

```
Puls: 72.0      Gennemsnitpuls: 72.0
SP02: 93.66570436516231%
Puls: 72.0      Gennemsnitpuls: 72.0
SP02: 96.81996154026461%
Puls: 78.0      Gennemsnitpuls: 72.0
SP02: 96.60367186570119%
Puls: 72.0      Gennemsnitpuls: 72.0
SP02: 95.94168642315849%
Puls: 75.0      Gennemsnitpuls: 75.0
SP02: 97.28933747640781%
Puls: 72.0      Gennemsnitpuls: 75.0
SP02: 94.66878038243958%
Puls: 66.0      Gennemsnitpuls: 75.0
SP02: 96.15433582949616%
Puls: 68.0      Gennemsnitpuls: 75.0
SP02: 95.98003426897891%

Process finished with exit code -1
```

Hjerte/lunge kredsløbet - i teori og praksis

Af Büsra Bilgin (s205501), Naveed Shah (s205491), Therese Wagner (s205497) og Troels Kiib (S205492)
Dato 20-01-2021

In this project we used the MAX30102 component to create a functional pulse oximeter. We defined the anatomical structures and organs we measured on and explored the physiological properties of said organs. We encountered issues but prevailed, creating the pulse oximeters software. During the project we had to explore the different components the MAX30102s electric circuit were made of. This was vital because the components inner properties, had to be discussed when creating the structure of the software. We used UML when planning software.

Indholdsfortegnelse

Indledning.....	2
Sundhedsvidenskab.....	2
Hjertet	2
Blodet.....	10
Respiration.....	18
Måledevice og Arduino.....	21
Dokumentation af tilslutning af Arduino modulet	21
Arduino Kode.....	22
Arduino kode afprøvning	23
Tests.....	23
Serial kommunikation til Java.....	25
Arduino.....	27
Arduino program:.....	27
Arduino hardware	27
Microcontroller	28
ATmega382	28
Hvordan fungerer komponenten.....	33
LED.....	33
Fotodetektor	36
18-bit ADC	36
Data FIFO	38
I2C kommunikation	39
Konfiguration.....	40
UML	43
Kravspecifikation.....	43
Analyse	43
Design.....	46
Implementation	50
Afprøvning.....	59
Gruppernes indsats.....	63
Diskussion.....	63
Konklusion.....	64
Bilag filer.....	65
Litteraturliste.....	65

Indledning

Vores formål er at lave et pulsoximeter og beskrive den tilhørerende sundhedsvidenskabelige teori. Til dette har vi fået udleveret en Max30102 komponent. Opgaven er at lave et stykke software, der kan aflæse puls og SPO₂. Til dette skal der følge en forklaring af udviklingsprocessen, samt en forståelse af komponent og den bagvedliggende teori.

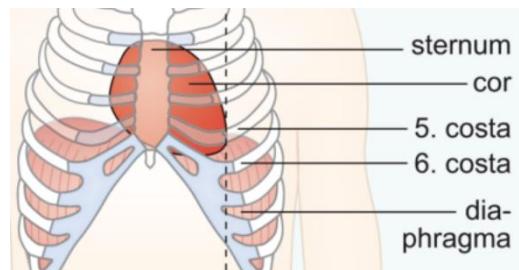
Sundhedsvidenskab

Det følgende er på baggrund af hånden på hjertet Anatomi og fysiologi af Mette Juel Bojsen-Møller/ Oluf Falkenberg Nielsen. 2. udgave og 1 udgave. Samt hånden på hjertet Sygdomslærer af Arene Lykke Viborg/ Anette Walsøe Torup 3.udgave

HJERTET

Hjertes anatomi

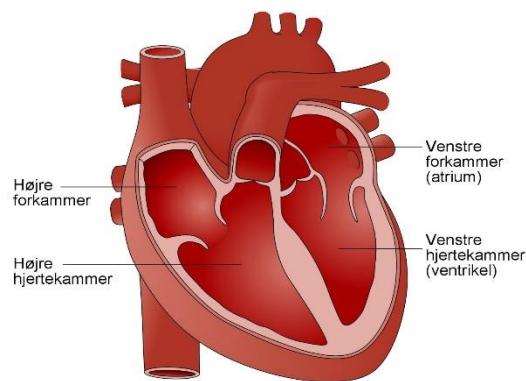
Cor er et muskelorgan, der er på størrelse med en knytnæve, som er placeret i thorax bag sternum og 2 til 6 costae. Cor ligger i et mellemrum mellem pulmonis, som kaldes for mediastinum, og hviler på diaaphragma.



Figur 1: Illustreres cor placering.¹

Hjertets opdeling og inddeling

Hjertet er opdelt i to hjertehalvdele: Dextra halvdel og sinistra halvdel, som er adskilt af septum cordis. Den adskiller det iltede blod i hjertes sinistra halvdel fra det afiltede blod i hjertes dextra halvdel. De to hjertehalvdele er inddelt i fire kamre: to forkamre: artium dxt. og sin., hvor blodet løber ind i hjertet, og to hjertekamre ventriculus dxt. og sin., hvor blodet bliver presset ud af hjertet gennem arterierne. Dette sker når hjertemuskulaturen i hjertevæggen, omkring kamrene, kontraherer.



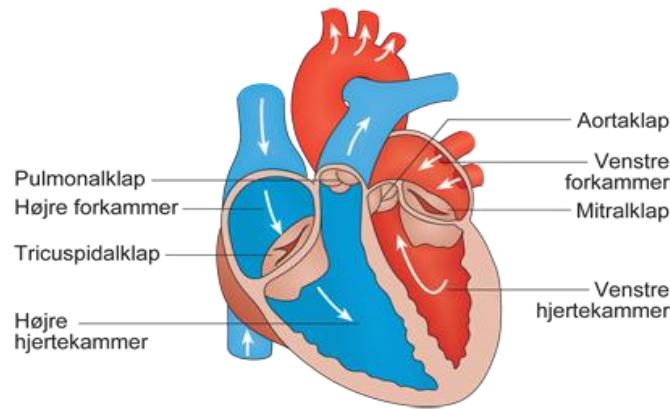
© Birthe Lærne-Baach 2011

Figur 2: opdelingen og inddelingen af hjertet.²

Hjertes klapper

Hjertekamrene adskilles af 4 hjerteklapper: Valva tricuspidalis, valva bicuspidalis, trunci pulmonalis og valva aortae. Valva tricuspidalis og valva bicuspidalis adskiller atrium og ventriculus fra hinanden, så der ikke kan strømme blod fra ventriculus og tilbage til atrium. Valva tricuspidalis sidder i den dxt. hjertehalvdel mellem atrium dxt. og ventriculus dxt. Valva bicuspidalis sidder i sin. hjertehalvdel mellem atrium sin. og ventriculus sin.

Når blodet har passeret valva tricuspidalis og valva bicuspidalis, skal blodet nu passere de to andre klapper, valva trunci pulmonalis og valva aortae. Valva trunci pulmonalis sidder i dxt. hjertehalvdel mellem ventriculus dxt. og truncus pulmonalis. Valva aortae sidder i sin. hjertehalvdel mellem ventriculus sin. og aorta.



Figur 3: Viser hvor de forskellige klapper.³

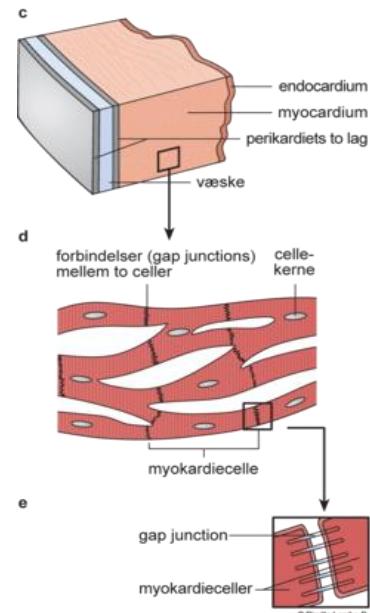
Hjertes væg

Vævet omkring hjertekamrene består af 3 lag: Endocardium, myocardium og pericardium, som har forskellige funktioner.

Endocardiet er det inderste lag, som er en tynd og glat bindevævshinde, bestående af et enkelt lag mesothelceller og et submesothelialt lag, der indeholder blodkar og nerver, som er beklædt af endotel (énlaget pladeepitel). Det glatte endotels funktion er, at gøre det nemmere for blodet at strømme gennem hjertets kamre, for at minimere modstanden i kamrene.

Myocardium er det midterste lag og tykkeste lag, som består af hjertemuskulatur. Hjertemuskulaturen indeholder myocytter, som er specielle muskelceller. Deres cellemembraner er tæt forbundet ved hjælp af gap junctions, som er en membran forbindelse. Det er gap junctions, som gør, at cellerne kan sende elektriske impulser fra en celle til en anden celle.

Perikardiet er det yderste lag, som dækker over hele hjertets ydre overflade, og består af to tynde bindevævshinder, ved navn pericardium viscerale og pericardium parietale. Pericardium viscerale er den inderste hinde, og er tæt forbundet med myokardiet. Pericardium parietale er den yderste hinde, og er hæftet fast på diaphragma, lungerne og de store blodkar ved hjertets øverste ende. Pericardium parietale sikrer dermed, at hjertet bliver holdt på plads i thorax. Imellem de to bindevævshinder findes liquor pericardii, som er en væske, med formål at fugte fladerne og nedsætte friktionen mellem dem. Samtidig med det har perikardiet også en glat overflade, som sammen med væsken imellem de to bindevævshinder, nedsætter gnidningsmodstanden når hjertet arbejder.



Figur 4: Hjertets vægs opdeling⁴

Hjertes 3 faser

Et hjerteslag har 3 faser: atriesystole, ventrikelsystole og fælles diastole.

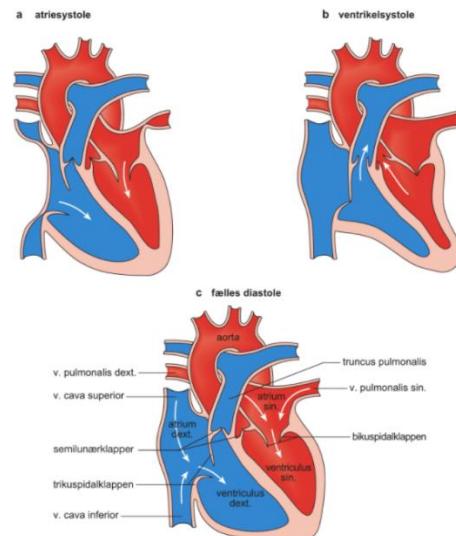
Første fase er atriesystole, som sker i atria, hvor de kontraherer og presser blod ned i ventriculi. Når myokardiet i begge forkamre kontraherer, bliver rumfanget i begge forkamre mindre. Dermed presses der blod, med et højt tryk, gennem valva tricuspidalis og valva bicuspidalis ned til de nedre hjertekamrene.

Derefter starter anden fase, ventrikelsystolen, som sker i ventriculi. Begge ventriculi kontraherer samtidig, hvorved der dannes et overtryk i ventriculi, frem for aorta og truncus pulmonalis, så deres klapperne åbnes, og blodet presses ud.

Der er stor forskel på arbejdsbelastningen af ventriculus dxt. og sin. Ventriculus sin. skal sætte et stort tryk på blodet, da blodet skal kunne strømme til de yderste dele af kroppen. Hvorimod ventriculus dxt. skal sende blodet ud til pulmonis, som ligger lige ved siden af hjertet. Derfor er ventriculus sin. muskulaturen større end ventriculus dxt.

Lukningen af valva tricuspidalis og valva bicuspidalis høres som et lille smæld, og kaldes den første hjertelyd, som markerer starten på ventrikelsystolen.

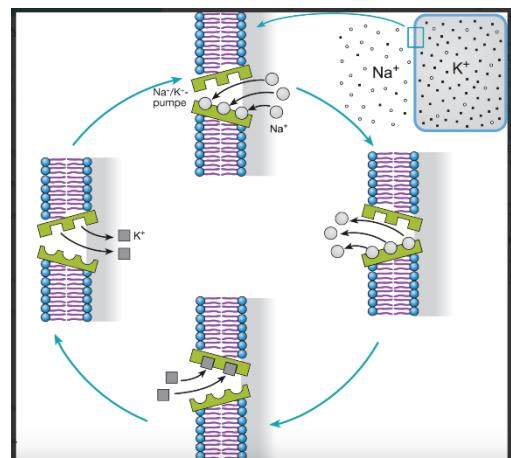
Derefter starter tredje fase, som er fællesdiastolen. Her slapper hjertet af. Ved fælles diastole strømmes en ny portion blod fra veneerne ind i atria og videre ned i ventriculi. Den nye portion blod vil blive presset ud ved næste ventrikelsystole, efter at atriesystolen har fyldt ventriculi op.



Figur 5: Illustrerer hvordan de 3 faser fungerer.⁵

Hjertes impulsdannelsen og udbredelse

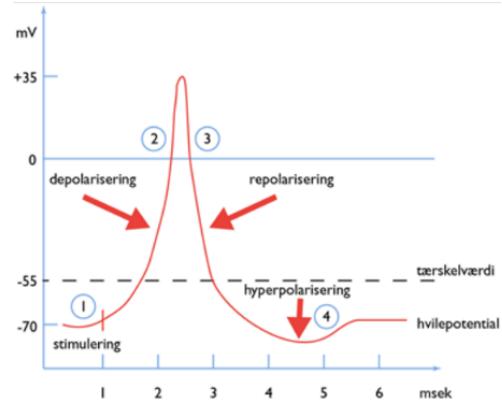
Elektriske impulser fra Cor kan igangsætte atriasystole og ventriculisystole. En elektrisk impuls får hjertemuskelcellerne til at kontrahere. Disse impulser dannes og udsendes via membranpotentialet, som er en elektrisk spændingsforskål mellem indersiden og ydersiden af cellemembranen. Indersiden af cellemembranen indeholder negative ioner, som er for store til at diffundere gennem cellemembranen, hvor ydersiden har positive ioner, såsom Na^+ og K^+ , der kan diffundere ind og ud af cellemembranen via ionkanaler. Membranpotentialet opretholdes af Na^+/K^+ pumpen, som bevarer koncentrationsforskål hen over membranen. Na^+/K^+ pumpen sender tre Na^+ ioner ind i cellen, og der transporteres to K^+ ud. Derfor er koncentrationen af K^+ højere inde i cellerne, end udenfor cellerne. Derimod er koncentrationen af Na^+ højere uden for cellerne, end inde i cellerne. Denne ulige koncentration af Na^+ og K^+ vil få begge stoffer til at diffundere, så snart ionkanalerne åbner. K^+ -kanaler står mere åbne end Na^+ -kanaler, som resulterer i en større diffusion af K^+ ud af cellen, end den tilsvarende diffusion af Na^+ ind i cellen. Denne forskel kaldes hvilemembranpotentialet, og er på ca. -70 mV .



