



## AARHUS SCHOOL OF ENGINEERING

SUNDHEDSTEKNOLOGI  
3. SEMESTERPROJEKT

---

## Dokumentation

---

### *Gruppe 4*

Mads Fryland Jørgensen (201403827)  
Jeppe Tinghøj Honeré (201371186)  
Nicoline Hjort Larsen(201405152)  
Freja Ramsing Munk (201406736)  
Sara-Sofie Staub Kirkeby (201406211)  
Tine Skov Nielsen (201404233)

### *Vejleder*

Projektvejleder  
Thomas Nielsen  
Aarhus Universitet

15. december 2015



*Gruppemedlemmer*

Mads Fryland Jørgensen (201403827)	Dato
Jeppe Tinghøj Honoré (201371186)	Dato
Freja Ramsing Munk (201406736)	Dato
Nicoline Hjort Larsen (201405152)	Dato
Sara-sofie Staub Kirkeby (201406211)	Dato
Tine Skov Nielsen (201404233)	Dato

*Vejleder*

Thomas Nielsen	Dato
----------------	------

# Ordliste

---

Ord	Forklaring
BDD	Blok Definition Diagram
DAQ	Data Acquisition
(F)URPS+	Functionality, Usability, Reliability, Performance & Supportability
GUI	Graphic User Interface
IBD	Internal Blok Diagram
IHA	Ingeniør Højskolen Aarhus
UML	Unified Modeling Language
SQL	Structured Query Language

# Indholdsfortegnelse

---

<b>Ordliste</b>	<b>ii</b>
<b>Kapitel 1 Indledning</b>	<b>1</b>
<b>Kapitel 2 Kravspecifikation</b>	<b>2</b>
2.1 Indledning . . . . .	2
2.2 Funktionelle krav . . . . .	3
2.2.1 Aktør-kontekst diagram . . . . .	3
2.2.2 Aktørbeskrivelse . . . . .	3
2.3 Use cases . . . . .	4
2.3.1 Use case diagram . . . . .	4
2.3.2 Use case 1 . . . . .	4
2.3.3 Use case 2 . . . . .	5
2.3.4 Use case 3 . . . . .	6
2.3.5 Use case 4 . . . . .	6
2.3.6 Use case 5 . . . . .	7
2.3.7 Use case 6 . . . . .	8
2.4 Ikke-funktionelle krav . . . . .	8
2.4.1 (F)URPS+ . . . . .	8
<b>Kapitel 3 Design</b>	<b>11</b>
3.1 Indledning . . . . .	11
3.2 Hardware arkitektur . . . . .	11
3.2.1 Design af forstærker . . . . .	12
3.2.2 Design af analogfilter . . . . .	14
3.2.3 Grænseflader . . . . .	16
3.3 Software arkitektur . . . . .	17
3.3.1 GUI . . . . .	17
3.3.2 Domænemodel . . . . .	19
3.3.3 Appliktionsmodel . . . . .	20
<b>Kapitel 4 Implementering</b>	<b>27</b>
4.1 Hardware implementering . . . . .	27
4.1.1 Hardware overvejelser . . . . .	28
4.2 Software implementering . . . . .	30
4.2.1 3-lagsmodellen . . . . .	30
4.2.2 Tråde . . . . .	35
4.2.3 Observer . . . . .	37
4.2.4 UML klassediagram . . . . .	40
4.2.5 Kode udsnit & diagrammer . . . . .	43

<b>Kapitel 5 Test</b>	<b>52</b>
5.1 Indledning . . . . .	52
5.2 Modultest hardware . . . . .	52
5.2.1 Modul test af forstærkeren . . . . .	52
5.2.2 Modul test af det analoge filter . . . . .	55
5.3 Integrationstest hardware . . . . .	58
5.3.1 Integrationstest for forstærker og analogt filter . . . . .	58
5.3.2 Integrationstest med vandsøjle . . . . .	60
5.4 Modultest software . . . . .	62
5.5 Integrationstest software . . . . .	62
5.6 Integrationstest system . . . . .	63
<b>Kapitel 6 Accepttest</b>	<b>66</b>
6.1 Indledning . . . . .	66
6.2 Accepttest af Use Cases . . . . .	66
6.2.1 Use Case 1 . . . . .	66
6.2.2 Use Case 2 . . . . .	67
6.2.3 Use Case 3 . . . . .	69
6.2.4 Use Case 4 . . . . .	69
6.2.5 Use Case 5 . . . . .	69
6.2.6 Use Case 6 . . . . .	70
6.3 Accepttest af ikke-funktionelle krav . . . . .	71
<b>Bilagsliste</b>	<b>74</b>
Bilag 1 - Modul og integrationstest af hardware . . . . .	74
Bilag 2 - Logbog . . . . .	74
Bilag 3 - Mødereförart . . . . .	74
Bilag 4 - Kode . . . . .	74
Bilag 5 - Samarbejds aftale . . . . .	74
Bilag 6 - Blodtrykssignal . . . . .	74
Bilag 7 - Hjemmesider . . . . .	74
Bilag 8 - Scrum Pivotal Tracker . . . . .	74
Bilag 9 - Tidsplan . . . . .	74
Bilag 10 - Udviklingsværktøjer . . . . .	75
Bilag 11 - Hæmodynamik og hjertekarsygdomme . . . . .	75
Bilag 12 - Vejledning til dokumentation af semesterprojekter . . . . .	75
Bilag 13 - National Instruments kodeudsnit . . . . .	75
<b>Litteratur</b>	<b>76</b>

# Indledning 1

---

## Ansvarsområde

### Initialer:

JTH - Jeppe Tinghøj Honoré  
TSN - Tine Skov Nielsen  
SSK - Sara-Sofie Staub Kirkeby  
NHL - Nicoline Hjort Larsen  
FRM - Freja Ramsing Munk  
MFJ - Mads Fryland Jørgensen

Afsnit	Ansvarlig
Kravspecifikation	Alle
Alt hardware-relateret	SSK, NHL, FRM
Alt software-relateret	JTH, TSN, MFJ
Accepttest	Alle

Det følgende dokument er en dokumentation, som ligger til baggrund for projektrapporten og omhandler udvikling af et blodtrykssystem.

I denne dokumentation er kravspecifikationen, hovedscenariet og use cases beskrevet. De ikke-funktionelle krav er blevet beskrevet ved hjælp af vurderingsmetoden FURPS+, og er efterfulgt af en beskrivelse af de ikke-funktionelle krav. Kravene er efterfulgt af en opstilling af alle use cases, hvor de bliver beskrevet med fully-dressed use cases, med tilhørende extensions.

Denne dokumentation indeholder desuden en dybdegående beskrivelse af udviklingen af software- og hardware-komponenter. Disse er beskrevet detaljeret igennem alle tre udviklingsfaser, design, implementering og test. For softwaren, er use casene blevet beskrevet ved hjælp af en domænemodel og sekvensdiagrammer. Herudover er systemet yderligere beskrevet ved hjælp af flere SysML-diagrammer og skitser af interfaces.

Som en afslutning er der beskrevet en accepttest, med det formål at teste de opstillede krav. Accepttesten omfatter alle use cases og alle de ikke-funktionelle krav. Denne har til formål at understøtte systemets funktionalitet.

# Kravspecifikation 2

---

Version	Dato	Ansvarlig	Beskrivelse
0.1	8/09 2015	Alle	Oprettelse af dokument
0.2	21/09 2015	Alle	Første udkast til kravspecifikation og use cases
0.3	24/09 2015	Alle	Fully-dressed use cases
0.5	30/09 2015	Alle	Udarbejdelse af ikke-funktionelle krav
0.6	07/10 2015	Alle	Kravspecifikation rettelser efter review
1.0	04/11 2015	TSN, JTH, MFJ	Nye fully-dressed use cases, mere software relaterede

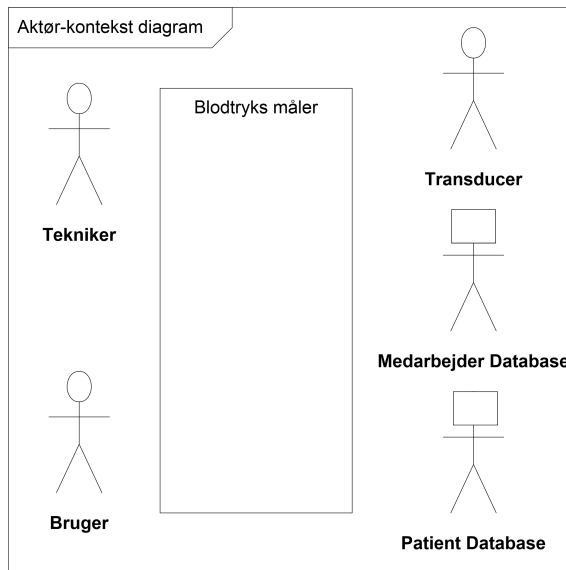
## 2.1 Indledning

Et blodtryksmålingssystem er blevet udarbejdet, der helt generelt er opbygget af en hardware-del og en software-del. Hardware-delen er opbygget af en forstærker, transducer og analogt filter hvis formål er at kunne forstærke og filtrere blodtrykssignalet og software-delen kalibrerer, nulpunktsjusterer og analyserer signalet, samt viser signalet grafisk på en brugergrænsefald.

Kravspecifikationen beskriver produktets kunnen ud fra opstillede funktionelle og ikke-funktionelle krav til systemet. De ikke-funktionelle krav er krav, der er opstillet til selve systemet, og de funktionelle krav er beskrevet i use cases, som omhandler systemets aktiviteter i forhold til aktører og mål.

## 2.2 Funktionelle krav

### 2.2.1 Aktør-kontekst diagram



Figur 2.1: Aktør-kontekst diagram

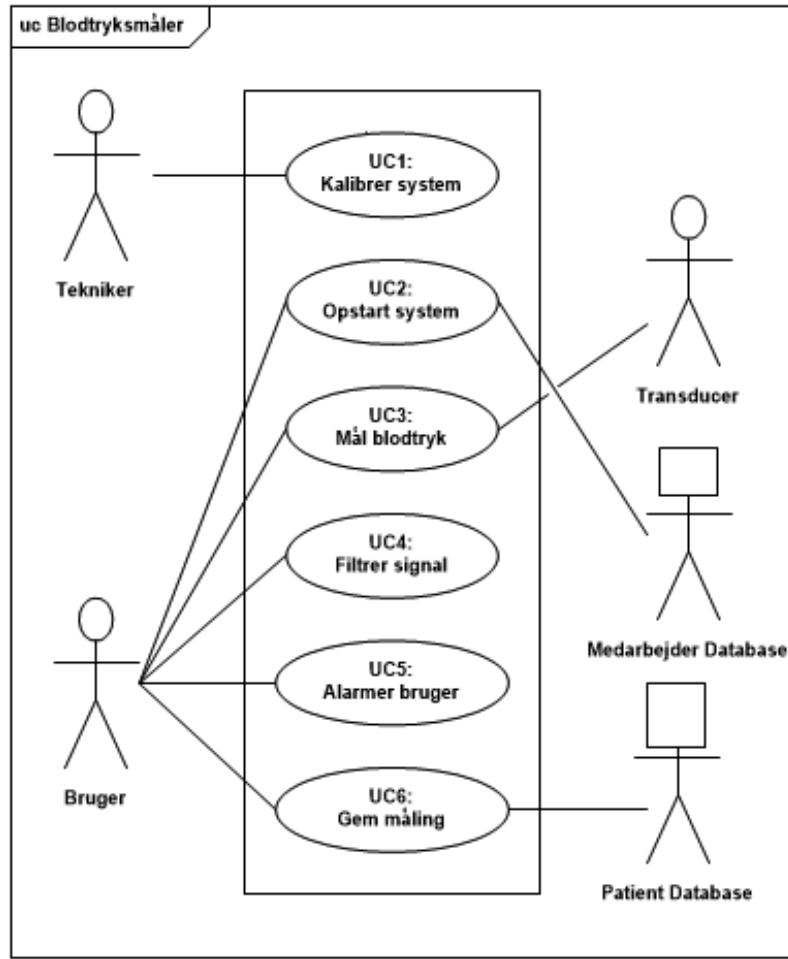
### 2.2.2 Aktørbeskrivelse

Aktørnavn	Type	Beskrivelse
Bruger	Primær	Brugeren er den aktør, der foretager blodtryksmålingerne. Brugeren er en person, der har kendskab til systemet, samt tilladelse til at benytte systemet. F.eks. en læge eller anæstesisygeplejerske
Tekniker	Primær	Tekniker er den aktør, der foretager den årlige kalibrering af systemet. Teknikeren er en person, der har kendskab til den tekniske del af systemet. F.eks. en medicotekniker på et sygehus
Transducer	Sekundær	Transducere omformer blodtryk til et elektrisk signal
Medarbejder database	Sekundær	Medarbejder-databasen er det sted, hvor medarbejderens login valideres
Patient-database	Sekundær	Patient-databasen er det sted, hvor blodtryksmålingens data gemmes og patientens CPR-nummer valideres

Tabel 2.2: Aktørbeskrivelse

## 2.3 Use cases

### 2.3.1 Use case diagram



Figur 2.2: Use case diagram

### 2.3.2 Use case 1

#### Use case 1

---

Navn	Kalibrer system
Use case ID	1
Samtidige forløb	1
Primær aktør	Tekniker
Initialisere	Tekniker
Mål	Tekniker ønsker at foretage kalibrering
Forudsætninger	Tekniker har adgang til systemet

Resultat	Systemet er kalibreret
Hovedforløb	<ol style="list-style-type: none"> <li>1. Tekniker mäter och antecknar atmosfärisk tryck</li> <li>2. Tekniker trycker på systemet tre kända tryck</li> <li>3. Tekniker läser svarerna</li> <li>4. Tekniker antecknar avvikelse från de kända tryck [4.a <i>Det finns ingen avvikelse</i>]</li> <li>5. Tekniker kalibrerar i förhållande till avvikelsen</li> </ol>
Undtagelser	4.a Use case avslutas

Tabel 2.3: Fully-dressed use case 1

### 2.3.3 Use case 2

#### Use case 2

Navn	Opstart system
Use case ID	2
Samtidige forløb	1
Primær aktør	Brugeren
Sekundær aktør	Medarbejder-database
Initialisere	Brugeren
Mål	Systemet er opstartet
Forudsætninger	Forbindelse til database
Resultat	Systemet er nulpunktsjusteret og brugeren er klar til at blive forbundet til systemet
Hovedforløb	<ol style="list-style-type: none"> <li>1. Brugeren indstiller transduceren til att mäta det atmosfäriska trycket</li> <li>2. Brugeren trycker på "nulstil"-knappen. Systemet laver nulpunktsjustering</li> <li>3. Brugeren indstiller transduceren till att mäta blodtryck</li> <li>4. Brugeren indtaster login-uppgifter och trycker på "Log in"-knappen. Systemet kontrollerar i databasen om uppgifterna är giltiga [4.a <i>Felaktig login</i>]</li> </ol>

Undtagelser      4a. Besked om forkert login vises. Use case fortsættes fra punkt 4

---

*Tabel 2.4: Fully-dressed use case 2*

### 2.3.4 Use case 3

#### Use case 3

---

Navn	Mål blodtryk
Use case ID	3
Samtidige forløb	1
Primær aktør	Brugeren
Initialisere	Brugeren
Mål	Blodtryksmåling er igangsat
Forudsætninger	UC2 er gennemført
Resultat	Blodtrykket vises kontinuerligt i en graf. Puls, systoliske og diastoliske værdier vises i blodtryksvinduet
Hovedforløb	<ol style="list-style-type: none"> <li>1. Brugeren trykker på "Start"-knappen i blodtryksvindue</li> <li>2. Transduceren mäter blodtryk</li> <li>3. Systemet modtager blodtryksmåling fra transduceren</li> <li>4. Blodtrykgraf, systolisk, diastolisk og puls vises grafisk i blodtryksvinduet</li> </ol>

---

*Tabel 2.5: Fully-dressed use case 3*

### 2.3.5 Use case 4

#### Use case 4

---

Navn	Filtrer signal
Use case ID	4
Samtidige forløb	2
Primær aktør	Brugeren
Initialisere	Brugeren

Mål	Der kan foretages en digital filtrering af signalet
Forudsætninger	UC3 er gennemført
Resultat	Det filtrerede signal vises i blodtryksgrafen
Hovedforløb	<ol style="list-style-type: none"> <li>1. Brugerens trykker på "Til"-knappen under filter</li> <li>2. Systemet filtrerer signalet</li> <li>3. Det filtrerede signal vises i blodtryksvindue</li> </ol>

Tabel 2.6: Fully-dressed use case 4

### 2.3.6 Use case 5

#### Use case 5

Navn	Alarmer bruger
Use case ID	5
Samtidige forløb	2
Primær aktør	Brugerens
Initialisere	Brugerens
Mål	Systemet kan alarmere bruger ved for højt/lavt blodtryk
Forudsætninger	UC3 er gennemført
Resultat	Systemet alarmerer bruger
Hovedforløb	<ol style="list-style-type: none"> <li>1. Brugerens tilpasser diastoliske og systoliske grænseværdier ud fra patientens normale blodtryk</li> <li>2. Systemet tjekker om grænseværdier er overskredet</li> <li>3. Målte blodtryk overskrider valgte grænseværdier [3.a Målte blodtryk overskrider ikke valgte grænseværdier]</li> <li>4. Alarm startes [4.a Bruger udskyder alarm]</li> </ol>
Undtagelser	3.a Use case fortsættes fra punkt 2 4.a Alarm udskydes med 3 minutter, og use case startes fra punkt 2 igen

*Tabel 2.7: Fully-dressed use case 5*

### 2.3.7 Use case 6

#### Use case 6

Navn	Gem måling
Use case ID	6
Samtidige forløb	1
Primær aktør	Brugeren
Mål	Brugeren ønsker at afslutte systemet og gemme måling
Forudsætninger	UC3 er gennemført
Resultat	Blodtryksmålingens data er gemt i database, og bruger er logget ud af systemet
Hovedforløb	<ol style="list-style-type: none"> <li>1. Brugeren trykker på "Afslut"-knappen. "Gem"- vindue åbnes. [1.a <i>Bruger ønsker ikke at afslutte</i>]</li> <li>2. Brugeren indtaster CPR-nummer og trykker på "Gem"-knappen [2.a <i>CPR-nummer er ikke gyldigt</i>]</li> <li>3. Besked om at data er gemt vises. Brugeren trykker på "Ok"-knappen. Systemet logger ud, og use case 2 startes</li> </ol>
Undtagelser	<ol style="list-style-type: none"> <li>1.a. Bruger trykker på "Annuler"-knappen. Systemet lukker "Gem"-vinduet ned</li> <li>2.a. Besked om at CPR-nummer ikke er gyldigt, vises. Nyt CPR-nummer indtastet. Use case fortsættes fra punkt 2</li> </ol>

*Tabel 2.8: Fully-dressed use case 6*

## 2.4 Ikke-funktionelle krav

### 2.4.1 (F)URPS+

MoSCoW er angivet i parentes ved hhv. M, S, C og/eller W, for Must, Should, Could og Won't

#### Functionality

1. (M) Brugeren skal kunne starte en ny måling inden for 30 sekunder efter opstart af programmet
2. (M) Systemet skal kunne forstærke signalet fra transduceren 400 gange
3. (M) Systemet skal kunne filtrere signalet med det indbyggede analoge filter med en båndbredde på 50Hz, samt en cut-off frekvens ved 50Hz
4. (M) Programmet skal kunne vise blodtrykssignalet kontinuert
5. (M) Programmet skal programmeres i C#
6. (M) Programmet skal kunne lagre de målte data i en database
7. (M) Systemet skal kalibreres årligt
8. (S) Programmet bør kunne måle puls

## Usability

1. (M) Blodtrykstallene der udskrives på brugergrænsefladen er røde
2. (S) Pulsmålingen bør udskrives på brugergrænsefladen med grønne tal
3. (M) Brugergrænsefladen skal leve op til figuren udarbejdet i designafsnittet, se afsnit 3.3.1

## Reliability

1. (M) Systemet skal kunne køre uden fejl i et år
2. (M) Systemet skal have en ”mean time to restore” på højst 24 timer

Systemet får herved en tilgængelighed beregnet ved

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}} = \frac{365}{365 + 1} = 0,997 = 99,7\% \quad (2.1)$$

## Performance

1. (S) Systemet bør kunne gemme data på 5 sekunder +/-10%

## Supportability

1. (M) Softwaren skal opbygges efter trelagsmodellen

# Design 3

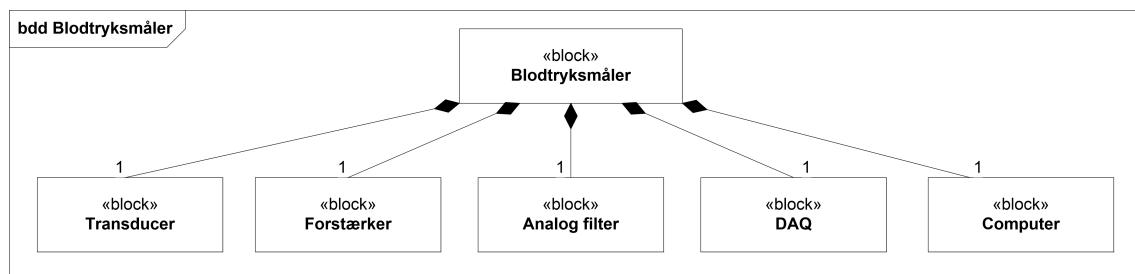
Version	Dato	Ansvarlig	Beskrivelse
0.1	8/09 2015	Alle	Oprettelse af dokument
0.2	27/10 2015	SSK, NHL, FRM	Påbegyndelse af hardware design
0.3	04/11 2015	TSN, JTH, MFJ	Påbegyndelse af software design
0.2	06/11 2015	Alle	Første udkast til færdigt design afsnit
0.3	16/11 2015	Alle	Review rettelser af design afsnittet

## 3.1 Indledning

Formålet med designafsnittet er at beskrive hardwaren og softwaren for blodtryksmålingssystemet. Systemet beskrives ved hjælp af udregninger, diagrammer, figurer og skitser som tydeliggør, hvordan de forskellige delelementers funktionalitet er, samt hvilke tanker der ligger til grund for den endelige implementering (se afsnit 4.1 for hardware og afsnit 4.2 for software).

## 3.2 Hardware arkitektur

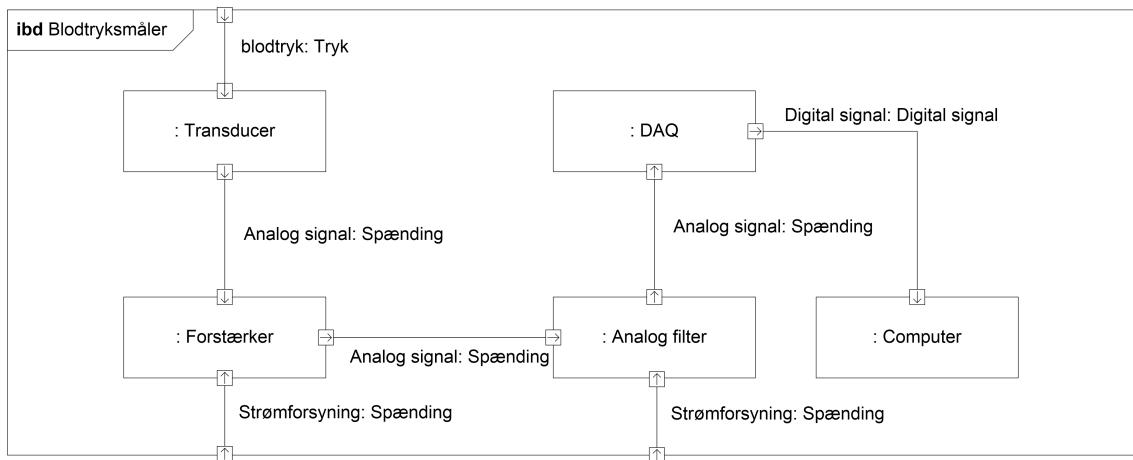
I følgende afsnit beskrives blodtryksmålingssystemet og dets delkomponenters opbygning.



Figur 3.1: Blokdiagram for blodtryksmålingssystemet.

Ud af blokdiagrammet, figur 3.1, kan man se, at blodtryksmålingssystemet består af en

transducer, en forstærker, et analogt filter, en DAQ og en computer.



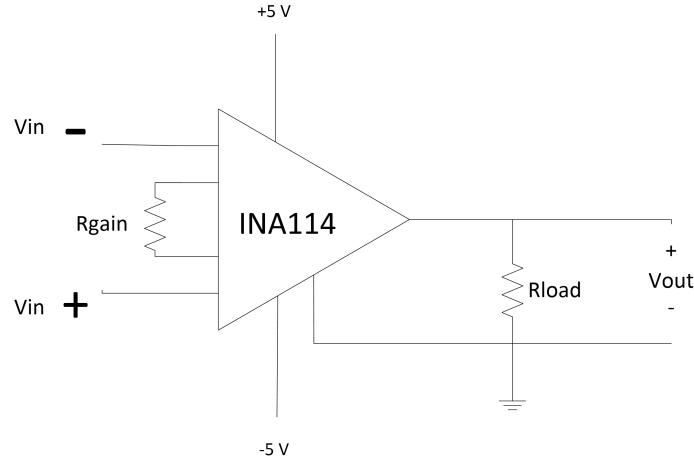
Figur 3.2: Internt blokdiagram for blodtryksmålingssystemet.

