

Projektrapport 3. semester

E3PRJ3 - GRUPPE 08

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

Resumé

Der er i forbindelse med semester projektet forsøgt at realisere en rugemaskine, i første omgang med udrugning af hønseæg i fokus.

I projektets startforløb, er der anvendt forskellige UML og SysML værktøjer til at give overblik over projektet, og der er i denne periode anvendt en form for V-model som arbejds metode. I projektes design og implementerings fase er der forsøgt at anvende arbejds metoden Scrum.

Der er i projektet blevet anvendt to systemer, den ene, en microcontroller af typen PSoC3, og den anden, et embedded system, DevKit8000. De to systemer har i projektet skulle kommunikere med hinanden via SPI.

Til PSoC3 er der primært fokuseret på at anvende forskellige I2C sensorer, reguleringssløjfer samt kontrollere DC- og step-motorer. Disse forskellige komponenter er der blevet arbejdet med, både hardware of software mæssigt. Softwaren der er skrevet til PSoC3, er skevet i programmerings sproget C, og der er anvendt programmet PSoC Creator.

Det embeddede system, DevKit8000 et sat op med et Linux baseret operativ system. Til DevKit8000 er der udarbejdet en driver, i form af et Linux kerne modul, til at håndtere SPI kommunikationen til PSoC3, samt en interruptlinie. Derudover er der blevet lavet en GUI applikation, som har skulle benytte den førnævnte driver til at vise/sende data fra/til PSoC3. Applikationen er skrevet i C++ og anvender Qt frameworket.

Projektet involverede regulering af temperatur og luftfugtighed. Til regulering af temperatur er anvendt et PWM styret varmelegeme, og til luftfugtighedsregulering anvendes en forstørret af en magnetventil.

Abstract

In the semester project we sought to develop a hatching machine, initially with the hatching of chicken eggs in focus.

During the start of the project, there were used various UML and SysML tools to provide an overview of the project. During this period the develop metode, V - model was used. In the project's design and implementation phase it was attempted to apply the working method Scrum.

The project involves two systems, one of them, a microcontroller of the type PSoC3, the other an embedded system, DevKit8000. Part of the project, was to make the two systems communicate via SPI.

For PSoC3 the primary focus was on applying different types of I2C sensors, control loops as well as controlling DC- and stepper motors. These components have been worked with both hardware of software wise. The software written to PSoC3, was written in the programming language C, and the program PSoC Creator was used.

The embedded system, DevKit8000, was set up with a Linux based operating system. For the DevKit8000 a driver was made, in the form of a Linux kernel module, which had to deal with the SPI communication to PSoC3 and an interrupt line. In addition, a GUI application was developed, which used the aforementioned driver to show/send data from/to the PSoC3. The application is written in C++, using the Qt framework.

The project involved regulation of temperature and humidity. For regulation of the temperature, a PWM controlled heater was used, and for regulation of the humidity, an atmomizer controlled by a solenoid valve was used.

Indholdsfortegnelse

Resumé	iii
Abstract	iv
Kapitel 1 Indledning	1
1.1 Ordliste	1
Kapitel 2 Opgaveformulering	2
2.1 Projektformulering - Projekt rugemaskine	2
2.1.1 Baggrund	2
2.1.2 Projektdefinition	2
Kapitel 3 Projektafgrænsning	4
Kapitel 4 Systembeskrivelse	5
Kapitel 5 Krav	6
5.1 Generelt	6
5.2 Use Cases	6
5.2.1 Aktører	7
Kapitel 6 Projektbeskrivelse	8
6.1 Projektgennemførelse	8
6.2 Metoder	8
6.3 Specifikation og analyse	9
6.4 Systemarkitektur	10
6.4.1 Block Diagrammer	10
6.4.2 Software Diagrammer	12
6.5 Design og implementering	13
6.5.1 Sensor - Dannie	13
6.5.2 Varmelegeme - Andreas	14
6.5.3 Befugter - Jens	15
6.5.4 Lågekontakt - Jens	16
6.5.5 Aktuator interface - Jens	17
6.5.6 Regulering - Morten	18
6.5.7 Æggenvending - Andreas	22
6.5.8 Sekvenstimer - Simon og Stine	23
6.5.9 PSoC SPI - Simon og Stine	24
6.5.10 PSoC main - Simon og Stine	26
6.5.11 Devkit driver - Mathias	27
6.5.12 Devkit GUI - Mathias	28
6.6 Resultater og diskussion	30

6.7	Udviklingsværktøjer	31
6.8	Erfaringer	32
6.9	Fremtidigt arbejde	33
6.9.1	Mangler	33
6.9.2	Fremtidige muligheder	33
Kapitel 7 Konklusion		34
7.1	Personlige konklusioner	34
7.1.1	Andreas	34
7.1.2	Dannie	34
7.1.3	Jens	35
7.1.4	Mathias	35
7.1.5	Morten	36
7.1.6	Simon	36
7.1.7	Stine	37
7.2	Generel konklusion	37
Litteratur		38

Indledning 1

Dette 3. semestersprojekt går ud på at udvikle en rugesmaskine.

Projektet er underlagt nogle krav, specifiseret i det udleverede projektoplæg:

- Systemet skal via sensorer/aktuatorer interagere med omverdenen
- Systemet skal have brugerinteraktion
- Systemet skal indeholde faglige elementer fra semesterets andre fag.
- Systemet skal anvende DevKit8000 og PSoC3 teknologi.

Ydermere er der følgende fokuspunkter:

- Udarbejdelse af et abstract rettet mod eksterne folk om projektet.
- Implementering og test af et udviklingsprojekt med både HW og SW, der integrerer semesterets kurser.
- Definition af en kravspecifikation for projektet.
- Samarbejde i grupper med både HW og SW udviklerroller
- Arbejdsmetode orienteret mod at udvikle nye produkter baseret på HW og SW.

1.1 Ordliste

Denne ordliste er med for at give forståelse for nogen af de begreber der bliver brugt igennem rapporten.

Ord	Betydning
SysML	System Modeling Language.
Emner	Kyllinger og æg.
Maskine	Betegner de fysiske rammer.
System	Betegner de funktionelle rammer.
Ægtype	Typen af æg, der skal udruges.

Opgaveformulering 2

2.1 Projektformulering - Projekt rugemaskine

Miljøstyring for en rugemaskine til udrugning af æg.

2.1.1 Baggrund

Styring af miljøet i et udrugningsmiljø er kritisk for udrugning af æg. Derfor kræver det et sensoroveråget og -styret rum, som kan justere på følgende parametre:

- Temperatur
- Ventilation/udluftning
- Fugtighed
- Vending
- Afkøling

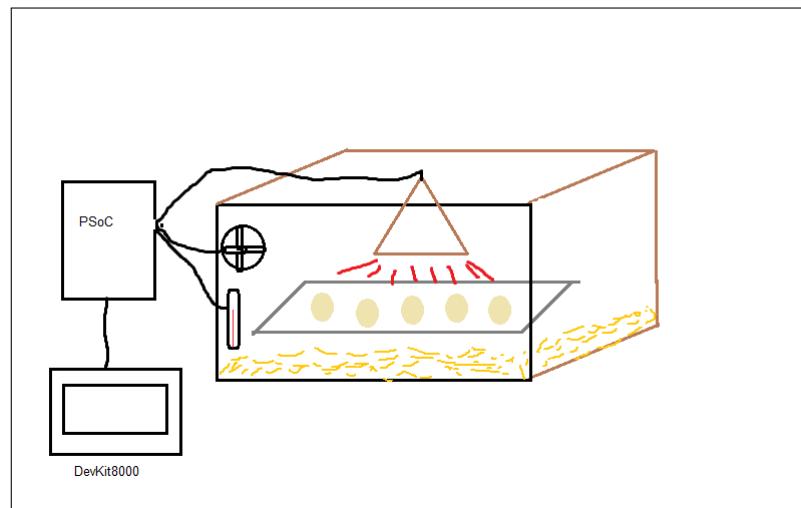
Det optimale forløb for udrugningen af et hønseæg indebærer en temperatur på omkring 37 grader, at luften omkring æggene bliver udskiftet jævnligt, at fugtigheden først i forløbet er mellem 25 og 50 procent, at fugtigheden de sidste dage af udrugningen er så høj som mulig, at æggene vendes 2-3 gange dagligt, og til sidst at æggene afkøles en gang om dagen.

2.1.2 Projektdefinition

Der skal fremstilles et system til overvågning og styring af en rugekasse. Systemet skal opfylde nedenstående systemkrav:

- Systemet skal kontinuert overvåge og styre miljøet i rugekassen ud fra brancheanerkendte parametre, og ud fra en fastlagt procedure vende og afkøle æggene. I denne overvågning og styring skal DevKit8000 og PSoC3 indgå.
- Brugeren skal, vha. DevKit8000, kunne interagere med systemet.
- Brugeren skal, vha. DevKit8000, kunne se rummets nuværende sensorstatus, herunder temperatur.

Projektet er skitseret på figur 2.1



Figur 2.1. Skitse af projekt oversigt

Projektafgrænsning 3

Der er fra projektgruppens side valgt at kører hele projektet digitalt, og derved kunne der undgås at tage stillingen til for meget støj på signalerne.

Systemet er oprindeligt tænkt til at kunne håndtere flere forskellige typer af æg. Det er valgt at fokusere på kun een type, da dette illustrerer funktionaliteten af systemet, og udvidelse kan udføres på basis af denne prototype.

