

Remote Ischemic Conditioning

Implementeringsdokument

Simon Vammen Grønbæk
Karl-Johan Schmidt
Aarhus University
Aarhus School of Engineering
Efteråret 2015



Titel:

System Arkitektur

Projekt:

Remote Ischemic Conditioning

Godkendelse:

Karl-Johan Schmidt

Projektperiode:

Juli 2015 - December 2015

Projektgruppe:

15155

Simon Vammen Grønbæk

Deltagere:

Simon Vammen Grønbæk

Karl-Johan Schmidt

Peter Johansen

Vejledere:

Peter Johansen

Rolf Blauenfeldt

Projektudbyder:

Rolf Blauenfeldt

Oplagstal: 10

Sidetal: ??

Afsluttet 18-12-2014

Indholdsfortegnelse

Kapitel 1 Indledning	7
1.1 Formål	7
1.2 Projektreferencer	7
1.3 Læsevejledning og dokumentstruktur	7
1.4 Definitioner og forkortelser	7
Kapitel 2 Software	9
2.1 Klasse diagram	9
2.2 Namespace: GUI Laget	10
2.2.1 Klasse: Display	10
2.2.1.1 Metode: initDisplay()	10
2.2.1.2 Metode: clearAreaDisp()	10
2.2.1.3 Metode: initConditioning()	10
2.2.1.4 Metode: initOcclusion()	11
2.2.1.5 Metode: initSetup()	12
2.2.1.6 Metode: moveSquare()	13
2.2.1.7 Metode: updateConditioning()	13
2.2.1.8 Metode: updateOcclusion()	14
2.2.1.9 Metode: updateSetup()	14
2.2.1.10 Metode: getNoCycles()	14
2.2.1.11 Metode: setNoCycles()	14
2.2.1.12 Metode: updateTimeLeft()	15
2.2.1.13 Metode: updateNoOfCycles()	15
2.2.1.14 Metode: updateStopWatchTime()	15
2.2.2 Klasse: Buttons	15
2.2.2.1 Metode: readModeSwitch()	15
2.2.2.2 Metode: startStopConditioning()	15
2.2.2.3 Metode: btPressure()	15
2.2.2.4 Metode: startStopOcclusion()	16
2.2.2.5 Metode: changer()	16
2.2.2.6 Metode: selector()	16
2.3 Namespace: Logik laget	17
2.3.1 Klasse: BPalgorithm	17
2.3.1.1 Metode: calculateMap()	17
2.3.1.2 Metode: calculateSYS()	18
2.3.1.3 Metode: calculateDIA()	18
2.3.2 Klasse: DigitalFiltering	19

2.3.2.1	averagingZeroGroupDelay()	19
2.3.3	Klasse: Scenarios	19
2.3.3.1	Metode: bloodPressure()	19
2.3.4	Klasse: Timer	19
2.3.4.1	Metode: setTimeStamp()	19
2.3.4.2	Metode: getTimeStamp()	20
2.3.4.3	Metode: countdown()	20
2.3.4.4	Metode: stopWatch()	20
2.3.4.5	Metode: displayTimer()	20
2.3.4.6	Metode: getTimerStatus()	21
2.3.4.7	Metode: setTimerStatus()	21
2.3.4.8	Metode: timeToString()	21
2.3.5	Klasse: MemoryParser	21
2.3.5.1	Metode: getNoOfCycles()	22
2.3.5.2	Metode: setNoOfCycles()	22
2.3.5.3	Metode: getTimePerCycle()	22
2.3.5.4	Metode: setTimePerCycle()	22
2.3.5.5	Metode: writeToSDCard()	23
2.4	Namespace: Data laget	23
2.4.1	Klasse: PressureControl	23
2.4.1.1	Metode: runMotor()	23
2.4.1.2	Metode: runValve()	23
2.4.1.3	Metode: turnMotorOn()	23
2.4.1.4	Metode: turnMotorOff()	24
2.4.1.5	Metode: turnValveOn()	24
2.4.1.6	Metode: turnValveOff()	24
2.4.2	Klasse: ExternalMemory	24
2.4.2.1	Metode: initializeSDCard()	24
2.4.2.2	Metode: generateRandomNumber()	24
2.4.2.3	Metode: checkFilesSD()	24
2.4.2.4	Metode: createFileTemplate()	25
2.4.2.5	Metode: writeToSDCard()	25
2.4.3	Klasse: InternalMemory	25
2.4.3.1	Metode: writeToEEPROM()	25
2.4.3.2	Metode: readFromEEPROM()	26
2.4.4	Klasse: PressureSampling	26
2.4.4.1	Metode: getCuffPressure()	26
2.4.4.2	Metode: runningPeakDetect()	26
2.5	Namespace: Global	27
2.5.1	Klasse: Utilities	27
2.5.1.1	Metode: rawToMmHG()	27
2.5.1.2	Metode: mmHgToRaw()	27
2.5.2	Klasse: Konditioneringsapparat.pde (Main fil)	27
2.5.2.1	Metode: intCon_ISR(), intBT_ISR(), intOcc_ISR(), intCha_ISR() og intSel_ISR()	27
2.5.2.2	Metode: setup()	28

2.5.2.3 Metode: loop()	28
Kapitel 3 Filter design	29
3.1 Analoge filtre	29
3.2 Komponent udregninger	30
3.2.1 Anden ordens butterworth filter	30
3.2.2 Høj pas filter	30
3.2.3 Lav pas filter	31
3.2.4 Gain på manchet oscillationer	32
3.2.5 Gain på manchettryk signal	32
3.3 Praksis	33
3.4 Digital filtrering	37
Kapitel 4 Hardware	39
4.1 Schmetic	39
4.2 Knapper	39
4.3 Filter	41

1 | Indledning

Implementeringsdokument giver et overblik over hvordan både hardware og software er blevet implementeret i udviklingen af prototypen. Dokumentet indeholder beskrivelse af strukturen af software klasser og deres funktionalitet. For hardware delen er der beskrevet de forskellige hardware blokke og hvordan de er blevet implementeret for at kunne leve op til kravene stille i kravspecifikationen.

1.1 Formål

Dette dokument har til formål at give læseren et teknisk indblik i *Konditioneringsapparatets* funktionalitet og opbygning, samt skabe fuld forståelse for alle systemets under dele. Som en forlængelse af system arkitekturen, som beskrev for dette system skulle designes, beskriver dette dokument det færdig design og hvordan det har opnået sin funktionalitet

1.2 Projektreferencer

- Reference til kravspecifikation
- Reference til accepttest
- Reference til system arkitekturen
- Reference til software

1.3 Læsevejledning og dokumentstruktur

Da dette dokument er en del af udviklingsdokumentation, er det vigtigt at læse i sammenhæng med kravspecifikationen og systemarkitekturen. Undervejs i dokumentet vil der være referencer til kravspecifikationen, disse reference vil fortælle hvilke(n) krav den implementerede funktionalitet opfylder. Dette dokument skal også ses som en forklaring på software implementeringen, og derfor passer navne og overskrifter i software beskrivelse overens med navne på metoder og klasser i softwaren.

1.4 Definitioner og forkortelser

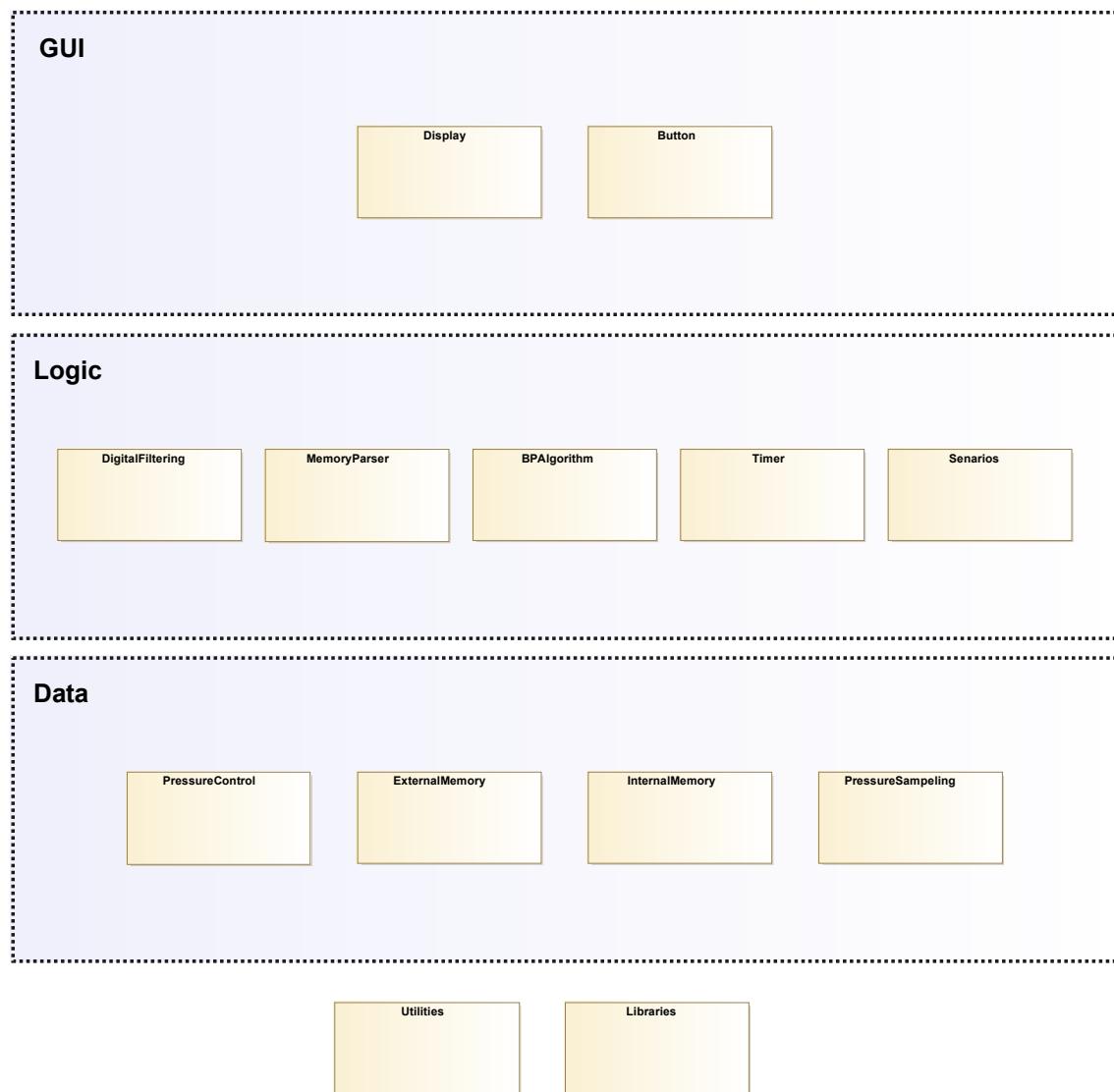
Udtryk / Forkortelse	Forklaring
Modeswitch	Knap til at styre hvilket program Konditioneringsapparatet skal køre

Tid pr cyklus	Variable som indeholder hvor mange sekunder et konditioneringscyklus skal vare
Antal cyklusser	Variable som indeholder hvor mange cyklusser et konditioneringsforløb skal vare

2 | Software

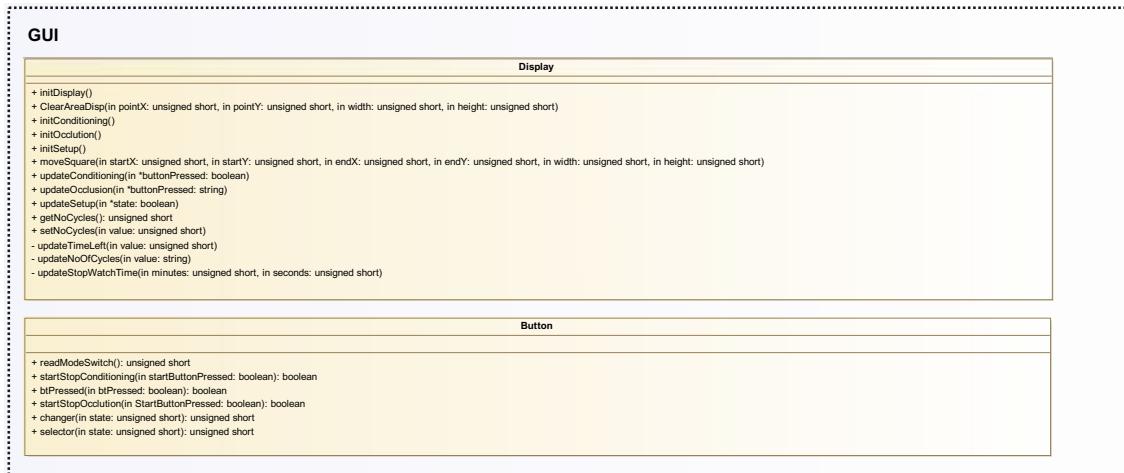
2.1 Klasse diagram

Oversigtsklassediagram, se figur 2.1, som fortæller strukturen af namespaces og klasse, men for overskueligheden er alle metoder undladt, se disse under deres respektive afsnit



Figur 2.1. Forsimplet klasse diagram

2.2 Namespace: GUI Laget



Figur 2.2. Klasse diagram over namespacet GUI

2.2.1 Klasse: Display

Denne klasse gør brug af to biblioteker for at kunne bruge TFT skærmen, henholdsvis Adafruit_GFX og Adafruit_ILI9340. For at kunne kommunikere med displayet gøre der brug af disse bibliotekters indbyggede funktioner. Derfor oprettes et objekt af klassen Adafruit_ILI9340 kaldet TFTscreen.