Ud af det interne blokdiagram, figur 3.2, kan det ses, at blodtrykket i form af det målte tryk kommer ind i transduceren. Transduceren, som omformer det målte tryk til et spændingssignal, sender signalet videre til forstærkeren. Fra forstærkeren sendes signalet over i det analoge filter og derfra ind i DAQ'en. Endeligt sendes det digitale signal fra DAQ'en over i en computer, der fortolker signalet som et billede, der vises til omverdenen.

### 3.2.1 Design af forstærker

Forstærkeren er designet med tanke på, at det er meget små spændinger, der arbejdes med. Grundet dette, er en almindelig operationsforstærker fravalgt, da dens reelle indgangsimpedans er for lav. En instrumentationsforstærkers indgangsimpedans i den virkelige verden er højere, og den kan dermed opfange meget små signaler som f.eks. blodtryk, der opererer i mV.

Vejleder rådede herefter til, at der skulle bruges instrumentationsforstærkeren INA114. Forstærkerens design er valgt ud fra instrumentationsforstærkerens datablads anbefalinger og kan ses på figur 3.3.



Figur 3.3: Det overordnede design af forstærkeren.

$R_{gain}$  er modstanden, som bestemmer hvor meget forstærkning instrumentationsforstærkeren skal give og  $R_{load}$  er den belastning, der kommer efter kredsløbet. I dette tilfælde er belastningen det analoge filter. For at finde  $R_{gain}$ 's størrelse, kræves viden om, hvor meget forstærkning, der er brug for. Dette findes ved at bestemme den maksimale spænding, som transduceren kan give i en blodtrykssituation. Dette regnestykke kan ses realiseret i ligning 3.1:

$$VT_{max} = T_{max} \cdot V_{max} \cdot Hg_{max} = 5 \frac{\mu V}{V/mmHg} \cdot 5V \cdot 250mmHg = 6,25mV \quad (3.1)$$

Spændingen ønskes skaleret op til DAQ'ens dynamikområde, som ligger omkring  $+/- 2,5V$ . Forstærkningsfaktoren udregnes ved simpel brøkregning, som ses på ligning 3.2:

$$G = \frac{2,5}{6,25 \cdot 10^{-3}} = 400 \quad (3.2)$$

INA114's datablad giver en ligning for udregning af forstærkning. Da forstærkningen er kendt, omskrives ligning 3.3, så modstanden  $R_{gain}$ 's værdi i stedet bestemmes:

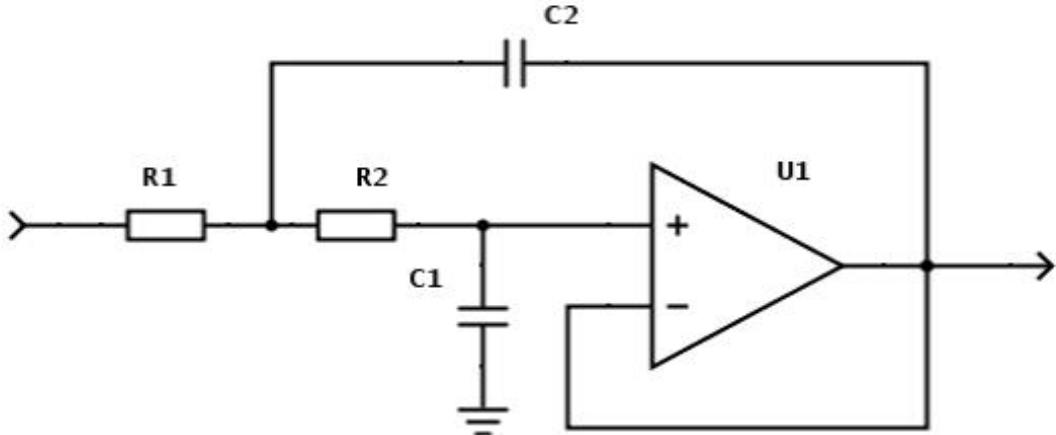
$$Gain = 1 + \frac{50k\Omega}{R_{gain}} \rightarrow G - 1 = \frac{50k\Omega}{R_{gain}} \rightarrow \frac{50k\Omega}{G - 1} = R_{gain} \quad (3.3)$$

Herefter kan den ohmske værdi af  $R_{gain}$  bestemmes, hvilket sker i ligning 3.4:

$$R_{gain} = \frac{50k\Omega}{400 - 1} = 125,31\Omega \quad (3.4)$$

### 3.2.2 Design af analogfilter

Filteret, som ses på figur 3.4, skulle realiseres som et aktivt 2. ordens lavpasfilter af typen Sallen-Key med unity gain, hvor båndbredden er på 50 Hz (se 3.4). Desuden skulle filteret yderligere designes som et Butterworth-filter med cutoff-frekvens på 50 Hz. C2 skulle vælges til at være 680 nF, det blev oplyst, at R1 skulle være lig med R2. Operationsforstærkeren blev opgivet til at være af typen OP27.



Figur 3.4: Unity gain 2. ordens Sallen-Key lavpas konfiguration

Et Butterworth-filter har en dæmpningsfaktor på 0,7, fordi der, i et Butterworth-filter, er blevet prioriteret et fladt frekvensområde, frem for hurtig dæmpning. Der blev brugt en hjemmeside som hjælpemiddel til at finde overføringsfunktionen for Butterworth-lavpasfilteret.[1] Denne ligning kan ses i ligning 3.5:

$$\frac{V_{out}(S)}{V_{in}(S)} = \frac{\frac{1}{R_1 C_1 R_2 C_2}}{s^2 + s(\frac{1}{R_2 C_2} + \frac{1}{R_1 C_2}) + \frac{1}{R_1 C_1 R_2 C_2}} \quad (3.5)$$

Da det er blevet opgivet, at  $R_1 = R_2$  kan overføringsfunktionen forkortes, som set på ligning 3.6.

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{\frac{1}{R^2 C_1 C_2}}{s^2 + s(\frac{2}{RC_2}) + \frac{1}{R^2 C_1 C_2}} \quad (3.6)$$

Dernæst sammenlignes med standardformlen for overføringsfunktionen for et andet ordens filter på figur 3.7:

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{\frac{1}{R^2 C_1 C_2}}{s^2 + s(\frac{2}{RC_2}) + \frac{1}{R^2 C_1 C_2}} = \frac{\omega_0^2}{s^2 + s(2\zeta\omega_0) + \omega_0^2} \quad (3.7)$$

Ud fra dette kan komponentværdierne regnes for R, idet vi har en opgivet værdi for  $\frac{2}{RC_2}$ , som vist på figur 3.8:

$$\frac{2}{RC_2} = 2\zeta\omega_0 \quad (3.8)$$

Der blev herefter brugt Mathcad til at isolere R, og udregne værdien af denne. Dette kan ses på figur 3.9:

$$\frac{2}{R \cdot 680 \cdot 10^{-9}} = 2 \cdot 0.7 \cdot (50 \cdot 2 \cdot \pi) \text{ solve}, R \rightarrow \frac{250000}{17 \cdot \pi} = 6.687 \times 10^3 \quad (3.9)$$

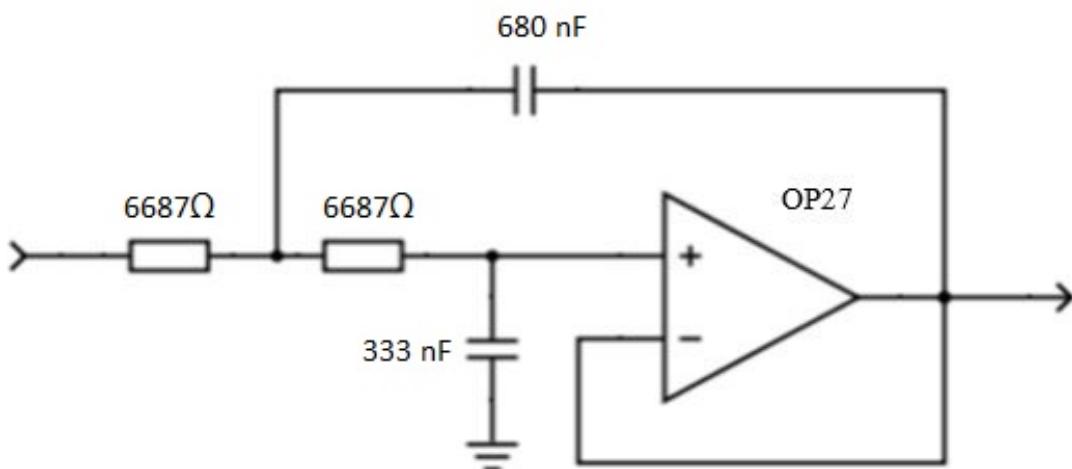
Dernæst kan komponentværdien for C1 udregnes:

$$\frac{1}{R^2 C_1 C_2} = \omega_0^2$$

Ved hjælp af Mathcad isoleres C1.

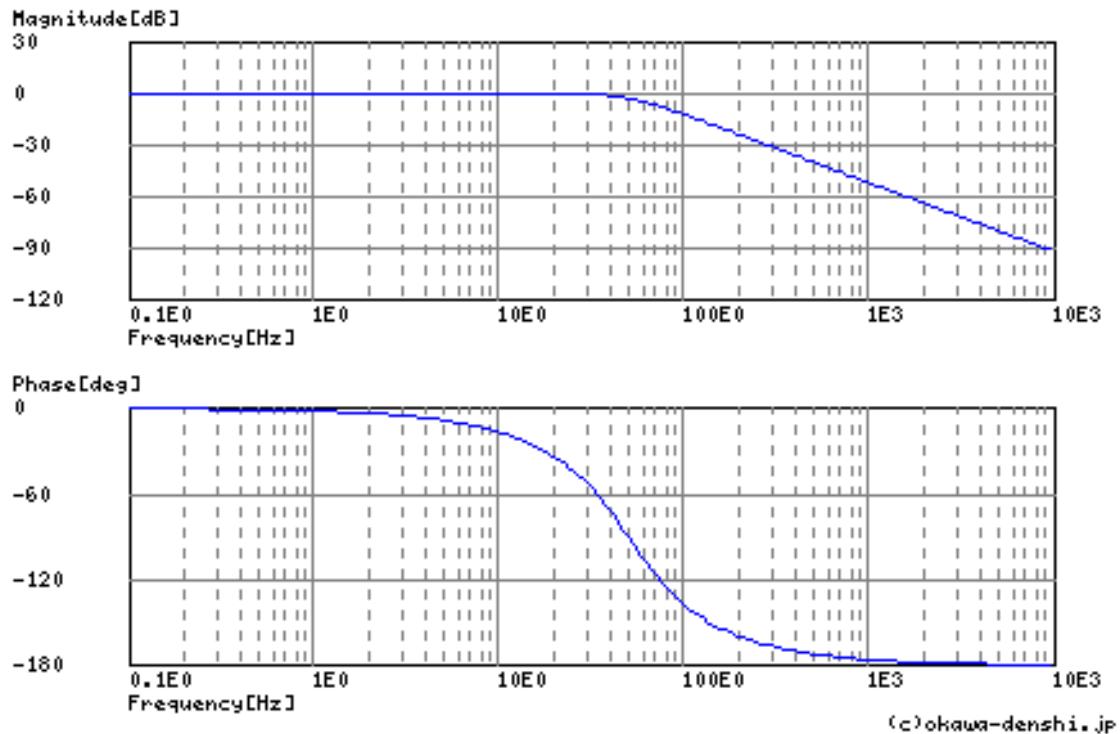
$$\frac{1}{(6.687 \times 10^3)^2 \cdot (680 \cdot 10^{-9}) \cdot C_1} = (50 \cdot 2\pi)^2 \text{ solve}, R \rightarrow 333, 2 \times 10^{-9} \quad (3.10)$$

Derved er komponentværdierne for kredsløbet fundet og kan ses indskrevet på 3.4.



Figur 3.5: Unity gain 2. ordens Sallen-Key lavpas konfiguration med indsatte komponentværdier.

For at underbygge teorien omkring filteret, blev der ved hjælp af værktøjet "Sallen-Key Low-pass Filter Design Tool" udarbejdet et bodeplot [1]. Dette kan ses nedefor på figur 3.6.



Figur 3.6: Bodeplot af overføringsfunktionen

### 3.2.3 Grænseflader

Grænsefladerne for hardwareblokkene er beskrevet i tabel 3.2:

Navn	Input	Output	Interval	Beskrivelse
Transducer	Tryk	Spænding	Ind; -50 mmHg Ud; 0 til 6,25 mV	Transduceren, i form af en straingauge, reagerer i forhold til trykændringer, og udsender en spænding, som ændrer sig i forhold til tryk.
Forstærker	Spænding	Spænding	Ind; 0 til 6,25 mV Ud; -2,5V til 2,5V	Forstærkeren modtager det svage signal fra transduceren, og forstærker signalet op, så det matcher DAQ'ens dynamikområde.
Filter	Spænding	Analogt signal	Ind; -2,5V til 2,5V Ud; -2,5V til 2,5V	Filteret modtager det forstærkede signal fra forstærkeren og filtrerer signalet.
DAQ	Analogt signal	Digitalt signal	Ind; -2,5V til 2,5V Ud; -2,5V til 2,5V	DAQ'en konverterer det analoge signal fra filteret til et digitalt signal, som computeren modtager
Computer	Digitalt signal	Grafisk billede		Computeren modtager et digitalt signal fra DAQ'en, som bliver behandlet i koden.

Tabel 3.2: Grænsefladetabel

### 3.3 Software arkitektur

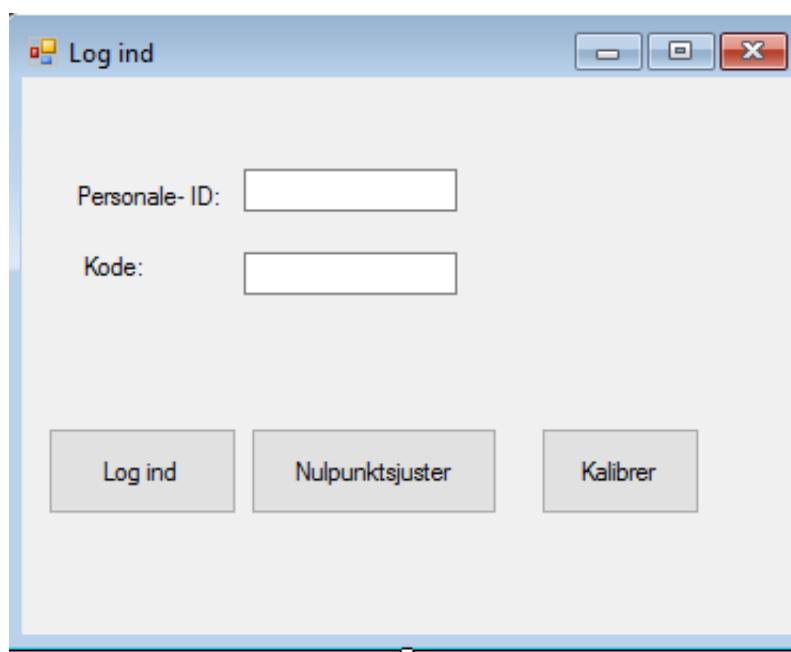
#### 3.3.1 GUI

Dette afsnit beskriver, hvilke tanker og overvejelser der er gjort i forbindelse med designet af diverse brugergrænseflader. Nedenfor ses udkast til disse brugergrænseflader. Den endelige udgave af brugergrænseflader kan ses i afsnit 4.2.

Designet af brugergrænsefladen er blevet udført ud fra de 16 principper for gode brugergrænseflader [2]. Der er ligeledes i "Form1" hentet inspiration fra allerede eksisterende blodtryksmonitor. I designet af brugergrænsefladen bestræbes der efter at gøre det så virkelighedsnært som muligt ud fra de redskaber og oplysninger, der er til rådighed.

Brugergrænsefladen er inddelt i tre forskellige vinduer. De tre vinduer er henholdsvis "Log ind", "Form1" og "Gem": De personer, der interagerer med brugergrænsefladen er sundhedsfagligt personale som typisk vil være en læge eller sygeplejerske. Dette er der også taget højde for i designet, da der aldrig vil være andre end det sundhedsfaglige personale, altså en skærpel målgruppe, der interagerer med brugergrænsefladen.

Nedenfor kommer der en uddybende beskrivelse af brugergrænsefladen.



Figur 3.7: Login vindue

I figur 3.7 er der taget udgangspunkt i, at brugergrænsefladen skal være enkel og overskuelig opbygget. Der skal ikke være nogle overflødige ting og de knapper, der er vigtige som bruger skal benytte, skal stå tydeligt frem. Opbygningen skal afspejle brugerens logik.

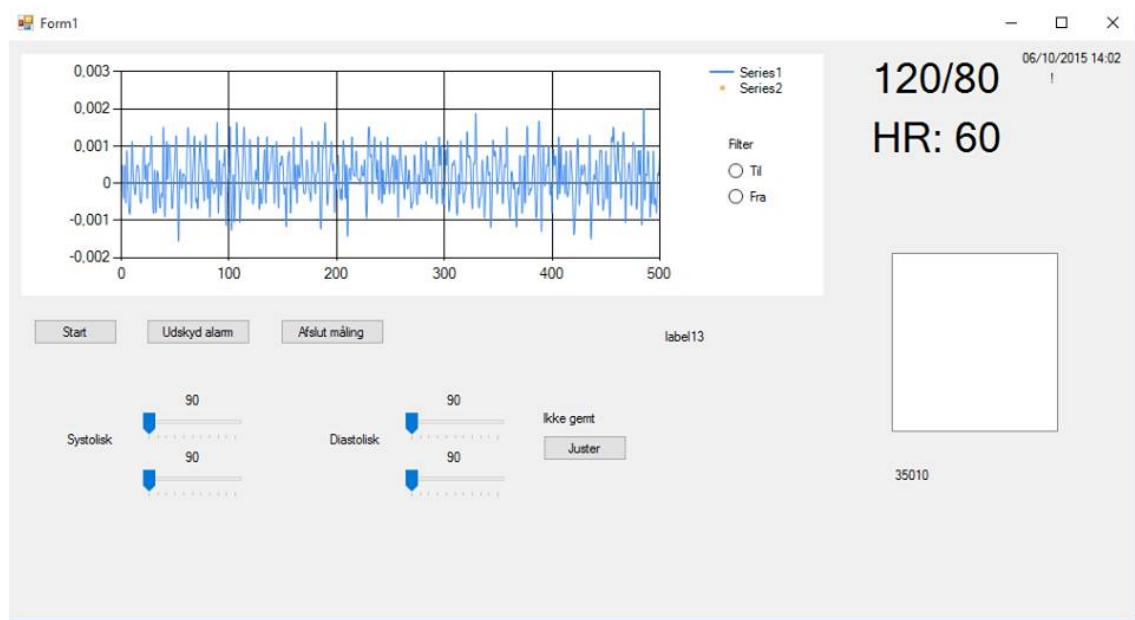
Tankerne omkring den enkle opbygning er, at hvis der opstår en akut situation, hvor systemet skal i gang hurtigt, skal det være nemt og hurtigt at logge ind i systemet. Det er vigtigt, at knapperne er selvforklarende og giver brugeren det, der forventes af knappen, når den benyttes.

Der er taget højde for feedback til bruger. Ved forkert login får brugeren en besked om, at login er indtastet forkert, og der skal prøves forfra.



Figur 3.8: Kalibrer vindue

Vist på figur 3.8, hvor der her er lagt vægt på det funktionelle med et enkelt og simpelt design. Der er vejledende tekst, som gør det nemt for aktøren at finde ud af systemet.



Figur 3.9: Blodtryksvindue

I blodtryksvinduet, vist på figur 3.9, er opbygningen mere kompliceret og med flere knapper end ved "Login" og "Gem": Der er taget højde for, at knapperne er selvforklarende og deres navne afspejler de bagvedliggende handlinger. Dette medfører, at brugeren stadig har et

overblik over brugergrænsefalden og stadig selv kan kontrollere hvad der skal ske. Denne brugergrænseflade er ikke tiltænkt nye brugere, men derimod brugere, der har et kendskab til den og til hvilke sundhedfaglige værdier og udtryk, der vises i vinduet. Derfor benyttes brugerens sprog på brugergrænsefalden, altså med fagterminer.

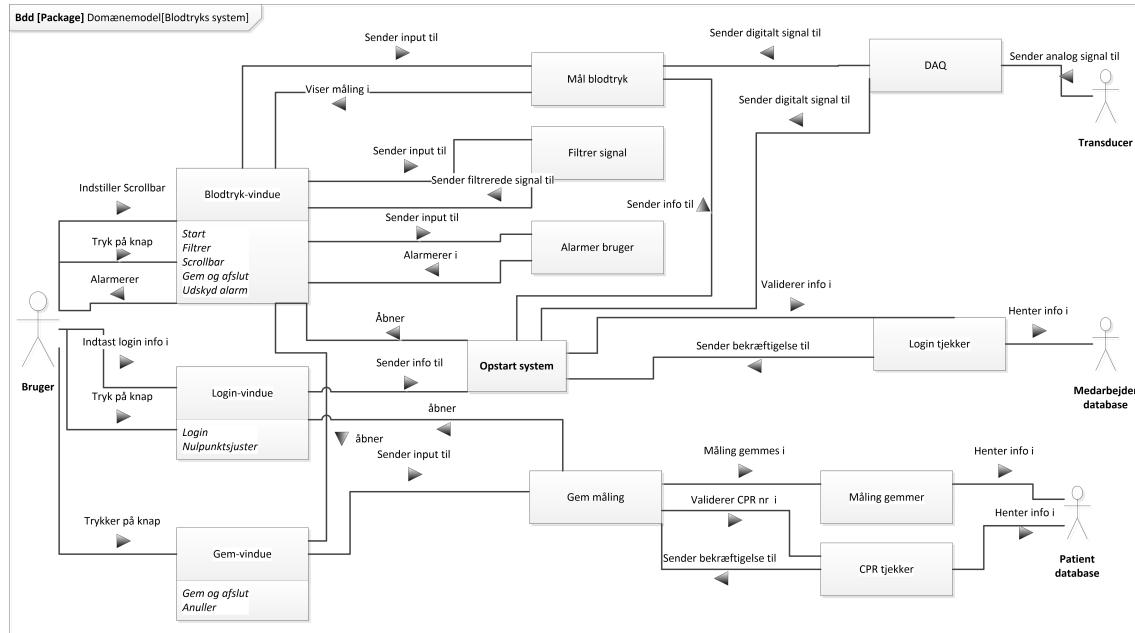


Figur 3.10: "Gem" -vindue

På figur 3.10, "Gem"-vinduet, er der igen taget udgangspunkt i en enkel opbygning og at knapperne, der skal bruges er tydelige og selvforklarende. Det er her også brugerens logik, der afspejles og ingen unødvendige ting er inddraget i brugergrænsefladen. Det er her muligt at afbryde situationen ved, at der er en alternativ udvej, som kan bruges, hvis den opstartede handling fortrydes. Der er taget højde for feedback til brugeren, hvis der indtastes forkert CPR-nummer, får brugeren det at vide i et pop op-vindue.

### 3.3.2 Domænemodel

Diagrammet viser en domænemodel for blodtryksmålingssystemet. Dette diagram giver et godt overblik over systemet som helhed og hvilke elementer, der indgår i systemet. Diagrammet viser, hvordan brugeren interagerer med systemet, og hvordan systemet forløber, efter det igangsættes af bruger.