Følgende ting bliver beskrevet som del af systemet, men er ikke blevet implementeret:

- Den mekaniske del af mekanismen til vending af æg. Mekanismen blev bedømt til ikke at være nødvendig for at demonstrere funktionen til at vende æg, da dette kan illustreres ved at stepermotoren drejer rundt. Hvad angår valget af motor til vendemekanismen, er der ikke taget yderligere forudsætninger hvad ang. moment, omdrejnings hastighed osv. Dette skyldes at det var funktionen ved en stepermotor der skulle fremvises i projektet, frem for det færdige resultat.
- De overordnede, fysiske rammer for rugemaskine, selve kassen. Dette skyldes at der vendemekanismen ikke er blevet designet/bygget, det gør at der ikke kan tages stilling til den de enelige mål. Dog er der anvendt en simple kasse (prototype), således at reguleringen har været mulig og teste.
- Der er ikke blevet lavet en selvforsyrende befugter. Dette skyldes at luftfugtigheden bliver øget ved at forstøve vand ind i rugemaskinen, og skulle der laves et system der selv kan opretholde et fast tryk til forstøvelse, vil opgaven blive for omstænding. Derfor er der fundet en løsning med en trykflaske, med dertil en pumpe som øger trykke i beholderne, som gøre det muligt at forstøve vand ind i kassen.

Systembeskrivelse 4

Udrugningssystemet vil bestå af to styringsenheder: DevKit8000 der udgør brugerens interface til systemet, og en PSoC3 der styrer udrugningen.

Derudover vil systemet bestå af sensorer til måling af temperatur og luftfugtighed, aktuatorer til påvirkning af temperatur, luftfugtighed, samt til at vende æggene. Yderligere vil en kasse udgøre miljøet hvori æggene placeres. Kassen vil desuden have en sensor til registrering af åbning og lukning af lågen.

Figur 6.1 på side 10 viser systemets komponenter.

Krav 5

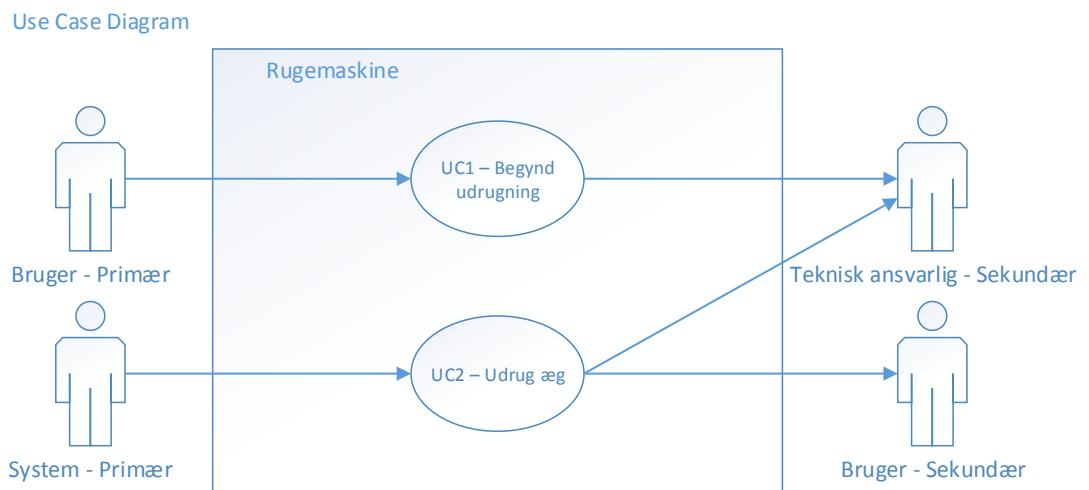
5.1 Generelt

Der er i dette projekt oprettet 3 typer af krav. Der er forhåndskrav bestående af krav til brugen af platformtyper samt indhold af sensorer og/eller aktuatorer. Der er Udrugningskrav som består af fundne anbefalinger til et udrugningsforløb. Det endelige valg er taget fra hjemmesiden¹. Den sidste type af krav er personlige krav til systemets funktionalitet, det er her bl.a. kravene til tests af systemet bliver beskrevet. De personlige krav er stærkt fremkomne i de udarbejdede Use Cases, samt i nogle af de ikke funktionelle krav. Der er i rapporten valgt kun at fokusere på Use Casene, og derfor kan de ikke funktionelle krav kun findes i kravspecifikationen i dokumentationsrapporten.

5.2 Use Cases

Kravene for projektet er formuleret ved brug af Use Cases.

De anvendte Use Cases er illustreret på figur 5.2.1. Denne figur viser Use Case diagrammet som blev udviklet under projektforløbet.



Figur 5.1. Use Case diagram

¹Hønsehuset.dk [2]

Det ses at de funktionelle krav er blevet opdelt i 2 Use Cases, Begynd udrugning og Udrug æg.

Begynd udrugning: Denne Use Case omhandler setup processen, den er beskrevet ud fra brugerens synspunkt. Den beskriver hvorledes brugeren placerer æg i maskinen, samt den efterfølgende process hvor brugeren vælger æggetypen, og derefter starter udrugningsprocessen svarende til det trufne valg.

Udrug æg: Denne Use Case følger direkte efter Use Case 1, og omhandler systemets håndtering af udrugningsprocessen, samt den afsluttende fase når den ordinære process er færdiggjort. Der beskrives reguleringen af varme og fugtighed, samt hvordan der tjekkes for tidspunkter for æggenvending og udluftning. I den afsluttende fase skal der stadig reguleres temperatur og luftfugtighed, men der skal ikke vendes æg.

5.2.1 Aktører

Der er til Use Casene tre vigtige aktører som interagerer med rugemaskinen:

- **Bruger**

Dette er brugeren som ønsker at udruge æg. Brugeren er hovedsageligt aktiv i den indledende og afsluttende fase af udrugningen hvor der skal ilægges æg og fjernes kyllinger og/eller ikke udrugede æg.

- **System**

Systemet er den automatiserede styring af udrugningsprocessen. Det er systemet, der håndterer reguleringen af miljøet og styringen af aktuatorerne der er tilstede i rugemaskinen, samt grænsefladen til brugeren.

- **Teknisk ansvarlig**

Den tekniske ansvarlige er en person med tilstrækkelig viden om rugemaskinen til at kunne fejlfinde og reparere eventuelle fejl.

Projektbeskrivelse

6

6.1 Projektgennemførelse

I starten af projektforløbet blev gruppen enige om en overordnet tidsplan for forløbet. Her skulle medregnes de officielle deadlines for afholdelse af reviews og afleverings dato. Derudover var tidsplanen også med til at give et samlet overblik over hvor mange uger der var til rådighed i projektet. Vha. dette overblik kunne gruppen aftale fælles deadlines for f.eks. færdiggørelse af systemarkitektur og implementering. Dette især for at sikre en passende periode afsat til at afslutte projektet i sidste ende.

Fra begyndelsen af forløbet blev gruppen ligeledes delt op i mindre grupper, alt efter hvilke ansvarsområder de skulle have.

Det blev besluttet at de tre IKT-studerende skulle stå for størstedelen af software opgaverne, heriblandt programmering af DevKit8000. De tre Elektro- og den ene Stærkstrøm-studerende skulle stå for de hardware mæssige opgaver, som f.eks. varmeregulering. Derudover skulle en af gruppemedlemmerne agerer Scrum-master og stå for det praktiske i gruppen, såsom aftale mødetidspunkter.

Opdelingen i undergrupperne skulle først ske efter de indledende faser af projektet var gennemført. Her er tale om den indledende fase, hvor projekt-produktet skal findes, specifikationen af krav samt fremstilling af systemarkitekturen. Dette skulle gøres i fællesskab, for at skabe fælles overblik og enighed over produktet.

6.2 Metoder

Som udgangspunkt blev arbejdsmetoden Scrum valgt som den overordnede arbejdsmetode i projektet. Scrum er en agil arbejdsmetode der ligger vægt på at dele opgaverne op i mindre dele, hvorefter en projektgruppe arbejder på disse dele intensivt i et såkaldt sprint. I dette projekt blev lagt fokus på at lægge fokus på en enkelt funktionalitet i projektet pr sprint. Det blev anset som meget vigtigt at få en funktionalitet til at virke hele vejen igennem systemet. F.eks skulle der som udgangspunkt oprettes kommunikation imellem DevKit8000, PSoC3 og sensoren. Der blev lagt fokus på at holde de ugentlige statusmøder som Scrum-metoden også har fokus på. Disse møder skulle foregå to gange om ugen, hvor hvert gruppemedlem gav en status deres individuelle opgaver.

I begyndelsen af projektet, hvor systemarkitekturen for projektet skulle fastlægges, blev der brugt den såkaldte V-model. Denne model betegnes som en udvidelse af vandfaltsmodellen. I V-modellen bliver der lagt fokus på at have et overordnet overblik over alle faser i

projektforløbet; bl.a design, implementering og test. V-modellen blev af projektgruppen brugt i den indledende fase af projektet; dvs. under fastlæggelse af projektformulering, kravsspecifikation og systemarkitektur. Denne metode blev valgt, idet det blev anset som vigtigt at alle gruppemedlemmer havde en overordnet forståelse for det endelige produkt og dens ønskede funktionalitet.

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. Systemarkitekturen blev som sagt udarbejdet i fællesskab. Dette gjorde at alle ville have en overordnet idé om, hvilke funktionaliteter der skulle være med i det endelige produkt. Derudover fik alle en mulighed for at få indflydelse på designet af produktet.

6.3 Specifikation og analyse

Under undersøgelse omkring hvad et godt udrugningsmiljø¹ er, stod det klart at den korrekte temperatur og luftfugtighed var et kritisk element i systemet. Disse oplysninger afkaster det ret skrappe krav om at systemet skulle være i stand til at holde temperaturen på $\pm 1^{\circ}\text{C}$, samt luftfugtigheden på ± 10 procentpoint.

Der findes mange forskellige holdninger til hvordan æggene roteres bedst. Nogen mener at æggene skal roteres den samme vej og andre at æggene skal roteres skiftevis med og mod uret. Disse oplysninger førte til kravet om at ægget skal roteres begge veje.

