

# **Underskrifter**

---

Dato for aflevering: 17/12-2014

Gruppemedlemmer:

---

Bjørn Sørensen (BS) 201270432

---

Jakob Schmidt (JS) 201270402

---

Jesper Christensen (JC) 201270478

---

Lennart Balle (LB) 201371023

---

Mick Kirkegaard (MK) 201370970

---

Poul Overgaard (PO) 201370794

---

Simon Kirchheimer (SK) 201270090

---

Vejleder: Tore Arne Skogberg

# Indholdsfortegnelse

---

<b>Kapitel 1 Ordliste</b>	<b>5</b>
<b>Kapitel 2 Kravspecifikation</b>	<b>7</b>
2.1 Indledning . . . . .	7
2.2 Aktører . . . . .	9
2.3 Usecases . . . . .	10
2.3.1 UC1: Tilføj/fjern enhed . . . . .	11
2.3.2 UC2: Konfig . . . . .	12
2.3.3 UC3: Aktiver/deaktiver . . . . .	13
2.3.4 UC4: Databehandling . . . . .	13
2.3.5 UC5: Tjek status . . . . .	14
2.3.6 UC6: Udskriv log . . . . .	14
2.4 Ikke-funktionelle krav . . . . .	15
2.5 Grafiske brugerflade-skitser (BS) . . . . .	16
<b>Kapitel 3 Forundersøgelse</b>	<b>21</b>
3.1 Sprinkler . . . . .	21
3.1.1 Sprinkler valg . . . . .	21
3.2 Temperatursensor (SK) . . . . .	22
3.3 Bevægelsessensor (SK) . . . . .	23
3.3.1 Bevægelsessensor valg . . . . .	23
3.4 Fugtsensor (SK og LB) . . . . .	24
3.4.1 Fugtsensor valg . . . . .	25
<b>Kapitel 4 Systemarkitektur</b>	<b>27</b>
4.1 Domænemodel (JC) . . . . .	27
4.2 Hardware beskrivelse (HW) . . . . .	28
4.2.1 BDD . . . . .	28
4.2.2 BDD Master . . . . .	28
4.2.3 BDD Enhed . . . . .	29
4.2.4 IBD . . . . .	29
4.2.5 Grænseflade . . . . .	31
4.2.6 Signal beskrivelse . . . . .	32
4.3 Software beskrivelse . . . . .	33
4.3.1 Logical View (JC) . . . . .	33
4.3.2 Deployment View (JC) . . . . .	35
4.3.3 Implementation View (BS) . . . . .	36
4.3.4 Data View (BS) . . . . .	37
4.3.5 Kommunikationsprotokol (BS MK PO) . . . . .	38
<b>Kapitel 5 Hardwaredesign</b>	<b>41</b>

5.1	Prototypemodel . . . . .	41
5.2	Temperatur- og fugtsensor (JS LB) . . . . .	41
5.2.1	Foranliggende filter . . . . .	41
5.2.2	Præcisions- og støjhåndtering . . . . .	42
5.3	PIR-sensor (MK PO SK) . . . . .	44
5.4	Sprinkler-pumpe system (MK PO SK) . . . . .	45
5.4.1	230V/5V relæ . . . . .	45
5.4.2	Alpha 2 pumpen . . . . .	45
5.4.3	Relæ styring . . . . .	46
5.4.4	Driver . . . . .	47
5.5	Strømforsyning og tilslutnings-print (MK PO SK) . . . . .	47
5.5.1	Strømforsyning . . . . .	47
5.5.2	Tilslutningsprint . . . . .	51
5.6	SPI (MK PO SK) . . . . .	51
5.6.1	Kernemodul og Driver . . . . .	53
5.6.2	SPI API . . . . .	53
5.6.3	SPI Handler . . . . .	54
5.7	Hardware forbindelser (HW) . . . . .	55
5.7.1	Enhed (HW) . . . . .	55
5.7.2	Master (HW) . . . . .	56
<b>Kapitel 6</b>	<b>Softwaredesign</b>	<b>57</b>
6.1	Dataprotokol (BS) . . . . .	57
6.2	Applikationsmodeller (JC BS) . . . . .	58
6.2.1	Master . . . . .	58
6.2.2	Enhed . . . . .	64
6.3	Klassebeskrivelser . . . . .	66
6.3.1	Master (JC) . . . . .	66
6.4	Statisk Klassediagram Devkit8000 (JC) . . . . .	73
6.4.1	Enhed . . . . .	74
<b>Kapitel 7</b>	<b>Implementering</b>	<b>81</b>
7.1	Fugt- og temperatursensor (LB) . . . . .	81
7.2	Tilslutningsprint (SK) . . . . .	81
7.3	PSoC API (JS) . . . . .	82
7.4	230V relæ (PO) . . . . .	85
7.5	SPI (MK PO) . . . . .	86
7.5.1	SPI Driver . . . . .	86
7.5.2	SPI API . . . . .	87
7.5.3	SPI Handler . . . . .	90
7.6	PIR API (SK MK) . . . . .	94
<b>Kapitel 8</b>	<b>Modultest</b>	<b>97</b>
8.1	SPI (MK PO) . . . . .	97
8.2	PIR (SK) . . . . .	100
8.3	Tilslutningsprint (SK) . . . . .	100
8.4	230V relæ (PO) . . . . .	102

8.5 PSoC API . . . . .	102
8.6 Fugt- og temperatur-sensor print (JS LB) . . . . .	103
<b>Kapitel 9 Integrationstest</b>	<b>107</b>
9.1 SPI (MK PO) . . . . .	107
9.2 PIR (SK) . . . . .	110
<b>Kapitel 10 Accepttestspezifikation</b>	<b>111</b>
<b>Kapitel 11 Bilag (CD-indhold)</b>	<b>119</b>

# Ordliste 1

---

- AASH** Antal af samtidige hændelser
- API** Application programming interface
- Bunker** En fordybning på et golfhul fyldt med fint sand
- Devkit8000** Udviklingsboard fra Embest baseret på Linux (Beagleboard)
- Enhed** Autonomt undersystem som håndterer sensorer mv. baseret på PSoC-boardet
- Enhedstimer** En timer i Enhed som er defineret i ikke-funktionelle-krav
- ETX** End og TeXt ifm. seriell SPI kommunikation
- FT-sensor** Fugt og Temperatur sensor
- Golfbane** En golfbane består af 18 golfhuller
- Golfhul** Et golfhul består af teested, fairway, rough, green og hazards
- GUI** Graphical User Interface (Grafisk brugergrænseflade)
- IDLE tid** Up-time, oppetid eller standby tid
- KP** Komponentpakke. Del af systemet som indeholder en temperatur- og fugtsensor samt en sprinkler
- Master** Hovedenhed i systemet baseret på Devkit8000
- MTBF** Mean Time Between Failure
- PIR-sensor** Passive Infrared sensor
- PSoC** Udviklingsboard fra Cypress
- STX** Start of TeXt ifm. seriell SPI kommunikation
- Teested** Tee eller Teested er starten af et golfhul, hvor man slår sit første slag
- UI** User Interface (brugergrænseflade)
- Vandingsstatus** Indikerer hvorvidt vandingssprinkler er aktiv eller ej (1/0)



# Kravspecifikation 2

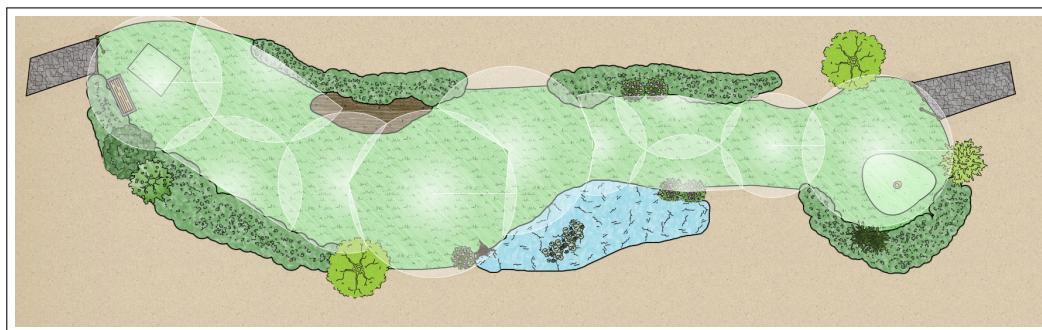
---

## 2.1 Indledning

Gruppen vil udvikle et system til vanding af golfbaner. I forbindelse med stigende temperaturer bliver det mere kritisk, at vandingen af golfbaner sker på de rigtige tidspunkter, for at holde banen spilbar. For at spare på ressourcerne er det også kritisk, ikke at spilde store mængder vand, på vanding af områder som ikke trænger til det.

Med et system af intelligente enheder der arbejder autonomt, men som modtager indstillinger for vandingen fra en masterenhed styres vandingen på hele Golfbanen centralt. På denne måde sparar man arbejdstid for greenkeeperen og vandingen sker kun når det er nødvendigt. Systemet kaldes for EasyWater8000.

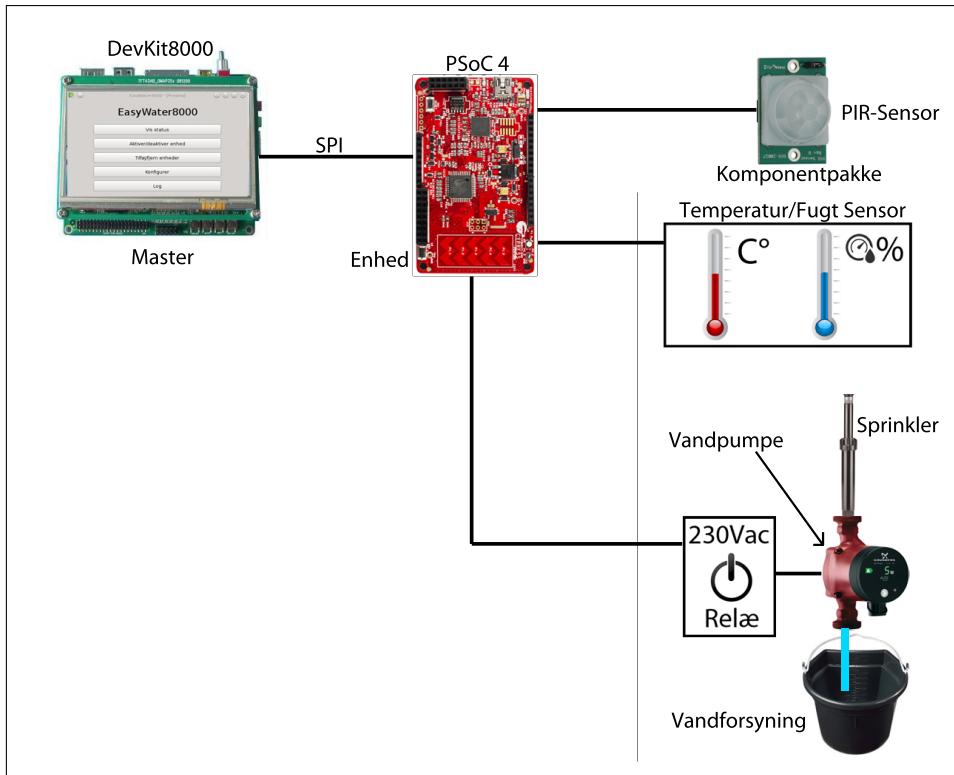
Normalt vil man placere en af disse enheder ved hvert hul på golfbanen og lave et netværk af sensorer lokalt til denne enhed. Enheden kan herved overvåge området og vande hvis nødvendigt. Alle enhederne forbinder til et netværk, som er koblet sammen med en Master. Grænseværdierne, for f.eks. jordfugtigheden, der afgør hvornår enheden skal påbegynde vanding kommer fra Master og styres igennem denne af greenkeeperen.



*Figur 2.1.* Overblik af golfhul med system

Figur 2.1 illustrerer et hul på en given golfbane. Dette hul er inddelt i 11 områder, med hver sin sprinkler. Spinklere er markeret ved halv til helcirkler på figuren. Til venstre på figuren ses tee-stedet med tilhørende indgang. Ved denne indgang er der placeret en bevægelsessensor, som sørger for at der ikke kan vandes på banen, når der er spillere.

Oversigt over blokkene i systemet kan ses i figur 2.2. De tre hovedbegreber for systemet er Master, som er hovedenheden baseret på Devkit8000. Enhed som er de autonome undersystemer på hvert hul, hvor hovedenheden er PSoC. Og Komponentpakker som er et begreb for pakken med temperatur- og fugtsensor samt sprinkler. Disse er distribueret ud på selve golfhullet.



**Figur 2.2.** Systemoverblik

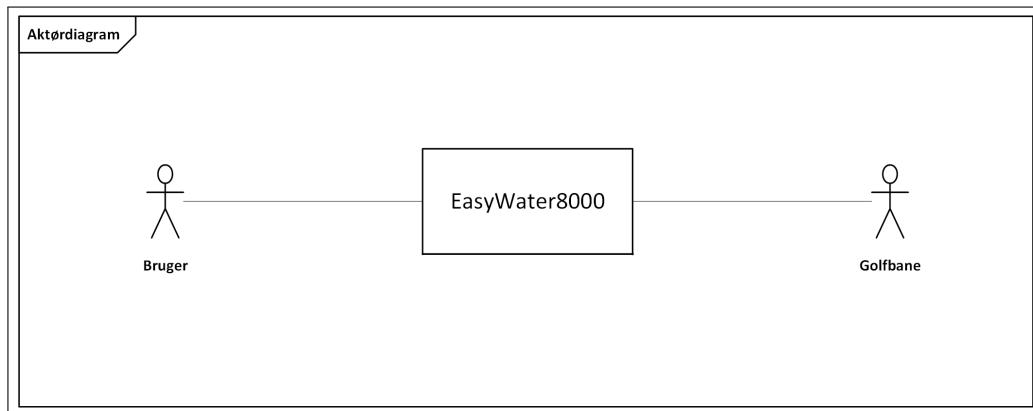
En Enhed består altså af:

1. Fugtighedssensor(er)
2. Temperatursensor(er)
3. Bevægelsessensor(er)
4. Sprinkler(e)
5. PSoC controller board

Fugt- og temperatursensorerne registrerer banens behov for vanding. Bevægelsessensoren registrerer om der er spillere på det pågældende hul. Sprinkleren sørger for vandingen når der skal vandes. Denne skjules nede i græsset og kommer op når vandingen skal være aktiv. PSoC controller-boardet bliver hjernen i Enheden. Denne styrer kommunikationen til Master og holder styr på de generelle funktioner for Enheden så som udlæsning af data, aktivering af sprinkler mv.

## 2.2 Aktører

Her beskrives systemets aktører. Disse vil blive refereret til, i de efterfølgende usecase-beskrivelser.

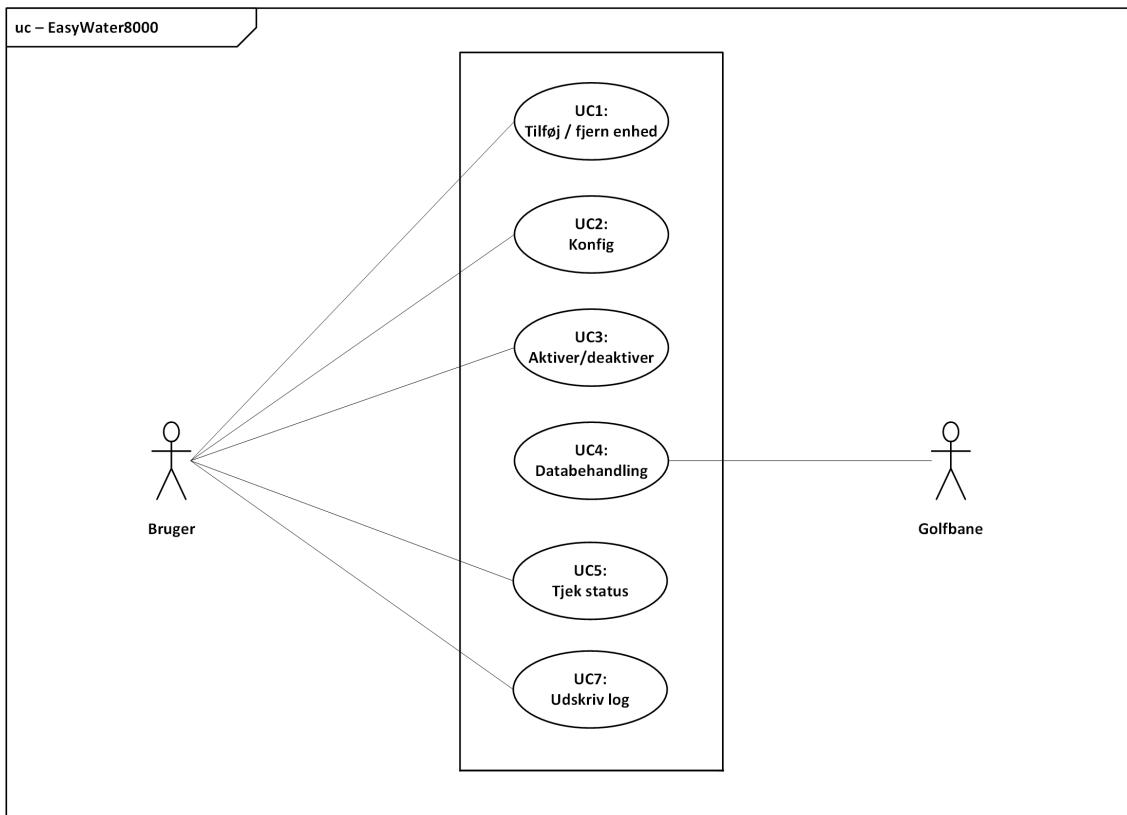


**Figur 2.3.** Kontekst diagram

Aktør navn	Beskrivelse
Bruger	Bruger vil normalt være greenkeeperen. Det er vedkommende som kontrollerer og betjener systemet. (Primær)
Golfbane	De almene omgivelser på golfbanen, som har indflydelse på systemets sensorer. Det indebærer temperatur, fugtighed og bevægelse i området omkring systemet. (Sekundær)

## 2.3 Usecases

Her følger en dybere beskrivelse af systemets opbygning og måde at virke på. Dette gøres med fulde usecase-beskrivelser hvor systemets virkning er beskrevet i detaljer.



*Figur 2.4.* Usecase diagram

### 2.3.1 UC1: Tilføj/fjern enhed

<b>UC1: Tilføj/fjern enhed</b>	
<b>Mål</b>	Bruger kan tilføje eller fjerne enheder fra systemet
<b>Initialisering</b>	Bruger
<b>Aktører og Stakeholders</b>	Bruger(Primær)
<b>Referencer</b>	Ingen
<b>AASH</b>	1
<b>Efterfølgende tilstand</b>	Ønsket Enhed er tilføjet eller fjernet fra systemet.
<b>Hovedforløb</b>	<ol style="list-style-type: none"> <li>1. Bruger vælger "Tilføj/fjern enhed" i hovedmenu</li> <li>2. Bruger vælger "Tilføj enhed"</li> <li>3. En liste af opsatte enheder præsenteres på skærmen             <ol style="list-style-type: none"> <li>a) Master beder bruger om at indtaste informationer</li> <li>b) Bruger indtaster hul-nr. og adresse på Enhed</li> </ol> <p><b>[Undtagelse 3b.a]</b> Indtastede værdier er ikke gyldige</p> </li> <li>c) Master tilføjer Enhed til systemet med default parametre</li> <li>d) Enhed forbindes til kommunikationsnetværket</li> <li>e) Master verificerer forbindelsen til Enhed og sender dato og tidspunkt</li> <p><b>[Undtagelse 3e.a]</b> Enheden kan ikke verificeres</p> <li>f) Enhed gemmer dato og tidspunkt</li> </ol> <ol style="list-style-type: none"> <li>4. Bruger vælger "Fjern enhed"</li> <li>5. En liste af opsatte enheder præsenteres på skærmen             <ol style="list-style-type: none"> <li>a) Bruger indtaster adresse på Enhed</li> <li>b) Master deaktivere Enhed</li> <li>c) Master sletter Enhed fra systemet</li> </ol> </li> <li>6. Master opdaterer liste med opsatte enheder</li> <li>7. Bruger kan returnere til hovedmenu eller opsætte ny enhed (Gå til UC1.2)</li> </ol>

...fortsat fra forrige side

<b>UC1: Tilføj/fjern enhed</b>	
<b>Undtagelser</b>	<p>3a.a Master viser fejlbesked omkring ugyldige værdier Gå til UC1.3a</p> <p>3e.a Master viser fejlbesked angående verificering af enheden</p> <p>3e.a Bruger kan forsøge igen (Gå til UC1.3e eller afbryde)</p>

### 2.3.2 UC2: Konfig

<b>UC2: Konfig</b>	
<b>Mål</b>	Ændre default parametre på en Enhed
<b>Initialisering</b>	Bruger
<b>Aktører og Stakeholders</b>	Bruger(Primær)
<b>Referencer</b>	Ingen
<b>AASH</b>	1
<b>Efterfølgende tilstand</b>	Ingen
<b>Hovedforløb</b>	<ol style="list-style-type: none"> <li>1. Bruger vælger ”Konfig” i hovedmenu</li> <li>2. Bruger vælger den Enhed, der ønskes omkonfigureret.</li> </ol> <p><b>[Undtagelse 2.a]</b> Bruger vælger afbryd</p> <ol style="list-style-type: none"> <li>3. Bruger ændrer parametre på den valgte Enhed.</li> </ol> <p><b>[Undtagelse 3.a]</b> Der indtastes ugyldige værdier</p> <ol style="list-style-type: none"> <li>4. Bruger vælger ”Gem”</li> </ol> <p><b>[Undtagelse 4.a]</b> Bruger vælger afbryd</p> <ol style="list-style-type: none"> <li>5. Master sender de nye indstillinger til den valgte Enhed.</li> </ol>
<b>Undtagelser</b>	<p>2.a Bruger vælger ”afbryd” Skærm viser hovedmenu</p> <p>3.a Der indtastes ugyldige værdier Skærm viser fejlbesked</p> <p>4.a Bruger vælger ”afbryd” Skærm viser Konfig-menu</p>

### 2.3.3 UC3: Aktiver/deaktiver

<b>UC3: Aktiver/deaktiver</b>	
<b>Mål</b>	At aktivere / deaktivere Enhed
<b>Initialisering</b>	Bruger
<b>Aktører og Stakeholders</b>	Bruger(Primær)
<b>Referencer</b>	Ingen
<b>AASH</b>	1
<b>Efterfølgende tilstand</b>	Enhedstilstand ændres aktiv/deaktiv
<b>Hovedforløb</b>	<ol style="list-style-type: none"> <li>1. Bruger vælger ”Aktiver/Deaktiver” i hovedmenu</li> <li>2. Bruger markerer Enhed ud fra liste af opsatte Enheder</li> <li>3. Bruger vælger ”Aktiver” eller ”Deaktiver” for at ændre Enheden [Undtagelse 3.a] Bruger vælger ”Afbryd”</li> <li>4. Master udskriver på skærmen, at ønsket Enhed er aktiveret eller deaktiveret</li> <li>5. Bruger vælger ”Afslut”</li> <li>6. Master viser hovedmenu</li> </ol>
<b>Undtagelser</b>	<p>3.a Bruger vælger ”Afbryd” Master viser hovedmenu</p>

### 2.3.4 UC4: Databehandling

<b>UC4: Databehandling</b>	
<b>Mål</b>	Indsamle data fra Golfbane og vande
<b>Initialisering</b>	System
<b>Aktører og Stakeholders</b>	Golfbane(Primær)
<b>Referencer</b>	Ingen
<b>AASH</b>	1 per Enhed
<b>Efterfølgende tilstand</b>	Data gemt i Enhed og logget i Master + evt. vanding

...fortsat fra forrige side

<b>UC4: Databehandling</b>	
<b>Hovedforløb</b>	<p>I et fast interval udføres nedenstående forløb på Enhed for hver tilkoblede Komponentpakke:</p> <ol style="list-style-type: none"> <li>1. Sensorernes data for temperatur og fugtighed udlæses fra Golfbanen</li> <li>2. Vanding startes på områder med for lave værdier [Undtagelse 2.a] Bevægelse registreret</li> <li>3. Enhed gemmer data i en buffer til senere udlæsning fra Master</li> </ol> <p>I et andet fast interval udføres nedenstående forløb på Master for hver opsatte Enhed:</p> <ol style="list-style-type: none"> <li>4. Master henter information fra bufferen i Enheden</li> <li>5. Master gemmer data i log</li> </ol> <p>Bevægelsessensoren aktiverer følgende forløb:</p> <ol style="list-style-type: none"> <li>6. Enhed modtager besked ved bevægelse</li> <li>7. Vanding deaktiveres 30 min</li> <li>8. Enhed gemmer hændelse i en buffer til senere udlæsning fra Master</li> </ol>
<b>Undtagelser</b>	2.a. Se punkt 6 og punkt 7 i hovedforløbet

### 2.3.5 UC5: Tjek status

<b>UC5: Tjek status</b>	
<b>Mål</b>	At se status på systemet
<b>Initialisering</b>	Bruger
<b>Aktører og Stakeholders</b>	Bruger(Primær)
<b>Referencer</b>	Ingen
<b>AASH</b>	1
<b>Efterfølgende tilstand</b>	Hovedmenuen vises på Master
<b>Hovedforløb</b>	<ol style="list-style-type: none"> <li>1. Bruger vælger ”Tjek status”</li> <li>2. Aktuel status vises på Master ud fra sidste logføring fra UC4</li> <li>3. Bruger vælger ”Tilbage”</li> </ol>

### 2.3.6 UC6: Udskriv log

<b>UC6: Udskriv log</b>	
<b>Mål</b>	Log udskrives på Masters skærm

...fortsat fra forrige side

<b>UC6: Udskriv log</b>	
<b>Initialisering</b>	Bruger
<b>Aktører og Stakeholders</b>	Bruger(Primær)
<b>Referencer</b>	Ingen
<b>AASH</b>	1
<b>Forudsætning</b>	Aktiv Master
<b>Efterfølgende tilstand</b>	Hovedmenuen vises på Master
<b>Hovedforløb</b>	<ol style="list-style-type: none"> <li>1. Bruger vælger ”Udskriv log” i hovedmenuen</li> <li>2. Log udskrives på Master</li> <li>3. Bruger vælger ”Tilbage”</li> </ol>

## 2.4 Ikke-funktionelle krav

### Brugbarhed

1. Opsætningen skal ske af autoriseret personale
2. UI skal kunne benyttes af bruger efter gennemlæst manual

### Pålidelighed

3. Mean Time Between Failure (MTBF): 2 år
4. Software IDLE tid: minimum 1 måned uden restart

### Ydeevne

5. Master skal kunne håndtere op til 18 enheder
6. Sprinkler skal kunne vande et areal af 360 grader med radius på 3.5 m
7. Der skal kunne tilføjes yderligere Enheder efter opsætning af systemet
8. Master skal hente data fra tilkoblede Enheder hvert 15. minut
9. Enhed skal hente data fra sensorer konstant

### Vedligeholdelse

10. Enheder skal være udskiftelige uden at det er nødvendigt at tilgå sensorer eller sprinkler
11. Sprinklere skal være let tilgængelige for personale

### Enheder

12. Enhed skal kunne operere autonomt efter denne er sat op fra Master
13. Enheder skal gemme log-information ifm. vanding og målinger, indtil Master udlæser denne

## Begrænsninger

- Der udarbejdes en skaleret prototype, da vi ikke har adgang til en hel golfbane
- Grundet tidsbegrænsninger udvikles der ikke et fuldt system
- Som minimum skal der produceres funktionel Master og én komplet Enhed
- Systemet udvikles således, at der til hver sprinkler tilhører en pumpe. Denne løsning vil ikke vælges i en produktionsudgave, her vil der være én vandforsyning, med et tryk stort nok, til at tilføre samtlige sprinklere den nødvendige mængde vand.

## 2.5 Grafiske brugerflade-skitser (BS)

Inden det endelige GUI udvikles laves nogle skitser som udviklingen læner sig op af. Ud fra UC beskrivelserne findes de skærmbilleder som skal vises på Master og skitseres.

Her følger korte beskrivelser af hver skitse samt skitsen selv.

### Startmenu

Figur 2.5 er den første menu brugeren kommer til. Her kan brugeren vælge hvilke funktioner vedkommende ønsker udført i systemet.



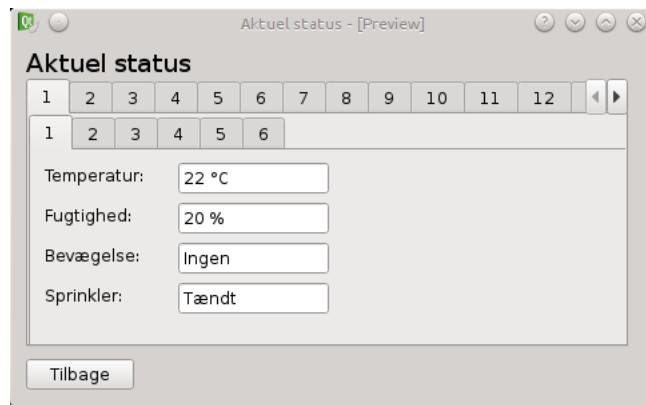
**Figur 2.5.** Skitse af startmenu på GUI

### Vis Status

På figur 2.6 vises den aktuelle status af systemets enheder. Den øverste række tal er tilkoblede Enheder. Den anden række tal er komponentpakker tilkoblet den enkelte enhed. Brugeren kan altså bevæge sig rundt og se den seneste udlæsning af data fra de tilkoblede enheder og deres komponentpakker.

### Aktiver og deaktiver enhed

Skitsen på figur 2.7 viser brugerens mulighed for at aktivere og deaktivere enkelte enheder i systemet. Der præsenteres en liste af opsatte enheder, hvor man kan markerer den ønskede enhed og vælge en funktion for denne.



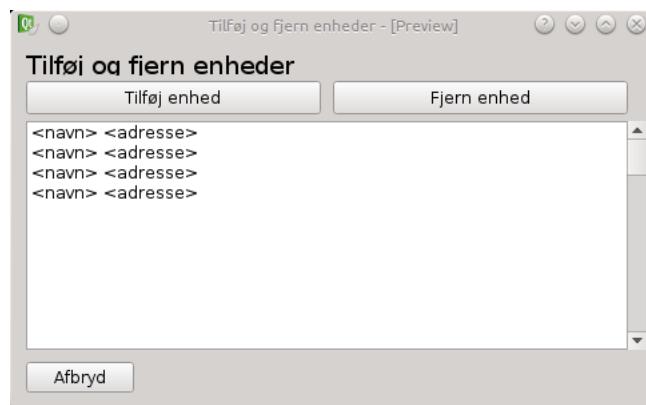
**Figur 2.6.** Skitse af "Vis status" på GUI



**Figur 2.7.** Skitse af "Aktiver og deaktiver enhed" på GUI

### Tilføj/fjern enheder

De præsenterede muligheder i forbindelse med at fjerne og tilføje enheder til systemet er vist på figur 2.8. Igen vises en liste af opsatte enheder. Det er muligt at tilføje en ny, eller markerer en eksisterende og fjerne denne.

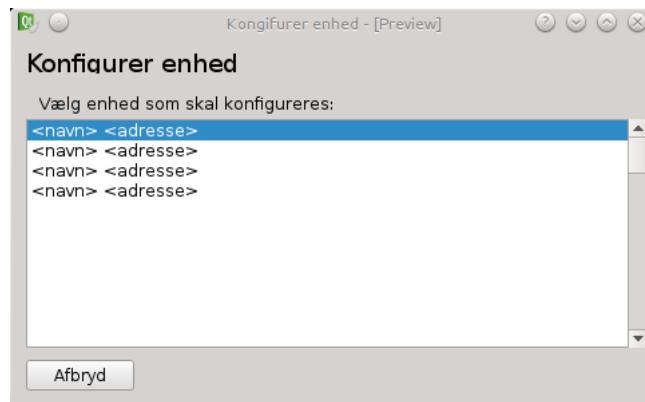


**Figur 2.8.** Skitse af "Tilføj/fjern enheder" på GUI

### Konfigurer

Når brugeren skal konfigurere en enhed bruges skitsen på figur 2.9. Her vælger brugeren hvilken enhed der skal konfigureres. Vinduet skifter til "Indstil parametre"-vinduet når en

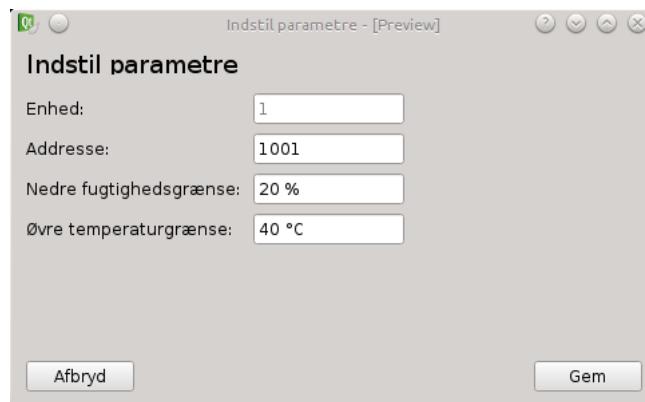
enhed markeres.