Figur 3.11: Domænemodel

### 3.3.3 Appliktionsmodel

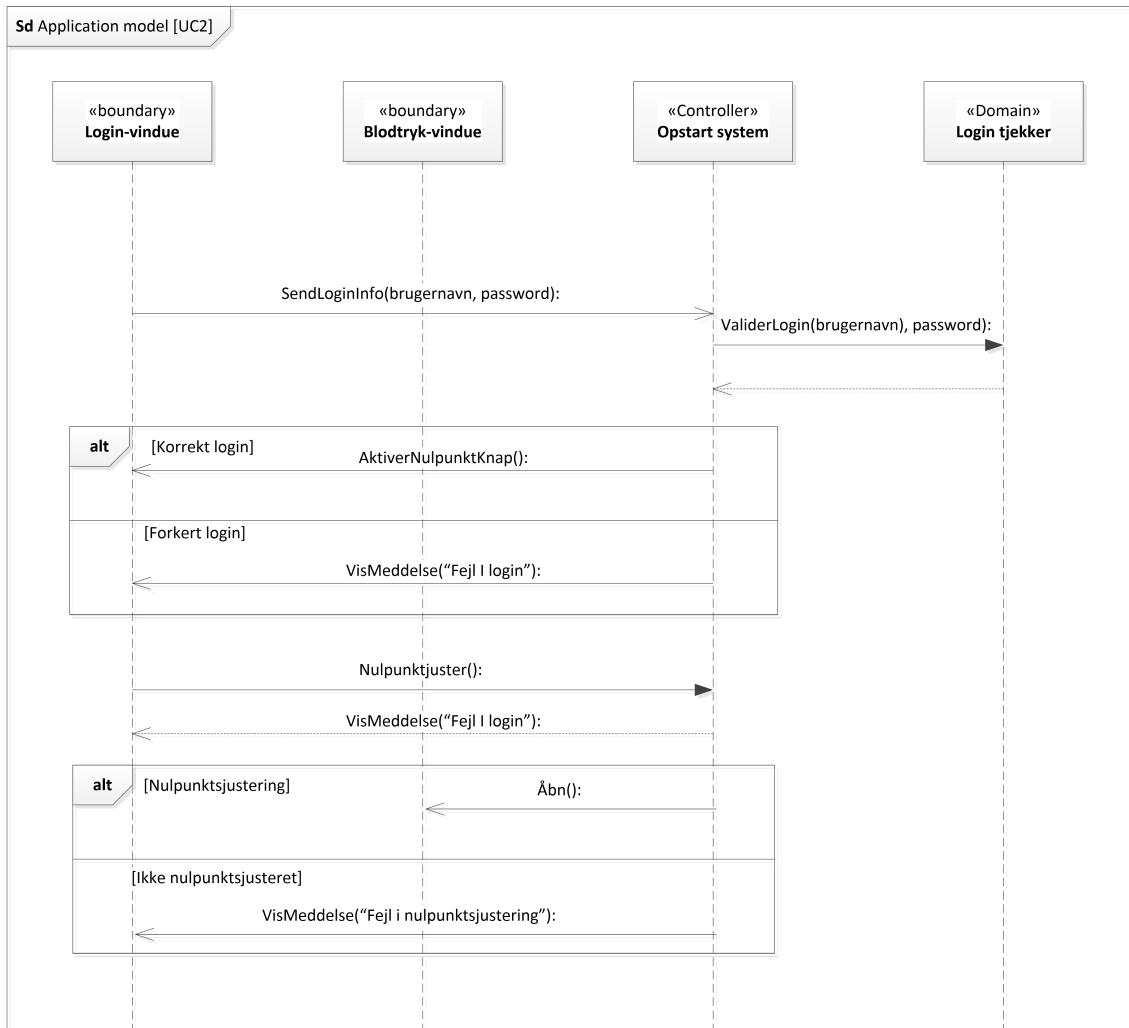
Appliktionsmodellen er en model der på baggrund af domænemodel og use cases udarbejdes software-relaterede klasse-appliktionsmodeller, sekvensdiagrammer og opdaterede klasse-appliktionsmodeller.

Her er det ikke-opdateret klassediagram udeladt, hvorfor sekvensdiagram og opdaterede klasse-appliktionsmodel er udarbejdet på baggrund af fiktive metoder. De faktiske metoder kan ses i afsnit 4.2.4.

### Sekvensdiagram

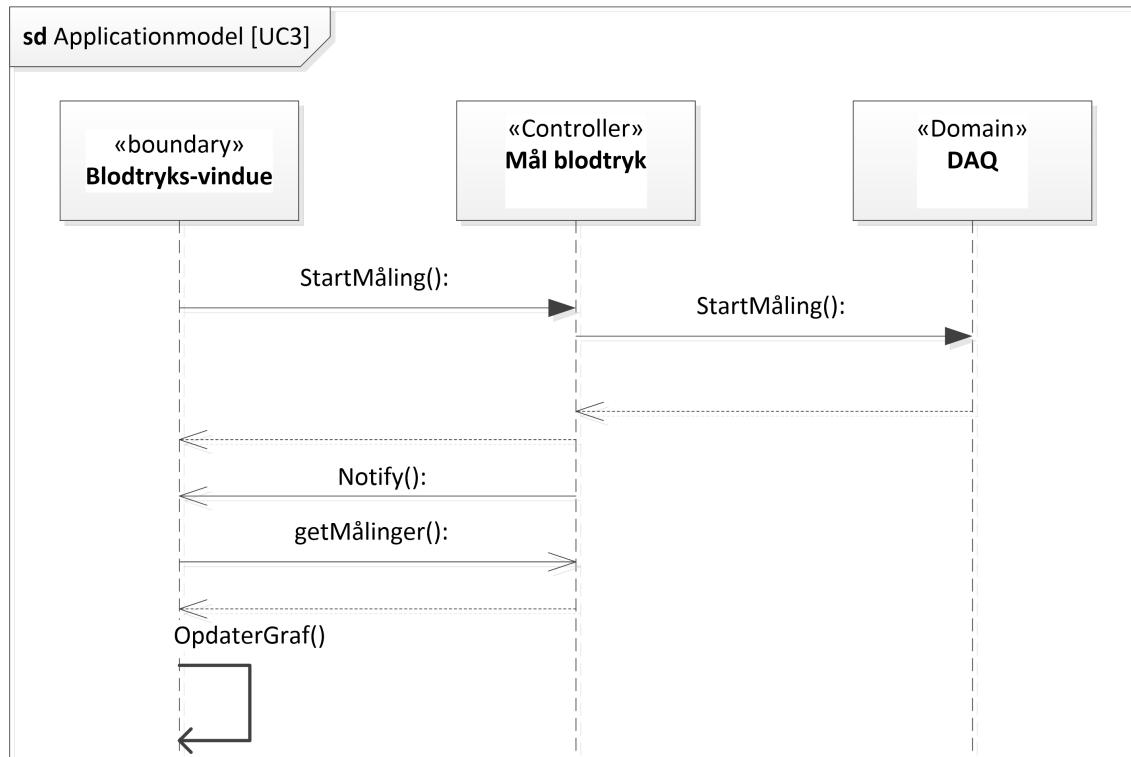
Sekvensdiagrammet er et interaktionsdiagram, der viser, hvordan processer i systemet forløber. Use casene ligger til baggrund for udarbejdelsen af sekvensdiagrammerne.

Der er blevet lavet et sekvensdiagram for hver use case for at give det bedste overblik over hver handling i forhold til systemet. I sekvensdiagrammet har vi brugt virtuelle metodekald til at beskrive forløbet. I use case 2-7 er det brugeren, der interagerer med systemet og er initiatør for, at handlingerne bliver udført.



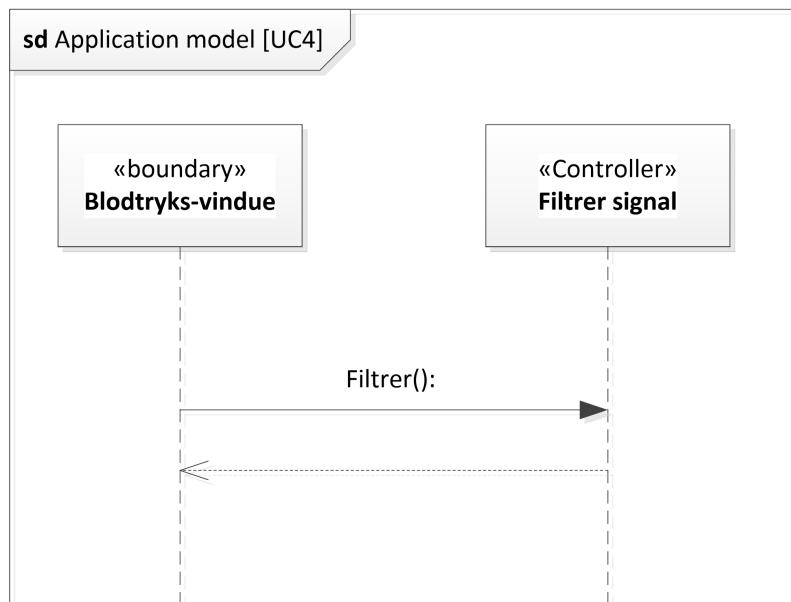
Figur 3.12: Sekvensdiagram UC2

På figur 3.12 ses det, hvordan brugeren logger ind i systemet, og hvordan brugeren får systemet nulpunktsjusteret. Det er brugeren, der interagerer med systemet og er initiatør for, at handlingerne bliver udført.



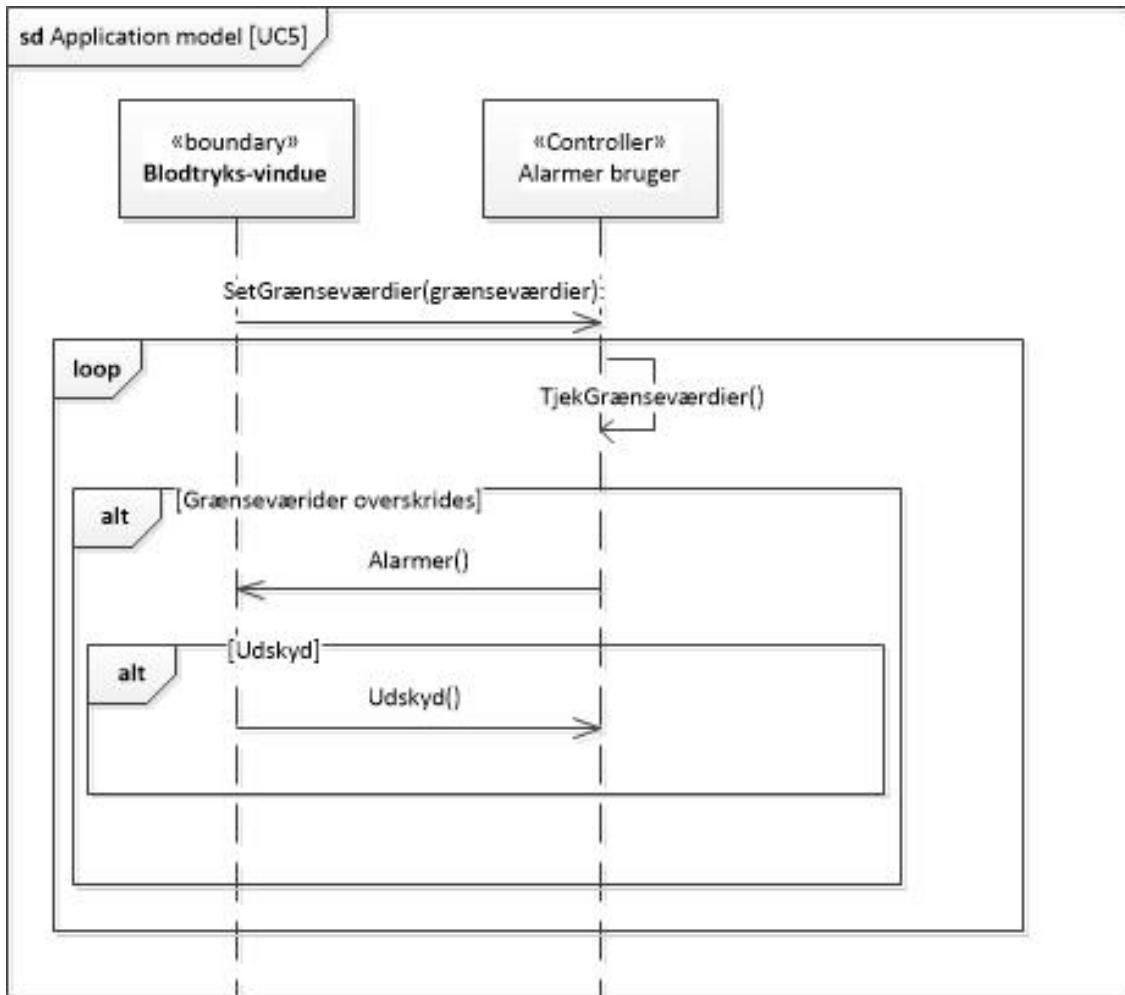
Figur 3.13: Sekvensdiagram UC3

Figur 3.13 viser, hvordan brugeren starter målingen, og hvordan graferne vises på brugergrænsefladen.



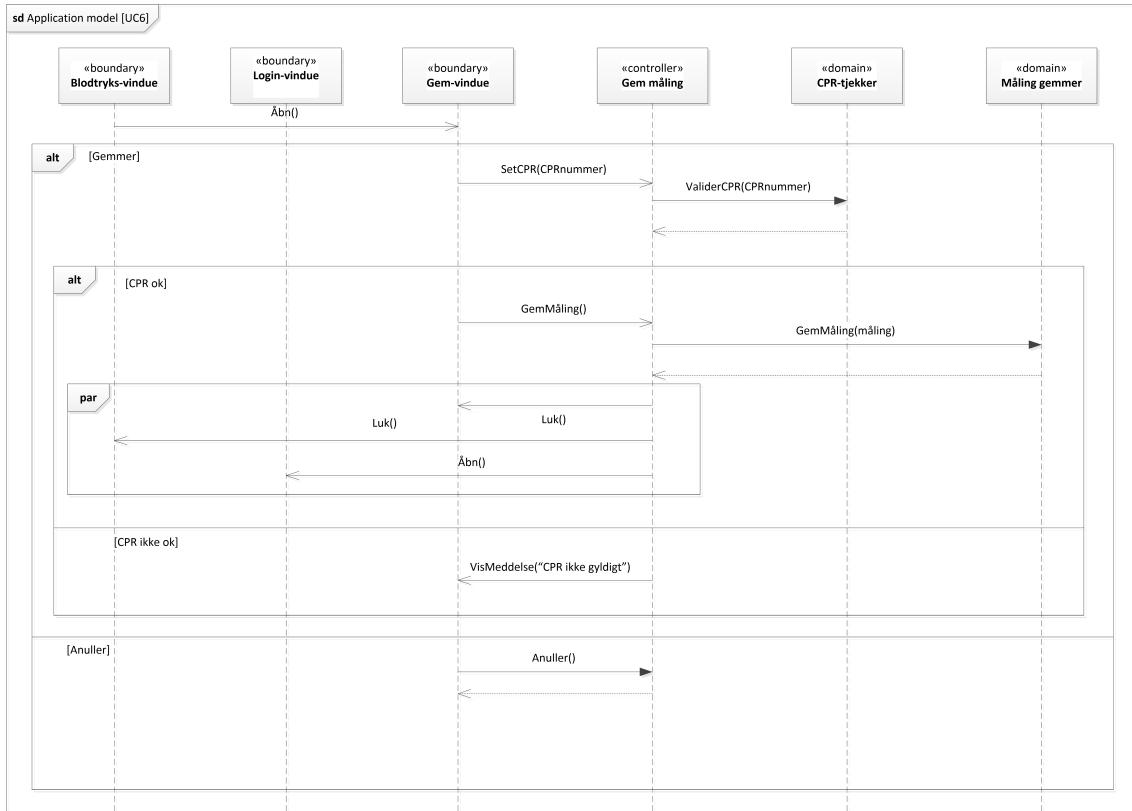
Figur 3.14: Sekvensdiagram UC4

I sekvensdiagrammet figur 3.14 ses, hvordan brugeren kan vælge om blodtrykssignalet skal filtreres eller ej.



Figur 3.15: Sekvensdiagram UC5

Det ses på figur 3.15, hvordan brugeren justerer grænseværdierne for patientens blodtryk. Denne justering sker ud fra patientens målte blodtryk. De justerede grænseværdier ligger til grundlag for alarmen. Desuden viser figuren, at brugeren har mulighed for at udskyde alarmen.

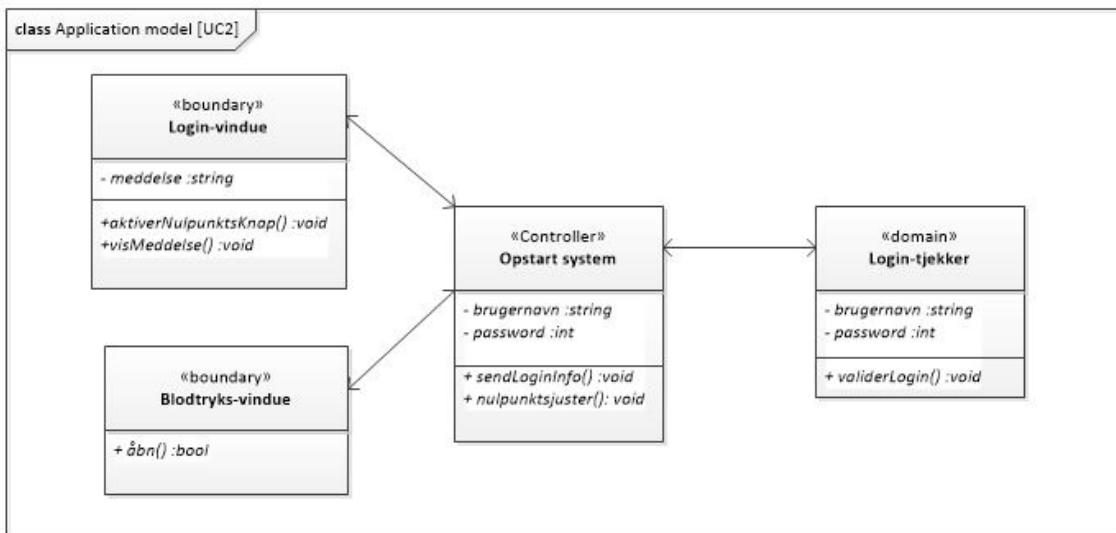


Figur 3.16: Sekvensdiagram UC6

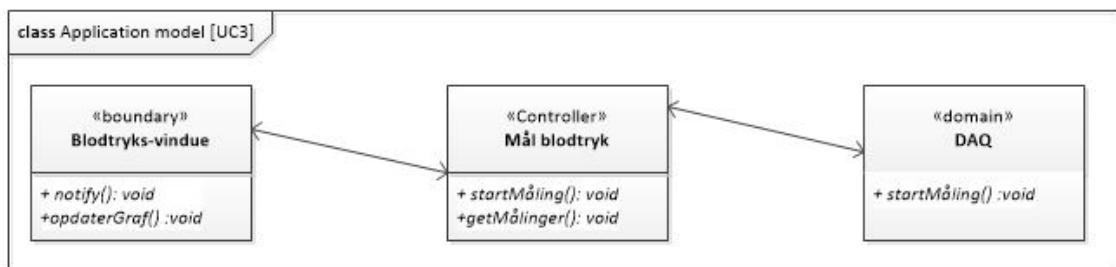
Figur 3.16 viser, hvordan systemet gemmer og afslutter en måling.

### Opdateret applikationsmodel

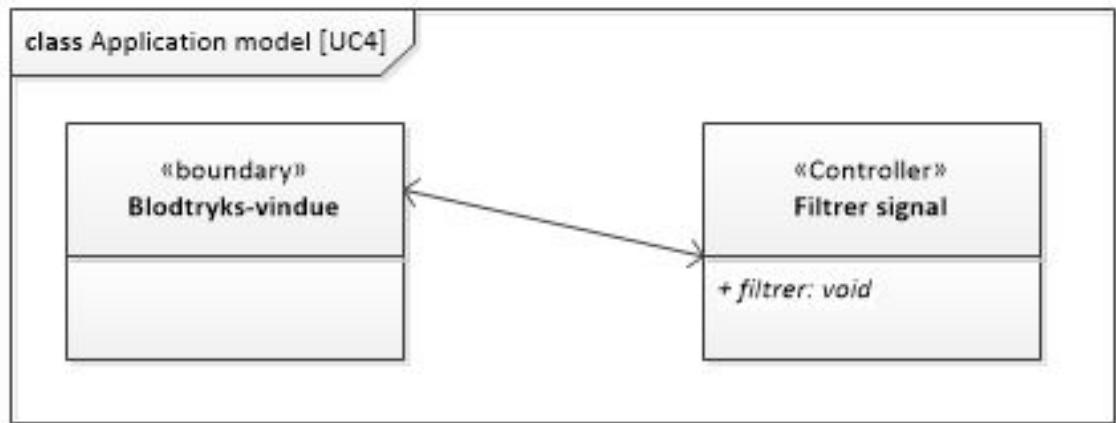
Klasse-applikationsmodellerne er udarbejdet ud fra use casene, sekvensdiagrammer og domænemodellen. Der er ligesom ved sekvensdiagrammerne lavet et klassediagram for hver use case. Der er også her brugt virtuelle metoder.



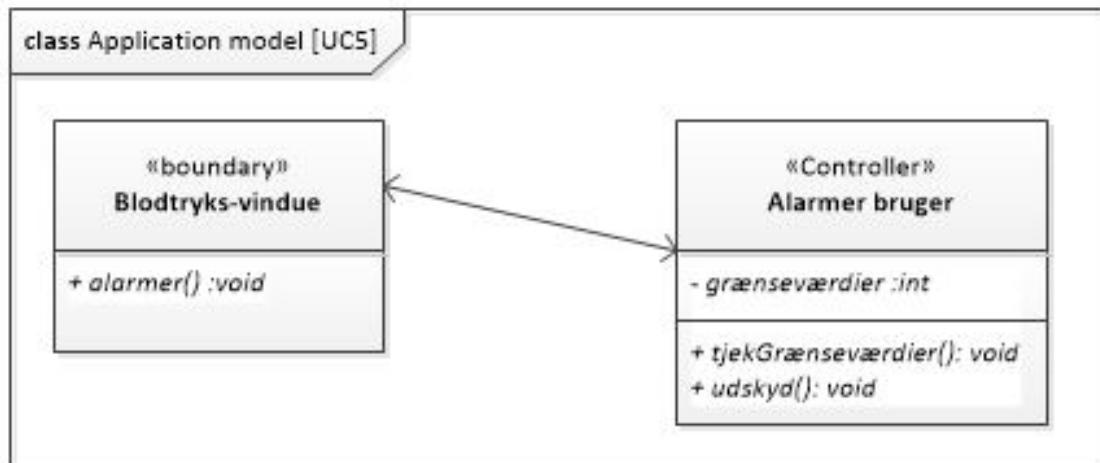
Figur 3.17: Klassediagram UC 2



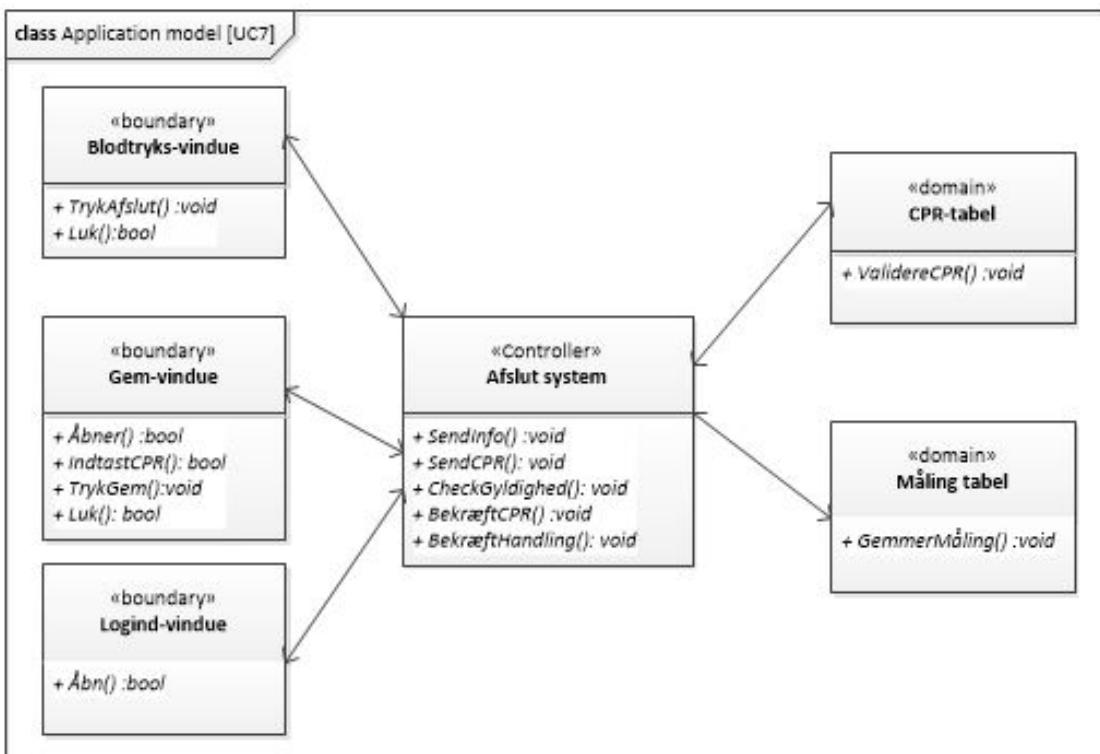
Figur 3.18: Klassediagram UC 3



Figur 3.19: Klassediagram UC 4



Figur 3.20: Klassediagram UC 5



Figur 3.21: Klassediagram UC 7

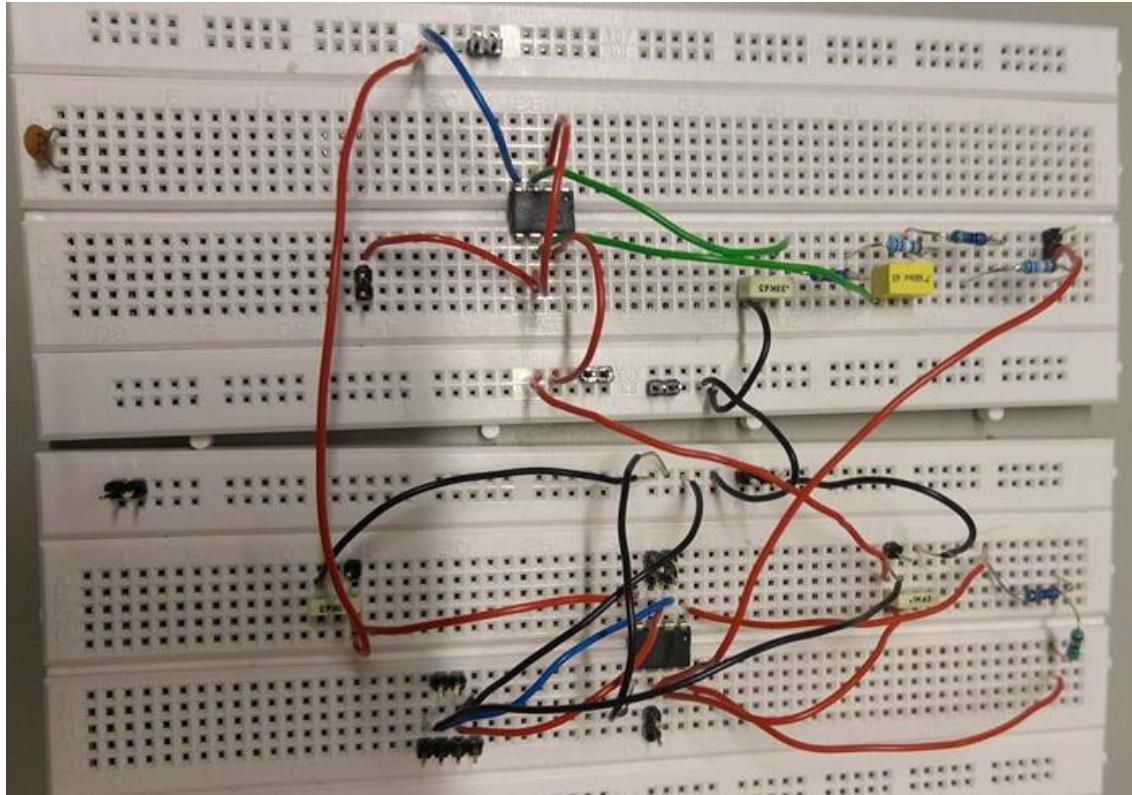
# Implementering 4

---

Version	Dato	Ansvarlig	Beskrivelse
0.1	8/09 2015	Alle	Oprettelse af dokument
0.2	07/12 2015	SSK, NHL, FRM	Påbegyndelse af hardware implementeringsafsnit
0.3	10/12 2015	TSN, JTH, MFJ	Påbegyndelse af software implementeringsafsnit

## 4.1 Hardware implementering

Efter teorifasen blev de to blokke bygget op. Forstærkeren og filteret blev bygget på hver sit fumlebræt, dels på grund af pladsmangel og dels på grund af større sammenhæng mellem arkitekturen og det endelige produkt.