Figur 6: Natrium- og kaliumpumpen⁶

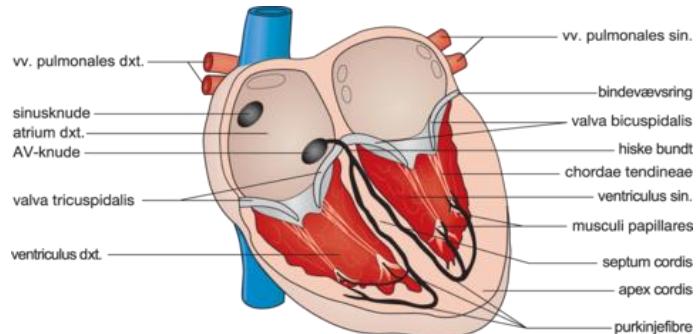
Der opstår et aktionspotentiale i membranpotentialet, som sker via en stimulation af Na^+ -kanalerne, så de åbner sig kortvarigt. Denne kortvarige åbning af Na^+ -kanalerne, lader Na^+ ioner passere ind gennem cellemembranen, så indersiden bliver positiv i forhold til ydersiden. Denne påvirkning sker konstant, men hvis den er stor nok, opstår der et aktionspotentiale. Tærskelværdien ligger på -55mV , og hvis påvirkningen opnår denne, bliver aktionspotentialet nødt til at ske. En depolarisering opstår, når der sker en ændring i cellemembranen, som gør at de negative forhold, på ydersiden af cellemembranen, tiltrækker de positive ioner, der er diffunderet ind i cellen. Dette får de positive ioner til at skifte plads med de negative ioner, og dermed bliver der et overskud af positive ladninger på indersiden af cellemembranen. Ved repolariseringen sker det modsatte af

depolariseringen. Efter repolariseringen opstår der en hyperpolarisering, hvor både Na^+ - og K^+ -kanaler lukker sig igen, og cellens Na^+/K^+ -pumpe sørger for at pumpe Na^+ ud og K^+ ind. Herefter genoptages hvilemembranpotentialet.



Figur 7: Aktionspotentiale⁷

Ledningssystemet i hjertet består af sinusknuden, av-knuden, det hiske bundt og purkinjefibrene. Sinusknuden er en samling af hjertemuskelceller, som sidder øverst i atrium dxt. posteriort.



Figur 8: Hjertets ledningssystem⁸

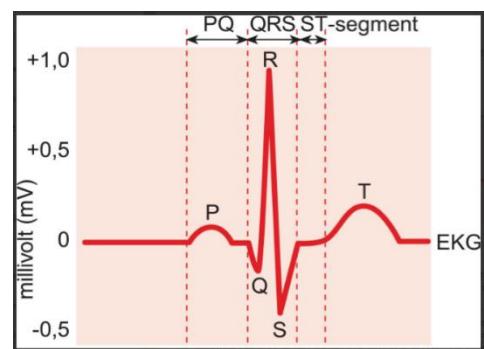
Alle hjertemuskelceller kan danne impulser, men sinusknudens hjertemuskelceller er specialiserede til impulsdannelse. Impulserne spredes i myokardiets celler via gap junctions. Fra sinusknuden udbredes impulserne igennem atria, hvilket forårsager atriesystole. Valva tricuspidalis og valva bicuspidalis kan ikke lede elektriske impulser, hvilket sikrer, at impulsen kun påvirker atriemyokardiet, og kun igangsætter atriesystolen. Det er også derfor, der er en forsinkelse mellem arterisystolen og ventrikelsystolen.

For at starte ventrikelsystolen, skal impulsen fra atria sendes til purkinjefibrene, som er nærliggende apex cordis. Impulsen fra atria opfanges af AV-knuden, som er placeret caudalt i atrium dxt. i septum cordis, hvor impulsen bagefter bliver ført gennem det hiske bundt.

Det hiske bundt gør, at impulsen kan føres igennem de isolerende bindevævsringe. Herefter splittes det hiske bundt til purkinjefibrene, der løber ned gennem septum cordis, og fører impulsen til apex cordis. Herfra sendes impulsen ud i ventriculus myocardium, som vil kontrahere af impulsen. Dermed starter ventriculi systolen.

Der kan opstå lednings blokke, som er svage elektriske impulser, eller en hel afbrydelse af de elektriske impulser i cor. Dette vil medføre enten en pause eller en langsom hjerterytme, der vil give en faldende puls.⁹

Impulserne fra ledningssystemet kan måles på et EKG (elektrokardiogram), som viser udbredelse af impulsen over hjertemuskulaturen. Et EKG er opdelt i P, Q, R, S og T takker.



Figur 9: EKG Diagram¹⁰

P-takken viser impulserne, der dannes i sinusknuden og dens udbredelse over atrierne, som er atriesystole. Der er en lige linje efter P-takken, som afspejler den tid, det tager for impulsen at komme gennem AV-knuden til det hiske bundt. De næste tre takker, QRS-komplekset, viser impulsens forløb via purkinjefibrene, videre ned gennem septum cordis og ud i ventrikelsvæggen. QRS-komplekset repræsenterer ventrikelsystolen.

T-takken afspejler repolariseringen af ventriculi. Takernes størrelse afhænger af tykkelsen af det væv, impulsen skal igennem. Derfor er P-takken mindre end QRS-komplekset, da atriernes muskelmasse er mindre end ventriculi.

Impulsernes forløb fra hjertet videre ud i kroppens væv, kan måles på hudoverfladen.

Puls

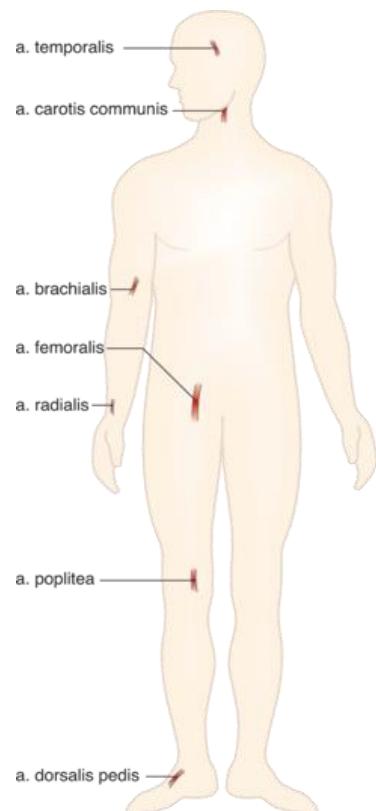
Pulsen måles i minutter, og måles som antal gange der opnås en ventrikelsystole, og der presses blod ud i aorta og truncus pulmonalis. Pulsen angiver hvor ofte Cor slår, og hermed hvor ofte der er en ventrikelsystole.

Man har defineret et normalområde for hvad pulsen er, da det kan variere fra person til person. Hvilepulsen for voksne er ca. 60-80 slag pr. minut. Er pulsen højere end ca. 100 slag pr. minut, er

der tale om tachycardia. Har man en puls, som er lavere end 50 slag pr. minut, er der tale om bradycardia. Pulsen siger derfor noget om, hvor meget hjertet skal arbejde, for at holde organerne og cellerne i kroppen iltet. En forhøjet puls vil derfor betyde, at hjertet har et højere slid.

Tachycardia kan opstå, når man er fysisk aktiv, da kroppen har brug for mere ilt, så hjertet pumper blodet hurtigere ud til kroppen. Tachycardia skyldes udo over fysisk aktivitet; atrieflimren, atrieflagren og rytmeforstyrrelser fra hjertekamrene. Hvorimod bradycardia kan være forårsaget på baggrund af god kondition, eller af sygdomme i hjertets ledningssystem. Det bliver først et problem at have bradycardia, hvis hjertet bliver ved med at pumpe langsomt og ineffektivt, da hjertet ikke bliver i stand til at pumpe tilstrækkeligt ilt/blod ud til kroppen.¹¹ Det kan undersøges, om der er tale om bradycardia eller tachycardi via en EKG-undersøgelse.

Man kan føle og tælle pulsen de steder på kroppen, hvor en arterie ligger lige under hudens overflade. Disse 7 arterier er: A. temporalis superficialis, a. carotis communis, a. brachialis, a. radialis, a. femoralis, a. poplitea og a. dorsalis pedis.



Figur 10: Arterier hvor man kan måle pulsen¹²

Når arterievæggen udvides under ventrikelsystole, kan pulsen mærkes. Arterievæggene udvider sig, fordi blodet presser på væggen. Det betyder, at når hjertet arbejder hårdt, vil ventrikelsystolen

være kraftig, og slagvolumen vil stige. Hvis hjertet arbejder svagt, vil ventrikelsystolen være svag, så slagvolumen vil falde. Dette kan mærkes tydeligt på arterierne¹³.

BLODET

Blodtrykket

Blodtrykket angiver, hvor stort et tryk cor pumper blodet ud i kroppen med. Det er det tryk, som blodet udøver mod blodkarrenes vægge, når der pumpes ud af hjertet. Trykket er forskelligt i de forskellige blodkar. Det højeste blodtryk i kroppen er i arterierne, og det laveste blodtryk er i venerne. Når man skal måle blodtrykket, måler man det arterielle blodtryk. Det arterielle blodtryk består af det systoliske blodtryk og det diastoliske blodtryk. Disse to tryk angives i mmHg, hvor det gennemsnitlige normale blodtryk i hvile for en voksen, er 120/70 mmHg.

Hvis blodtrykket er over 140/90 mmHg, har man forhøjet blodtryk, som også kaldes hypertension. Har man et blodtryk, der er lavere end 90/60 mmHg, har man for lavt blodtryk, altså hypotension. Blodtrykket kan ændres efter kroppens behov for blod og ilt. Blodtrykket ændres også i forhold til kroppens tilstand, som f.eks. fysisk anstrengelse, stress og hvile.

Hypertension kan blive til et problem, da hjertet vil blive overbelastet i en længere periode.

Det vil igennem årene medføre, at myokardiet bliver større, for at tilpasse sig ilt kravet til musklen. Overbelastningen af hjertet kan resultere i hjerteinsufficiens, hvor hjertet pumper utilstrækkeligt. Hypertension belaster ikke kun hjertet men også arterierne, hvilket kan forårsage aterosklerose, som øger risikoen for blodprop.¹⁴

Med det kan vi sige, at jo højere det arterielle blodtryk er, desto mere arbejdskraft skal cor yde, for at pumpe blod ud til aorta.

Blodtrykkets størrelse afhænger af 3 faktorer: Hjertets minutvolumen, den totale perifere modstand og blodvolumen. Faktorerne kan reguleres, så kroppen er i stand til at regulere blodtrykket.

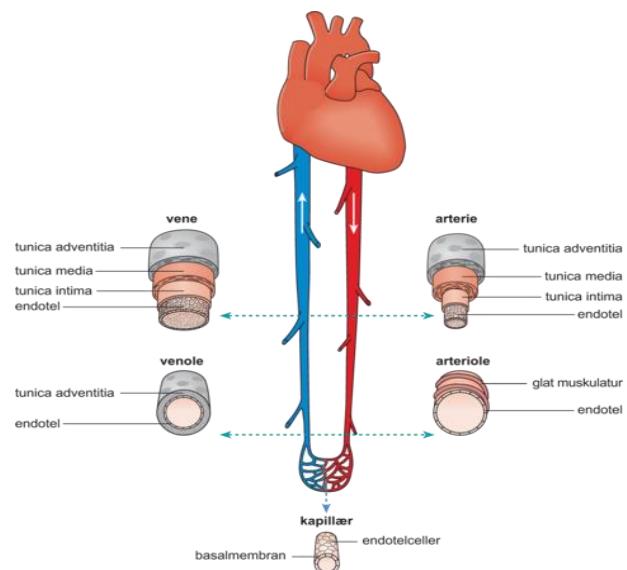
Det vasomotoriske center

Det vasomotoriske center er placeret i medulla oblongata, og styrer iltindholdet i blodet. Hvis iltindholdet bliver for lavt, vil det vasomotoriske center sørge for en kontraktion i nogle af kroppens arterioler, samtidig med at blodtrykket hæves. Iltindholdet i blodet registreres af kemoreceptorer, som sidder i arcus aorta og a. carotis communis, som sender via n. vagus og n. glossopharyngeus signaler til det vasomotoriske center og respirationscentret. Kemoreceptorerne er dem, som registrerer blodets pH (syre/base) og indhold af CO₂ og O₂. Hvis der sker en ophobning af

CO₂ i blodet eller fald af O₂, vil centret øge respirationsmusklernes arbejde. Hvis der derimod registreres lav koncentration af CO₂ eller høj O₂-koncentration, vil det vasomotoriske center sende færre impulser til respirationsmusklerne, og vejrtrækningen vil derfor blive langsommere og mere overfladisk.

Blodkar og blodkarrenes opbyggelse

Når blodet skal transporteres rundt i kroppen, transporteres det i blodkar. Der findes fem typer blodkar i kroppen: Arterier, arterioler, kapillærer, vene og venoler. Blodkar kan indeholde 3 lag i deres væg: Tunica intima, tunica media og tunica adventitia, som hver har sin funktion.



Figur 11: Blodkars opdeling ¹⁵

Tunica intima er væggens inderste lag, og består af bindevævsceller, som både vender mod blodet og lumen. Den del som vender mod blodet, er beklædt med endotel, som sidder på en basalmembran. Den del der vender mod lumen, har også endotel, som er en glat overflade, så blodet kan strømme gennem blodkarret med en lav gnidningsmodstand.

