Danmarks Tekniske Universitet

Embedded Systems 02131

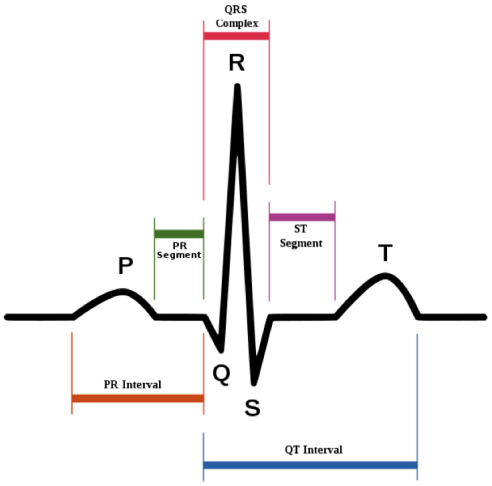
Assignment 1

Gruppemedlemmer:
Cebrail Erdogan s113414
Christian Mathias Rohde Kiær s123812
Jonathan Binner Becktor s123094

30. september 2013

Introduktion

Vi har fået stillet til opgave at implementere Pan-Thompkins QRS detection algorithm. Algoritmens formål er at analysere et ECG signal og QRS komplekset. QRS komplekset giver det elektriske billede af ventriklernes sammentrækning. Et eksempel på et QRS kompleks ses på figur 1.



Figur 1: QRS Kompleks

Vores mål med algoritmen er derfor at give en vurdering af R peaks i QRS komplekset. Dette vil kunne alamere patienter om eventuelle hjerteproblemer. Vores algoritme skal derfor kunne registrere følgende:

- Intensiteten af R-peaks.
- Varigheden af R-peaks.
- Tiden siden sidste R-peak.
- Advare hvis intensiteten af R-peaket er under en vis grænse eller hvis intervallet mellem R-peaks er ustabilt.

Indhold

1 Kravspecifikationer **3**
1.1 Funktioner . . . 3
1.1.1 Data erhvervelse . . . 3
1.1.2 Filtrering . . . 3
1.2 Bruger output . . . 4

2 Programdesign **5**
2.1 Data indlæsning . . . 6

3 Implementering **6**
3.1 Filtre . . . 6
3.1.1 Low-Pass, High-pass, Derivative og Moving window implementation . . . 6
3.1.2 Squared . . . 6
3.2 QRS Detection algorithm . . . 6
3.2.1 peaks . . . 7
3.2.2 Gemme peaks . . . 7
3.2.3 Beregning af RR intervallet . . . 7
3.2.4 Søg baglæns for peaks . . . 7

4 Resultater **8**

5 Diskussion **10**

6 Konklusion **11**

7 Bilag **12**
7.1 HelloWorld.c . . . 12
7.2 ReadFile.c . . . 12

1 Kravspecifikationer

Vi vil her beskrive kravene for programmet,

1.1 Funktioner

1.1.1 Data erhvervelse

Vores program skal kunne simulere en ECG måling. Der skal derfor indlæses et datasæt, som representere data fra en rigtigt patient, i realtime for så nøjagtigt som muligt at simulere en ECG måling.

1.1.2 Filtrering

Da algoritmen kræver at datapunkterne behandles med stor præcision, anvendes 5 forskellige algoritmer til at filtrere den data vi har fået. De forskellige algoritmer gør alle noget forskelligt og meget spesefikt for datapunkterne. Low-Pass filtrer som i navnet lader de lave frekvenser komme forbi og blokerer de alt for høje. Highpass gør det modsatte. Derivative forstærker de signaler vi er interesseret i. Square filtrer tildeles for at vise større forskel mellem datapunkterne og for at gøre alle datapunkterne positive. Tilsidst bruges Moving window implementation filtret for at gøre signalet mere "blødt"altså at fjerne ekstra støj for at fobredre visualiseringen af R-peaks. De 5 filtre skal anvendes i følgende orden:

- Low-pass filter
- High-pass filter
- Derivative
- Squaring
- Moving window integration