Figur 4.1: Opstilling af forstærker og filter

#### 4.1.1 Hardware overvejelser

I udviklingen af hardwaren, opstod der problemstillinger, som gjorde, at der var nogle komponentvalg, som var nødvendige at genoverveje.

Et af de største problemer var brugen af operationsforstærker i forstærkerblokken. En reel operationsforstærker har ikke en uendelig indgangsmodstand, som i den ideelle verden, og der er derfor en risiko for, at den lave spænding fra transduceren simpelthen ikke ville kunne passere forstærkeren. Det blev derfor valgt at bruge en instrumentationsforstærker i stedet, da den reelle komponents egenskaber ligger langt tættere på dens ideelle modpart, og derfor var bedre egnet i kredsløbet. En anden fordel ved instrumentationsforstærkeren er, at den er meget let at justere forstærkning på.

En anden stor fordel ved instrumentationsforstærkeren er, at common mode rejection ratio er meget høj. Common mode noise er støj, fra det omkringliggende miljø. Da en instrumentationsforstærkers forstærkning kommer fra forskellen mellem dens to indgange, kan støj påvirke forstærkningen, især når der arbejdes med så lave forstærkninger, som dem, der kommer fra transduceren. Ofte er der dog støj på begge indgange, og i den ideelle verden ville dette ikke påvirke forstærkningen. I den reelle verden kan støjen dog også ende med at blive forstærket. Common mode rejection ratio beskriver hvor god en komponent er til at sortere støj fra. Den brugte instrumentationsforstærker har en common mode rejection ratio på 120dB, hvilket vurderes til en fornuftig værdi.

En anden ting ved forstærkeren, som blev nødt til at blive ændret, var det dynamik område, som skulle udnyttes. Oprindeligt havde gruppen bestemt, at det filtrerede signal skulle ligge imellem  $-5V$  og  $+5V$ . Grundet en forstærkers manglende evne til at forstærke et signal op til dens forsyningsspænding, så blev det vurderet, at det var nødvendigt at ned sætte signalområdet fra  $-2,5V$  til  $+2,5V$ . Det var muligt at forstærke signalet yderligere, men grænsen blev sat til denne værdi, fordi signalområdet matcher et dynamikområde på DAQ'en.

Designet af filteret, lå fast på forhånd, og der var derfor ikke meget der kunne ændres i dette. Operationsforstærkeren var oplyst til at skulle være af typen OP27, da denne ligeledes har en god common mode rejection. Grundet mangel på præcise komponenter, blev der sat to modstande i serie, for at komme så tæt på den udregnede værdi, som muligt.

Det blev desuden valgt at systemet skulle forsynes af Analog Discovery, frem for et batteri. Dette skyldes at Analog Discovery giver en stabil strøm, som ikke har brug for at blive afbalanceret af en spændingsudligner. Det blev vurderet at usikkerhederne ville udligne hinanden, og der var flere fordele ved at bruge Analog Discovery.

### Implementering af forstærkeren

Den samlede stykliste for forstærkeren er vist i tabel 4.2:

Komponent	Antal	Type
Modstand	1	$120\Omega$
Modstand	1	$4.8\Omega$
Kondensator	2	$100nF$
Instrumentationsforstærker	1	INA114

Tabel 4.2: Forstærkertabel

For forstærkeren gælder det, at den beregnede  $R_{gain}$  er  $125,31\Omega$  som det kan ses ud af komponentlisten består  $R_{gain}$  i praksis af to modstande på henholdsvis  $4,8\Omega$  og  $120\Omega$ , som er sat i serie.  $R_{gain}$  modstanden er i praksis  $124,8\Omega$ . I praksis er  $R_{gain}$   $0,51\Omega$  mindre end den i teorien skulle have været.

### Implementering af filteret

Den samlede stykliste for filteret er som vist på tabel 4.3:

Komponent	Antal	Type
Modstand	2	6.2kΩ
Modstand	2	470Ω
Kondensator	1	680nF
Kondensator	1	330nF
Operationsforstærker	1	OP27G

Tabel 4.3: Filtertabel

Det analoge filter består blandt andet af en 330nF kondensator,  $C_1$ , som i teorien er beregnet til at skulle have været 333,2nF. I praksis er kondensatoren  $C_1$  3,2 nF mindre, end den i teorien er beregnet til at skulle have været. Filteret består desuden af to modstande  $R_1$  og  $R_2$ , som er identiske. I det realiserede analoge filter består hver modstand af to modstande på henholdsvis 6200Ω og 470Ω, som er sat i serieforbindelse. Dermed er både  $R_1$  og  $R_2$  6670Ω i praksis. I teorien var  $R_1$  og  $R_2$  udregnet til at skulle være 6687Ω. I praksis er modstanden derfor 17Ω mindre end teorien foreskriver.

Generelt er der valgt at se bort fra de afvigelser, der er for komponentværdierne i praksis sammenlignet med de i teorien beregnet. Det er valgt da afvigelserne er relativt små i forhold til det pågældende komponent. For modstandene er der desuden 1% usikkerhed, hvilket betyder, at man alligevel ikke kan være helt sikker på komponentværdien.

På baggrund af de i praksis anvendte komponenter er den reelle knækfrekvens for det analoge filter beregnet. Til det formål er formlen som set på ligning 4.1 anvendt.

$$f_c = \frac{1}{2\pi\sqrt{R_1C_1R_2C_2}} = \frac{1}{2\pi\sqrt{6687 \cdot 333,2 \times 10^{-9} \cdot 6687 \cdot 680 \times 10^{-9}}} = 50,37Hz \quad (4.1)$$

Desuden er den reelle  $\zeta$  for dette filter, ifølge Okawa-Denshi, 0,697.

## 4.2 Software implementering

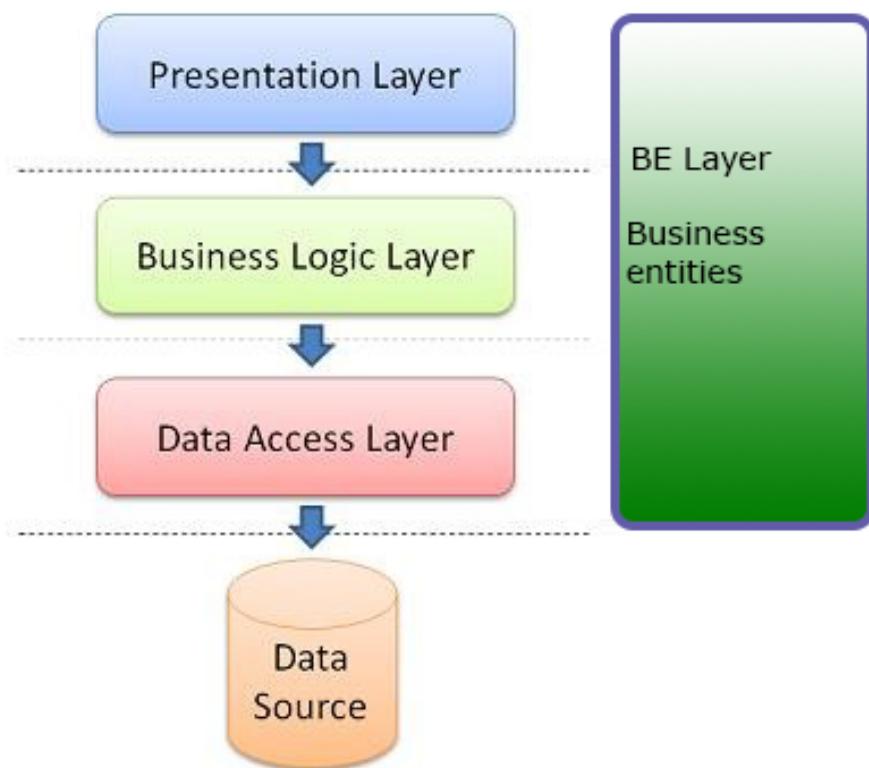
Implementeringsafsnittet beskriver, hvilke faktiske handlinger der er foretaget ud fra arkitektur- og designafsnittet. Yderligere beskriver afsnittet, hvilke softwareorienterede metoder, der er benyttet, for at løse problemstillingen.

### 4.2.1 3-lagsmodellen

Softwareen er først og fremmest implementeret ud fra 3-lagsmodellen. 3-lagsmodellen er bestående af lagene præsentationslag, logiklag og datalag. Denne opdeling af lagene gør det langt lettere at vedligeholde systemet, fordi der kan ændres

i et enkelt lag, uden det har indflydelse på resten af programmet.

3-lagsmodellen er desuden en god softwarearkitektur at bruge til systemudarbejdelse, når der arbejdes i grupper. Dette skyldes, at der kan arbejdes på to forskellige lag af to personer samtidigt, hvis bare grænsefladerne bliver overholdt. 3-lagsmodellen er bygget op som vist på figur 4.2:

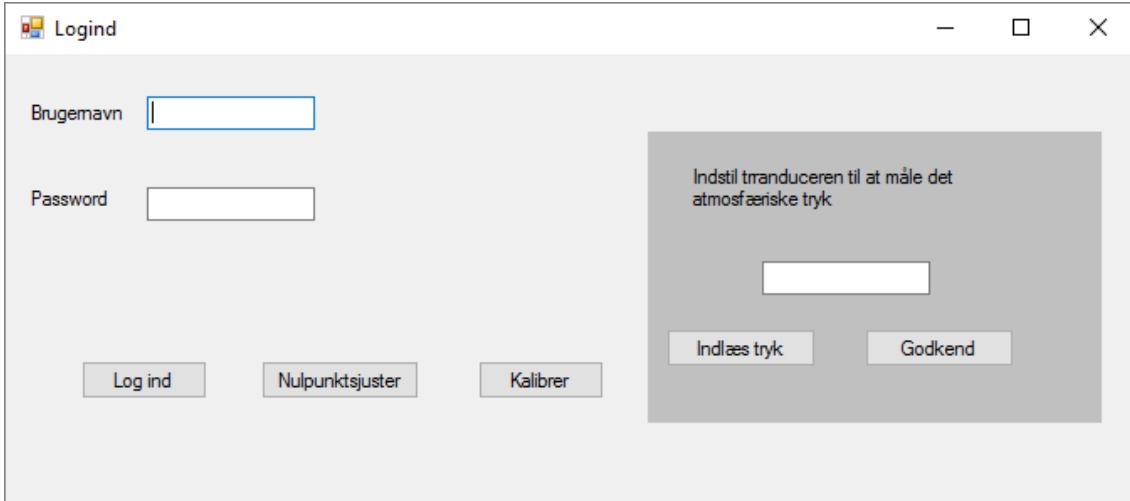


Figur 4.2: 3-lagsmodellen

### Præsentationslag

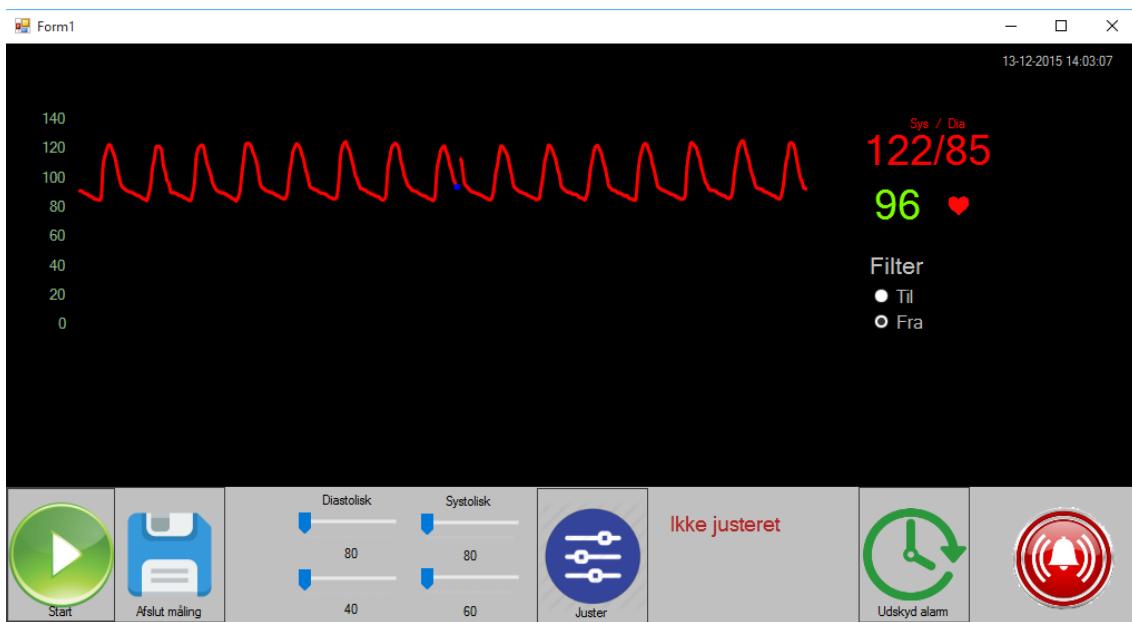
Som vist på figur 4.2, er blodtrykssystemet bygget op med først et præsentationslag bestående af tre vinduer, som brugeren kan interagere med.

"Login"-vinduet, bestående af tekstbokse til brugernavn og kodeord, samt paneler til både kalibrering og nulpunktsjusteringen, som vist på figur 4.3:



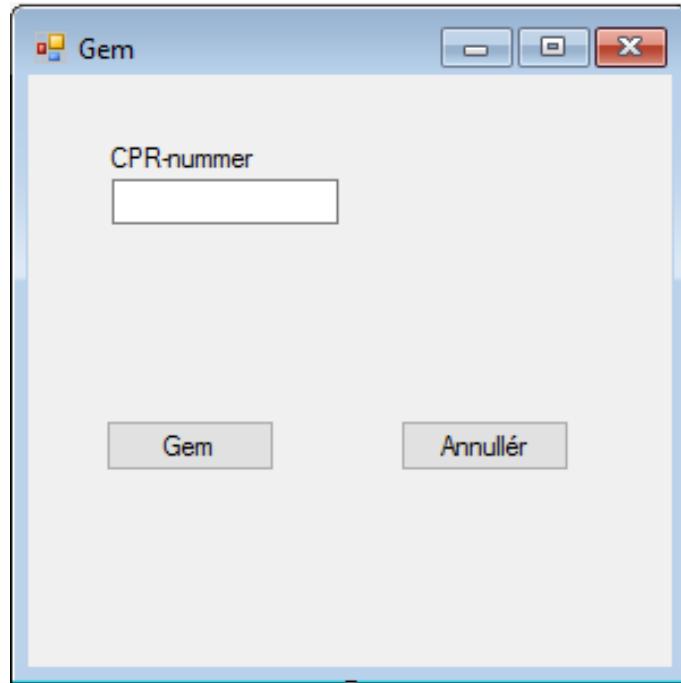
Figur 4.3: Login GUI

Dernæst åbnes hovedvinduet, hvor blodtrykssignalet vises grafisk, samt brugeren har mulighed for at benytte en række funktioner i form af knapper og trackbars, som set på figur 4.4



Figur 4.4: Blodtrykssystem hoved brugergrænseflade

Til sidst vises et vindue for gemmefunktionen, hvor brugeren skal indtaste et CPR-nummer, samt bekraeft afslutning af målingen, se figur 4.5:



Figur 4.5: Gem GUI

### Logiklag

Logiklaget er det lag, der spiller sammen med både data- og præsentationslag. Sammenspillet med præsentationslaget fungerer ved brugerens interaktion med systemet bearbejdes i logiklaget. Et eksempel herpå er det digitale filter, hvor brugeren gennem brugergrænsefladen tilslutter filteret. Dette igangsætter en metode i logiklaget, som sørger for, at signalet filtreres. Metoden er vist i figur 4.7 og figur 4.6:

```
if (filterOn)
{
    List<double> filterliste = new List<double>();
    for (int i = (grafcount < 5 ? 0 : grafcount - 5); i < grafcount; i++)
    {
        filterliste.Add(ufiltreredeGrafværdier[i]);
    }
    grafværdier[grafcount] = filter.filtrer(filterliste, ufiltreredeGrafværdier[0]);
}
```

Figur 4.6: Dataudvælgelse funktionen i logik-klassen

```

1 reference
public double filtrer(List<double> liste, double lastValue)
{
    if (liste.Count > 0)
    {
        return liste.Average();
    }

    return lastValue;
}

```

*Figur 4.7: Filtrer() i filter-klassen*

Figur 4.6 tager fem værdier og sender videre til filtrer(), som tager gennemsnittet på disse værdier, hvorefter gennemsnittet sendes tilbage til logik-klassen. Dette kaldes for moving average, som er en form for FIR-filter.

### Datalag

Datalaget er det lag, som spiller sammen med databasen og logiklaget. Det vil sige, at datalaget modtager brugerinputs fra brugergrænsefladen gennem logiklaget. Herefter henter datalaget informationer i en database, som efterfølgende sendes retur til logiklaget. Eksempelvis bliver login-oplysninger tjekket i databasen via datalaget, hvilket er vist i figur 4.8:

```

1 reference
public bool checkLogin(string username, string password)
{
    bool resultat = false;
    cmd = new SqlCommand("select kodeord from personale where personale_id ='" + username +
    conn.Open();

    // Udfør det ønskede SQL statement på DB
    rdr = cmd.ExecuteReader();
    if (rdr.Read())
    {
        resultat = (password.Equals(rdr.GetString(0)));
    }

    conn.Close();
    return resultat;
}

```

*Figur 4.8: checkLogin()*

Figur 4.8 viser, hvordan checkLogin() opretter forbindelse til en privat database, hvorefter de indtastede værdier tjekkes i databasen. Til slut lukkes forbindelsen igen. Værdierne, der tjekkes med, kan ses i personale-tabellen i databasen, som vist i figur 4.9:

	personale_id	kodeord
▶	1234	fido
*	NULL	NULL

Figur 4.9: Personale-tabel

Herudover er det også i datalaget, at data bliver indlæst fra DAQ'en. Dette gøres ved at udnytte, at DAQ'en har et asynkront kald i sig. Dette asynkrone kald gør, at der bliver leveret data fra DAQ til datalag løbende istedet for i intervaller, som ville være tilfældet ved synkrone kald. Det asynkrone kald tillader altså, at portioner af koden bliver udført løbende.

#### 4.2.2 Tråde

Blodtrykmålingssystemet skal simultant opfange målinger og vise disse på brugergrænsefladen. Dette stiller nogle krav til softwareopbygningen, da koden skal køre på samme tid. For at løse denne problemstilling, er der anvendt trådprogrammering. Trådprogrammering er et integreret værktøj i C#, som giver mulighed for at køre flere ting samtidigt på flere kerner i computerens CPU.

Softwareen er opbygget efter 3-lagsmodellen, og der findes en tråd i hvert lag, som har forskellige opgaver.

I datalaget laver DAQ-klassen asynkrone kald, når der skal opsamles målinger fra hardwareneden. Ved at lave asynkrone kald vil opsamlingsprocessen køre i en tråd for sig selv, hvilket forhindrer at programmet låser, mens der måles. Et eksempel herpå ses på figur 4.10:

```

analogCallback = new AsyncCallback(AnalogInCallback);

// Use SynchronizeCallbacks to specify that the object
// marshals callbacks across threads appropriately.
analogInReader.SynchronizeCallbacks = true;
analogInReader.BeginReadWaveform(SamplesPerChannel,
analogCallback, myTask);

```

Figur 4.10: Kodestykke fra startMåling() i klassen "DAQklasse". Her ses det, der er anvendt et asynkront kald

I logiklaget er der også en tråd, og denne tråd sørger for at opsamle og behandle data fra datalaget. Ved at anvende en tråd her, er der mulighed for at udføre logiske operationer på datamængden, såsom filtrering, og sende denne videre til grafen i præsentationslaget. Når man arbejder med Windows Forms i C#, vil disse pr. automatisk ligge i sin egen tråd,

hvilket deraf også er tilfældet her.

For at opdatere form-elementer fra en anden tråd, skal disse opdateres gennem en invoke-metode. Dette betyder, at tråden tjekker, om den kan komme til, og hvis det er tilfældet, opdateres elementet, som vist i figur 4.11:

```
private delegate void UpdateUICallback();

private void opdaterGraf()
{
    List<double> liste;
    liste = blodtryk;

    if (liste.Count > 0)
    {
        if (label1.InvokeRequired)
        {
            UpdateUICallback d = new UpdateUICallback(opdaterGraf);
            this.BeginInvoke(d);
        }
        else
        {

```

Figur 4.11: Eksempel på en `UpdateUICallback`. Metoden kalder sig selv ind til den har fået udført kodestykkerne, der efterfølger "else"

Der er også andre ting, man skal være opmærksom på, når man arbejder med tråde. Der har i programmeringsarbejdet været problemer med, at der opstår en konflikt, når to tråde prøver at skrive til det samme objekt på samme tid.