**Figur 2.9.** Skitse af "Konfigurer" på GUI

### Indstil parametre

Når brugeren har valgt en enhed som skal konfigureres vises skærmen på figur 2.10. Her kan brugeren indtaste parametrene for enheden og gemme dem i systemet.



**Figur 2.10.** Skitse af "Indstil parametre" på GUI

### Log

Loggen præsenterer de indsamlede data fra alle enhederne. Det er muligt at se alle data på en gang, som vist på figur 2.11, hvor den øverste fanerække er numre på opsatte enheder i systemet, eller man kan vælge de enkelte enheder og se deres forskellige komponentpakker, som vist på figur 2.12, hvor den anden fanerække er komponentpakker koblet op på systemet.

Log - [Preview]

**Log**

Alle	1	2	3	4	5	6	7	8	9	10	11
Dato	Enhed	KP	Temp	Fugt	Bevægelse	Sprinkler					
08-10-14 12:15	1		25	30%	Ingen	Tændt					
08-10-14 12:15	1		31	10%	Ingen	Tændt					
08-10-14 12:15	2		26	40%	Ingen	Slukket					
08-10-14 12:30	3		29	60%	Registreret	Slukket					
08-10-14 12:30	4		23	40%	Ingen	Tændt					

Afbryd      Generer graf

*Figur 2.11.* Skitse af "Log" for alle enheder på GUI

Log - [Preview]

**Log**

Alle	1	2	3	4	5	6	7	8	9	10	11
1	2	3	4	5	6						
08-10-14 12:15	25			30%		Ingen	T				
08-10-14 12:15	31			10%		Ingen	T				
08-10-14 12:15	26			40%		Ingen	S				
08-10-14 12:30	29			60%		Registreret	S				

Afbryd      Generer graf

*Figur 2.12.* Skitse af "Log" for enkelt enhed på GUI



# Forundersøgelse 3

## 3.1 Sprinkler

<b>Løsning</b>	1/2" Hunter PS
<b>Producent</b>	Hunter Industries / Agrometer A/S
<b>Tilslutning</b>	1/2" gevindmuffe
<b>Beskrivelse</b>	Simpel pop-up sprinkler som gemmes i græsset og kommer op når der påtrykkes vand. Kan indstilles til at vande i 45-360 grader eller kvadratisk.
<b>Krav</b>	Ekstern vandpumpe
<b>Fordele</b>	Simpel i opsætning og nem kaskadekobling af flere sprinklere hvor kun pumpens kapacitet skal udvides.
<b>Ulemper</b>	-
<b>Pris</b>	Cirkulær: 66,25 kr. Rektangulær: 77,50 kr.
<b>Link</b>	<a href="http://goo.gl/bzS1By">http://goo.gl/bzS1By</a>



### 3.1.1 Sprinkler valg

Hunter PS sprinkleren er en simpel løsning som nemt skjules på banen og er meget fleksibel når den skal tilpasses de enkelte golfhuller.

### 3.2 Temperatursensor (SK)

<b>Løsning</b>	LM75, Digital temperatursensor med two-wire interface.
<b>Producent</b>	National semiconductors
<b>Tilslutning</b>	I2C
<b>Beskrivelse</b>	Hardware modul der kan tilkobles enheden via I2C
<b>Krav</b>	Indgående kendskab til I2C og PSoC creator
<b>Fordele</b>	Der er blevet lavet en øvelse i GFV, der omhandler denne type sensor.
<b>Ulemper</b>	Den kommunikerer over I2C som er mere besværligt at arbejde med. Der skal konstrueres en "beholder" til sensoren så den kan tåle at komme ned i jorden.
<b>Pris</b>	Hentet fra værkstedet på IHA
<b>Link</b>	<a href="http://datasheets.maximintegrated.com/en/ds/LM75.pdf">http://datasheets.maximintegrated.com/en/ds/LM75.pdf</a>



<b>Løsning</b>	DS18B20 temperatursensor.
<b>Producent</b>	Dallas
<b>Tilslutning</b>	-
<b>Beskrivelse</b>	Vandtæt temperaturprobe der kan tilkobles enhver microcontroller med en enkelt digital benforbindelse
<b>Krav</b>	Indgående kendskab PSoC creator.
<b>Fordele</b>	Der er mulighed for at tilslutte flere af dem til den samme forbindelse, den kræver ingen ekstra HW.
<b>Ulemper</b>	Dallas 1-Wire-protokollen er lidt kompliceret, og kræver en del kode for at trække kommunikationen ud
<b>Pris</b>	89 kr
<b>Link</b>	<a href="https://elextre.dk/main.aspx?page=article&amp;artno=H15942">https://elextre.dk/main.aspx?page=article&amp;artno=H15942</a>



### 3.3 Bevægelsessensor (SK)

<b>Løsning</b>	HC-SR501, PIR-bevægelsessensor
<b>Producent</b>	Vides ikke
<b>Tilslutning</b>	-
<b>Beskrivelse</b>	En PIR-sensor, når bevægelse er registeret gives et højt eller lavt output(High 3.3 V /Low 0V), alt efter hvad jumperen er indstillet til
<b>Krav</b>	Indgående kendskab til PSoC creator
<b>Fordele</b>	Denne PIR-sensor kræver ingen ekstra HW. Ju-sterbar følsomhed og forsinkelse via potentiome-ter
<b>Ulemper</b>	-
<b>Pris</b>	Hentet fra værkstedet på IHA
<b>Link</b>	<a href="http://goo.gl/c6GyTL">http://goo.gl/c6GyTL</a>



#### 3.3.1 Bevægelsessensor valg

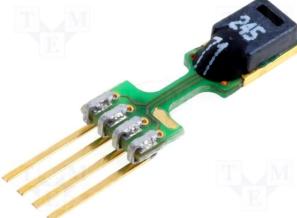
HC-SR501, PIR-bevægelsessensor. Opfylder alle de behov der er brug for, den er lille og kompakt dertil kan følsomhed/forsinkelse let indstilles.

### 3.4 Fugtsensor (SK og LB)

<b>Løsning</b>	Jord-hygrometer modul til Arduino
<b>Producent</b>	Vides ikke
<b>Tilslutning</b>	-
<b>Beskrivelse</b>	En jord-fugtighedssensor, når fugtigheden er høj giver output modulet en høj værdi og omvendt ved lav fugtighed
<b>Krav</b>	Indgående kendskab til PSoC creator
<b>Fordele</b>	Denne fugtighedssensor er klar til at blive sat i jorden, kræver ingen ekstra HW. Justerbar følsomhed via potentiometer
<b>Ulemper</b>	-
<b>Pris</b>	79 kr
<b>Link</b>	<a href="http://eleextra.dk/main.aspx?page=article&amp;artno=H11086">http://eleextra.dk/main.aspx?page=article&amp;artno=H11086</a>
	

<b>Løsning</b>	SHT21P - Fugtighed- og temperatursensor IC
<b>Producent</b>	Sensirion
<b>Tilslutning</b>	Analog
<b>Beskrivelse</b>	Hardware modul der giver et analog output (PWM)
<b>Krav</b>	Indgående kendskab til analog signaler og PSoC creator
<b>Fordele</b>	Fugtighed- og temperatursensor samlet i en lille kompakt enhed.
<b>Ulemper</b>	Der skal konstrueres noget HW og en "beholder" til sensoren så den kan tåle at komme ned i jorden
<b>Pris</b>	27.55
<b>Link</b>	<a href="http://www.farnell.com/datasheets/1847262.pdf">http://www.farnell.com/datasheets/1847262.pdf</a>
	

<b>Løsning</b>	SHT71P - Fugtighed- og temperatursensor IC
<b>Producent</b>	Sensirion
<b>Tilslutning</b>	I2C Variation
<b>Beskrivelse</b>	Hardware modul der kommunikeres med via en I2C variation
<b>Krav</b>	Indgående kendskab til I2C kommunikation og PSoC creator
<b>Fordele</b>	Fugtighed- og temperatursensor samlet i en lille kompakt enhed.
<b>Ulemper</b>	Der skal konstrueres noget HW og en "beholder" til sensoren så den kan tåle at komme ned i jorden
<b>Pris</b>	Hentes i værkstedet
<b>Link</b>	<a href="http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT7x_Datasheet_V5.pdf">http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT7x_Datasheet_V5.pdf</a>



### 3.4.1 Fugtsensor valg

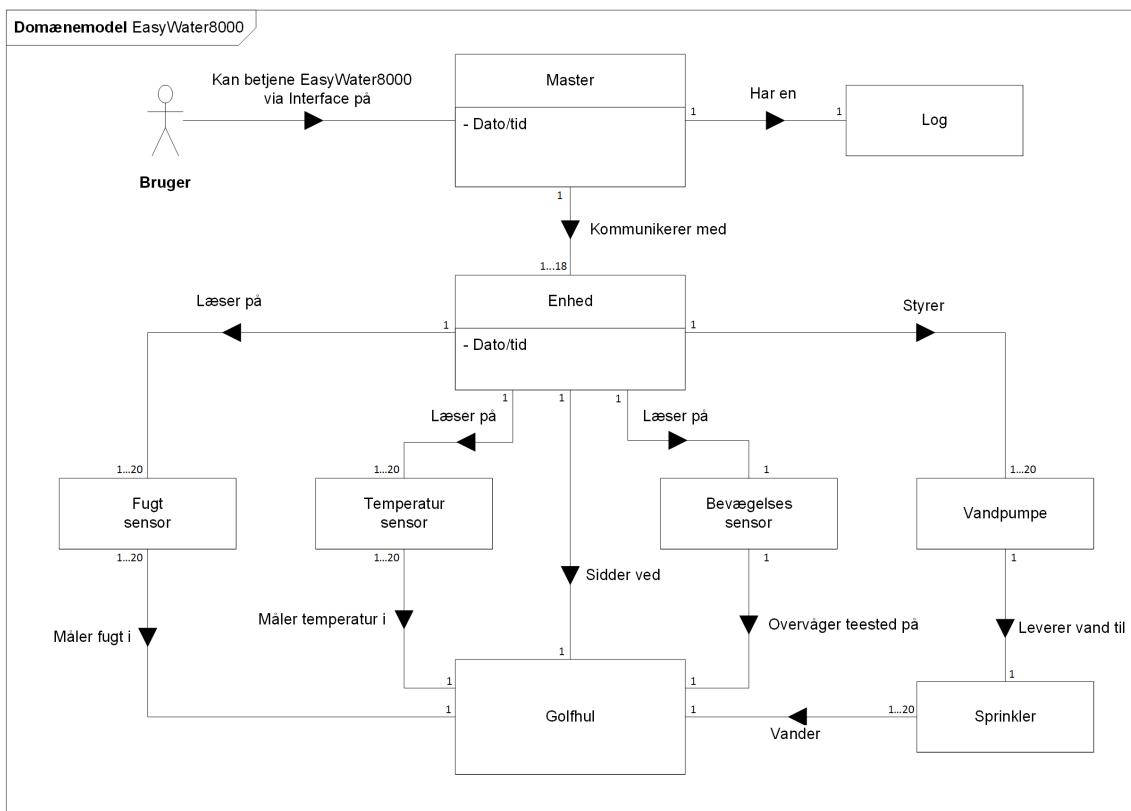
SHT21p. Har den store fordel at både fugt og temperatur er samlet i et, samt at den leverer et analog signal som er nemt at aflæse.



# Systemarkitektur 4

Systemarkitekturen består af en række forskellige diagrammer med dertilhørende tekst.  
Diagrammerne er opbygget efter SysML standarden<sup>1</sup>

## 4.1 Domænemodel (JC)



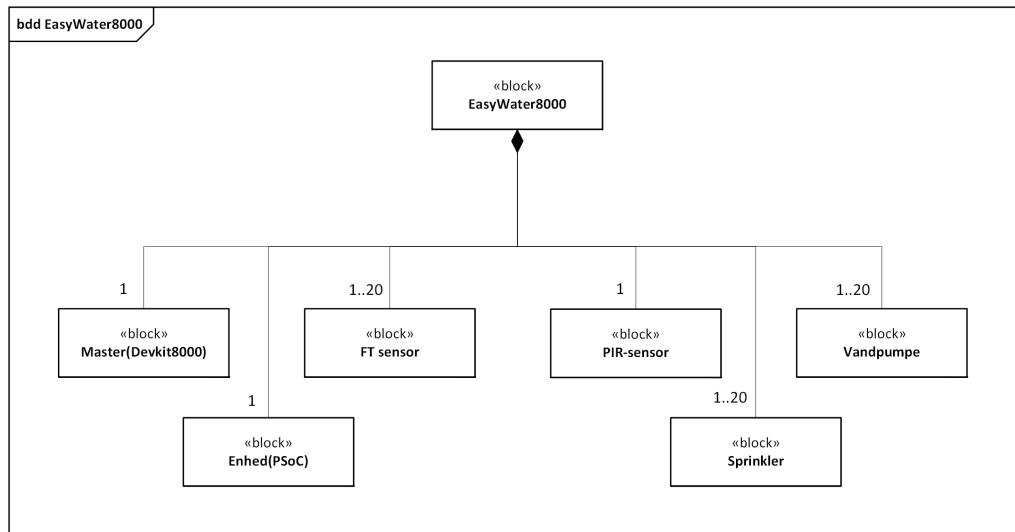
Figur 4.1. Domænemodel for EasyWater8000

Domænemodellen i figur 4.1 beskriver EasyWater8000 systemets funktionalitet og de forskellige deles indbyrdes sammenhæng.

<sup>1</sup>ISE undervisningen på 2. semester

## 4.2 Hardware beskrivelse (HW)

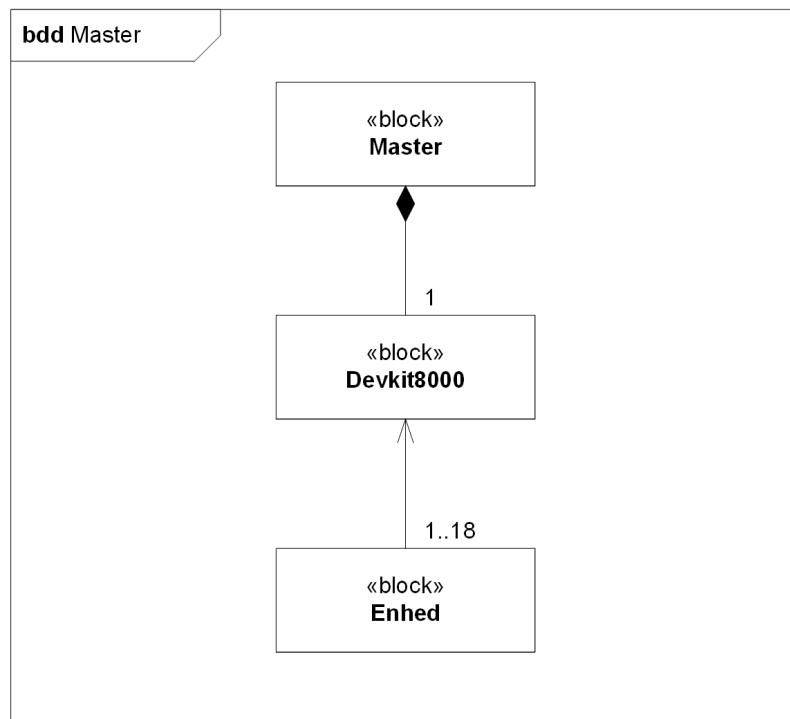
### 4.2.1 BDD



*Figur 4.2.* BDD for EasyWater8000

BDD diagrammet giver et overblik over hvad det samlede EasyWater8000 system består af, samt indbyrdes multiplicitet. FT sensoren er en kombineret fugt- og temperatursensor.

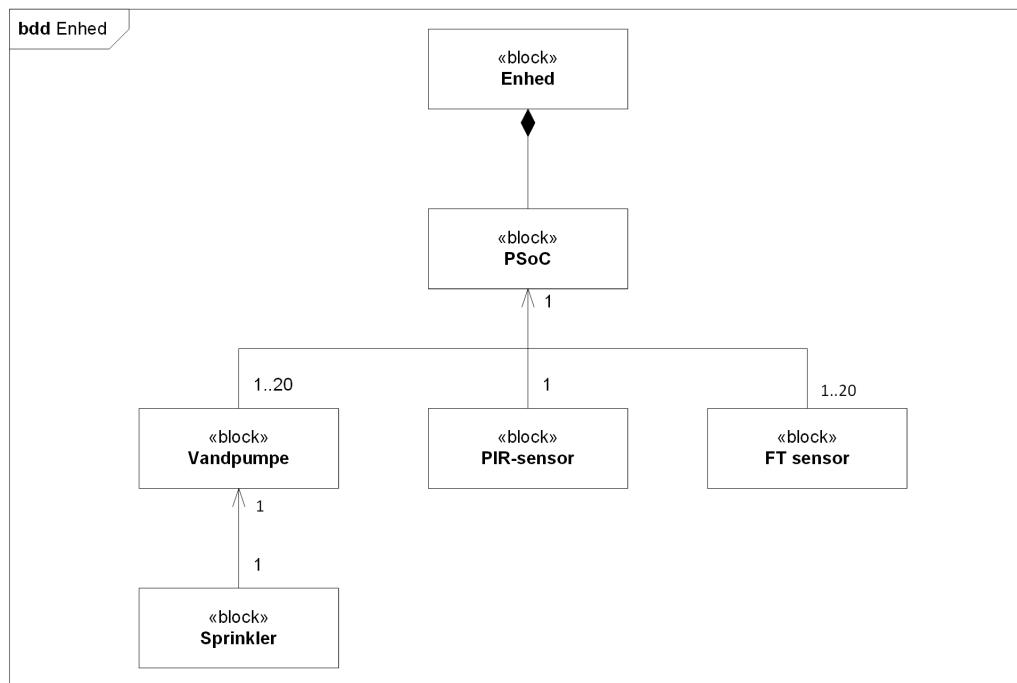
### 4.2.2 BDD Master



*Figur 4.3.* BDD Master

BDD diagrammet for Master, viser at Master består af ét Devkit8000, som kobles op med en til 18 Enheder.

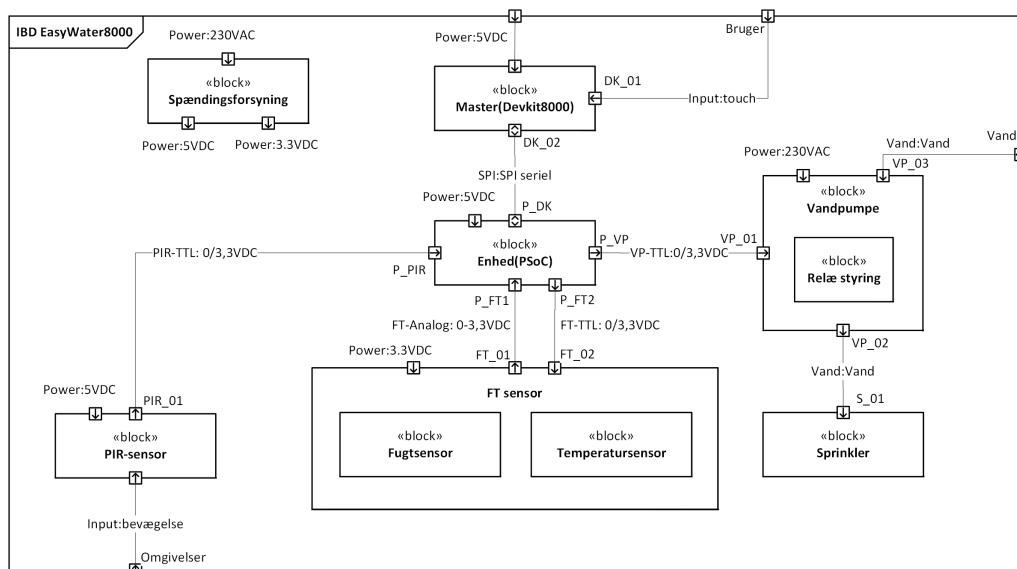
#### 4.2.3 BDD Enhed



*Figur 4.4.* BDD Enhed

BDD diagrammet for Enhed, viser at én Enhed består af én PSoC som kan kobles sammen med op til 20 Vandpumper, 20 FT sensorer (Fugt- og Temperatursensorer) samt 1 PIR-sensor. Sprinkler bliver i systemet koblet sammen med én Vandpumpe.

#### 4.2.4 IBD



*Figur 4.5.* IBD for EasyWater8000

IBD diagrammet giver et internt overblik over hvordan hele systemet er forbundet. Der ses hvilke typer signaler der bliver sendt imellem de forskellige blokke.

**Spændingsforsyning:** Spændingsforsyningens forbindelser er ikke påtegnet da dette ville give et uoverskueligt diagram, de er i stedet beskrevet med standardflowports. Spændingsforsyningen forsyner enhed (PSoC), FT sensor, PIR-sensor samt relæet i vandpumpen. Master(devkit8000) har sin egen spændingsforsyning.

**FT sensor:** Blokken beskriver at fugt- og temperatursensor er integreret i en chip.

**Relæ styring:** Blokken beskriver at vandpumpen består af et relæ, der står for at tænde/slukke for den.

### 4.2.5 Grænseflade

For at opnå forståelse for signalerne mellem blokkene laves en grænseflade beskrivelse, der beskriver de enkelte blokkes porte og hvilke signaler der løber imellem disse.

#### Blok beskrivelse

Til at beskrive blokkene nærmere anvendes tabeller som ses herunder. Her er hvert signal i en respektiv blok kommenteret og blokkens funktion er kort beskrevet.

**Tabel 4.1.** Tabel med beskrivelse af respektive blokke

Bloknavn	Funktion	Signaler	Kommentar
Master(Devkit8000)	Styreenhed der modtager input fra touchskærm og kommunikere serielt med Enhed(er)	Input:Touch	Masteren modtager inputs fra brugeren via touch
		SPI:SPI seriel	Seriell data kommunikation
Enhed(PSoC)	Enhed er grænsefladen til den fysiske verden igennem sensorer	SPI:SPI seriel	Seriell data kommunikation
		PIR-TTL	Signal fra PIR-sensor
		VP-TTL	Signal til vandpumpe
		FT-TTL	Signal til FT-sensor
		FT-Analog	Signal fra FT-sensor
FT-sensor	Måler temperatur og fugt	FT-Analog	Analog signal til Enhed
		FT-TTL	Signal fra Enhed
PIR-sensor	Detektere bevægelse	PIR-TTL	Signal til Enhed
Vandpumpe	Forsyner sprinkler med vand	Vand:Vand	Vand til Sprinkler
Sprinkler	Fordeler vand på golfbane	Vand:Vand	Vand til Golfbane
Spændingsforsyning	Forsyner EasyWater8000, Master har sin egen spændingsforsyning	Power:3,3VDC Max strømbelastning: 3,4A	Forsyning til FT-sensor
		Power:5VDC Max strømbelastning:2,2A	Forsyning til PIR-sensor, Enhed

#### 4.2.6 Signal beskrivelse

For at fuldende beskrivelsen af grænsefladen er der lavet en signaltabel som kan ses herunder. Hvert signal er beskrevet. Området et signal er defineret under, er også beskrevet. Blok og terminal indgår også.

**Tabel 4.2.** Tabel over signaler med terminaler

Signal-navn	Funktion	Område	Port 1	Port 2	Kommentar
SPI:SPI seri-el	Seriell kom-munikation		Master (DK_02)	Enhed (P_DK)	
PIR-TTL	TTL	0(lav)/ 3,3V(høj)	PIR-sensor (PIR_01)	Enhed (P_PIR)	Interrupt rea-gerer på høj flanke
VP-TTL	TTL	0(lav)/ 3,3V(høj)	Enhed (P_VP)	Vandpumpe (VP_01)	
FT-TTL	TTL	0(lav)/ 3,3V(høj)	Enhed (P_FT2)	FT-sensor (FT_02)	
FT-Analog	Analog	0-3,3V +/-10%	FT-sensor (FT_01)	Enhed (P_FT1)	PWM signal

## 4.3 Software beskrivelse

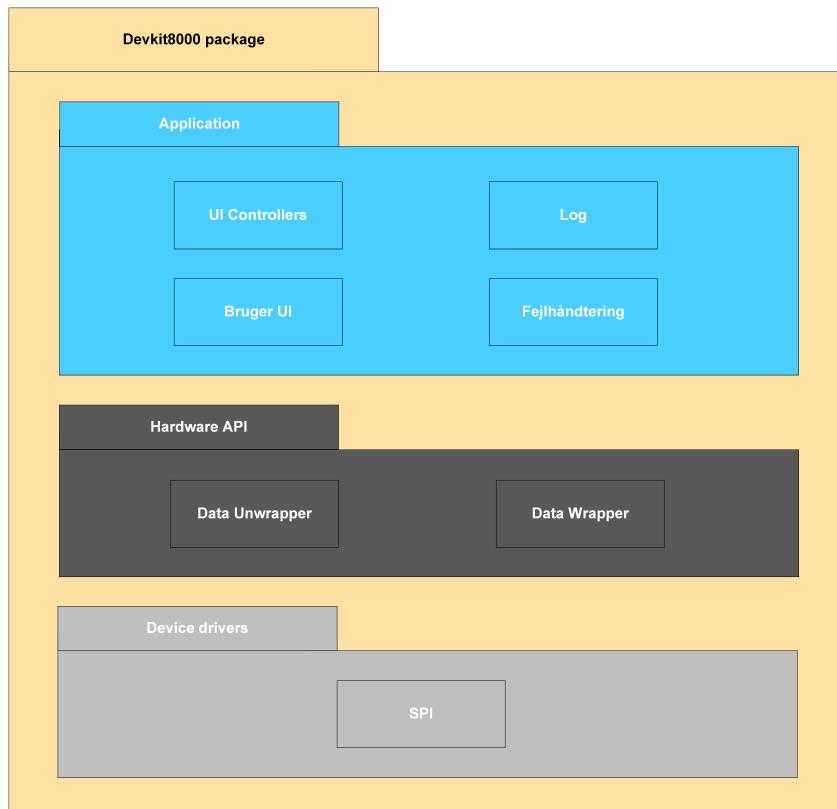
For at danne overblik over software-udviklingen inden det egentlige design, laves bruges N+1 modellen<sup>2</sup>. Denne beskriver fire faser som tager hånd om de overordnede ting inden for software, alt sammen med use-cases som den røde tråd. De fire faser er:

1. Logical View
2. Deployment View
3. Implementation View
4. Data View

Disse punkter er beskrevet i detaljer herefter.

### 4.3.1 Logical View (JC)

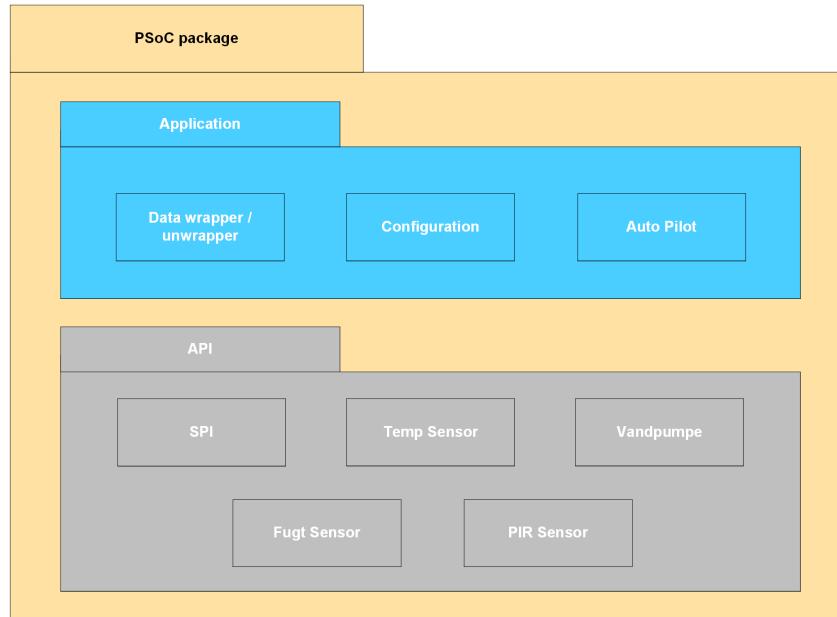
Logical View skal danne et overblik over hvilke softwarepakker der befinner sig på vores platforme. Blokkene inde i de respektive pakker kan sammen med domænemodellen hjælpe med at give et overblik over hvilke klasser og kernemoduler der skal bruges.



**Figur 4.6.** Logical view for Devkit8000 illustrerer hvilke softwarepakker der befinder sig på Masteren

Figur 4.6 illustrerer hvilke softwarepakker der ligger på Masteren. I bunden er *Device drivers*-pakken som håndterer SPI kommunikationen imellem Devkit8000 og PSoC. I midten ligger *Hardware API*-pakken som håndterer protokol-vedtægter ifm. kommunikationen. *Application*-pakken tager sig af alt UI samt log og fejlhåndtering.

<sup>2</sup><http://goo.gl/kU89oe> [2014-11-02]

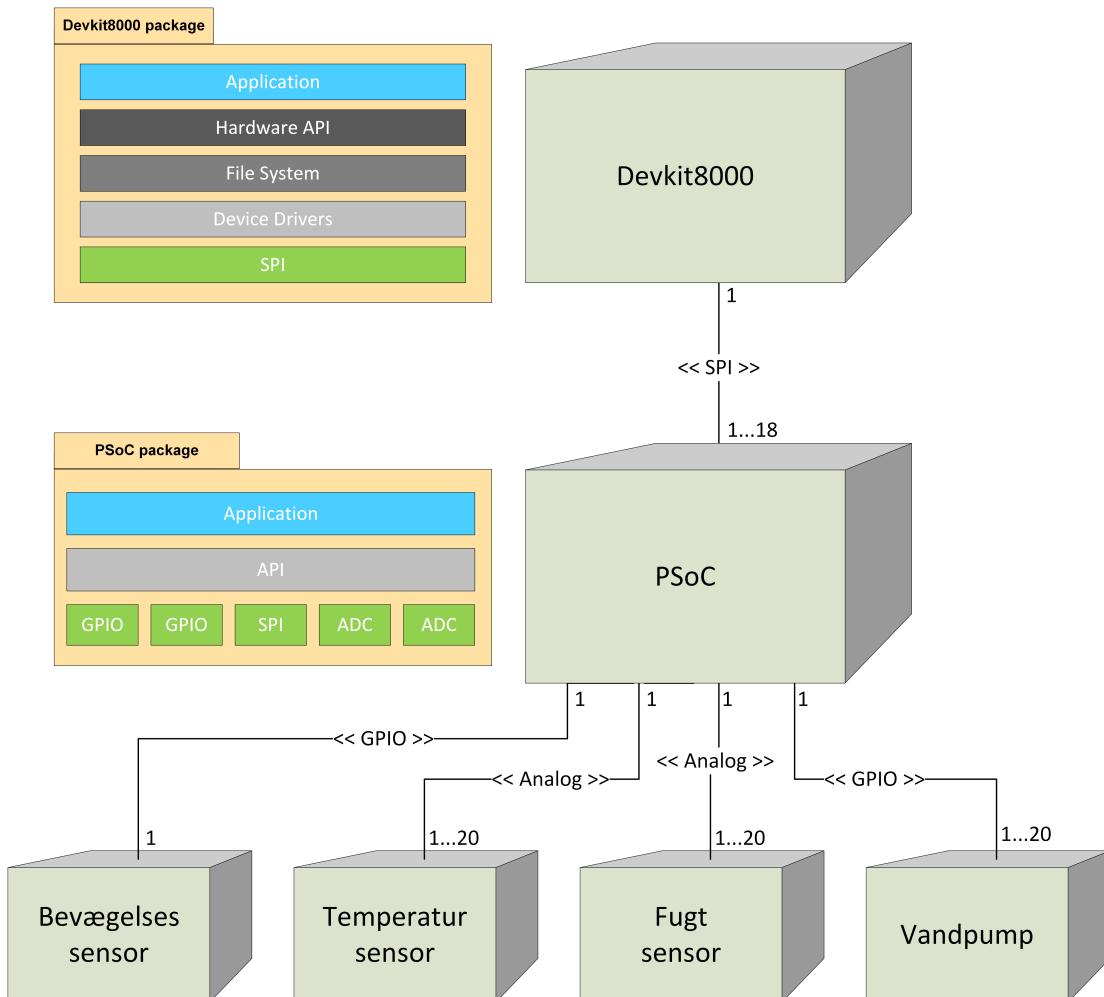


**Figur 4.7.** Logical view for PSoC illustrerer hvilke software pakker der befinder sig på enhederne

Figur 4.7 illustrerer hvilke softwarepakker der ligger på Enhederne. *API*-pakken består af den software som håndterer hardwaren, dvs. den tager imod input og får formateret det til noget brugbart for *Application*-pakken. *Application*-pakken håndterer den indsamlede data som den får fra sensorene igennem *API*-pakken. Denne data sammensættes iht. protokollen og sendes til *API* pakken som får det sendt til Devkit8000. Pakken skal også håndterer data fra Devkit8000 til at konfigurerer de parametre der styrer automatiseringen af vandingen.