Endvidere blev det beslutte at:

- DevKit8000 skulle fungere som brugerens indgang til styring af systemet. En grafisk brugergrænseflade skulle udvikles til og afvikles på denne.
- PSoC3 skulle fungere som den regulerende del af systemet. Den skulle fungere uafhængigt af DevKit8000 når udrugningsprocessen er i gang.
- Der skulle være en klar skillelinie mellem DevKit8000 og resten af systemet. Da DevKit8000 er en ret dyr del af systemet, og i forbindelse med evt. masseproduktion er det en del, der gerne måtte erstattes af en billigere komponent.
- Der skal benyttes digitale sensorer til måling af temperatur og luftfugtighed. Færdigkalibrerede, digitale sensor fjerner nødvendigheden af at der skal fortages besværlige og tidskrævende kalibreringer af sensorer. Samtidigt er miljøet der skal måles på, velegnet til denne type sensorer. Ydermere vil et valg af bestemte typer gøre udviklingen hurtigere, da udviklingsmiljøet for PSoC3 har indbygget standard-metoder til at håndtere disse.
- PSoC3 og DevKit8000 skal kommunikere med hinanden gennem SPI, da der er erfaring med at koble disse to enheder sammen gennem dette interface.
- Sensorerne skal være af typen I2C, da der ligeledes er erfaring med at benytte dette interface, og dette interface understøttes direkte af udviklingsmiljøet for PSoC3.

¹Hønsehuset.dk [2]

- Opvarmningen af udrugningsmiljøet skal foretages med et varmelegeme, dette er for at få en ligelig fordeling af varmen når der er påkoblet en blæser. Varmelegemet er også nemt til brug af PWM-styring, da det kan styres med et simpelt mosfetkredsløb.

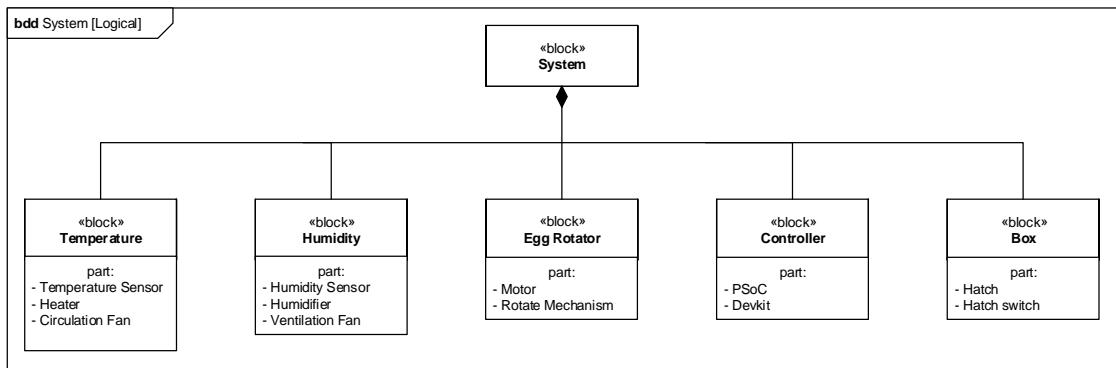
6.4 Systemarkitektur

Dette afsnit omhandler de strukturelle rammer, som systemet blev lavet ud fra.

6.4.1 Block Diagrammer

De krav, der er stillet til systemet i projektoplægget - PSoC3 og DevKit8000 skal anvendes, der skal være bruger-interaktion, og der skal anvendes aktuatorer og følgere til interaktion med omverdenen - førte til beslutningen om at systemet skulle kunne måle temperatur og luftfugtighed, og regulere disse igennem eksterne komponenter. Det blev også besluttet at systemet skulle kunne rotere emner ved hjælp af en stepermotor. Samtidig blev der truffet beslutning om, at DevKit8000 skulle benyttes til brugerinterface, og dermed være brugerens indgang til styring af systemet. PSoC3 skulle fungere som den regulerende, automatiserede del af systemet.

Til at illustrere disse valg blev der udarbejdet følgende logiske BDD:

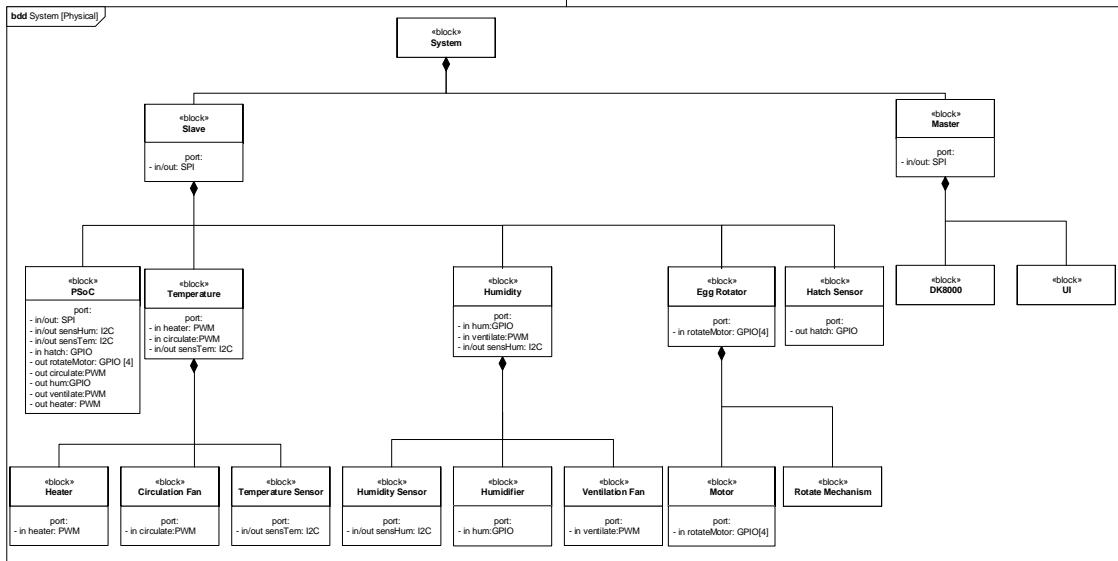


Figur 6.1. Logisk BDD for Systemet

Figur 6.1 viser de fornævnte dele samt nogle tilføjelser. For at opfylde krav om udluftning i Systemet er der blevet tilføjet en ventilations part under "Humidity"blokken, og for at nå en bedre varmefordeling er der ligeledes placeret en cirkulations part i "Heater"blokken.

"Box"blokken henviser til den fysiske rugekasse, der ikke er blevet lavet, og på samme måde henviser "Rotate Mechanism"parten i "Egg Rotator"blokken til den fysiske mekanisme, der fysisk skulle rotere emner i rugekassen.

Systemet blev herefter delt op i to fysiske dele: En Master del, som udgøres af DevKit8000 alene, samt en Slave del, der udgøres af PSoC3, samt de dele den kontrollerer. Der blev også taget stilling til hvordan de forskellige elementer skulle interagere med hinanden og igennem hvilket type interface. Dette er opsummeret i følgende fysiske BDD:

**Figur 6.2.** Fysisk BDD for Systemet

Som det fremgår af Figur 6.2, blev det besluttet at kommunikation imellem de to blokke, PSoC3 og DevKit8000, skulle ske igennem SPI. Dette valg blev truffet ud fra foregående erfaringer med at kommunikere imellem disse igennem dette interface, opnået i forbindelse med andre opgaver. Som det også fremgår, blev det ligeledes besluttet, at alle sensorer skulle være digitale, og at de skulle være af I2C typen, da det ligeledes var noget vi havde erfaring med gennem tidligere opgaver.

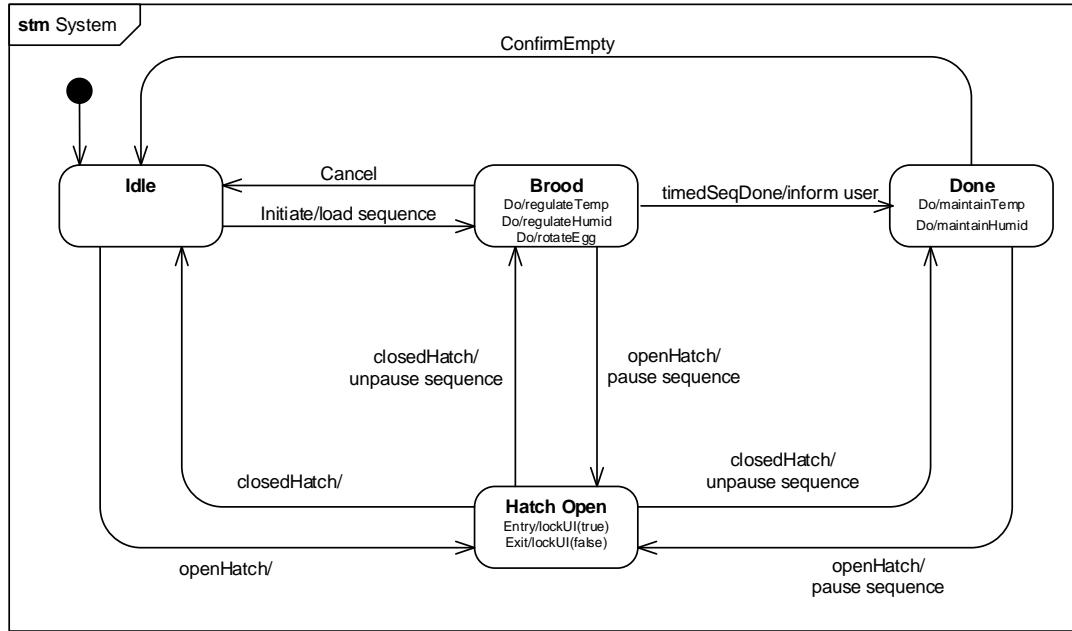
Siden systemet grundlæggende udvikles til at skulle kunne håndtere forskellige typer af æg, der evt. har forskellige krav til omgivelsernes temperatur, blev det også valgt at den styrende udgang til varmeregulering skulle være af en velkendt, variabel type, og her faldt valget på et PWM signal.

For flere detaljer omkring de interne forbindelser henvises til IBD'erne i dokumentationen.