2.2.1.1 Metode: initDisplay()

Parameter: *void*

Returtype: *void*

Beskrivelse: Her initieres skærmen med funktion *.begin()*. Rotationen og baggrundsfarven af skærmen sættes også når denne metode kaldes. Skærmrotationen er sat 3, hvilken betyder at skærmen er i “landscape mode”.

2.2.1.2 Metode: clearAreaDisp()

Parameter: *unsigned short pointX, unsigned short pointY, unsigned short width, unsigned short height*

Returtype: *void*

Beskrivelse: Da skærmen baggrundsfarven er sat til sort, medtager denne metode 4 parameter hhv. start x-koordinat, start y-koordinat, bredde og højde. Disse parameter fortæller hvor og hvor en del af skærmen der skal farves sort, og dermed slette det område.

2.2.1.3 Metode: initConditioning()

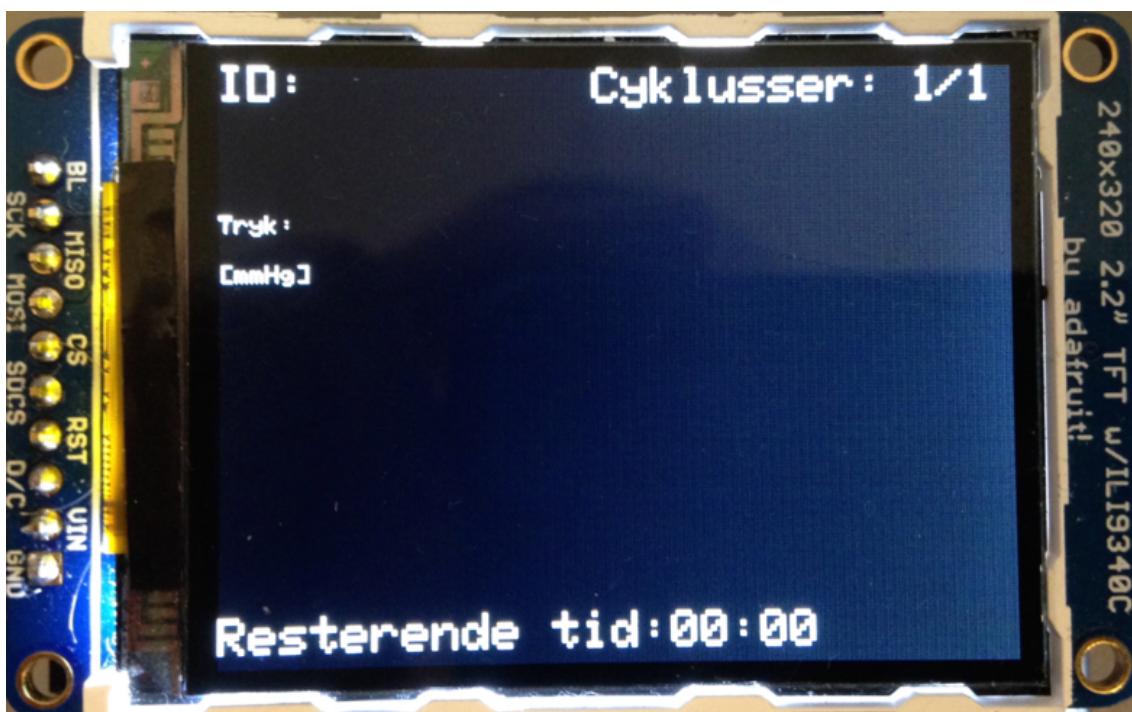
Parameter: *void*

Returtype: *void*

Beskrivelse: Når denne metode kaldes skrives de “faste” værdier på skærmen til et konditioningsforløb. På billedet nedenfor ses hvordan skærmen ser ud, når metoden er kørt. Et eksempel på hvordan der skrives tekst på skærmen:

```
1 TFTscreen.setTextColor(ILI9340_WHITE); TFTscreen.setTextSize(2);
2 TFTscreen.setCursor(0, 0);
3 TFTscreen.println("ID: ");
```

Først fortælles hvilken farve teksten skal have, dernæst tekststørrelse og placering. Til sidst angives hvilken tekst der skal printes på skærmen



Figur 2.3. Eksempel på layout når initConditioning() bliver kaldt

2.2.1.4 Metode: initOcclusion()

Parameter: *void*

Returtype: *void*

Beskrivelse: Denne metode bruges til at opsætte skærmen for okklusionstræningsforløb. Der skrives tid og enheden for tryk på skærmen. Se billedet nedenfor for layoutet.



Figur 2.4. Eksempel på layout når initOcclusion() bliver kaldt

2.2.1.5 Metode: initSetup()

Parameter: *void*

Returtype: *void*

Beskrivelse: Her opsættes skærmen for setup programmet. Teksten "Tid pr cyklus" og "Antal cyklusser" er faste værdi på skærmen. Men værdierne hentes fra logik laget, så de er opdateret.



Figur 2.5. Eksempel på layout når initSetup() bliver kaldt

2.2.1.6 Metode: moveSquare()

Parameter: *unsigned short startX, unsigned short startY, unsigned short endX, unsigned short endY, unsigned short width, unsigned short height*

Returtype: *void*

Beskrivelse: Denne metode bruges til at flytte cursoren på skærmen under setup. For at slette noget på skærmen skal det farves samme farve som baggrunden. Derfor får metoden x- og y-koordinaterne for den firkant der skal slettes, samt x- og y-koordinaterne for hvor den nye firkant skal tegnes henne. Desuden skal metode også have bredde og højde på firkanten. For at sikre at der ikke kan lavet et interrupt inde i metode, gør metode brug af den indbyggede funktion *noInterrupt()*. Et interrupt på det forkerte tidspunkt ville betyde at skærm ikke ville slette den forrige firkant eller ikke ville tegne det nye.

2.2.1.7 Metode: updateConditioning()

Parameter: *volatile bool *buttonPressed*

Returtype: *void*

Beskrivelse: Metoden modtager en pointer, som peger på værdien af *buttonPressed*. Hvis værdien af denne er sand, samt at antallet af kørt cyklusser ikke er lig med nul, vil metoden køre et loop, hvor der ved hjælp af timer klassen fra logik laget bliver starten en nedtælling, så loop vil køre indtil tiden er løbet ud. Mens loopet kører, opdaterer skærmen konstant værdien fra tryksensor og fortæller trykket i manchetten. Strukturen på metoden ser ud på følgende måde:

```

1   while(*buttonPressed && getNoCycleLeft() !=0){
2       if(!timer.getTimerStatus()){
3           //Running conditioning threatment
4       }else

```

```

5           //Reseting the timer
6       }
7   if(!*buttonPressed)
8       //Handle if the conditioning threatment has ended, and the user want to
      reset

```

Derfor er det værdien af *buttonPressed* og *getNoCycleLeft()* der afgører om loopet eksekveres. Hvis værdien af *buttonPressed* er falsk, så skrives det sidste værdi af timeren og sensorværdien slettes på skærmen.

buttonPressed styres ved knaptryk og bruger kan derfor starte og stoppe konditionerings forløbet på denne måde. Hvis forløbet stopper af sig selv, altså hvis antallet af tilbageværende cyklusser er nul, så håndtere metoden af brugeren ikke skal trykke to gange på knappen for at starte et nyt forløb.

```

1   if(memory.getNoOfCycles() == 0)
2       *buttonPressed = false;

```

2.2.1.8 Metode: updateOcclusion()

Parameter: *volatile bool *buttonPressed*

Returtype: *void*

Beskrivelse: Denne metode køres når der apparatet er sat på okklusions træningsforløb. Denne metode bruger også pointeren til *buttonPressed*, hvis den er true eksekveres et while loop hvor der startes et stopur og trykket fra manchetten vises på skærmen. Hvis værdien er falsk slettes sensorværdien på displayet, og slut tiden vises fra stopuret.

2.2.1.9 Metode: updateSetup()

Parameter: *volatile bool *state*

Returtype: *void*

Beskrivelse: Til styring af cursoren på displayet i setup. Denne metode består af en switch case struktur, som har fire cases og casevalg afgøres af værdien af **state*. Case 0 og 1 er gøre brug af metoden *moveSquare(..)* som flytter cursoren. Case 2 og 3 sørge for at vise værdien af hhv tid pr cyklus og antal cyklusser. Da cursoren styres med interrupt er interrupts slået fra så længe koden afvikles inde i case 2 og 3.

2.2.1.10 Metode: getNoCycles()

Parameter: *void*

Returtype: *unsigned short*

Beskrivelse: Bruges til at hente *antal cyklusser* fra logik laget. Denne værdi aflæses fra EEPROM via data laget, men for at overholde 3-lags modellen skal kommunikation gå via logik laget.

2.2.1.11 Metode: setNoCycles()

Parameter: *unsigned short value*

Returtype: *void*

Beskrivelse: Bruges til at sætte at værdien af *antal cyklusser*.

2.2.1.12 Metode: updateTimeLeft()

Parameter: *unsigned short value*

Returtype: *void*

Beskrivelse: Denne metode er lavet til opdate tiden under konditioningsforløbet. Da det kræver 5 metodekald af metoder fra biblioteket *Adafruit_ILI9340* for at skrive tekst på skærmen er dette indkapslet i én metode, som blot skal have en String value.

2.2.1.13 Metode: updateNumberOfCycles()

Parameter: *String value*

Returtype: *void*

Beskrivelse: Når metoden kaldes opdateres antallet cyklusser på skærmen.

2.2.1.14 Metode: updateStopWatchTime()

Parameter: *unsigned short minutes, unsigned short seconds*

Returtype: *void*

Beskrivelse: Denne metode opdatere tiden fra stopuret på displayet under okklusionstræningsforløbet

2.2.2 Klasse: Buttons

2.2.2.1 Metode: readModeSwitch()

Parameter: *void*

Returtype: *unsigned short*

Beskrivelse: Denne metode læser tre digitale pins hhv; 22, 24, og 26. Her vil én pin være høj og de to andre lave. Metoden afgøre ud fra hvilken pin der er høj, om der skal køres hhv; konditioningsforløb, okklusionstræning eller setup. Denne metode bliver kaldt i én gang når arduinoen startes op. Skal der ændres på hvilken forløb apparatet skal køre, skal arduinoen genstartes.

2.2.2.2 Metode: startStopConditioning()

Parameter: *volatile bool startButtonPressed*

Returtype: *bool*

Beskrivelse: Styring af værdien for *startButtonPressed*. Hver gang metode køres inverteres værdien af *startButtonPressed*. Desuden indeholder metoden et *if/else* statement, der hvis værdien af *startButtonPressed* er falsk sletter den gamle måling på skærmen og resetter antallet af kørte cyklusser. Hvis værdien er sand sættes et tidsstempel, så timeren passer når et nyt forløb startes.

2.2.2.3 Metode: btPressure()

Parameter: *volatile bool btPressed*

Returtype: *bool*

Beskrivelse: Styring af værdien for *btPressed*. Når denne metode kaldes inverteres værdien af *btPressed*. Hvis værdien er falsk, slettes den sidste måling på skærmen.