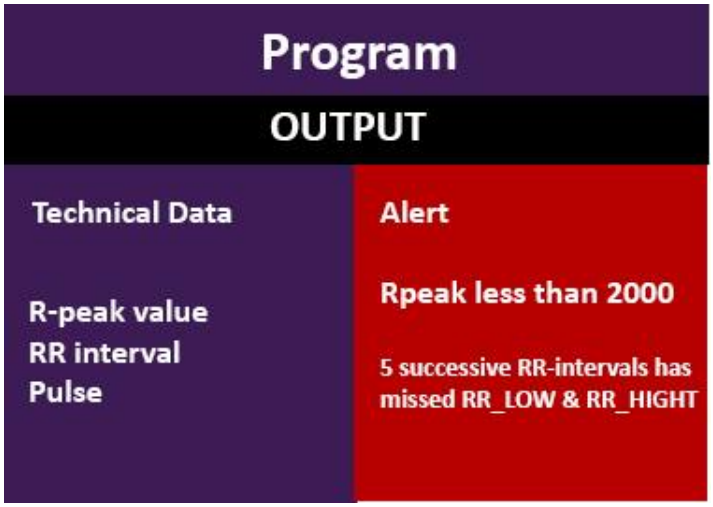
1.2 Bruger output

Programmet vil være i stand til at informere brugeren om praktiske oplysninger. I løbet af programmets kørsel vil brugeren blive informeret både om tekniske ting og advarsler angående patienten. De tekniske data indeholder:

- Rpeak værdi: Værdien af nuværende Rpeak
- RR interval: Tiden fra sidste Rpeak til nuværende
- Pulse: Hjertets puls. Beregnet ved at sige: ((250/RR_ AVERAGE1)*60).

Alarmering:

- Rpeak under 2000: Skyldes for svag hjerteslag.
- RR intervallet er ikke mellem RR_ LOW og RR_ HIGH 5 gange i træk: Skyldes usædvanlig interval mellem hjerteslag.

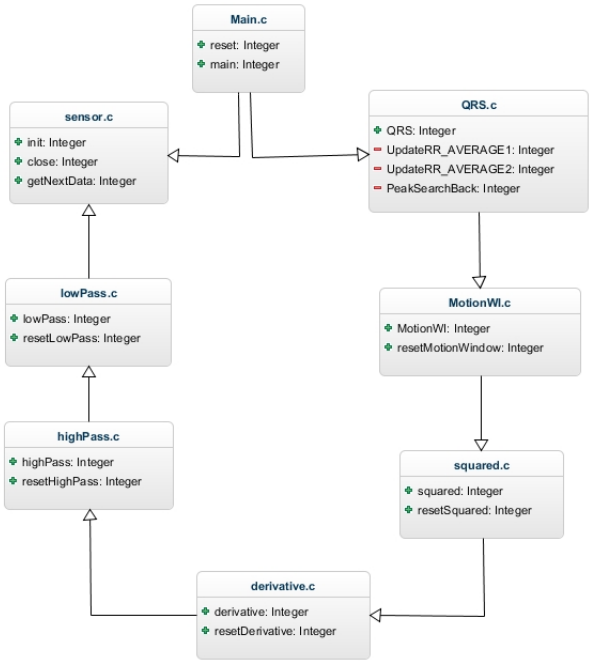


Figur 2: Output til brugeren

2 Programdesign

Struktur

Vi har taget udgangspunkt i de udleverede program filer til senoren, og efterfølgende oprettet klasser for alle filtre og QRS-registrings delene af programmet. Vi har en main klasse der kører selve programmet, og derfra kalder de nødvændige funktioner.
Filtrene er oprettet med navn, og funktionerne i det pågældende filter bliver derfra kaldt (eks. er low-pass filteret oprettet som lowPass.c). R-peak registrering forgår i QRS.c, hvorfra også udregningerne af intensiteten, varigheden og tiden siden sidste R-peak foregår.
Opdelingen er sket på denne måde, da vi synes en klasse for hver filter gav mest overskuelighed. Det er meget let derfra at gå ind og rette på en enkelt del af et givent filter, uden det giver problemer i resten af programmet. At holde QRS funktionerne adskildt fra resten af programmet giver samme fordele. Nedenfor ses et klassediagram for vores program.



Figur 3: Klassediagram

2.1 Data indlæsning

Da programmet skal forestille at være en simulation af en ECG måling, ved brug af Pan-Thompsons QRS detection algorithm, skal programmet kunne læse data fra en fil. Til dette har vi vidreudviklet på sensor.c filen, og implenteret funktionen getNextData(). Dette vil give os muligheden for at hente data fra en fil, og bruge dette datasæt til analyse af QRS detection algoritmen. Sensor.c bruges derefter i main.c og henter alle vores signal data ind til analysen.