Tunica media er det mellemste lag, og består af glatte muskelceller og elastiske fibre.

Tunica adventitia er det yderste lag, og består af bindevæv, og bl.a. indeholder kollagene fibre, som giver blodkarrene styrke. Udover dette indeholder karvæggen sin egen kaforsyning, dette kaldes for *vasa vasorum*.

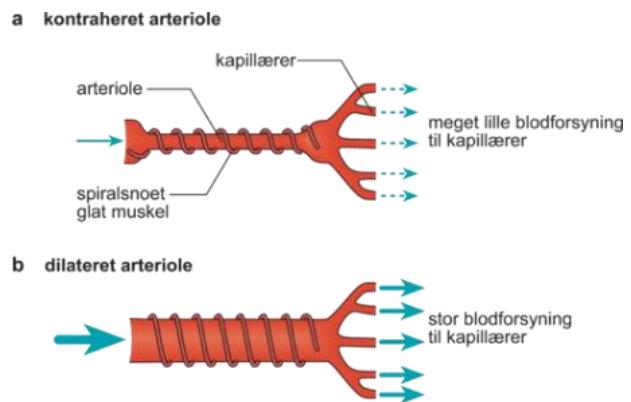
Arterier

Arteria er de største iltrige blodkar i kroppen, og har med formål at føre det iltede blod rundt i kroppen. Arteria skal kunne tåle et højt blodtryk, da det er de blodkar, hvor blodtrykket er højest. Derfor har de også den største karvæg i forhold til dets lumen. Arterievæggen består således af tunica adventitia, tunica media og tunica intima. Arterievæggen kan udvide sig, så der kan justeres mængden af blod, der strømmer igennem. Dette kan de, fordi de indeholder elastiske fibre, og derved opnår en stor elasticitet. Jo tættere på hjertet arteriet er, jo flere fibre indeholder det. De må dog ikke kunne udvide sig for meget, og derfor indeholder de kollagene fibre, der afstiver væggen.

Arterioler

Arterioler består også af tunica intima, tunica media og tunica adventitia, dog er tykkelsen på disse mindre end i arteria. Arteriolerne har en væsentlig funktion, som er at skabe en modstand for blodet. Dette kan gøres fordi der rundt om arteriolen er spiralsnoet muskler, der kan klemme på karret, hvilket forårsager at karret kontraherer. Denne funktion er væsentlig, da den er med til at regulere blodforsyningen til legemet.

Når iltindholdet i et vævsområde er lavt, vil arteriolerne dilatere. Der vil hermed strømme mere blod ud til vævsområdet og ilt indholdet vil stige igen. Når iltindholdet er steget, vil arteriolerne kontrahere.



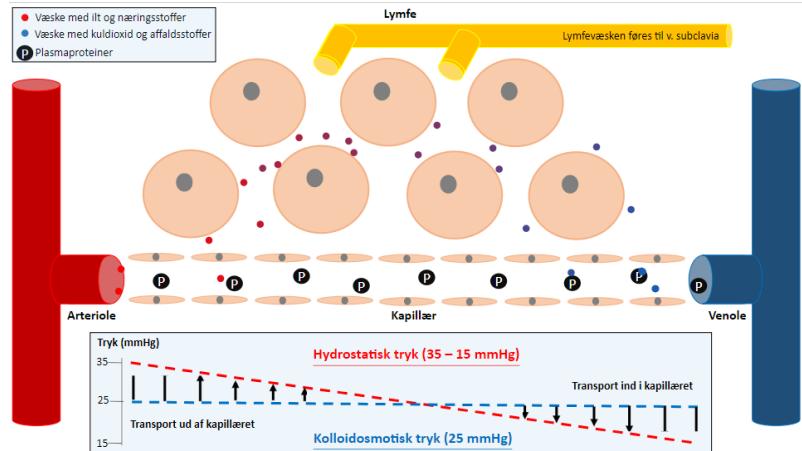
Figur 12: En kontraheret og dilateret arteriole.¹⁶

Kapillærer

Kapillærer er en af de mindste blodkar. De består af tunica intima. Det er vigtigt at kapillærerne har en tynd væg, da det er i kapillærerne, der sker en udveksling af O₂, CO₂ og andre næringsstoffer fra blodet, ud til vævet, også kaldet den kapillærer udveksling.

Blodtrykket falder derfor inden blodet når kapillærerne, så denne udveksling kan finde sted.

Måden den kapillærer udveksling foregår på, er at der er to tryk, det kolloidosmotiske tryk og det hydrostatiske tryk. Hvor det kolloidosmotiske tryk suger vævsvæske ind i kapillærerne og det hydrostatiske tryk, sørger for væske fra blodet, løber ud af karret.



Figur 13: Kapillærer¹⁷

Venoler

Når det affiltede blod har været gennem kapillærerne, kommer blodet til venolerne, som er opbygget af endotel og et tyndt lag bindevæv. Kun de største venoler har muskelceller i tunica media. Venolerne fungere som en transportvej for blodet fra kapillærerne og videre til veneerne.

Vener

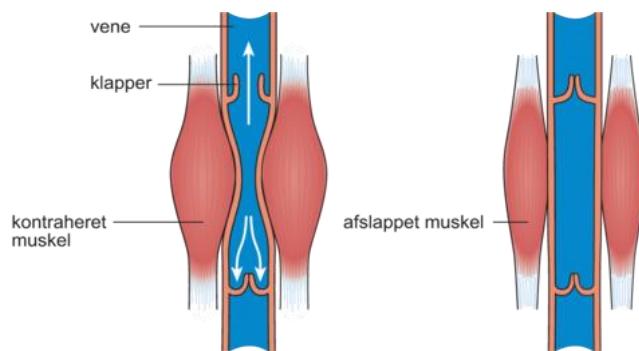
Vener er de største blodkar, der fører ikke iltet blod. Venernes opgave er at opsamle blodet der har været igennem kapillærerne og venolerne, og fører det tilbage til hjertet. Venernes væg indeholder ligesom arteria, tunica intima, tunica media og tunica adventitia, men da trykket i veneerne er meget lavt, er tunica media tyndere, og indeholder ikke lige så mange elastiske fibre. På grund af dette kan veneerne holde mere blod, uden at blodtrykket stiger. Derfor indeholder veneerne også $\frac{2}{3}$ af kroppens blodvolumen. Da veneerne indeholder $\frac{2}{3}$ af kroppens blodvolumen, kalder man også veneerne for kapacitets kar, da de fungerer som et blod reservoir for kroppen. I ekstremitet veneerne indeholder veneerne klapper. Klapperne sørger for, at blodet ikke bliver presset den forkerte vej, når vi pumper blodet rundt vha. venepumperne. Der findes dybe og overfladiske vene i kroppen, de dybe vene er mellem musklerne, og de overfladiske vene ligger lige under huden.

Venepumperne

Venepumperne er pumpesystemer, der hjælper blodet i veneerne til komme tilbage til cor. Behovet for pumperne er især vigtigt for veneerne i benene, da blodet har svært ved at strømme imod tyngdekraften. Dog også når vi skal bruge ekstra blod i kroppen som f.eks. ved fysisk aktivitet.

Pumpesystemet består af to dele: muskelpumpen og thoraxpumpen. Muskelpumpen har primært sin virkning på veneerne i ekstremiteterne, mens respirationspumpen påvirker veneerne i thorax og abdomen. Ved samarbejde mellem disse to pumper, kan blodet i alle kroppens veneer bringes hele vejen til cor.

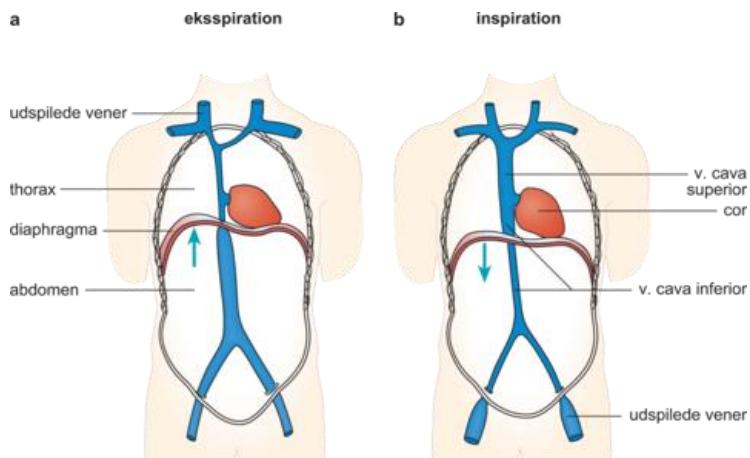
Muskelpumpen fungerer således, at når man kontraherer musklene omkring veneerne, klemmer man på venen, derved skabes der et moment, der presser blodet imod cor.



Figur 14: Muskelpumpen enten er afslappet eller kontraheret.¹⁸

Respirations pumpen virker således, at under inspiration, så bliver pulmonis større, og derved skubber diafragma ned. Dette danner et undertryk i thorax og overtryk i abdomen. Hvor blodet så vil blive skubbet mod cor.

Ved ekspiration så kontrahere pulmonis, og skaber et overtryk i thorax og et undertryk i abdomen. Blodet fra veneerne i thorax løber derfor ind i cor, og blodet fra benene, fylder veneerne i abdomen ud.



Figur 15: Respirationspumpen som enten har eksspiration eller inspiration.¹⁹

Blodet

Blodet består af celler og væske. Cellerne bliver inddelt således: Erytrocytter, leukocytter og trombocyttter. Cellerne i blodet udgør 40-45%, hvor væsken udgør 55-60%.

Det er vigtigt, at der er en konstant opretholdelse af et optimalt indre miljø i kroppen, altså homeostasen, hvilket afhænger af en balancede sammensætningen af stoffer og celler i blodet.

Væsken i blodet kaldes for plasma. Her bliver der opløst forskellige stoffer. Inde i plasmaet findes der forskellige stoffer, såsom næringsstoffer, affaldsstoffer, mineraler, hormoner, vitaminer og plasmaproteiner. Plasmaproteinerne er som andre proteiner i kroppen opbygget af aminosyrer. Dannelsen af proteinerne bliver styret af kroppen, og derfor varierer dannelsen en del i betragtning af egenskaber såsom; pH-værdien, positiv/negativ ladning og vand- eller fedtopløselighed. Der kan, ved at kombinere de 20 forskellige aminosyrer, opnås specifikke egenskaber hos hver type plasmaprotein. Plasma proteinernes funktion er, at de skaber det kolloidosmotiske tryk, der holder væsken inde i blodkarrene. De opfører sig som buffere i syre-base-reguleringen. De transporterer mange stoffer i blodet. De indgår i koagulationen og er med til bekæmpelsen af infektion.²⁰ En erytrocyt er den celle i blodet, som vi tager udgangspunkt i, da det er erytrocytterne der giver blodet den røde farve. Dette giver mening, da vi kommer til at måle blodets farve for at måle puls og iltmætningen.

Erytrocytterne danner i den røde knoglemarv, som alle andre blodceller. Dannelsen sker, ved at der bliver dannet en ny stamcelle. Den nye stamcelle forbliver en stamcelle, og deler sig. For at stamcellen bliver til en erytrocyt, skal den stimuleres af hormonet erythropoietin, der kendes som

EPO. Det bliver dannet, når iltindholdet er lavt i blodet. Iltindholdet i blodet reguleres af nyrerne. Mindskes iltilførslen til nyrerne, vil der blive frigivet erythropoetin, som føres til stamcellerne via blodet. Erythropoetin stimulere stamcellerne til at dele sig, så der dannes flere erytrocytter. Erytrocytterne er røde og formet som en fladmast boldt, denne form kaldes for bikonkav. Den røde farve skyldes hæmoglobin, da den er rød. Blodet er derfor lysets i arterierne og mørkest i veneerne, da iltens indholdet i veneerne er lav. I hæmoglobinet er der jernmolekyler, som får iltten til at binde sig til hæmoglobinet. Erytrocytternes funktion er at transportere O₂ fra alveolesækkene til hele kroppen, og fører CO₂ tilbage til lungerne så det kan udåndes.

Hele transporten foregår via blodet.²¹

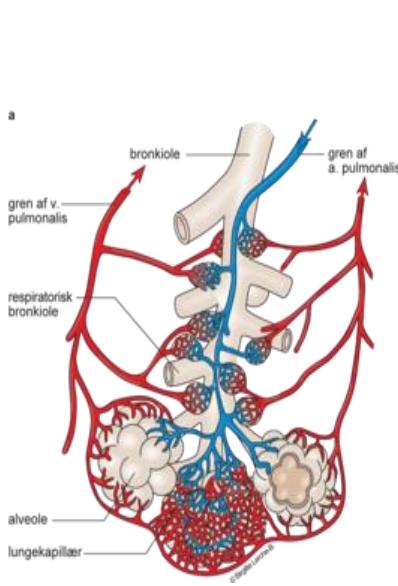
Det lille kredsløb

I det lille kredsløb er det arterier, der fører det affiltede blod rundt, hvor veneerne fører iltet blod tilbage til cor.