2.2.2.4 Metode: startStopOcclusion()

Parameter: *volatile bool startButtonPressed*

Returtype: *void*

Beskrivelse: Her invertes værdien af *startButtonPressed* og returneres. Hvis denne værdi er falsk kaldes en metode fra display klasse som fjerner en værdi på skærmen og der sættes et tidsstempel, fordi at værdien af *startButtonPressed* går fra falsk til sand og derfor skal der startes et okklusionstræningsforløb

2.2.2.5 Metode: changer()

Parameter: *volatile unsigned short state*

Returtype: *unsigned short*

Beskrivelse: Denne metode bruges til at styre cursoren når der skal ændre i antallet af cyklusser og tiden pr. cyklus. Den modtager værdien *state*, som kan være et tal mellem nul og tre. Hvis *state* nul ændres værdien til en og omvendt, det betyder at den skifter mellem at pege på *antal cyklusser* og *tid pr cyklus*. Hvis *state* har værdien to ændres der på *tid pr. cyklus* og hver gang metoden kaldes forøges værdien *tid pr. cyklus* med 30 sekunder. Desuden håndtere metoden at denne værdi kun kan ændres i intervallet mellem 180 til 480 sekunder, dvs 3 til 5 minutter. Hver gang værdien *at tid pr cyklus* ændres skrives den nye værdi til EEPROM via metoden *InternalMemory::setTimePerCycles()*. Når *state* er lige med 3, ændres værdien af antal cyklusser og denne værdi forøges med 1 cyklus hver gang knappen trykkes. Denne værdien kan ændres i intervallet mellem 1 og 5 cyklusser. Den nye værdi skrives til EEPROM.

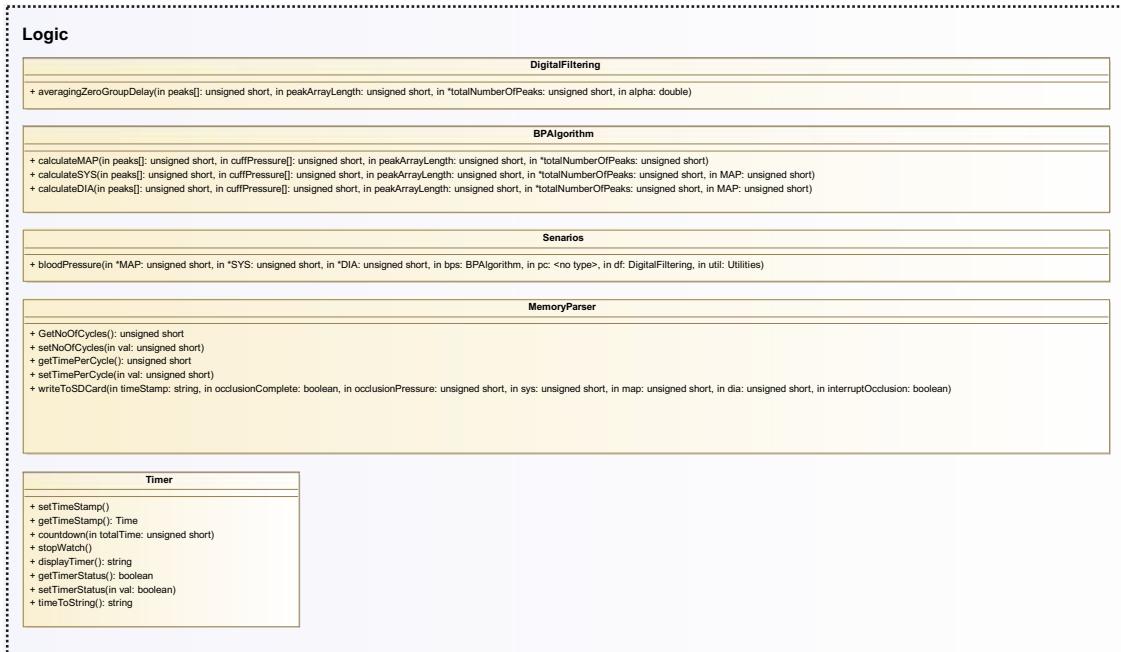
2.2.2.6 Metode: selector()

Parameter: *volatile unsigned short state*

Returtype: *unsigned short*

Beskrivelse: Metoden *selector()* ændres udelukkende på værdi af *state* og returnerer den nye værdi af *state*.

2.3 Namespace: Logik laget



Figur 2.6. Klasse diagram over namespacet Logic

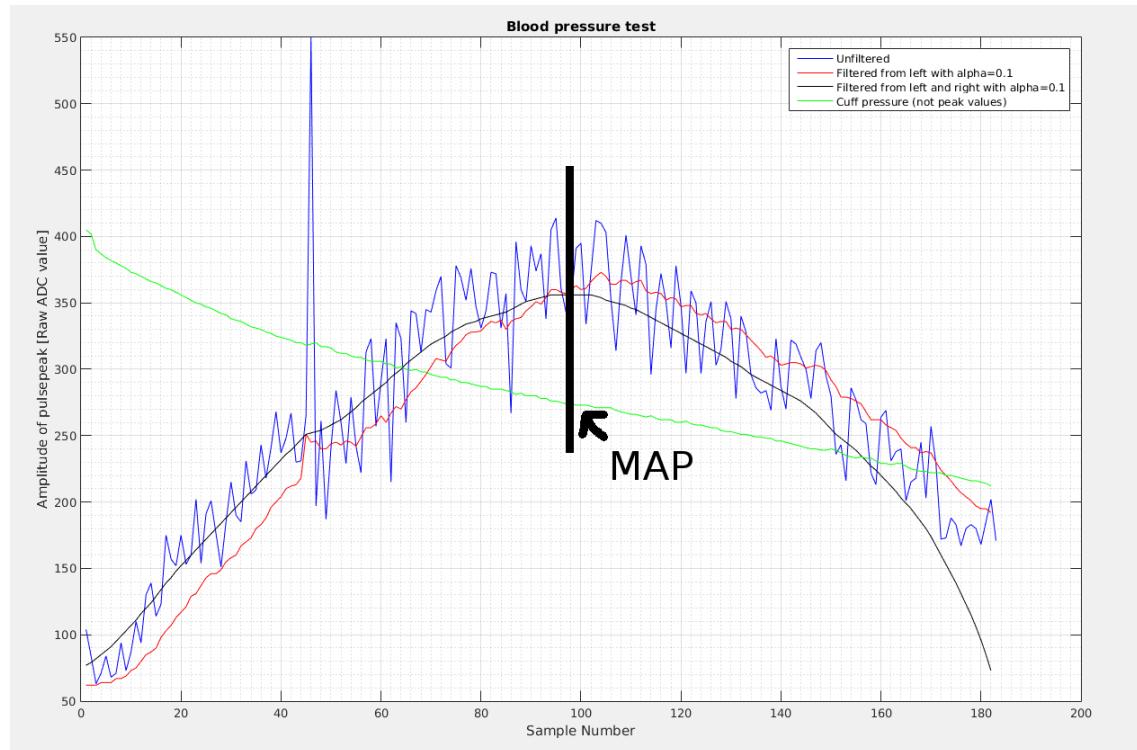
2.3.1 Klasse: BPAlgorithm

2.3.1.1 Metode: calculateMap()

Parameter: `unsigned short peaks[], unsigned short cuffPressure[], unsigned short peakArrayLength, unsigned short *totalNumberOfPeaks`

Returtype: `unsigned short`

Beskrivelse: Denne metode beregner MAP ud fra digitalt filtreret peakdata i `peaks[]` og `cuffPressure[]`. MAP findes som trykket i manchetten ved den højeste peak amplitude (se figur 2.7)



Figur 2.7. Graf med data fra en blodtryksmåling

Her ses at det rå signal er støjfyldt. Grøn er manchet trykket, blå er de rå peak amplituder, rød er filtreret en gang fra venstre mod højre, sort er filtreret fra begge sider.

2.3.1.2 Metode: calculateSYS()

Parameter: `unsigned short peaks[], unsigned short cuffPressure[], unsigned short peakArrayLength, unsigned short *totalNumberOfPeaks, unsigned short MAP`

Returtype: `unsigned short`

Beskrivelse: Denne metode beregner SYS ud fra MAP og digitalt filtreret peakdata i `peaks[]` og `cuffPressure[]`. SYS findes som trykket i manchetten ved peak amplityder på XX% af MAP.¹

2.3.1.3 Metode: calculateDIA()

Parameter: `unsigned short peaks[], unsigned short cuffPressure[], unsigned short peakArrayLength, unsigned short *totalNumberOfPeaks, unsigned short MAP`

Returtype: `unsigned short`

Beskrivelse: Denne metode beregner DIA ud fra MAP og digitalt filtreret peakdata i `peaks[]` og `cuffPressure[]`. DIA findes som trykket i manchetten ved peak amplityder på XX % af MAP².

¹Fixme Fatal: ret procentsats.

²Fixme Fatal: ret procentsats

2.3.2 Klasse: DigitalFiltering

2.3.2.1 Metode: averagingZeroGroupDelay()

Parameter: *nsigned short peaks[], unsigned short peakArrayLength, unsigned short *totalNumberOfPeaks, double alpha*

Returtype: *void*

Beskrivelse: Denne metode anvender eksponentiel midlgsfilter teknik uden group delay til at midle over parameteren peaks.

```

1     peaks[0] = startValue;
2     peaks[totalNOPeaks] = startValue;
3     for(i = 1;i<totalNOPeaks; i++)
4     {
5         peaks[i] = alpha*peaks[i]+(1-alpha)*peaks[i-1];
6     }
7
8     for(i = totalNOPeaks-1;i>0; i--)
9     {
10        peaks[i] = alpha*peaks[i]+(1-alpha)*peaks[i+1];
11    }

```

2.3.3 Klasse: Scenarios

2.3.3.1 Metode: bloodPressure()

Parameter: *unsigned short *MAP, unsigned short *SYS, unsigned short *DIA, BPAAlgorithm bpa, Data::PressureControl pc, Data::PressureSampling ps, Logic::DigitalFiltering df, Utilities util*

Returtype: *void*

Beskrivelse: Denne metode indeholder opskriften til en blodtryksmåling. Det vil sige kaldes denne metode udføres det en blodtryksmåling og alle andre klasser og metoder, som skal bruges til dette selv eksekveres inde i denne metode. Pointerne til de tre variabler får værdierne MAP, SYS og DIA fra blodtryksmålingen.

2.3.4 Klasse: Timer

Klassen timer gør brug af en hardware Real Time Clock(RTC) med IC'en *DS1302*. For at kommunikere med denne RTC, så gør klassen brug af et biblioteket *DS1302*. Derfor oprettet et objekt af klassen *DS1302.h* ved navn timestamp. Et Time objektet indeholder hhv: år, måned, dag, time, minut, sekund og ugedag.

2.3.4.1 Metode: setTimeStamp()

Parameter: *void*

Returtype: *void*

Beskrivelse: Denne metode aflæser den nuværende værdi af timeren og sætter en variable til denne værdi.

2.3.4.2 Metode: getTimeStamp()

Parameter: *Void*

Returtype: *Time*

Beskrivelse: Returnerer et Time objekt med værdien af timestamp.

2.3.4.3 Metode: countdown()

Parameter: *unsigned short totalTime*

Returtype: *void*

Beskrivelse: Denne metode modtager parameteren *totalTime*, som indeholder det ønskede antal sekunder nedtællingen skal være. Variablen *elapsedTime* indeholder den nuværende tid og timestamp indeholder et tidsstempel der bliver sat når timeren skal starte. Hver gang metoden køres trækkes den nuværende tid i hhv timer, minutter og sekunder fra hinanden. Dernæst omregnes de forskellige difference til samlede antal sekunder, hvorefter det tal omregnes til sekunder og minutter. For at få antal sekunder tage differencen mellem *totalTime* og *elapsedTotalSeconds* udregner modulus 60 til dette tal. Dette samme gøres for minutter blot hvor seconds også trækkes fra. Se kode nedenfor.

```

1   timerHasEnded = false;
2   Time elapsedTime = rtc.time();
3   String elapsedTimeString;
4   unsigned short hoursToSec = (elapsedTime.hr - timestamp.hr) * 24 * 60;
5   unsigned short minutesToSec = (elapsedTime.min - timestamp.min) * 60;
6   unsigned short elapsedTotalSeconds = hoursToSec + minutesToSec + (elapsedTime.sec
    - timestamp.sec);
7   seconds = (totalTime - elapsedTotalSeconds) % 60;
8   minutes = (totalTime - elapsedTotalSeconds - seconds)/60;
9
10  if(minutes == 0 && seconds == 0)
11      timerHasEnded = true;

```

Når det er regnes ud hvor mange minutter og sekunder der er tilbage i nedtællingen, gemmes det er de lokale variable minutes og seconds.