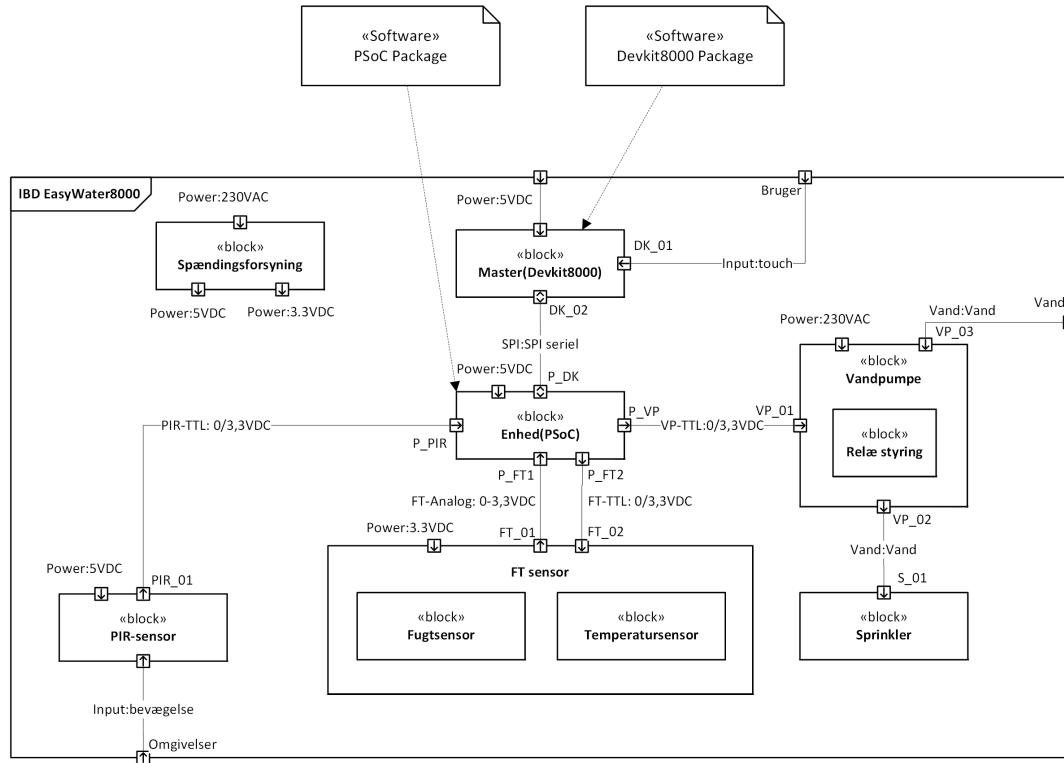
### 4.3.2 Deployment View (JC)

Deployment View skal illustrerer hvor hvilke lag af software der ligger på vores platforme. Devkit8000 kører Linux og har derfor flere softwarelag end PSoC'en.



**Figur 4.8.** Deployment model illustrerer de forskellige software og hardware(grønne) lag

Figur 4.8 viser hvilke lag der er i de respektive pakker og hvor pakkerne hører til. *File system* er en del af Linux systemet og ikke noget der skal implementeres.



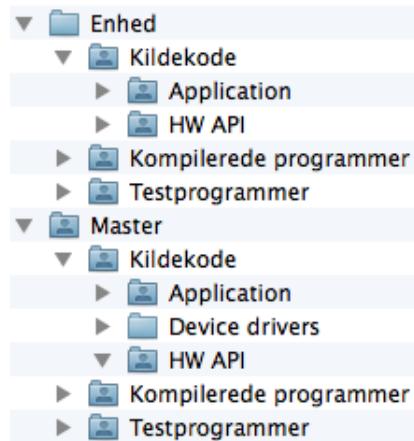
Figur 4.9. IBD med software packages

Figur 4.9 viser pakkernes placering på det interne blokdiagram, så man kan se præcist hvor softwaren kommer til at ligge på hardwaren.

### 4.3.3 Implementation View (BS)

Inden programmerne designes, fastlægges en struktur for kildekoden. På den måde er det nemmere for flere programmører at arbejde med delene i programmet samtidigt.

Strukturen skal være som vist i figur 4.10. Under mappen ”Kildekode” skal hver klasse have en mappe med der til hørende filer. Ligeledes med ”Testprogrammer” mappen, som indholder testprogrammer som verificerer funktionaliteten af de enkelte moduler. Mappen ”Kompilerede programmer” er til de endelige programmer.



Figur 4.10. Mappestruktur for software

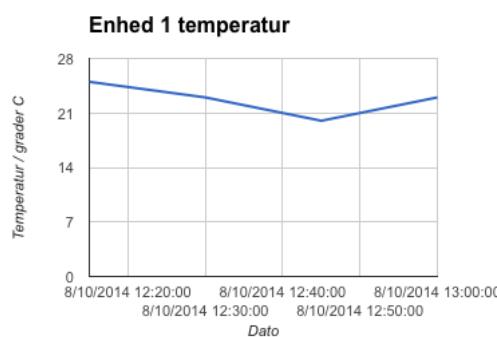
#### 4.3.4 Data View (BS)

I forbindelse med EasyWater8000s log skal der gemmes data på en nem og håndterbar måde. Det skal være muligt at gemme følgende data:

1. Tidsstempel
2. Temperatur
3. Fugtighed
4. Bevægelse
5. Vanding

Ydermere skal disse informationer gemmes for hver enhed. Så hvis der er 18 huller med i alt 18 enheder, skal ovenstående gemmes for alle 18 enheder.

Når informationen skal præsenteres for bruger skal det ske i en tabel som vist på figur 2.11 i afsnit 2.5, data for en enkelt enhed som på figur 2.12 i afsnit 2.5 eller på en graf så man kan se ændringer over tid som vist på figur 4.11.



**Figur 4.11.** Graf for temperatur for én enhed

Tabellen skal have en fane for hver opkoblet enhed. Her kan man se informationer fra hver enkelt enhed. Det bør også være muligt at se alle enheders informationer samtidigt.

Dataen struktureres i semikolon-separerede filer (.csv) på Master. Hver enhed har en fil hvor alle data er samlet med udgangspunkt i strukturen vist i liste 4.1.

**Listing 4.1.** Semikolon-separeret datafil til log af enheder

```

1 <enheds-nr>;
2 <KP-nr>; <dato>; <temperatur>; <fugtighed>; <bevaeglse>; <vanding>;
3 <KP-nr>; <dato>; <temperatur>; <fugtighed>; <bevaeglse>; <vanding>;
4 ...
5 <KP-nr>; <dato>; <temperatur>; <fugtighed>; <bevaeglse>; <vanding>;

```

Dette resulterer i en filstruktur som vist på liste 4.2 hvis der er koblet 18 enheder op på Master.

**Listing 4.2.** Filstruktur for logfiler på Master

```

1 <log>/
2   <enheds-nr1>.csv
3   <enheds-nr2>.csv
4   ...
5   <enheds-nr17>.csv
6   <enheds-nr18>.csv

```

Hæufigheden for målingerne og logningen er beskrevet i de ikke-funktionelle krav, 2.4.

### 4.3.5 Kommunikationsprotokol (BS MK PO)

En del data skal flyttes mellem Master og de tilkoblede Enheder. Her følger beskrivelsen af hvordan kommunikationen mellem Master og Enhed foregår. Den elektriske protokol som anvendes er SPI. Information omkring pakningen af data forefindes under Dataprotokol.

Opsætningen er som følger:

- Hastighed: 1 MHz
- SPI mode: 0 (CPOL 0 - CPHA 0)
- Antal bits: 1 char pr. transmission

Hastigheden er valgt på baggrund af I3HAL Exercise 7<sup>3</sup>. PSoC'en kan køre 8MHz, men der er ikke behov for så høj hastighed. Stabiliteten blev forbedret væsentligt ved at vælge en lavere hastighed.

SPI mode: 0 er valgt på baggrund ad default indstillinger.

CPOL = 0 vil sige at clocken er lav når den er passiv (aktiv-høj).

CPHA = 0 vil sige at data udlæses på rising-edge.

Der transmitteres en karakter pr. transmission dvs. 8 bits.

Ud fra UC-beskrivelserne er der identificeret følgende scenarier hvor der sendes data mellem Master og Enhed.

1. Master kontrollerer om Enhed er koblet til systemet (UC1)
2. Master sender parametre til Enhed (UC2)
3. Enhed aktiveres eller deaktiveres af Master (UC3)
4. Master beder om data fra Enhed (UC4)
5. Enhed sender data til Master (UC4)

#### Kommandoer til SPI-kommunikationen

*Tabel 4.3.* Kommandoer for SPI-kommunikation

ASCII	HEX	Funktion
'A'	0x41	Aktiver Enhed
'D'	0x44	Deaktiver Enhed
'P'	0x50	Parametre sendes til Enhed
'V'	0x56	Verifier Enhed i systemet
'L'	0x4c	Forespør logdata fra Enhed
'C'	0x4c	Write buffer og clearing af tx-buffer
'R'	0x4c	Læsning af data fra Enhed

#### Aktiver

Aktiver handler ikke på enhedsnummeret og derfor

<sup>3</sup>Hardware abstraktioner. Exercise 7: LDD with SPI. Øvelse med SPI Kommunikation

**Tabel 4.4.** Data-formatering for aktiver

Byte	<0>	<1>
MOSI	'A'	'C'
MISO	NULL	NULL

## Deaktiver

Deaktiver handler ikke på enhedsnummeret.

**Tabel 4.5.** Data-formatering for deaktiver

Byte	<0>	<1>
MOSI	'D'	'C'
MISO	NULL	NULL

## Verifier

Ved verificering af en Enhed sendes et enhedsnummer til Enheden, som der verificeres på i forhold til Enhedens enhedsnummer. I tabel 4.6 vises et eksempel hvor der verificeres på Enhed nr. 1.

**Tabel 4.6.** Data-formatering for verifier

Byte	<0>	<1>	<2>
MOSI	'V'	'R'	'C'
MISO	NULL	'1'	NULL

## Send parametre

Ved parameter indstilling skal der sendes følgende parametre til Enheden.

1. Enhedsnummer (1-18)
2. Øvre temperaturgrænse
3. Nedre fugtighedsgrænse

Parametrene skal sendes i ovenstående rækkefølge og resulterer altså i en kommando som vist i tabel 4.7, hvor der ønskes en øvre temperaturgrænse på 24,0 °C og en nedre fugtighedsgrænse på 30. Temperaturgrænsen er begrænset til 3 heltaal og én decimal. Fugtighedsgrænsen er begrænset til 3 heltaal.

**Tabel 4.7.** Eksempel på data-formatering for parametre

Byte	<0>	<1-5>	<6-8>	<9>
MOSI	'P'	'0' '2' '4' '.' '0'	'0' '3' '0'	'C'
MISO	NULL	NULL NULL NULL NULL NULL	NULL NULL NULL	NULL

## Log

Når Master skal hente log fra Enhed, sendes der et 'L' for at starte modtagelsen af loggen. Herefter laves 1 læsning for at udlæse data bufferens størrelse. Og herefter laves der én læsning ad gangen indtil der kommer enten et 'D' for Data eller et 'E' for Fejl. Ses i tabel 4.8 som et X antal gange.

**Tabel 4.8.** Data-formatering for log

Byte	<0>	<1>	<X>
MOSI	'L'	'R'	'R'
MISO	NULL	'SIZE'	'D'/'E'

### Log-data

Når der kommer et 'D' læses der iht. dataprotokollen 10 gange. I tabel 4.9 vises et eksempel på en udlæsning af temperaturen 28,5 °C, en fugtighed på 39, ingen vanding og ingen bevægelse.

**Tabel 4.9.** Data-formatering for log (data)

Byte	<1-10>									
MOSI	'R'	'R'	'R'	'R'	'R'	'R'	'R'	'R'	'R'	'R'
MISO	'0'	'2'	'8'	'.'	'5'	'0'	'3'	'9'	'0'	'0'
Indhold	'T'	'T'	'T'	'.'	'T'	'F'	'F'	'F'	'V'	'B'

### Log-error

Når der kommer et 'E' læses der iht. dataprotokollen 3 gange. I tabel 4.10 vises et eksempel på en udlæsning af fejl 13.

**Tabel 4.10.** Data-formatering for log (error)

Byte	<1>	<2>	<3>
MOSI	'R'	'R'	'R'
MISO	'0'	'1'	'3'

# Hardwaredesign

5

## 5.1 Prototypemodel

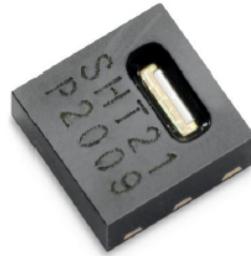
Hardwaredesignet tager udgangspunkt i den ønskede prototype, der beskrives herunder.

Prototypen skal bestå af en Master (DevKit8000) der via SPI kommunikerer med én Enhed (PSoC4). Denne enhed har koblet én komponentpakke, bestående af én PIR-sensor (HC-SR501), én kombineret fugt-, og temperatursensor (SHT21P), én 230V/5V relæstyring, én vandpumpe (Alpha2) samt én sprinkler (Hunter PS).

Prototypen sætter herved begrænsninger i forhold til ovenstående beskrivelser af systemarkitekturen.

De enkelte dele af prototypen designes herunder.

## 5.2 Temperatur- og fugtsensor (JS LB)

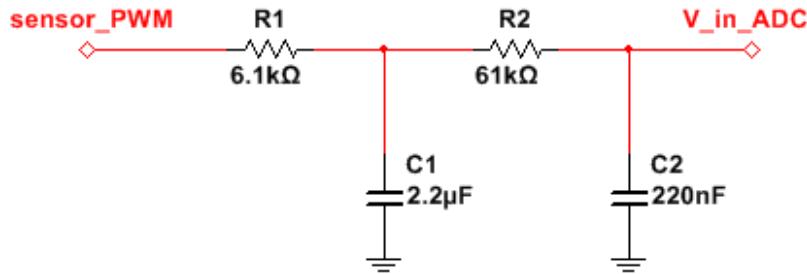


*Figur 5.1.* Fysisk afbildning af SHT21P

Til indsamling af temperatur- og fugtdata for golfhuller anvendes den kombineret temperatur og fugtsensor SHT21P. Denne er valgt ud fra at der således ikke behøves en sensor for hver af de to målinger, dermed sparres der et komponent. SHT21P sender en PWM ud som midles til en DC-spænding med et 2. ordens lavpasfilter. Denne spændingen sendes ind i en A/D-convertor i PSoCen hvor denne behandles i en funktion, for så at returnere en temperatur eller fugt. Et select ben på SHT21P angiver hvorvidt denne mäter temp. eller fugt. SCL HIGH(1) giver fugt output, SCL LOW(0) giver temperatur output.

### 5.2.1 Foranliggende filter

2. ordens filteret er sammensat af to lavpasfiltre. Det første filter, bestående af R1 og C1 bestemmer knækfrekvensen. Det efterfølgende filter, R2 og C2, er valgt som en faktor 10



**Figur 5.2.** Multisim tegninger af 2. ordens filter

af R1 og C1 hvilket vil resultere i et 2. ordens filter, men samtidig vil de to ikke påvirke hinanden og kan derfor forbindes direkte uden en buffer.

Filterets knækfrekvens ( $f_c$ ) er designet ud fra frekvensen på PWM signalet. Denne er opgivet til 120 Hz i databladet. Ved at designe filteret med en  $f_c$  der ligger væsentligt under 120 Hz, vil der opnås en stor dæmpning, således at signalet tilnærmer sig en fast DC-spænding svarende til middelværdien af PWM. PWM'en oscillerer i området VSS-VDD. Sensoren er forsynet med 3,3 VDC og VSS er forbundet til GND, hvilket giver en PWM i området 0-3,3 V. DC-spændingen efter filteret er testet ved 10 % dutycycle og 90 % dutycycle hvilket gav henholdsvis 211 mV og 2610 mV efter filteret. Ligningerne for udregning af fugt og temp., er opgivet som en funktion af PWM'en, er begge lineære og der kan derfor regnes med et step i volt på grader celsius. Funktionerne herfor er opgivet i databladet for SHT21P. På figur 5.3 kan det ses at kurven for fugtighed går fra -6 % til 120 %, denne er dog kun defineret ved 0 til 100 %.

### 5.2.2 Præcisions- og støjhåndtering

Ved 10% duty cycle og 90% duty cycle vil DC-spændingen give henholdsvis 33 mV og 2970 mV  $\Leftrightarrow \Delta V = 2970$  og en temperatur differens gående fra -29 ... 111 °C  $\Leftrightarrow \Delta T = 140^\circ\text{C}$ . Da denne er en lineær funktion kan den beregnes et step:

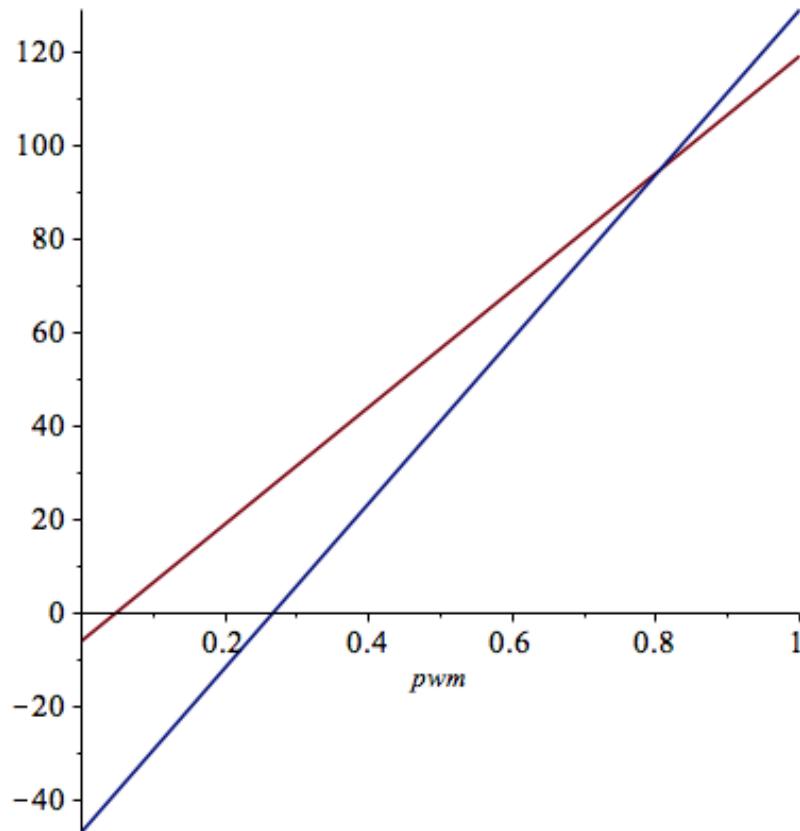
$$\text{step} = \frac{2970}{140} = 21 \frac{\text{mV}}{\text{°C}} \quad (5.1)$$

$$f_c = \frac{1}{2\pi * R1 * C1} = \frac{1}{2\pi * 6,1 * 10^3 * 2,2 * 10^{-6}} = 12\text{Hz} \pm 5\% \quad (5.2)$$

Da komponenternes værdi har en afvigelse på  $\pm 5\%$ .  $f_c$  ligger således en dekade under 120 Hz og da filteret er designet som et 2. ordens filter, vil det dæmpe 40 dB pr. dekade. Amplituden vil da være dæmpet 100 gange og tilbage er den tilnærmede DC-værdi.

$$\text{Gain} = 20 * \log_{10}(x) \Leftrightarrow 40\text{dB} = 20 * \log_{10}(x) \Leftrightarrow x = 100 \quad (5.3)$$

Da det er en dæmpning der er tale om, vil gain være lig med -40 dB hvilket vil give en x-værdi på 0,01 som er det samme som 100 gange dæmpning.



**Figur 5.3.** Plot af fugt(rød kurve) og temperatur(blå kurve) som funktion af pwm

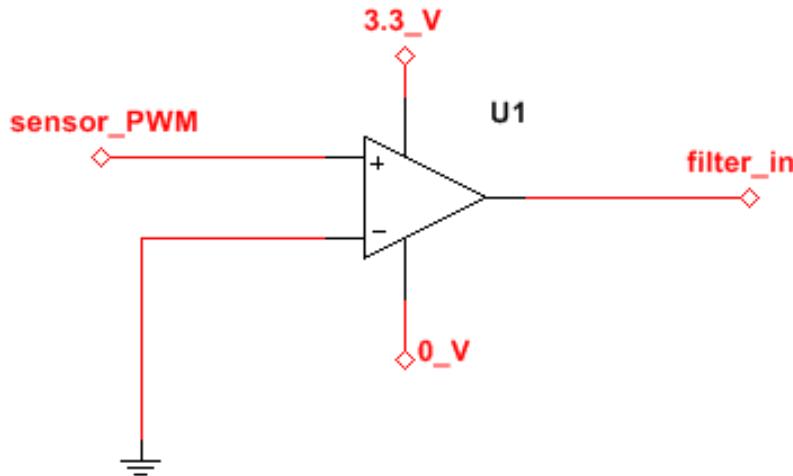
Grundet at værdien kun er tilnærmet skyldes at der stadig vil være oscillation tilbage fra PWM'en og det antages at denne svarer til 33 mV. Der vil altså være en usikkerhed i DC-værdien svarende til 1,6 °C alene fra filteret. De 33 mV er en ideel antagelse ud fra filteret, men nøjagtigheden herfor kan ikke garanteres og der laves derfor en antagelse.

Dette kan løses i softwaren ved evt. at tage et antal samples og så midle de værdier der er målt med simple gennemsnitsberegning.

$$V_{inADC_{avg}} = \frac{V_{in1} + V_{in2} + \dots + V_{inN}}{N} \quad (5.4)$$

hvor  $N =$  antal samples

En anden støjfaktor der er værd at tage hensyn til er filterets belastning af udgangen på sensoren. Denne kan have indflydelse på området hvori PWM'en går højt og lavt. Det kan altså ikke med sikkerhed siges at signalet vil gå fra 0 V til 3,3 V, men kan have et offset og en evt. dæmpning. Denne problematik kan løses ved indsættelse af en OpAmp, forbundet som en komparator, med kendt spændingsforsyning i det ønskede område, lige før filteret. Denne vil sørge for, at signalet kommer til at ligge i det valgte område. En OpAmp fra HC14 familien ville kunne klare denne opgave.

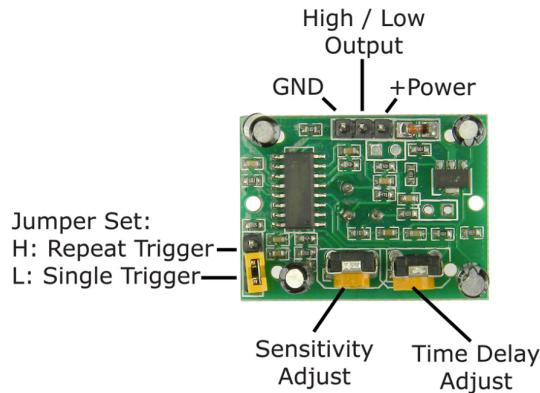


**Figur 5.4.** OpAmp til udbedring af evt. støj forårsaget af lavpasfilter

### 5.3 PIR-sensor (MK PO SK)

Den PIR-sensor(HC-SR501) der er valgt til projektet har 3 ben, +POWER, OUTPUT og GND, 2 potmetre til at indstille sensitiviteten og forsinkelse samt en jumper til at indstille triggeren. Forsyningsspændingen skal ligge mellem 5V - 20V, og den har et strømforbrug på 65 mA. Forsinkelse kan justeres til mellem 0.3 - 5 min.

Output fra PIR-sensoren afgiver et TTL signal på 3.3V ved bevægelse og 0V ved stilstand.



**Figur 5.5.** Billede af selve printet og dets indstillingsmuligheder

### Driver

Driveren der skal drive PIR sensoren på Enhed skal kunne registrere bevægelse via en get-metode der returnere 1(høj) ved bevægelse og 0(lav) ved stilstand.

### Pseudokode

```

1 CY_ISR(PIR){
2     Call movement method here
3 }
4

```

```

5  Init interrupt(PIR) on PIR-pin
6
7  for(;;){
8  Detect HIGH(1) on PIR-pin
9 }

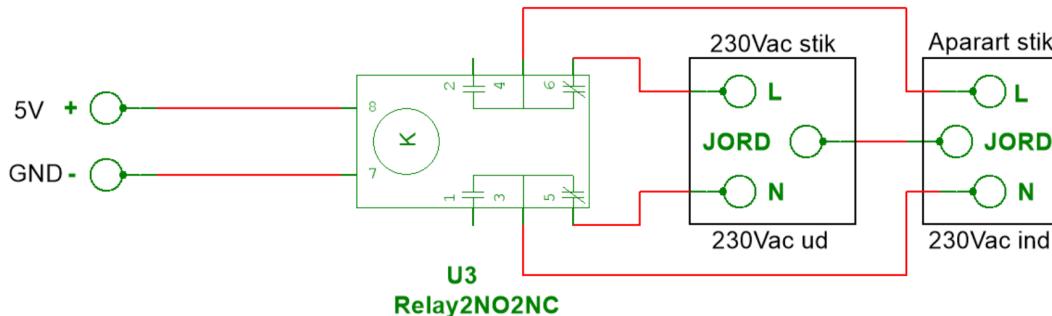
```

## 5.4 Sprinkler-pumpe system (MK PO SK)

Sprinkler-pumpe systemet består af en Hunter PS Pop-up sprinkler og en Alpha 2 pumpe fra Grundfoss, disse forbindes med passende slanger og fittings. For at styre sprinkleren, skal Alpha 2 pumpen kunne tændes og afbrydes. Dette gøres via et 230V/5V relæ. Dette relæ styres via én pin Enheden.

### 5.4.1 230V/5V relæ

For at 230V/5V relæet kan blive en realitet, er det pålagt, at dette bygges i en lukket kasse og godkendes af en elektronikværksteds-ansvarlig. Kassen som bygges har et 230Vac apparatstik ind og et 230Vac stikkontakt ud. For at skabe forbindelse/afbrydelse af denne 230Vac strøm benyttes et relæ, dette relæ styrers af 5V. Når 5V tilføjes relæets spole, klikker relæet og der skabes gennemgang fra apparatstik til stikkontakt. Dette lukkede kredsløb bygges som sagt ind i en lukke kasse med gennemsigtigt låg. Relæet der benyttes er et Finder 40.52s. Figuren 5.6 viser kredsløbet for dette lukkede kredsløb.



*Figur 5.6. 230V/5V relæ*

### 5.4.2 Alpha 2 pumpen

Alpha 2 pumpen skal blot tilsluttes 230Vac + jord. Der medfølger et special stik til selve pumpen, dette forbindes til en 230V stikprop, med 3-ledet kabel (Leder, Nul og Jord). Stikproppen kan nu sættes i 230V/5V relæets 230V stikkontakt.



**Figur 5.7.** Alpha2 Cirkulationspumpe fra Grundfoss

### 5.4.3 Relæ styring

I følge databladet for Finder 40.52s (figur 5.8), kræver relæet 100mA ved 5V forsyning, som databladet viser.

**DC coil data - 0.5 W sensitive (types 40.31/51/52/61)**

Nominal voltage $U_N$ V	Coil code 7.005	Operating range		Resistance R Ω	Rated coil consumption I at $U_N$ mA
		$U_{min}^*$ V	$U_{max}^{**}$ V		
5	7.005	3.7	8.8	50	100

**Figur 5.8.** Datablad for Finder 40.52s - 100mA, side 7

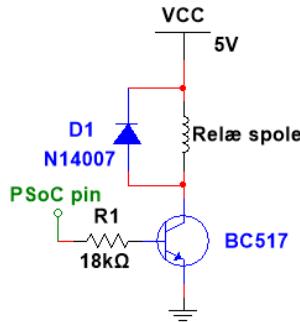
PSoC'ens udgang giver hverken strøm eller spænding nok til at trække relæet. PSoC'en afgiver 3,3V og relæet kræver 3,7V-8,8V. For at opnå dette benyttes en transistor til styringen af relæet. Relæet kræver 100mA, derfor skal transistoren opfylde dette krav, det gør BC517 den kan leverer 500 mA.

R1(Ibase) modstanden er indsat for at begrænse den strøm der går til transistorens base ben, ifølge databladet for transistoren må Ibase strømmen max være 100 mA. Spændingen ved PSoC'en er 3,3V (Spændingen kan ske at falde en smule hvis den bliver belastet, men det har ingen betydning for kredsløbet) og spændingen ved transistoren base ben er 1,4V, det giver en spænding på 1,8V over modstanden. Ibase strømmen er beregnet udfra ohmslov og der er valgt en modstand på 18 kohm, der begrænser strømmen til 0,1 mA, se formel 5.5.

Strømforstærkningen er faktisk 30000 gange i transistoren, så en strøm på 3 uA burde være nok, men der er taget et valg om at strømmen skal være 0,1 mA. BC517 er oplyst til at have et spændingsfald på 1V i maettet tilstand, det efterlader 4 V til relæet, hvilket også er nok da relæet virker indenfor 3,7V - 8,8V, se figur 5.8. Der er foretaget en observation af at hvis modstanden blev alt for stor ville relæet tage længere tid om at

trække, det er ikke lykkedes at finde en fornuftigt forklaring på dette. Dioden er indsatt for at beskytte transistoren, den beskytter mod tilbageløbende strøm fra spolen i relæet.

$$I_{base} = \frac{3,3V - 1,4V}{18k\Omega} = 0,1mA \quad (5.5)$$



**Figur 5.9.** BC517 opsætning

Når PSoC'ens pin til BC517 transistoren går høj (3,3V), så skabes der forbindelse mellem collector og emittier på transistoren, herved er der 4V over relæets spole og relæets kontaktsæt klikker herved.

#### 5.4.4 Driver

Driveren der skal håndtere Sprinkler-pumpe systemet, skal kunne aktivere/deaktivere det forudbestemte sprinklerrelæ. Metoden modtager en adresse parameter samt en on/off parameter. Herefter sætter metoden en af den forudbestemte pin på Enheden høj/lav (3.3V/0V). Herefter søger relæstyringen og 230V/5V relæet for hhv. tænde/slukke for sprinkleren.

#### Pseudokode

```

1 void controlSprinkler(int address, bool on_off){
2     Manage Sprinkler address
3     Activate/deactivate corresponding Sprinkler
4 }
```

## 5.5 Strømforsyning og tilslutnings-print (MK PO SK)

### 5.5.1 Strømforsyning

For at frigøre sig fra laboratoriets strømforsyninger udarbejdes egen strømforsyning. Vha. en 230Vac/12Vdc switch-mode konverter og egnede spændingens-regulatorer designes en strømforsyning til netop dette projekt. Masteren (DevKit8000) har sin egen strømforsyning og denne ændres ikke. Enheden (PSoC4) skal forsynes med 5V dc via USB. Relæstyringen skal forsynes med 5V dc. PIR sensoren skal forsynes med 5-20 V dc. Temperatur/fugt sensoren skal forsynes med 3,3V dc.

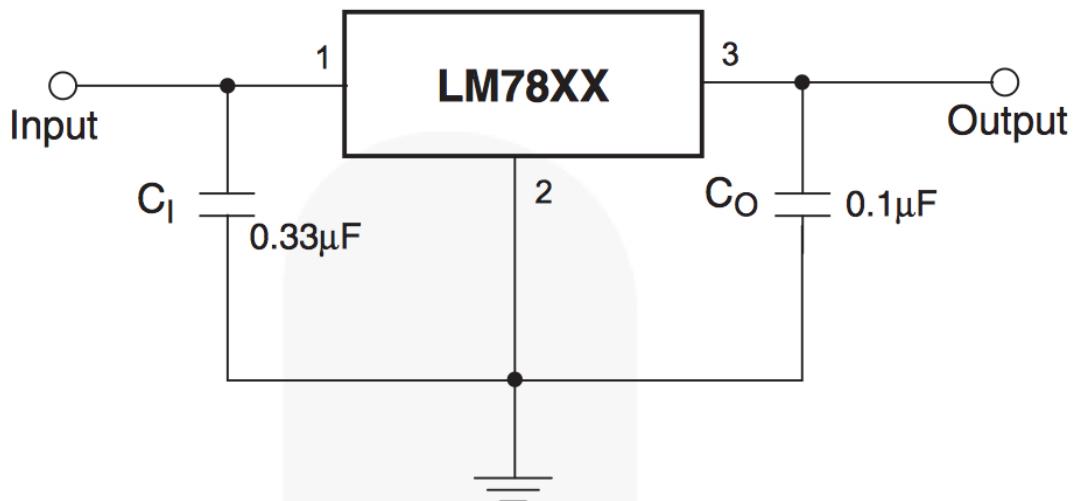
230Vac/12Vdc switch-mode konverteren er fra en gammel computer. Denne kan give 2 amperre og er forsynet med et DC stik (12V) og en stikprop (230Vac). Denne transformer bruges som den er.