6.4.2 Software Diagrammer

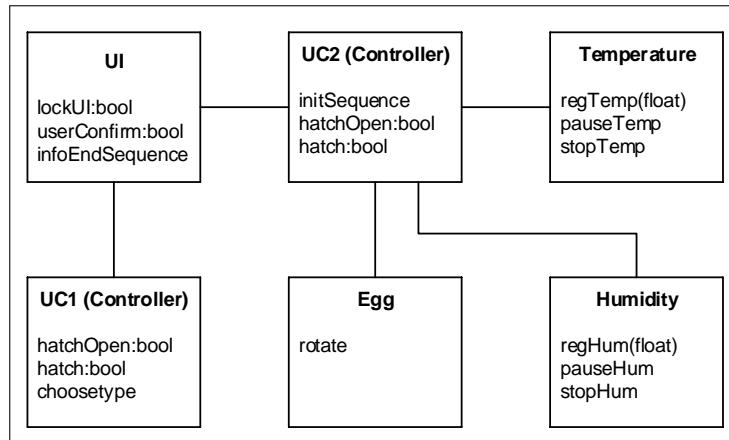
Fra Use Casene blev den grundlæggende software arkitektur fastlagt ved brug af de sædvanlige metoder; domæne-modeller, sekvens- og klasse-diagrammer og state machines.

På Figur 6.3 ses state machine diagrammet for systemet, som giver et overblik over opførelsen af systemet.



Figur 6.3. State Machine for Systemet

På Figur 6.4 ses klassediagrammet, som viser de funktioner, der skal implementeres:



Figur 6.4. Klassediagram for Systemet

Det er værd at bemærke, at hovedparten af dette software afvikles på PSoC3.

For sekvensdiagrammerne henvises til dokumentationen.

6.5 Design og implementering

6.5.1 Sensor - Dannie

Design

Til udrugningen skal luftfugtigheden og temperaturen holdes på et fast niveau. Ved lidt hurtigt søgning 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 2,3 V og 5,5 V[5], da PSoC3 spændingsniveau ligger på 3,3 V eller 5 V vil det ikke være et problem uanset PSoC3 spændingsniveau. Da sensoren følger en I2C protokol[4], skal der ikke tages hensyn til støj.

Implementering

Der er implementeret 5 funktioner til HIH6121-021 softwaren:

1. `init_HIH61xx()` har til formål at implementere I2C standardkomponentens kode i PSoC3 programmet.
2. `measure_HIH61xx()` har til formål at kommunikere med sensoren. Den får sensoren til at måle luftfugtighed og temperatur.
3. `read_HIH61xx()` har til formål at aflæse de målte værdier fra sensoren. Den aflæser 4 data bytes, som funktionen allokerer og behandler lokalt.
4. `getTemp()` har til formål at returnere den sidste aflæste temperaturværdi fra sensoren.
5. `getHumid()` har til formål at returnere den sidste aflæste luftfugtighedsværdi fra sensoren.

Funktionerne er delt op, således at hver enkelt modul som skal bruge data fra sensoren, ikke skal kommunikere hver gang de skal bruge data, de kan simpelt kalde `getTemp()` eller `getHumid()` og derved få de sidste opdaterede værdier. `measure_HIH61xx` og `read_HIH61xx` er lavet således at de kan blive kaldt når main programmet har tid til at opdatere værdierne af luftfugtighed og temperatur, så er det op til main programmet at bestemme hvor ofte værdierne skal opdateres.

6.5.2 Varmelegeme - Andreas

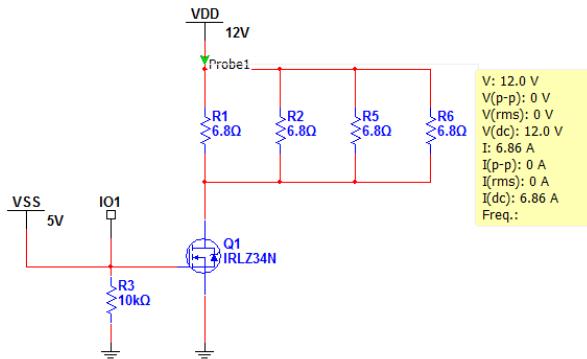
Design

Der er i løbet af designfasen for varmelegemet fremkommet 2 designtyper. Den ene er brugen af en simpel varmelampe, det andet design er en køleplade med effektmodstande monteret ovenpå for at opvarme selve pladen og derved afgive varme til omgivelserne. Det er besluttet at beholde den sidste idé med effektmodstandene. Dette skyldes nem tilgang til komponenterne der skal bruges, samt en varmelampe bruger 230V som ikke er forbundet til vores system umiddelbart. Der er for god ordens skyld tilføjet en blæser til at lave en mere ligelig fordeling af varmen.

Det er vurderet at der skal bruges en effekt på 60-80W til opvarmning af miljøet.

Implementering

Varmelegemet blev fremstillet med 4 effektmodstande på hver 6.8Ω og max 50W. Med en 12V strømforsyning og uden hensyn til spændingsfald over en MOSFET transistor, så ender den teoretiske effekt på 84.7W og er simuleret til 82.32W med det simulerede kredsløb på figur 6.5.



Figur 6.5. Varmelegeme simuleringskredsløb

6.5.3 Befugter - Jens

Design

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

Der har været følgende design idéer i spil:

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

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

- En beholder fyldt med vand under et konstant tryk via en kompressor, selve styringen af forstøvningen af vandet skulle forgå ved at åben/lukke en magnetventil.

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

Implementering

Nedenstående figur 6.6 illustrerer den færdige implementering af hardware delen til befugtning af luften.



Figur 6.6. Havesprøjte med på monteret magnetventil

Via en empirisk tilgang blev der fundet frem til at den korteste puls der kunne åbnes for befugtningen med var 10ms. Denne tilgang bestod i at skrive et test program til PSoC3 til aktivering af magnetventilen, hvor det gradvist blev prøvet at sænke varigheden på åbnings impulsen.

- 5ms var ikke nok til at magnetventil kunne nå at åben.
- 7ms var lige grænsen for hvad der kunne lande sig gøre, da dette gav et udstabilt resultat

6.5.4 Lågekontakt - Jens

Design

For at muliggøre registreringen af lågens status er det nødvendig med den form for kontakt, som har følgende egenskaber: Et GPIO i form af 5VDC, samt have den fysiske udførelse der muliggør overvågning af ledningsbrud.

Lågekontakten signal skal både forbindes til PSoC og DevKit8000, da begge enheder har brug for at kende lågens status.

Dette kan udføres på flere måder, bl.a.

- To separate kontakter. En til PSoC3 samt en til DevKit.
- En kontakt med et signal til PSoC3, samt et signal til DevKit.
- En kontakt med signalet til PSoC3 og herfra videre route signalet til DevKit8000.

Løsningen der blev valgt var: *En kontakt med signalet til PSoC3 og herfra videre route signalet til DevKit8000.* Dette valg blev fortaget hovedsageligt pga. at det er muligt at fjerne prel fra signalet på PSoC3, således at ekstra hardware komponenter kan udgås.

Implementation

Kontaktprellet bliver fjernet via en debouncer, som er en standard komponent i PSoC3 creator.

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

q udgangen på debounce filteret er ført videre til DevKit8000. Denne udgang er filtreret for kontakt prel.

Der bliver kaldt en ISR på både den positive og negative flanke, denne ISR toggeler et flag som bliver brugt i "Main" til at overvåge lågens status.

6.5.5 Aktuator interface - Jens

Aktuator interfacet er bindelede imellem PSoC3 og de fysiske aktuatore. Da der ikke er forskel på hvordan aktuator drives i systemet er der i det følgende afsnit kun beskrevet generelt.

Design

Der er flere måder at foretage styringen af aktuatorer på, dette kunne fx. være via relæ, solid state relæ, effekt transistor, MOSFET eller Darlington array.

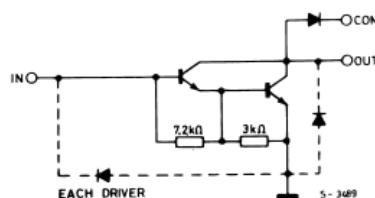
Til at trække stepmorten, magnetventilen, samt de to blæsere faldt designvalget på et Darlington array.

Dette designvalg blev truffet på baggrund af følgende overvejelser.

- Relæet ville ikke være optimal til PWM sammenhæng pga. det store mekaniske slid dette ville medføre på relækontaktorerne, det er heller ikke muligt at køre med en ret høj PWM frekvens pga. spolen i relæet.
- Solid state relæ kunne havde været en mulighed, da der ikke er noget mekaniske slid i denne form for relæer. Solid state relæ blev valgt fra pga. prisen og den fysiske byggestørrelse på komponenten.
- Effekttransistor 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.
- MOSFETen kunne have været et oplagt valg, men blev valgt fra da det ønskes at holde antallet af elektriske komponenter på et minimum, hvilket et darlington array muliggører.

Implementering

Aktuator interfacet er implementeret med to Darlington arrays af typen ULN2001 (figur 6.7 viser princippet vha. et diagram). Grunden til at der er anvendt to Darlington arrays er at der anvendes to forskellige spændinger i systemet, hhv. 5VDC til stepmotoren og 12VDC til blæsere samt magnetventilen. Denne type Darlington array indholder 7 undgange der hver er i stand til at trække en strøm på 500mA. Det er muligt at parallel koble disse udgangs strømmme. Denne egenskab udnyttes således at det er muligt at trække magnetventilen der trækker strøm på 1A. På hver indgang er der monteret en $2.7\text{k}\Omega$ modstand for at tilpasse indgangsstrømmen til 5VDC CMOS kredse.



ULN2001 (each driver)

Figur 6.7. 1/7 af ULN2001

6.5.6 Regulering - Morten

Regulering af luftfugtighed og temperatur er implementeret på stort set samme måde, så begge vil blive beskrevet i dette afsnit. Selve design og implementering er en kombination af PSoC3 hardwareblokke og software, til at styre disse hardwareblokke. Funktionerne dette afsnit omhandler, er funktionerne beskrevet under "Humidity" og "Temperature" i figur 6.4.