2.3.4.4 Metode: stopWatch()

Parameter: *void*

Returtype: *void*

Beskrivelse: Metode til at styre simulere og styre et stopur under okklusionstræning, den forløbne tid udregnes på samme måde *countdown()*, blot hvor det er tidsstemplet der trækkes fra den nuværende tid. Denne metode gemmer også minutter og sekunder i to lokale variabler, når udregning er den forløbne tid er færdig.

2.3.4.5 Metode: displayTimer()

Parameter: *void*

Returtype: *String*

Beskrivelse: Denne metode bruges til at konvertere tiden fra enten stopuret eller nedtællingen til formatet mm:ss. Desuden konverteres tiden til en string.

```
1   String minString = String(minutes, DEC);
```

```

2     String secString = String(seconds, DEC);
3     String timeString;
4
5     if(0 <=minutes && minutes < 10)
6         minString = String("0" + minString);
7     else
8         minString = String(minutes, DEC);
9
10    if(0 <=seconds && seconds < 10)
11        secString = String("0" + secString);
12    else
13        secString = String(seconds, DEC);
14    return timeString = String(minString + ":" + secString);

```

For at sikre at tiden vises på formatet mm:ss, tjekker metoden for om *seconds* eller *minutes* er større eller lig med 0 og mindre en 10. Hvis det er tilfældet tilføjes et nul foran værdien

2.3.4.6 Metode: getTimerStatus()

Parameter: *void*

Returtype: *bool*

Beskrivelse: Metode til at returnere værdien af statussen for timeren, hvis værdien er sand er timeren slut og omvendt hvis værdien er falsk kan timeren stadig være igangværende.

2.3.4.7 Metode: setTimerStatus()

Parameter: *bool val*

Returtype: *void*

Beskrivelse: Denne metode bruges til at sætte værdien af statussen for timeren, den sættes til true for at stoppe timeren. Derfor modtager en metode en parameter af typen bool.

2.3.4.8 Metode: timeToString()

Parameter: *void*

Returtype: *String*

Beskrivelse: For at kunne gemme tidsstemplar på SD kortet, er denne metode lavet til at konvertere et tidsstempel til en string på formatet: *tt:mm:ss DD-MM-YY*. Ligesom metoden *displayTimer()* tager denne metode også hånd hvilket enten timer, minutter eller sekunder er mindre end 10 og sætter et nul for and. Metoden returnerer en samlede string med et tidsstempel

2.3.5 Klasse: MemoryParser

Denne klasse er lavet for at overholde 3-lags modellen. Da informations læsning fra fx EEPROM og SD kort skal foregå i data laget, skal der en række metoder til at sende information igennem logik laget og videre til GUI laget. Derfor indeholder denne klasse som udgangspunkt kun get og set metoder.

2.3.5.1 Metode: getNumberOfCycles()

Parameter: *void*

Returtype: *unsigned short*

Beskrivelse: Denne metode returnere værdien fra data laget via metoden “*intMem.readFromEEPROM()*”.

2.3.5.2 Metode: setNumberOfCycles()

Parameter: *unsigned short val*

Returtype: *void*

Beskrivelse: Metode der skriver til data laget via metoden: *intMem.writeToEEPROM(200, val)*. Parameteren *val* videres til med denne metode.

2.3.5.3 Metode: getTimePerCycle()

Parameter: *void*

Returtype: *unsigned short*

Beskrivelse: Da værdien af *timePerCycle* indeholder hvor mange sekunder én konditioneringscyklus skal vare, er denne værdi ofte større end 255. Denne værdi skal læses fra EEPROM og en plads kan indeholde værdier på maks 255. Derfor sørger metoden for at hente værdien over 2 plads, hvis den er større en maks værdien.

```

1   unsigned short val = intMem.readFromEEPROM(205);
2   unsigned overloadVal = 0;
3   if(val == 255)
4       return overloadVal = val + intMem.readFromEEPROM(206);
5   else
6       return val;

```

2.3.5.4 Metode: setTimePerCycle()

Parameter: *unsigned short val*

Returtype: *void*

Beskrivelse: Beskrivelse: Når denne metode køres skrives tiden pr. cyklus til EEPROM, som forklaring i metoden *getTimePerCycle()*, skal den metode håndtere overload.

```

1   if(val > 255){
2       unsigned short rest = val % 255;
3       unsigned short valToFit = val - rest;
4       intMem.writeToEEPROM(205, valToFit);
5       intMem.writeToEEPROM(206, rest);
6   }
7   else
8       intMem.writeToEEPROM(205, val);

```

Det er forinden bestemt at adresserne 205 og 206 bruges til at gemme *TimePerCycle*. Metoden får en parameter som indeholde antal sekund en cyklus skal vare, og for sørge for værdien skrives korrekt til EEPROM udregnes modulus 255 af antallet af sekunder og den rest gemmes på adressen 206.

String timeStamp, boolean occlusionComplete, unsigned short occlusionPressure, unsigned short sys, unsigned short map, unsigned short dia, boolean interruptOcclusion

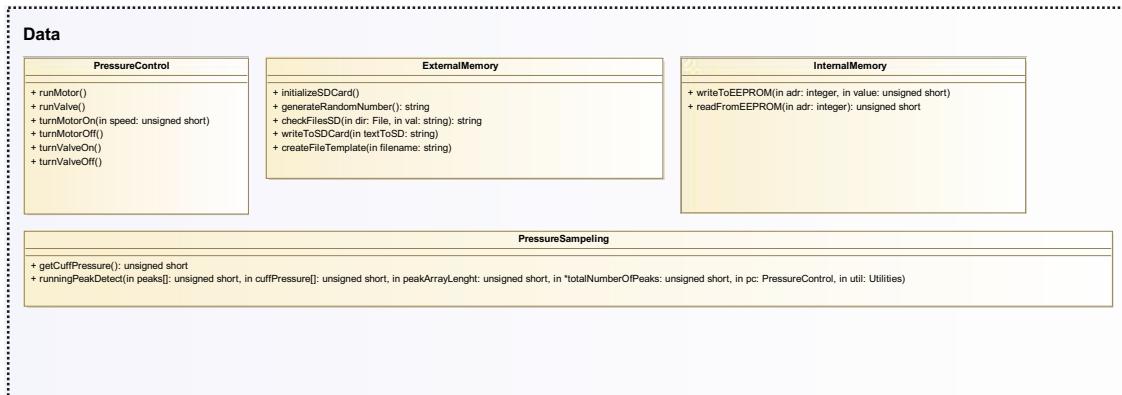
2.3.5.5 Metode: writeToSDCard()

Parameter: *String timeStamp, boolean occlusionComplete, unsigned short occlusionPressure, unsigned short sys, unsigned short map, unsigned short dia, boolean interruptOcclusion*

Returtype: *void*

Beskrivelse: Denne metode syv parametre af typerne *String, boolean og unsigned short*. Metoden konverterer og samler alle parametrene til en string og ved denne sendes videre til datalaget.

2.4 Namespace: Data laget



Figur 2.8. Klasse diagram over namespacet data

2.4.1 Klasse: PressureControl

2.4.1.1 Metode: runMotor()

Parameter: *void*

Returtype: *void*

Beskrivelse: Funktionen starter motoren og lader den kører ved fuld PWM ind til interrupt knappen på pin 18 ikke længere leverer en høj.

2.4.1.2 Metode: runValve()

Parameter: *void*

Returtype: *void*

Beskrivelse: Funktionen åbner for ventilen og lader den kører ved fuld PWM ind til interrupt knappen på pin 19 ikke længere leverer en høj.

2.4.1.3 Metode: turnMotorOn()

Parameter: *void*

Returtype: *void*

Beskrivelse: Funktionen tænder for motoren med den hastighed, angivet i parameteren (0-255).

2.4.1.4 Metode: turnMotorOff()

Parameter: *void*

Returtype: *void*

Beskrivelse: Funktionen stopper for motoren ved at sætte pin 3 lav

2.4.1.5 Metode: turnValveOn()

Parameter: *void*

Returtype: *void*

Beskrivelse: Funktionen åbner for ventilen ved at sætte pin 11 høj

2.4.1.6 Metode: turnValveOff()

Parameter: *void*

Returtype: *void*

Beskrivelse: Funktionen lukker for ventilen ved at sætte pin 11 lav

2.4.2 Klasse: ExternalMemory

Denne klasse gør brug af to arduino biblioteker hhv SD og SPI. De to biblioteker muliggøre kommunikation med SD kortet via en SPI forbindelse.

2.4.2.1 Metode: initializeSDCard()

Parameter: *void*

Returtype: *void*

Beskrivelse: Simple metode der starter kommunikation med SD kortet.

2.4.2.2 Metode: generateRandomNumber()

Parameter: *void*

Returtype: *String*

Beskrivelse: Denne metode genererer et 6-cifret tilfældigt nummer. Arduinos indbyggede funktion *random()* laver et tilfældigt nummer i det interval man specificerer, men den genererer dem i samme rækkefølge hver gang. For at gøre nummeret “mere tilfældigt” styres rækkefølgen af de genererede numre af værdien fra en analog port, som svæver.

```

1   randomSeed(analogRead(A5));
2   long randNumber = random(100000, 999999); /
3   String randNumberHEX = String(String(randNumber, HEX) + ".csv");
4   return randNumberHEX;

```

Når der er lavet et 6-cifre tilfældigt nummer, bliver dette konverteret til en HEX, så værdien nu indeholder tal mellem 0-9 og bogstaver mellem A-F. Denne værdi bliver returneret som en string.

2.4.2.3 Metode: checkFilesSD()

Parameter: *File dir, String val*

Returtype: *String*

Beskrivelse: Metode der kontrollere alle filer på SD kortet. Hver gang der findes en fil, tjekkes der for om de sidste 4 karaktere matcher med karaktererne “.csv”. Hvis den fundne fil matcher dette, bliver hele filnavnet gemt på en variable og returneret.

2.4.2.4 Metode: `createFileTemplate()`

Parameter: `String filename`

Returtype: `void`

Beskrivelse: Når der skal laves en ny fil på SD-kortet, skal der skrives en header til hver kolonne i .csv filen. Denne metode modtager et filnavn, som den åbner og skriver en header til.

2.4.2.5 Metode: `writeToSDCard()`

Parameter: `String textToSD`

Returtype: `void`

Beskrivelse: Denne metode gør brug af de tre forrige metoder; `generateRandomNumber()`, `checkFilesSD()` og `createFileTemplate()`. Først oprettes et objekt af typen file og ved hjælp af funktion `SD.open("/")` og “.csv” vil `checkFilesSD` nu tjekke alle filer på SD kortet og se som der findes en .csv fil. Hvis metode finder en fil, konverteres det fundne filnavn til et char array og filen åbnes. Nu skrives `textToSD` til SD kortet og filen lukkes igen.

```

1   File file;
2   File root = SD.open("/");
3   String nameReadFromSD = checkFilesSD(root, ".csv");
4   char bufName[nameReadFromSD.length()+1];
5   if(!nameReadFromSD.equalsIgnoreCase("empty")){
6       nameReadFromSD.toCharArray(bufName,
7           nameReadFromSD.length());
8       file = SD.open(bufName, FILE_WRITE);
9       file.println(textToSD);
10      file.close();
11  } else{ //Create new file with the random ID
12      createFileTemplate(generateRandomNumber());
13  }

```

Hvis metoden `checkFilesSD()` returnere en string med værdien “empty” kaldes metoderne `createFileTemplate()` og `generateRandomNumber()` og der laves en ny .csv fil med den korrekte header.

2.4.3 Klasse: InternalMemory

Klassen gør brug af biblioteket `EEPROM`, dette bibliotek gør kommunikation muligt med EEPROMen.

2.4.3.1 Metode: `writeToEEPROM()`

Parameter: `int adr, unsigned short value`

Returtype: `void`

Beskrivelse: Simple metode der skriver `value` til adressen `adr` på EEPROM.

2.4.3.2 Metode: readFromEEPROM()

Parameter: *int adr*

Returtype: *unsigned short*

Beskrivelse: Metode der læser værdien på adressen *adr* på EEPROM.