For at imødekomme dette problem, er der anvendt en lås i form af en "mutex": På denne måde forhindre man, at datalagets dataopsamler skriver til den samme liste, som logiklaget er i gang med at udføre logiske operationer på, vist på figur 4.12:

```

M.WaitOne();
målinger.Clear();
M.ReleaseMutex();
foreach (AnalogWaveform<double> waveform in sourceArray)
{
    for (int sample = 0; sample < waveform.Samples.Count; ++sample)
    {
        M.WaitOne();
        målinger.Add(waveform.Samples[sample].Value);
        M.ReleaseMutex();
    }
    Notify(målinger);
    currentLineIndex++;
    antalmålinger++;
}
}

```

Figur 4.12: Eksempel på en UpdateUICallback. Metoden kalder sig selv ind til den har fået udført kodestykkerne, der efterfølger "else"

#### 4.2.3 Observer

Observermønstret er et mønster, hvor et objekt, kaldet et subject, informerer en liste af observers, når noget er ændret eller gennemført ved at kalde en af deres metoder. Dette mønster har været nødvendigt at anvende, da grafen i præsentationslaget skal informeres om, hvornår der er nye data af hhv. logiklaget og datalaget. Det vil sige, at Logik-klassen og DAQ-klassen begge fungerer som subjects, mens GUI-laget og Logik-klassen fungerer som observers. Logik-klassen er subject for GUI, og Data-klassen er subject for Logik. Mønstret er sat op ved at lave to interfaces, ISubject og IObserver, som de respektive klasser arver fra og tvinges dermed til at implementere metoderne, som er beskrevet i interfaces. Dette ses i de tre efterfølgende figurer (figur 4.13, 4.14 og 4.15):

```

interface ISubject
{
    7 references
    void Subscribe(IObserver observer);
    3 references
    void Unsubscribe(IObserver observer);
    6 references
    void Notify(List<double> liste);
}

```

Figur 4.13: Interface ISubject, som giver observers mulighed for at abonnere på subjects

```
public interface IObserver
{
    7 references
    void Opdater(List<double> liste);
}
```

Figur 4.14: Interface `IObserver`. Subject sørger for at kalde metoden "Opdater()" for alle dens abonnenter

```
public void Subscribe(IObserver observer)
{
    observers.Add(observer);
}

3 references
public void Unsubscribe(IObserver observer)
{
    observers.Remove(observer);
}

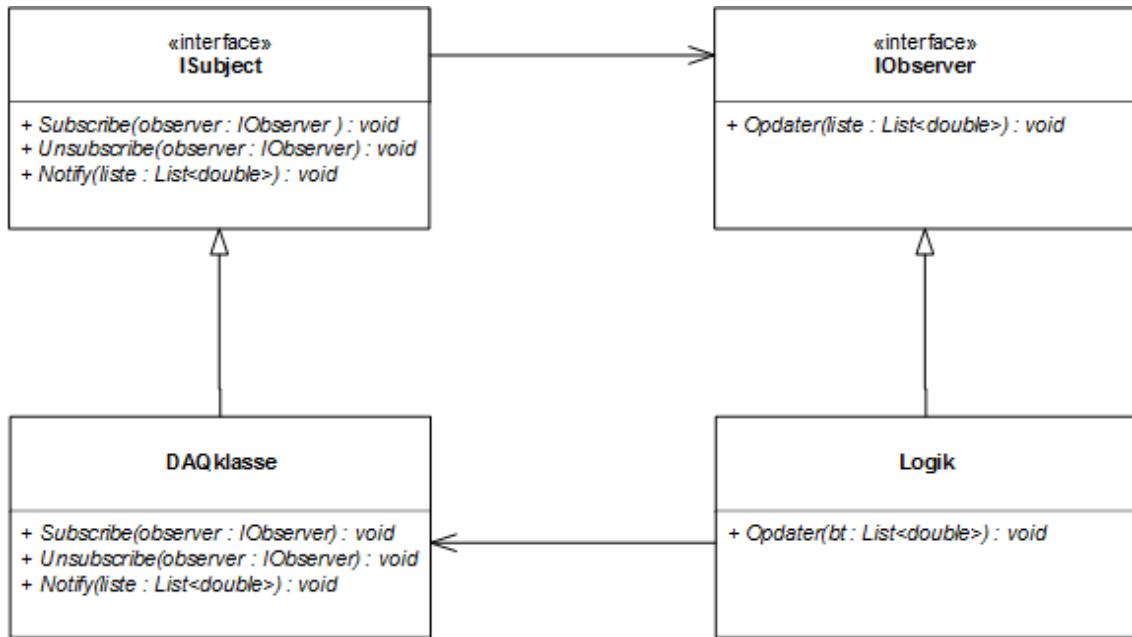
6 references
public void Notify(List<double> liste)
{
    foreach (var a in observers)
    {
        a.Opdater(liste);
    }
}
```

Figur 4.15: Implementeringen af subject-metoderne i Logik-klassen

Der er anvendt push-metode, hvilket vil sige, at subject sender sin data med som parameter i `Notify()`.

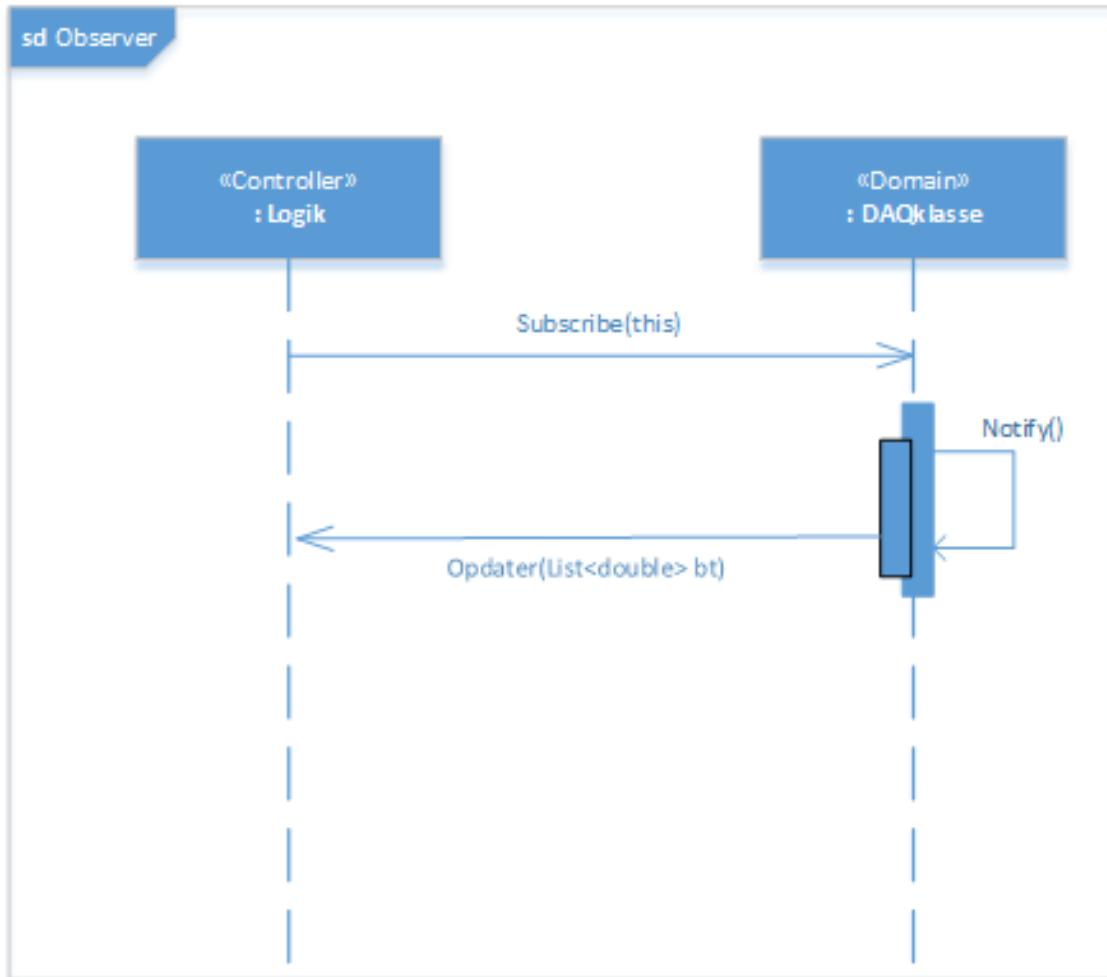
Observer-mønstret har vist sig nyttigt, da vi dermed undgår, at observers hele tiden skal tjekke, om der er kommet ny data. Dette sparer computerkrafter, hvilket er en nødvendighed, da mange af de opståede problemer er opstået på baggrund af, at computeren ikke har kunnet arbejde hurtigt nok.

I figur 4.16 er vist et UML-klassediagram over observermønstrene. Man kan se at logik arver fra `IObserver` og `DAQklassen` fra `ISubject`.



Figur 4.16: Klassediagram over observer mønsteret

Ud fra figur 4.16 er der udarbejde et sekvensdiagram, som ses på figur 4.17:

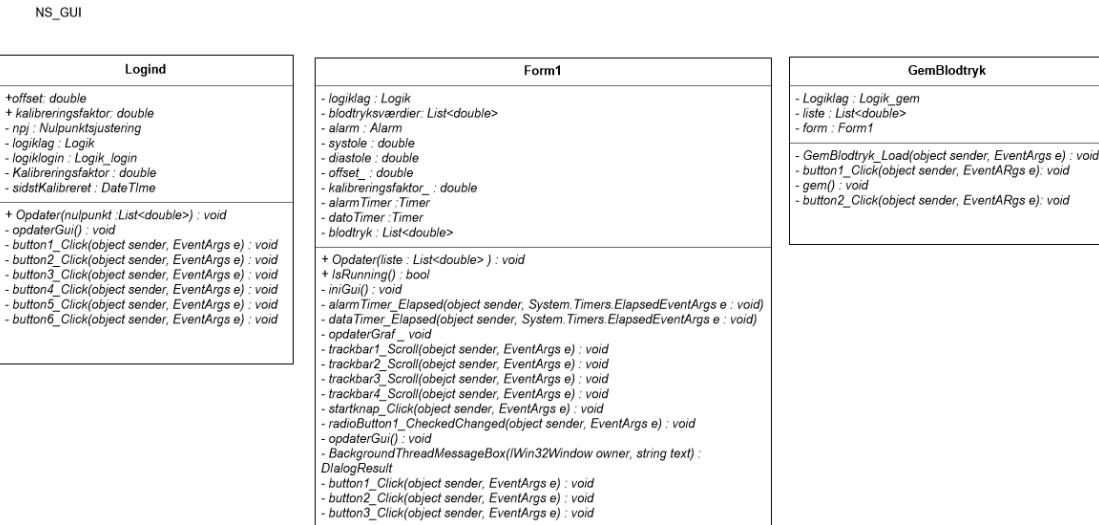


Figur 4.17: Sekvensdiagram over observer mønsteret

#### 4.2.4 UML klassediagram

I dette afsnit er der udarbejdet et klassediagram, der viser opbygningen af koden. Diagrammet er blevet ind delt i 3.

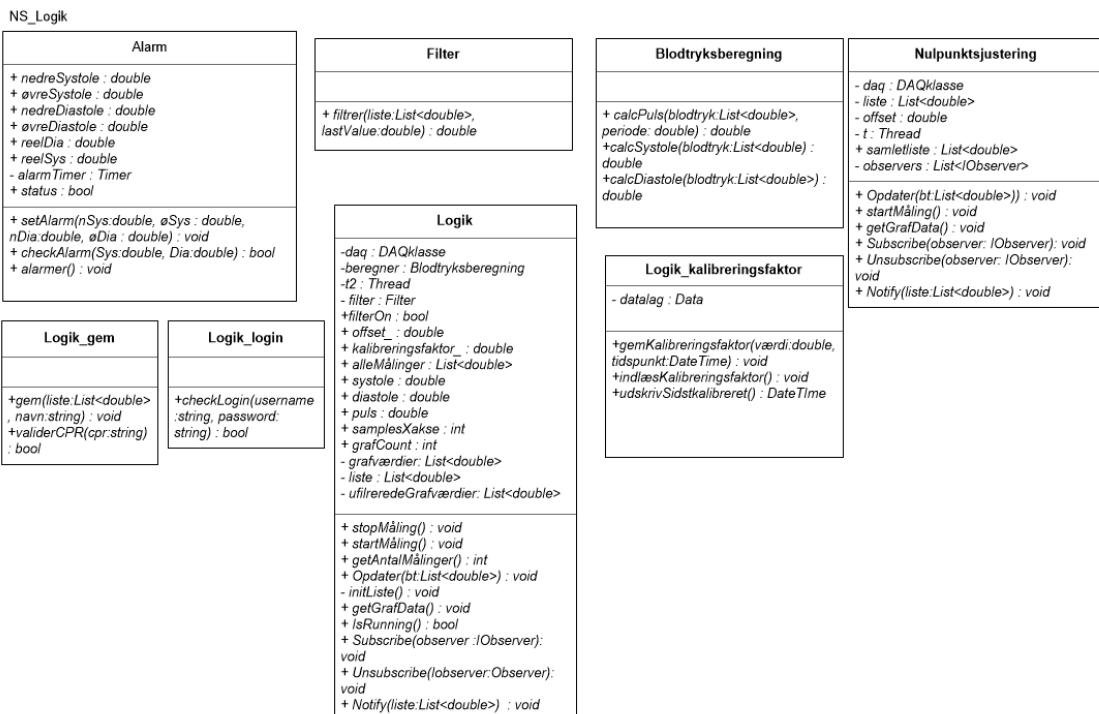
Først er der blevet lavet et klassediagram for klasserne i præsentationslaget. Dette ses på figur 4.18



Figur 4.18: Klassediagram

Det ses her, at præsentationslaget består af 3 klasser. Klasserne for præsentationslaget er her vist sammen med tilhørende attributter, parametre, typer og metoder.

Herefter blev der udarbejdet et klassediagram for logiklaget. Dette ses på figur 4.19

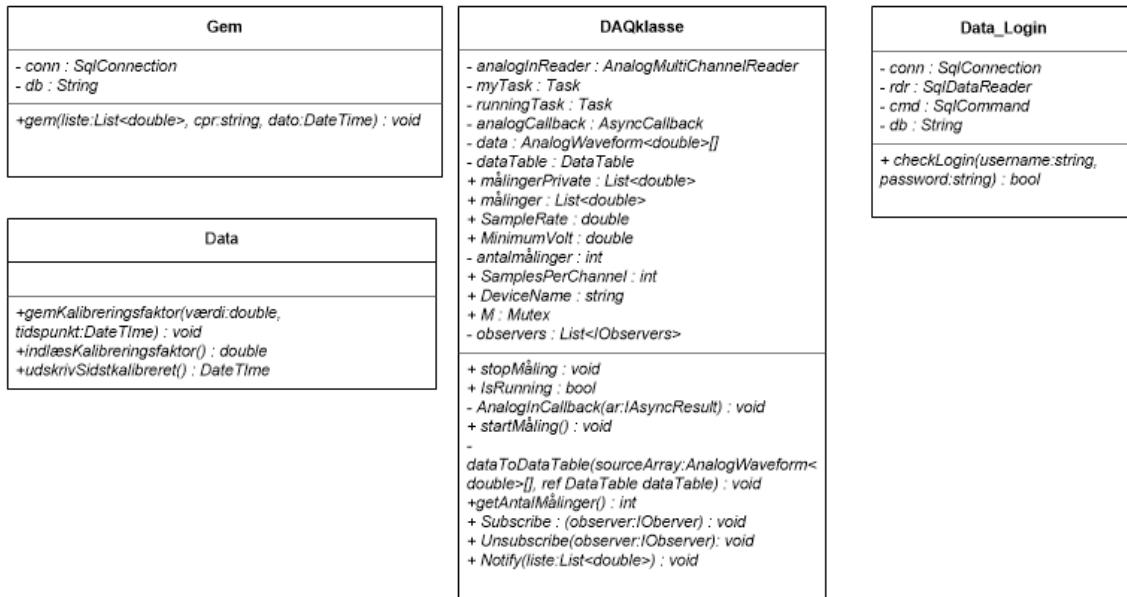


Figur 4.19: Klassediagram logiklaget

Det ses her, at logiklaget er mere komplekst og består af flere klasser end præsentationslaget.

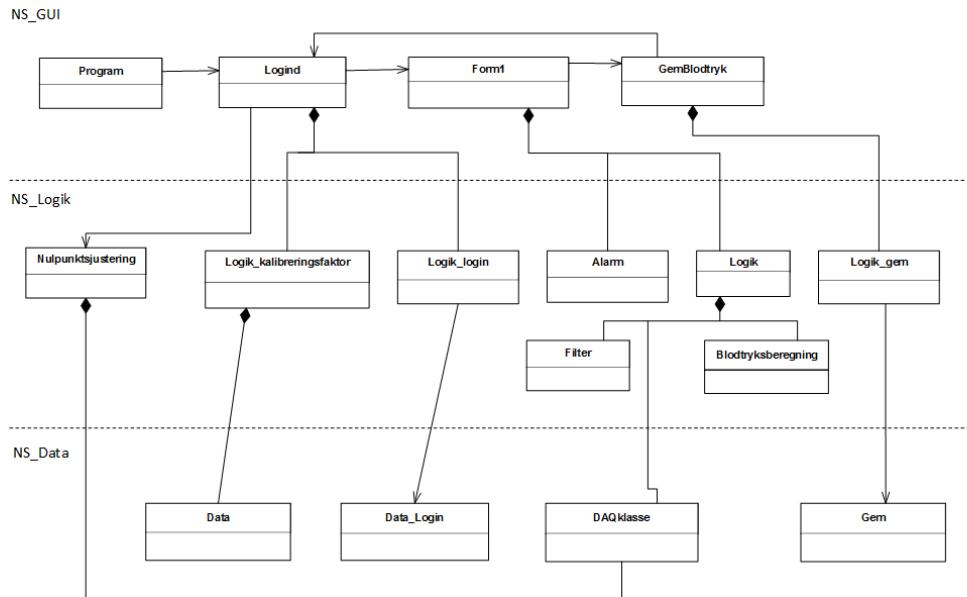
Som den sidste del af klassediagrammet blev der udarbejdet et klassediagram for datalaget. Dette ses på figur 4.20

NS\_Data



Figur 4.20: Klassediagram datalag

På ovenstående diagrammer er klasserne og dets metoder og attributter vist. Disse diagrammer viser dog ikke hvordan klasserne spiller sammen med hinanden. Sammenspillet mellem klasserne er beskrevet i et klassediagram uden attributter og metoder, men forholdet mellem klasserne. Det ses i figur 4.21



Figur 4.21: Forholdet mellem software klasser

Ud fra dette ses det at nogle af klasserne har en komposition i mellem sig og andre har en association. Kompositionen mellem to klasser fortæller om de to klasser starter samtidig. Association fortæller at klasserne har kendskab til hinanden, men de starter ikke samtidig.

#### 4.2.5 Kode udsnit & diagrammer

I dette afsnit vil der være udsnit af det vigtigste og mest interessante kode. Der vil ligeledes være en beskrivelse af, hvilken funktion koden har.

Koden beskrives yderligere ud fra sekvensdiagrammer og aktivitetsdiagrammer. Dette vil danne et systemisk overblik over hvordan metoderne i koden fungerer.

Der er gjort overvejelser om, hvordan algoritmerne for beregning af hhv. diastole, systole og puls skal beregnes. Algoritmerne er implementeret som metoder, hvilket giver mulighed for genbrug og bør ideelt set virke på ethvert påsat blodtrykssignal.

De pågældende metoder ligger i klassen "Blodtryksberegning" og modtager alle en liste som parameter, kodeudsnittet hertil ses i figur4.22:

```

public double calcSystole(List<double> blodtryk)
{
    List<double> liste = blodtryk;
    int antal = liste.Count;

    List<double> blodtrykliste = new List<double>();
    blodtrykliste = blodtryk;
    double baselineoverskredet = 0;
    double baseline = blodtrykliste.Max()*0.8;
    double maxTrykOverBaseline = baseline;
    List<double> toppunkter = new List<double>();

    for (int i = 0; i < antal - 1; i++)
    {
        if (liste[i] > maxTrykOverBaseline)
        {
            maxTrykOverBaseline = liste[i];
            baselineoverskredet++;
        }
        else
        {
            if (i + 2 >= antal)
            {
                if (liste[i] < maxTrykOverBaseline && baselineoverskredet > 0)
                {
                    toppunkter.Add(liste[i - 1]);
                    baselineoverskredet = 0;
                    maxTrykOverBaseline = baseline;
                    i = i + antal / 25;
                }
            }
            else
            {
                if (liste[i] < maxTrykOverBaseline && baselineoverskredet > 0 && liste[i] > liste[i - 1])
                {
                    toppunkter.Add(liste[i - 1]);
                    baselineoverskredet = 0;
                    maxTrykOverBaseline = baseline;
                    i = i + antal / 25;
                }
            }
        }
    }

    if (toppunkter.Count != 0)
    {
        return toppunkter.Average();
    }
    return 0;
}

```

Figur 4.22: Metoden til beregning af systolisk blodtryk

Som det ses på figur 4.22, modtager metoden en liste blodtryksværdier af typen double. Hernæst bliver de anvendte variable initieret. Algoritmen tager udgangspunkt i en baseline,

som beregnes til at være 80% af den maksimale værdi på listen. Hernæst tjekkes der løbende om en baseline og foregående værdi overskrides, og hvis dette pludselig ikke er tilfældet mere, vil punktet blive anset for værende punktet efter en systoleværdi, og denne tilføjes til en liste. Metoden er forbedret ved at tjekke 2 målinger ud i fremtiden, hvilket forhindrer, at evt. støj bliver sat til systolen. Når toppunkter er fundet, findes gennemsnittet af disse. Man kunne have valgt en simpelere løsning ved blot at kalde den højeste værdi på listen for systolen, men den metode er fravalgt, da DAQ'en kan give nogle peaks, der uhensigtsmæssigt ville igangsætte alarmen. Ved at tage gennemsnittet formindsker man altså risikoen for, at systolen bliver en unormal høj værdi. En ulempe ved den valgte algoritme er, at baseline findes ud fra den maksimale værdi. Hvis den maksimale værdi er unaturlig høj, vil baselinien blive sat for højt til, at toppunkterne kan detekteres.

Hvis et toppunkt ikke detekteres af en eller anden årsag, er det ikke katastrofalt, da gennemsnittet blot bliver beregnet ud fra de andre toppunkter.

Algoritmen for diastole er implementeret efter samme princip, blot hvor der tjekker for de laveste værdier, som det ses på figur 4.23:

```
public double calcPuls(List<double> blodtryk, double periode)
{
    List<double> blodtrykliste = new List<double>();
    blodtrykliste = blodtryk;
    double baselineoverskredet = 0;
    double baseline = blodtrykliste.Max() * 0.8;
    double maxTrykOverBaseline = baseline;

    List<double> pulsslug = new List<double>();

    for (int i = 0; i < blodtrykliste.Count; i++)
    {
        if (blodtrykliste[i] > maxTrykOverBaseline)
        {
            maxTrykOverBaseline = blodtrykliste[i];
            baselineoverskredet++;
        }
        if (blodtrykliste[i] < maxTrykOverBaseline && baselineoverskredet > 0)
        {
            pulsslug.Add(i);
            baselineoverskredet = 0;
            maxTrykOverBaseline = baseline;
            i = i + blodtrykliste.Count / 25;
        }
    }
    //return (pulsslug.Count - 1) / (pulsslug[pulsslug.Count - 1] * periode - pulsslug[0] * periode);
    return pulsslug.Count / (periode * blodtrykliste.Count) * 60;
}
```

Figur 4.23: Algoritme til detektering af puls

Algoritmen til detektering af puls foregår også efter dette princip. Her skal man blot

kende perioden eller frekvensen på samplingen og medsende denne som parameter. I denne algoritme er det mere uhensigtsmæssigt, hvis et toppunkt ikke detekteres, da det kan give en for lav puls.

### Dataopsamling

Følgende afsnit beskriver dybdegående, hvordan data fra DAQ'en bliver sendt fra DAQklassen til visning på brugergrænseflade. Samspillet mellem klasserne er uddybet i afsnittet med tråde og observer-mønstret.

### DAQ-klassen

I denne klasse forgår kontakten med DAQ'en, som indstilles og dernæst anvendes til dataopsamling. DAQ'en fungerer som en A/D-converter, og det er det digitale signal, vi får indlæst i programmet. DAQ-klassen er bygget med et i forvejen fungerende program fra National Instruments som skabelon[3].

```
public DAQklasse()
{
    SampleRate = 1000;
    MinimumVolt = -10;
    MaximumVolt = 10;
    SamplesPerChannel = 20;
    DeviceName = "Dev1(ai0";
    antalmalinger = 0;
    malinger = new List<double>();
    observers = new List<IObserver>();
    M = new Mutex();
}
```

Figur 4.24: DAQ-klassens constructor

I DAQ-klassens constructor sættes DAQ'en op, og de nødvendige lister, låse og attributter oprettes. Det er et krav fra IHA's side, at samplingen skal foretages med 1000Hz. Det valgt, at der skal indlæses 20 samples adgangen, hvilken vil sige, at programmet modtager målinger hvert 1/50 sekund, vist i figur 4.25.

```

private void dataToDataTable(AnalogWaveform<double>[] sourceArray, ref DataTable dataTable)
{
    // Iterate over channels
    int currentLineIndex = 0;

    M.WaitOne();
    målinger.Clear();
    M.ReleaseMutex();
    foreach (AnalogWaveform<double> waveform in sourceArray)
    {

        for (int sample = 0; sample < waveform.Samples.Count; ++sample)
        {
            M.WaitOne();
            målinger.Add(waveform.Samples[sample].Value);
            M.ReleaseMutex();
        }

        Notify(målinger);

        currentLineIndex++;
        antalmålinger++;
    }
}

```

Figur 4.25: Metoden `dataToDataTable()`, der sørger for at putte målinger over i en liste af typen `double`

Metoden `dataToDataTable()` sørger for at putte målinger over i listen "målinger", som tømmes før metoden køres, hvilket sikrer, at der kun ligger 20 målinger i listen.