For at regulere fra de 12 V til 5 V benyttes en spændingsregulator. LM7805 er beregnet netop til dette formål. Ud fra databladet 5.10 ses forbindelsen af LM7805'ern. Databladet foreskriver også at regulatoren kan give et output på max 1A ved korrekt køling.

Uden køling må der max afsættes 1 W i serieregulatoren, da dens arbejdstemperatur ved 1 W ligger på 100°C. I formel 5.6 er der fortaget en effektberegning på LM7805. Spændingsforskellen imellem  $V_{DC}$  og  $V_{OUT}$  ganges på det samlede strømforbrug. Strømforbruget er fundet i samtlige datablade for hver komponent plus  $I_X$  som er strømmen der løber ned i kondensatoren, den er opgivet i databladet. Effektudregningen i 5.7 viser at ved fuld belastning er det nødvendigt at køle, det er der gjort med køleplader.

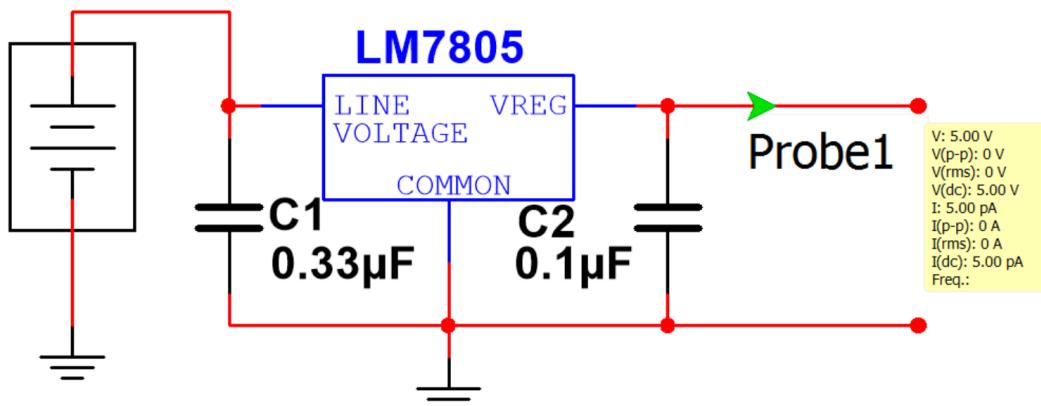
$$P = (V_{DC} - V_{out}) * (I_{samlet} + I_X) \quad (5.6)$$

$$P = (12V - 5V) * (0,6A + 10mA) = 4,27W \quad (5.7)$$



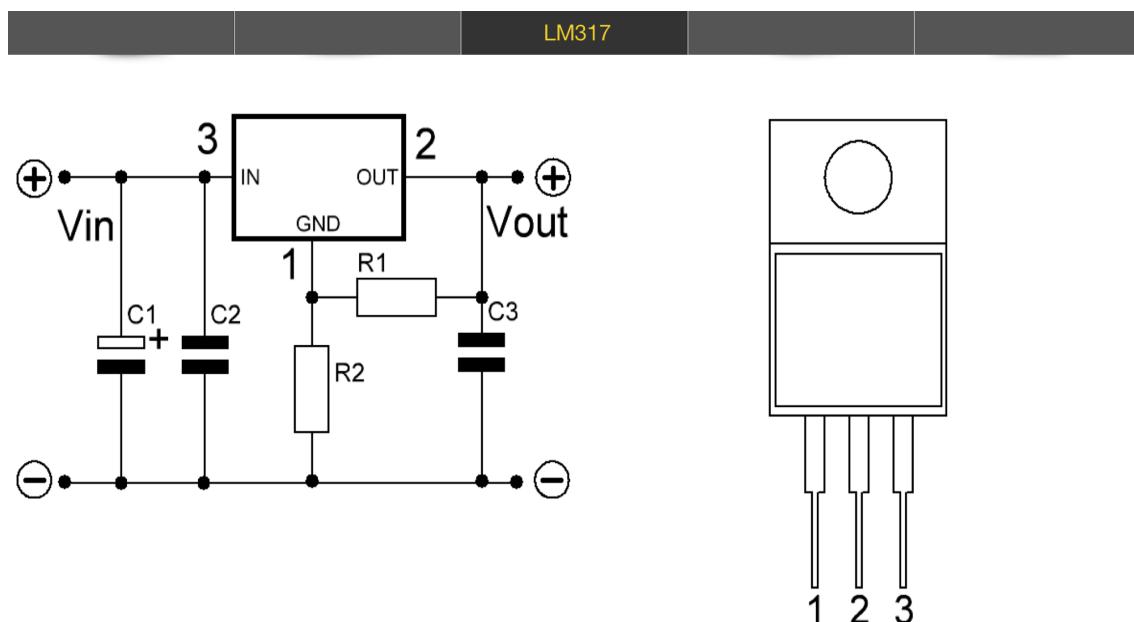
**Figur 5.10.** Forbindelse af LM7805 - Side 18/24 LM7805.pdf

Spændingsregulatoren LM7805 er opbygget i multilisim, sammen med 12V dc forsyningen. Kredsløbet er simuleret og dette viser at outputtet er de ønskede 5 V.

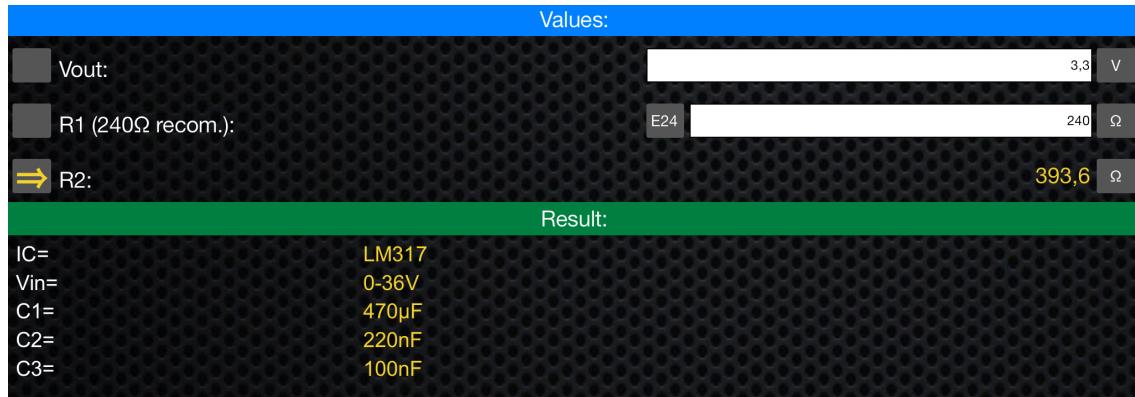
**12V DC 2A****Figur 5.11.** LM7805 Simulering

For at opnå 3,3V benyttes en justerbar spændingsregulator LM317. Denne regulator kan levere 1,2V-33V 3A. Det er fugt,-temperatursensoren der skal forsynes med 3,3V/180uA, det er LM317'erns opgave.

LM317 er slæjt op iPad appen "Electronic Toolbox", vha. denne gives hvordan opsætningen af denne skal være for at opnå 3,3V

**Figur 5.12.** Kredsløb for LM137

Figur 5.13 viser hvorledes appen beregner en værdi for modstanden  $R_2$ , når  $V_{out}$  er sat til 3,3V og modstanden  $R_1$  anbefales til 240 ohm. I følge appens beregning skal  $R_2$  være 393,6 ohm.



**Figur 5.13.** Kredsløb for LM137

Det er muligt at beregne udgangsspændingen ud fra følgende formel 5.8 som er oplyst i databladet for LM317 og i Analogteknik<sup>1</sup>. Strømmen  $I_Q$  er strømmen der løber fra ben 1, den er mindre end 100 uA og typisk på 50 uA, de 1,25 V er spændingen imellem udgangen og referencen. Det giver følgende resultat i formlen 5.9.

$$V_{out} = 1,25V * \left(1 + \frac{R2}{R1}\right) + I_Q * R2 \quad (5.8)$$

$$V_{out} = 1,25V * \left(1 + \frac{390\Omega}{240\Omega}\right) + 50uA * 390\Omega = 3,3V \quad (5.9)$$

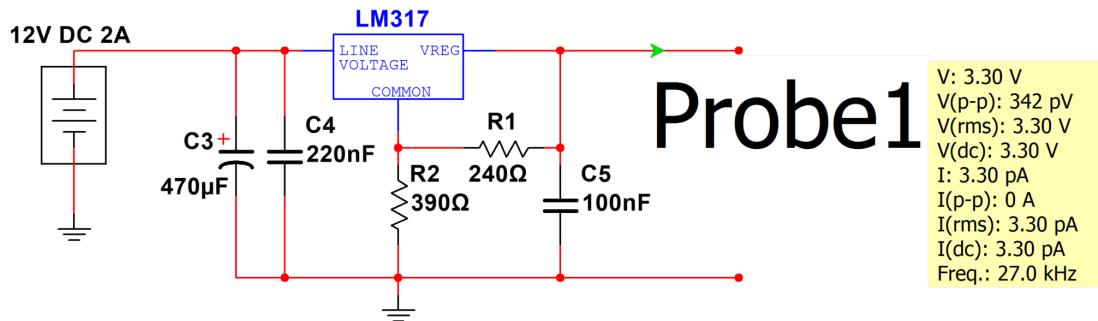
I formel 5.11 er der fortaget en effektberegning på LM317. Den bliver belastet med FT-sensoren, som har et strømforbrug på 180 uA plus  $I_X$  som er strømmen der løber ned i kondensatoren og modstanden, den er opgivet i databladet. Effektudregningen viser at det ikke er nødvendigt med køling, men der er dog fortaget køling med køleplader alligevel. Det skyldes at det er altid en god ide at køle på et komponent hvis der er mulighed for det, det sikre bl.a længere levetid og berøringssikkerhed.

$$P = (V_{DC} - V_{out}) * (I_{FT-sensor} + I_X) \quad (5.10)$$

$$P = (12V - 3,3V) * (180uA + 10mA) = 0,09W \quad (5.11)$$

Figur 5.14 viser simuleringen af LM317 spændingsregulatoren. Outputtet er 3,3V som forventet. Der simuleres med en modstand R2 på 390 ohm og ikke 393,6 ohm.

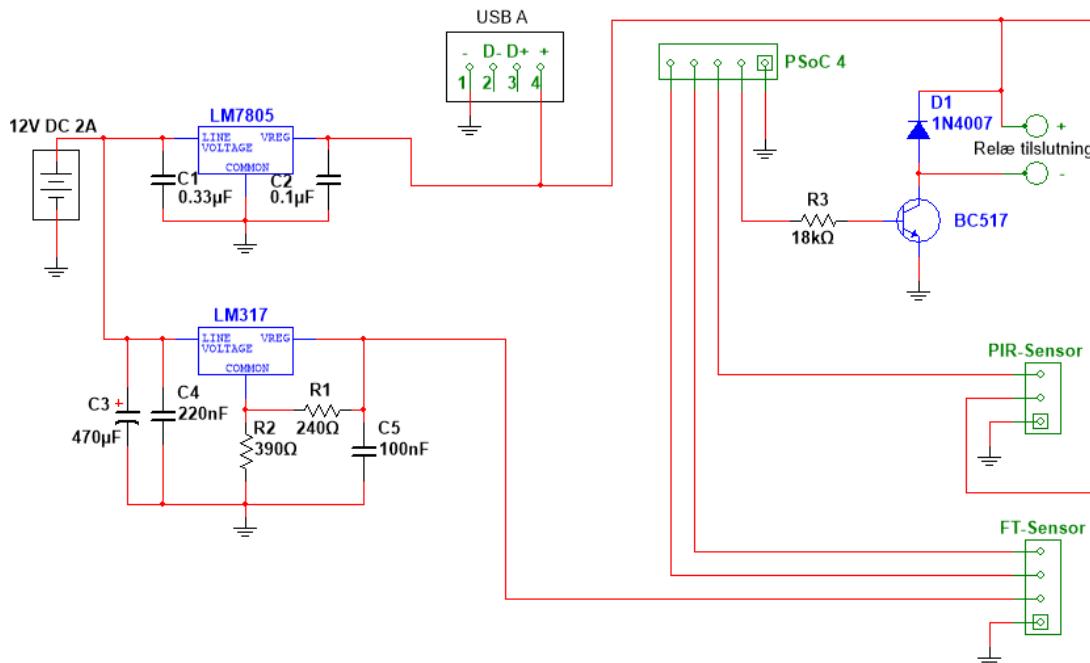
<sup>1</sup>Analogteknik, s 259, afsnit 6.8 Serieregulator



Figur 5.14. Simulering i Multisim ud fra Electronic Toolbox appens værdier

### 5.5.2 Tilslutningspunkt

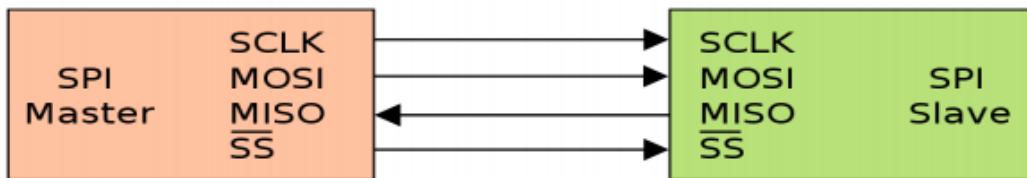
Tilslutningspunktet, se figur 5.15, sørger for at sammenkoble Enheden med de ønskede sensorer, samt at forsyne Enhed og sensorer med ønsket forsyning.



Figur 5.15. Kredsløb over samlet spændingsforsyning (3,3V og 5V) og tilslutninger til PSoC4

## 5.6 SPI (MK PO SK)

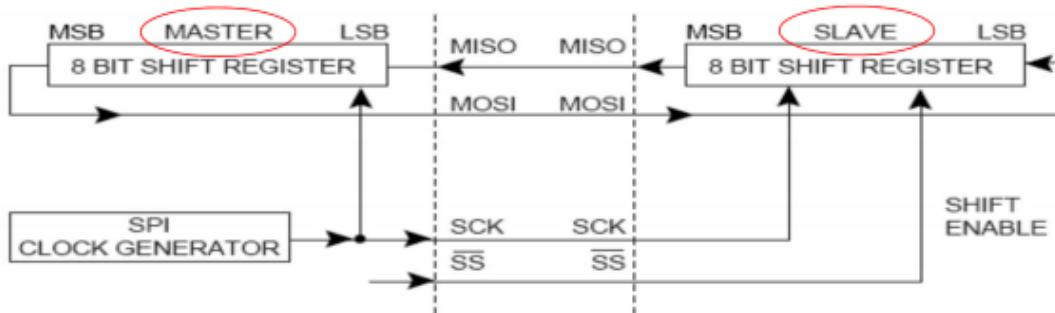
Serial Parallel Interface (SPI) er en måde at lave hurtig seriell dataudveksling på. SPI er udviklet af Motorola og fungerer ved at man, som regel, har en enkelt master enhed der styrer flere slave enheder. Ved SPI er der ingen fejl-check men adressering af flere ender kan dog være HW krævende. I EasyWater8000 projektet er der SPI kommunikation imellem Master(devkit8000) og Enhed(PSoC), det giver muligheden for at overføre flere data på samme tid imellem disse to.

**Figur 5.16.** SPI

På figur 5.16 ses en typisk opkobling imellem Master og Slave, det kræver 4 tråde.

- SS står for "Slave select".
- MISO står for "Master in slave out".
- MOSI står for "Master out slave in".
- SCLK står for "Serial clock"

SPI kommunikation er baseret på skifteredister princippet, som ses på figur 5.17. Der vælges hvilken slave der ønskes at skrives til ved slave select(SS), derefter shiftes et 8-bit register 1 bit ad gangen. Serial clock er den clock der sørger for at shiftningen af bits sker korrekt, clockfrekvensen må ikke overstige grænsen for hvad enhederne kan håndtere.

**Figur 5.17.** SPI register

En oftes anvendt kommunikation mellem master og slave foregår ved at master sætter SS til 0. Den fortæller nu til slaven at den er klar til dataoverførelse. Nu sender masteren en bit over til slaven og påbegynder en full duplex data transmission, samtidigt sender slaven en bit over til masteren. Denne process fortsætter indtil masteren har sendt alle sine bits, derefter stopper masteren med at toggle på clocken og slave enheder bliver frigivet.

SPI kan køre både Full- og halfduplex, dvs. at der kan være 2-vejs kommunikation ved Full-duplex eller 1-vejs kommunikation ved Half-duplex. I EasyWater8000 er der behov for at der kommunikeres 2 veje, så det er Full duplex. Men da Enheden fungere som slave, kan den ikke selvstændigt sende til Masteren, men behøver et kald om at nu skal den sende information. Dette tages der forbehold for i implementeringen.

### 5.6.1 Kernemodul og Driver

I SPI-kommunikationen er der taget udgangspunkt i HAL Exercise 7<sup>2</sup>, der er dog lavet kraftigt om i driveren, så det tilpasses EasyWater8000s behov. Kernemodulet er der ikke lavet de store ændringer i så denne beskrives ikke i projektet.

Driveren opbygges med en Skrive og en Læse metode, der begge kan læse eller skrive én 8 bits karakter(char).

```

1 int write(data){
2 Write 1 char to SPI network
3 }
4 int read(*datapointer){
5 Read data
6 Move data to datapointer
7 }
```

### 5.6.2 SPI API

SPI\_api er en klasse til at styre SPI kommunikationen. Den består af en række metoder beskrevet i afsnit 6.13. I de følgende sektioner beskrives i pseudo-kode hvordan metoderne skal implementeres. Alle metoderne afsluttes med at sende et 'C' til Enheden. Det sørger for at TxBufferen til SPI nulstilles, så den er klar til næste opgave.

#### **int activate(int unit) const**

Metoden skal sørge for at åbne kernefilen, skrive til den og lukke den igen.

```

1 Open file
2 Write 'A' to file
3 Write 'C' to file
4 Close file
5 Return 0
```

#### **int deactivate(int unit) const**

Metoden skal sørge for at åbne kernefilen, skrive til den og lukke den igen.

```

1 Open file
2 Write 'D' to file
3 Write 'C' to file
4 Close file
5 Return 0
```

#### **int verify(int unit) const**

Metoden skal sørge for at åbne kernefilen, skrive til den, læse fra den og lukke den igen. Den skal også sammenligne parameternummeret og det udlæste enhedsnummer fra Enheden og returnere 0 hvis der er et match og en fejlkode ved mismatch.

```

1 Open file
2 Write 'V' to file
3 Read unitNo from file
4 Parse unitNo to int
5 Write 'C' to file
6 Close file
```

---

<sup>2</sup>Hardware abstraktioner. Exercise 7: LDD with SPI. Øvelse med SPI Kommunikation

```

7 Compare unitNo with unit
8 Return 0 if match
9 Return error if mismatch

```

### **int config(int unit, float temp, float humi)**

Metoden skal sørge for at åbne kernefilen, skrive til den og lukke den igen. Den håndtere 2 floats, som skal konverteres til chars for at kunne sendes via SPI.

```

1 Open file
2 Write 'P' to file
3 Write temp to file
4 Write humi to file
5 Write 'C' to file
6 Close file
7 Return 0

```

### **int getLog(vector<string> &data, int \* units, int size)**

Metoden sørger for at åbne kernefilen, skrive til den, læse fra den og lukke den igen. Den udlæser et buffer array fra Enheden som pakkes ned i en vector til data parameteret.

```

1 Open file
2 Write 'L' to file
3 Read buffersize from file
4 Read buffer buffersize times
5 Build datastring if 'D' are present
6 Build errorstring(s) if 'E' are present
7 Build vector from data and errorstrings
8 Move vector to data
9 Close file
10 Return 0

```

#### **5.6.3 SPI Handler**

SPI handleren skal være en interrupt rutine, der startes når der kommunikeres over SPI. Den skal håndtere alle metoderne fra SPI\_api. Og kalde metoder på Enhed.

#### **ISR**

```

1
2 CY_ISR()
3 {
4     switch {
5         case 'A':
6             Turn on Green LED
7             Call activate method:
8         case 'D':
9             Turn on Red LED
10            Call deactivate method:
11        case 'P':
12            Turn on White LED
13            Save humidity and temperature data
14            Call config method
15        case 'V':
16            Turn on Purple LED
17            Write UnitNo to tx buffer
18        case 'L':
19            Turn on Blue LED
20            Get log data buffer
21            Write data buffer to tx buffer

```

```

22     case 'R':
23         Write tx buffer to Master
24     case 'C':
25         Clear tx-buffer
26         Reset spiCounter
27 }

```

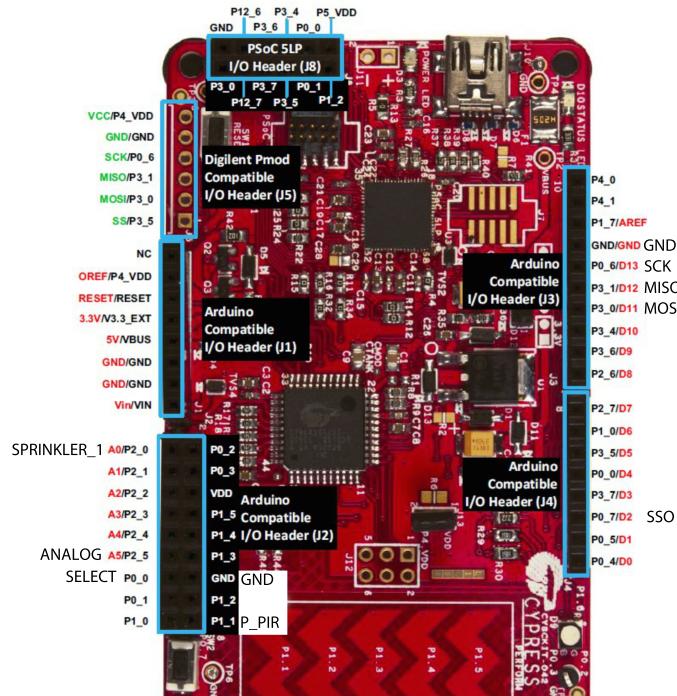
## 5.7 Hardware forbindelser (HW)

### 5.7.1 Enhed (HW)

Her beskrives hvor mange og hvilke pins der skal benyttes på PSoC4 til de forskellige forbindelser.

*Tabel 5.1.* Tabel der viser pins på PSoC4

Forbindelse	Antal pins	Signalnavne	Pinnavn	Kommentar
SPI (P_DK)	5	MOSI	P3[0]	
		MISO	P3[1]	
		SCK	P0[6]	
		SS0	P0[7]	
		GND	J3(GND)	
PIR-TTL (P_PIR)	1	P_PIR	P1[1]	
VP-TTL (P_VP)	1	SPRINKLER_1	P2[0]	
FT-analog (P_FT1)	1	ANALOG	P2[5]	Analog niveau
FT-TTL (P_FT2)	1	SELECT	P0[0]	Fugt/Temp.
Fælles GND	1	GND(Forb. print)	J2(GND)	Forb. print



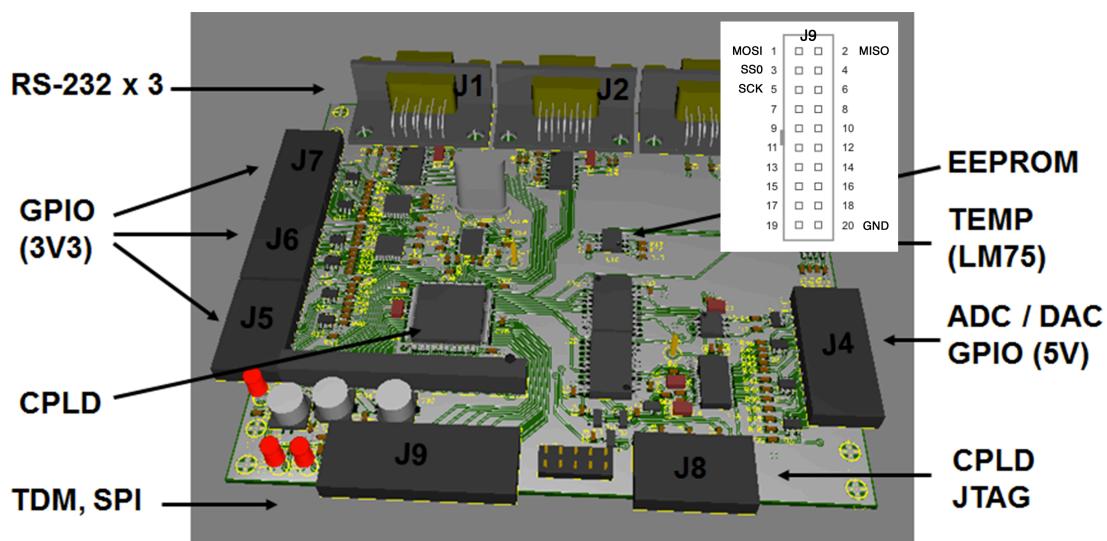
*Figur 5.18.* Oversigt over fysiske pin-navne på PSoC4

### 5.7.2 Master (HW)

Her beskrives hvor mange og hvilke pins der skal benyttes på Devkit8000 til de forskellige forbindelser.

**Tabel 5.2.** Tabel der viser pins på Devkit8000

Forbindelse	Antal pins	Signalnavne	Pinnavne	Kommentar
SPI (P_DK)	5	MOSI	J9.1	
		MISO	J9.2	
		SCK	J9.5	
		SS0	J9.3	
		GND	J9.20	



**Figur 5.19.** Model af tilføjelses-print og J9-stik

# Softwaredesign 6

---

## 6.1 Dataprotokol (BS)

### Log

Når data flyttes data mellem Master og Enhed, ifm. logning, anvendes følgende dataprotokol i mellem *Application*-lagene.

Den data som skal flyttes er følgende:

1. Temperatur
2. Fugtighed
3. Bevægelsesregistrering
4. Påbegyndt vanding

Systemet ved ikke på forhånd hvor meget data der skal flyttes. Derfor deles det op i to typer. Data fra sensorene samt besked om bevægelse og vanding, og fejlregistreringer. Fejlregistreringen anvendes kun i debug-øjemed.

Fra standardbiblioteket for C++ vælges datatypen `vector` som container. Dette er en dynamisk array-struktur som automatisk udvides og mindskes efter behov. Ved at anvende `vectoren` med `string` som undertype er det nemt at identificerer indholdene og formaterer dem som nødvendigt. På Enheden som implementeres på en PSoC er standardbiblioteket til C++ ikke tilrådighed. Derfor implementeres bufferen på denne som et almindeligt statisk allokeret `char`-array. Formateringen vist her under gælder for begge platforme.

Som nævnt er der flere muligheder for ”datapakker”. I tabellerne 6.1 og 6.2 er deres opbygninger vist.

Første streng der læses afgør hvor mange af de efterfølgende strenge som høre her til. Hvis det første der modtages er ”D”, betyder det at der kommer information fra sensorene og der skal læses fire efterfølgende strenge. Hvis der modtages et ”E” er der en fejl og én streng med fejkoden fra enheden læses.

**Tabel 6.1.** Dataformatering ifm. sensordata

Type	Temperatur	Fugtighed	Vanding	Bevægelse
”D”	”TTT.T”	”FFF”	”0” / ”1”	”0” / ”1”

**Tabel 6.2.** Dataformatering ifm. fejl

Type	Fejlkode
”E”	”XXX”

Et eksempel på strukturen af en datahentning er vist i tabel 6.3. Her kan man se, at der siden sidste udlæsning har været hentet sensordata med værdierne 14.5 grader og 20% fugtighed, der har ikke været tændt for sprinkleren men der har været bevægelse på hullet, D014.502001 og der har også været en fejl 13, E013. Bemærk at rækkefølgen ikke er bestemt på forhånd. Det er altså muligt at der kun ligger fejl på en Enhed, hvorfor der vil blive læst et antal fejl ud, men ingen datapakker.