3 Implementering

3.1 Filtre

De forskellige filtre er belvet implementeret i hver sin klasse. Her er en gennemngang af nogle af dem da nogle af dem er meget ens hvor det eneste der gør dem forskellige er den matematiske formel.

3.1.1 Low-Pass, High-pass, Derivative og Moving window implementation

Her initializeres 2 arrays en til resultaterne og en til dataen. Størrelsen af arrayet afhænger af hvor langt tilbage formlen siger vi skal have resultater fra. Fx. skal vi have x_{n-12} i Low-Pass filtret, vi skal altså have et array der gemmer x værdierne til og med x_{n-12}. Det bliver gjort med en forløkke som sætter den positon næste lig den forrige. Hvorefter vi så returner resultatet. Derudover har vi også tilføjet en reset metode hvis man skal bruge filtrene igen. Den sætter bare alle positionerne i arraysne til nul.

3.1.2 Squared

Squared filtret ligger også i sin egen klasse. her får den input fra Derivative og returner inputtet ganget sig selv.

3.2 QRS Detection algorithm

QRS algoritmen er uden tvivl den vigtigste komponent i hele systemet. Den har til formål at finde ”peaks” – altså toppe – og behandle de fundne peaks. Algoritmen gør det muligt at fortolke frekvenserne. Nogle af de fundne peaks vil blive fortolket som hjertepuls (Rpeaks),

mens nogle af dem fortolkes som støj. Hvis intervallet mellem Rpeaks er usædvanligt vil programmet kommet med output - advarselser. De essentielle dele af algoritmen bliver uddybet nedenfor.

3.2.1 peaks

Peaks er simple at finde. Der hentes data fra filteret. De hentes en ad gangen, men man kunne også hente tre ad gangen. Grunden til at der skal bruges tre ”datapoint” skyldes betingelsen: $Xn - 1 < xn, xn > xn + 1$,

3.2.2 Gemme peaks

Peaks er defineret som ”struct”. Struct gør det muligt at oprette et object og give den forskellige variabler. Så alle vores peaks har variablerne ”value”,”RR” og ”index”. Nu er peak en type som kan tildeles. De fundne peaks gemmes i et array med typen peak.

3.2.3 Beregning af RR intervallet

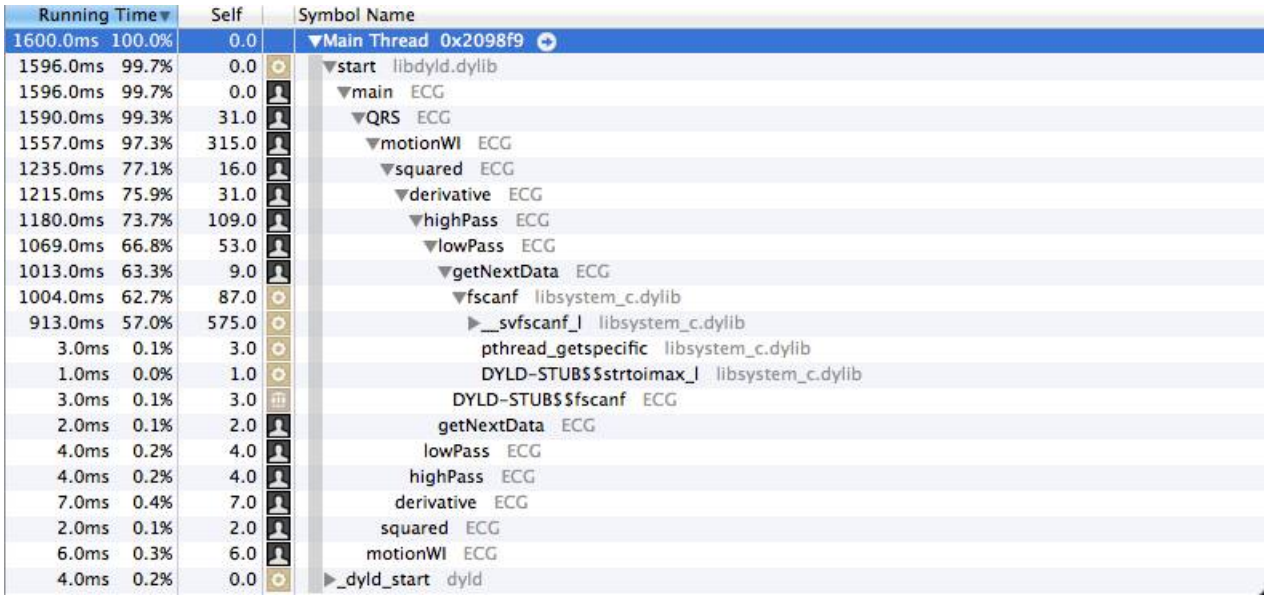
Intervallet af Rpeaks afgøre patientens tilstand, derfor er det også dem der bruges til at de forskellige udsagn i algoritmen. Vi beregner intervallet ved at sige: Datapoint_index - index of last $Rpeak = interval$ Hvis RR intervallet ligger mellem RR_LOW og RR_HEIGHT er der intet galt, men hvis dette viser sig ikke at være tilfældet i 5 gange i træk, vil programmet komme med en advarsel. De seneste RR intervaller gemmes i en array med størrelsen 8, de fundne bruges til at finde gennemsnittet.