Efter endt dataopsamling, informeres logiklaget med metoden "`Notify()`", hvilket jf. observermønstret kalder metoden "`Opdater()`" med listen som parameter, hvilet ses på figur 4.26.

```

public void Opdater(List<double> bt)
{
    liste.Clear();
    daq.M.WaitOne();
    liste.AddRange(bt);
    alleMålinger.AddRange(liste);
    daq.M.ReleaseMutex();
}

```

Figur 4.26: Metoden `Opdater()` som ligger i laget "Logik"

I figur 4.26 bliver listen fra DAQklassen kopieret over i logikklassen til videre bearbejdning.

```

public void getGrafData()
{
    while (IsRunning())
    {

        list.Clear();
        list.AddRange(liste);

        if (list.Count > 0)
        {
            //M.WaitOne();
            grafværdier[grafcount] = list.Average() * kalibreringsfaktor_ - offset_;
            ufiltreredeGrafværdier[grafcount] = list.Average() * kalibreringsfaktor_ - offset_;

            int antal = list.Count;
            if (filterOn)
            {
                List<double> filterliste = new List<double>();
                for (int i = (grafcount < 5 ? 0 : grafcount - 5); i < grafcount; i++)
                {
                    filterliste.Add(ufiltreredeGrafværdier[i]);
                }
                grafværdier[grafcount] = filter.filtrer(filterliste, ufiltreredeGrafværdier[0]);
            }

            Notify(grafværdier);
            //M.ReleaseMutex();
            grafcount++;
            if (grafcount == samplesXakse)
            {
                systole = beregner.calsSystole(grafværdier);
                diastole = beregner.calsDiastole(grafværdier);
                grafcount = 0;
                puls = beregner.calcPuls(grafværdier, 0.02);
            }
        }

        Thread.Sleep(5);
    }
}

```

Figur 4.27: *getGrafData()* i laget Logik

Metoden *getGrafData()*, som ses i figur 4.27, kører i sin egen tråd og sørger løbende for, at logiske operationer udføres på listen sideløbende med, at data hentes. Det er altså denne metode, der klargører listen til at blive vist på GUI'en. Det er også her, kalibreringsfaktoren ganges og offsettet trækkes fra på samtlige tal i listen. Det bemærkes desuden, at de 20 målinger, der modtages fra DAQklassen er skåret ned til en ved at tage gennemsnittet af de tyve. På denne måde overbelastes systemet ikke ved at skulle vise 1000 målinger pr. sekund i en graf, men samtidig er alle målinger repræsenteret ligevægtigt. Hvis brugeren har slået filteret til, vil der blive skåret fem målinger ud af listen som sendes til filtret før listen med grafens x-værdier opdateres. Når listen med x-værdier er fyldt ud, beregnes systole, diastole og puls. Denne klasse er altså på mange måder en controller-klasse, der sørger for, at tingene sker efter hensigten.

Der er til sidst implementeret samme observermønster, hvilket sender målingerne til GUI-laget efter endt behandling. Dette vises 4.28:

```

private void opdaterGraf()
{
    List<double> liste;
    liste = blodtryk;

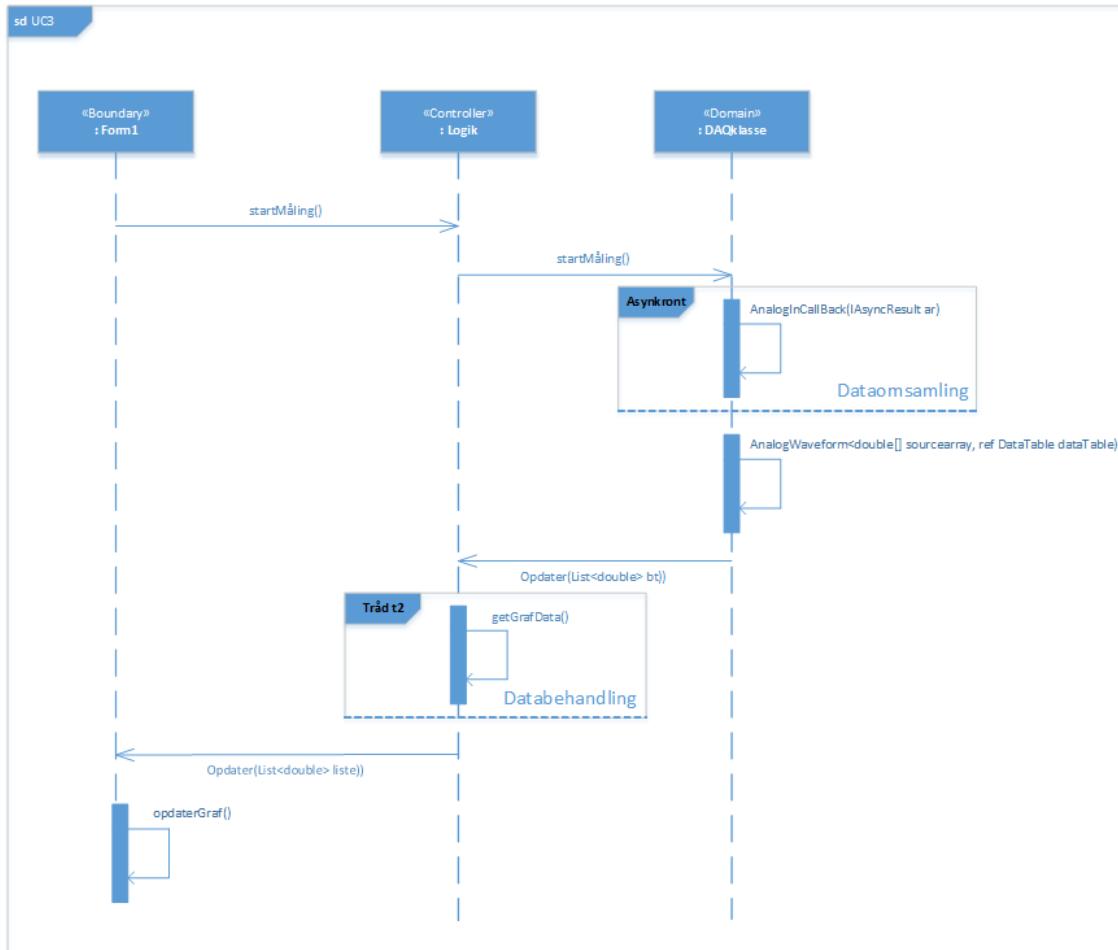
    if (liste.Count > 0)
    {
        if (label1.InvokeRequired)
        {
            UpdateUICallback d = new UpdateUICallback(opdaterGraf);
            this.Invoke(d);
        }
        else
        {
            int counter = logiklag.grafcount;
            //logiklag.M.WaitOne();
            chart1.Series[0].Points.DataBindY(liste);
            chart1.Series[1].Points.DataBindY(liste);
            chart1.Series[1].Color = Color.Transparent;
            chart1.Series[0].Color = Color.Red;
            diastole = logiklag.diastole;
            systole = logiklag.systole;
            if (systole == 0)
            {
                label1.Text = "---";
                label2.Text = "--";
            }
            else
            {
                label1.Text = Math.Round(systole, 0).ToString() + "/" + Math.Round(diastole, 0).ToString();
                label2.Text = Math.Round(logiklag.puls, 0).ToString();
            }
            if (counter > 0)
            {
                chart1.Series[1].Points[counter - 1].Color = Color.Blue;
                chart1.Series[0].Points[counter].Color = Color.Transparent;
            }
            if (counter < logiklag.samplesXakse - 1)
            {
                chart1.Series[0].Points[counter + 1].Color = Color.Transparent;
            }

            if (alarm.checkAlarm(systole, diastole))
            {
                pictureBox3.Show();
            }
            else
            {
                pictureBox3.Hide();
            }
            //logiklag.M.ReleaseMutex();
        }
    }
}

```

Figur 4.28: Metoden opdaterGraf() i Form1-klassen

Metoden er også beskrevet ved et sekvensdiagram i figur 4.29:



Figur 4.29: Sekvensdiagram for UC3

Sekvensdiagrammet viser samspillet mellem de nævnte tre klasser, når en måling foretages. Nulpunktsjusteringen foregår efter samme princip, hvor der dog kun hentes en enkelt måling ind. Nulpunktsjusteringen ligger i login-vinduet, som er det første, der starter. Nulpunktjusteringen foretages før der logges ind.  
 Login-oplysningerne tjekkes i en ekstern SQL-database, som er opbygget i forbindelse med projektet, hvilet ses i figur 4.30:

```

public Data_Login()
{
    conn = new SqlConnection("Data Source=webhotel10.ihb.dk;Initial Catalog=" + db + ";Persist Security Info=True;User ID=" + db + ";Password=" + db + "");
}

public bool checkLogin(string username, string password)
{
    bool resultat = false;
    cmd = new SqlCommand("select kodeord from personale where personale_id ='" + username + "'", conn);
    conn.Open();

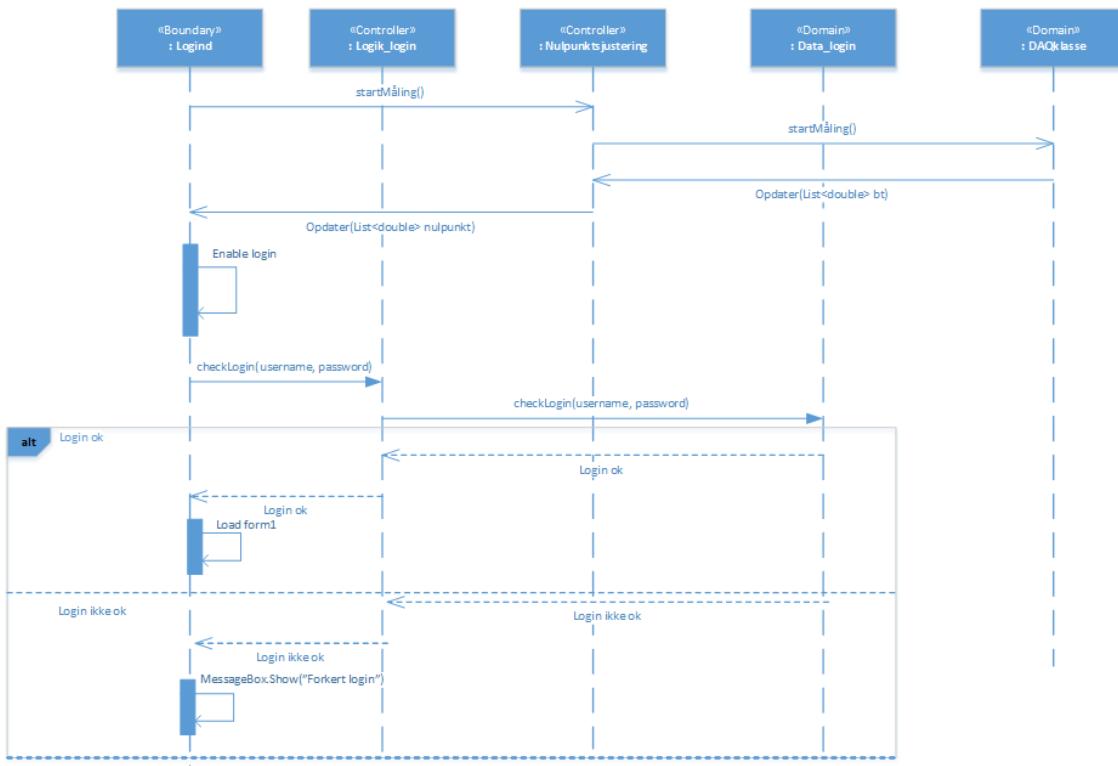
    // Udfør det ønskede SQL statement på DB
    rdr = cmd.ExecuteReader();
    if (rdr.Read())
    {
        resultat = (password.Equals(rdr.GetString(0)));
    }

    conn.Close();
    return resultat;
}

```

Figur 4.30: Klassen Data\_Login. Der er vist, hvordan databasen tjekker, hvorvidt loginoplysningerne er korrekte

Yderligere er denne metode også beskrevet ved et sekvensdiagram i figur 4.31:



Figur 4.31: Sekvensdiagram for UC2 med undtagelsen medtaget. Sekvensdiagrammet er medtaget for at vise, hvordan opstartsfasen forløber. En nulpunktsjustering giver anledning til login.. Klassen Data\_login tjekker oplysningerne i en ekstern SQL-database

# Test 5

---

Version	Dato	Ansvarlig	Beskrivelse
0.1	8/09 2015	Alle	Oprettelse af dokument
0.2	09/12 2015	SSK, NHL, FRM	Påbegyndelse af hardware testafsnit
0.3	09/12 2015	TSN, JTH, MFJ	Påbegyndelse af software testsafsnit

## 5.1 Indledning

Testafsnittet giver en beskrivelse af hvilke tests, der løbende er blevet foretaget på både hardware og software. Testene består af modultest, hvor enkelte delelementer af systemet testes uafhængigt af hinanden. Modultesten er til for at verificere de enkelte enheder virker efter hensigten.

En mere omfattende test, i form af en integrationstest, hvor de større dele af både hardware og software testes afhængigt af hinanden. Ved integrationstesten ligger fokus på, hvordan systemet fungerer, set fra udviklers synspunkt.

Begge tests ligger til grund for den endelige accepttest, hvor testen ses fra kundens synspunkt.

## 5.2 Modultest hardware

Til test af hardwaredelen blev der først udført modultest på henholdsvis forstærkeren og det analoge filter. Efterfølgende blev der udført integrationstest på forstærkeren og det analoge filter sat sammen med tryktransduceren. Sluttligt blev den samlede hardwaredel bestående af tryktransducer, forstærker og analogt filter testet på en vandsøje.

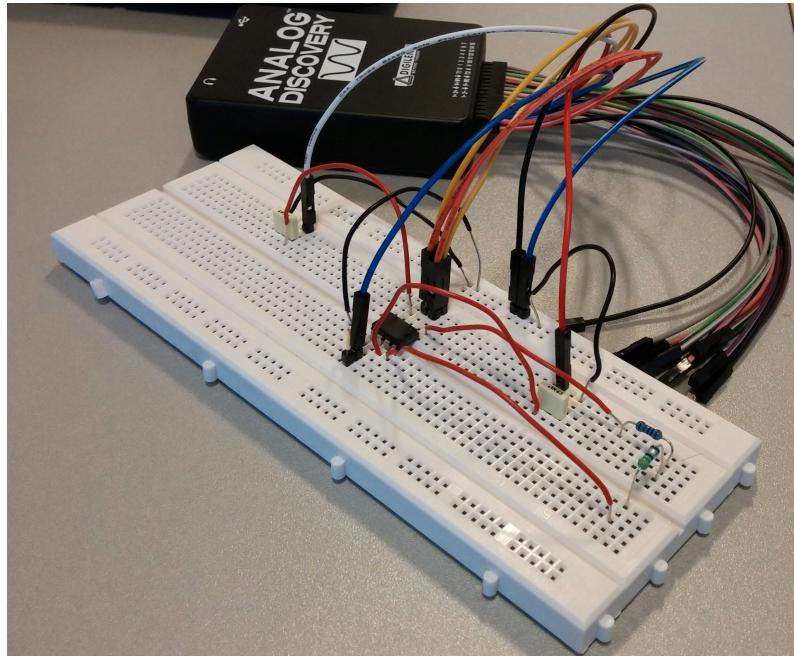
Resultater kan ses i bilag 6.3.

### 5.2.1 Modul test af forstærkeren

I modultesten af forstærkeren blev Analog Discovery brugt som spændingsforsyning, signalgenerator og oscilloskop på to forskellige målepunkter placeret ved henholdsvis indgangssignalet og udgangssignalet.

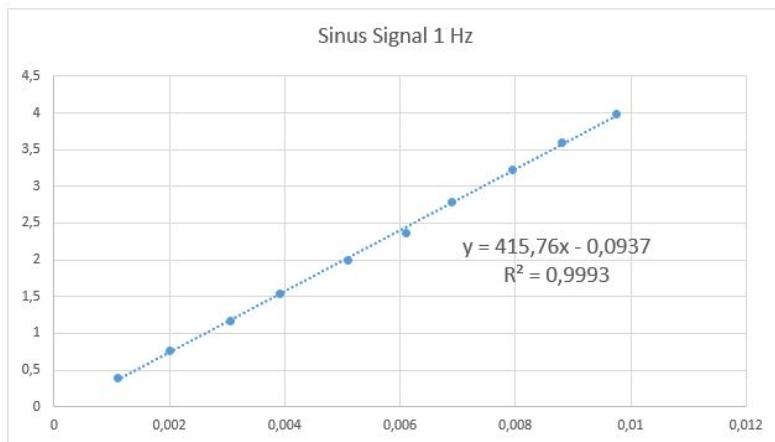
Da tryktransduceren forventes at have output-spændinger i området 0 til 6,25mV blev signalet fra transduceren simuleret ved en sinus på 1Hz uden offset. Den laveste amplitude

for indgangssignalet var 1mV. For hver måling blev amplituden for indgangs signalet sat op med 1mV indtil den sidste måling, hvor amplituden var nået op på 10 mV. Således blev der 10 forskellige målinger. Reelt set var det kun nødvendigt at teste op til 6mV eller 7mV, da det er herimellem, at man kan forvente et max tryk fra transduceren vil ligge. Da der alligevel er taget målepunkter op til 10mV, skyldes det, at der ved flere målinger kan laves en ”pænere” tendenslinje ved lineær regression. Opstillingen til denne test ses på figur 5.1:



Figur 5.1: Måleopstilling ved modul test af forstærkeren.

Ud af målingerne blev der foretaget lineær regression over de 10 målepunkter. Tendenslinjen der kom ud af den lineære regression blev som vist på figur 5.2:



Figur 5.2: Målpunkter samt tendenslinje for målingerne ved modultest af forstærkeren med 1Hz sinus-signal.

I figur 5.2 kan Y beskrives som værende forstærkningen ved en given frekvens. Konstanten

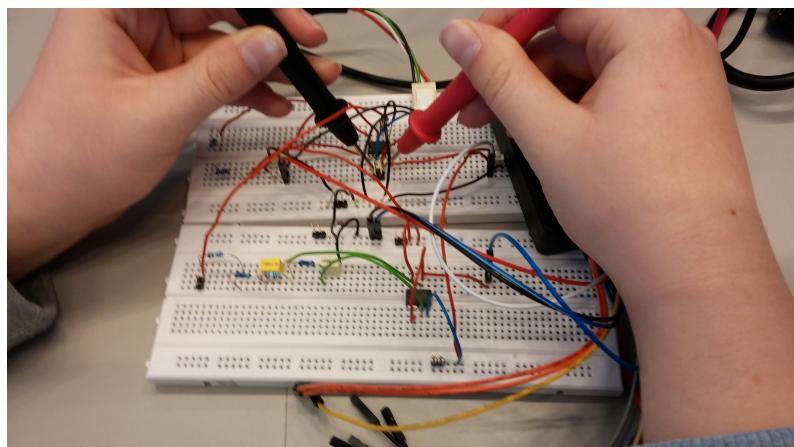
415,8 er den reelle forstærkning som er målt for forstærkeren. Skæringen med y-aksesen burde være 0, men grundet måleusikkerheder er den blevet -0,09, hvilket også er acceptabelt. Den høje  $R^2$  værdi indikerer, at der er en tydelig lineær sammenhæng mellem den påtrykte spænding og spændingen af output, dvs. forstærkningen er lineær.

Senere påtryktes samme måleopstilling et DC-signal med amplituder startende fra 3mV op til 10mV. Igen blev amplituden øget med 1mV for hvert måling således, at der blev 7 målepunkter. Ved målinger foretaget på signaler med både 1mV og 2mV er offsettet i Analog Discovery så stort, at målingerne ikke kan foretages.

Forstærkning, som blev beregnet på baggrund af de målte output og input, viste sig at være meget varierende. Der blev beregnet forstærkninger fra et sted mellem 455 gange og 784 ganges forstærkning. Jo lavere offset desto større forstærkning. Ud fra denne iagttagelse blev det antaget, at denne meget store afvigelse måtte skyldes et negativt offset i Analog Discovery.

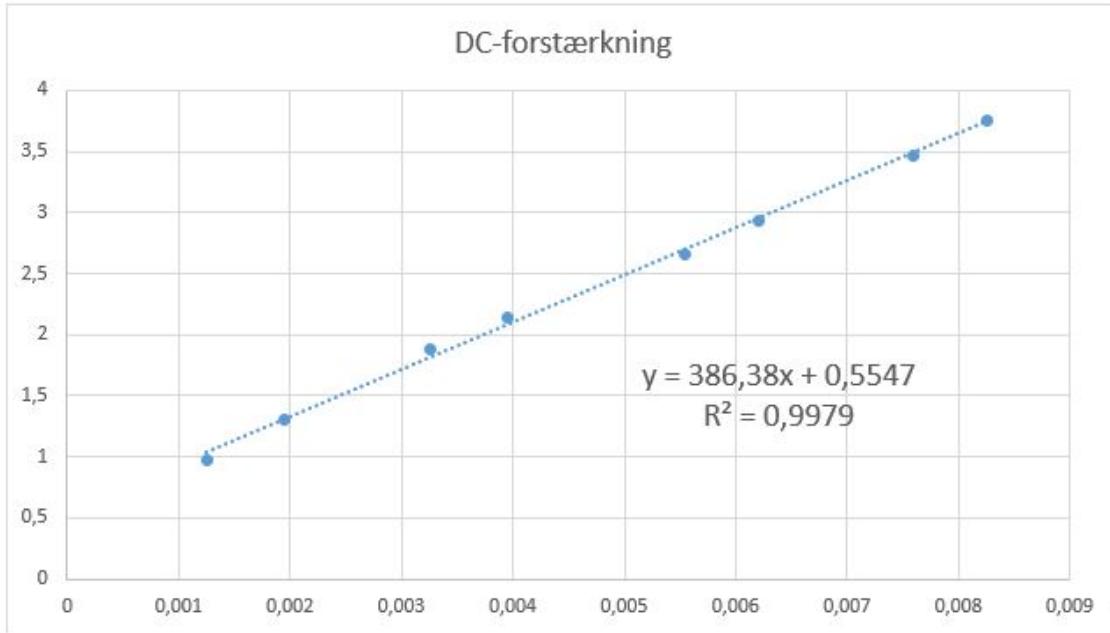
For at finde frem til dette offset blev output-spændingen divideret med 400, som var den ventede forstærkning. Det målte input og resultatet fra den forrige udregning blev trukket fra hinanden. Heraf sås det, at offsettet for Analog Discovery måtte være 1,2mV.

Dette blev eftertestet med et digitalt multimeter, der rigtig nok målte 1,2mV højere på indgangen af forstærkeren end det som Analog Discovery målte. Opstillingen hertil ses i figur 5.3:



Figur 5.3: Måling på indgangen af forstærkeren med det digitale multimeter.

Efter at have foretaget lineær regression på de syv målepunkter så tendenslinjen ud som følger på figur 5.4:

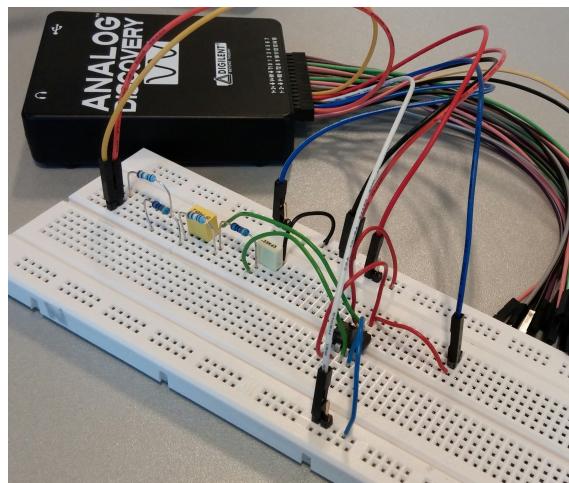


Figur 5.4: Målpunkter samt tendenslinje for målingerne ved modultest af forstærkeren med DC-signal.

Ud af figur 5.4 ses, at forstærkningen givet ud fra de syv målpunkter er på 386, hvilket er en del under den målte forstærkning, når systemet påtrykkes et sinussignal i stedet for et DC-signal. Dette kan skyldes, at der i målingen af DC-signalet er færre målpunkter, og at det dermed bliver en mere usikker måling, da selv en mindre måleusikkerhed har en større indflydelse på den målte forstærkning.