**Tabel 6.3.** Dataformatering ifm. log-information

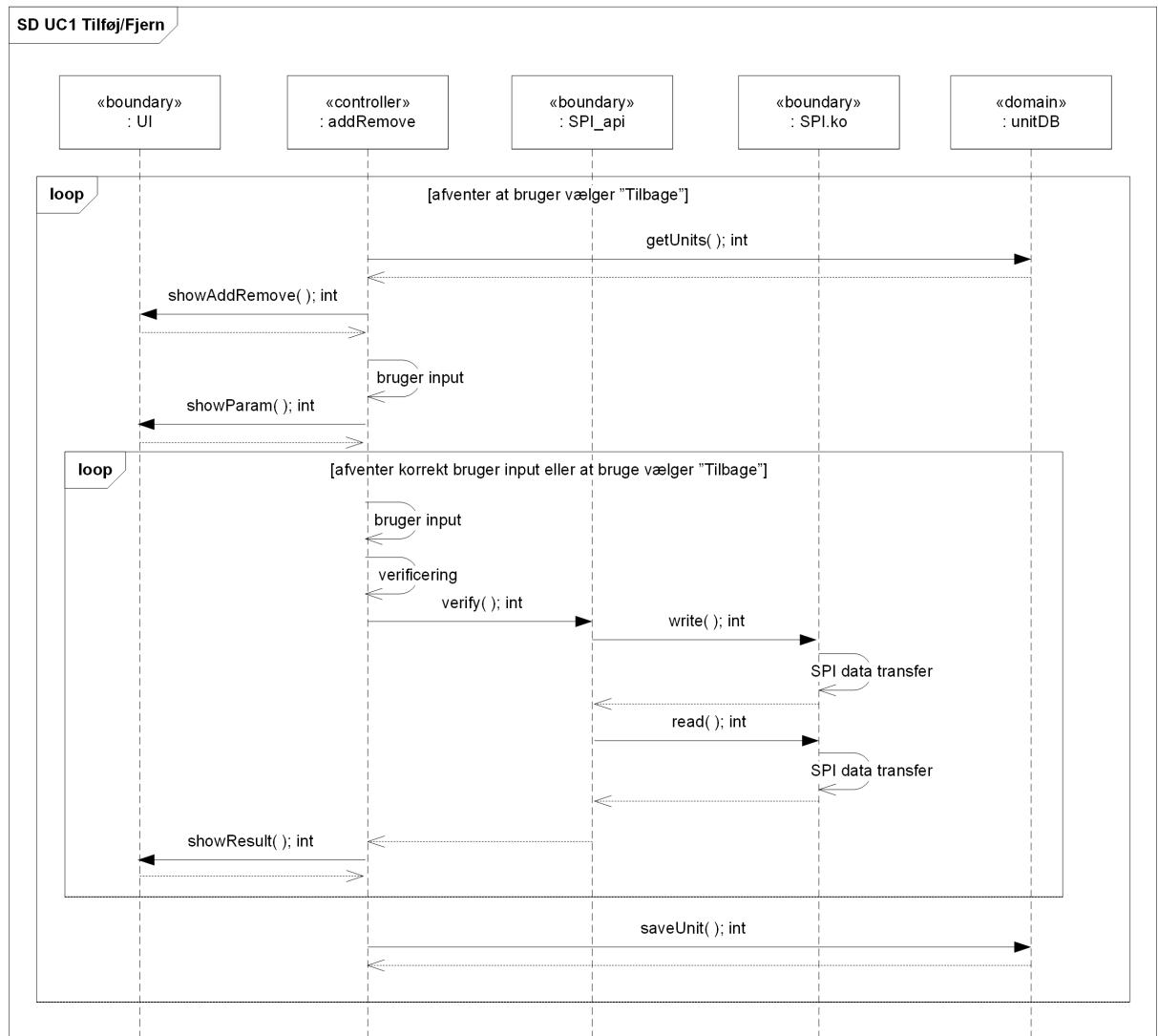
vector<string> [0]	[1]
"D014.502001"	"E013"

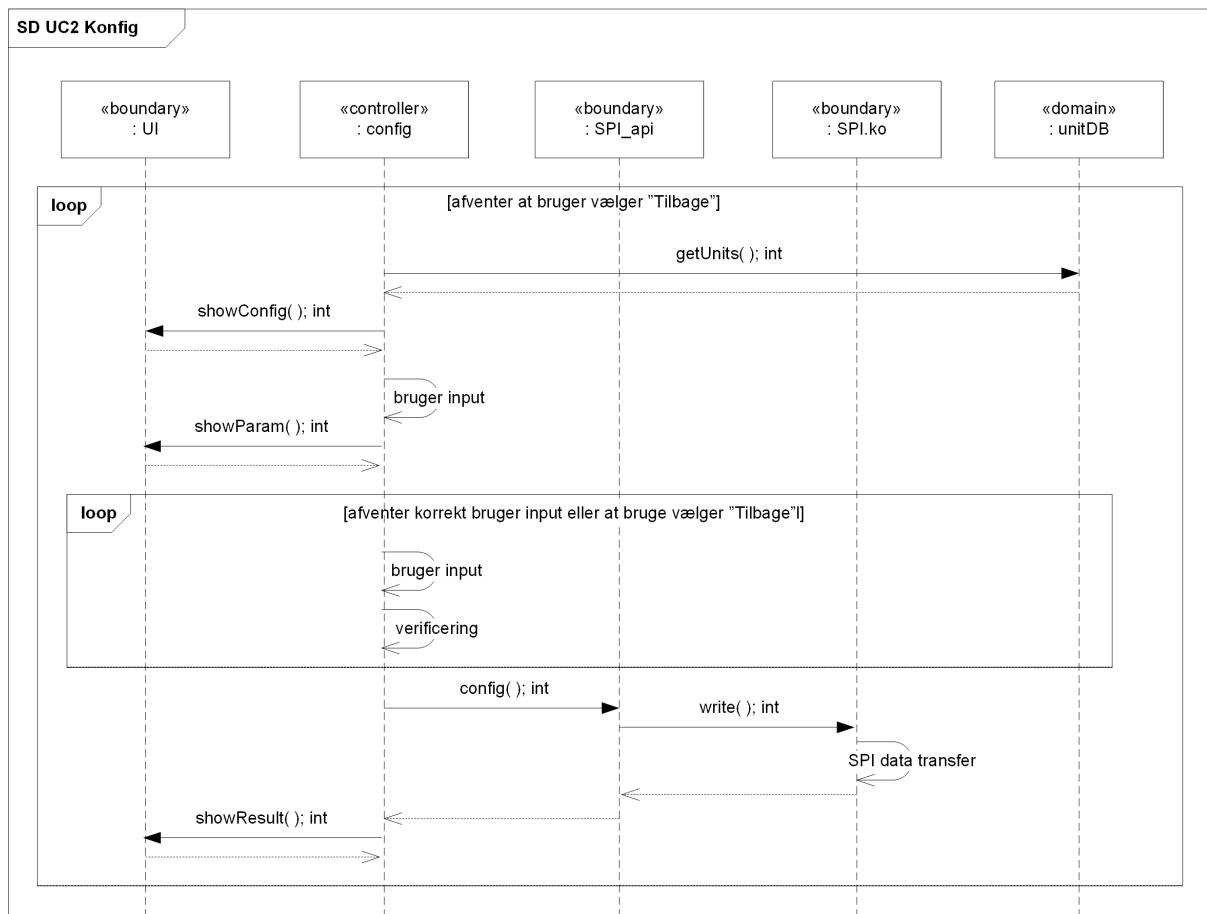
## 6.2 Applikationsmodeller (JC BS)

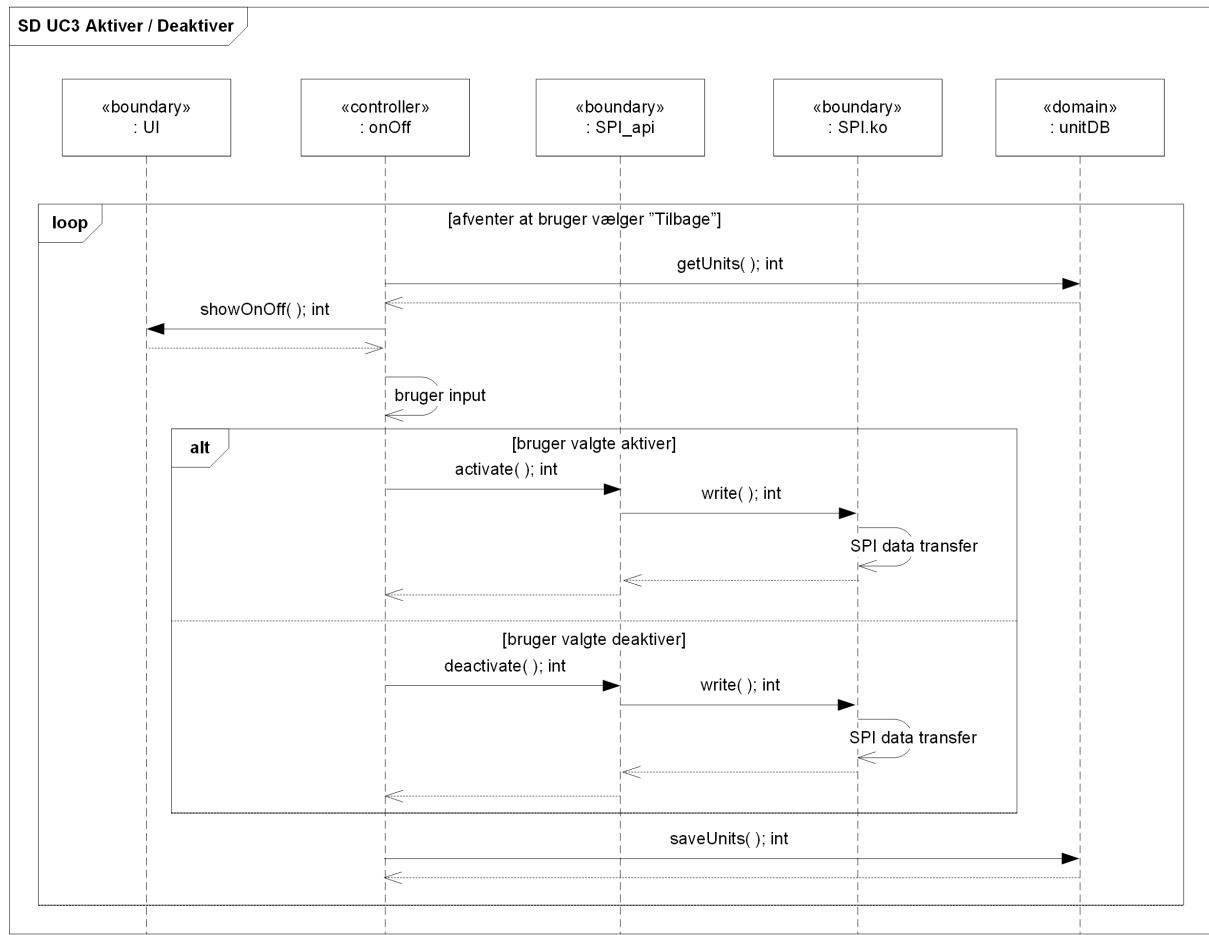
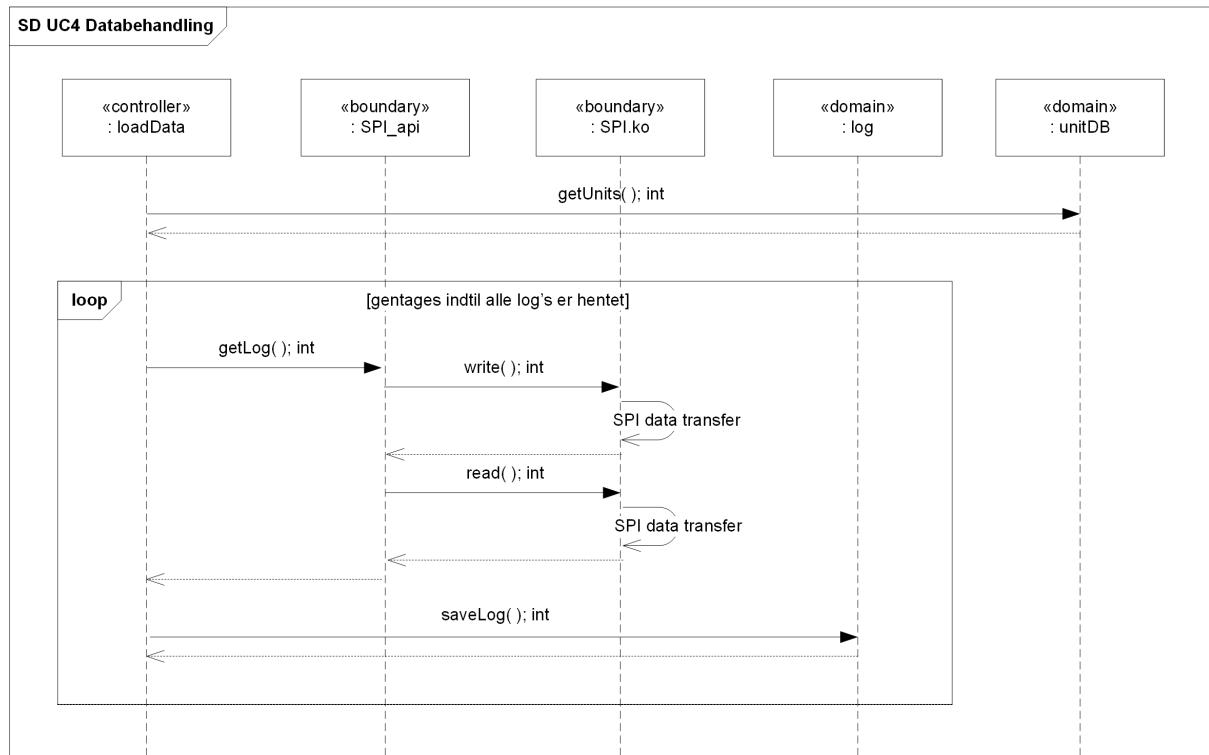
Selve designet af softwaren bygger på de følgende applikationsmodeller. Her laves der sekvens- og klassediagrammer over hver del af systemet samt klassebeskrivelser hvor funktionen for de enkelte metoder beskrives.

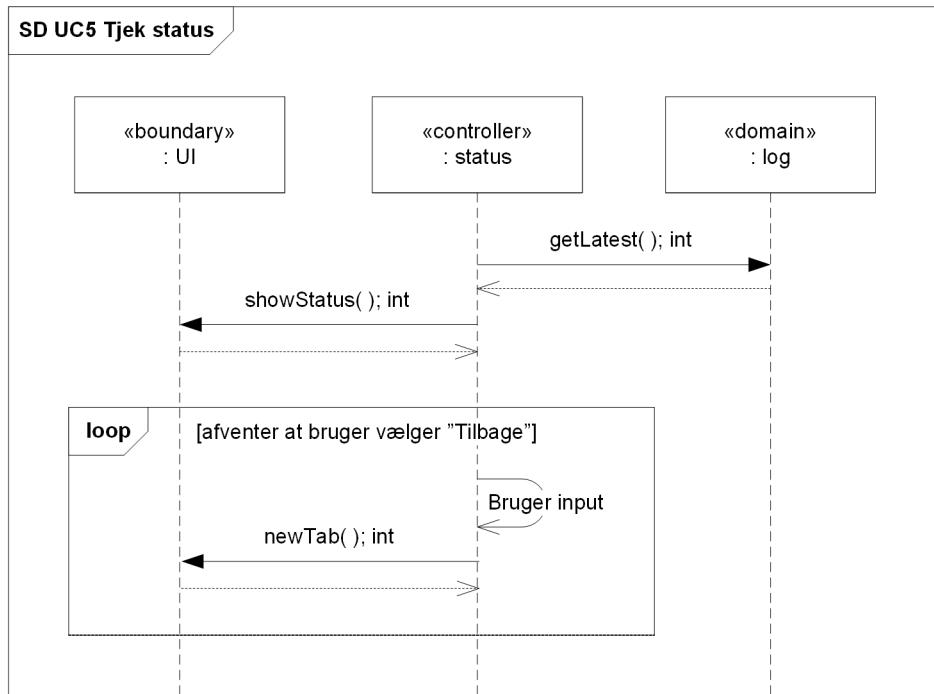
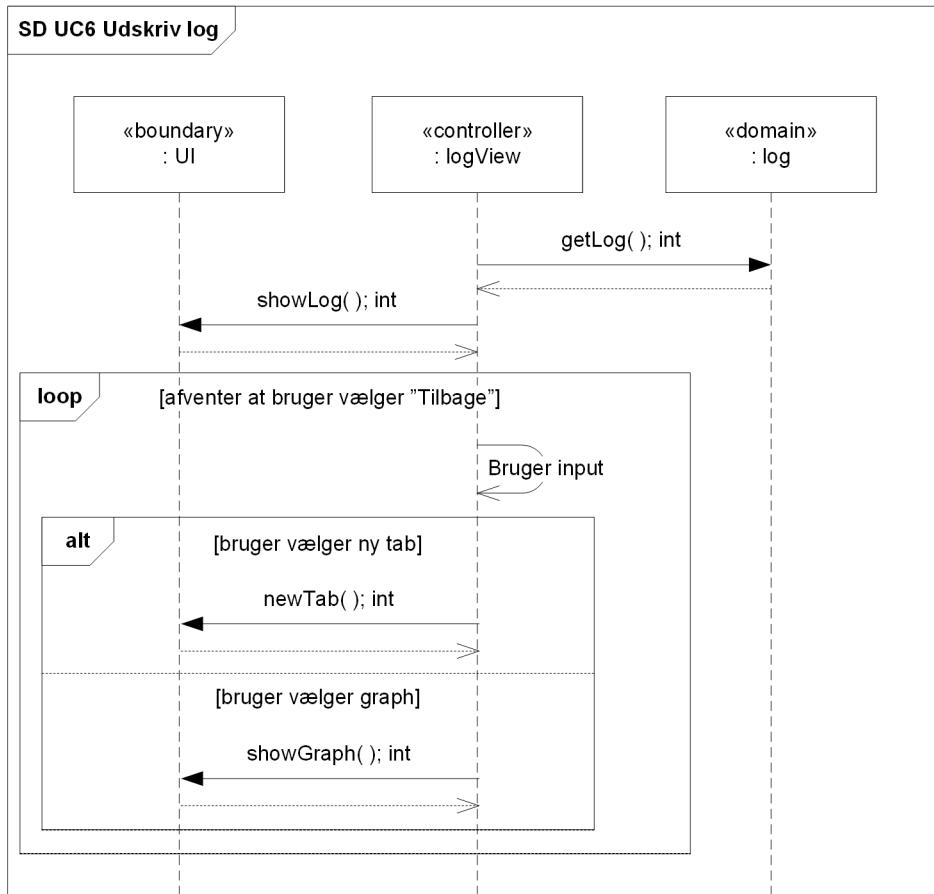
### 6.2.1 Master

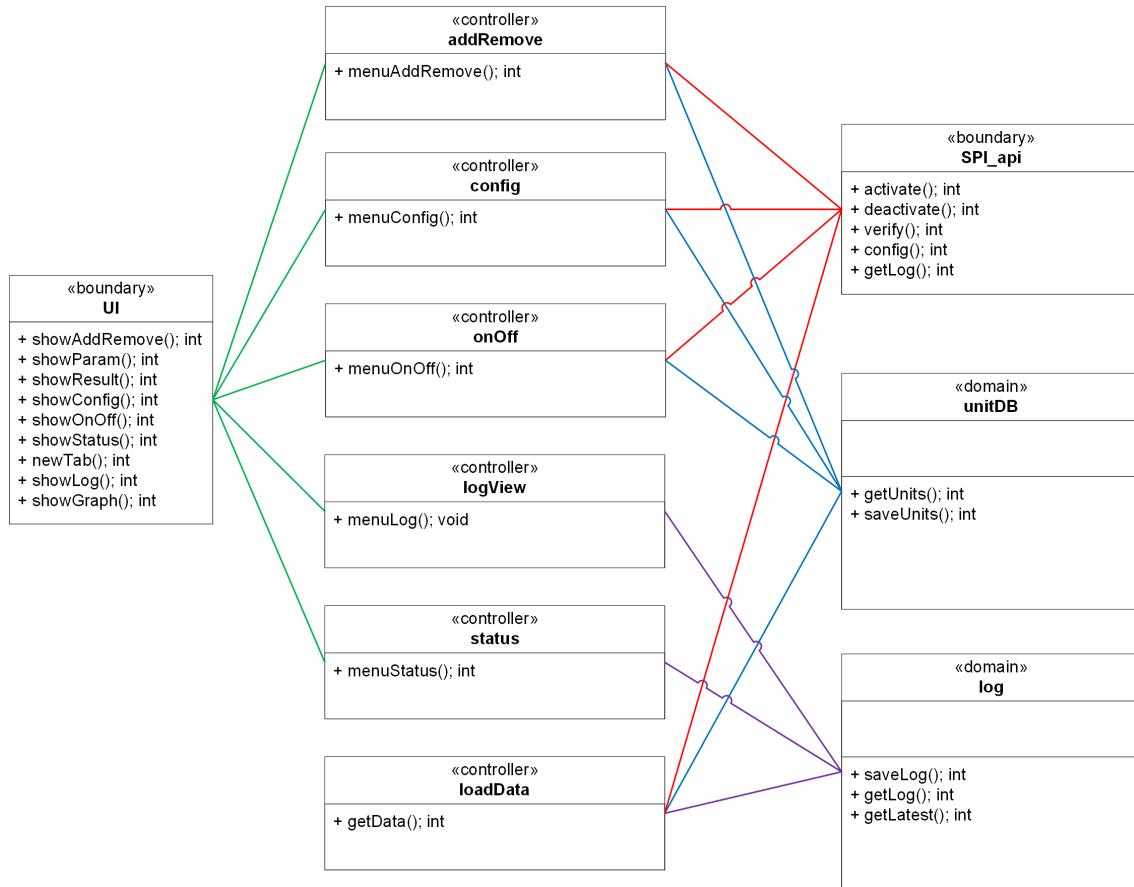
Applikationsmodeller for Master.

*Figur 6.1.* Sekvensdiagram UC1

*Figur 6.2.* Sekvensdiagram UC2

*Figur 6.3.* Sekvensdiagram UC3*Figur 6.4.* Sekvensdiagram UC4

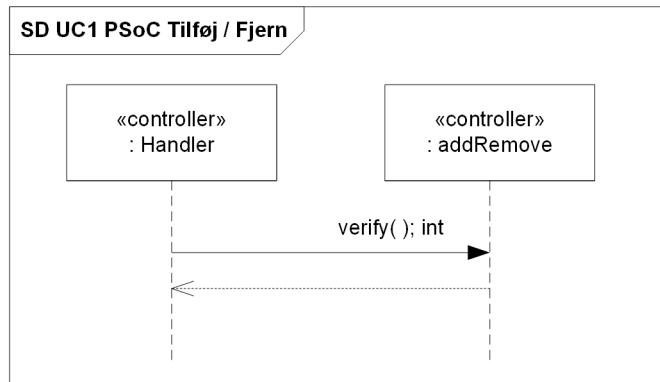
**Figur 6.5.** Sekvensdiagram UC5**Figur 6.6.** Sekvensdiagram UC6

**Figur 6.7.** Klassediagram for Master (Devkit8000)

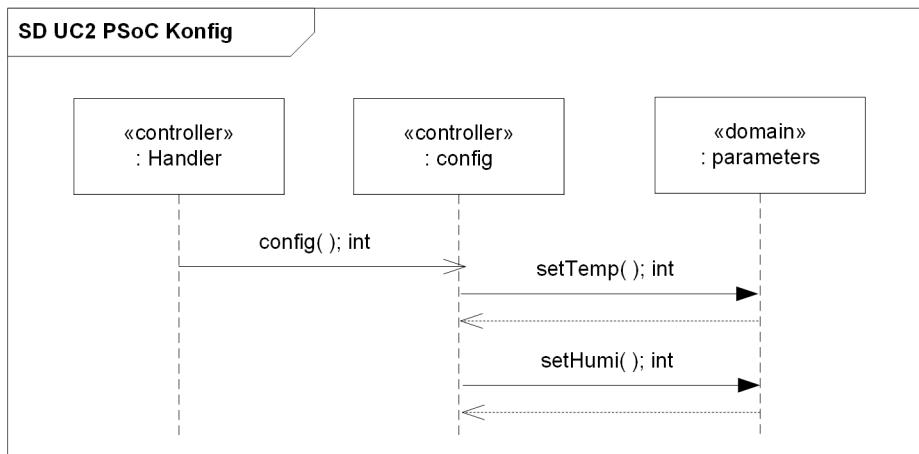
Efter udarbejdelsen af sekvensdiagrammer samles alle metode kald imellem klasserne til et klassediagram som ses ovenfor på figur 6.7. Efter dette klassediagram udarbejdes en klassebeskrivelse hvor der tænkes over hvilke attributter de forskellige metoder skal have for at kunne udføre deres ansvar. Det bliver så fulgt op med et endeligt statisk klassediagram som inkluderer alle attributter og metoder.

### 6.2.2 Enhed

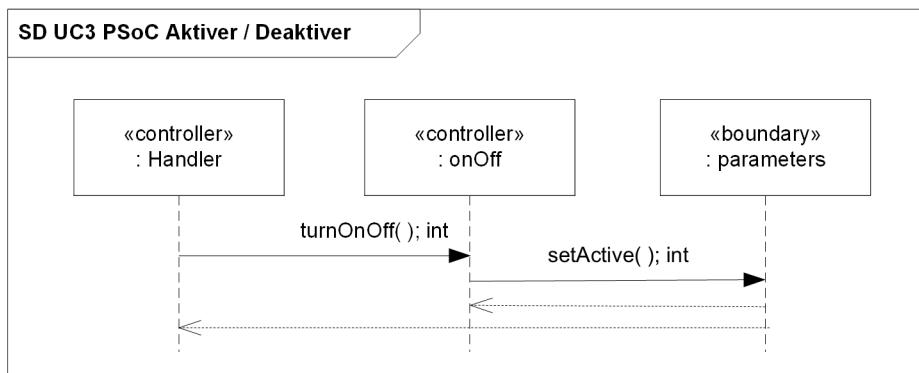
Applikationsmodeller for Enhed.



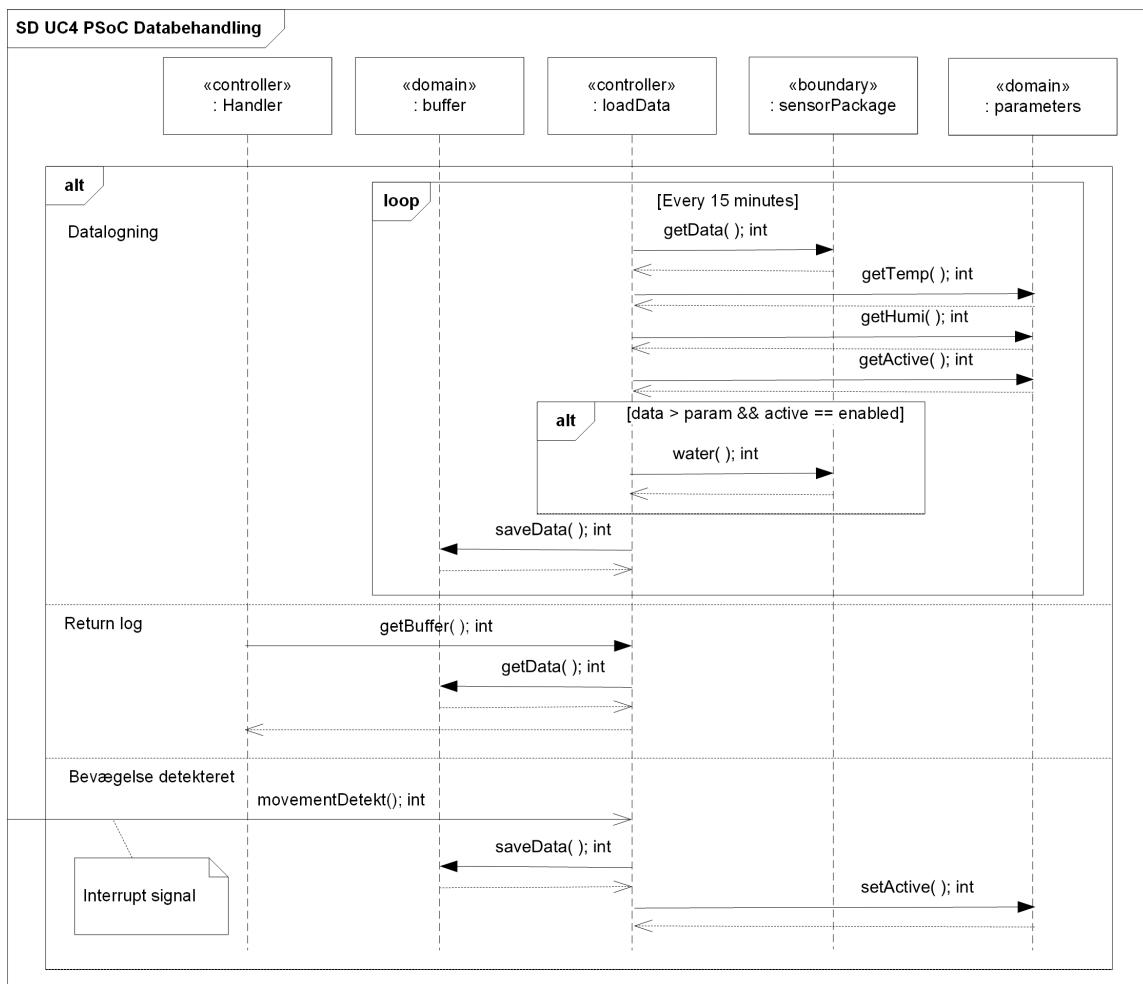
*Figur 6.8.* Sekvensdiagram UC1



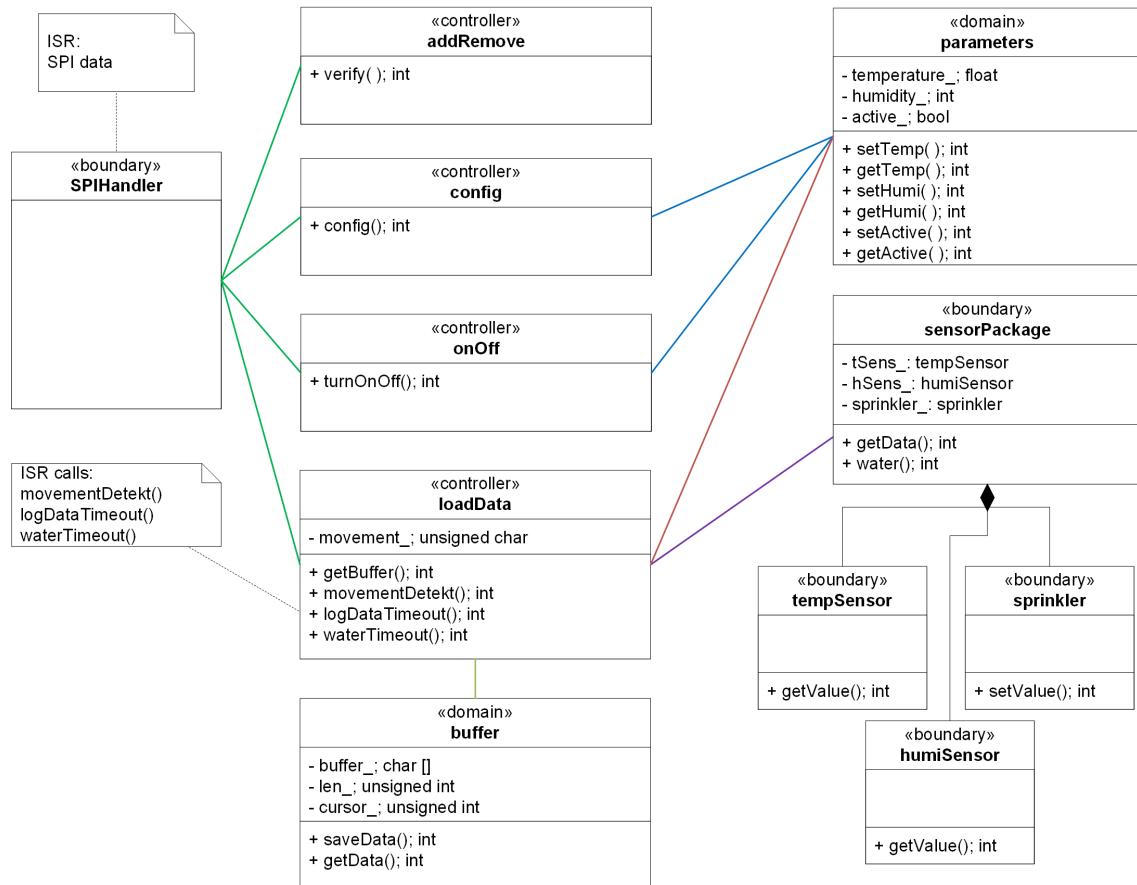
*Figur 6.9.* Sekvensdiagram UC2



*Figur 6.10.* Sekvensdiagram UC3

**Figur 6.11.** Sekvensdiagram UC4

Ovenstående sekvensdiagrammer ender også ud i et statisk klassediagram. Det er vist på figur 6.12.

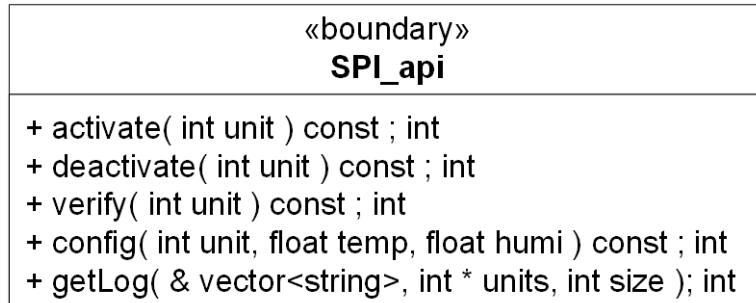


**Figur 6.12.** Klassediagram for Enhed (PSoC)

### 6.3 Klassebeskrivelser

Her følger klassebeskrivelser for de udledte klasser fra applikationsmodellerne.

#### 6.3.1 Master (JC)



**Figur 6.13.** klassediagram SPI API

**SPI\_api**

**Ansvar:** At være et lag imellem *applications*-laget og *device driver*-laget ifm. SPI kommunikation.

`int activate( int unit ) const`

**Parametre:** modtager en integer på enhed som skal aktiveres

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** metoden skal aktivere enheden `unit` over SPI netværket.

`int deactivate( int unit ) const`

**Parametre:** modtager en integer på følgende enhed som skal deaktiveres

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** metoden skal deaktivere enheden `unit` over SPI netværket.

`int verify( int unit ) const`

**Parametre:** modtager en integer på følgende enhed som skal verificeres

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** metoden skal verificere om enheden `unit` er tilkoblet SPI netværket.

`int config( int unit, float temp, float humi ) const`

**Parametre:** modtager en integer på følgende enhed som skal konfigureres. Derudover modtager den to floats med parametrene som skal skrives til Enhed.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

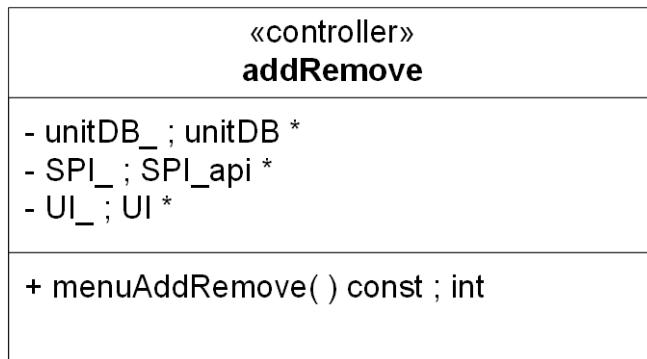
**Beskrivelse:** metoden skal sende konfigurations-parametrene i `float` til enheden `unit` over SPI netværket.

`int getLog( & vector<string> , int * units, int size )`

**Parametre:** modtager en referance til en vector af typen string som loggen skal gemmes i. Pointer til array af enheder som skal logges samt int med antallet af enheder i arrayet.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** metoden skal hente log fra enheder i arrayet `units` på SPI netværket og gemme dem i `vector` i henhold til dataprotokollen.



*Figur 6.14.* klassediagram addRemove

### addRemove

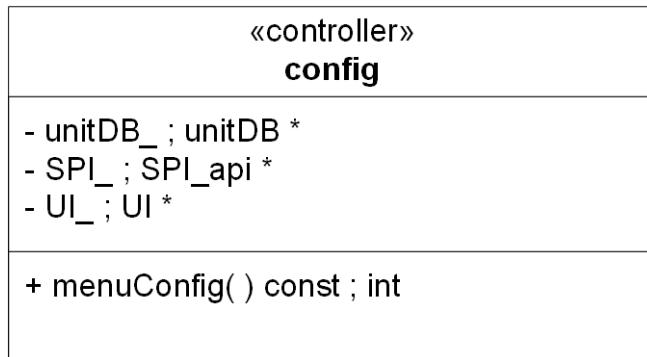
**Ansvar:** at styre forløbet i UC1: Tilføj / fjern enhed.