2.4.4 Klasse: PressureSampling

2.4.4.1 Metode: getCuffPressure()

Parameter: *void*

Returtype: *unsigned short*

Beskrivelse: Returnerer det aktuelle tryk i manchetten ved at sample 10 gange og tage middelværdien

2.4.4.2 Metode: runningPeakDetect()

Parameter: *unsigned short peaks[], unsigned short cuffPressure[], unsigned short peakArrayLength, unsigned short *totalNumberOfPeaks, PressureControl pc, Utilities util*

Returtype: *void*

Beskrivelse: Denne metode anvender en pointer til et array med peak værdier, et array med manchettryk værdier, en variabel med værdien tilsvarende længden af de to arrays, samt en pointer til en variabel hvor metoden skriver hvor mange peaks, som der er blevet fundet. De sidste parametre er bare objekter af de klasser som indeholder metoder der skal bruges i runningPeakDetect(). Metoden sampler et sample ad gangen (i alt 13) og tjekker om værdien er højere end middelværdien af de 6 samples før og de 6 samples efter. Hvis sample x(n-6) er størst og er højere end tresholdværdien for et detekteret puls signal, så gemmes peak værdien i "peaks[]" og manchettrykket i "cuffPressure[]".

```

1   if(timestamp<millis()-400 && n6>threshold && (n12+n11+n10+n9+n8+n7)/6 < n6 &&
2       (n+n1+n2+n3+n4+n5)/6 < n6)
3   {
4       peaks[tN0Peaks] = n6;
5       cuffPressure[tN0Peaks] = getCuffPressure();
6       tN0Peaks++;
7   }

```

If sætningen sikre også at peaks har mindst 400ms i afstand. Ved at have minimum afstand mellem detekteret pulssignal sikres at flere peaks på en puls ikke detekteres. Metoden fortsætter med at sample ind til manchet trykket når under 40mmHg, hvorefter den stopper.

```

1   while(currentPressure > util.mmHgToRaw(40))
2   {

```

Fordi de to arrays, som indeholder henholdsvis peaks og cuffpressure er prealokeret i hukommelsen, sættes de resterende ubrugte pladser = NULL. Dette sikrer at de værdier, som ellers måtte være liggende i hukommelsen ikke bliver forvekslet med en peak amplitude.

```

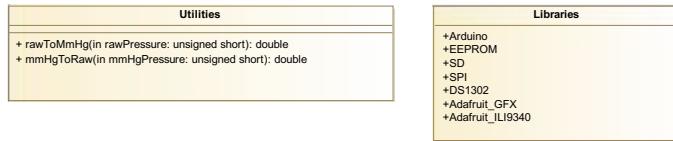
1   for(i = tN0Peaks; i < peakArrayLength; i++)
2   {
3       peaks[i] = NULL;
4       cuffPressure[i] = NULL;

```

5 }

Til sidst lukkes den restererythende luft ud af manchetten, altså de sidste 40mmHg.

2.5 Namespace: Global



Figur 2.9. Klasse diagram over det globale namespace

2.5.1 Klasse: Utilities

Denne klasse tilhører det globale namespace og kan tilgås af alle namespaces. Klassen skal ses som en hjælpe klasse med funktion der kan være brugbare flere steder

2.5.1.1 Metode: rawToMmHG()

Parameter: *unsigned short rawPressure*

Returtype: *double*

Beskrivelse: Metoden konverterer rå ADC værdi til mmHg.

2.5.1.2 Metode: mmHgToRaw()

Parameter: *unsigned short mmHgPressure*

Returtype: *double*

Beskrivelse: Metoden konverterer mmHg værdi til rå ACD værdi.

2.5.2 Klasse: Konditioneringsapparat.pde (Main fil)

Denne klasse er softwaren main fil, her samles funktionaliteten i to metoder: *setup()* og *loop()*, samt 5 interrupt service rutiner. Dette er begge metoder som arduino skal have. *setup()* bliver kørt som først når arduinoen startes og derefter køres det uendeligt *loop()*. Klassen kender kun til de to klasser i GUI laget, Buttons og Display Da interrupt ikke kan sættes fra andre steder end main filen, bliver der initieret 5 interrupt i denne klasse, hhv to til konditioneringsforløbet, en til okklusionstræning og to til setup.

2.5.2.1 Metode: intCon_ISR(), intBT_ISR(), intOcc_ISR(), intCha_ISR() og intSel_ISR()

Parameter: *void*

Returtype: *void*

Beskrivelse: Disse fem interrupt service rutiner kalder alle sammen deres respektive metode fra klassen Buttons. Alle metoder indeholder et delay på 100 ms og derefter et if statement, som tjekker om interrupt pin'en stadig er høj, dette software imødekommer debouncing, se afsnit 4.2

2.5.2.2 Metode: setup()

Parameter: *void*

Returtype: *void*

Beskrivelse: Denne metode bruges til opsætning af arduino, og det er derfor også her at det afgøres om arduinoen skal køre hhv konditioneringsforløb, okklusionstræning eller setup. Derfor aflæses værdien af 3 digital porte for at afgøre hvilket program der skal køres. Derefter opsættes de respektive interrupt for det valgt program. Dette er nødvendigt da prototypen kun har 2 knapper og disse skifter funktion efter hvilket program der er valgt

```

1   programToRun = btt.readModeSwitch();

2

3   //Setup for i interrupts
4   pinMode(interruptPin0, INPUT);
5   pinMode(interruptPin1, INPUT);

6

7   switch(btt.readModeSwitch()){
8     case 1: //Conditioning
9       attachInterrupt(digitalPinToInterruption(interruptPin0), intCon_ISR, RISING);
10      attachInterrupt(digitalPinToInterruption(interruptPin1), intBT_ISR, RISING);
11      break;
12    case 2: //Occlusion
13      attachInterrupt(digitalPinToInterruption(interruptPin0), intOcc_ISR, RISING);
14      break;
15    case 3: //Setup
16      attachInterrupt(digitalPinToInterruption(interruptPin0), intCha_ISR, RISING);
17      attachInterrupt(digitalPinToInterruption(interruptPin1), intSel_ISR, RISING);
18      break;
19  }

```

2.5.2.3 Metode: loop()

Parameter: *void*

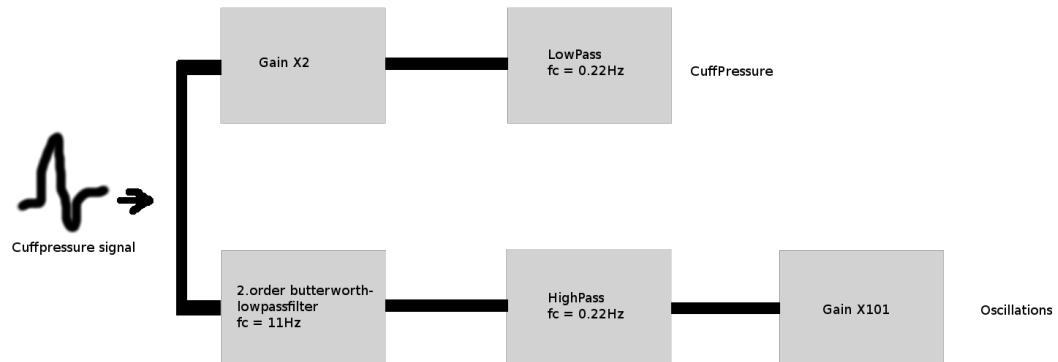
Returtype: *void*

Beskrivelse: Uændeligt loop der eksekveres efter metoden setup() har kørt. Metoden indeholder en switch case struktur, og program valget afgøre hvilken case der køres. Der kan kun skiftes case, hvis arduinoen genstartes.

3 | Filter design

3.1 Analoge filtre

Det analoge filter design har blandt andet funktionen at isolerer og forstærke DC niveauet fra tryksensoren, som er koblet til manchetten. Tryksensoren (MPX5100) er lineær og kan derfor ud fra en enkel koefficient kalibreres. 1 DC niveauet skal forstærkes for at øge ADC opløsningen i forhold til mmHg per bit. Arduino MEGA 2560 har en ADC opløsning på 10bit. I volt er dette en opløsning på $5/1023=4.9\text{mV}$. Sensoren har en sensitivitet på $45\text{mV}/\text{kPa}$, svarende til $6\text{mV}/\text{mmHg}$. Uden DC forstærkning er opløsningen altså $4.9/6=0.817\text{mmHg}$. Ved at anvende gain X2 øges opløsningen til $0.817/2=0.408\text{mmHg}$. Ren DC opnås ved at lavpas filteret fjerner alt AC over knækfrekvensen.



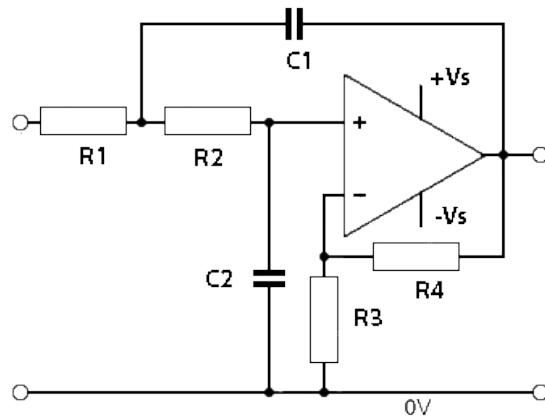
Figur 3.1. Filter model med inputsignal til venstre og de to output signaler til højre

CuffPressure er forstærket DC og Oscillations er forstærket AC mellem 0.22Hz og 11Hz

Oscillationerne i manchetten isoleres med 11Hz anden ordens butterworth for at opnå god dæmpning på 50Hz brummen. L.A. Geddes et al. Sætter knæk frekvensen til 30Hz², men efter som at puls signalet befinner sig på ca 1Hz og at Geddes med høj sandsynlighed har anvendt kaskade filtre sættes butterworth filterets knæk til 11Hz, svarende til elve afledte af puls grund frekvensen. Højpasfilteret fjerner DC ved at knække ved 0.22Hz. For at forstærke oscillationerne op i en størrelse, som arduinoen kan sample forstærkes det pulserende signal med gain X101.

3.2 Komponent udregninger

3.2.1 Anden ordens butterworth filter



Figur 3.2. 2. ordens butterworth filter

$$s^2 + 1.414s + 1 = 0 \quad (3.1)$$

$$\zeta = \frac{1.414}{2} = 0.707 \quad (3.2)$$

Der bruges en anden metode til at bestemme R og C_1 og C_2 . Det bestemmes at $R = 100k\Omega$, for at mindske spændingsdelingen mellem kredsløbet og spændingskilden.

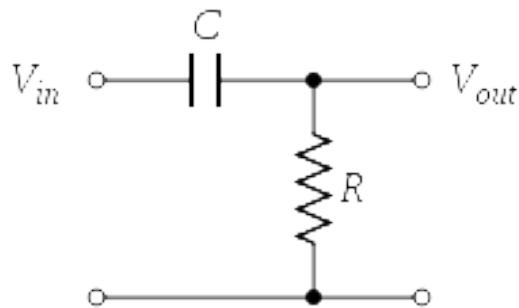
Den ønskede knækkfrekvens er $f_c = 11Hz$ og efter som $f_c = \frac{\omega_c}{2\pi}$ så er $\omega_c = f_c * 2 * \pi = 22\pi$

Nu løses de to ligninger med de to ubekendte $R\sqrt{C_1 * C_2} = \frac{1}{\omega_0}$ og $\frac{C_2}{C_1} = \zeta^2$ og C_1 og C_2 bliver:

$$C_1 = 2.046 * 10^{-7} \text{ eller } C_1 = -2.046 * 10^{-7}$$

$$C_2 = 1.023 * 10^{-7} \text{ eller } C_2 = -1.023 * 10^{-7}$$

3.2.2 Høj pas filter



Figur 3.3. 1. ordens høj pas filter

Knæk frekvens $f_c = 0.3$

Modstand værdi, $R = 1M\Omega$

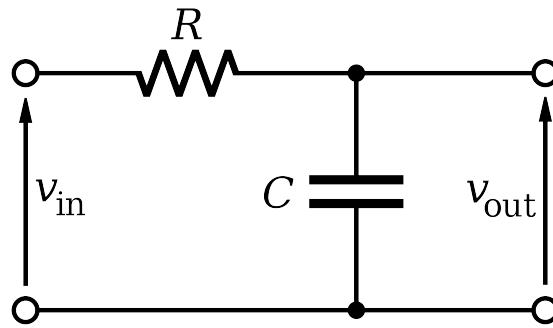
Nu udregnes værdien af kondensatoren ved at isolere C:

$$f_c = \frac{1}{2\pi R C} \Rightarrow C = 5.3052 * 10^{-7}$$

Dette bliver en kondensator på 530nF, den nærmeste komponent hedder 680nF og derfor udregnes den nye knæk frekvens til:

$$f_c = \frac{1}{2\pi R * 680 * 10^{-9}}$$

3.2.3 Lav pas filter



Figur 3.4. 1. ordens lav pas filter

Knæk frekvens $f_c = 0.3$

Modstand værdi, $R = 1M\Omega$

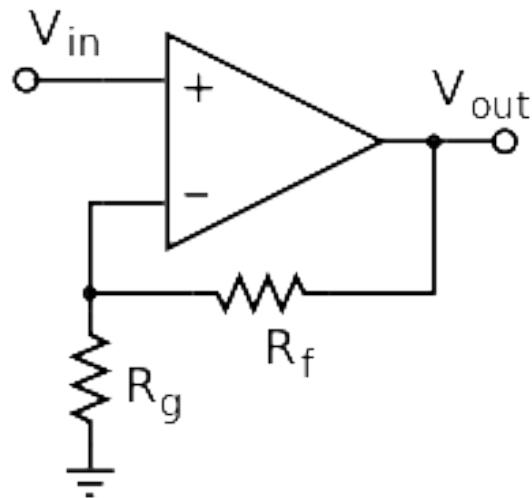
Nu udregnes værdien af kondensatoren ved at isolere C:

$$f_c = \frac{1}{2\pi R C} \Rightarrow C = 5.3052 * 10^{-7}$$

Dette bliver en kondensator på 530nF, den nærmeste komponent hedder 680nF og derfor udregnes den nye knæk frekvens til:

$$f_c = \frac{1}{2\pi R * 680 * 10^{-9}}$$

3.2.4 Gain på manchet oscillationer



Figur 3.5. Gain til manchet oscillationer

Udregning af gain:

