fungerer så flydende som muligt, blev der anvendt flertrådet programmering. Det blev valgt, at der blev benyttet en "main"-tråd til at håndtere selve den grafiske, eventorienterede programmeringsdel. Der blev dertil lavet to datatråde, som hver især står for at skulle håndtere data og sende til "main"-tråden, der så står for at håndtere det grafiske, som fx at vise data'en, der blev hentet på SPI kommunikationen. De to tråde blev lavet som klasserne UpdateThread og InterruptThread og blev, ud fra Qt dokumentationen omkring flertrådet programmering, lavet som nedarvninger fra klassen QThread[3].

MainWindow: MainWindow er klassen, der står for at håndtere selve "main"-vinduet. Klassen har nogle metoder, der håndterer de to trykknapper samt nogle forskellige slot's. De forskellige slot's anvendes til at lave kommunikationen fra de to datatråde og giver dem et slot at sendes deres signals hen i. Derved kan MainWindow reagere på den data, der bliver modtaget, når fx Interrupt tråden emitter signalet InterruptHigh().

MainWindow står for at oprette de to datatråde InterruptThread og UpdateThread og har kendskab til SPI klassen. SPI klassen benytter den, når der skal sendes et start- eller stopsignal ud på SPI kommunikationen. Dette sker ved tryk på henholdsvis Start- og Stopknapperne på GUI'et.

De tre slots, onTempChanged, onHumidChanged og onTimeChanged, benyttes, når UpdateThread har været ude og hente information via SPI kommunikationen. Den sendte QString fra UpdateThread bliver derefter vist på den tilhørende lable.

SPI: SPI klassen er en klasse, der er tilpasset til at tage sig af kommunikationen til PSoC3 via den udarbejde driver. SPI klassen opretter derfor tre c++ file streams til at gå ud og læse på de tre driver nodes med temperatur, luftfugtighed og tid, og returnere dem ved kald på getTemp, getHumid og getTime. Til at håndtere skrivningen over SPI kommunikationen blev der anvendt et C file handle. Dette blev anvendt frem for C++ fil behandlingen, da der er større kontrol over buffere, når der anvendes almindelig C, hvilket er vigtigt, når der kommunikeres ud til en driver og videre ud til hardware. Da der er flere tråde, der skal kunne anvende SPI klassen, blev den givet en mutex, som den i hver er get og write metoderne låser i starten og frigiver i slutningen. For at sikre at trådsynkroniseringen bliver holdt, er det nødvendigt, at der kun benyttes en fælles instans af klassen af de andre klasser. Det blev gjort via metoden getInstance. Selve SPI klassens constructor er gjort privat, hvilket gør, at der ikke er mulighed for at oprette en instans af klassen. Den kan dog tilgås ved metoden getInstance:

```
SPI& SPI::getInstance()
{
    static SPI spi;
    return spi;
}
```

Idet metoden er erklæret static, kan metoden tilgås, selvom der ikke er oprette en instans af klassen. Første gang metoden så kaldes, opretter den en static instans af SPI klassen, som den så returnere en reference af. Dette sikre, at når metoden kaldes næste gang, er instansen allerede oprettet. Dette er generelt ikke en god idé, men da der er tale om en klasse, der kommunikere med en driver, er den en god metode for at sikre, at der ikke bliver lavet flere henvendelser til driveren, end hvad den kan håndtere.

UpdataThread: Klassen UpdataThread bruges til at gå ud og hente: temperatur, luftfugtighed og tid, via SPI klassen. UpdateThread står herefter for at lave de hentede integer værdier om til Qstrings, som kan skrives i Qlables, samt at lave den nødvendige formatering på tiden, som via spi modtages som antal minutter. Denne formateres om til Dage:Timer:minuter. Når de forskellige Qstrings er klar, emitteres de forskellige signaler til MainWindow, som så viser data'en i de forskellige lables.

InterruptThread: InterruptThread står for at læse på interrupt linien i driveren. Da driveren er lavet med interrupt synkronisering, kan InterruptThread kun læse data, når der er sket ændringer på interrupt linien. Den værdi, som driveren så læser på GPIO'en, bruger InterruptThread til at vurdere, om der skal emitteres et signal med interruptHigh eller interruptLow. De to signaler modtages så at MainWindow. Hvis der sendes en interruptHigh, vises der en Qmessagebox, der informerer brugeren om, at lågen er åben. InterruptLow står så for at lukke denne Qmessagebox igen.

4.9.3 Test

For at teste blev der anvendt samme setup, som der blev brugt ved test af driveren. Denne gang blev der dog, istedet for at benytte echo og cat til at læse/skrive fra/til filerne, brugt GUI applikationen. Det blev derved testet, at GUI driverens interaktion virkede. Der blev ligeledes testet, om interrupt linien virkede ved at anvende Devkittets boot_key.

Generelt blev flere forskellige senarerier testet på GUI'et, fx at trykke flere steder samtidigt. Der manglede dog en konkret måde at teste den grafiske brugerflade.