`int menuAddRemove( ) const`

**Parametre:** Modtager ingen parametre

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** Modtager information fra brugeren omkring enhed, verificere enheden over SPI netværket og gemme information i databasen..



*Figur 6.15.* klassediagram config

### Config

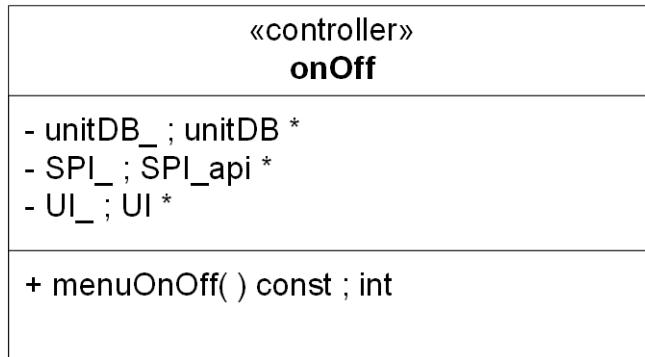
**Ansvar:** at styre forløbet i UC2: Konfig.

`int menuConfig( ) const`

**Parametre:** Modtager ingen parametre

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** Modtager information fra brugeren omkring enhed og parametre. Parametrene skrives til den pågældende enhed på SPI netværket.

*Figur 6.16.* klassediagram onOff**onOff**

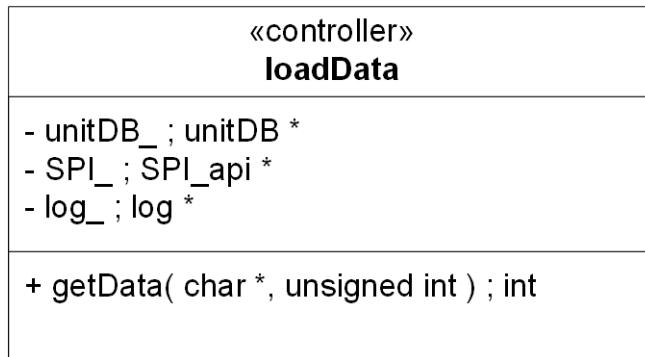
**Ansvar:** at styre forløbet i UC3: Aktiver / deaktiver.

`int menuOnOff( ) const`

**Parametre:** Modtager ingen parametre

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** Modtager information fra brugeren omkring hvilken enhed som ønskes aktiveret eller deaktiveret. Enheden modtager informationen over SPI netværket.

*Figur 6.17.* klassediagram loadData**loadData**

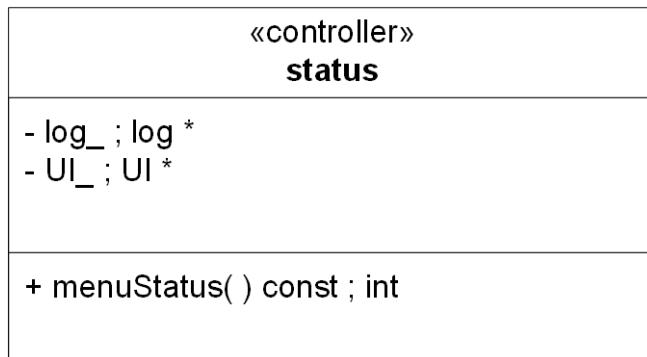
**Ansvar:** at styre forløbet i UC4: Databehandling .

`int menuloadData( )`

**Parametre:** Modtager ingen parametre

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** Sættes igang med et interrupt styret af en timer. Henter log over SPI netværket og gemmer den.

*Figur 6.18.* klassediagram status**status**

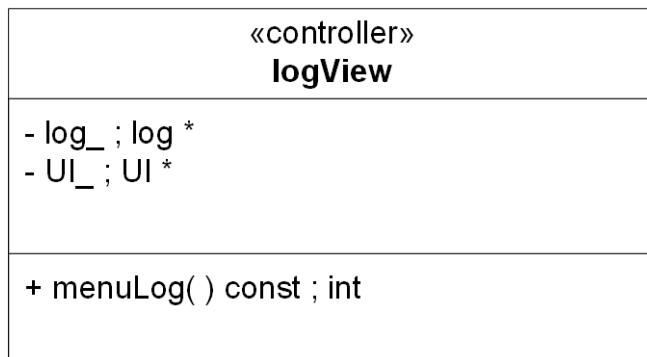
**Ansvar:** at styre forløbet i UC5: Tjek status.

`int menuStatus( ) const`

**Parametre:** Modtager ingen parametre

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** Henter den seneste log i hukommelsen og viser brugeren den, for den ønskede enhed.

*Figur 6.19.* klassediagram logView**logView**

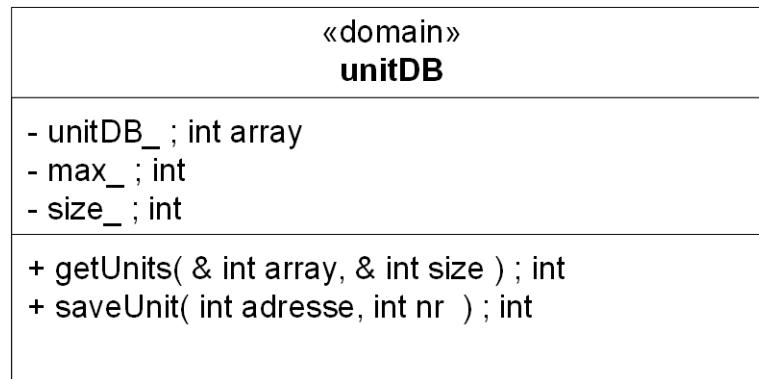
**Ansvar:** at styre forløbet i UC6: Udskriv log.

`int menuLog( ) const`

**Parametre:** Modtager ingen parametre

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** Henter hele loggen i hukommelsen og viser brugeren den, for den ønskede enhed.

**Figur 6.20.** klassediagram unitDB**unitDB**

**Ansvar:** at holde styr på hvor mange enheder der er og deres adresser.

`int getUnits( & int array, & int size )`

**Parametre:** Modtager en adresse på et array af ints og en tilhørende reference til at skrive size i.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

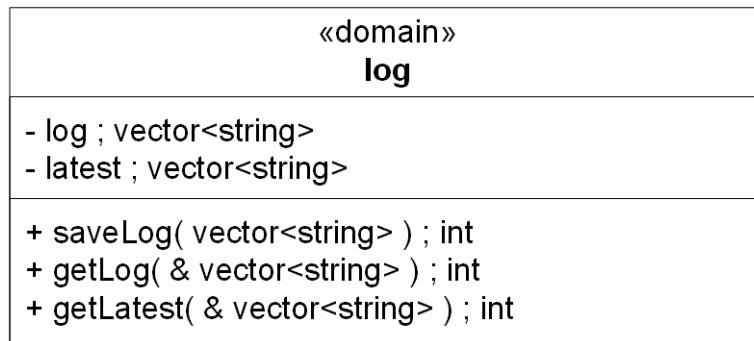
**Beskrivelse:** Metoden skriver sit adresse array over i det array som modtages. Derudover skriver den størrelsen på arrayet over i size så modtageren ved hvor stort arrayet er. (hvor mange golfhuller der er)

`int saveUnit( int adresse, int nr = size_ )`

**Parametre:** Modtager en adresse af typen int og et index nr af typen int.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** Metoden gemmer adressen som modtages på plads nr. Hvis der ikke modtages et index nr, så lægges adressen i array[size] og lægger 1 til size\_.

*Figur 6.21.* klassediagram log**log**

**Ansvar:** Gemme information loggen til senere brug.

`int saveLog( vector<string> )`

**Parametre:** Modtager en vector af typen string.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** Modtager log fra enhed og skriver den over i den samlede log og gemmer den i latest. Loggen skal gemmes i en txt fil som der læses fra under startup.

`int getLog( & vector<string> )`

**Parametre:** Modtager en adresse til en vector af typen string.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** Metoden skriver klassens medlems data log over i den adresse som den har modtaget som parameter.

`int getLatest( & vector<string> )`

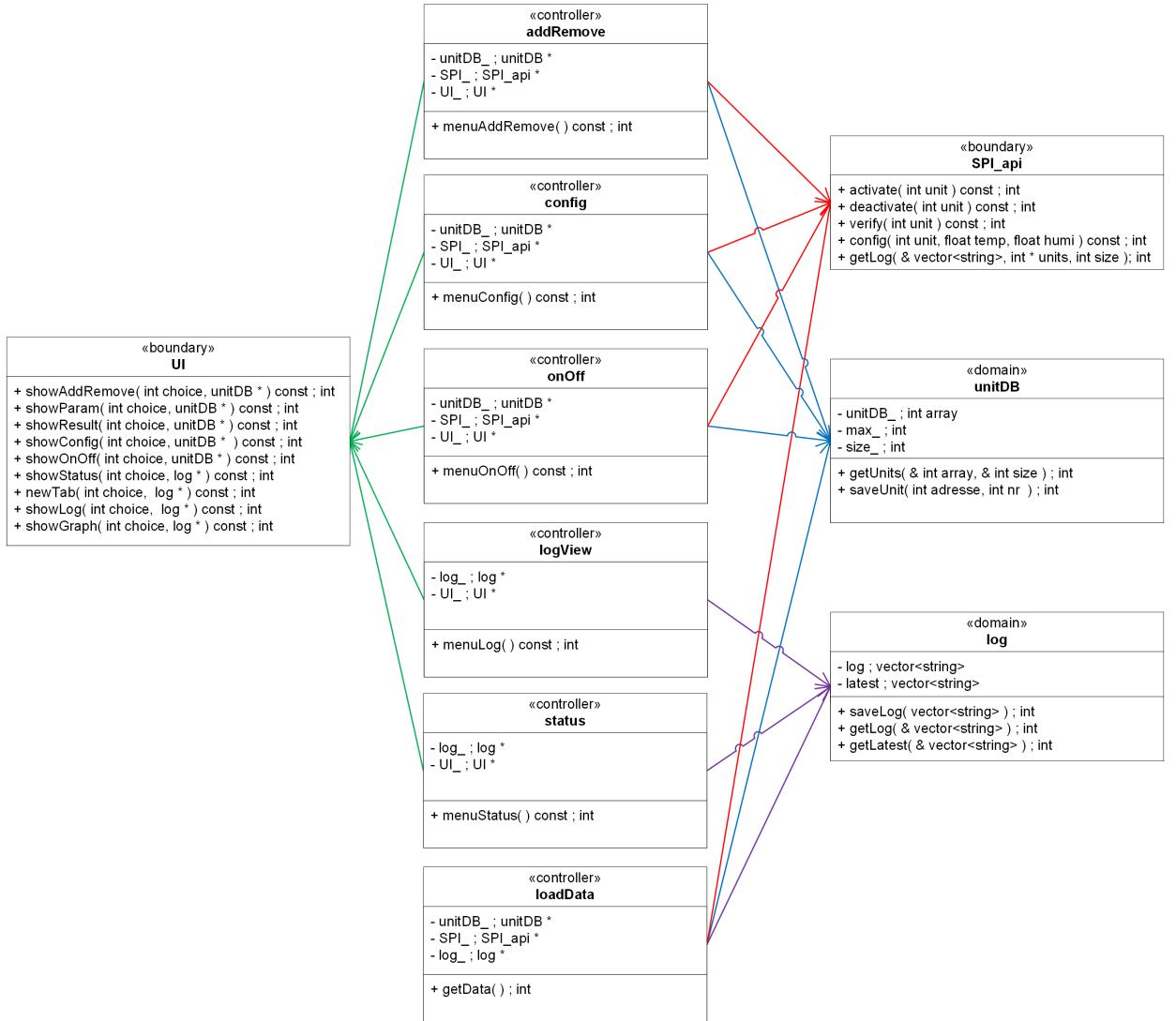
**Parametre:** Modtager en adresse til en vector af typen string.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen

**Beskrivelse:** Metoden skriver klassens medlems data latest over i den adresse som den har modtaget som parameter.

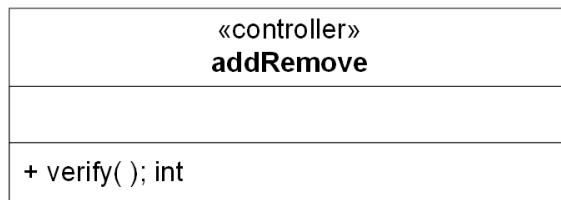
## 6.4 Statisk Klassediagram Devkit8000 (JC)

Efter sekvensdiagrammerne, klassediagrammet og klassebeskrivelser er vi kommet så langt med software designet af vi kan sammensætte et statisk klassediagram som viser hele software designet.



**Figur 6.22.** statisk klassediagram Devkit8000

#### 6.4.1 Enhed



*Figur 6.23.* Klasse addRemove

#### addRemove

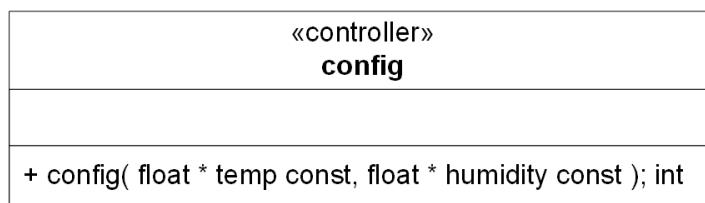
**Ansvar:** Kontrollerer hændelsesforløbet ifm. usecase 1.

**int verify( )**

**Parametre:** Ingen

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Returnerer kun 0. Bruges til at verificerer kommunikation mellem Master og Enhed.



*Figur 6.24.* Klasse config

#### config

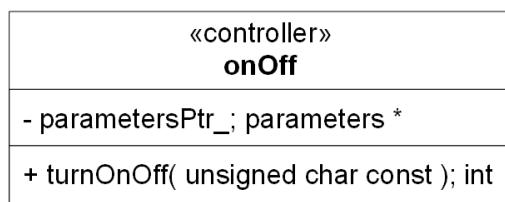
**Ansvar:** Kontrollerer hændelsesforløbet ifm. usecase 2.

**int config( float \* temp const, float \* humidity const )**

**Parametre:** To pointere til hhv. temperatur og fugtighedsgrænser

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Skal gemme parametre i et objekt af typen parameters ved at kalde metoderne `setTemp()` og `setHumi()`.



*Figur 6.25.* Klasse onOff

#### onOff

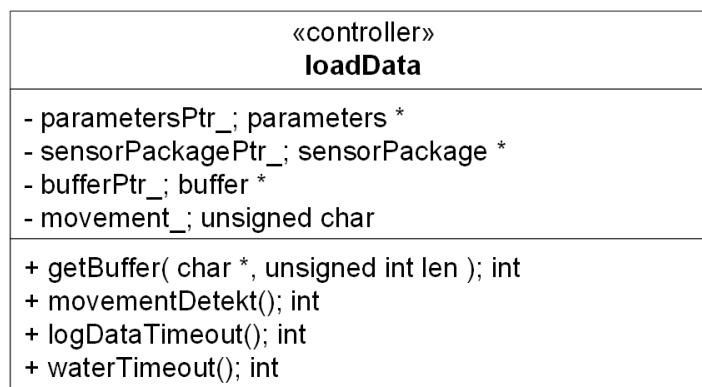
**Ansvar:** Kontrollerer hændelsesforløbet ifm. usecase 3.

```
int turnOnOff( unsigned char const )
```

**Parametre:** En bool som angiver on = true, off = false

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Skal sætte flaget **active\_** i **parameters**-objektet ud fra den modtagende parameter. Gyldige værdier er 0 og 1.



*Figur 6.26.* Klasse loadData

### loadData

**Ansvar:** Kontrollerer hændelsesforløbet ifm. usecase 4.

```
int getBuffer( char * buf, unsigned int len )
```

**Parametre:** Pointer til at skrive adressen til bufferen i og en længde.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Skal hente data fra **buffer**-objektet og gemme det i parametrene **buf** og **len**.

```
int movementDetekt( )
```

**Parametre:** Ingen.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Skal deaktivere vanding ved at sætte **active\_-**flaget i **parameters**-objektet til 0. Skal også starte en timer med udløb på 30 minutter. Skal sætte flaget **movement\_** til 1, så der næste gang der gemmes data, registreres bevægelse.

```
int logDataTimeout( )
```

**Parametre:** Ingen.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Skal aflæses data fra **sensorPackage** og gemme disse i **buffer**-objektet. Afhængig af **movement\_-**flaget, tilføjes passende **char** efter målt data i hht. dataprotokollen, og flaget sættes til 0.

```
int waterTimeout( )
```

**Parametre:** Ingen.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Skal aktivere muligheden for vanding ved at sætte **active\_-**flaget i **parameters**-

objektet til 1.

«domain»
parameters
- temperature_; float
- humidity_; float
- active_; unsigned char
+ setTemp( float const ); int
+ getTemp( float * ); int
+ setHumi( float const ); int
+ getHumi( float * ); int
+ setActive( unsigned char const ); int
+ getActive( unsigned char * ); int

*Figur 6.27.* Klasse parameters

### parameters

**Ansvar:** Gemme information om grænseværdier mv. til det autonome vandingssystem.

`int setTemp( float temp const )`

**Parametre:** temperatur.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Gemmer modtagne temperatur i medlemsdata `temperature_`.

`int getTemp( float * temp )`

**Parametre:** Pointer til at gemme temperatur i.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Returnerer medlem `temperature_` i reference.

`int setHumi( float humi const )`

**Parametre:** humidity.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Gemmer modtagne humidity i medlemsdata `humidity_`.

`int getHumi( float * humi )`

**Parametre:** Pointer til at gemme humidity i.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Returnerer medlem `humidity_` i reference.

`int setActive( unsigned char const )`

**Parametre:** 1 = aktiv, 0 = inaktiv.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Gemmer modtagne char i medlemsdata `active_`.

```
int getActive( unsigned char * )
```

**Parametre:** Pointer til at gemme unsigned char i.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Returnerer medlem **active\_** i pointer.

«domain»
<b>buffer</b>
<ul style="list-style-type: none"> <li>- buffer_; char []</li> <li>- len_; unsigned int</li> <li>- cursor_; unsigned int</li> </ul>
<ul style="list-style-type: none"> <li>+ saveData( char * const, unsigned int const ); int</li> <li>+ getData( char *, unsigned int len * ); int</li> </ul>

*Figur 6.28.* Klasse buffer

### buffer

**Ansvar:** Holde data fra sensorerne indtil de udlæses af Master

```
buffer( )
```

**Parametre:** Ingen.

**Returværdi:** Ingen.

**Beskrivelse:** Skal initialisere char array **buffer\_** med plads til én datamåling og 10 fejl iht. dataprotokol.

```
int saveData( char * buf const, unsigned int len const )
```

**Parametre:** Pointer til buffer med len data.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Skal gemme modtaget data i **buffer\_-medlemmet** og flytte **cursor\_-antallet** af karakterer frem.

```
int getData( char * buf, unsigned int * len )
```

**Parametre:** Pointer til char array til data.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Skal returnerer data fra **buffer\_** i pointer og nulstille **cursor\_-medlemmet**.

«boundary»
<b>sensorPackage</b>
<ul style="list-style-type: none"> <li>- tSens_; tempSensor</li> <li>- hSens_; humiSensor</li> <li>- sprinkler_; sprinkler</li> </ul>
<ul style="list-style-type: none"> <li>+ getData( float * temp, float * humi ); int</li> <li>+ water( unsigned char const ); int</li> </ul>

*Figur 6.29.* Klasse sensorPackage

### sensorPackage

**Ansvar:** Holder styr på tilkoblede sensorer.

**sensorPackage( )**

**Parametre:** Ingen

**Returværdi:** Ingen

**Beskrivelse:** Skal oprette sensor- og sprinklerobjekter og udføre den nødvendige opsætning af disse.

**int getData( float \* temp, float \* humi )**

**Parametre:** Pointers til at gemme aflæste temperatur og fugtighed i.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

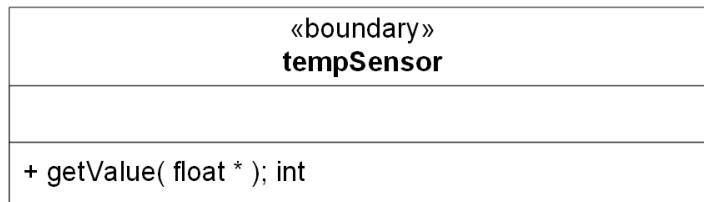
**Beskrivelse:** Aflæs data fra temperatur- og fugtighedssensor og returner disse i de modtagende referencer.

**int water( unsigned char const )**

**Parametre:** 1 = tænd sprinkler, 0 = sluk sprinkler

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Tænd eller sluk tilkoblede sprinkler ud fra modtagende parameter.



*Figur 6.30.* Klasse tempSensor

### tempSensor

**Ansvar:** Håndterer kommunikation med temperatursensor.

**tempSensor( )**

**Parametre:** Ingen.

**Returværdi:** Ingen.

**Beskrivelse:** Initialisere nødvendige indstillinger for at kunne kommunikerer med sensoren.

**int getValue( float \* )**

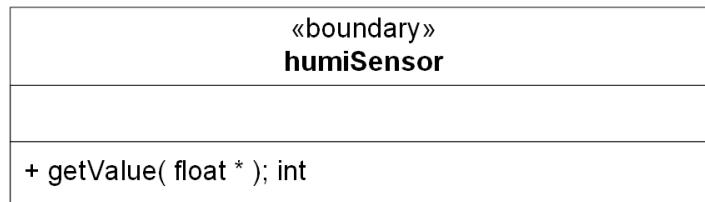
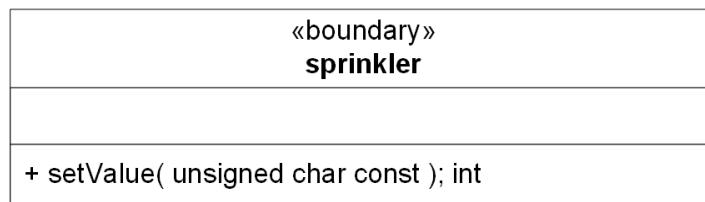
**Parametre:** Pointer til at gemme temperatur i.

**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.

**Beskrivelse:** Returnerer aflæst temperatur i pointerparameteren.

### humiSensor

**Ansvar:** Håndterer kommunikation med fugtighedssensor.

*Figur 6.31.* Klasse humiSensor**humiSensor( )****Parametre:** Ingen.**Returværdi:** Ingen.**Beskrivelse:** Initialisere nødvendige indstillinger for at kunne kommunikere med sensoren.**int getValue( float \* )****Parametre:** Pointer til at gemme fugtighed i.**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.**Beskrivelse:** Returnerer aflæst fugtighed i pointerparameteren.*Figur 6.32.* Klasse sprinkler

### sprinkler

**Ansvar:** At aktiverer og deaktiverer tilkoblede sprinkler**sprinkler( )****Parametre:** Ingen.**Returværdi:** Ingen.**Beskrivelse:** Skal initialisere nødvendige dele for at kunne bruge GPIO.**int setValue( unsigned char const )****Parametre:** 1 = tænd sprinkler, 0 = sluk sprinkler.**Returværdi:** 0 ved succes ellers negativ i overenstemmelse med fejl-listen.**Beskrivelse:** Skal aktiverer eller deaktiverer sprinkler ved hjælp af GPIO.

### Interrupt ISRer

**ISR( SPIHandler )****Parametre:** Ingen.**Returværdi:** Ingen.**Beskrivelse:** Modtaget SPI kommando afkodes, iht. kommunikationsprotokollen, hvor

efter de relevante controllerer kaldes.

ISR( movement )

**Parametre:** Ingen.

**Returværdi:** Ingen.

**Beskrivelse:** Metoden movementDetect() kaldes i loadData-objektet.

ISR( waterTimeout )

**Parametre:** Ingen.

**Returværdi:** Ingen.

**Beskrivelse:** Metoden waterTimeout() kaldes i loadData-objektet.

ISR( logDataTimeout )

**Parametre:** Ingen.

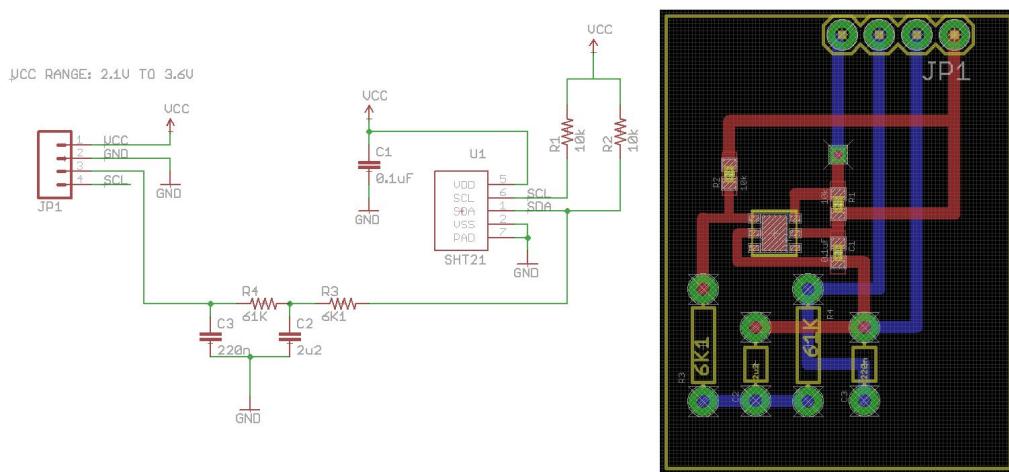
**Returværdi:** Ingen.

**Beskrivelse:** Metoden getLogTimeout() kaldes i loadData-objektet.

# Implementering

## 7.1 Fugt- og temperatursensor (LB)

Implementeringen af sht21p er sket ved at designe et print som indeholder sht21p og et 2. ordensfilter. Da sht21p-komponentet er et SMD-komponent var det ikke muligt at opstille kredsløbet på vero-board. Det var derfor nødvendigt at få et print produceret. Til design af printet blev CadSoft EAGLE PCB design software brugt. Nedenfor ses diagram og PCB layout af printet.

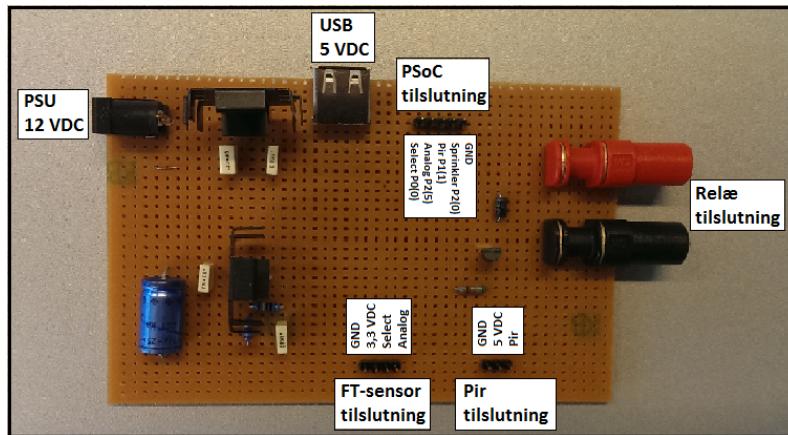


**Figur 7.1.** Schematic og layout af SHT21P kredsløb

Kredsløbet på figur 7.1 blev eksporteret til gerberfiler og sendt til elektronikværkstedet som så ætsede printet og derefter monteret op.

## 7.2 Tilslutningsprint (SK)

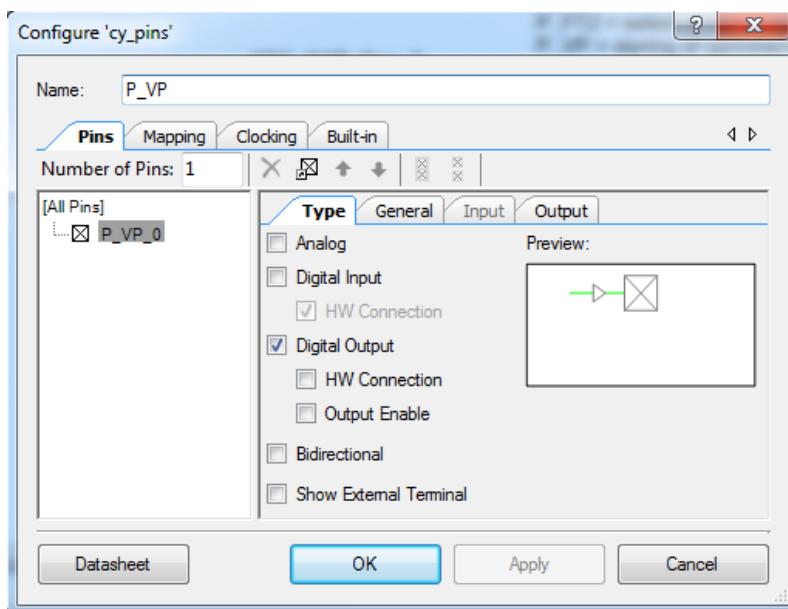
Implementeringen af tilslutningsprintet er lavet ved at konstruere et veroboard print. Tilslutnings mulighederne på printet består af en indgang til 12 VDC fra PSU'en, en USB udgang der forsyner enheden, 12 harwin stik til de forskellige sensorer og en udgang der styrer sprinklerrelæet. På figur 7.2 kan det færdige resultat ses.