Design

Funktionerne til at styre reguleringen af temperatur og luftfugtighed er

- `regTemp(float)/regHum(float)` - Skal starte reguleringen af hhv. temperatur og luftfugtighed
- `stopTemp()/stopHum()` - Skal afbryde reguleringen
- `pauseTemp()/pauseHum()` - Skal afbryde regulering når den kaldes, og igangsættes reguleringen når den kaldes igen

Det ønskes, at reguleringen skal foregå automatisk efter at være igangsat. Dertil benyttes timer til at, med faste mellemrum, generere interrupts, og reguleringen vil da foregå i Interrupt service rutinen. Da reguleringen ikke er noget der er tidskritisk vil disse ISR have lav prioritet.

Temperatur og luftfugtighed har forskellige forudsætninger, og det giver anledning til følgende designbeslutninger:

- Varmeudveksling med omgivelserne vil gøre, at systemets temperatur konstant vil falde og systemet skal derfor have tilført en bestemt mængde energi hele tiden. Da varmelegemet styres ved hjælp af et PWM signal, skal ISR'en udføre en PID regulering og output af denne skal bestemme Duty Cycle af PWM signalet.
- Under forudsætningen af at systemet er godt isoleret, vil luftfugtigheden ikke falde. Så snart temperaturen er stabil, vil luftfugtigheden også være stabil. Da øgningen af luftfugtighed foregår via åbning og lukning af en magnetventil, skal ISR'en blot vurdere om luftfugtigheden er for lav, og åbne for ventilen hvis den er.

Ydermere bemærkes det at

- Da systemet ikke underbygger en mulighed for aktivt at sænke luftfugtigheden, er det vigtigt, at reguleringen ikke tilfører for store mængder fugtighed.
- Da luftfugtigheden er afhængig af temperaturen er det vigtigt, at PID reguleringen ikke laver oversving, da dette kan resultere i for høj luftfugtighed.

Begge reguleringer kræver information vedrørende hhv. systemets aktuelle temperatur og luftfugtigheden, og til dette anvendes de funktionerne beskrevet i afsnittet 6.5.1 om sensoren.

Ligeledes laves der i ISR'erne, er en underroutine, der skal holde øje med stabiliteten af systemet ved at

- Monitorere hvorvidt systemet når de krævede niveauer indenfor den tilladte tid
- Monitorere om systemet er stabilt og holder de krævede niveauer indenfor de tilladte grænser
- Meddele brugeren hvis de to forrige punkter ikke overholdes

som er påkrævede funktionaliteter iht. kravspecifikationen.

Implementering

Alle `reg.pause` og `stop` funktioner er implementeret ved hjælp af de standardfunktioner, der er tilknyttet de hardwareblokke, der blev benyttet til de forskellige formål; hver regulering havde sin egen timer blok og tilhørende funktioner, og temperaturreguleringen havde ligeledes en PWM blok, til styring af varmelegemet samt cirkulations-ventilatoren. Cirkulationsventilatorens duty-cycle sættes til en fast værdi.

Som eksempel på implementering af funktionerne vises `pauseHum()`:

```

136 void pauseHum(){
137     if(isRunning && paused){
138         paused = 0;
139         Humidity_timer_Start();
140     }
141     if(isRunning && !paused){
142         paused = 1;
143         Humidity_timer_Stop();
144     }
145 }
```

Figur 6.8. Implementering af `pauseHum()`

Som det ses af Figur 6.8 benyttes funktionerne `Humidity_timer_Stop()` og `Humidity_timer_Start()`, der starter og stopper for timeren, der skaber de interrupts, der foretager reguleringen af luftfugtigheden. De øvrige funktioner er implementeret på samme vis. Flagene `isRunning` og `paused` benyttes i rutinerne til at afgøre om ting startes, stoppes eller der ingenting skal foretages. Se kildekoden og dokumentation for flere detaljer ang. implementering af disse.

Selve reguleringen foregår ud fra differensen mellem de aktuelle og de ønskede værdier for temperatur og luftfugtighed; til regulering af luftfugtighed benyttes en simpel if-sætning, og til varmeregulering er implementeret følgende hjælpe-rutine:

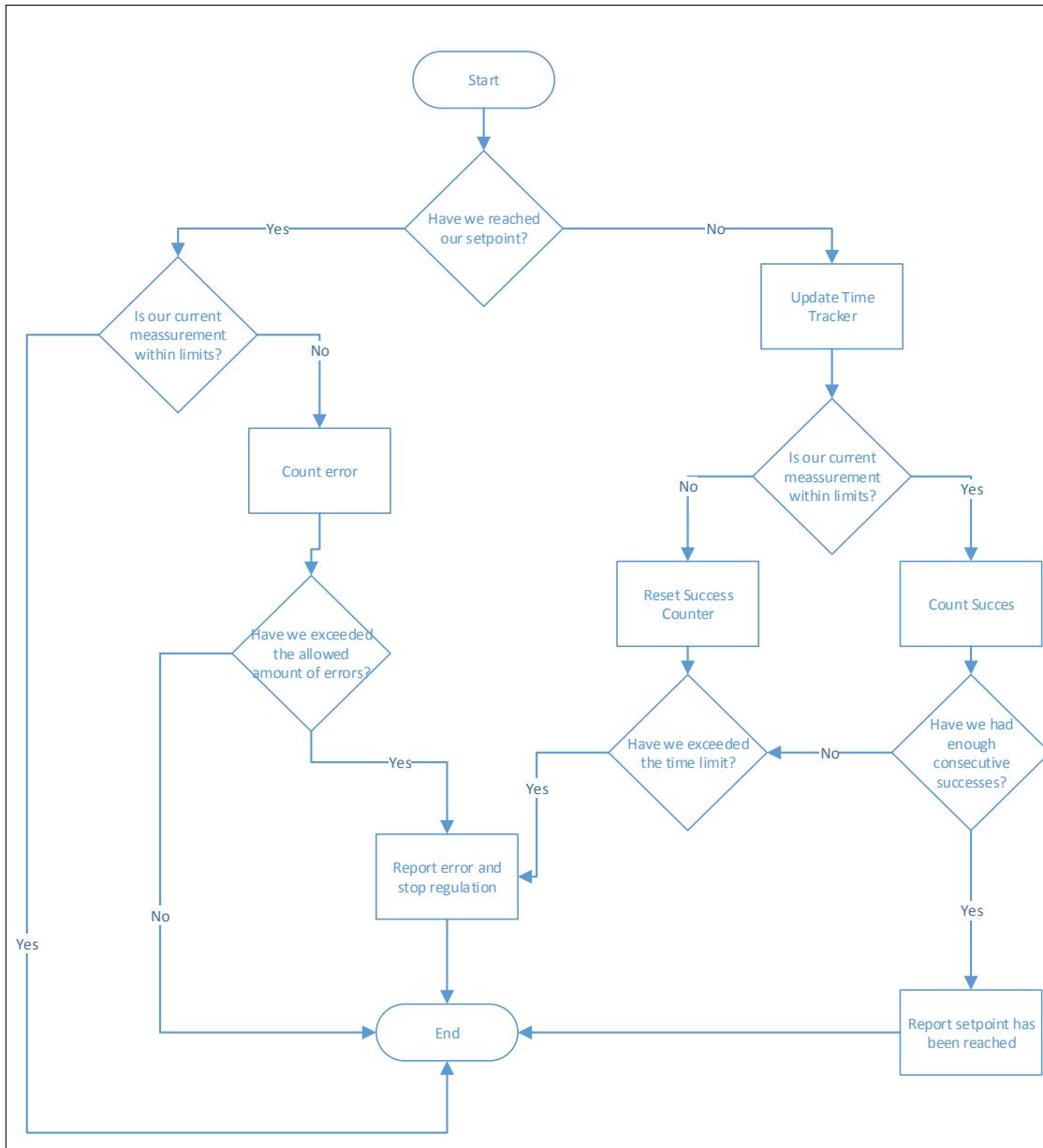
```

166 void regulate(float error){
167     int pwm;
168     float derivative;
169
170     // Calculate integral
171     integral = integral + (error * UPDATE_INTERVAL);
172
173     // Limit Integral
174     if(integral >= 2000){
175         integral = 2000;
176     }
177
178     if(integral <= -2000){
179         integral = -2000;
180     }
181
182     // Calculate derivative
183     derivative = (error - last_error)/ UPDATE_INTERVAL;
184
185     // Calculate PWM value
186     pwm = ((Kp * error) + (Ki * integral) + (Kd * derivative));
187
188     // Limit PWM range
189     if (pwm > 255){
190         pwm = 255;
191     }
192
193     if (pwm < 0){
194         pwm = 0;
195     }
196
197     // Set Duty Cycle
198     Heater_PWM_WriteCompare1(pwm);      // Set Duty-Cycle
199
200     // Save error for future use
201     last_error = error;
202
203 }
```

Figur 6.9. PID hjælperutine

Som det ses af linie 198 anvendes igen standard funktioner.

Til at undersøge stabiliteten af systemet er implementeret en hjælpefunktion kaldet **stabilityControl(float)**. Den er opbygget af en kringlet mængde if-sætninger, flag og counters, så funktionaliteten kan bedst beskrives gennem dette flowchart:

**Figur 6.10.** Flowdiagram for `stabilityControl(float)`

For mere information henvises til kildekoden.

6.5.7 Åggevending - Andreas

Hele æggemekanismen indeholder 3 dele, disse er en fysisk stepermotor, en mekanisk vendeoapsætning, samt en software styring af stepermotoren. Den fysiske stepermotor er valgt som en unipolær, gearet stepermotor². Den mekaniske del er ikke implementeret, men der er foretaget designovervejelser over denne del, og software styringen er beskrevet i den følgende del af rapporten.

Design

Til stepermotorstyringen er der overvejet at lave med hensyn til stepping metode. De relevante metoder til dette projekt er enkelt faset full stepping, dobbelt faset full stepping, eller half stepping. Da kravene til motorens præcision er meget lave, så er half stepping, som har en højere oplosning, hurtigt sorteret fra. Valget der nu skal tages er mellem større moment eller lavere strømforbrug med evne til hurtigere hastighed. Her er der som udgangspunkt valgt at tage 2 faset stepping som har det højere moment.

Efter stepping metoden er på plads, så skal udførelsen af tidsintervallerne mellem hvert step overvejes. Her blev det bestemt at styringen ikke skulle være tidskritisk, og derfor blev denne lavet med et interrupt og en timer.