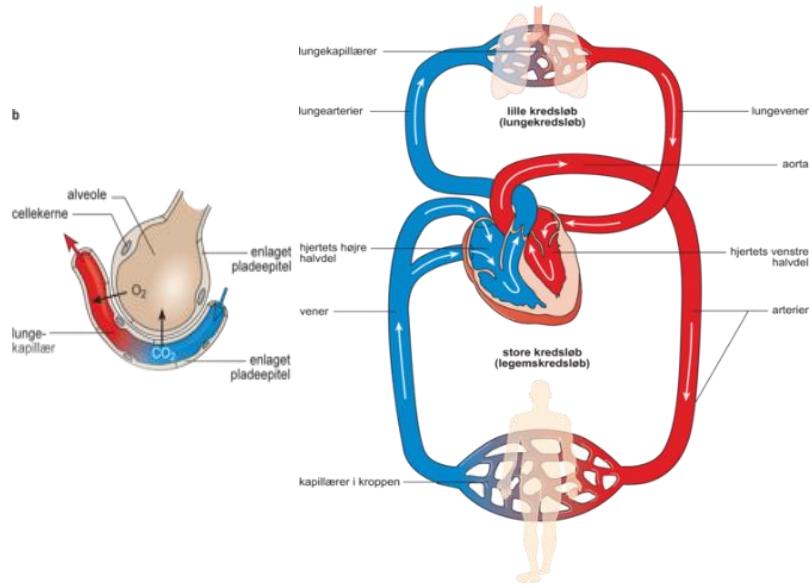
Det affiltede blod, i højre ventrikkel, bliver pumpet ud gennem truncus pulmonalis ud i a. pulmonalis dxt. og sin. Hvor arteria deler sig op i arterioler, og yderligere til kapillærer. Det er kapillærerne i pulmonis, ved alveolesækkene, hvor gasudveksling finder sted. Efter gasudvekslingen vil det iltede blod føres ned til hjertet gennem vv. pulmonales, ned i atrium sin.

Det store kredsløb

Det iltede blod, der er kommet gennem vv. pulmonales, bliver pumpet ud gennem aorta. Arcus aorta fører blod ud til thorax og overekstremiteterne. Aorta abdominalis fører blodet ud til organerne i abdomen. Efter abdomen deles aorta op, hvor den fører blodet ned til underekstremiteterne. Arteria der er kommet fra aorta, bliver til arterioler, som bliver til kapillærer, hvor den kapillærer udveksling sker. Derefter løber blodet gennem venoler ud i vene, hvor venepumpen fører det affiltede blod ind i atrium dx. gennem v. cava superior/inferior.



Figur 16: Lungekapillærer²²



Figur 17: Det store og lille kredsløb²³

Respirationscentret

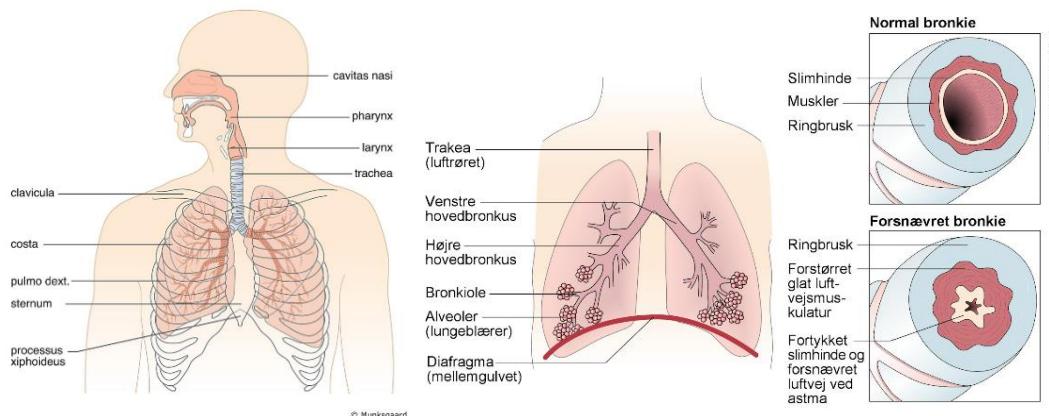
Respirationscentret er placeret samme sted som VMC, og styrer respirationsfrekvensen og dybden, så der kan holdes en passende O₂- og CO₂-koncentration i blodet. Respirationscentret informeres om indholdet af O₂ og CO₂ i blodet fra kemoreceptorerne. Der findes også kemoreceptorer i selve respirationscentret, som mäter CO₂, pH og O₂ i cerebrospinalvæsken. Når respirationscenteret har fået informationer fra kemoreceptorer, vil det justere respiration- og ringmusklerne omkring bronkierne og bronkiolerne vha. nerveimpulser.

Når der skal ske en inspiration, skal respirationscenteret sende nerveimpulser til diaphragma og mm. intercostalis via n. phrenicus og nn. intercostales. Når muskler slapper af, sker der en eksspiration. Eksspiration musklerne får herefter nerveimpulser fra respirationscentret, så de kontraherer, og dermed laver en kraftig respiration. Respirationscentret modtager også signaler fra strækreceptorer i lungevævet, om hvor meget pulmonis udspiles ved hver inspiration. Dette medfører at centret kan styre den efterfølgende eksspiration, som skal blive præcis så lang som det inspireret mængde luft, som blev inspireret.

RESPIRATION

Luftveje

Kroppens luftveje opdeles i den konduktive- og respiratoriske del. Den konduktive del er fra Cavitas nasi ned til bronkiolerne, og den respiratoriske del går fra bronkiolerne ned til alveolesækkene, hvor gasudvekslingen sker. Luftvejenes væg består af en slimhinde, og har som funktion at rense, fugte og opvarme luftstrømmene, før de ankommer til pulmonis. Det yderste på slimhinden kaldes respirationsepitel der indeholder bæger -og ciliebærende celler. Bægerceller danner muscus, og de ciliebærende celler bære, som navnet påstår, ciller, der er lange tynde udløbere, som kan bøjes og strækkes.



Figur 18: De konduktive luftveje²⁴

Figur 19: Den respiratoriske luftvej²⁵

Gasudvekslingen

Afstanden mellem alveolerne og lungekapillærerne er så kort, at CO₂ og O₂ kan diffundere imellem sig. Dette betyder, at blodet udskiller CO₂, samtidigt med at blodet optager O₂ fra pulmonis. Dette kaldes gasudvekslingen.

For at der kan opnås en hurtig og effektiv diffusion, skal koncentrationsforskellen være høj, diffusionen arealet være stort, diffusionsafstanden kort og luftstrømmene skal have en nærliggende temperatur på omkring 37 grader celsius. Begrundelsen for at koncentrationsforskellen skal være høj, er fordi, at vi har at gøre med diffusion. Kroppen er dog smart nok til, at holde koncentrationen af O₂ i alveolerne høj, med en konstant vejrtrækning, også selvom at der er en konstant udtømning af O₂ til blodet i lungekapillærerne. Derved kan kroppen regulere, om der er nok O₂ til stede i lungekapillærerne. Dette kan man se under fysisk anstrengelse, hvor respirationen bliver hyppigere. Det vil sige, at hvis man stopper med at trække vejret, eller har skader i luftvejene, ville

konzentrationsforskellen blive lav, og diffusionen blive dårligere. Dette kan medføre en ophobning af CO₂ i blodet, da udskillelsen ikke er hurtig nok.



CO₂ fra vævscellerne diffunderer ind i kapillærerne, hvor de binder sig sammen med H₂O i erytrocytten. Der dannes derfor H₂CO₃. Derefter bliver H₂CO₃ fraspaltet, så det bliver til hydrogen-carbonat HCO₃ og et hydrogen ion, H. Hydrogencarbonat diffundere ud fra erytrocytten, så det flyder frit i blodet. Når Blodet er ved alveolerne, vil HCO₃ og H ind i erytrocytten, og følge den kemiske process modsat, til vi får H₂O og CO₂, hvor CO₂ så transporteres ud af kroppen gennem luftvejene.

Transport af CO₂

Kroppen sørger for at cellernes energi dannelsen af CO₂, ikke høbes op i kroppens væv, ved at CO₂'en bliver transporteret til lungekapillærerne via blodet. Her udskilles CO₂'en ved gasudveksling i alveolerne.

Det er svært at få opløst O₂ og CO₂ i plasmaet, og derfor sker der en omdannelse af CO₂, så det er let opløseligt i vand. Omdannelsen sker i erytrocysterne. CO₂ i kroppens væv bliver optaget af erytrocysterne i kapillærer, hvorefter CO₂'en omdannes til kulsyre, som deler sig til hydrogencarbonat og hydrogen ioner. Hydrogencarbonat er en base, som er vandopløselig. Dette er afgørende fordi, at den ikke må forstyrre den let basiske pH-værdi på 7,4, når hydrogencarbonatet er i plasmaet. Hydrogencarbonaten ender inde i plasmaet, ved at den transportereres ud af erytrocysterne. Derimod binder hydrogenionerne sig til hæmoglobin, ved at de forbliver inde i erytrocysterne.

Hæmoglobin er et transportprotein, der indeholder jern, og som består af fire proteinmolekyler, globin, og fire hæmgrupper.

Hæmgruppen er den del, der indeholder den røde farve og jern, hvor globin er proteindelen.

I hæmoglobinmolekylet har hver globin en hæmgruppe, det er i hæmgruppen hvor O₂ binder sig til hæmoglobinet. Hæmgruppen er en kemisk forbindelse, hvor jern molekylet får O₂'en til at binde sig til hæmgruppen, via den kemiske forbindelse, der opfylder oktetreglen.

Der kan bindes O₂ til hver af de fire hæmgrupper, hvor den første binding vil forstærke den næste hæmgroupes O₂ binding. Den anden binding vil forstærke den næste binding, og på den måde vil det fortsætte.

Er hæmoglobinkoncentration lav, kan der være tale om f.eks. anæmi, altså at der transportereres mindre ilt med hver liter blod, dermed iltmangel.

Det vil sige at der kan være meget ilt til optagelse, men fordi hæmoglobinkoncentrationen er lav, transporterer der mindre ilt, da der ikke er nok hæmoglobin i blodet, som ilten kan binde sig til. Når hydrogenioner binder sig til hæmoglobinet, kaldes hæmoglobinet, det reduceret hæmoglobin. Det reducerede hæmoglobin, får blodets farve til at blive blåligt rødt. Blodets farve er derfor mere blåligt, når det løber væk fra kroppens væv igennem veneerne. Det kan ses på huden, på de overfladiske vene, som får en blågrønt skær. Blodet som strømmer væk fra kapillærer, tager det CO₂ der er i cellerne med, hvor størstedelen er omdannet til hydrogencarbonat og hydrogenioner.²⁶

Iltmætningen

Når blodet strømmer videre fra pulmonis, har nærmest alle hæmoglobinmolekylerne bundet sig til den maksimale mængde af O₂ de kan. Hvor meget O₂ der bliver transporteret, afhænger af koncentrationen af hæmoglobin. Jo flere erytrocytter med hæmoglobin, der er i blodet, jo højere er hæmoglobinkoncentration. Hvilket betyder mere transport af ilt til kroppen.

Man snakker om, hvor meget ilt hæmoglobinmolekylerne har optaget, ved at tale om iltmætningen eller saturation. Iltmætningen i blodet angives i procent. Den normale iltmætning ligger på 95-100%. Hvis antallet af hæmoglobin i erytrocytterne er højt, vil det medføre, at saturationen også er høj. Har man en lav saturation kan det være tegn på dårlig lungefunktion dvs. at iltoptagelsen i alveolerne er for lav.

Transporteres der mindre O₂, end hvad kroppen har behov for, resulterer det i, at blodets hastighed stiger gennem kredsløbet, samtidig med at respirationsfrekvensen også stiger. Det er en reaktion, som kroppen kommer med, for at forsyne sig med tilstrækkelig nok ilt, når der er iltmangel. Denne reaktion vil få pulsen, blodtrykket og respirationsdybden til at stige, da blodet skal transporteret hurtigere.²⁷

Kobling til Elektronik

Iltmætningen kan måles med et pulsoximeter, som er et lille apparat, der indeholder lyskilder, der kan lyse igennem eller på fingeren. Lyset vil her reflektere farven af erytrocytterne tilbage til en optisk detektor, som registreres som en måling. Da iltet erytrocyt absorberer infrarødt lys, og uiltet erytrocyt absorberer rødt lys, kan man ved at lyse med begge, igennem fingeren, finde forholdet imellem dem. Ud fra det forhold kan vi vurdere, hvor mange af erytrocytter der er iltet.²⁸

Udover dette vil målingerne vise en puls, da intensiteten af målingerne vil vise mængde af blodet i fingeren, og da blodet er pulserende, ville man kunne lave en graf af intensiteten.

Alt dette kan bruges af det sundhedsfaglige personale, da vi kender til hvilken grænse, pulsen skal være indenfor. Hvis pulsen beregnes til at være uden for denne grænse, kan vi gentage pulsmålingen, men samtidig vide, hvad den foretagne puls kan være forårsaget. Om brugerens sensoren har en forstyrrelse i hjertet, forhøjet eller for lav puls, samtidig med at tænke om personen har forhøjet eller for lavt blodtryk.

Tager man alt denne viden i betragtning, når man måler pulsen for en bruger, kan man tage det næste skridt, ved at foretage en dybere undersøgelse.

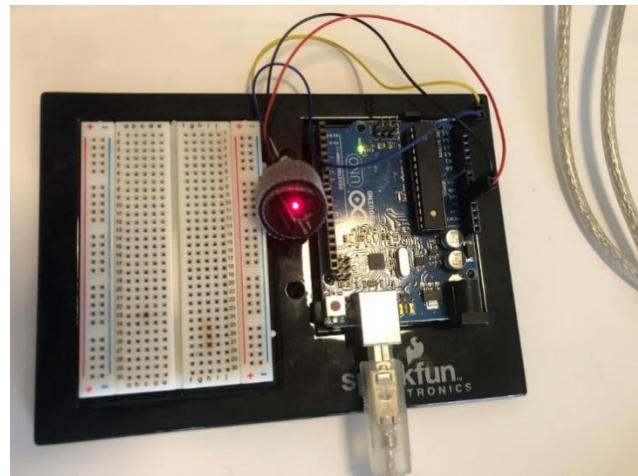
Det samme gælder, når vi på et sundhedsfagligt niveau, skal fortolke iltmætningen. Af samme årsag skal vi have den teori, som er skrevet i dette projekt, i baghovedet. Den normal værdi for iltmætningen er mellem 95-99%, og hvis brugerens saturation måles til at være under dette, skal man først tage højde for, om brugerens hjerte-lunge sygdomme, eller om det er brugerens alder, der har en indflydelse. Efter man har taget højde for brugerens alder, og allerede eksisterende sygdomme, kan man vurdere om iltmætningen er for lav. Er iltmætningen stadig for lav, kan det skyldes, enten at brugerens blodmangel, altså at bruger ikke har mange hæmoglobinmolekyler i blodet, eller så kan det skyldes, at personen har respirationsbesvær, og dermed ikke kan optage rimelige mængder af ilt i blodet.