*Figur 7.2.* Tilslutningsprint

### 7.3 PSoC API (JS)

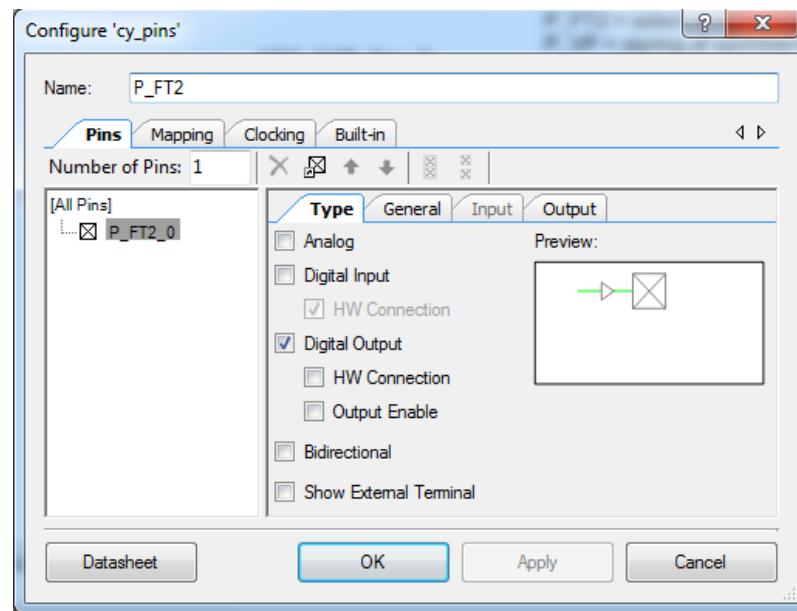
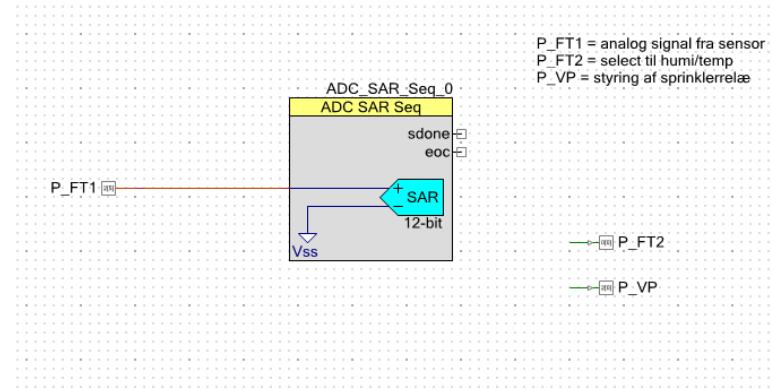
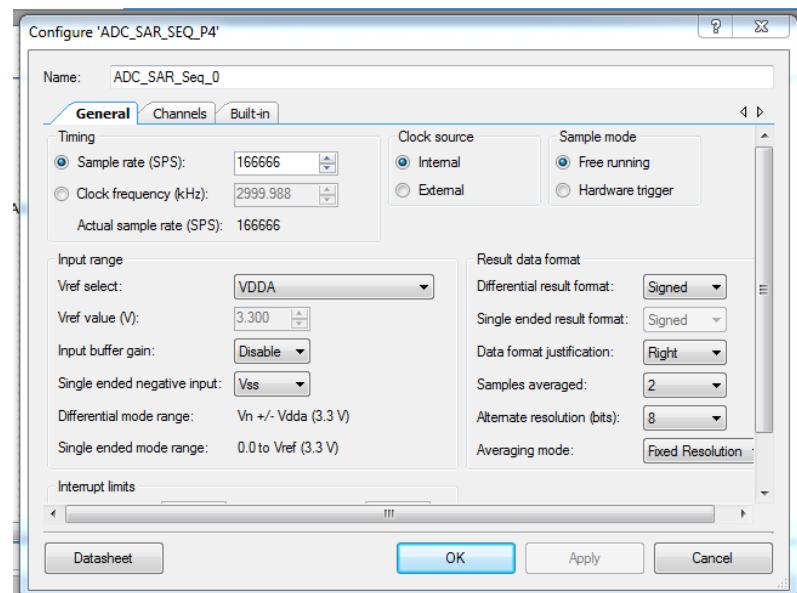
API til styring af PSoC, herunder sensor og sprinkler, er implementeret i PSoC Creator. Top designet består af en ADC\_SAR\_Seq, en analog pin og to digitale pins til at styre henholdsvis sprinkler og Select til bestemmelse af temperatur- eller fugtighedsdata for SHT21P. I top designet er der angivet følgende pins. "P\_FT1", "P\_FT2", "P\_VP" disse forbindes til henholdsvis pin P2[5], P0[0] og P2[0]. P\_FT1 oprettes som en analog pin og de to andre som digitale output pins. Under konfiguration for de digitale pins fravælges "HW Connection" under Digital Output. For nærmere opsætning af pins henvises til figur 7.3 og figur 7.4

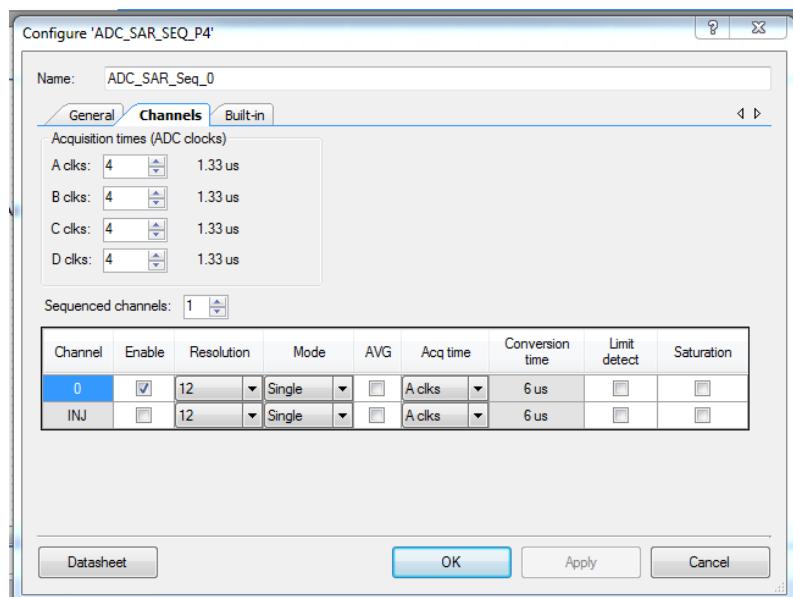


*Figur 7.3.* Konfig. for water pin

Under konfiguration for ADC-komponenten vælges Vref til VDDA og Single ended negative input vælges til Vss. Dette sætter ADCen op til at bruge 0 V som reference spænding.

Da PSoC kun har en SAR komponent er det nødvendigt at initialisere denne i sensorPackage-driveren.

*Figur 7.4.* Konfig. for SLC pin*Figur 7.5.* Top Design for PSoC API*Figur 7.6.* Konfiguration af Vref og Single ended negative input



**Figur 7.7.** Konfiguration af Channels for PSoC API

## 7.4 230V relæ (PO)

Ingeniørhøjskolen ved Aarhus Universitet tillader kun at arbejde med spændinger op til 48V uanset tidligere baggrund<sup>1</sup>. Der er dog givet særskilt tilladelse til at bygge 230V relæet, hvis dette indbygges i en lukket kasse som beskrevet i design delen. Relæet er godkendt af Torben Lund Jensen fra værkstedet på IHA.

Relæet er implementeret som beskrevet i designdelen. Figur 7.8 viser relæet som det er implementeret. Relæet er indbygget i en lukket kasse med transparent låg. Veroboard er benyttet som sokkel for relæet. Installationsledning er benyttet til 230V spændingen 0,75 kvadrat. Den lyseblå leder er nul og den sorte leder er 230V fasen Jord/beskyttelseslederen er den gul/grønne, denne går direkte fra indgang (apparatstik) til udgang (230V stikkontakt).



**Figur 7.8.** Relæet uden påmonteret låg

Figur 7.9 viser relæet fra begge ender. Veste side viser indgangen, som består af ét 230V apparatstik og 2 bananstik. Apparatstikket tilsluttes en stikkontakt som er tændt. Bananstikkene tilsluttes tilslutningsprintet. Højre side viser 230V stikkontakten som tændes/slukkes af relæet. Vandpumpen tilsluttes denne stikkontakt.



**Figur 7.9.** Relæet set fra begge ender.

---

<sup>1</sup>Poul er ikke uddannet elektriker, men har lavet el til husbehov. Relæet kan sammenlignes med en forlængerledning med enafbryder, hvilket er lovligt at lave privat

## 7.5 SPI (MK PO)

SPI kommunikationen har en API klasse på Master skrevet i C++ og en handler på Enhed skrevet i C.

For at add-on printet på Devkit8000 kan bruge GPIO benene til SPI kommunikationen er det nødvendigt at rute dem korrekt, dette gøre ved at bruge følgende 2 kommandoer.

- echo 0x3 > /sys/class/cplddrv/cpld/spi\_route\_reg
- echo 0x1 > /sys/class/cplddrv/cpld/ext\_serial\_if\_route\_reg

Kernemodulet skal også indsættes og oprette en system node, i APIen bruges /dev/spi\_dev så denne er fast. Alt dette er skrevet i et shell script der håndtere alt indsætning og oprettelse af kernefilen, se bilag `insert_SPI_dev.sh`

### 7.5.1 SPI Driver

Driveren til SPI består af 2 hovedmetoder, der er en skrive-metode og en læse-metode. Disse sender og læser karakterer(char's) én ad gangen. Der er taget udgangspunkt i HAL Exercise 7<sup>2</sup> men netop læse og skrive metoderne er blevet ændret for at tilpasse EasyWater8000 behov for datatransmission.

#### Write

Write metoden modtager en data char, som den ligger i tx-bufferen og skriver ud i spi\_sync. U8 cmd håndteres ikke for EasyWater8000, men er bibeholdt så den evt. kan bruges til andre projekter.

```

1 int psoc4_spi_write_reg8(struct spi_device *spi, u8 cmd, char data)
2 {
3     struct spi_transfer t[1];    // Number of packages
4     struct spi_message m;
5
6     /* Check for valid spi device */
7     if(!spi)
8         return -ENODEV;
9
10    /* Init Message */
11    memset(&t, 0, sizeof(t));
12    spi_message_init(&m);
13    m.spi = spi;
14
15    /* Configure tx/rx buffers */
16    t[0].tx_buf = &data;          // Write data to tx-buffer
17    t[0].rx_buf = NULL;          // Do nothing with rx-buffer
18    t[0].len = 1;
19    spi_message_add_tail(&t[0], &m);
20
21    /* Transmit SPI Data (blocking) */
22    spi_sync(m.spi, &m);          // Transmit SPI data
23
24    return 0;
25 }
```

---

<sup>2</sup>Hardware abstraktioner. Exercise 7: LDD with SPI. Øvelse med SPI Kommunikation

## Read

Read-metoden skriver altid et 'R' til target i form af en hardcoded char, derfor bruges `u8 addr` ikke i EasyWater8000 projektet. I samme transmission, læses der fra rx-bufferen på target. Det indsatte delay i transmissionen, er som sådan ikke nødvendig, da der kun læses én gang fra target og i mellem hver læsning er der minimum 300us. Men det er beholdt for en sikkerheds skyld, da det ikke er problematisk at en læsning tager 100us mere.

Rx-bufferen skrives til data, som flyttes til value char-pointeren, til videre håndtering af Masteren.

```

1 int psoc4_spi_read_reg8(struct spi_device *spi, u8 addr, char* value)
2 {
3     struct spi_transfer t[1];
4     struct spi_message m;
5     char data;           // Temp. var to hold readings
6     char cmd = 'R';      // Hardcoded 'R' char
7
8     /* Check for valid spi device */
9     if(!spi)
10         return -ENODEV;
11
12     /* Init Message */
13     memset(t, 0, sizeof(t));
14     spi_message_init(&m);
15     m.spi = spi;
16
17     /* Configure tx/rx buffers */
18     t[0].delay_usecs = 100;      // Delay for PSoC4 operation
19     t[0].tx_buf = &cmd;          // Write 'R' to tx-buffer
20     t[0].rx_buf = &data;          // Read data-char to data variable
21     t[0].len = 1;
22     spi_message_add_tail(&t[0], &m);
23
24     /* Transmit SPI Data (blocking) */
25     spi_sync(m.spi, &m);          // Transmit SPI data
26
27     *value = data;                // Move char to value
28
29 }
```

### 7.5.2 SPI API

SPI\_api er en klasse til at styre SPI kommunikationen. Den består af en række metoder beskrevet i afsnit 6.13.

Alle metoderne er stort set opbygget ens.

## Open

De har alle en `open()` metode der sørger for at åbne system filen der enten skal skrives eller læses fra. Hvis der sker en fejl i open, returnere den så en passende negativ fejl, i dette tilfælde fejlen `LOG_ERR`.

```

1 /* Open file */
2 fp = open("/dev/spi_dev", O_RDWR);
3 if(fp < 0){
4     printf("OPEN ERROR: %d\n", fp);
5     close(fp);
```

```

6         return -LOG_ERR;
7 }
```

## Close

Der er også altid en `close()` metode, der sørger for at lukke den åbne fil ned igen. Denne metode er også lagt i alt fejlhåndtering, for at undgå system fejl på Masteren, hvis der skulle ske fejl.

```

1 /* Close file */
2 close(fp);
```

## Write & Read

Ind i mellem disse er så en `write()` eller/og en `read()` metode, der sørger for at der bliver læst eller skrevet til Enheden via SPI. Her er nogle eksempler på metoderne.

`Write()` metoden bruges i starten af alle metoder til at vælge hvad der skal ske på Enheden ved at skrive en char kommando til den. Den bruges også, som her, til at skrive parametrene temp og humi til Enheden. Her modtages parametrene som floats, konverteres til chars og skrives enkeltvis til Enheden med en for-løkke.

```

1 int SPI_api::config(int unit, float temp, float humi)
2 {
3     char tempArray[6];    // T T T . T \0
4     char humiArray[4];    // F F F \0
5
6     /* Parse floats to CharArrays */
7     sprintf(tempArray, sizeof(tempArray), "%05.1f", temp);
8     sprintf(humiArray, sizeof(humiArray), "%03.0f", humi);
9 ...
10 ...
11 ...
12 /* Write temp to target without 0-termination */
13 for(int i = 0 ; i < 5 ; i++){
14     err = write(fp, &tempArray[i], dataLen);
15     if(err < 0){
16         printf("WRITE ERROR: %d\n", err);
17         close(fp);
18         return -CONF_ERR;
19     }
20 }
21
22 /* Write humidity to target without 0-termination */
23 for(int i = 0 ; i < 3 ; i++){
24     err = write(fp, &humiArray[i], dataLen);
25     if(err < 0){
26         printf("WRITE ERROR: %d\n", err);
27         close(fp);
28         return -CONF_ERR;
29     }
30 }
31 ...
32 ...
33 }
```

`Read()` metoden modtager chars fra Enheden og behandler dem. I koden nedenfor ses implementeringen for `getLog()` metoden som læser på en buffer på Enheden hvori der ligger Data og Fejl. Disse chars bliver så lagt i en `vector<string>` så programmet på

Masteren har en nem måde at håndtere fejl og log-data på. Vector og alle string variablerne nulstilles med `clear()` for en god ordens skyld, da der nødigt skulle stå noget forkert i dem inden de pushes med ny data.

Den første `read()` metode man støder på, læser bufferlængden fra Enheden, som der sørger for der ikke læses på noget data der ikke er tilstede, senere i metoden.

Så er der en for-løkke der starter med at kigge på om der er et 'D' for data eller 'E' for fejl. Disse laver hhv. 10 og 3 læsninger, som der bygges en string af, der pushes til en vector. Hvis der kommer et 'D' ligges det på den første plads i en string sammen med de næste 10 læsninger. Dette er tilsvarende ved en fejl, dog kun med 3 læsninger efter det læste 'E'.

```

1 int SPI_api::getLog(vector<string> &data, int * units, int size)
2 {
3 ...
4 ...
5 ...
6 char charArrayLen;           // Variable for charArray size from target
7 char charResult;            // Used for buffering read chars
8 string stringResult;
9     stringResult.clear();
10 string stringDataResult;
11     stringDataResult.clear();
12 string stringErrorResult;
13     stringErrorResult.clear();
14 vector<string> vectorResult;
15     vectorResult.clear();
16 ...
17 ...
18 ...
19
20 /* Read charArray length from target */
21 err = read(fp, &charArrayLen, dataLen);
22 if(err < 0){
23     printf("READ ERROR: %d\n", err);
24     close(fp);
25     return -LOG_ERR;
26 }
27
28 /* Vector Builder looking for D's or E's */
29 for(i = 1 ; i < charArrayLen ; i++){
30     err = read(fp, &charResult, dataLen);
31     if(err < 0){
32         printf("READ ERROR: %d\n", err);
33         close(fp);
34         return -LOG_ERR;
35     }
36
37     if(charResult == 'D'){
38         stringDataResult.push_back(charResult); // Push 'D' to stringDataResult
39
40         // Build rest of stringDataResult
41         for(int c = 0 ; c < dataStringLen ; c++){
42             err = read(fp, &charResult, dataLen);
43             if(err < 0){
44                 printf("READ ERROR: %d\n", err);
45                 close(fp);
46                 return -LOG_ERR;
47             }
48             stringDataResult.push_back(charResult);
49
50             // Error handling, prevent to read on non-existing data
51             if(i >= (charArrayLen-1)){

```

```

52             printf("Error in buffer from unit\n");
53             return -LOG_ERR;
54         }
55         i++; // Increment 1st for-loop counter
56     }
57
58     // Push string to vectorResult
59     vectorResult.push_back(stringDataResult);
60     stringDataResult.clear();
61 }
62
63 if(charResult == 'E'){
64     stringErrorResult.push_back(charResult); // Push 'E' to stringErrorResult
65
66     // Build rest of stringErrorResult
67     for(int c = 0 ; c < errorStringLen ; c++){
68         err = read(fp, &charResult, dataLen);
69         if(err < 0){
70             printf("READ ERROR: %d\n", err);
71             close(fp);
72             return -LOG_ERR;
73         }
74         stringErrorResult.push_back(charResult);
75
76         // Error handling, prevent to read on non-existing data
77         if(i >= (charArrayLen-1)){
78             printf("Error in buffer from unit\n");
79             return -LOG_ERR;
80         }
81         i++; // Increment 1st for-loop counter
82     }
83
84     // Push string to vectorResult and clear string
85     vectorResult.push_back(stringErrorResult);
86     stringErrorResult.clear();
87 }
88 }
```

### Clear buffer

Alle metoderne har også en Clear buffer funktion. Denne benytter `write()` metoden, til at skrive et 'C' til Enheden for at nulstille tx-bufferen. Den bruges også for at tilgå switchen for alle write-only metoderne.

```

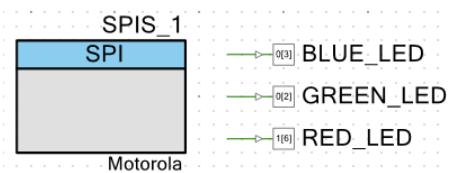
1 /* Clear buffer */
2 err = write(fp, &CL_BUF, 1);
3 if(err < 0){
4     printf("CLEAR ERROR: %d\n", err);
5     close(fp);
6     return -CONF_ERR;
7 }
```

Resten af implementeringen findes i bilag.

#### 7.5.3 SPI Handler

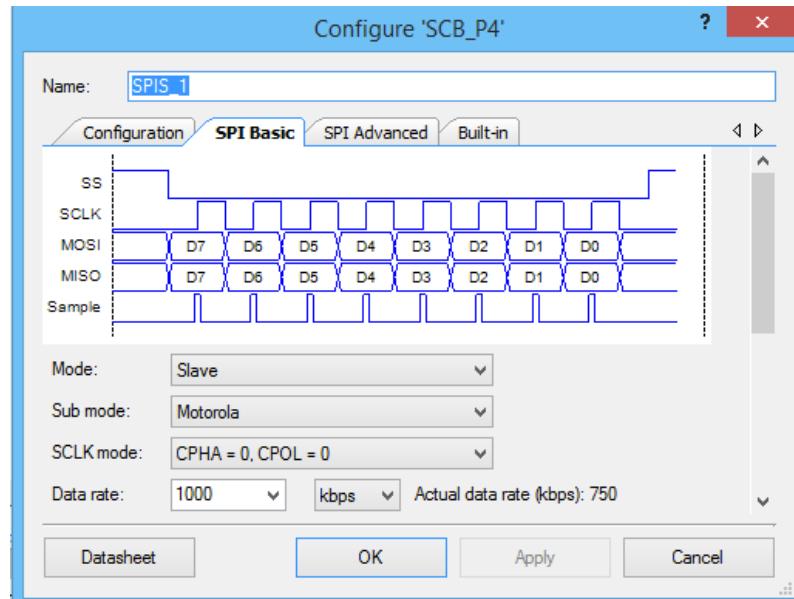
##### Top-design - PSoC Creator 3.0

Topdesignet (figur7.10) består af en SPI blok, samt 3 digitale output pins. Herunder beskrives hvordan elementerne i topdesignet skal indstilles.



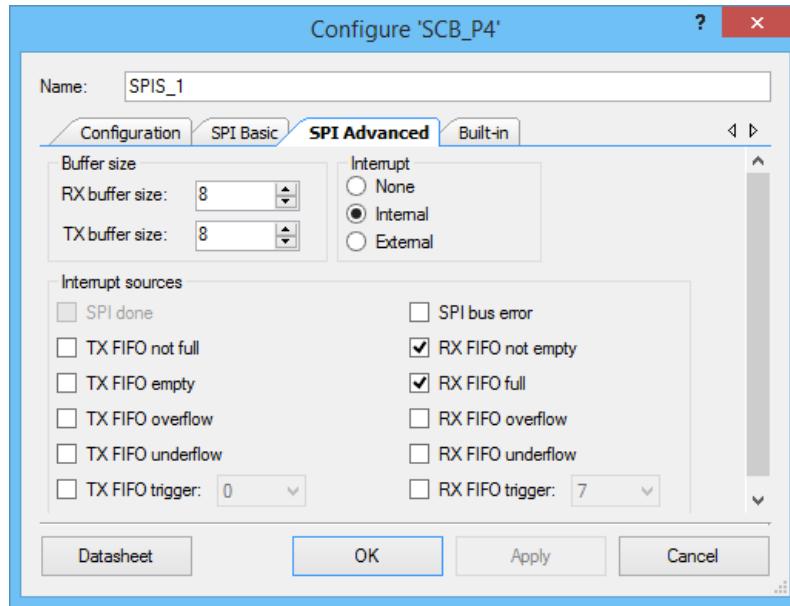
**Figur 7.10.** Topdesign for SPI og led

Basis indstillingerne for SPI blokken ses i figur 7.11. Mode sættes til slave, og SCLK mode sættes til CPOL=0, CPHA=0.



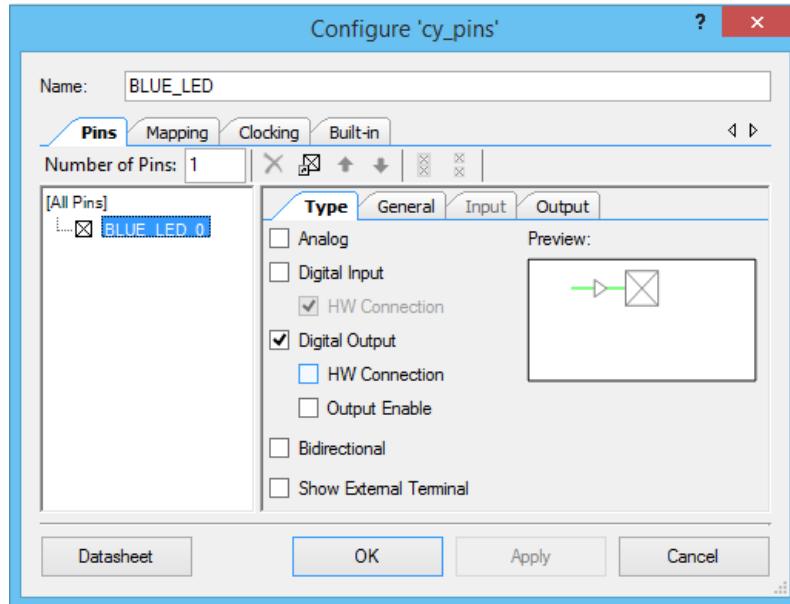
**Figur 7.11.** Konfigurering af SPI (SPI basic)

De avancerede indstillinger for SPI blokken ses i figur 7.12. Buffer size sættes til 8 bit for både RX og TX. Interruptet sættes til internt og interrupt kilden er RX FIFO not empty og RX FIFO full.



**Figur 7.12.** Konfigurering af SPI (SPI advanced)

De 3 output pins (GREEN\_LED, BLUE\_LED, RED\_LED) skal sættes til digital output uden hardware connection. Figur 7.13 viser hvordan indstillingen skal være, dette er gældende for alle 3 pins.



**Figur 7.13.** Konfigurering af pins

## ISR

SPI handleren består stort set kun af én interrupt rutine. Der tages udgangspunkt i de vigtige dele af rutinen. Og resten af koden kan ses i bilag.

Selv interrupt-rutinen er bygget op med en switch, der vælger en case alt efter input fra Masteren. Der tilgås kun switchen hvis der kommer en kommando der skal svare tilbage til

Master ('R', 'L', 'V') eller en Clear buffer kommando 'C', derefter kigges der på plads [0] i spiBuffer arrayet, og handles alt efter hvad der står der. Dvs. hvis Enheden skal aktiveres, sendes først et 'A' fra Master som læses ind i spiBuffer[] arrayet efterfulgt af et 'C' som tilgår switchen.

```

1 char spiBuffer[64];
2 ..
3 ..
4 ..
5 CY_ISR(isr_spi_rx) {
6
7     char cmd = '0';
8     ..
9     ..
10    cmd = SPIS_1_SpiUartReadRxData();
11    spiBuffer[spiCounter] = cmd;
12    spiCounter++;
13    ..
14    ..
15    if ((cmd == 'R') || (cmd == 'C') || (cmd == 'L') || (cmd == 'V')){
16        switch (spiBuffer[0]) {
17            ..
18            ..

```

Alle kommandoer der kræver et svar tilbage på SPI, bruger 'R' casen, men eftersom PSoC4 kræver noget tid<sup>3</sup> til at hente og skrive i tx-bufferen er koden lavet sådan at man er på forkant med en læsning. Dvs. når der laves en læsning 'R' fra Masteren, er der allerede skrevet til tx-bufferen, så man reelt læser data ud med det samme.

Det er derfor switchen tilgåes ved alle "Read" kommandoer, så hvor der f.eks. skal verificeres, laves der en læsning af Enhedens enhedsnummer. I eksemplet nedenfor er en global char brugt som enhedsnummer, denne læses ind i tx-bufferen når case 'V' køres. Og når så case 'R' køres, læses unitNo variablen først, hvorefter der skrives en ny værdi ind i tx-bufferen til næste 'R' køres.

```

1 char unitNo = '1';
2 int spiCounter = 0;
3 int spiReadCounter = 0;
4 ..
5 ..
6 ..
7 CY_ISR(isr_spi_rx) {
8     ..
9     ..
10    ..
11    case 'V':
12        ..
13        ..
14        ..
15        SPIS_1_SpiUartClearTxBuffer();
16        SPIS_1_SpiUartWriteTxData(unitNo);
17        spiCounter = 0;
18        break;
19        ..
20        ..
21        ..
22    case 'R':
23        SPIS_1_SpiUartClearTxBuffer();

```

<sup>3</sup>HAL Exercise 7: Oplyses det at PSoC4 kræver 50us, men dette er ikke et problem da der laves én læsning ad gangen og der er min. 300ms i mellem hver læsning

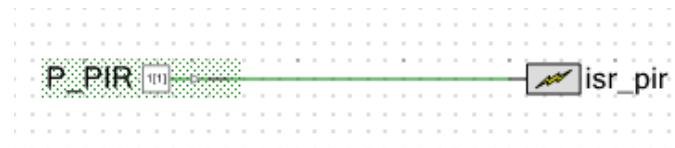
```

24     SPIS_1_SpiUartWriteTxData(spiTxBuffer[spiReadCounter]);
25     spiCounter = 0;
26     spiReadCounter++;
27     break;

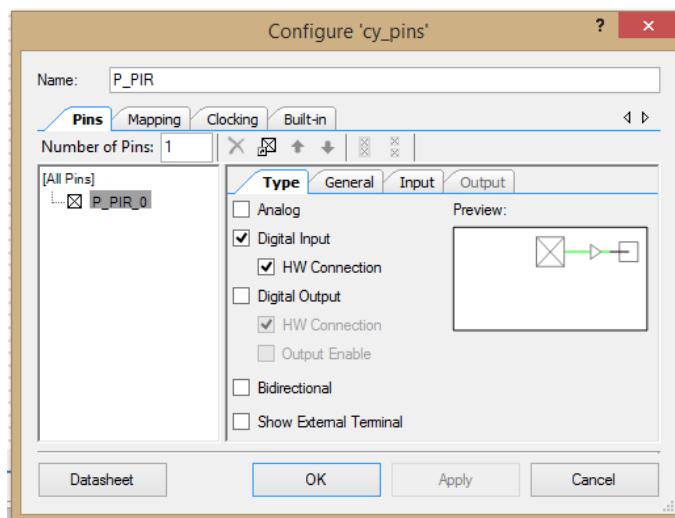
```

## 7.6 PIR API (SK MK)

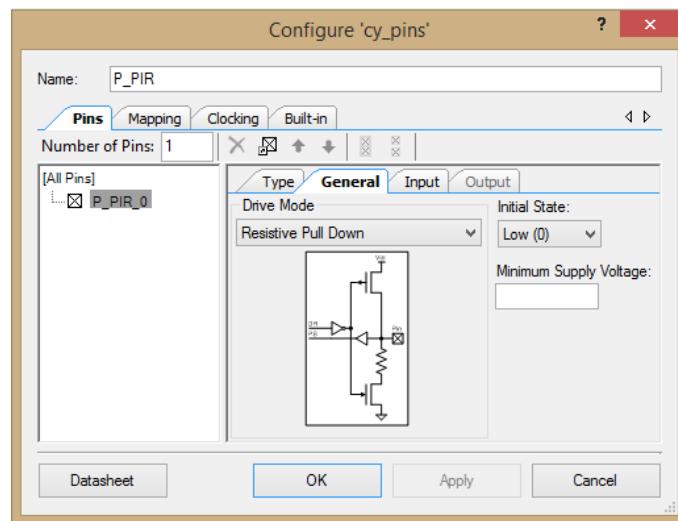
Pir API'en er implementeret i PSoC creator. Top designet består af en input pin (P1[1]) der modtager et signal fra pir-sensoren når der er bevægelse, se figur 7.14. På figur 7.15 og 7.16 ses opsætningen af P-PIR, den har en HW connection til isr\_pir og er indstillet til at have en pulldown modstand for at sikre at den står lav på pinen når der ikke er signal. Denne pin går ind og aktiverer en ISR routine. I main filen bliver global interrupt og pir interrupt aktiveret og i source filen ses det at selve ISR funktionen går ind og kalder metoden "movementDetekt();" fra controlleren, der deaktiverer sprinkleren og starter en timer på 30 min før der igen kan vandes.



*Figur 7.14.* Top Design for pir API



*Figur 7.15.* Konfiguration af pir pin



Figur 7.16. Konfiguration af pir pin

### Main fil

```
1 int main()
2 {
3
4     CyGlobalIntEnable; /* Her bliver global interrupts aktiveret. */
5
6     isr_pir_StartEx(P_PIR); /*Her bliver pir interruptet startet*/
7
8     for(;;)
9     {
10         /* Place your application code here. */
11     }
12 }
```

### Source fil

```
1 CY_ISR(P_PIR)
2 {
3     loadData_movementDetekt();
4 }
```



# Modultest 8

## 8.1 SPI (MK PO)

### 8.1.1 API

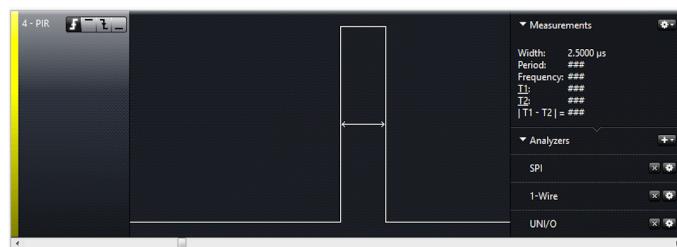
### 8.1.2 Handler

## 8.2 PIR (SK)

Modultesten er udarbejdet ved at lave et testprogram. Testprogrammet ses i source filen og sætter en test pin høj, figur 8.6 viser en test måling og det kan konkluderes at PSoC programmet fungerer efter hensigten, da billedet viser at programmet går korrekt ind i ISR rutinen.

### Source fil

```
1 CY_ISR(P_PIR)
2 {
3     TEST_Write(1);
4     TEST_Write(0);
5 }
```



Figur 8.1. Analyse billedet for pir aktivering

## 8.3 Tilslutningsprint (SK)

Der er lavet modultest på tilslutningsprintet ud fra denne tabel 8.3.

<b>Tilslutningsprint</b>				
	<b>Test</b>	<b>Forventet Resultat</b>	<b>Resultat</b>	<b>Kommentar</b>
<b>1</b>	Forsyningen til tilslutningsprintet fra PSU'en.	Der måles en værdi på 12 VDC +/- 0,5 VDC, imellem + og - ved komponenten.	12,02 VDC målt	
<b>2</b>	USB forsyningen fra tilslutningsprintet.	Der måles en værdi på 5 VDC +/- 0,5 VDC, imellem + og - ved komponenten.	5,02 VDC målt	Den målte værdi stemmer fint overens med det forventede resultat fra HW-design delen
<b>3</b>	Relætilslutningen fra tilslutningsprintet.	Der måles en værdi der ligger indenfor 3,7-8,8 VDC når sprinkler pin på PSoC tilslutningen bliver tilsluttet 3,3 VDC.	4,48 VDC målt	Spændingen er ikke 4VDC som beskrevet i design delen af relæet, da transistoren ikke er helt mættet
<b>4</b>	Pir forsyningen fra tilslutningsprintet	Der måles en værdi på 5 VDC +/- 0,5 VDC, imellem + og - ved tilslutningen.	5,02 VDC målt	
<b>5</b>	FT-sensor forsyningen fra tilslutningsprintet	Der måles en værdi på 3,3 VDC +/- 0,5 VDC, imellem + og - ved tilslutningen.	3,36 VDC målt	
<b>6</b>	Der testet om der er gennemgang fra select pin på PSoC tilslutningen til FT-sensor tilslutningen	Der måles gennemgang.	Gennemgang er målt	
<b>7</b>	Der testet om der er gennemgang fra analog pin på PSoC tilslutningen til FT-sensor tilslutningen	Der måles gennemgang.	Gennemgang er målt	

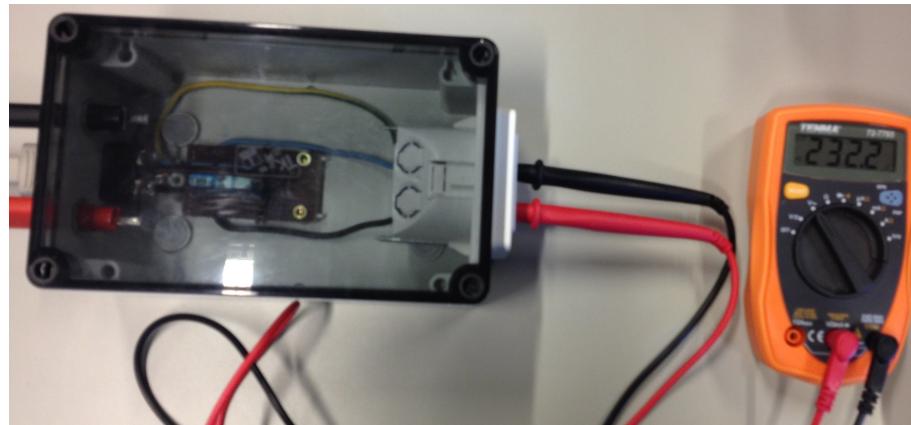
...fortsat fra forrige side

	Test	Forventet Resultat	Resultat	Kommentar
8	Der testet om der er gennemgang fra pir pin på PSoC tilslutningen til pir-sensor tilslutningen	Der måles gennemgang.	Gennemgang er målt	

## 8.4 230V relæ (PO)

Efter godkendelse af relæet ved Torben Lund Jensen er relæets funktion testet. Laboratoriets spændingsforsyning er indstillet til 4 V og 100mA som er det relæet modtager fra tilslutningsprintet. Apparatstikket tilsluttes en aktiv 230V stikkontakt, bananbøsningerne tilsluttes laboratorie spændingsforsyningen. Et multimeter indstilles til vekselstrøm om målepindene indsættes i stikkontakten på relækassen.