Accepttest 5

Da vi ikke formåede at sammensætte og teste systemet i dets helhed, kan vi ikke godkende nedenstående accepttests. Vi har dog gennemført og godkendt flere af testene i et "subset" af systemet, som inkluderede sammensatte moduler.

5.1 Accepttest for Use Case 1

5.1.1 Hovedscenarie

Krav	Udførelse	Forventet resultat	Resultat
Ved lågens åbning låses UI. Det forbliver låst indtil lågen lukkes.	Maskinen står i idle tilstand og lågen åbnes	UI låser	
	Lågen lukkes igen	UI låser op	

5.2 Accepttest for Use Case 2

5.2.1 Hovedscenarie

Krav	Udførelse	Forventet resultat	Resultat
Systemet skal kunne regulere og holde en temperatur indenfor grænserne angivet i ikke funktionelle krav.	Systemet opstilles i omgivelser der ligger indenfor de påkrævede rammer. Systemet indstilles til at skulle holde 37°C. Systemet sættes i gang og temperaturen aflæses efter 10 minutter.	Maskinen kan regulere og holde temperaturen indenfor de angivne rammer.	

<p>Systemet skal kunne regulere og holde en luftfugtighed indenfor grænserne angivet i ikke funktionelle krav.</p>	<p>Systemet opstilles i omgivelser der ligger indenfor påkrævede rammer. Systemet indstilles til at skulle holde 70% relativ luftfugtighed. Systemet sættes i gang og luftfugtigheden aflæses efter 10 minutter.</p>	<p>Maskinen kan regulere og holde luftfugtigheden indenfor de angivne rammer.</p>	
<p>Maskinen kan vende æg med et bestemt interval. Æggene roteres indenfor de i ikke funktionelle krav specificerede grænser.</p>	<p>Æg-type specificeres, og systemet indstilles til at skulle vende æggen hvert 5. minut i en periode på 30 minutter. Æggernes vertikale akse markeres med en pil, der peger op. Systemet sættes i gang og kører i 30 minutter. Ved hver vending måles rotation af hvert æg, og deres rotation noteres. Efter hver rotation vendes hvert æg manuelt så orienteringsindikatoren (pilen) peger opad.</p>	<p>Maskinen roterer alle æg indenfor den angivne grænse.</p>	
<p>Systemet skal ved afsluttet udrugning informere brugeren om dette.</p>	<p>Maskine indstilles til et 5 minutters udrugningsprogram og igangsættes. Det observeres at systemet informerer brugeren når sekvensen afsluttes.</p>	<p>Maskinen vil ved testsekvensens afslutning informere brugeren.</p>	

5.2.2 Undtagelser

Krav	Handling	Forventet resultat	Resultat
Ved lågens åbning låses UI og udrugningssekvensens afbrydes midlertidigt. UI forbliver låst indtil lågen lukkes. Når lågen lukkes genoptages udrugningssekvensen.	Maskinen står I udrug-tilstand og lågen åbnes.	UI låser og udrugningssekvensen stoppes.	
	Lågen lukkes igen.	UI låser op og udrygningssekvensen genoptages.	
Hvis det ikke er muligt at overholde temperaturen skal systemet informere brugeren.	Maskine opstilles i et lokale med temperatur indenfor de angivne grænser for opstilningsmiljø. Systemet indstilles til at skulle holde temperaturen på 40°C, varmelegemet frakobles, og systemet aktiveres.	Maskinen informerer brugeren.	

Hvis det ikke er muligt at overholde luftfugtigheden skal systemet informere brugerne.	Maskine opstilles i et lokale med luftfugtighed indenfor de angivne grænser for opstilningsmiljø. Systemet indstilles til at skulle holde luftfugtigheden på 70% relativ luftfugtighed, luftfugtighedsreguleringsmekanismen frakobles, og systemet aktiveres.	Maskinen informerer brugerne.	
--	---	-------------------------------	--

Litteratur

- [1] I3hal, exercise7 - ldd spi, gruppe: julie.
https://redmine.ihb.dk/courses/projects/i3hal_f2014_julie/wiki/Exercise7_-_LDD_SPI [Kan tilgås gennem IHA VPN. Sidst besøgt 26.05.2014].
- [2] Rugetips til udrugning. <http://www.xn-hnsehus-q1a.dk/opdraet/rugetips/> [Sidst besøgt 26.05.2014].
- [3] Starting threads with qthread. <http://qt-project.org/doc/qt-4.8/threads-starting.html> [Sidst besøgt 26.05.2014].
- [4] Fairchild Semiconductor Corporation. *1N5817-1N5819*, 2001.
- [5] Honeywell. I2c_comms_humidicon_tn_009061-2-en_final_07jun12.pdf. Technical report, Honeywell, 2012.
- [6] Honeywell. honeywell-sensing-humidicon-hih6100-series-product-sheet-009059-6-en.pdf. Technical report, Honeywell, 2013.
- [7] Kiatronics. *28BYJ-48 – 5V Stepper Motor*.
- [8] STMicroelectronics. *ULN2001, ULN2002, ULN2003, ULN2004*, 2002.
- [9] TE Connectivity. *Aluminium Housed Power Resistors*, 2012.