Æggene skal efter anbefalinger roteres en halv omgang ad gangen, og der skal ikke foretages hele omdrejninger, derfor skal der mellem hver halv-rotation ændres retning af motoren.

Implementering

Stepmotorstyringen blev som udgangspunkt designet og testet med den to-faset full step styring som var planlagt, men grundet en fejl der opstod i en afsluttende test, så blev en tidlig version som var lavet med enkelt faset full stepping brugt i stedet. Antal steps der skulle tages for at en halv rotation af motoren endte op på 1024 stk.

Tidsintervallerne der blev testet med stepermotorstyringen blev sat til 1, 2 og 5ms, der opstod fejl ved 1ms intervallerne, mens der ingen var ved 2 og 5ms. Den senere test der lavede fejl opstod ved 2ms interval og der blev forhøjet til 2.73ms som var den umiddelbare maximum interval på den allerede indførte 16bit timer, dette var dog ikke nok, og en enkelt faset full step styring blev genbrugt.

Efter revideringen af motorstyringen opstod der ingen problemer og denne kunne også implementeres under den samlede integrationstest. Der er ikke taget hensyn til en evt. gearing af den mekaniske del af æggemekanismen.

²Se datablad [6]

6.5.8 Sekvenstimer - Simon og Stine

Design

Processen at udruge et æg tager op til flere uger, og det er vigtigt at kunne holde styr på de forskellige faser i udrugningen. Derfor er der brug for en timer, som over lang tid kan holde styr på tiden.

Umiddelbart har vi tre muligheder på PSoC3-platformen, som opfylder ovenstående ønsker:

- 1) En timer som periodisk genererer et interrupt, som kan bruges til at optælle en variabel.
- 2) En count-down timer som udfører en handling, når timeren har talt ned. Timeren indstilles herefter til tiden indtil den næste handling skal udføres.
- 3) En Real-Time Clock (RTC), hvor en alarm kan sættes til det næste tidspunkt hvorpå der skal ske en handling.

Mulighed nr. 1 har fordeloen, at vi hele tiden har en variabel, som beskriver hvor lang tid der er gået - denne information kan bruges, når brugeren skal informeres om tiden.

Implementering

Implementeringen af timeren foregår i PSoC Creator, hvor der vælges en timerkomponent, en clock og en ISR (Interrupt Service Routine). Clocken sættes til timerens clock-indgang, og ligeledes sættes ISR-komponenten til timerens interrupt-udgang.

Timer og clock indstilles så interruptet fremkommer hvert minut.

I softwaren, i ISR-funktionen, sættes et flag til "true" hver gang timeren genererer et interrupt. Dette flag tjekkes i main-løkken, hvorefter minuttælleren forøges med 1.

6.5.9 PSoC SPI - Simon og Stine

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 en brugergrænseflade 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.

DevKittet kan sende følgende beskeder til PSoC3:

- En start kommando - Starter udrugningssekvensen. Starter timer, sætter temperatur og luftfugtighed.
- En slut kommando - Stopper udrugningssekvensen. Stopper timer, temperatur og luftfugtighed.
- En status kommando for temperatur - Send status på temperatur.
- En status kommando for luftfugtighed - Send status på luftfugtighed.
- En status kommando for tid - Send den forløbne tid.

En protokol for SPI bliver udarbejdet for at sikre, at DevKit8000 og PSoC3 sender og modtager det samme. Det besluttes, at der sendes 16 bit ad gangen. Af disse 16 bit er de 4 første reserveret til adressen, de næste 4 til kommandoen og de sidste 8 til data. Start/Slut beskeden får samme adresse, men hver sin kommando. Status beskederne for temp, luftfugtighed og tid får hver sin adresse, men samme kommando. Adresserne og kommandoerne er blevet valgt som følgende:

Adresser er markeret med **rød** og kommandoer er markeret med **grøn**.

Beskrivelse	Bitmønster
Start og Slut	0001 0001 og 0010
Temperatur	0010 0011
Luftfugtighed	0011 0011
Tid	0100 0011

Tabel 6.1. Adresse og kommando bitmønster

Implementering

SPI kommunikationen blev implementeret i en interrupt service rutine, der står for dekodningen af data fra SPI. Implementeringen betød i at oprette en switch case, der skifter på adressen sendt fra DevKit8000. Inde i switch case'en blev der også tilføjet en if-sætning for at kunne skifte imellem Start kommandoen og Slut kommandoen, idet disse deler adresse.

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 **Stop** kommandoen derimod registeres skal følgende ske:

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

Implementeringen af status på temperatur, luftfugtighed og tid er ens, men varierer i form af det der bliver sendt tilbage til DevKit8000. For at sende data til Devkittet skal der sendes en SPI besked. Denne besked indeholder data, indsamlet fra sensoren.

Implementeringen for afsendelsen af data ses nedenfor, hvor der bliver sendt en temperatur.

```
/*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;
```

Som det ses på koden er det adressen 0x2 der svarer til temperatur. Herefter skal kommandoen checkes. Kommandoen for 'Status' er 0x3, som nævnt i design afsnittet. For at sende temperatur data'en benyttes SPI-kommandoen for at skrive data. Parameteren der skal sendes er i dette tilfælde, `getTemp()`. Denne parameter stammer fra sensoren, og indeholder den temperatur, der skal vises på DevKittet.

6.5.10 PSoC main - Simon og Stine

Design

For at forenkle implementeringen af udrugningssekvenser er det blevet besluttet at starte med kun at implementere forløbet for en enkel ægtype. Følgende forløb ønskes:

- Tiden nulstilles når udrugningssekvensen påbegyndes.
- 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 DevKit8000 informeres.

Forløbet for udrugningssekvensen er inspireret fra hjemmesiden³.

Forløbet ønskes implementeret som en for-løkke, der bliver kørt igennem en gang i minuttet. Der skal tages hensyn til de tidsbegrensninger der er i udrugningssekvensen, f.eks. 12 timer og 18 dage. For at håndtere dette, vælges det at omregne tiden til minutter, hvorefter man vha. if-sætninger, kan håndtere begrænsningerne.

Tidsbegrænsningerne omregnet til minutter:

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

Implementering

Implementeringen af forløbet for udrugningen blev lavet i et PSoC projekt.

Operatoren modulus blev valgt til at hjælpe med at implementere de tidsbegrænsede dele af forløbet. En variabel countInMinutes blev oprettet til at indeholde tiden gået siden start. Da countInMinutes indeholder tiden der er gået, kan modulus bruges til at overholde tidsbegrænsningerne. Eksempel på brug af modulus:

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

Denne if-sætning checker på, om det er tid til æggenvending, hvilket er hvert 720. minut.

For hver af punkterne i forløbet (se design afsnit) blev der oprettet en if-sætning, magen til den ovenfor. Ud over if-sætningerne for 12 timer, 24 timer, 18 dage og 21 dage skulle der oprettes en if-sætning til at håndtere simuleringen af at hønen forlader reden. Hver 24 time bliver temperaturen sat til en lavere temperatur og den samtidige tid, countInMinutes,

³Hønsehuset.dk [2]

bliver gemt i en variabel timeStamp. Derudover bliver et flag sat. Dette flag samt timeStamp benyttes i en if-sætning der sørger for at temperaturen kun er sænket i 30 minutter.

6.5.11 Devkit driver - Mathias

Design

Da Devkit8000 er et embedded system, med et Linux baseret OS, skal der for at tilgå fysiske resourcer benyttes et driverlag.

Driveren der skrives til Devkit8000 har brug for følgende funktionalitet:

- Skal kunne kommunikere over SPI protocol til PSoC3.
- Skal kunne modtage interrupt fra lågen.

Til SPI Driveren er der allerede udarbejdet et skelet i faget I3HAL, hvor der er lavet en driver, som kan kommunikere over SPI⁴. Der tages udgangspunkt i denne, der ønskes dog tilføjet følgende funktionalitet:

- Tilpasning til den tidligere definerede protocol i projektet, se afsnit 6.5.9.
- Automatisk oprettelse af node filer ved at tilføje oprettelse af **driver class** og **driver device**.
- Tilføjelse af en interrupt line, GPIO samt ISR til interrupt linien.
- Der skal ændres i read funktionen (funktionen der bruges til at tilgå driveren) så den kan bruges til både SPI og 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.

Implementering

For at oprette node filerne automatisk, som er applikationers tilgang til driver, tilføjes objekter af typen **device** og **class**. Der oprettes en enkelt **device class**, samt et **device** for hver fil der skal oprettes. For at tilpasse driveren til den ønskede SPI protocol oprettes tre "read"filer til at hente "Temp", "Humid"og "Time"samt en read fil til interrupt linien. Til SPI delen oprettes ligeledes en SPI "write".

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 lavet en "Switch case", som kigger på driverens minor nummer (filen der bliver tilgået).

Der blev desuden lavet et script, for at insætte driveren, hotplug modulet, samt ændre indstillinger på den allerede eksisterende cpld driver.

⁴Redmine Driver [1]

6.5.12 Devkit GUI - Mathias

Design

Til den grafiske burgerflade ønskes der et meget simpelt design. Den grafiske brugerflade skal have følgende funktionaliteter:

- Skal kunne starte og stoppe for udrugningssekvensen,
- Skal kunne vise den forløbne tid, samt den aktuelle temperatur og luftfugtighed.
- Skal give besked til brugerne, når lågen er åben.

Dette giver et basisudkast at arbejde ud fra til implementeringsfasen.

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++.