Figur 8.7 viser outputtet når relæet er klikket. Som det ses er stikkontakten tændt med en spænding på 232,2Vac. Når relæetafbrydes slukkes stikkontakten som forventet.

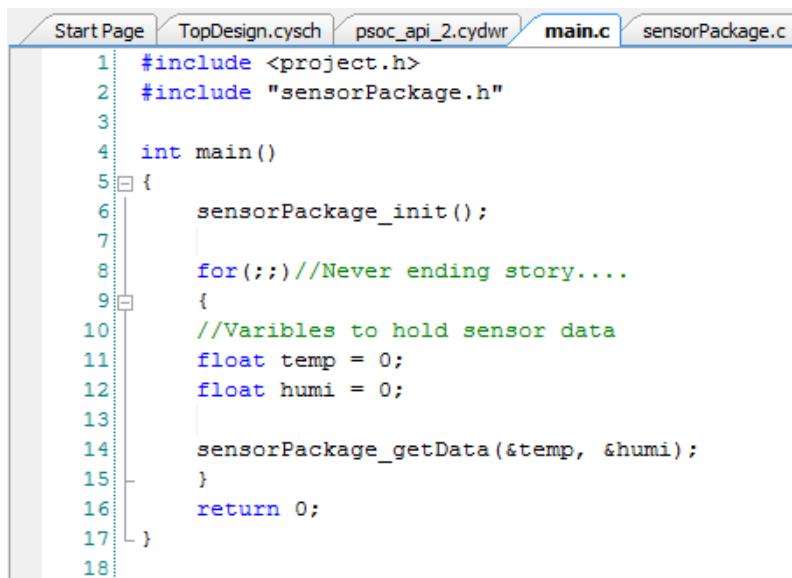


*Figur 8.2.* Aktivt relæ med 232 Vac på udgangen

## 8.5 PSoC API

PSoC APIen testet i PSoC creator med den indbyggede debug-funktion. I main filen skrives et lille test program som kan ses på figur 8.8.

Først initialiseres de nødvendige 'objekter' hvor opsætninger af komponenter også initialiseres. I en uendelig for-løkke oprettes to variabler til lagring af temperatur- og fugtighedsdata og disse sættes lige med nul. Herefter kaldes funktionen til aflæsning af data, denne kalder to funktioner, en for temperatur og en for fugtighed. Disse funktioner sørger selv for at sensorens select ben bliver sat lav for temperatur og høj for fugtighed.



```

1 #include <project.h>
2 #include "sensorPackage.h"
3
4 int main()
5 {
6     sensorPackage_init();
7
8     for(;;)//Never ending story....
9     {
10        //Variables to hold sensor data
11        float temp = 0;
12        float humi = 0;
13
14        sensorPackage_getData(&temp, &humi);
15    }
16    return 0;
17 }
18

```

**Figur 8.3.** main test program

ADC'en forbides til GND og VCC for at teste nogle værdier, dette vil give hvad der svarer til 0 og 100 % duty cycle.

Locals					
Name	Value	Address	Type	Radix	
temp	-47.0216026	0x20000FB4 (All)	float	Default	
humi	-6.12207127	0x20000FB0 (All)	float	Default	

**Figur 8.4.** Resultat for ADC forbundet til GND

Locals					
Name	Value	Address	Type	Radix	
temp	128.785919	0x20000FB4 (All)	float	Default	
humi	118.940186	0x20000FB0 (All)	float	Default	

**Figur 8.5.** Resultat for ADC forbundet til VCC

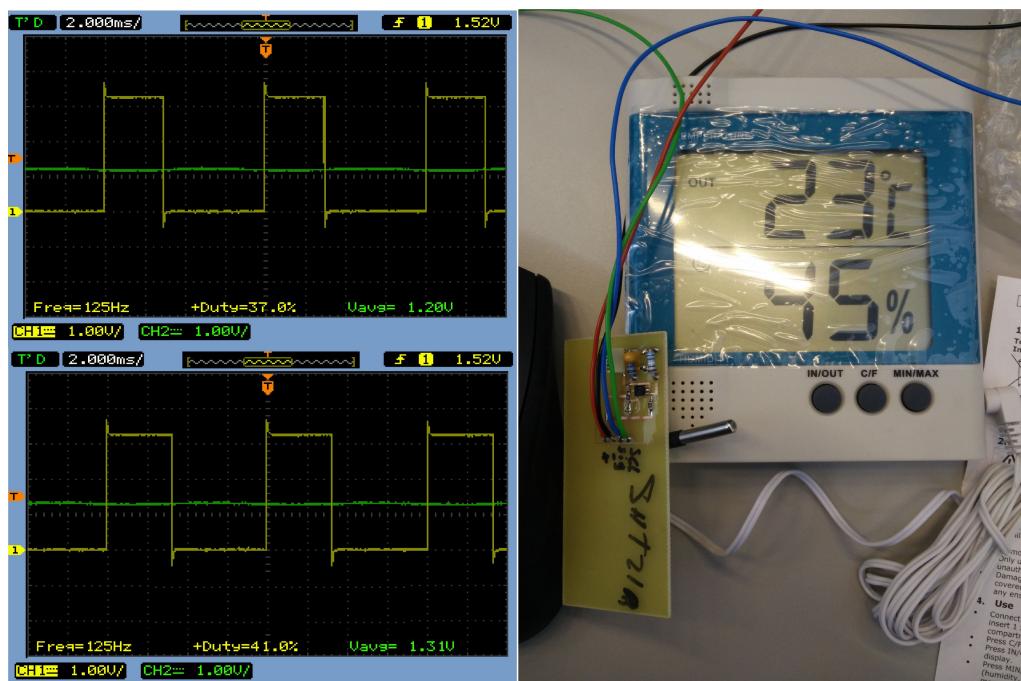
Under debug kan måleresultaterne aflæses som det ses på figur 8.9 og 8.10. For 0 % duty cycle vil *temp* = -46,85 og *fugt* = -6,0 i forhold til databladet. for 100 % duty cycle vil *temp* = 128,9 og *fugt* = 119 i forhold til databladet. Fugtigheden er, som nævnt under design, kun defineret ved 0...100 % duty cycle og ovenstående værdi er udelukkende teoretisk og bruges kun for at teste funktionen.

Disse værdier stemmer godt overens, men en minimal afvigelse, med de udlæste værdier for debug og testen kan derfor antages som godkendt.

## 8.6 Fugt- og temperatur-sensor print (JS LB)

I følgende afsnit testes temperatur- og fugtighedssensoren samt lavpasfilteret monteret på printet sammen med. Der testes for om temperatur og fugtighedsændringer registreres korrekt og om omregningen fra DC-niveau til værdi stemmer overens med scenariet i test. Scopbillederne er taget med et agilent oscilloskop.

Første test laves ved stuetemperatur. Til test af SHT21P sensoren bruges et færdigt termometer og fugtighedsmåler, af modellen MODEL PÅ BJØRNS TERMOMETER. På billedet ses det at begge sensorer er i samme rum og at de bliver udsat for samme forhold.



**Figur 8.6.** Billede af test incl. oscilloskop billede, øverste scopbillede er fugtighed og nederste er temperatur

Som scop billedet viser måles både en DC og et PWM signal. DCen er signalet efter lavpasfilteret og PWM'en er den rå data fra sensoren. Sensorens SCL-ben er sat højt og sensoren måler derfor fugtighed. PWM'en måles til at have en dutycycle på XX % og spændingen efter filteret måles til XX V. Ud fra databladet kan dutycyclen omregnes til fugtighed med følgende formel:

$$-6 + 125 * \text{dutycycle} \quad (8.1)$$

### LIGNING FOR FUGTIGHED MED PWM

Med følgende formel regnes fugtigheden ud fra DC-værdien.

### LIGNING FOR FUGTIGHED MED DC

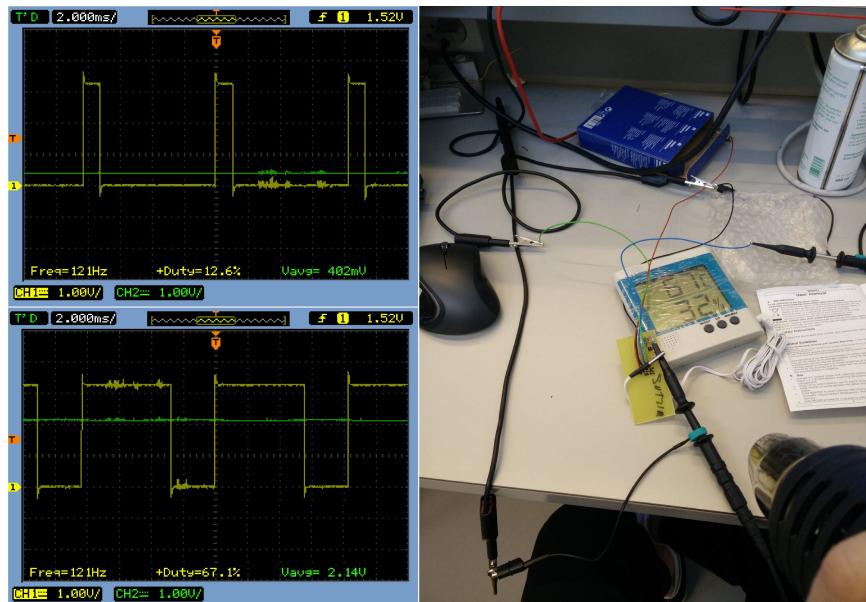
SCL-benet ligges nu lav og temperaturen måles. Ud fra PWM regnes nu temperaturen.

## LIGNING FOR TEMPERATUR MED PWM

Temperaturen regnes ud fra DCen som følger.

## LIGNING FOR TEMPERATUR MED DC

For at teste andre forhold bruges en varmepistol til at hæve temperatur og sænke fugtigheden. Samme målinger foretages:



**Figur 8.7.** Billede af test incl oscilloskop billede

Igen regnes værdierne for temperatur og fugtighed ud fra PWM'en med en dutycycle på XX % og DC'en med en amplitude på XX V

## LIGNING FOR FUGTIGHED MED PWM

Med følgende formel regnes fugtigheden ud fra DC-værdien.

## LIGNING FOR FUGTIGHED MED DC

SCL-benet ligges nu lav og temperaturen måles. Ud fra PWM regnes nu temperaturen.

## LIGNING FOR TEMPERATUR MED PWM

Temperaturen regnes ud fra DCen som følger.

## LIGNING FOR TEMPERATUR MED DC

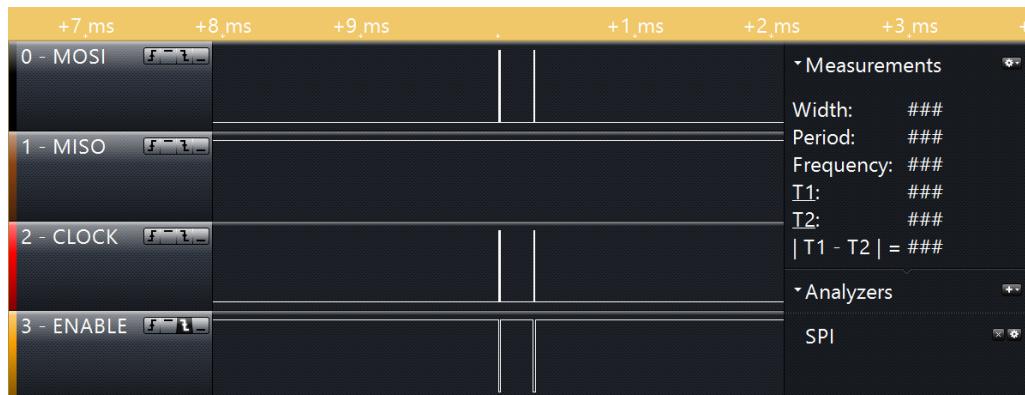
# Integrationstest 9

## 9.1 SPI (MK PO)

I det følgende testes SPI kommunikationen. Alle kommandoer testes for sig og dokumenteres herunder ved logik analyse billeder samt tabeller der er udarbejdet af eksporterede .txt filer fra Salae Logic<sup>1</sup>.

### Aktiver

Aktiver kommandoen testes vha. testprogrammet ved kommandoen:



*Figur 9.1.* Analyse billede for kommandoen aktiver

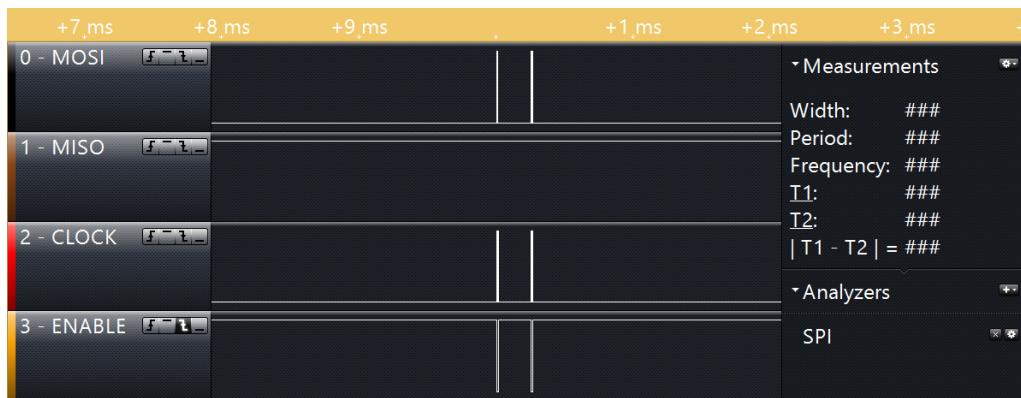
Tabel 9.1 viser dataoverførelsen. Som det ses har kun MOSI betydning for aktiver, da aktiver er en write metode.

*Tabel 9.1.* Analyse data eksporteret til tabel

Char nr	0	1
MOSI	'A'	'C'
MISO	'255'	'255'

**Deaktiver** Deaktiver kommandoen testes vha. testprogrammet ved kommandoen:

<sup>1</sup>Salae Logic er softwaren der benyttes til USB logic analyser. Det er herfra at analyse billeder samt data til tabellerne er fra



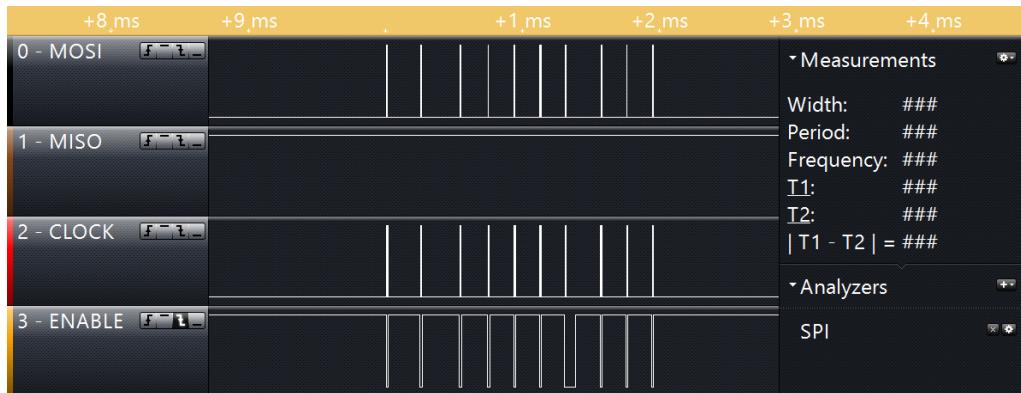
**Figur 9.2.** Analyse billede for kommandoen deaktiver

Tabel 9.2 viser dataoverførelsen. Som det ses har kun MOSI betydning for deaktiver, da deaktiver er en write metode.

**Tabel 9.2.** Analyse data eksporteret til tabel

Char nr	0	1
MOSI	'D'	'C'
MISO	'255'	'255'

**Config** Config kommandoen testes vha. testprogrammet ved kommandoen:



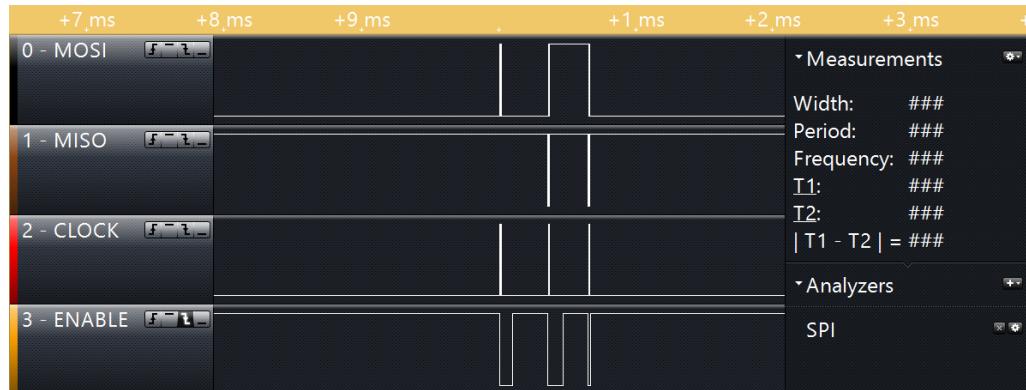
**Figur 9.3.** Analyse billede for kommandoen config

Tabel 9.3 viser dataoverførelsen. Som det ses har kun MOSI betydning for config, da config er en write metode.

**Tabel 9.3.** Analyse data eksporteret til tabel

Char nr	0	1	2	3	4	5	6	7	8	9
MOSI	'P'	'1'	'0'	'0'	'. '	'1'	'0'	'4'	'8'	'C'
MISO	'255'	'255'	'255'	'255'	'255'	'255'	'255'	'255'	'255'	'255'

**Verifier** Verificer kommandoen testes vha. testprogrammet ved kommandoen:



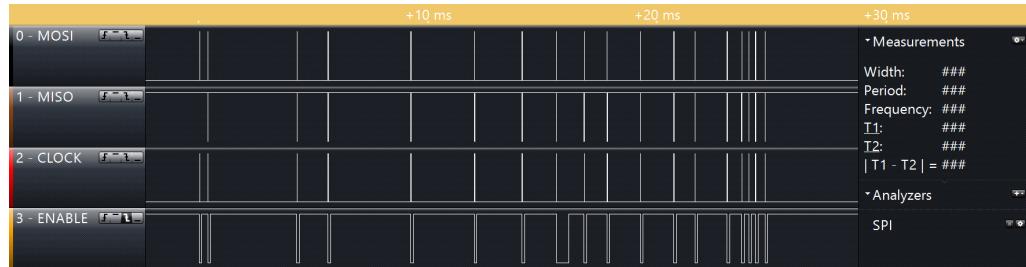
**Figur 9.4.** Analyse billede for kommandoen verificer

Tabel 9.4 viser dataoverførelsen. Som det ses overføres der data på både MOSI og MISO linjen. Det sker da verificer er en read-metode. Testen her er udført med tilkoblet Enhed. Hvis enheden kobles fra og der ikke modtages et matchende enhedsnummer, returneres en fejlkode i programmet.

**Tabel 9.4.** Analyse data eksporteret til tabel

Char nr	0	1	2
MOSI	'V'	'R'	'C'
MISO	'255'	'1'	'0'

**Log** Getlog kommandoen testes vha. testprogrammet og med en test buffer i form af et char array "DTTT.TFFFVBEXXX" på Enheden:



**Figur 9.5.** Analyse billede for kommandoen getLog

Tabel 9.5 og 9.6 viser dataoverførelsen.

**Tabel 9.5.** Analyse data eksporteret til tabel del 1

Char nr	0	1	2	3	4	5	6	7	8	9
MOSI	'L'	'R'	'R'	'R'	'R'	'R'	'R'	'R'	'R'	'R'
MISO	'255'	'16'	'D'	'T'	'T'	'T'	'.'	'T'	'F'	'F'

**Tabel 9.6.** Analyse data eksporteret til tabel del 2

Char nr	10	11	12	13	14	15	16	17
MOSI	'R'							
MISO	'F'	'V'	'B'	'E'	'X'	'X'	'X'	'O'

Der er ligeledes testet med et mere komplekst char array med flere data og error strenge. Disse læses også som de skal.

### Begrænsninger

Ovenstående modultest er udført med et 7cm langt fladkabel til forbindelsen mellem Master og Enhed. Der er testet med et 22cm langt fladkabel, her var der en fejlrate på mere end 10%. Ydermere er det nødvendigt at være opmærksom på ikke at lade nogle kabler krydse fladkablet, da dette giver støj og vil medføre fejl. Der er ikke oplevet fejl ved at benytte fladkablet på 7cm, så længe støjkilder holdes på afstand.

## 9.2 PIR (SK)

# Accepttestspezifikation

10

Punkterne i accepttestspezifikationen er skrevet ud fra punkterne i hovedforløbet for de enkelte usecases beskrevet i kapitel 2.3.

UC1: Tilføj/fjern enhed				
	Test	Forventet Resultat	Resultat	Godkendt/Kommentar
1	Bruger vælger ”Tilføj/fjern enhed” i hovedmenu	Visuel: Menu fremkommer på skærmen	N/A	N/A
2	Bruger vælger "Tilføj enhed"	Visuel: Menu for "Tilføj enhed" fremkommer på masters skærm	N/A	N/A
3	Der udføres visuelt en test for om listen med opsatte enheder fremkommer	Visuel: Listen vises for bruger med opsatte enheder	N/A	N/A
3.a	Der testes visuelt om indtastning er mulig	Felt for indtastningen er til rådighed for bruger	N/A	N/A
3.b	Bruger indtaster navn (hul 1) og adresse (0x01) for Enhed	Visuel: Den indtastede information vises på skærmen	N/A	N/A
3.b.a	Bruger indtaster navn: <tom> og adresse: 0xFF	Master giver fejlmeddeelse omkring ugyldig indtastning	N/A	N/A
3.c	Masterenhed tilfører Enhed til systemet med default parametre	Visuel: Enheden tilføjes med de indtastede informationer	N/A	N/A
3.d	Enheden forbindes til kommunikationsnetværket	Testes i 3.e	N/A	N/A
3.e	Master verificere forbindelse til Enhed og sender dato og tidspunkt	Visuel: Master viser godkendt Enhed	N/A	N/A

...fortsat fra forrige side

	<b>Test</b>	<b>Forventet Resultat</b>	<b>Resultat</b>	<b>Godkendt/ Kommentar</b>
<b>3e.a</b>	Forbindelsen mellem enhed og master afbrydes	Visuel: Master giver besked om mistet kommunikation	N/A	N/A
<b>4</b>	Bruger vælger "Fjern enhed"	Visuel: Menu for "Fjern enhed" fremkommer på skærmen	N/A	N/A
<b>5</b>	Der udføres visuelt en test for, om listen med opsatte enheder fremkommer	Listen præsenteres på skærmen	N/A	N/A
<b>5.a</b>	Bruger indtaster adresse (0x01) på Enhed der ønskes fjernet	Visuel: Bekræftelse vises på skærmen	N/A	N/A
<b>5.b</b>	Master deaktivere Enhed	Visuel: Adressen accepteres og Enhed deaktivieres	N/A	N/A
<b>5.c</b>	Der testes visuelt at master giver information omkring at Enhed er slettet fra systemet	Al information om enhed slettes fra systemet	N/A	N/A
<b>6</b>	Der testes visuelt at enheden er fjernet fra listen over opsatte enheder	Den opdateret liste præsenteres på skærmen	N/A	N/A
<b>7</b>	Brugeren vælger at returnere til hovedmenuen	Visuel: Master returnerer til hovedmenuen	N/A	N/A

#### UC2: Konfig

	<b>Test</b>	<b>Forventet Resultat</b>	<b>Resultat</b>	<b>Godkendt/ Kommentar</b>
<b>1</b>	Bruger vælger "Konfig" i hovedmenuen	Visuel: Liste over opsatte enheder fremkommer på skærmen	N/A	N/A
<b>2</b>	Bruger vælger en Enhed der ønskes omkonfigureret	Visuel: Menu for konfiguration for valgt Enhed vises på skærmen	N/A	N/A
<b>2.a</b>	Bruger vælger at afbryde konfigurationen	Visuel: Master forlader menuen og går tilbage til hovedmenuen	N/A	N/A

...fortsat fra forrige side

	<b>Test</b>	<b>Forventet Resultat</b>	<b>Resultat</b>	<b>Godkendt/ Kommentar</b>
<b>3</b>	Bruger indtaster nye værdier for den valgte enhed	Indstillingsmulighederne er tilgængelige	N/A	N/A
<b>3.a</b>	Der indtastes temperatur: -273 og fugt: -1	Visuel: Fejlmeddeelse vises på skærmen	N/A	N/A
<b>4</b>	Bruger gemmer de ønskede indstillinger gemmes	Visuel: Indstillingerne for den valgte enhed gemmes	N/A	N/A
<b>4.a</b>	Bruger vælger at afbryde konfigurationen	Visuel: Indstillingerne slettes og master går tilbage til hovedmenu. Master gemmer ikke indstillingerne	N/A	N/A
<b>5</b>	Der testes at den valgte enhed reagerer på de nye indstillinger	Enheden opererer efter de nye parameter	N/A	N/A

### UC3: Aktiver/Deaktiver

	<b>Test</b>	<b>Forventet Resultat</b>	<b>Resultat</b>	<b>Godkendt/ Kommentar</b>
<b>1</b>	Bruger vælger ”Aktiver/Deaktiver” i hovedmenuen	Visuel: Menu for ”Aktiver/Deaktiver” fremkommer på skærmen	N/A	N/A
<b>2</b>	Bruger markere en opsat Enhed i menuen	Visuel: Den valgte bliver markeret	N/A	N/A
<b>3</b>	Bruger vælger ”Aktiver” for den valgte enhed	Visuel: Enhed indikere at den er aktiv  Der analyseres på SPI kommunikationen at Enhed modtager Aktivering	N/A	N/A
<b>3</b>	Bruger vælger ”Deaktiver” for den valgte enhed	Visuel: Enhed indikere at den er deaktiv  Der analyseres på SPI kommunikationen at Enhed modtager Deaktivering	N/A	N/A

...fortsat fra forrige side

	<b>Test</b>	<b>Forventet Resultat</b>	<b>Resultat</b>	<b>Godkendt/ Kommentar</b>
<b>3.a</b>	Bruger vælger at afbryde handlingen	Visuel: Master returnerer til hovedmenuen	N/A	N/A
<b>4</b>	Master udskriver på skærmen, at ønsket Enhed er aktiveret eller deaktiveret	Visuel: Besked om aktivering eller deaktivering vises på skærmen	N/A	N/A
<b>5</b>	Bruger vælger "Afslut"	Visuel: Master returnerer til hovedmenuen	N/A	N/A
<b>6</b>	Der testes visuelt at Master går tilbage til hovedmenu	Hovedmenuen præsenteres for bruger	N/A	N/A
<b>3.a</b>	Bruger vælger "Afbryd"	Visuel: Hovedmenu vises	N/A	N/A

<b>UC4: Databehandling</b>				
	<b>Test</b>	<b>Forventet Resultat</b>	<b>Resultat</b>	<b>Godkendt/ Kommentar</b>
<b>1</b>	Der laves en testopstilling hvori det er muligt at ændre omgivelserne til sensoren således at den udlæste data ændres	Visuel: Udlæst data svarer overens med fysiske ændringer	N/A	N/A
<b>2</b>	Der ændres på omgivelserne således at grænseværdier for Enheden overskrides	Pumpe starter og begynder at vande	N/A	N/A
<b>2.a</b>	PIR-sensoren påvirkes med bevægelse	Vandingen bliver afbrudt på baggrund af registreret bevægelse.	N/A	N/A
<b>3</b>	Der testes visuelt at Enhed har gemt behandlerne af data i buffer	Data er gemt og kan tilgås via Masterens log	N/A	N/A
<b>4</b>	Master henter information fra bufferen i Enhed	Testes i 5	N/A	N/A
<b>5</b>	Bruger tilgår log i Master	Visuel: Informationen ligger tilgængelig i loggen	N/A	N/A

...fortsat fra forrige side

	<b>Test</b>	<b>Forventet Resultat</b>	<b>Resultat</b>	<b>Godkendt/ Kommentar</b>
<b>6</b>	Ved test måles det at der sendes et signal til Enhed ved registreret bevægelse	Enhed registrer signal fra PIR-sensor	N/A	N/A
<b>7</b>	Ved test ændres værdier for Enhed således at denne ville starte en vanding	Vandingen er ikke mulig i 30 min.	N/A	N/A
<b>8</b>	Der testes visuelt at Enhed har gemt behandlingerne af data i buffer	Testes i 5	N/A	N/A

<b>UC5: Tjek status</b>				
	<b>Test</b>	<b>Forventet Resultat</b>	<b>Resultat</b>	<b>Godkendt/ Kommentar</b>
<b>1</b>	Bruger vælger ”Tjek status” på Master	Visuel: Det er muligt at vælge ”Tjek status”	N/A	N/A
<b>2</b>	Der testes visuelt at Master giver information omkring status	Master henter seneste status fra log-register og udskriver dette på skærmen	N/A	N/A
<b>3</b>	Bruger vælger ”Tilbage”	Visuel: Det er muligt at vælge ”Tilbage”	N/A	N/A

<b>UC6: Udskriv log</b>				
	<b>Test</b>	<b>Forventet Resultat</b>	<b>Resultat</b>	<b>Godkendt/ Kommentar</b>
<b>1</b>	Bruger vælger ”Udskriv log” på Master	Visuel: Det er muligt at vælge ”Udskriv log”	N/A	N/A
<b>2</b>	Det testes visuelt at loggen udskrives på Master	Informationen vises på skærmen	N/A	N/A
<b>3</b>	Bruger vælger muligheden ”Tilbage”	Visuel: Det er muligt at vælge ”Tilbage”	N/A	N/A

<b>Ikke-funktionelle krav</b>				
	<b>Test</b>	<b>Forventet Resultat</b>	<b>Resultat</b>	<b>Godkendt/ Kommentar</b>
<b>1</b>	Systemet skal opsættes af autoriseret personale	Ikke testbar	N/A	N/A
<b>2</b>	Udenforstående bruger gennemlæser manual	Bruge har nu information nok til at kunne betjene systemet	N/A	N/A
<b>3</b>	Systemet skal kunne have en MTBF på min. 2 år	Ikke testbar	N/A	N/A
<b>4</b>	Software oppe tid > 1 måned, uden nødvendigt genstart	Ikke testbar	N/A	N/A
<b>5</b>	Der oprettes 18 Enheder i Master med unikke adresser	Ikke testbar. Der laves kun én Enhed	N/A	N/A
<b>6</b>	Sprinkler indstilles til at vande et areal af 360 grader	Sprinkler vander arealet 360 grader med min. 3,5 m radius	N/A	N/A
<b>7</b>	Der tilsluttes en yderligere Enhed efter opsætning af systemet	Ikke testbar. Der laves kun én Enhed	N/A	N/A
<b>8</b>	Systemet køre i 15 minutter. Der kontrolleres visuelt at loggen indeholder målinger efter det 15. minut	Loggen indeholder data fra tilkoblet Enhed.	N/A	N/A
<b>9</b>	Testet i punkt 8.	N/A	N/A	N/A
<b>10</b>	Sprinkler udskiftes uden det er nødvendigt at tilgå anden hardware	Sprinkler udskiftes og punkt 6 udføres igen	N/A	N/A
<b>11</b>	Personale skal nemt kunne tilgå sprinklere	Sprinklere er monteret således at disse let kan tilgås af personale	N/A	N/A
<b>12</b>	Enhed opsættes og forbindelse til Master afbrydes	Enheden opererer autonomt	N/A	N/A
<b>13</b>	Måling af lufttemperatur udføres ved stuetemperatur	Korrekt data for temperatur sendes til Master fra Enhed	N/A	N/A





# **Bilag (CD-indhold)**

11

---