Vi gør brug MAX30102 pulsoximeter komponent til dette.

Måledevice og Arduino

DOKUMENTATION AF TILSLUTNING AF ARDUINO MODULET.

Vores MAX30102 komponent er tilkoblet til Arduinoen således, at ground wiren fra komponenten skulle sidde i ground på Arduino, V_in til komponenten skulle sidde i enten +3,3V eller +5V på Arduinoen. SCL og SDA pins på komponenten skulle være placeret i A5 og A4, da ArduinoUNO bruger A4 som SDA og A5 som SCL. Efter dette skulle der overføres kode til mikrochippen på Arduinoen, således at man kunne snakke med komponenten. Her hentede vi Arduino biblioteket "SparkFun MAX3010x Pulse and Proximity Sensor Library" der kan hentes direkte fra Arduino softwaren vha. manage libraries, eller via kildehenvisningen. Der skal dog tages forbehold for, at vi hentede den vha. Arduinoen. Derefter kunne man kontrollere LED driveren på komponenten, og derved tænde LED dioderne.



Figur 20: Billede af Arduino opsætning

Arduino bibliotek²⁹

ARDUINO KODE

```
#include <Wire.h>
#include "MAX30105.h" //Dette bibliotek kommer fra https://github.com/Protocentral/Pulse\_MAX30102
MAX30105 sensor;

void setup() {
    Serial.begin(115200);
    sensor.begin(Wire, I2C_SPEED_FAST);

    sensor.setup();
}

void loop() {
    Serial.print(sensor.getIR());
    Serial.print("B");
    Serial.print(sensor.getRed());
    Serial.print("A");
}
```

Figur 21: Arduino kode

Dette er vores Arduino kode, som gør brug af Sparkfuns bibliotek for Max30102 sensoren.

Vi sætter vores hastighed for vores I²C kommunikation til 400KHz, ved brug af Wire.h biblioteket, som er indbygget i Arduinoen, samt MAX30105.h Sensor.begin(wire, I2C_SPEED_FAST). I2C_SPEED_FAST er en kendt værdi i MAX30105.h biblioteket, der skrives i stedet for 400,000. Derefter gør vi brug af metoderne fra Sparkfuns bibliotek, til at printe IR og R værdierne, som er vores Infrarød og Rød LED's værdier.

Dette bliver så overført som en streng.

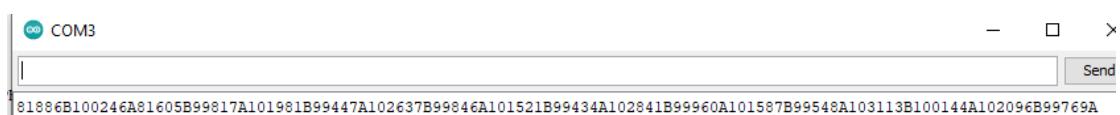
ARDUINO KODE AFPRØVNING

Hvis vi skal afprøve denne kode, så kan vi se på to måder at afprøve den. White- og Blackbox. Dog ser vi ingen grund til at køre whitebox afprøvning, da vi kun har 4 print funktioner i vores metode.

Derfor tager vi udgangspunkt i blackbox. Dette gøres ved at køre koden, og derefter kun have serial monitor åben, for at se, hvad vi får ud af den.

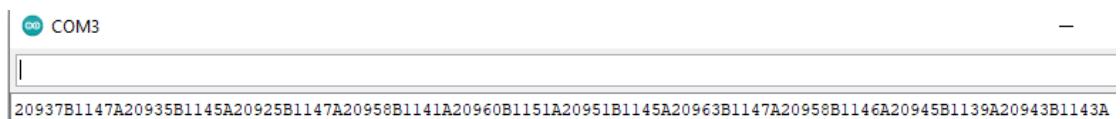
Her skal vi tage højde for, at vi kan have sensoren i to forskellige stadier, når vi skal bruge den.

En hvor vi har fingeren i, og en hvor vi ikke har. Derfor skal vi teste den for begge stadier.



The screenshot shows a terminal window titled "COM3". The window has a close button (X), a minimize button (-), and a maximize button (□). There is a "Send" button on the right side. The text area contains a long string of hex code: 81886B100246A81605B99817A101981B99447A102637B99846A101521B99434A102841B99960A101587B99548A103113B100144A102096B99769A.

Figur 22: Fingeren i



The screenshot shows a terminal window titled "COM3". The window has a close button (X) and a minimize button (-). The text area contains a different long string of hex code: 20937B1147A20935B1145A20925B1147A20958B1141A20960B1151A20951B1145A20963B1147A20958B1146A20945B1139A20943B1143A.

Figur 23: Fingeren ude

Her kan vi se, hvordan vores kode spyer data ud. Dette er også den samme måde, vores Java program modtager data strengene. Vi kan se, at vi får to skiltegn samt 2*6 cifrede numre, hvis vores finger er i. Det vil sige at vi får to forskellige størrelse strenge, afhængig af om der er en finger i eller ej.

Men på baggrund af dette kan vi konkludere, at vores Arduino kode gør præcis det, vi gerne vil have den til at gøre.

TESTS

For at vurdere hvor mange målinger vi fik i sekundet, skrev vi et stykke Arduino kode, der skrev den tid, der var mellem hver måling. Denne data overførte vi derefter manuelt til et Excel program, hvori vi regnede på den. Excel filen ligger i Bilag filer, da det er for stort til at have som screenshots. Test koden ligger også i Bilag filer.



The screenshot shows the Arduino IDE interface with the following details:

- Top bar: TEST | Arduino 1.8.13, Fil, Rediger, Sketch, Værktøjer, Hjælp.
- Toolbar: Save, Load, Open, Upload, Download, Refresh.
- Title bar: TEST §
- Code area:

```
#include <Wire.h>
#include "MAX30105.h" //Dette bibliotek kommer fra https://github.com/Protocentral/Pulse_MAX30102
MAX30105 sensor;

void setup() {
    Serial.begin(115200);
    sensor.begin(Wire, I2C_SPEED_FAST);

    byte ledBrightness = 0x1F; //Options: 0=Off to 255=50mA
    byte sampleAverage = 2; //Options: 1, 2, 4, 8, 16, 32
    byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green
    int sampleRate = 400; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200
    int pulseWidth = 411; //Options: 69, 118, 215, 411
    int adcRange = 4096;
    sensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth, adcRange);
}

void loop() {

    Serial.print(millis());
    Serial.print(sensor.getIR());
    Serial.print("B");
    Serial.print(sensor.getRed());
    Serial.println("A");

}
```

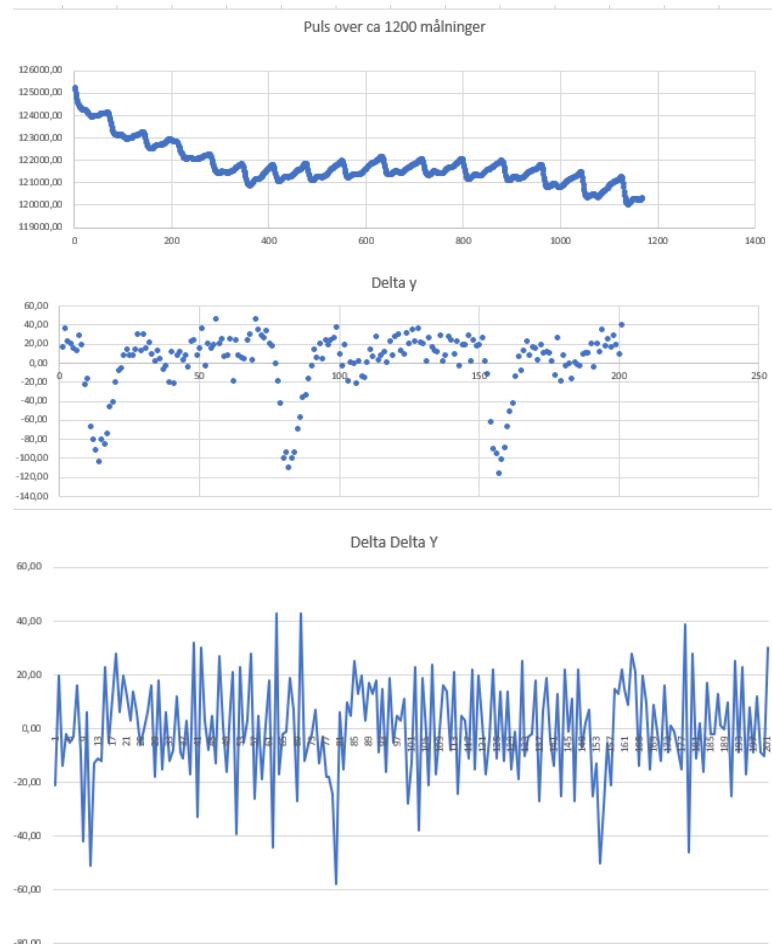
Figur 24: Arduino kode

Det vi fandt var, at der cirka blev overført 1 måling af den infrarøde og røde LED adskilt af et B med et A til sidst, hvert 10 ms. Det stykke kode vi bruger til den reelle måling, har denne syntaks, dog uden et linjeskift. Når data var overført til Excel, beregnede vi på den, og skrev et udtryk for pulsberetningen

Når Δy har et fald på under eller lig med -30, skal en tæller tælle op og vente, indtil at der er gået 27 målinger. Hvilket er fordi, vi ikke vil måle 2 værdier lige efter hinanden.

I vores Excel ark lavede vi grafer, der viser den første og anden afledte. Vi brugte dem til at vurdere, hvor vores grænse skulle være. Deraf kommer værdien -30.

Værdien, 27 målinger, kommer fra, at der maksimalt kan være 3,6667 pulsslag pr. Sekund, hvis man har en puls på 220. Dvs. at det er umuligt at have to toppunkter i et interval af 27 målinger.

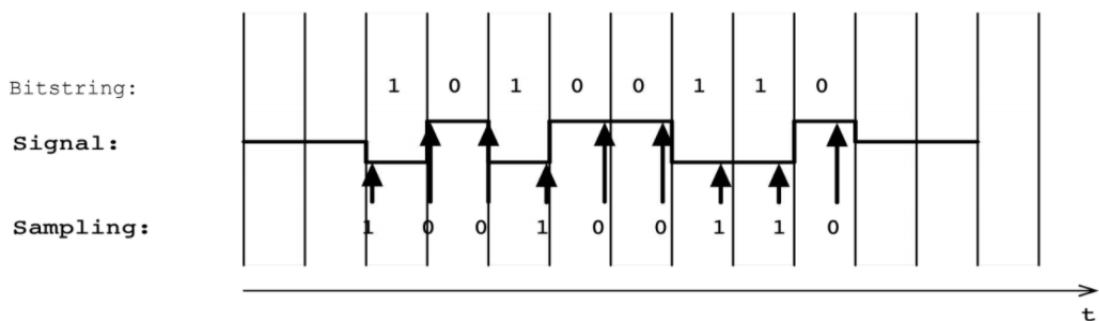


Figur 25: Graf over test data i Excel

I vores Java kode er regnemetoden nødt til at blive implementeret som en algoritme.

SERIAL KOMMUNIKATION TIL JAVA

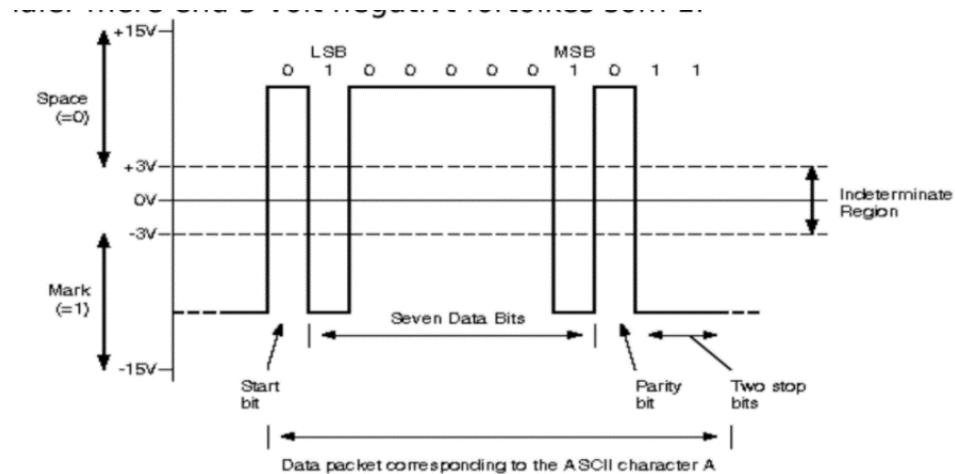
Når man skal kommunikere med ydre enheder, betyder det at man skal konvertere signalet mellem enheder, fra digitale til analoge signaler og tilbage igen. Dvs. at begge enheder skal være forbundet med et medie, og signalerne over dette medie skal være analoge. Ved afsendelse af de digitale data, bytes, skal de konverteres til analoge signaler igennem mediet, og aflæses på modtager enheder, og konverteres til et digitalt signal, altså bytes. Når man sender en serie af bits, sender man en bit ad gangen med en konstant hastighed i bits/sekund. Hastigheden bliver dog normalt defineret som baud, der er symbol/sekund, også kendt som baudrate. En baudrate kunne være f.eks. 9600, hvilket betyder, at signal lederens spænding ændre sig op til 9600 gange i sekundet. Bit hastigheden skal være den samme på modtager og afsender, ellers vil der kunne opstå synkroniseringsfejl. Data vil derved kunne blive aflæst anderledes end egentlig ment, da modtager enheden er uvidende, om hvornår der skal læse hvad.³⁰



Figur 26: Serial kommunikation³¹

Man sender normalt en start bit, en til to stopbits og eventuelt en paritetsbit. Disse bruges til at kunne vurdere, hvornår en sending, sendes og stoppes. Startbiten bliver sendt i starten, hvorimod stop bit/bitene bliver sendt til slut. Paritetsbiten bruges som en kontrol bit og kommer før stopbits. Paritetsbit kan enten være en lige paritetsbit type eller en ulige paritetsbit type, denne fungere ved enten at være et 0 eller 1. Typen betyder at biten enten giver et ulige antal 1 eller lige antal 1.