3.2.4 Søg baglæns for peaks

Når intervallet ikke er mellem RR_LOW og RR_HEIGHT vil den blive tjekket om den er større end RR_MISS, hvis det er tilfældet vil den starte proceduren ”searchback”. I ”searchback” gennemsøger vi baglæns gennem alle de fundne ”peaks” indtil videre. Lige så snart vi har en peak som er større end THRESHOLD2 fortsætter vi til algoritmens sidste step. Hver gang der betingelsen ikke er opfyldt går vi en skridt tilbage i arrayet.

4 Resultater

Resultaterne er som ventet når "patienten"har problemer med hjertet får vi en advarsel og vi finder alle andre Rpeaks.
Vi har derudover lavet en performance analyse af algoritmeten. Vores test PC har været en maskine med en i7 2.2ghz quadcore processor med 45 watt strømforbrug. Maskinen kørte programmet med den store ECG10800K.txt fil. Programmet kørte med ca. 25% af maskinens CPU kraft under udførslen. Performance chartet for kørslen af programmet kan ses nedenunder:



Running Time		Self	Symbol Name
1600.0ms	100.0%	0.0	▼Main Thread 0x2098f9
1596.0ms	99.7%	0.0	▼start libdyld.dylib
1596.0ms	99.7%	0.0	▼main ECG
1590.0ms	99.3%	31.0	▼QRS ECG
1557.0ms	97.3%	315.0	▼motionWI ECG
1235.0ms	77.1%	16.0	▼squared ECG
1215.0ms	75.9%	31.0	▼derivative ECG
1180.0ms	73.7%	109.0	▼highPass ECG
1069.0ms	66.8%	53.0	▼lowPass ECG
1013.0ms	63.3%	9.0	▼getNextData ECG
1004.0ms	62.7%	87.0	▼fscanf libsystem_c.dylib
913.0ms	57.0%	575.0	▶__svfscanf_l libsystem_c.dylib
3.0ms	0.1%	3.0	pthread_getspecific libsystem_c.dylib
1.0ms	0.0%	1.0	DYLD-STUB$$strtoimax_l libsystem_c.dylib
3.0ms	0.1%	3.0	DYLD-STUB$$fscanf ECG
2.0ms	0.1%	2.0	getNextData ECG
4.0ms	0.2%	4.0	lowPass ECG
4.0ms	0.2%	4.0	highPass ECG
7.0ms	0.4%	7.0	derivative ECG
2.0ms	0.1%	2.0	squared ECG
6.0ms	0.3%	6.0	motionWI ECG
4.0ms	0.2%	0.0	▶_dyld_start dyld

Figur 4: CPU performance chart ved brug af ECG10800K.txt

Det vides at bærbare ECG skanner skal kunne læse 250 punkter i sekundet. Ud fra dette kan det udregnes hvor stort watt forbrug en bærbar maskine vil have, ved brug af QRS algoritmen. Vi udregner først hvor mange clockcycles processoren bruger ved kørsel på en kerne i 1.60 sekunder:

$$clockscycles\_overall = 2.2 \cdot 10^9 hz \cdot 1.60s = 3.52 \cdot 10^9 clockcycles$$