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

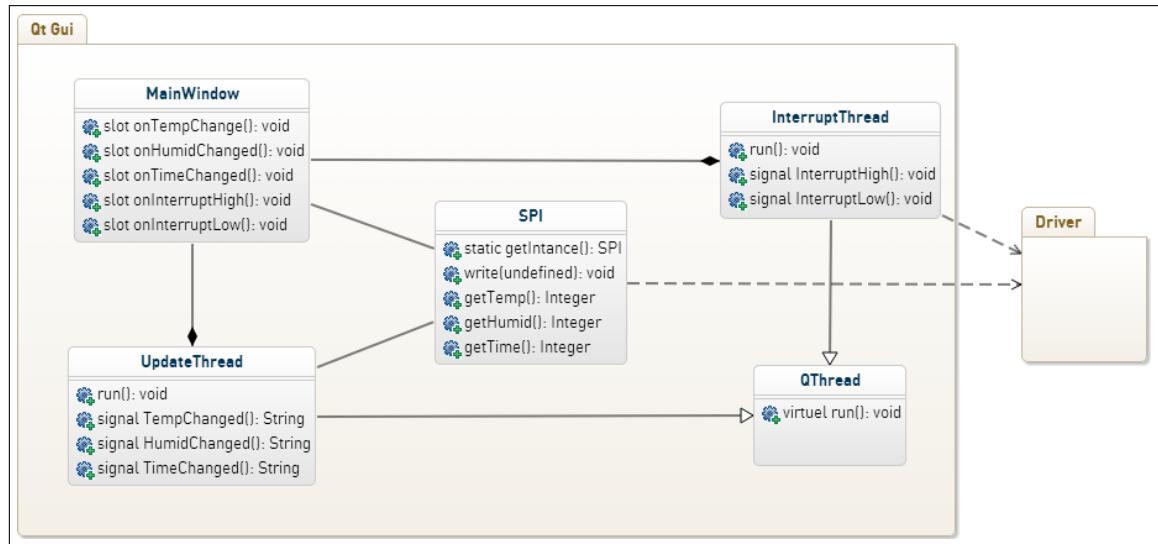
For at give et godt udgangspunkt at arbejde ud fra, blev den grafiske del af programmet først implementeret. Det endelige grafiske design, implementeret i Qt, kan ses på figur 6.11.



Figur 6.11. 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 underliggende software til applikationen.

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 6.12. 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. De to tråde blev lavet som klasserne UpdateThread og InterruptThread og blev, ud fra Qt dokumentationen omkring flertrådet programmering, lavet som nedarvninger fra klassen Qthread⁵.

MainWindow: MainWindow er klassen, der står for at håndtere selve "main"vinduet. Klassen har nogle metoder, der håndterer de to trykknapper samt nogle forskellige slots. De forskellige slots anvendes til at fange signaler emittet fra de to andre tråde.

MainWindow står for at oprette de to datatråde InterruptThread og UpdateThread og har kendskab til SPI klassen.

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. 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. Da der er flere tråde, der skal kunne anvende SPI klassen, blev den givet en mutex, som den i hver af `get` og `write` metoderne låser i starten og frigives

⁵Se Starting Threads with QThread[3]

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`, samt en privat constructor. SPI klassen kan derfor kun tilgås via 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 returnerer en reference dertil. Dette sikrer, at når metoden kaldes næste gang, er instansen allerede oprettet.

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, med det ønskede formatering, og emitte de forskellige signaler til `MainWindow`, som så viser data'en.

InterruptThread: `InterruptThread` står for at læse på interrupt linien i driveren. `InterruptThread` emmitterer så et signal alt efter hvilken værdi der læses fra driveren. 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.

6.6 Resultater og diskussion

Der er i dette projektforløb ikke opnået den ønskede fælles funktionalitet af en rugemaskine. Der opstod mange fejl i løbet af integrationstestene der blev foretaget. De største problemer opstod mens både DevKit8000 og sensoren var tilsluttet. Dette er i gruppen analyseret til at være et problem med 2 forskellige typer af protokoller på PSoC3, SPI til devkit og I2C til sensoren.

Problemet med 2 protokoller på samme platform tænkes at skyldes et timingproblem da begge styres med interrupts, og ikke er implementeret som tidskritiske elementer.

Der blev opnået en delvis funktionalitet med kommunikationen mellem DevKit8000 og PSoC3, så processen kunne initieres og stepmotoren kunne lave sin rotation på det ønskede tidspunkt.

Varmelegemet og befugteren kunne kun begrænset testes da reguleringen er en stor del af opfyldelsen for kravspecifikationen af disse dele. Der er kun lavet enhedstests på reguleringen, da der ikke var en aktuel temperatur som den kunne indregulere sig selv efter. Sensoren havde kun den ønskede funktionalitet hvis denne blev testet som en separat enhed. Der mangler i projektet også den mekaniske del til æggevending.

6.7 Udviklingsværktøjer

En kort beskrivelse af de anvendte udviklingsværktøjer og de erfaringer der er gjort med disse.

1. TortoiseSVN 1.8.7
 - a) TortoiseSVN er et program til at versionstyre og dele filer i projektet, SVN kildenten har været koldet op mod au's server. (<http://svn.nfit.au.dk/>)
2. PSoC Creator 3.0 Servies Pack 1
 - a) PSoC Creator bruges til at bygge software, der programmeres til PSoC3.
3. Qt
 - a) Qt er brugt til udviklingen af det grafiske brugerflade på DevKit8000.
4. MultiSim 11
 - a) MultiSim bruges til at simuler hardware kredsløb samt indtegningen af kredsløb.
5. Ultiboard 11
 - a) Ultiboard er brugt til design af print.
6. Rapportskrivning
 - a) TexStudio (2.7.0 Windows)
 - i. TexStudio er brugt som skriveprogram til dokumentation og rapportskrivning. TexStudio arbejder med filformaten latex(.tex).
 - b) texlive-full (Linux)
 - i. Bruge til at installere alle "packages" som bliver importeres i latex.
 - c) basic MikTex 2.9.5105 (Windows)
 - i. Bruge til at installere alle "packages" som bliver importeres i latex.
7. Microsoft Office Visio 2010
 - a) Visio er blevet brugt til indtegning af diagrammer.

Størstedelen af projektgruppen har skullet lære at bruge SVN og Latex på et mere anvendeligt niveau end blot kendskab til programmerne, hvilket i starte gav nogle problemer. De resterende programmer er enten blevet lært at kende igennem undervisning eller er kendt fra tidligere semestre.

6.8 Erfaringer

Gruppen har igennem semester-projektet erhvervet sig en del erfaringer omkring hvordan et projekt kan gennemføres bedre. En af de vigtigste erfaringer erhvervet er, at testen af funktionaliteterne i projektet skal fylde mere. Der skal afsættes mere tid af til det og gruppen skal være bedre til at teste enkelte dele sammen. I dette projekt blev hver enkel del testet alene, for så at teste det hele samlet til sidst. Dette gjorde, at da systemet ikke fungerede var det meget besværligt at finde frem til fejlen. Der blev ikke sat specifik tid af til test i tidsplanen. Ud over denne mangel blev tidsplanen fulgt fornuftigt, især i starten af projektet under de indledende faser.

En anden vigtig erfaring erhvervet er, at der hurtigere skal tages beslutninger om, hvorvidt en enhed skal skrottes eller ej. Der blev i projektets begyndelse valgt at bruge en type sensor, som senere viste sig at volde store problemer. Her skulle gruppen have været hurtigere til at vælge et alternativ, da sensoren udgør en så stor del af rugemaskinen. En hurtigere måde at vælge alternativ, kunne være at have en plan B, fra starten af. På den måde kunne man have det nemmere ved at skrotte den oprindelige idé. Da en erstatning endelig var blevet valgt, skulle denne også have været testet bedre med de andre dele, da denne del blev anset som værende en af fejlende ved den endelige test.

Derudover skal der også dannes en fælles enighed over, hvordan kode skrives. Dette vil hjælpe i sidste ende, når al koden skal sammensættes i ét program. Især er det en god idé tage højde for brugen af interrupts tidligt i forløbet. Det viste sig at nogen af de interrupts, der blev brugt, ikke fungerede ligeså godt ren timings mæssigt når hele projektet var samlet. Dette er med til at bringe mere fokus på de manglende test.

6.9 Fremtidigt arbejde

Følgende afsnit omhandler det fremtidigt arbejde med projektet. Afsnittet er delt ind i to afsnit, der henholdsvis omhandler hvad der mangler før at rugemaskine er færdig, samt fremtidige udviklingsmuligheder med projektet.

6.9.1 Mangler

For at rugemaskinen bliver funktionel mangler der følgende:

- Fejlhåndteringen skal implementeres, så den lever op til kravspecifikationen.
- Fejlhåndteringen bør også udbygges, så den omhandler evt. strømafbrydelser m.v.
- Måling af temperatur og luftfugtighed skal implementeres.
- Den mekaniske del af æggevendingen skal fremstilles.
- Selve "kassen", der udgør de fysiske rammer for rugemaskinen, skal fremstilles og alle del-komponenterne skal monteres i/på denne "kasse".
- Reguleringssløjferne skal justeres så de virker optimalt. Dette kræver, at "kassen" er lavet.

6.9.2 Fremtidige muligheder

Der er store fremtidige muligheder i rugemaskine. Følgende funktionaliteter kunne tilføjes:

- At give brugeren mulighed for at vælge mellem flere forskellige ægtyper.
- Overvågning af vand niveauet i befugteren, således at brugeren kan informeres, hvis befugteren er tom for vand.
- At koble rugemaskinen op på internettet, dette vil bl.a. give følgende udviklings muligheder.
 - En mobil app, således at bruger har mulighed for at følge med i udrugningsprocessen. Dette vil endvidere gøre det muligt at informere brugeren hvis der skulle opstå fejl eller utilsigtede hændelser.
 - Gøre det muligt at live-streame video samt lyd fra rugemaskine.
 - Muliggøre betjening/overvågning af mange rugemaskine fra et centraliseret sted. Denne mulighed kunne især have interesse i en større industrialiseret sammenhæng.
- Af de mere avanceret udviklingsmuligheder kunne der tilføjes en automatisk frasortering af ikke befrugtet æg.

Konklusion 7

7.1 Personlige konklusioner

7.1.1 Andreas

I projektet har jeg haft succesoplevelser men også mange frustrerende tidspunkter, hvor der er oplevet fejl som ikke gav logisk mening. Dette kunne være overseelser, som gav frustrationer nok til at revidere éns program til tidligere versioner, og derved føle at ens tid havde været forgæves.