Vores serial kommunikation imellem Arduino og pc fungerer via. RS-232/EIA232. Signaler i RS-232 bliver overført som spændingsniveauer for henholdsvis 0 og 1. Dette bliver repræsenteret ved, at modtageren fortolker spændingsniveauer mellem -25V til -3V, som 1 og +3V til +25V som 0. Vi bruger en parallelport således, at vi kan sende 8 bits på hver leder. Arduino kan også sende med serielpoort, men den sender kun 1 bit pr leder, derfor bruger vi parallelport.³²



Figur 27: RS-232 ADC-kommunikation³³

ARDUINO

Arduino er en single-board computer, til at gøre elektronik mere tilgængeligt. Softwaremæssigt er compileren og bootloaderen på printkortet.³⁴

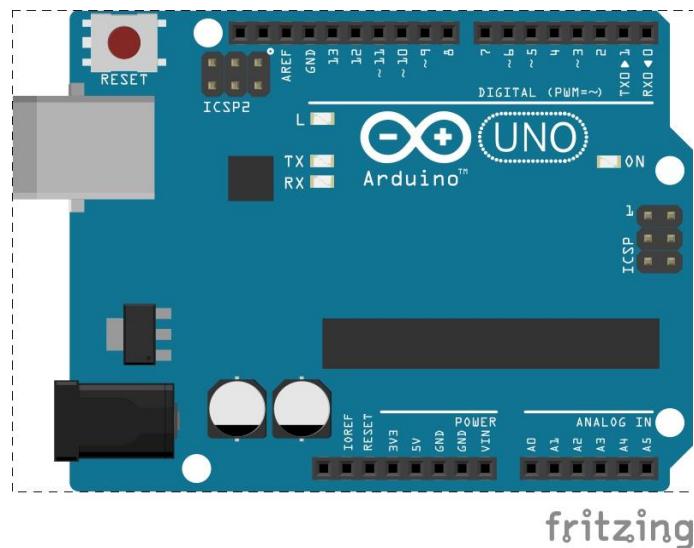
ARDUINO PROGRAM:

Når man koder et Arduino program, koder man i sproget Arduino C. Dette kodes i Arduino programmet, Arduino IDE. I dette program er main() metoden kodet for os, derfor skal man kun kode setup() og loop() metoderne.

I setup() angiver man de forskellige variabler og pinmodes, altså man sætter koden op.

I loop() angiver man så de instrukser, den skal udføre, som så køre i et loop.

ARDUINO HARDWARE



En Arduino er et board, som indeholder en microcontroller. Arduinoen er lavet sådan, at den gør det nemt for brugeren, at skrive og programmere til den microcontroller som sidder på den.

ArduinoUno bruger microcontrolleren ATmega328. Denne microcontroller har en RISC 8bit kerne (se kapitel om micro).

Arduino boardet kan deles op i underkategorierne Power, Digital og Analog Pins.

Power er de Pins, der bruges, når Arduinoen bruges som en spændingskilde. Arduinoen har en 5V, 3V og 2 jord til rådighed. Disse Pins bruges til f.eks. at give spænding til Analog sensorer.

Digital Pins er porte, hvor man gør brug af digitale komponenter, dvs. komponenter der gør brug af digitale signaler. De kan bruges både som en spændingskilde, men også som jord.

Hvis man bruger Digital Pins, som en spændingskilde til en komponent, så forsyner Arduinoen den pin med 5V. Derfor skal man ofte, når der arbejdes man Digital Pins, have eksterne modstande, da man ikke selv kan justere spændingen i Arduinoens pins

Analog Pins er porte, hvor man gør brug af komponenter, der sender Analogt signal.

Dette kan være f.eks. MAX30102, som vi arbejder med. Disse pins er benævnt med et A på boar-
det. De kan også bruges til at forsyne komponenten med spænding, eller at læse komponentens
output.

35, 36,[,]

MICROCONTROLLER

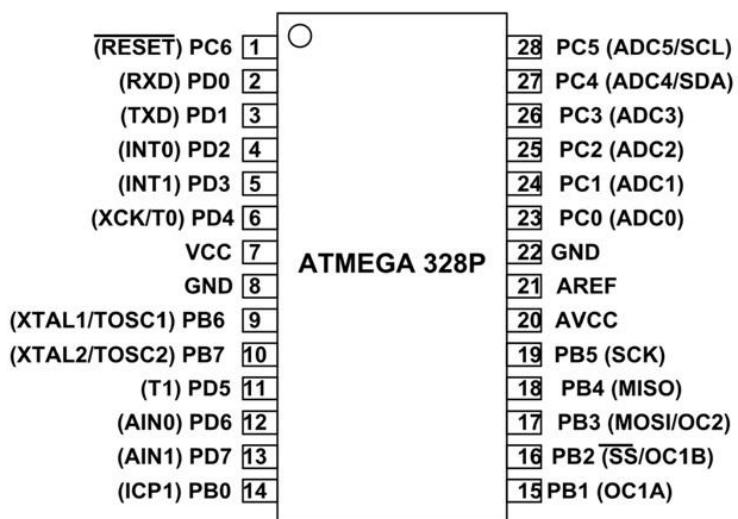
En microcontroller er et integreret kredsløb i en chip. En microcontroller er smart, fordi den består af tusindvis af små elektroniske kredsløb, hvilket gør at dens funktion kan ændres med en kode.
Det vil sige at man kan programmere til en microcontroller, og få den til at eksekvere et program.
Der findes forskellige typer af microcontrollere. Der findes RISC, hvilket er simple microcontrol-
lere, som kan eksekvere simple programmer. De er ofte hurtigere end en CISC, som er en mere
kompleks microcontroller, der kan eksekvere mere komplificerede programmer.³⁷

ATMEGA382

Når vi arbejder med vores Arduino, gør den brug af en ATmega382 microcontroller. Denne mi-
crocontroller er en simpel 8bit RISC microcontroller.

Microcontrolleren findes i to former, i en rektangulær og kvadrat form. Hvor den rektangulære har
28 pins, og den kvadrat formede har 32 pins.

Men da der visuelt kan ses, at den på vores Arduino board har 28 pins, kan vi konkludere, at det
er den rektangulære version, vi bruger



Figur 28³⁸

ATmega328 arbejder med forskellige spændinger og afhængigt af de spændinger, kan den arbejde med forskellige hastigheder.

Den kan arbejde med en spænding fra 2,7 til 5,5V. Dog kan den kun arbejde med en høj hastighed, hvis det minimum får 4,5 V.

Det vil sige at den kan arbejde med en hastighed på 8MHz med en spænding fra 2,7-5,5V.

Men kun arbejde med en hastighed på 16MHz fra 4,5-5,5V.

ATmega382 får spænding gennem VCC-porten, hvor den føres til jord gennem GND. Udover dette har den 23 programmerbar I/O lines. I/O står for Input/Output, det betyder at de porte både modtager og sender data. Dog har de 23 pins forskellige formål.

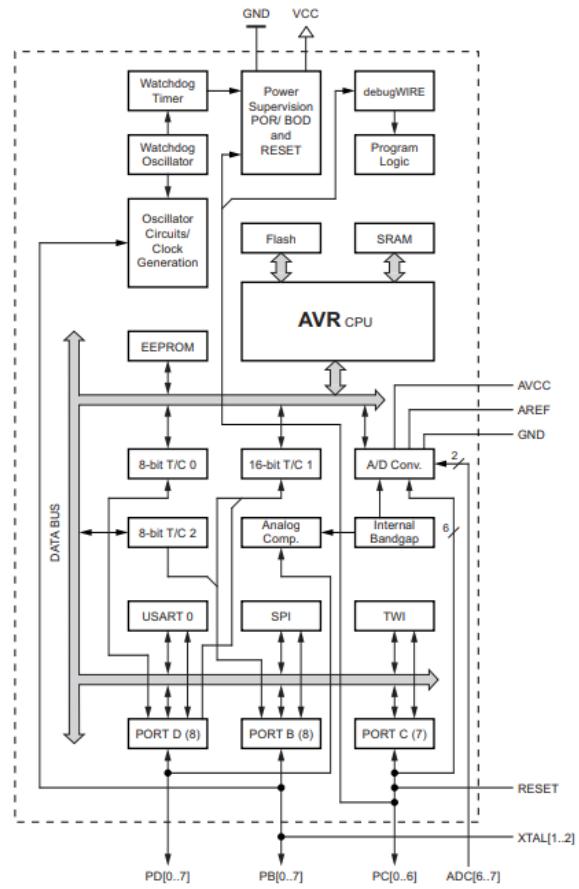
AVcc er spændingskilden til den indbyggede ADC-converter, som kan ses fra PC3-PC0.

Vores ADC-porte fungerer, som 10bit ADC. Når man har en ADC, skal den også have en reference spænding, dette fås fra AREF linjen.

Der findes yderligere tre kategorier af P porte, PB, PC og PD.

PB og PD er begge 8bit I/O porte, hvor PC er 7bit.

PB har dog også en intern RC-oscillator amp.



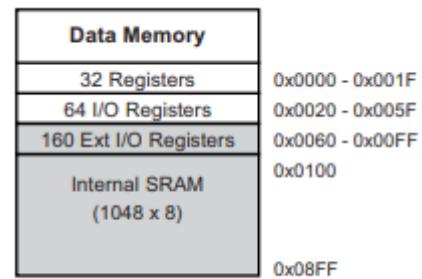
Figur 29: Integreret kredsløb for ATmega328³⁹

Denne figur viser, hvordan de forskellige porte kommunikerer med den interne Processor. Hvor de forskellige porte går ind til en databus, der sender og henter information fra AVR-cpu'en. CPU'en i vores microcontroller, har en bestemt mængde hukommelse, som er en plads at lagre den data, der kommer fra dens porte. Dette lagringsted kaldes også for SRAM.

For at sikrer, at der ikke opstår data fejl, er dens lagring delt op i to dele, dens boot up memory og dens data memory.

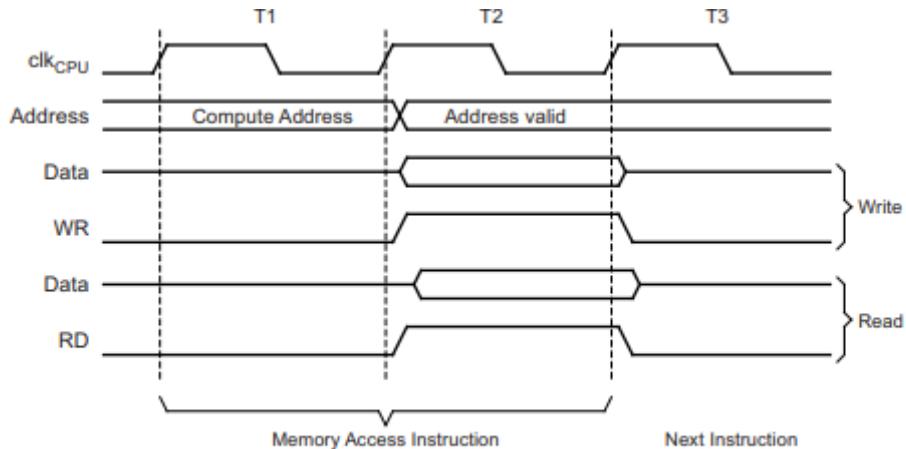
Den er opdelt på følgende måde:

32KB programmerbar hukommelse, bootuploaderen 0,5KB, SRAM har dedikeret 2KB lager også EEPROM, som har allokeret 1KB lager.⁴⁰



Figur 30: Data memory ⁴¹

På figur 30 kan man se microcontrollerens memory map. Der kan her ses, hvordan ATmega328 udover bootloaderen, deler sin hukommelse op. Den har dedikeret register til forskellige pins input, f.eks. har den 64 register til variabler fra I/O pins, hvor den også kan indstilles, til at bruge dens “extended” lager. Extended lager bruges kun, hvis der ikke er nok plads i dens normale lager.



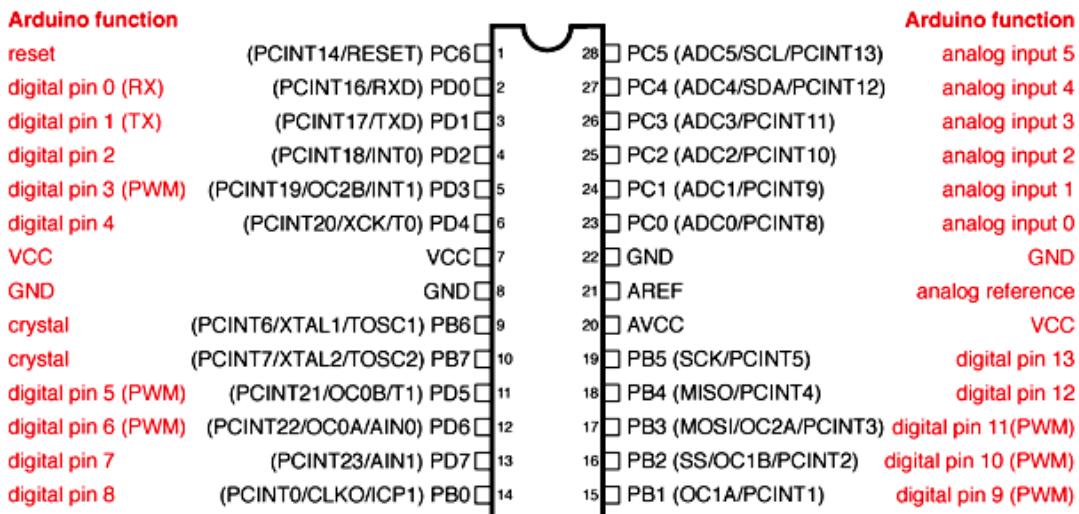
Figur 31: PWM for ATmega328 ⁴²

På figur 31, som viser controllerens PWM, kan vi se hvordan den til tiden T, som er en hel clk cycle, sender og læser data.

Det er en brugerdefineret kode, der bestemmer, om controlleren skal læse, sende eller gemme en variabel. Den kigger derfor først på den programmerbare kodes instrukser, for at vide hvilke variabler, der skal bruges til tilhørende. Den kigger først på ledige adresser i intervallet T1.