Der ønskes et gain på 2, dvs $A = 100$

$$A = \frac{V_0}{V_i} = 1 + \frac{R_f}{R_g}$$

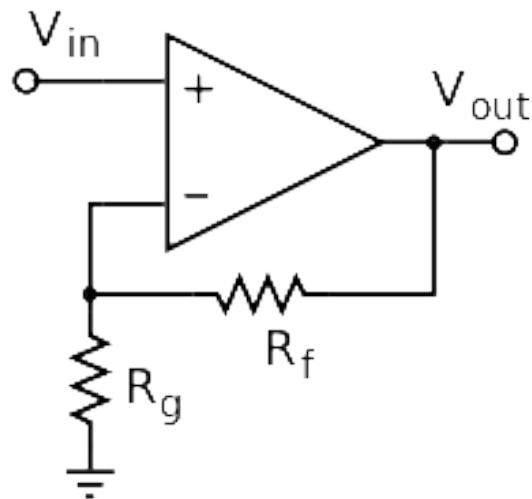
Udregning af komponenter:

Indsættelse af komponent værdier, R_f sættes til $100k\Omega$ og R_g udregnes:

$$A = 1 + \frac{R_f}{R_g} = 1010$$

Derfor vælger R_g til at være $1k\Omega$

3.2.5 Gain på manchettryk signal



Figur 3.6. Gain til manchettryk signal

Udregning af gain:

Der ønskes et gain på 100, dvs $A = 2$

$$A = \frac{V_o}{V_i} = 1 + \frac{R_f}{R_g}$$

Udregning af komponenter:

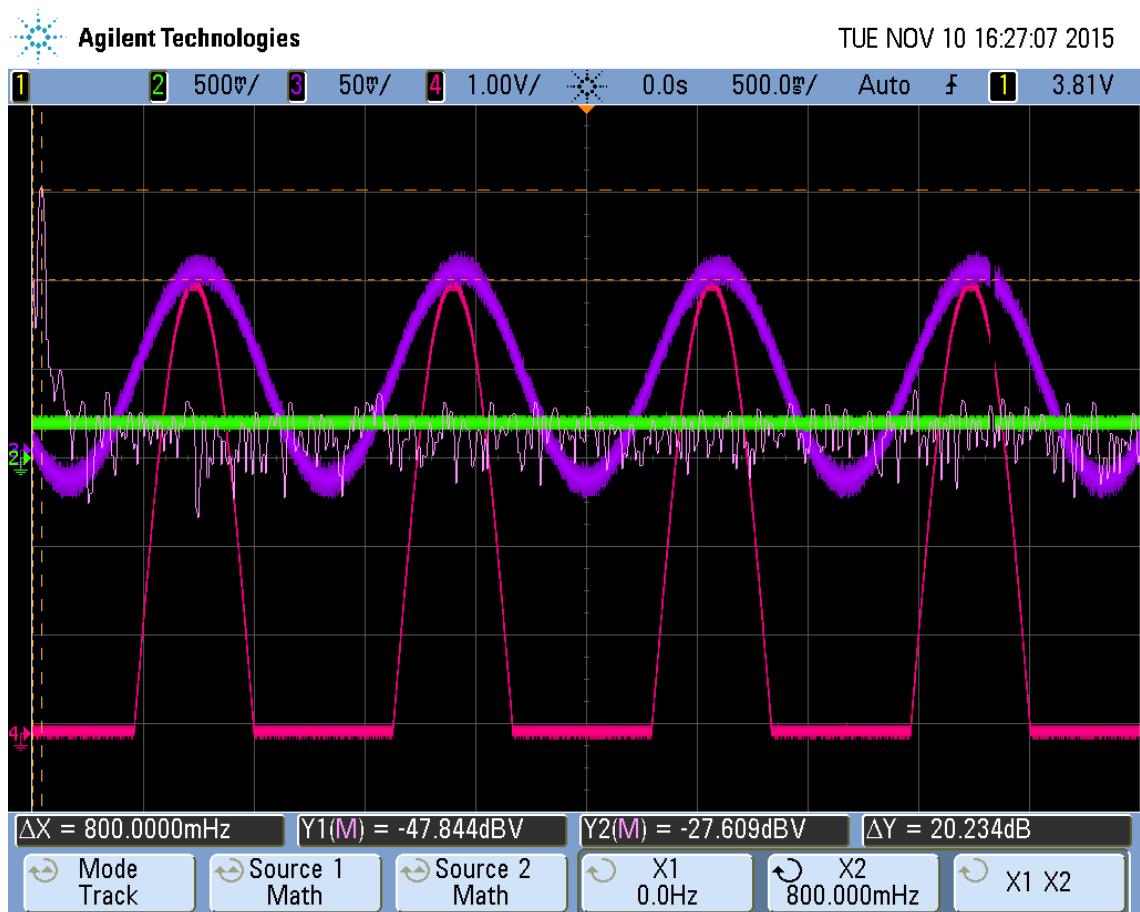
Indsættelse af komponent værdier, R_f sættes til $100k\Omega$ og R_g udregnes:

$$A = 1 + \frac{R_f}{R_g} = 100000$$

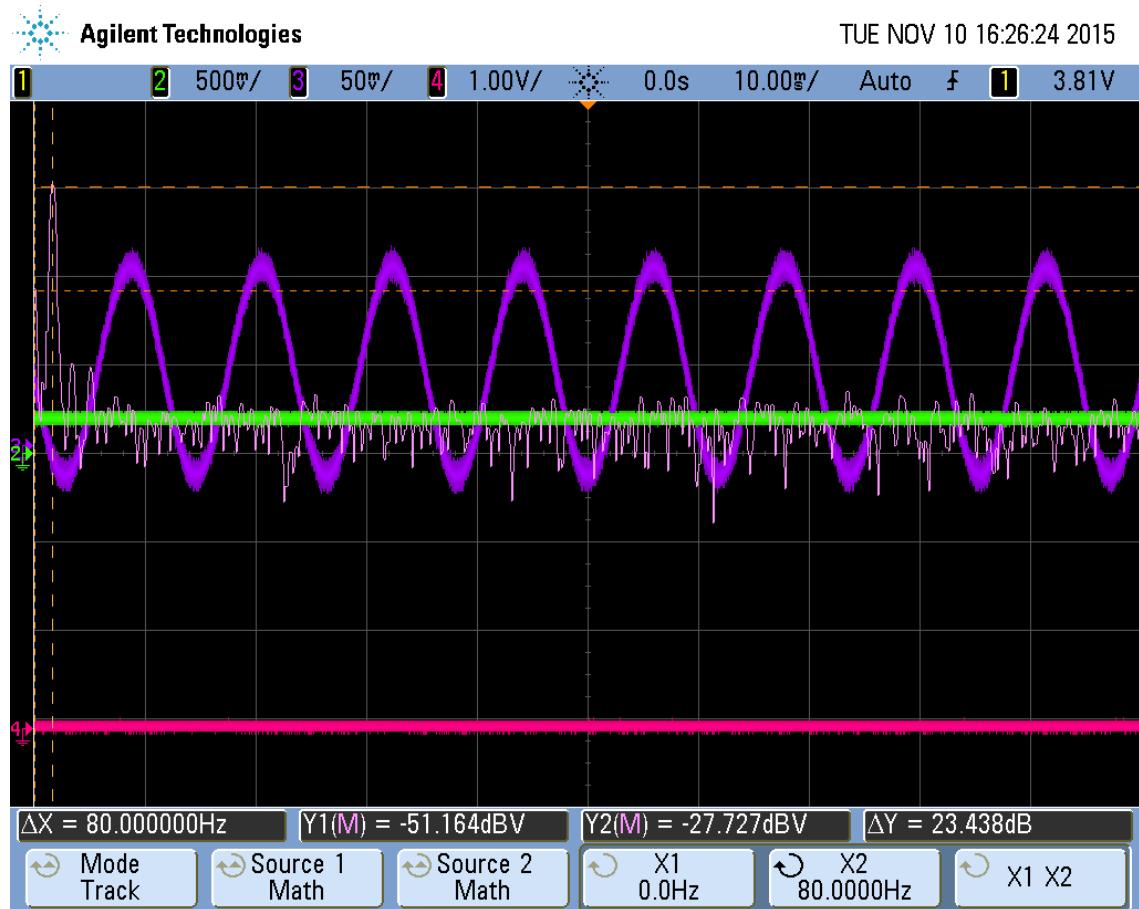
Derfor vælger R_g til at være $100k\Omega$

3.3 Praksis

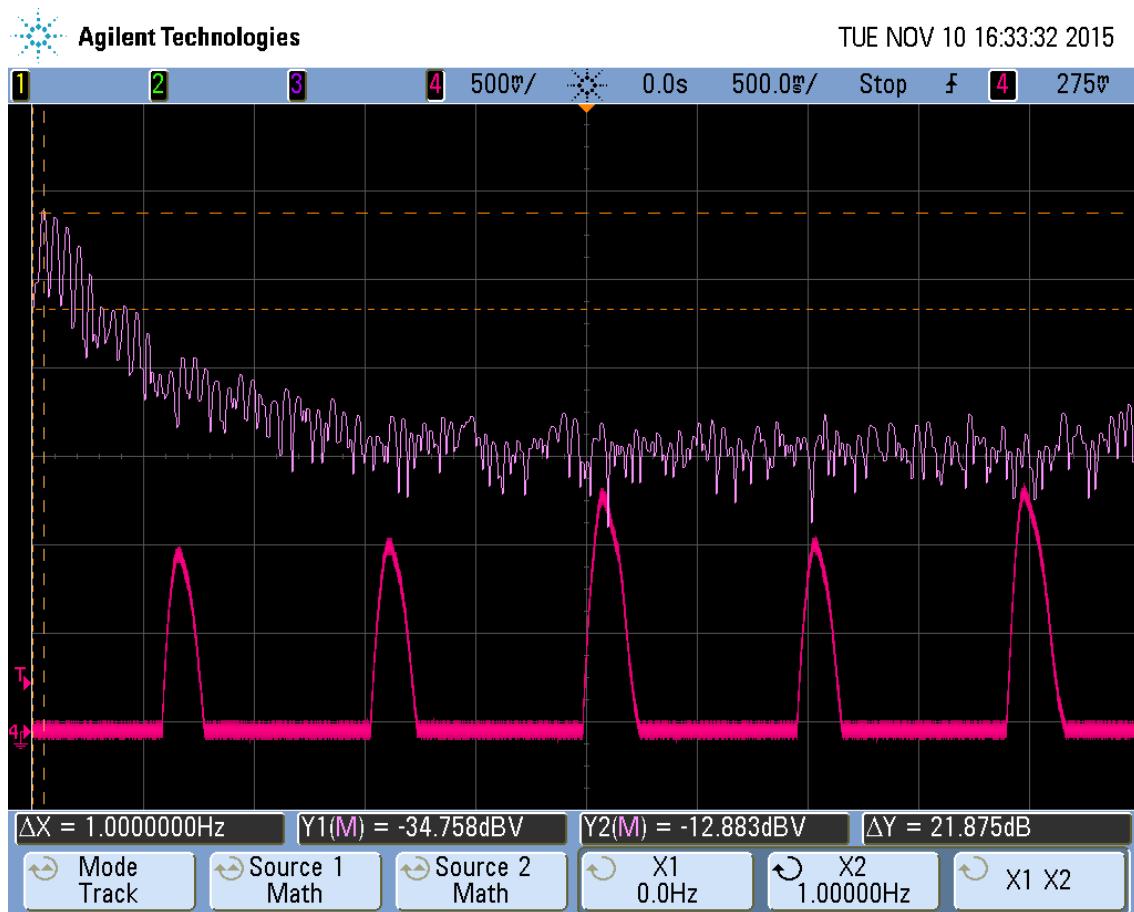
Signal generator med frekvens 0.8Hz svarende til en sund puls. Simuleringen er vist på billederne under. Den lilla kurve er inputsignalet, rød er det oscillérende udgangssignal efter filteret og den grønne kurve er udgangssignalet efter det ikke oscillérende filter.