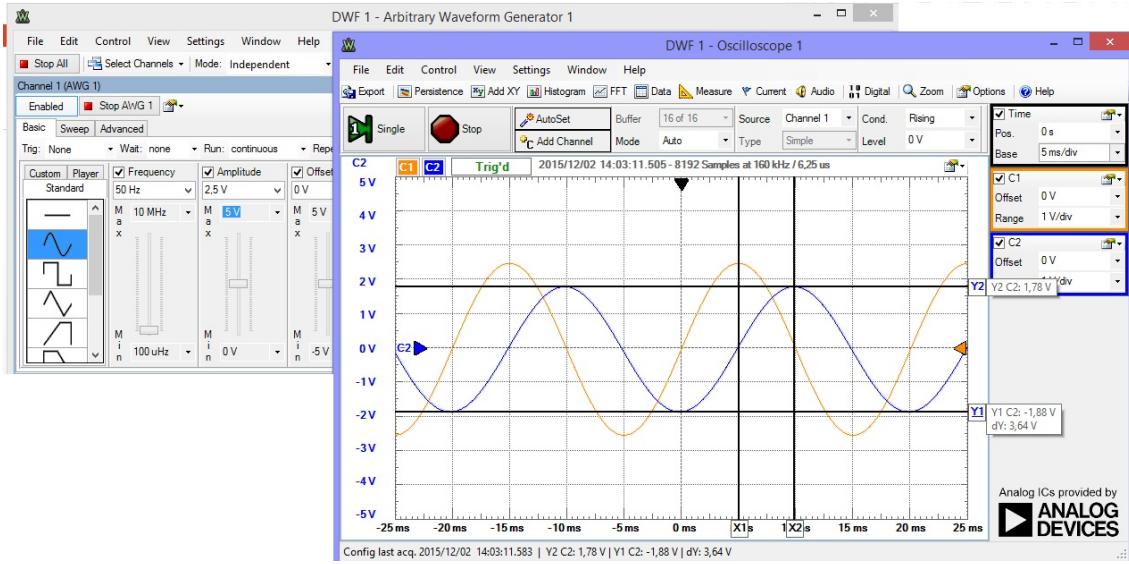
### 5.2.2 Modul test af det analoge filter

I modultesten af det analoge filter blev Analog Discovery brugt som spændingsforsyning, signalgenerator og som oscilloskop til at måle amplituderne for input-signalet og output-signalet for filteret. Måleopstillingen hertil ses på figur 5.5:



Figur 5.5: Måleopstilling ved modultest af det analoge filter.

I modultesten af det analoge filter ændres frekvensen for det påtrykte sinussignal på filter indgangen. Der blev målt på i alt 21 målepunkter som lå i intervallet 1Hz til 500Hz; se bilag 6.3 for de præcise angivelser af de udvalgte målefrekvenser. Output-amplituden samt  $\Delta t$  mellem de to grafer blev aflæst. På baggrund af disse målinger blev dæmpningen af signalet afbildet, hvilket ses på figur 5.6.



Figur 5.6: Aflæsning af amplitude størrelse for outputsignalet, C2, fra filteret når inputsignalet, C1, har en amplitude på 2,5V

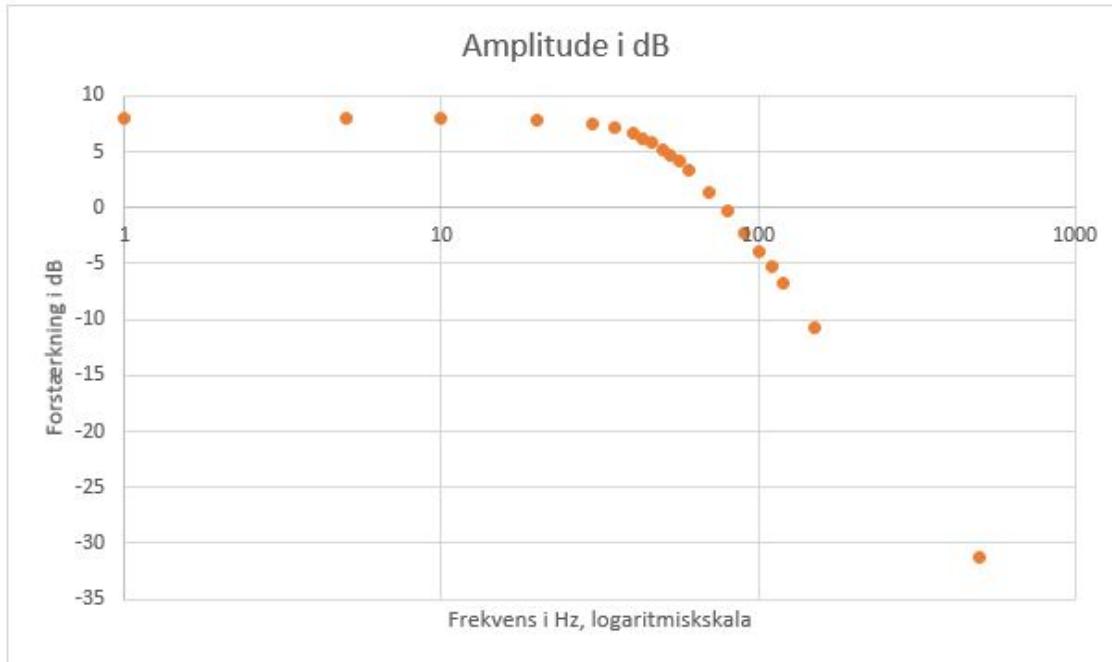
Ved brug af 3dB frekvensen er det muligt at vurdere om knækfrekvensen for filteret er som den skal være. Denne kan udregnes ved følgende ligning 5.1:

$$3dB_{frekvens} = \frac{V_{max}}{\sqrt{2}} \quad (5.1)$$

I dette tilfælde bliver udregningen som vist i ligning 5.2:

$$3dB_{frekvens} = \frac{2,5V}{\sqrt{2}} = 1,76V \quad (5.2)$$

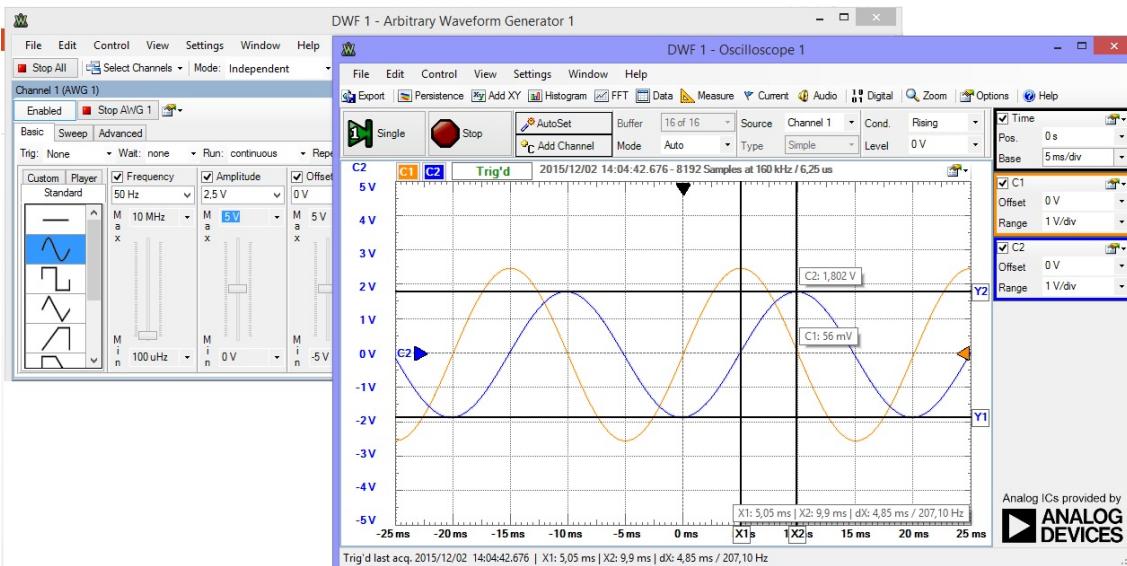
Ved omregning af de 1,76V til dB fås 4,9dB. Hvis filteret virker efter hensigten skal frekvensen ved 4,9dB være 50Hz. Denne kan aflæses i figur 5.7



Figur 5.7: Forstærkningen i dB set i forhold til den frekvens som det påtrykte signal har.

Som det kan ses ud af figur 5.7 er forstærkningen ved 3dB frekvensen et sted imellem de to målepunkter på 50Hz og 53Hz. Derfor må filterets knækfrekvens ligge mellem 50Hz og 53Hz.

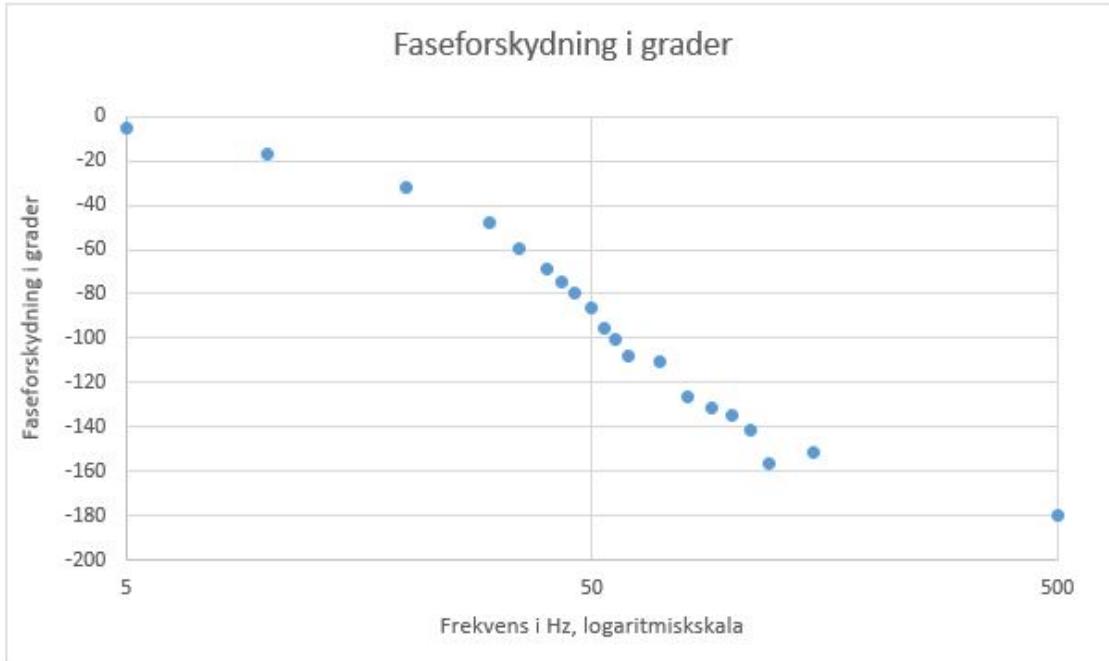
Herefter blev tidsforskydningen aflæst, således at fasedrejet kunne udregnes. Faseforskydningen ved de forskellige frekvenser er vist på figur 5.8:



Figur 5.8: Aflæsning af tidsforskydningen for outputsignalet, C2, fra filteret set i forhold til indgangssignalet C1

Det gælder generelt for et 2.ordens filter af standarttypen som der er arbejdet med i gen-

nem projektet, at fasen er  $0^\circ$  en dekade før filterets knækfrekvens og falder med  $90^\circ/\text{dekade}$  frem til dekaden efter knækfrekvensen. Derfor skal filteret i praksis have et fasedrej på  $-90^\circ$  ved knækfrekvensen 50Hz.



Figur 5.9: Faseforskysningen målt ved forskellige frekvenser for det analoge filter.

Som det ses ud af figur 5.9 er fasedrejet ved frekvensen 5Hz på  $-5,2^\circ$ , denne skulle reelt set have været  $0^\circ$ .

Ved knækfrekvensen som er sat til 50Hz er fasedrejet  $-86,4^\circ$ , hvor fasedrejet skulle have været  $-90^\circ$ . Ved målingen for 53Hz er fasedrejet  $-95,4^\circ$ . Dermed kan der argumenteres for, at filterets reelle knækfrekvens må ligge et sted imellem 50Hz og 53Hz. For målingen på 500Hz er fasedrejet  $-180^\circ$ , hvilket stemmer overens med teorien for 2.ordens filterets fasekarakteristik.

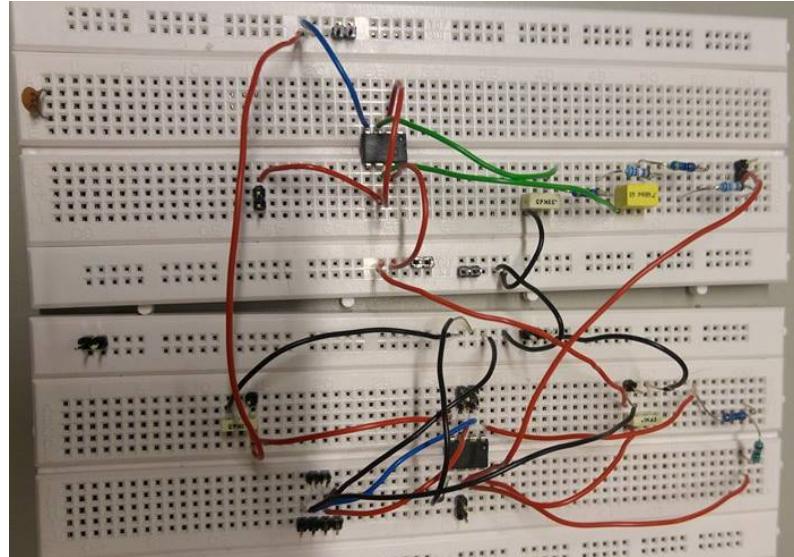
### 5.3 Integrationstest hardware

#### 5.3.1 Integrationstest for forstærker og analogt filter

Til integrationstesten af forstærkeren og det analoge filter blev Analog Discovery's signalgenerator koblet til indgangen på forstærkeren, som blev påtrykt et sinussignal. Filter og forstærker blev koblet sammen således output fra forstærkeren løb over i filterets input. Analog Discovery blev brugt som oscilloskop på det analoge filters udgang. Desuden fungerede Analog Discovery som spændingsforsyning for både forstærkeren og det analoge filter.

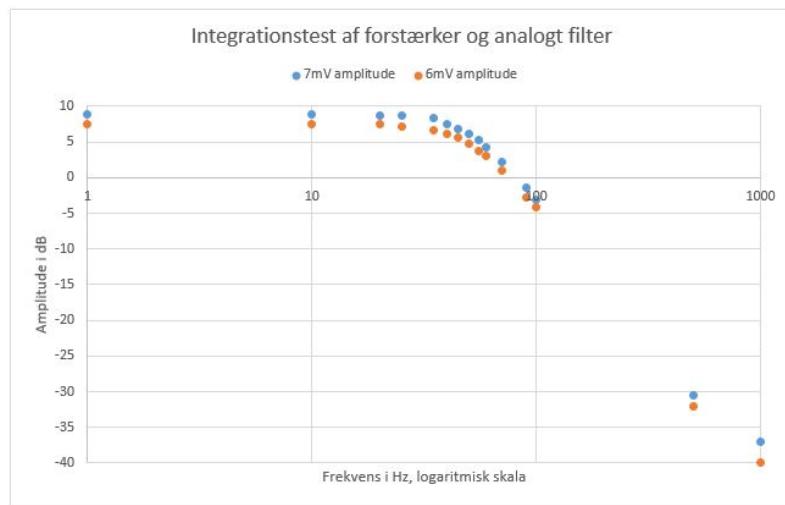
Der blev udført to målinger for hver udvalgte målefrekvens, hvor amplituden af sinussignalet var henholdsvis 6mV og 7mV. Der blev dermed foretaget to målinger for

hver af de 16 frekvenser, der lå i intervallet 1Hz til 1kHz, som ses i bilag 6.3. Opstillingen ses på figur 5.10:



Figur 5.10: Opstillingen der blev brugt til test af forstærker og analogt filter

Da projektet tager udgangspunkt i, at et maksimalt blodtryk vil have en amplitude på 6,25mV, er de to måle amplituder 6mV og 7mV blevet udvalgt for på den måde at kunne eftervise, at forstærkeren og filteret sammen får forstærket signalet op som det skal. Samtidig er de forskellige målefrekvenser udvalgt med henblik på at vise, at signalet, på trods af forskellige indgangsamplitude, har den samme knækfrekvens, hvilket man kan se ud af figur 5.11.



Figur 5.11: Integrationstest resultater ved test af forstærker og analogt filter

Som det kan ses ud af 5.11 viser amplitudekarakteristikken for signaler ved hver af de to forskellige amplituder bliver forstærket relativt lige meget dvs. de to amplitudekarakteristikker følges ad. Knækfrekvensen som med de pågældende komponentværdier er udregnet

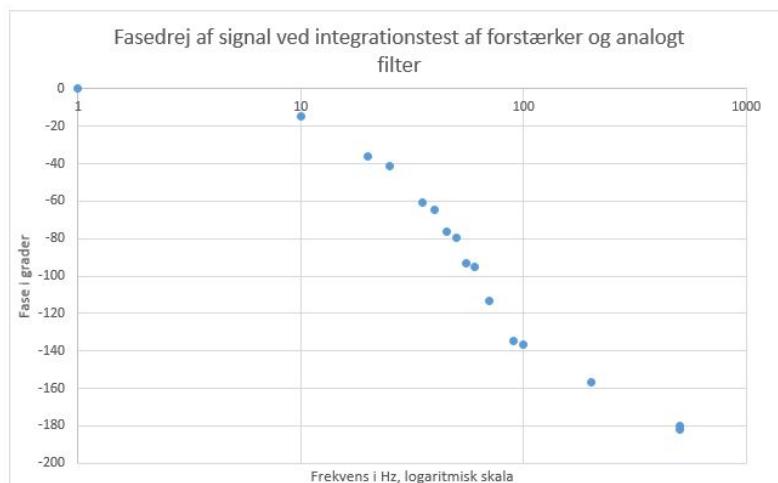
til værende 50,37Hz, se afsnit 4. Knækfrekvensen kan findes på grafen ved at aflæse 3dB frekvensen.

For signalet med amplitudekarakteristikken for 7mV bliver 3dB frekvensen dermed som følger i ligning 5.3:

$$3dB_{frekvens} = \frac{7mV}{\sqrt{2}} = 4,95mV \quad (5.3)$$

Ved aflæsning på grafen kan man ud for ca. 5mV se, at grafen på x-aksen kan aflæses til at ligge meget tæt på et målepunkt, der ligger i datasættet hedder (55Hz, 5,2dB), se bilag 6.3. Ud fra denne oplysning må det antages at den målte knækfrekvens ligger lige en anelse under 55Hz. Dog er den målte knækfrekvens for systemet højere end 50Hz da forstærkningen ved målingen for 50Hz er ca. 6dB.

Ud af grafen og det dertilhørende datasæt kan det også læses at forstærkningen falder med næsten 40dB/dekade efter knækfrekvensen med en afvigelse på 3,4dB.

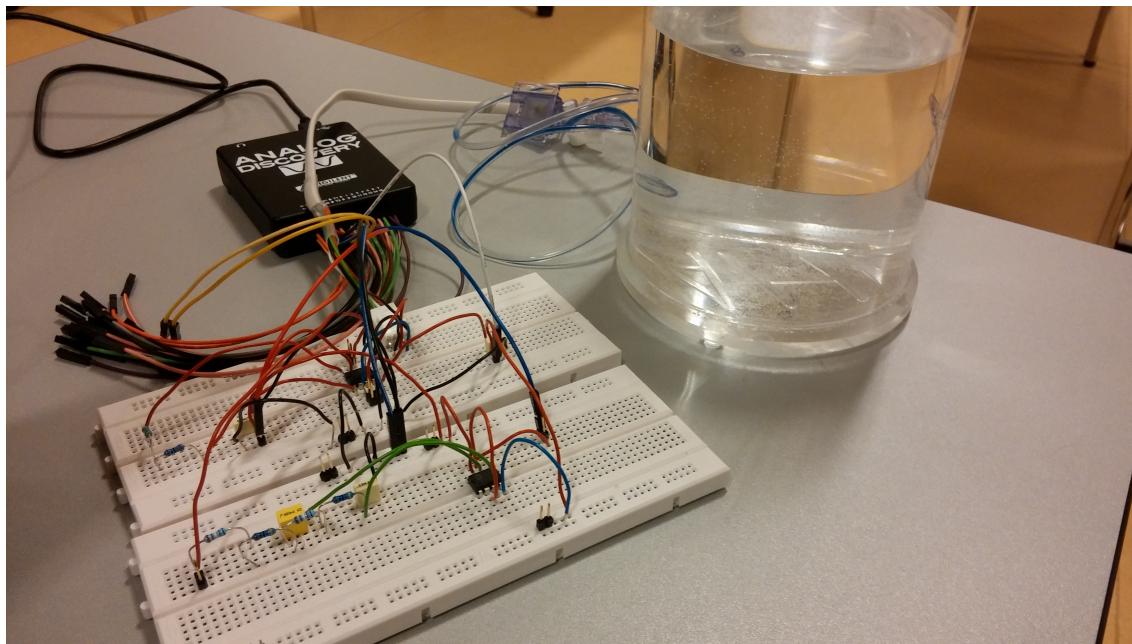


Figur 5.12: Fasedrej for signalet ved integrationstest af forstærker og analogt filter. Fasedrejningen af signalet er målt i grader i forhold til frekvens i Hz.

Ud af 5.12 ses det, at filteret og forstærkeren sammensat har et fasedrej på  $-90^\circ$  ved et sted mellem 50Hz og 55Hz. Desuden ses det, at faseforskydningen allerede begynder en dekade før knækfrekvensen og er nede i  $-180^\circ$  ved ca. en dekade over knækfrekvensen.

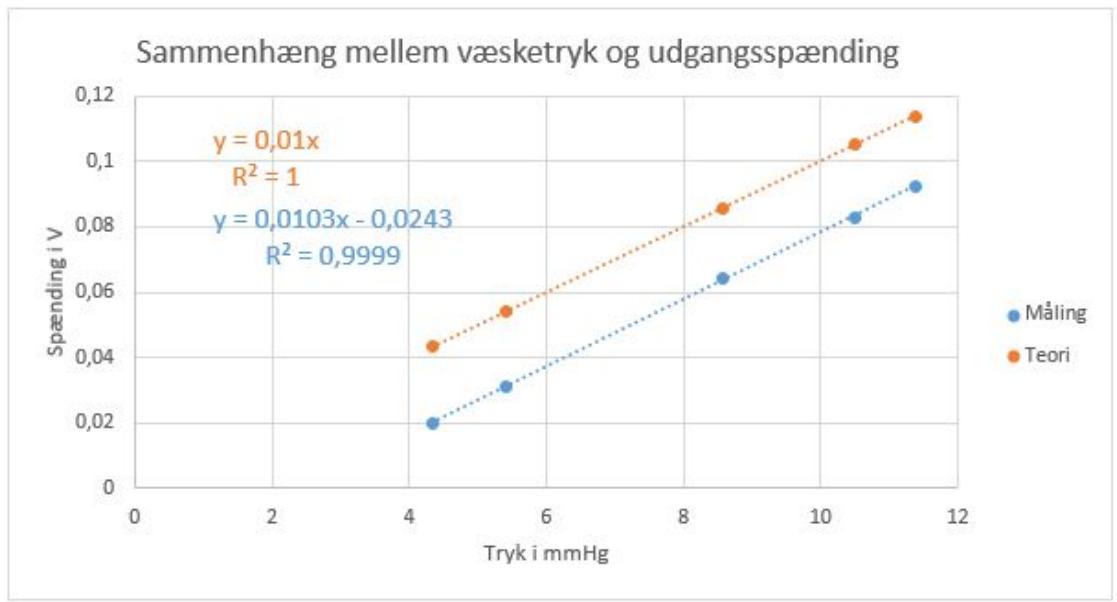
### 5.3.2 Integrationstest med vandsøjle

I integrationstesten med vandsøjlen blev hele hardwaren, dvs. transducer, forstærker og analogt filter, testet på en vandsøjle med et bestemt tryk. Transduceren blev koblet til vandsøjlen, som det ses af figur 5.13:



Figur 5.13: Opstillingen som blev brugt til integrationstest af hardwaredelen med vandsøjle

Ved det kendte vandsøjletryk i mmHg blev udgangsspændingen for det analoge filter noteret i bilag 6.3. Efter at have lavet lineær regression over de fem målepunkter blev tendenslinjens ligning sammenlignet med en tendenslinje, der var lavet på baggrund af fem teoretisk beregnede punkter.



Figur 5.14: Sammenhængen mellem væsketryk input på indgangen af forstærkeren og udgangsspændingen på filteret. Angivet ved henholdsvis teoretisk beregnet punkter og målte punkter.

Ud af figur 5.14 ses det, at de to tendenslinjer ligger næsten parallelt. Faktisk er den målte hældningskoefficient kun 0,0003 større, end det er tilfældet for den teoretisk beregnede

linje. Dvs. at de to linjer næsten er parallelle. Hvis de to linjer havde været fuldkommen parallelle, ville man med sikkerhed kunne sige, hvor stort væsketryk, der er i sjølen på baggrund af det spændingsoutput der er på udgangen af det analoge filter. I systemets tilfælde afviger hældningskoefficienten med 3% hvilket må siges at være acceptabelt grundet måleusikkerheder.