Hvor den bagefter at have valideret at pladsen er tom, går igang med sin Read og Write af data.

Det kan ses, at Read/Write funktionen bliver udført i T2. For at læse og gemme data, skal controlleren derfor igennem 2 tids cyklusser.



Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Figur 32: ATmega382 diagram⁴³

Denne microcontroller sidder på Arduinoen, derfor skal der også være kendskab til dens konfiguration.

På figur 32, kan vi se, hvordan ATmega382 er tilsluttet Arduino boardet. Her kan vi se, at analog input fører til ADC'en, PC-portene. De digitale pins, fører til PB- og PD-portene, samt vores VCC og GND er VCC og GND på Arduinoen også.

Reset pin føre frem til reset knappen på Arduinoen, som bruges til at resette programmet på microcontrolleren.

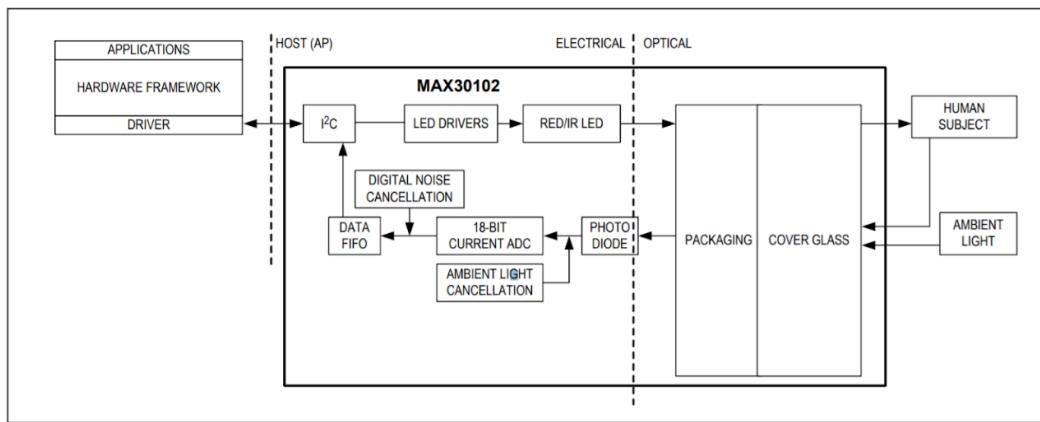
Arduinoen bruges som en forlængelse af microcontrolleren, og der kan også kodes direkte til ATmega382 microcontrolleren, uden Arduino. Det kræver dog, at man selv foretager de steps, Arduinoen gør for os.

Da vi gør brug af en Analog sensor, skal vi derfor gøre brug af ADC pins, det vil sige analog input. Hvorved vi tilslutter vores SCL og SDA stik fra sensoren direkte til microcontrolleren, ved brug af pins PC5 og PC4.

HVORDAN FUNGERER KOMPONENTEN

MAX30102 er et integreret pulsoximetri og pulsmåler modul. Den indeholder LEDs og Fotodetektorer. Komponenten fungerer således, at fotodetektorer registrerer hvor meget lys, der bliver reflekteret. Dette bliver konverteret til bits ved hjælp af dens 18-bit ADC konvertor, og bliver gemt i DATA FIFO, som er dets data lager. FIFO er et akronym for “first in first out”, og betyder kort sagt, at den data der bliver gemt først, er den data, der først bliver behandlet.

Alt dets data bliver overført til Arduinoen vha. I²C kommunikation, men man kan også overføre data fra Arduinoen til komponenten vha. I²C. Derved kan man vha. software ændre LED drivernes funktioner, til at kontrollere lysstyrken mm. af LEDs.⁴⁴

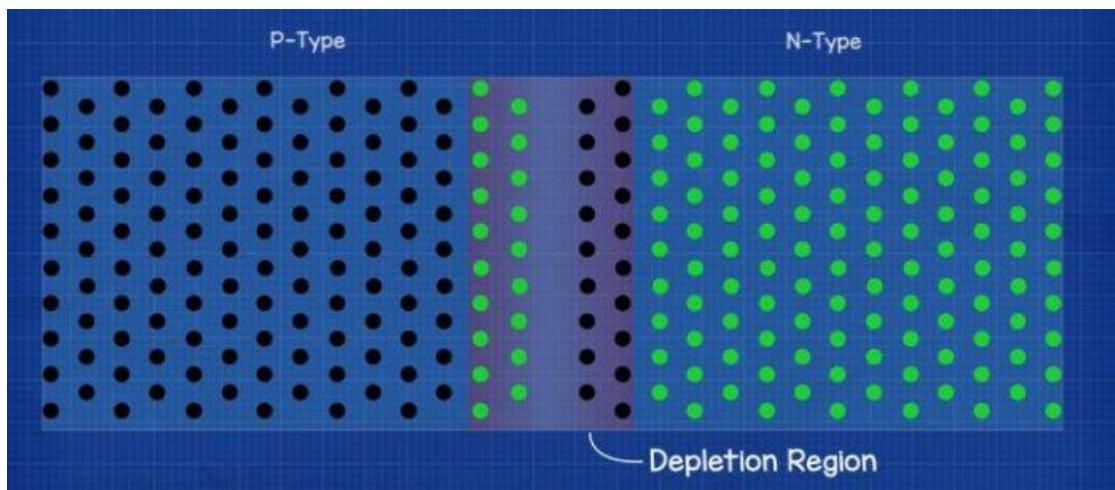


Figur 33: Kredsløb diagram⁴⁵

LED

En LED står for “Light Emitting diode”. En diode er opbygget således, at den ene ende hedder anode og den anden cathode. Disse ender begge to i små tynde plader. Imellem pladerne findes et materiale, man kalder en semikonduktorer. Semikonduktorer er forskellig fra en insulator eller en conductor, da dens insulators yderste elektronskal er tæt pakket med elektroner, og en conductors yderste elektronskal har en til tre elektroner. En insulators yderste elektronskal ville også ligge langt fra dets conductor band, hvor en conductor ville ligge tæt på dets conductor band. En semikonduktorer er anderledes, da dens yderste elektronskal har en antal af elektroner, der gør, at den er et sted imellem en konduktor og en insulator, samt at dets conductor band ligger tæt på den yderste skal. Dette gør, at hvis man tilfører energi, vil elektroner springe ud af den yderste skal, og medfører, at en semiconductor nu vil reagere som en conductor. I en diodens semikonduktorer, vil der være en P- og N-del. P-delen på anoden og N-delen på cathoden. Havde semikonductoren

været den hyppige brugte semiconductor silikone, som har fire elektroner i yderste skal, ville silikone atomer vha. kovalente bindinger, fylde deres yderste skal op. Derfor tilføjer man andre atomer i henholdsvis P- og N-delene, således at der i N-delen, vil være for mange elektroner, og derved ville kunne bevæge sig frit imellem elektronskallerne. I P-dellen sker det modsatte, så der nu mangler nogle elektroner. Imellem skallerne bliver der nu dannet et rumladningsområde, dette skaber, at der vil være en negativ region i P-dellen og en positiv region i N-delen, som skaber et elektrisk felt, der gør at de negative elektroner i N-delen ikke kan komme til P-dellen, medmindre der bliver tilført energi. Når man tilfører en energi, der vil være større end det potentielle, der bliver skabt af det elektriske felt, vil rumladningsområdet blive mindre, og elektronerne vil kunne springe over, og strømmen vil løbe igen.



Figur 34: Semikonduktør P og N type⁴⁶

I en LED diode bruger man to materialer, der frigiver lys med fotoner, som sine semikonduktorer. Lysfarven kan ændres ved ændringer i den mængde af energi, der bliver frigivet, som er afhængigt af materialet, fordi energi og bølgelængder er forbundet med hinanden.

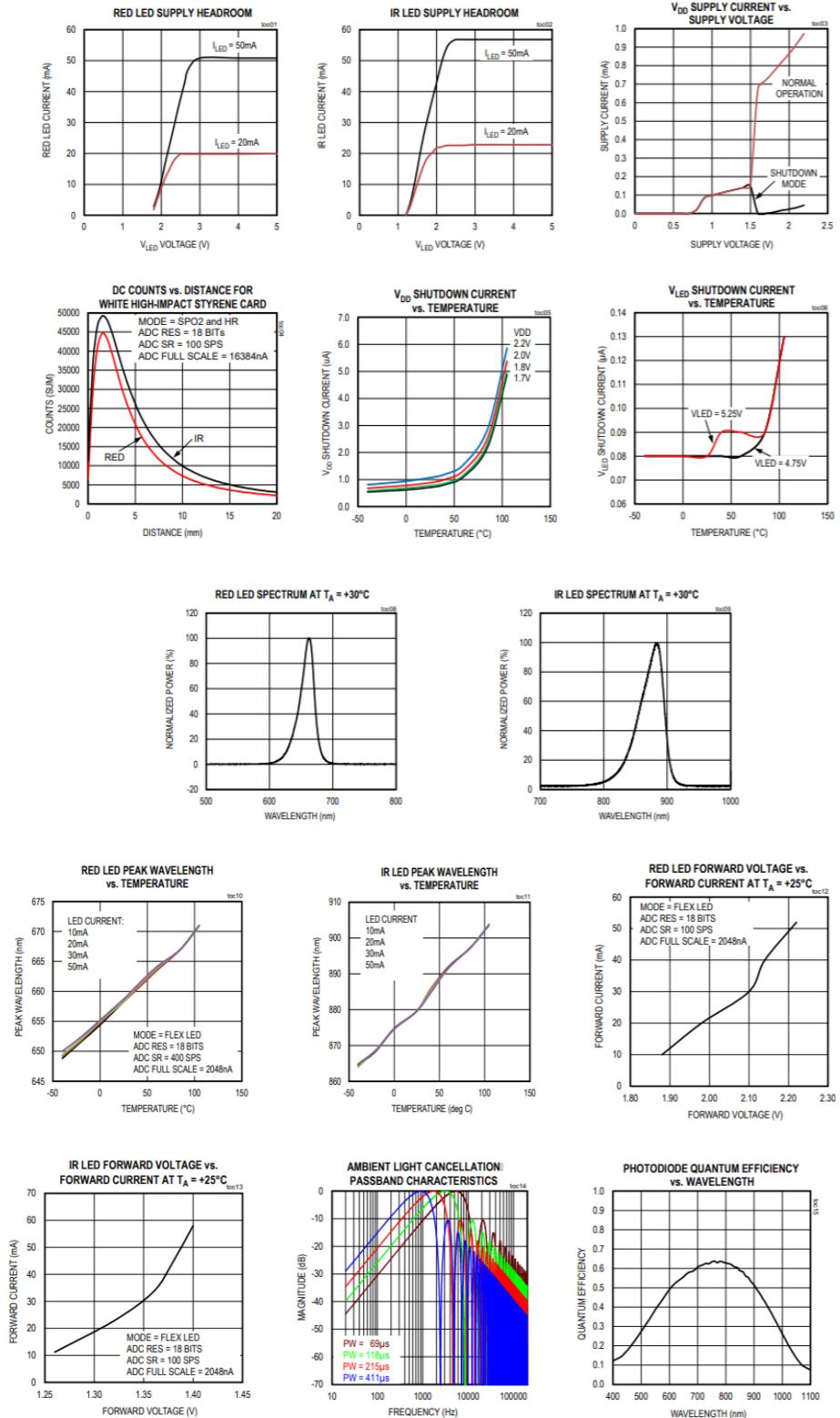
$$E=hc$$

Dvs. at når elektronerne springer, bliver der frigivet energi, og pga. materialet bliver denne energi frigivet som lys.⁴⁷

Rød LED og IR LED

MAX30102 kan kontrolleres med integrerede LED drivers. Driverne kan kontrollere strømmen, der bliver tilført til LED-dioden i et spektrum mellem 0 til 50mA samt puls bredden fra 69μs til

$411\mu\text{s}$. Dette betyder, at man kan optimere SPO₂ målingerne, da en temperaturforøgelse af både IR- og Red-LED, ændre bølgelængden for dioderne.



Figur 35: LED karakteristika ⁴⁸

Bølgelængden skal gerne være nogenlunde præcis, fordi formlen for SPO₂ beregningen fungere således, at AC(660)/DC(660)AC(940)/DC(940)=R og R værdien kan oversættes til SPO₂ således.
 $104-17 \cdot R = SPO_2$

Her skal der tages forbehold for, at forskellige komponenter skal kalibreres på forskellige måder. Derfor skal bølgelængden af både IR og RED LED være præcise, hvis ens SPO₂ måling skal være præcis. ⁴⁹

FOTODETEKTOR

Da man ikke kan finde hvilken optiske detektor, der er brugt til at lave MAX30102 komponenten, må man kort forklare, hvordan en optisk detektor fungere.

En optisk detektor er også kaldet en photodiode, og fungere ligesom dioden forklaret i “LED”. Forskellen er dog, at hvor vores LED diode omsætter strøm til lys vha. fotoner, imens vores optiske detektor omsætter lys til strøm. Dette gør den ved, at fotoner laver elektronpar i p-regionen, den region med “huller”. I vores tilfælde bliver LED lyset reflekteret i en finger og dens væv.⁵⁰

Man kan læse i MAX30102 databladet, at vores fotodetektor har nogle karakteristika, som f.eks. “Spectral range sensitivity” på (QE>50%) mm. Hvor QE står for “Quantum efficiency” og betyder:

“Quantum efficiency: The number of carriers (electrons or holes) generated per photon.”⁵¹

18-BIT ADC

I vores sensor sidder der en 18bit ADC-konverter.

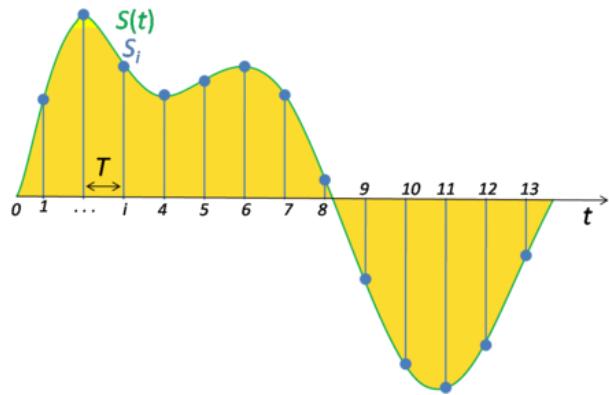
En ADC converters funktion er at lave et analogt signal om til et digitalt signal.