Figur 3.7. Lilla er input sinus 0.8Hz. Grøn er ren DC. Rød er det filtrerede signal med oscillationerne. Den lyserøde kurve er FFT af det lella signal, hvor det ses at det består af 0.8Hz grundtone.

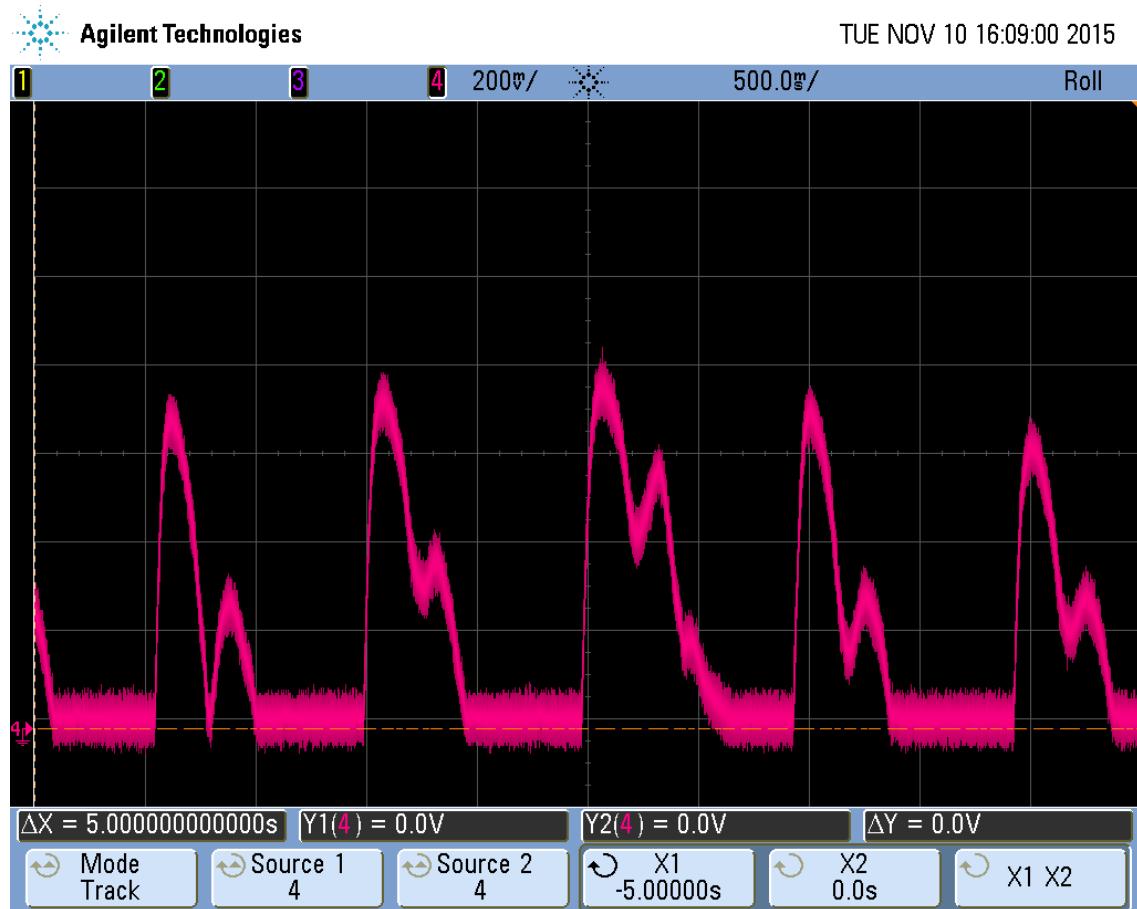


Figur 3.8. Lilla er input sinus 80Hz. Grøn er ren DC. Rød er det filtrerede signal med oscillationerne. Den lyserøde kurve er FFT af det lella signal, hvor det ses at det består af 80Hz grundtone.



Figur 3.9. Oscilloskop billedet viser udgangssignalet efter den oscillérende filtrering (Rød) og en FFT af signalet (lyserød). Her ses det tydeligt at grundtonen er ca 1Hz og har mindst 4 afdelte der af (2,3,4,5 Hz), som gider den spidse smalle form.

Hvis manchet signalet ikke simuleres, men i stedet måles på et rigtigt menneske ligner signalet ikke perfekte sinuser, men i stedet korte udsving, med en grundfrekvens og en masse overtoner. På nedenstående billede ses forskellige typiske udgangssignaler under en normal blodtryksmåling.



Figur 3.10. Når manchet trykket nærmer sig det systolliske tryk observeres der et signal som ser lidt anderledes u.

Det ses meget tydeligt på billede (se figur 3.9 og 3.10) at det pulserende signal svinger meget med hensyn til peak amplituden over tid. Dette fænomen ses kan skyldes mange ting, her iblandt respirationen og andre mekaniske bevægelse. Den analoge filtrering når sin begrænsning, da den ikke kan filtrere peak amplituden til at passe en optimal blodtryksmåling, som kan ses på figur 3.11, hvor amplituderne er konstant stigende ind til MAP og så aftagende efterfølgende. Til at håndterer dette problem anvendes der digital filtrering.

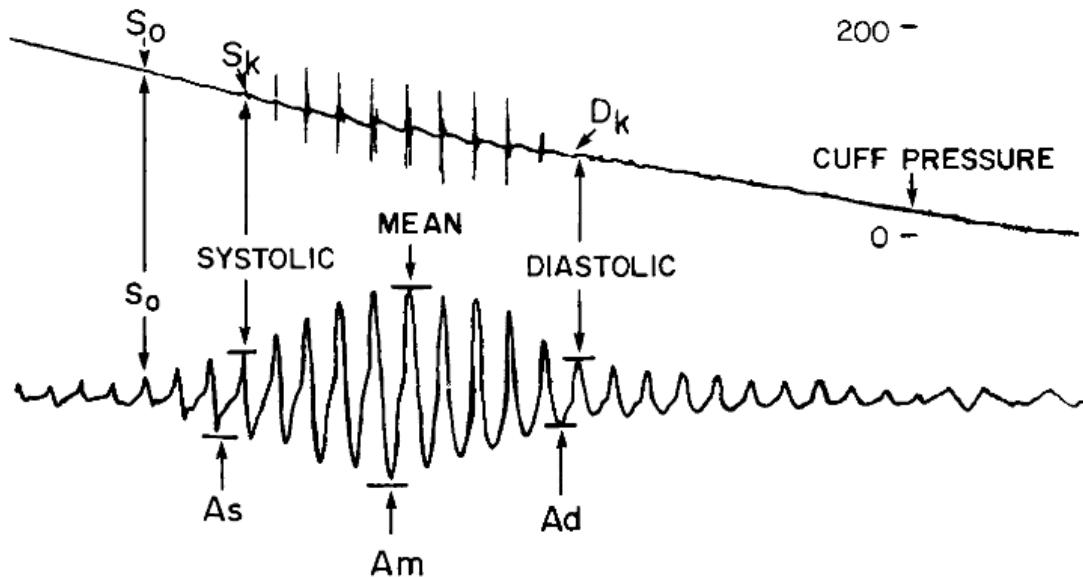


FIGURE 1. Cuff-pressure with superimposed Korotkoff sounds and amplified oscillations in cuff pressure. The symbols identify the measurements used to identify systolic and diastolic pressure (see text).

Figur 3.11. Signal fra oscilometrisk blodtryksmåling

Reference ¹

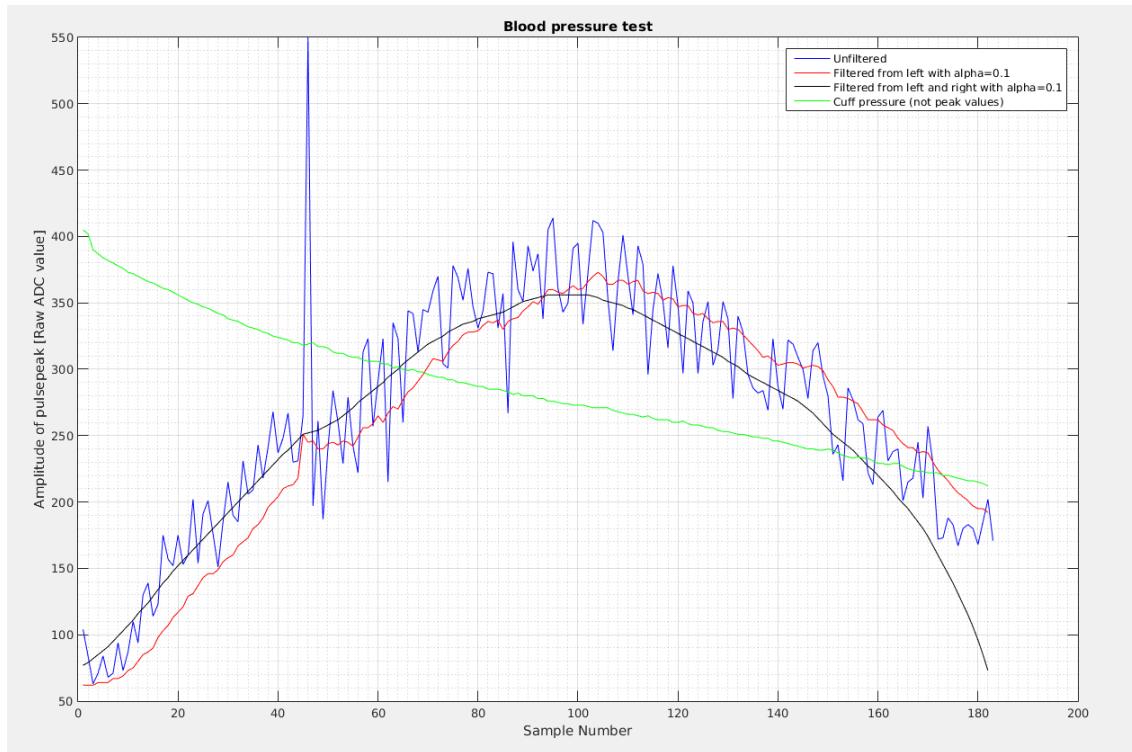
3.4 Digital filtrering

Til digital filtrering er der anvendt et eksponentiel midlingsfilter med zero group delay.

$$y(n) = \alpha x(n) + (1 - \alpha)y(n - 1) \quad (3.3)$$

Filteret midler peak amplitryderne, så den gennemsnitlige amplitydé forøgelse/dæmpning kommer til udtryk. Blodtryks filteret anvender en alfaværdi på $\alpha = 0.11$. Udermere filtreres data fra begge sider, altså først fra venstre mod højre og så bag efter fra højre mod venstre. Denne teknik sikrer ingen group delay, som er vigtigt for at kunne bestemme MAP, som det tryk der er i manchetten på samme tidspunkt som den maksimal målte peak amplitydé. Dette er bedst illustreret på figur XX, hvor manchettrykket er faldende over tid og peak amplituderne er stigende og efter MAP så faldende. Når det rå signal midles fra venstre mod højre opstår et group delay på peak amplitryderne, som ikke er på manchet tryk data'en. Af denne grund må der ikke være group delay på signalet.