## 5.4 Modultest software

Til test af softwaren er der først og fremmest foretaget modul test af kodeelementer, for at teste om de fungerer efter hensigten. Dette er gjort ved hjælp af debug funktionen i Visual Studio. Disse test er foretaget løbende i udviklingsprocessen, hver gang en metode er tilføjet eller ændret. De væsenligste dele er nedenfor beskrevet:

Login funktionen er den første funktion, der testes ved sammenligne tilladte brugernavne og kodeord i databasen med indtastede. Dvs. at kun de brugernavne og kodeords par der findes i databasen, som giver adgang til systemet.

Nulpunktsjusteringen er testet ved at påsætte systemet en DC-spænding på 2V, hvorefter det indlæste tryk aflæses og sammenlignes med de 2V.

I forbindelse med test af nulpunktjusteringen, hvor der påsættes 2V DC-spænding, er kalibreringen også testet ved at indtaste kalibreringsfaktoren på 2. Herefter er tjekkes at signalet er forstærket 2 gange.

Testene foretaget i forbindelse med indlæsning af blodtrykssignal og detektering af systole, diastole og puls er alle foretaget ved følgende:

Der indsættes et kendt blodtrykssignal, med kendt antal toppe og dermed kendt systole, diastole og puls. De kendte værdier sammenlignes med værdierne der vises grafisk, samt med værdierne der findes ved debugging af systemet.

Alermen er tjekket ved at påsætte et blodtrykssignal med kendte blodtryksværdier: systolisk  $> 160$  eller  $< 100$  og diastolisk  $> 100$  eller under 40), hvorefter det er testet om alermen lyder ved de fire tilfælde uafhængigt af hinanden.

Yderligere for alermen er der justering af grænseværdierne testet, ved at ændre disse grænseværdier således ovenstående værdier ikke er kritiske, for herefter at tjekke om alermen lyder.

Ved alarm, er det testet at lyden kan udskydes med 3 minutter, som skrevet i koden, ved at lade blodtrykssignalet overskride grænseværdierne i mere end 3 minutter.

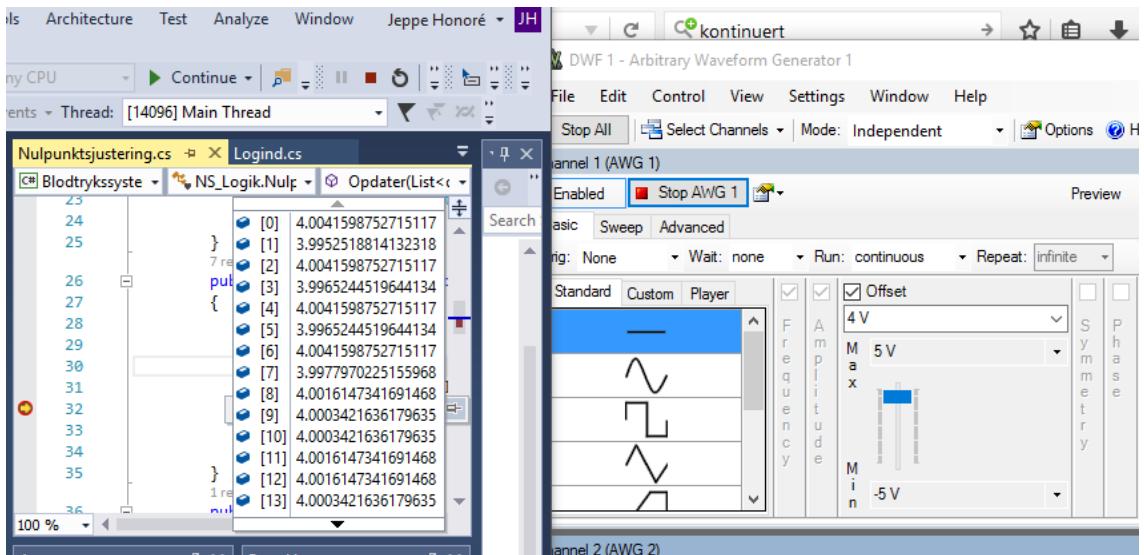
Gemmefunktionen er testet ved at sende et kendt blodtrykssignal igennem systemet, og tjekke om de data der gemmes i databasen stemmer overen med de kendte data. I samme forbindelse er CPR tjekker testet ved først at indskrive et gyldigt CPR nummer, som først tjekkes i CPR-tjekker metoden og dernæst i databasen. Derefter er samme procedure gentaget med et ikke gyldigt CPR-nummer.

## 5.5 Integrationstest software

I dette afsnit vil der beskrives hvordan systemets dele er blevet testet sammen. Dette fører videre fra unit testen, hvor de enkelte del elementer blev testes og nu vil disse del elementer

blive testet sammen som et helt system.

For at teste det samlede softwaresystem er der blevet lavet en integrationstest. Ved udførslen af integrationstesten genereres, der en kendt spænding vha. Analog Discovery, der bliver sendt igennem systemet og herefter bruges debug-funktionen i Visual Studio til at se om værdierne, der bliver indlæst af systemet stemmer overens med signalet der sendes ind. I denne test sendes der et signal igennem systemet på 4V. Det vil hermed sige at værdierne, der indlæses burde, hvis systemet virker efter hensigten, ligge tæt på 4.



Figur 5.15: Integrationstest software

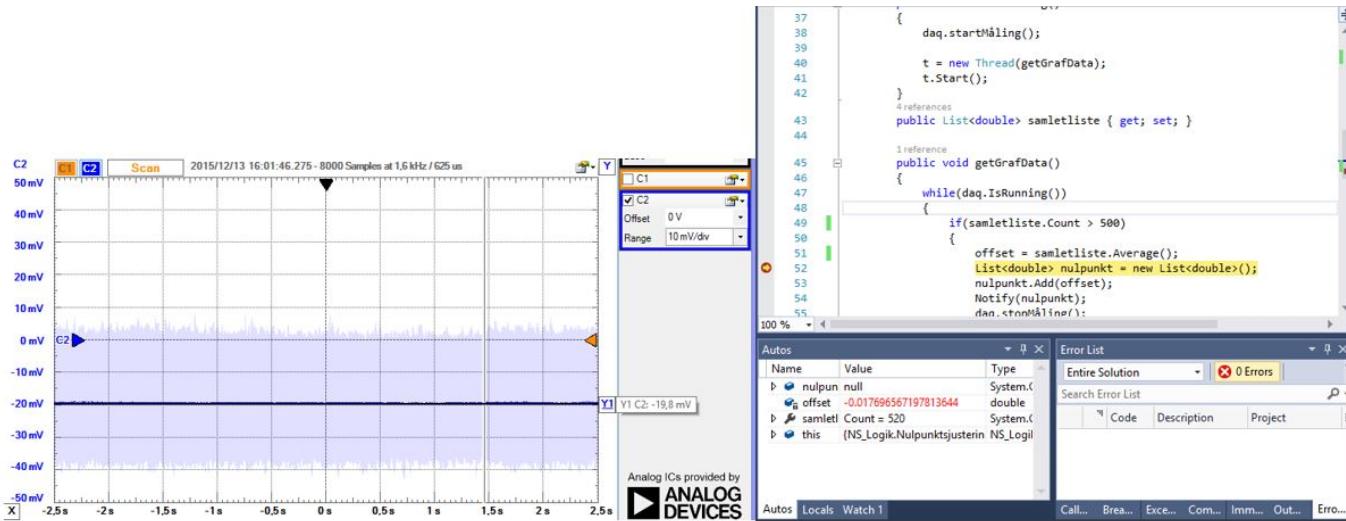
På figur 5.15 ses det, at de indlæste værdier ligger lige omkring 4. Det vil sige, at systemet indlæser værdierne efter hensigten.

## 5.6 Integrationstest system

I dette afsnit beskrives hvordan software-systemet testes sammen med hardware-systemet. Denne test forudsætter, at der er lavet integrationstest for både hardware og software hver for sig.

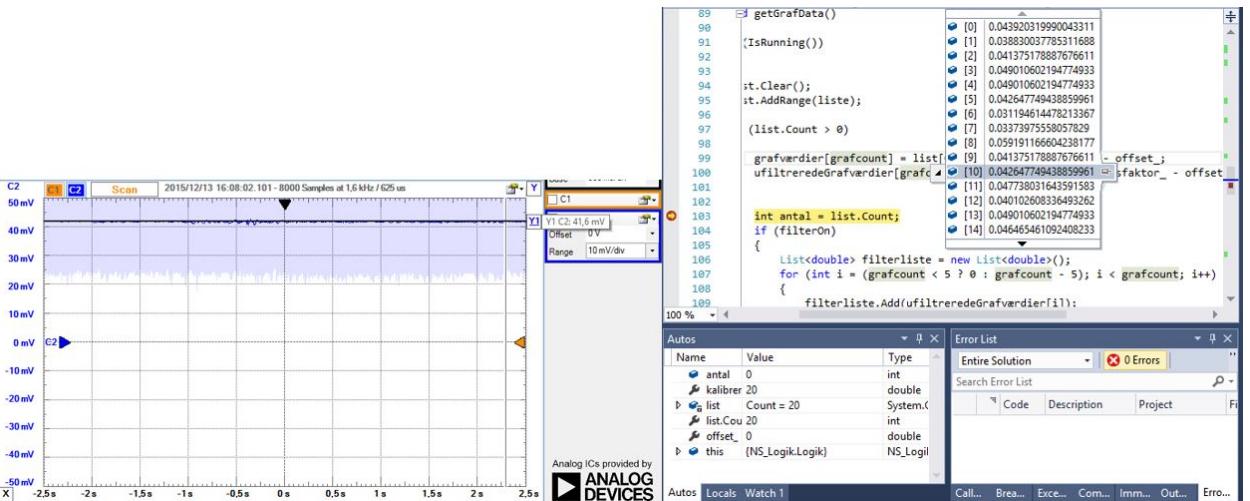
Denne integrationstest lægger op til accepttesten, som udføres med kunden. Det er derfor vigtigt at få de to systemer testet sammen, inden den endelige accepttest foretages.

Transduceren kobles til vandsøjlen, hvor højden fyldes til 9cm, dvs. et tryk på 6,61mmHg, se udregning i bilag 6.3. Transduceren kobles til hardwaren, og der måles på udgangene fra filtret vha. Analog Discovery. Udgangssignalet fra filtret sendes igennem DAQ'en og ind i computeren. Der sættes breakpoints i programmet ved værdien for nulpunktsjustering og værdien af signalet. Transduceren sættes til at måle atmosfærisk tryk, og der foretages en nulpunktsjustering i programmet. Værdien på Analog Discovery sammenlignes med værdien, der ses ved debugging i koden. Af figur 5.16 ses, at der kun er en forskel på 2,1mV, hvilket kan skyldes, at signalet svinger lidt, da trykket ikke er helt jævnt.



Figur 5.16: Værdien fra det atmosfæriske tryk målt med Analog Discovery til  $-19,8\text{mV}$  (til venstre) og med programmet til  $-0,0177\text{V}$  (til højre).

Transduceren sættes til at måle trykket fra vandsøjlen. Dernæst foretages en måling i programmet. I selve koden findes værdien, der måles og denne sammenlignes med målingen i Analog Discovery. Af figur 5.17 ses, at signalet, som programmet opfanger, svinger omkring værdien vi mäter med Analog Discovery. Dette kan skyldes, at vandsøjlens overflade står og vibrerer med rystelserne i bordet, hvorved der skabes støj i signalet.



Figur 5.17: Værdien fra væsketrykket målt med Analog Discovery til 41,6mV (til venstre) og med programmet i V (til højre).

Dernæst sammenlignes det kendte tryk med værdien, der vises i programmet. Her ses af figur 5.18, at værdien af mmHg i programmet ligger lidt under det tryk, der forventes i vandsøjlen. Dette kan skyldes måleusikkerheder fra målingen af vandsøjlens højde og ændringen af signalet undervejs i hardwaren og DAQ'en. Svingningerne af værdien i signalet skyldes stadig, at vandsøjlens overflade vibrerer når der er små vibrationer i bordet vandsøjlen står på.

The screenshot shows a debugger interface with several windows:

- Code Window:** Displays the C# code for a method named `Notify`. The cursor is positioned at the start of the `Notify` method.
- Watch Window:** Shows the current values of variables:
  - `filter`: `{NS_Logik.Filter}`
  - `filterOn`: `false`
  - `grafvæl`: `Count = 400`
  - `this`: `(NS_Logik.Logik)`
- Call Stack Window:** Shows the call stack with 31 entries, each containing a timestamp and a value.
- Error List Window:** Shows "0 Errors".

Figur 5.18: Værdien fra Væsketrykket målt med programmet i mmHg.

# Accepttest 6

---

Version	Dato	Ansvarlig	Beskrivelse
0.1	8/09 2015	Alle	Oprettelse af dokument
0.2	30/09 2015	Alle	Første udkast til accepttest
0.3	07/10 2015	Alle	Review rettelser
1.0	04/11 2015	Alle	Nyt udkast til accepttest, som følge af ændrede use cases
1.1	10/12 2015	Alle	Specifcierung af accepttest som følge af produktet
1.2	11/12 2015	Alle+vejleder	Godkendelse af accepttest

## 6.1 Indledning

I accepttesten testes de krav, der er opstillet i kravspecifikationen. Det fremgår af accepttesten, hvilke scenarier og krav, der er blevet opfyldt og implementeret i systemet, og hvilke, der enten ikke er testbare eller ikke er blevet implementeret.

Først vil der være accepttest af use casene, altså de funktionelle krav, efterfulgt af accepttest af de ikke-funktionelle krav.

Til test af use case 2-6 er der benyttet et signal fra PhysioNet. Signalet er et blodtrykssignal, som er vedlagt i bilag, se bilag 6.3.

## 6.2 Accepttest af Use Cases

### 6.2.1 Use Case 1

#### Kalibrer System

Test	Forventet resultat	Faktiske observationer	Godkendt
<i>Hovedscenarie</i>			

1. Forbind systemet til en kendt trykkilde	Systemet er forbundet til en kendt kilde	Systemet er forbundet til en kendt kilde	✓
2. Mål det atmosfæriske tryk	Det atmosfæriske tryk kan aflæses	Det atmosfæriske tryk kan aflæses	✓
3. Aflæs spændingen målt på udgangen af det analoge filter	At der forekommer en værdi, som efterfølgende kan bruges til lineærregression		✓
4. Skriv resultatet ned og gentag punkt 1 og 3. med to nye tryk	Der er tre værdier skrevet ned	Nye værdier kan noteres	✓
5. Foretag lineærregression	Afvigelseskoefficient er fundet	Afvigelseskoefficient noteres	✓
6. Indtast afvigelses koeficient i kalibrering panel i "Log ind"-vinduet	Værdien er synlig i tekstboksen	Værdien er synlig i tekstboksen	✓
7. Tryk på "Kalibrer"-knappen	Systemet er kalibreret	Systemet kalibreret	✓

Tabel 6.2: Accepttest af Use Case 1.

### 6.2.2 Use Case 2

#### Opstart system

Test	Forventet resultat	Faktiske observationer	Godkendt
<i>Hovedscenarie</i>			
1. Indstil transducer til at måle atmosfærisk tryk	Transduceren er instillet	Atmosfærisk tryk måles	✓

2.	Tryk på "Nulpunktsjuster"-knap	Panel vises med instruktioner om nulpunktsjustering	Panel vises med instruktioner	✓
3.	Tryk på "indlæs tryk"-knap	Et tryk vises tekstboksen	Tryk vises	✓
4.	Tryk på "Godkend"-knap	Panelet lukkes	Panelet lukkes	✓
5.	Indstil transducer til at måle blodtryk	Transducer er indstillet	Blodtryk måles	✓
6.	Indtast personale-ID i brugenavnsfeltet; "1234" og personlig kode i kodeordsfeltet; "fido"	Loginoplysninger blir udfyldt	De rigtige oplysninger vises	✓
7.	Tryk på "Log ind"-knappen	Log ind oplysninger er gyldige og stemmer overens med hinanden. "Blodtryks"-vinduet vises	Login vinduet lukkes og blodtryk vinduet vises	✓

---

*Undtagelser*

---

7.a	1. Indtast personale-ID i brugenavnsfeltet; "fido" og personlig kode i kodeordsfeltet; "1234"	Nye log ind oplysninger vises	Nye log ind oplysninger vises	✓
	2. Tryk "Log ind"	Besked om at de ikke findes vises	Besked vises	✓

*Tabel 6.3: Accepttest af Use Case 2.*

### 6.2.3 Use Case 3

#### Mål blodtryk

Test	Forventet resultat	Faktiske observationer	Godkendt
<i>Hovedscenarie</i>			
1. Påsæt signal fra PhysioNet	Signalet fungerer	Signalet fungerer	✓
2. Tryk på "start"-knappen i blodtryksvinduet	Graf og blodtryks værdier vises på brugergrænsefladen	Graf og værdier vises	✓

Tabel 6.4: Accepttest af Use Case 3.

### 6.2.4 Use Case 4

#### Filtrer signal

Test	Forventet resultat	Faktiske observationer	Godkendt
<i>Hovedscenarie</i>			
1. Påsæt realistisk signal fra fysionet	Signal vises i grafen	Signalet vises med støj	✓
2. Tryk på "Til"-knappen under filter	Signalet udglattes	Signalet udglattes	✓
3. Tryk på "Fra"-knappen under filter	Signal udglattes ikke	Signalet udglattes ikke	✓

Tabel 6.5: Accepttest af Use Case 4.

### 6.2.5 Use Case 5

#### Alarmer bruger

Test	Forventet resultat	Faktiske observationer	Godkendt
------	--------------------	------------------------	----------

*Hovedscenarie*

1. Indstil cursor til 100 for diastolisk øvre grænse	Der står 100 i pågældende tekst felt	Der står 100 i pågældende tekst felt	✓
2. Indstil cursor til 80 for diastolisk nedre grænse	Der står 80 i pågældende tekst felt	Der står 80 i pågældende tekst felt	✓
3. Indstil cursor til 100 for systolisk øvre grænse	Der står 100 i pågældende tekst felt	Der står 100 i pågældende tekst felt	✓
4. Indstil cursor til 80 for systolisk nedre grænse	Der står 80 i pågældende tekst felt	Der står 80 i pågældende tekst felt	✓
5. Tryk på 'Juster' knappen i blodtryks vindue	Blodtryksmåling startes og Alarm går igang	Blodtryksmåling startes og Alarm går igang	✓

*Undtagelser*

5a Tryk på "udskyd alarm"-knappen	Alarmen udskydes med 3 minutter	Alarmen udskydes med 3 minutter	✓
-----------------------------------	---------------------------------	---------------------------------	---

Tabel 6.6: Accepttest af Use Case 5.

**6.2.6 Use Case 6****Afslut system**

Test	Forventet resultat	Faktiske observatior- ner	Godkendt
------	--------------------	------------------------------	----------

*Hovedscenarie*

1. Tryk på "Afslut måling"-knappen	"Gemme"-vinduet vises	"Gemme"-vinduet vises	✓
2. Indtast CPR-nr "111111118"	CPR-nummeret synligt i pågældende tekst felt	CPR-nummeret synligt i pågældende tekst felt	✓

3.	Tryk på "Gem"-knappen	Bekræftigelse vises	Bekræftigelse vises	✓
3.	Tryk på "Ok"knappen	"Gemme"-vindue og "Blodtryks--vinduet lukkes. "Login"-vinduet vises	"Gemme"-vindue og "Blodtryks--vinduet lukkes. "Login"-vinduet vises	✓

*Undtagelser*

1.a.	Tryk på "annuller"-knap	"Gemme"-vinduet lukkes og "Blodtryk"-vinduet vises	"Gemme"-vinduet lukkes og "Blodtryk"-vinduet vises	✓
2.a.	Indtast CPR-nummeret "1234567890"	Beskeden "CPR ikke gyldigt." vises	Beskeden "CPR ikke gyldigt." vises	✓

Tabel 6.7: Accepttest af Use Case 7.

**6.3 Accepttest af ikke-funktionelle krav**

Ikke-funktionelt krav	Test/handling	Forventet resultat	Faktiske observationer	Godkendt
<i>Functionality</i>				
1. Brugeren skal kunne starte en ny maling indenfor 30 sekunder efter opstart	Start programmet, hvorefter en ny måling der vha. stopur måles opstarts-tiden	At programmet er opstartet og ny måling er igang efter 30 sekunder	15 sekunder	✓
2. Systemet skal kunne forstærke signalet fra transduceren 400 gange	Start systemet	At signalet er forstærket	Egentlige test ligger i integrationstest, se afsnit 5.2	(✓)

2. Systemet skal kunne forstærke signalet med det indbyggede analoge filter med en båndbredde på 50Hz
- Start systemet At signalet er forstærket Egentlige test ligger i integrationstest, se afsnit 5.2 (✓)
- 
3. Programmet skal kunne vise blodtrykssignalet kontinuert Tryk på "Start måling"-knap At blodtryks-signalet er vist kontinuert på brugergrænsefladen Blodtrykssignalet er vist kontinuert på brugergrænsefladen ✓
- 
4. Programmet skal programmeres i C# Start programmet At koden er i C# Programmet er i C# ✓
- 
5. Programmet skal kalibreres en gang årligt Systemet kalibreres Systemet er kalibreret inden for seneste år ✓
- 
6. Programmet bør kunne måle og afbillede puls Tryk på "Start måling" At pulsen er afbilledet på brugergrænseflade Pulsen af afbilledet Pulsen af afbilledet ✓

*Usability*

1. Blodtrykstallene der udskrives på brugergrænseflade er røde Tryk "Start måling"-knap At blodtrykstallene er røde Blodtrykstallene er røde ✓
- 
2. Pulsmålingen skal udskrives på brugergrænsefladen med grønne tal Tryk "Start måling"-knap At pulsen vises med grønne tal Pulsen vises med grønne tal ✓

3. Brugergrænseflade Opstart pro-  
lever op til figu- gram og log  
ren udarbejdet i ind  
design afsnittet,  
se afsnit 3.3.1
- At brugergræn-  
seflade inde-  
holder samtlige  
funktioner som  
på figuren
- Brugergrænseflade ✓  
indeholder  
samtlige funk-  
tioner som på  
figuren
- 

*Reliability*

1. Systemet skal kunne køre uden fejl i et år
- Start system op og vent et år
- At programmet efter et år kører fejlfrit
- 
2. Systemet skal have en "mean time to restore" på højest 24 timer
- Start systemet og herefter genstart, hvor der tages tid med et stopur
- At programmet er klar igen inden den for 24 timer
- Kan ikke testes
- 

*Performance*

1. Systemet bør kunne gemme data på 5 sekunder +/- 10%
- Tryk på "Gem og afslut"-knap og tag tid med et stopur
- At data er inden for 5 sekunder
- Afhænger af hvor stor måling er
- (✓)
- 

*Supportability*

1. Softwaren er opbygget af tre-lagsmodellen
- Kig i koden efter data-lag, logik-lag og GUI-lag
- At koden indeholder et data-lag, et logik-lag og et GUI-lag
- Koden indeholder et data-lag, et logik-lag og et GUI-lag
- ✓
- 

Tabel 6.8: Accepttest af Ikke-funktionelle krav

# **Bilagsliste**

---

## **Bilag 1 - Modul og integrationstest af hardware**

Excel dokument med diverse hardware udregninger og måledata til hardware modul og intergationsnlest.

## **Bilag 2 - Logbog**

Samlet dokument med logbøger, både for hardware-gruppen såvel som for software-gruppen.

## **Bilag 3 - Mødereférart**

Samlet dokument med mødereférater.

## **Bilag 4 - Kode**

Hele koden findes her som zip-fil.

## **Bilag 5 - Samarbejds aftale**

Gruppens samarbejds aftale.

## **Bilag 6 - Blodtrykssignal**

Det signal, som er benyttet til at teste software.

## **Bilag 7 - Hjemmesider**

Screenshots af hjemmesider anført i referencelisten i rapporten.

## **Bilag 8 - Scrum Pivotal Tracker**

Projektstyringsværktøj med oversigt over opgaver.

## **Bilag 9 - Tidsplan**

Oversigt over deadlines.

## **Bilag 10 - Udviklingsværktøjer**

Oversigt og beskrivelse af benyttede udviklingsværktøjer.

## **Bilag 11 - Hæmodynamik og hjertekarsygdomme**

Litteratur, pdf fra KVI undervisning skrevet af Peter Johansen.

## **Bilag 12 - Vejledning til dokumentation af semesterprojekter**

Litteratur, pdf fra IHA.

## **Bilag 13 - National Instruments kodeudsnit**

Kodeudsnit fra National Instruments benyttet til udvikling af software.

# Litteratur

---

- [1] OKAWA Electric Design, *Sallen-Key Low-pass Filter Design Tool*, URL: <http://sim.okawa-denshi.jp/en/0PseikiLowkeisan.htm> Version 2008
- [2] Wikibooks, *Usability for nerds*, URL: [https://en.wikibooks.org/wiki/Usability\\_for\\_Nerds](https://en.wikibooks.org/wiki/Usability_for_Nerds), version 16/09 2015
- [3] National Instruments, *National Instruments kodeudsnit*, Version 2015, Se bilag 13