Vi regner derefter ud hvor mange clockcycles der kører for hver linje:

$$clocks\_per\_line = \frac{3.52 \cdot 10^9}{10800000} = 325.93 \frac{clocks}{line}$$

Vi kan derefter regne ud hvor mange watt der bruges per linje på vores test pc. Da maskinen var på 25% og har et strømforbrug på 45w, er strømforbruget under kørsel af programmet på 11.25w.

$$joules\_per\_line = \frac{325.93 \frac{clocks}{line}}{2.2 \cdot 10^9 hz} = 0.00000167 \frac{j}{line}$$

Da det vides at den bærbare enhed skal kunne læse 250 linjer i sekundet, kan det udregnes hvad strømforbruget vil være.

$$watt = 0.00000167 \frac{j}{line} \cdot 250 = 0.00042w = 420 microwatt$$

Dette gør algoritmen meget effetktiv i en bærbar enhed, pga den gode hastighed og det lave strømfobrug. Derudover kræves der ikke meget af hardwaren for at benytte algoritmen.
Koden optog ca. 2.5 mb RAM mens det kørte, hvilket også stiller et forholdsvis lavt krav til hardware specifikationer.

5 Diskussion

Vi som gruppe havde alle lavet en forskellig version af programmet og så senere merget det og fået et program som virker som det skal. Vi var i tvivl hvordan vi skulle håndtere alle de forskellige variabler der ville komme og havde rigtigt mange store arrays sat op, vi fant dog ud af at bruge structs for at have noget der minder objekter.

Hvis vi skulle finde en hardware løsning til dette kan vi se på figur 3 og se hvilke dele af programmet som kører langsommest så vi kan finde ud af hvor lidt hardware vi egentlig har brug for.Det er tydeligt i køretiderne fra processoren at det der bruger længst tid er indlæsningen af data. Da det er en enhed som modtager maks ca. 220 datapunkter i minuttet kan vi se bort fra det. Da programmet ikke behøver at være meget hurtigt kan vi fokusere på at gøre det så energi besparende som overhovdedet muligt. Vi kan se at det som bruger nest mest energi er motionWI metoden som er en af filtrene der bruges til at gøre datapointse mere overskuelige. Den og alle de andre filtre kunne jo eventuelt hardcodes som hardware så der skulle bruges så lidt processor kraft som overhovdet muligt men igen størrelsen er jo altafgørende.

6 Konklusion

Der gives et overblik over hvordan et kompliceret software bygges og hvilken undersøgelser den skal igennem for at blive betragtet som "god software". Programmet blev bygget i tre faser. Første fase omhandler filtre , anden fase omhandler QRS og som sidste fase analysere vi programmet. Programmet simulere en ECG pulsmåler ved at hente data fra txt-filer.

Filtrene består af fem komponenter som har hver deres formål, at implementere dem var simpelt, da de alle sammen består af enkelte formler.

QRS algoritmen var derimod meget udfordrende. De enkelte trin i algoritmen kræver omhyggelig implementering. Det er let at lave fejl når man har med en så stor algoritme at gøre.

Analysering af programmet viste, at den ville fungere meget godt som bærbart enhed. Der blev hverken brugt meget CPU eller hukommelse, derfor er programmet godt optimeret.

Programmet giver en god eksempel på hvordan software implementeres i den virkelig verden, fordi man også tænker på hardwaredelen i baghovedet.

De kommende opgaver kommer til at handle om hardware-simulation, hvilket vil give os forståelse for hardwaredelen.

7 Bilag

7.1 HelloWorld.c

```
#include <stdio.h>

int main(int argc, const char * argv[])
{
    printf("Hello, World!\n");
    return 0;
}
```

7.2 ReadFile.c

```
#include <stdio.h>

int main(int argc, const char * argv[])
{
    static const char filename[] = "/users/christiankiaer/Downloads/ECG.txt";
    FILE *file = fopen ( filename, "r" );

    int maxReading;
    int i1;
    while (fscanf(file,"%i",&i1) != EOF)
    {
        if (i1 > maxReading)
        {
            maxReading = i1;
        }
    }

    printf("The highest EKG reading is: %i", maxReading);
    return 0;
}
```