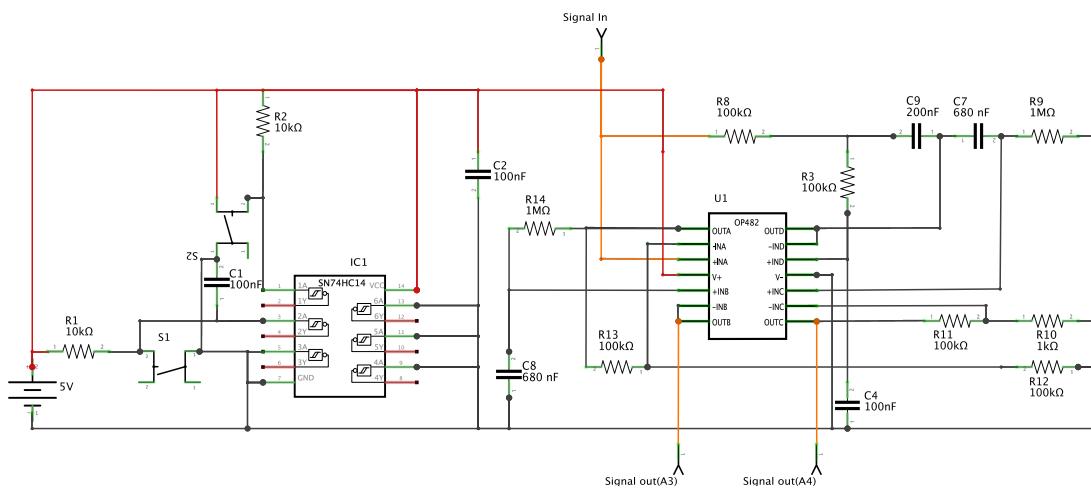
¹FiXme Fatal: Billedet er taget fra CHARACTERIZATION OF THE OSCILLOMETRIC METHOD FOR MEASURING INDIRECT BLOOD PRESSURE



Figur 3.12. Graf med data fra en blodtryksmåling. Her ses at det rå signal er støjfyldt. Grøn er manchet trykket, blå er de rå peak amplituder, rød er filtreret en gang fra venstre mod højre, sort er filtreret fra begge sider

4 | Hardware

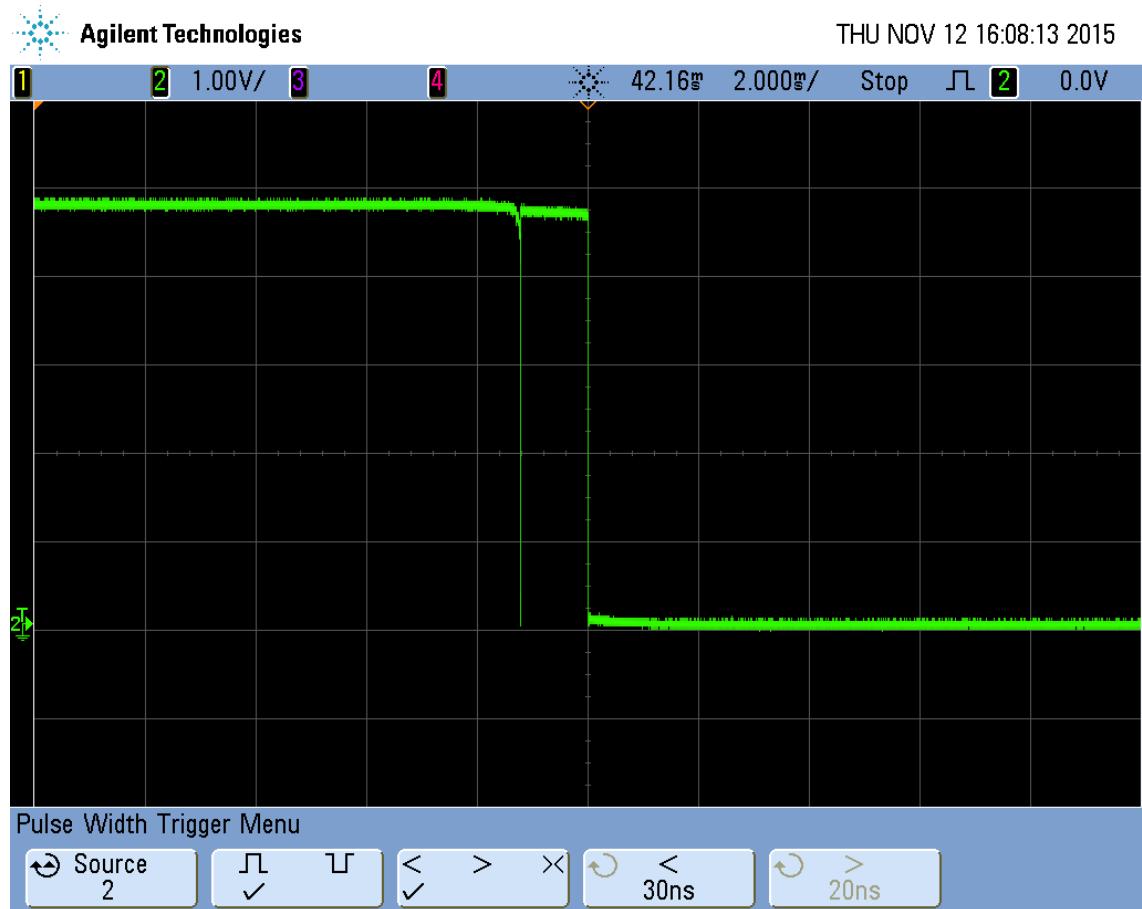
4.1 Schmetic



Figur 4.1.

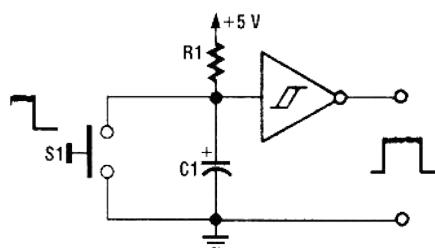
4.2 Knapper

For at imødegå debouncing ved tryk på knapperne, er der bygget et støttekredsløbs til disse. Debouning kan ses på figur 4.2 hvor det mekaniske splip på knappen, giver anledning til en løs forbindelse ses på multimeteret som en lav efterfulgt af en høj, for så at blive lav igen. Microcontrolleren er hurtig nok til at registrer dette som et nyt knaptryk.

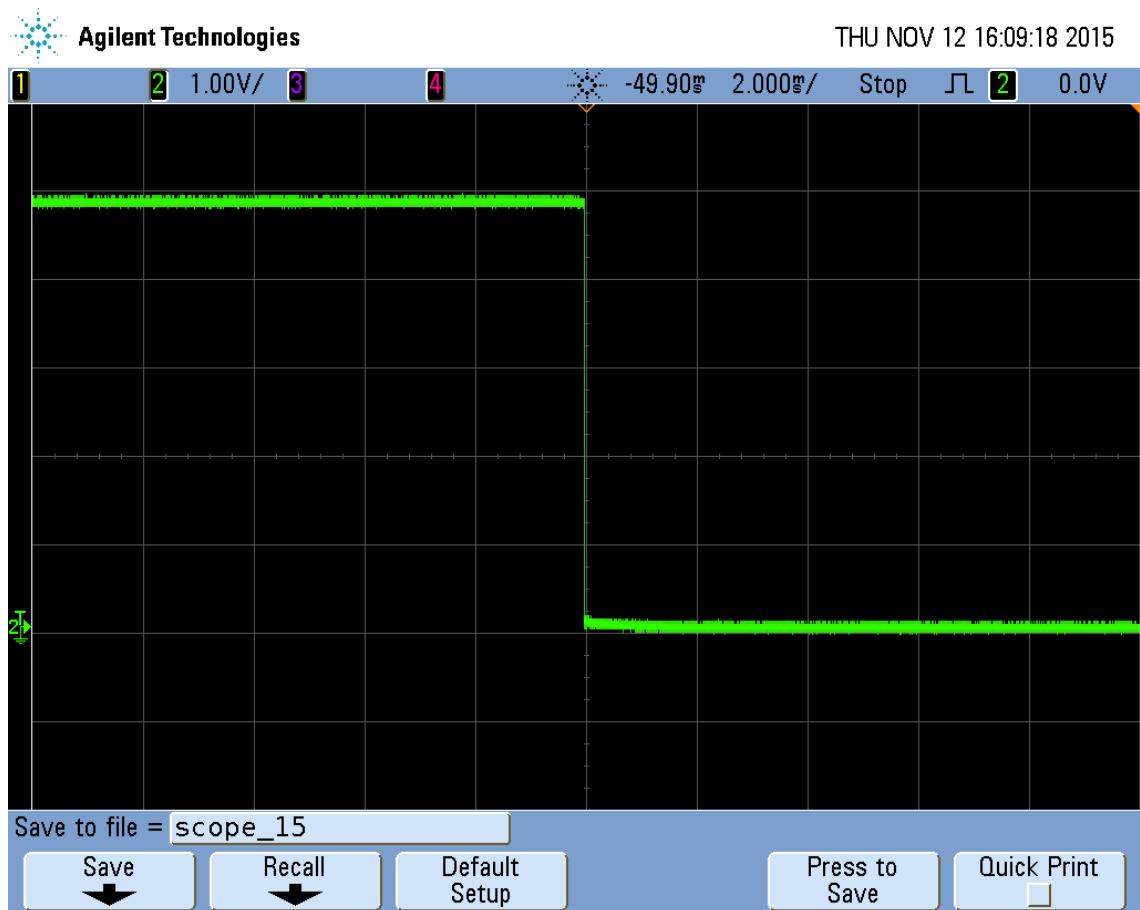


Figur 4.2. Figuren viser et knaptryk, hvor der opstår debouncing.

For at imødekomme er der implementeret et debounce kredsløb (figur 4.3) med en inverting schmitt trigger. schmitt triggeren leverer en stabil høj og kapacitoren forhindrer den lav ved kortvarig open circuit under knaptryk.



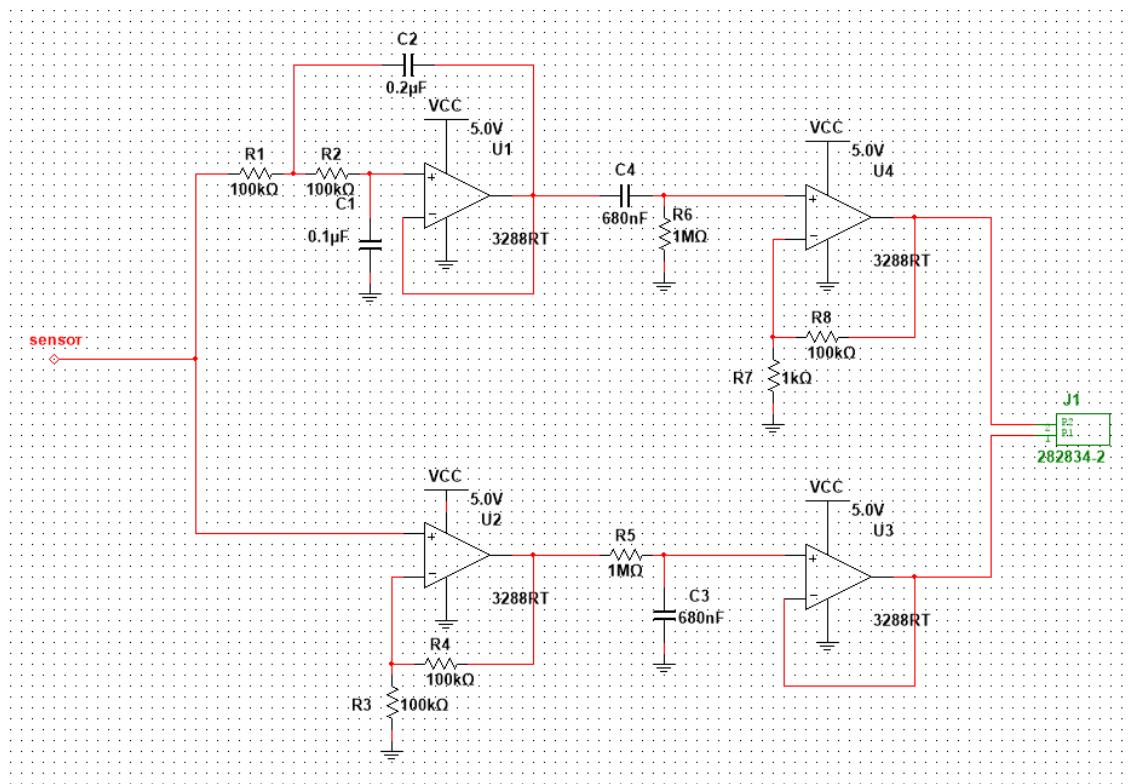
Figur 4.3. Anti debouncing kredsløb



Figur 4.4. Knaptryk hvor der er implementeret et anti debouncing kredsløb.

4.3 Filter

Filtreringen af signalet er opdelt i isolering af manchet tryk og isolering af pulserende signal. På figur 4.5 ses opdelingen af signalet i to til ADC'en. Dyberer forklaring af filterdesignet kan findes under afsnit 3 omkring filter design.



Figur 4.5. Schematics over filter design