Projektet startede godt ud med fælles diskussioner. Disse opstod især fordi der ingen erfaring var indenfor emnet med udrugning af høns. Da projektet nåede udviklingsfasen opstod der mange problemer i form af at flere overvejelser blev bortkastet pga. der var nem adgang til andre løsninger. Dette gjorde, at der blev mistet overblik over produktets sammenhæng som kan have haft en indvirkning på vores integrationsfase af produktet.

Ting jeg vil tage med videre til senere semestre er, at prøve at holde sig til de oprindelige idéer som blev opsat i den lange udviklingsfase. Der skal også sættes mere kritiske tidsbegrænsninger på opgaver så fejlfyldte dele af projektet kan løses tidligt, eller kasseres i tide til at rette op på den spilde tid. Der skal også laves flere tests, især mindre integrationstests tidligt i forløbet vil kunne sikre en bedre afsluttende fase på projektet.

7.1.2 Dannie

Semesterprojektet har denne gang haft nye udfordringer, særligt som formand for en ny gruppe, da jeg ikke tidligere har arbejdet med størstedelen af gruppen. Det giver nogle problemer med at fordele arbejde ud fra folks stærke sider. Semesterprojektet på 3. semester er et frit valg, hvilket har givet muligheden for at lave et spændende produkt, men da projektet ikke har været godt nok defineret fra starten har der været mange tolkninger af forskellige opgaver, hvilket også giver mange forskellige løsningsforslag. Det har givet mange gode diskussioner under vejs, og gjort at mange af beslutningerne skulle tages fælles. Projektet havde været forløbet endnu hurtigere hvis definitionerne havde lagt fast. Yderligere vil man nemmere kunne dele projektet op i mindre dele. Under implementeringen har jeg stået for en sensor til at registrering af luftfugtighed og temperatur. Den første valgte sensor havde dog en besværlig protokol at lave program til, og set i bakspejlet skulle den have været håndteret anderledes. PSoC3 har ikke været muligt at lave program til sensoren da den kræver meget præcise signaler, dette kunne have været løst ved at

mikroprocessorer til at håndtere sensoren selv. Det var en tidskravende læringsproces, og kostede mange timer der kunne være brugt bedre. Den anden valgte sensor havde en standard I2C protokol og var derfor forholdsvis nem at håndtere.

Overordet set har projektet ikke været en succes for mit vedkommende. Jeg havde svært ved at holde rollen som formand (og Scrum master) særligt fordi denne process kræver ekstra meget arbejde, når folk deler sig op, for at løse forskellige opgaver, samtidigt med at jeg rodede med sensor, som jeg ikke kunne få til at virker. Dog har processen været meget lærerig.

7.1.3 Jens

Projektet har været spændende at arbejde med, men til tider også frustrerende, da vores arbejdsproces har været for rodet. Det blev forsøgt at køre Scrum som udviklings metode. Erfaringen med Scrum er at denne form for udviklingsværktøj desværre ikke egner sig til semesterprojekter på IHA pga. de mange side løbene fag og projekter, samt der ikke er resurser til at have en scrum master på projektet. I fremtidige projekter skal der lægges mere vægt på test. Der skal ligeledes sættes klare deadlines for kritiske elementer således at de ikke ender med at forsinke resten af projektet. Derudover skal der lægges en handlingsplan for hvad man skal gøre hvis disse ikke kan overholdes.

Af de positive ting, vil jeg klart fremhæve beslutningen om at bruge SVN og LaTex til at skrive både rapporten og dokumentation i. Disse udviklingsværktøjer er nogle der bestemt er værd at genbruge til senere projekter.

7.1.4 Mathias

Semesterprojektet dette semester er blevet tilgået med en lidt anden tilgang end tidligere. Denne gang har der været større fokus på at skulle få et funktionelt produkt ud af arbejdet. Dette har også gjort, at fokus på at benytte de forskellige udviklingsværktøjer(UML diagrammer, usecase osv.) ikke har været i fokus. Det har været spændende at skulle stå for hele DevKit8000 siden, både at se hvordan en kommunikation imellem to microcontrollere kan forsimples ved brug at SPI, samt at udvikle et program med et GUI til systemet.

Til at udarbejde programmet til DevKit8000 brugte jeg Qt-creator. Qt er ikke noget vi har beskæftiget os med før, så det har været rigtig interessant at tage et helt nyt udviklingsværktøj i brug, med store ændringer i C++ sproget, og selv skulle tillære sig det. Megen af interessen i forhold til Qt kommer specielt i det, at man kan se flere af de ting vi har lært i I3ISU blive anvendt i Qt frameworket. Dette er fx flertrådet programmering i C++, message distribution systemer og tråd synkronisering(mutex).

Endnu en gang forsøgte vi at anvende Scrum, men jeg synes personligt det fejlede. Specielt er det manglen på en person som kun agere Scrum master.

Alt i alt, har det for mig været et læreridt projekt, og det har været både frustrerende, men også læreridt ikke at kunne få produktet til at virke i den sidste testfase. Bl.a. er det

tydeligt, at vi generelt tester vores software for dårligt. Men den viden må man så taget med videre til næste semester.

7.1.5 Morten

At kalde projektet en katastrofe er måske at skrue lige lovligt meget op for dramatiske retorik, men jeg vil gå så langt som til at sige, at det nærmest er pinligt, at der på skrivende stund ikke foreligger et produkt, der kan mere end det kan. Dette til trods for hvad der kan betragtes som en flyvende start - tidsplanen blev lagt og overholdt igennem stort set hele perioden, så skete der en ren nedsmeltnings under design- og implementations-fasen. Noget, der til tider førte til en direkte ubehagelig stemning i gruppen.

Personligt synes jeg dette er det største problem, dette projekt har haft: Vi har ikke magtet at arbejde som et team. Der har manglet værktøjer og regler for, hvordan man vurderer hinandens arbejdsindsatser, og hvordan man evt. kan tillade sig at kritisere andres uden at det bliver opfattet som et personligt angreb. Samtidig blev der blev efter forsøgt at arbejde med Scrum, men det døde mere eller mindre indenfor den første uge efter at være sat i gang.

Det rent faglige udbytte af projektet har været som det burde: Nye udviklingsværktøjer blev taget i brug, nye teorier blev bragt ind i løsningerne, og nye platforme blev testet og integreret i produktet. Der var også rigeligt med muligheder for at gå ud over ens eget fokusområde.

7.1.6 Simon

Rugemaskinenes sekvensrutine blev succesfuldt implementeret og testet for hønseæg, således at de tidsbestemte handlinger for denne type æg udføres rettidig. F.eks. at rotere æggene hver tolvte time, og udlufte æggene en gang i døgnet.

Der blev udfærdiget en datakommunikations-protokol til SPI-kommunikationen mellem DevKit8000 og PSoC3. Denne blev implementeret og testet, således at brugeren via touch-interface på DevKit8000 kan styre samt kontrollere udrugningsprogrammet på PSoC3.

Det lykkedes ikke at gøre sekvensruten generisk, således at den kan håndtere udruninger af andre ægtyper end hønseæg. Det lykkedes heller ikke at færdiggøre implementeringen af fejlhåndtering, således at brugeren kan se fejlbeskeder. Begge dele pga. mangel på tid.

7.1.7 Stine

3. semesterprojektet har for mig personligt været en god oplevelse. Jeg synes at vores gruppe har fungeret fint sammen, og at de fleste har levet op til de krav, der er stillet. Da de fleste i gruppen også arbejdede sammen på 2. semesterprojektet, var det nemt at arbejde sammen og afprøve nye ting, som vi syntes skulle forbedres sidste semester. En overordnet mangel i vores projektarbejde var, at vi ikke var gode nok til at teste de enkelte undergruppers funktionaliteter sammen. Dette gjorde at den samlede test til sidst i forløbet blev en lang og udtrukken proces. Det er helt klart noget vi må tage til efterretning på de næste projekter vi skal lave.

Igen forsøgte vi at bruge Scrum som arbejds metode. På dette semester gik vi mere målrettet efter at holde de ugentlige statusmøder og at vi fik en bedre rød tråd igennem implementeringsfasen. For mig fungerede de ugentlige statusmøder rigtig godt, og jeg synes at gruppen var god til at give input og sparre med hinanden. Den røde tråd igennem forløbet synes jeg forsvandt lidt. I det hele taget blev Scrum ikke fulgt, udover statusmøderne. Tilgengæld synes jeg at vi skal blive bedre til at sørge for at deadlines bliver overholdt og bedre til at skride til handling, hvis man kan se at en deadline ikke kan nås.

7.2 Generel konklusion

Gruppens generelle konklusion bygger især på manglende brug af test i implementerings fasen. Der skal afsættes mere konkret tid af til at teste, og disse tests skal udførers i mindre dele, istedet for at teste hele produktet samlet.

Derudover skal der i fremtiden være mere fokus på at lægge klare deadlines for kritiske elementer i projektet. Der skal tages hurtigere beslutninger angående valg og fravalg af elementer, således at man, hvis man kommer i problemer, ikke sidder fast for længe, og dermed spilder en masse tid.

Det blev valgt at benytte Scrum arbejds metoden, men denne blev ikke fulgt. En af grundene til dette anses for at være de andre fag på semestret. For at Scrum kan fungerer optimalt skal der være en Scrum Master, hvis eneste rolle er at styre slagets gang. Da alle i gruppen forventes at have taget aktiv del i forløbet, er det svært at have en enkel person, som kun agerer Scrum Master. Derudover er der de fem fag på semestret, der også fylder. Dette gør, at der ikke er den samme mulighed for at udfører sprints og daglige møder, som man kunne håbe.

Litteratur

- [1] I3hal, exercise7 - ldd spi, gruppe: julie.
https://redmine.ih.a.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] Honeywell. I2c_comms_humidicon_tn_009061-2-en_final_07jun12.pdf. Technical report, Honeywell, 2012.
- [5] Honeywell. honeywell-sensing-humidicon-hih6100-series-product-sheet-009059-6-en.pdf. Technical report, Honeywell, 2013.
- [6] Kiatronics. *28BYJ-48 – 5V Stepper Motor.*