Dette gøres fordi, når der arbejdes med analoge signaler, fås der en kurve med uendeligt mængder data, voltage outputs, hvor der ved digitalt signal arbejdes binært, så der arbejdes med enten on eller off. Dette gør, at man kan arbejde med sensorer, microcontrollere og andre komponenter mm. som arbejder analogt, hvor man kan lave det om til digitale signaler, og bruge det til behandling i et program.

Når man arbejder med en ADC, skal du først tage forbehold for, hvad din sampling rate skal være, ift. hvor mange målinger du gerne vil have.

Lad os antage vi har en sinuskurve, som repræsenterer et analogt signal. Hvis man har en ADC-converter, og man sampler med en lav frekvens, så vil den sample for lidt, og derfor vise et signal man ikke kan bruge.

Desto flere gange man sampler, desto mere præcis er ens data. Dog kan væsentligheden af ens sampling diskuteres.



Figur 36: Sampling diagram⁵²

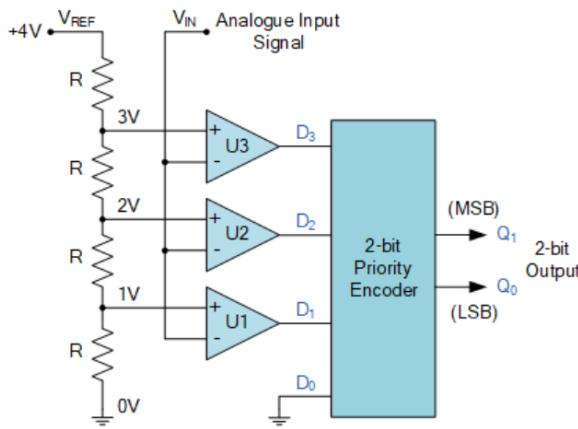
Når man arbejder med en ADC, skal man også tage højde for dens opløsning. Hvilket er den størrelse af bits vores ADC-konvertere.

Dette kan man regne ud med formlen; 2^{n-1} , hvilket er mængden af komparator man skal bruge for et n-bit størrelse resultat. Det vil sige, at hvis man arbejder med et 2bit ADC, skal man bruge 2^{2-1} antal komparator, hvilket er 3 komparator.

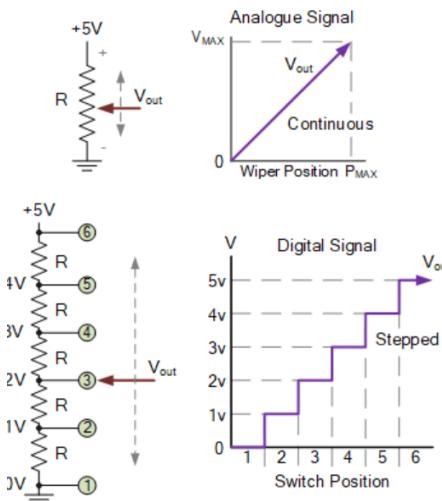
Derfor hvis vi ser på vores sensor, der har en 18 bit ADC, har den 2^{18-1} antal komparator.

Årsagen til, at man kan bruge komparator til en ADC-converter, er fordi en komparator fungere som et filter. Den har to inputs samt en reference spænding, hvor den derfra sampler inputtet i forhold til dens reference spænding, så outputtet kun bliver HIGH når komparatorens input er højere end referencespændingen.

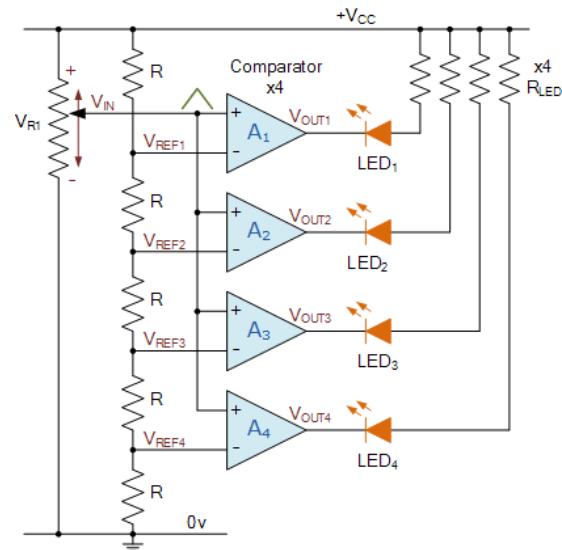
Derfor har komparatorerne i en ADC-converter forskellige referencespændinger, så hver komparator har forskellige output. En 2bit ADC har 3 komparator, derfor har den 3 forskellige output, 01,10 og 11. Der findes også 00, dette er output, når ingen komparator output er HIGH.



Figur 37: ADC



Figur 38: ADC⁵³



Figur 39: Spændingsdeler⁵⁴

Hvis vi ser på kredsløbet af en ADC og en voltage level detector, kan vi se, at de er ens.

En ADC gør brug af samme princip. Den tager nemlig spændingen, og deler den op ift. de forskellige referencespændinger. Så den første komparator har en VREF på $1/5V_{CC}$, den anden har en VREF på $2/5V_{CC}$ osv.

På denne måde kan man nemlig dele outputtet op afhængigt af V_{CC} -spænding.

DATA FIFO

FIFO står for first in first out, dette betyder at den data, der først kommer ind, er også den data, der først bliver behandlet og sendt ud. Dvs. at når vores sensor sender data til Arduinoen, er det den data, der først blev sendt til Arduinoen, der bliver behandlet af den.

Når vi foretager målinger og henter dem i vores JAVA program, så er dataen fremvist i den rækkefølge, Arduinoen modtog dataen fra sensoren.

Derfor kan man også se, at når vi slukker programmet, mens sensoren måler. Næste gang vi åbner programmet, vil de dataer, som stadig er i Arduinoens SRAM, være dem der bliver vist, forinden vi får de nye målingers data. Derfor er den første måling frataget vores målinger.

I2C KOMMUNIKATION

MAX30102 indeholder en I2C kompatibel system management bus, der indeholder ”2 wire serial interface” bestående af en serial data line (SDA) og en serial clock line (SCL) med en clock hastighed på op til 400kHz.

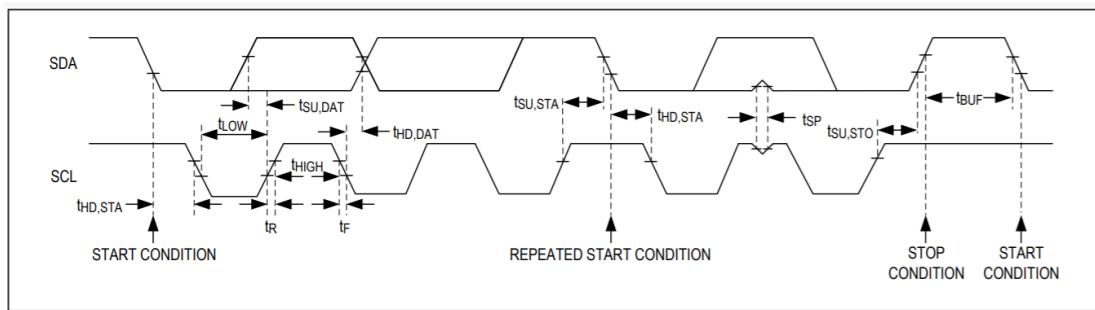
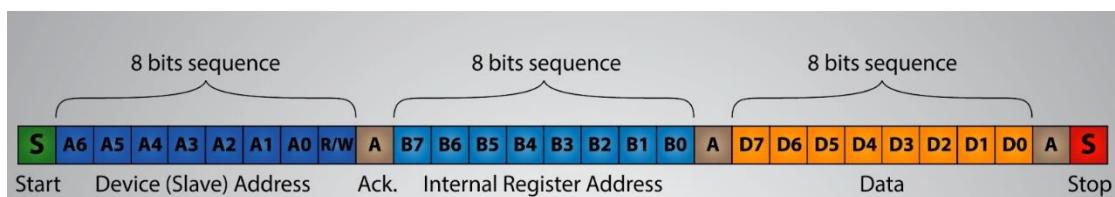


Figure 1. I²C-Compatible Interface Timing Diagram

Figur 40: I²C diagram ⁵⁵

I2C er en data protocol, der til overførsel af data og kommunikationen, fungerer således. Man sender først en start bit, dernæst en 8 bit sekvens, hvor den sidste bit er gemt til read/write. Denne bit sekvens bruges, til at allokerer en plads på slave enheden, hvorved der enten kan læses fra eller skrives data til. Dernæst sendes der en acknowledgement bit, for at vurdere, om 8 bit sekvensen er korrekt læst. Efter den kommer der en ny 8bit sekvens, som beskriver hvor på master enheden, dataen skal hentes til eller hentes fra. Igennem en acknowledgement bit, efterfulgt af en 8 bit data sekvens, med den aktuelle data, og til sidst en acknowledgement bit mere og en slut bit.⁵⁶



Figur 41: I²C kommunikation ⁵⁷

Ved MAX30102 komponenten er hvert ord 8 bit langt, efterfulgt af en acknowledgement clock pulse. Dvs. at når en Master læser data fra MAX30102, bliver der overført den rigtige slave

adresse, efterfulgt af en serie af 9 clock pulser. Dvs. at MAX30102 overfører data vha. SDA parallelt med den clock puls, der bliver generet af masteren. Hvilket betyder at en kommunikation nogenlunde ville se således ud:⁵⁸

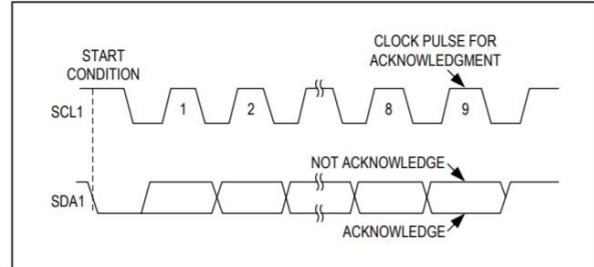
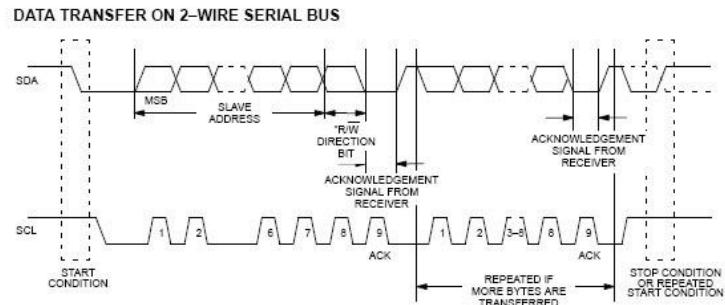


Figure 8. Acknowledge

Figur 42: I2C Diagram⁵⁹

Dette diagram viser dog lidt bedre, hvordan en generel dataoverførsel på en 2-wire serial bus fungerer, der skal dog bemærkes, at det ikke kommer fra MAX30102 databladet.



Figur 43: I2C Data transfer diagram⁶⁰

KONFIGURATION

Vi vælger at sætte sensor.setup() konfigurationen således:

```
byte ledBrightness = 0x1F; //Options: 0=Off to 255=50mA
byte sampleAverage = 2; //Options: 1, 2, 4, 8, 16, 32
byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green
int sampleRate = 400; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200
int pulseWidth = 411; //Options: 69, 118, 215, 411
int adcRange = 4096;
sensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth, adcRange);
```

Der hvor vi viger fra standardindstillingerne, er beskrevet herunder.

```
// Setup the IC with user selectable settings
void setup(byte powerLevel = 0x1F, byte sampleAverage = 4, byte ledMode = 3, int sampleRate = 400, int pulseWidth = 411, int adcRange = 4096);
```

Vores komponent indeholder ikke en grøn LED, derfor har vi kun ledMode på 2.

Udover dette kan man læse i tabellerne fra MAX30102 databladet, hvad værdierne betyder for komponenten.

Screenshot herunder er i rækkefølge:

powerLevel:

Table 10. Temperature Integer

REGISTER VALUE (hex)	TEMPERATURE (°C)
0x00	0
0x01	+1
...	...
0x7E	+126
0x7F	+127
0x80	-128
0x81	-127
...	...
0xFE	-2
0xFF	-1

Figur 44: Tabellerne kommer alle samme kilde⁶¹

sampleAverage:

Table 3. Sample Averaging

SMP_AVE[2:0]	NO. OF SAMPLES AVERAGED PER FIFO SAMPLE
000	1 (no averaging)
001	2
010	4
011	8
100	16
101	32
110	32
111	32

ledMode:

Table 9. Multi-LED Mode Control Registers

SLOTx[2:0] Setting	WHICH LED IS ACTIVE	LED PULSE AMPLITUDE SETTING
000	None (time slot is disabled)	N/A (Off)
001	LED1 (Red)	LED1_PA[7:0]
010	LED2 (IR)	LED2_PA[7:0]
011	None	N/A (Off)
100	None	N/A (Off)
101	Reserved	Reserved
110	Reserved	Reserved
111	Reserved	Reserved

sampleRate:

Table 6. SpO₂ Sample Rate Control

SPO ₂ _SR[2:0]	SAMPLES PER SECOND
000	50
001	100
010	200
011	400
100	800
101	1000
110	1600
111	3200

pulseWidth:

Table 7. LED Pulse Width Control

LED_PW[1:0]	PULSE WIDTH (μs)	ADC RESOLUTION (bits)
00	69 (68.95)	15
01	118 (117.78)	16
10	215 (215.44)	17
11	411 (410.75)	18

adcRange:

Table 5. SpO₂ ADC Range Control (18-Bit Resolution)

SPO ₂ _ADC_RGE[1:0]	LSB SIZE (pA)	FULL SCALE (nA)
00	7.81	2048
01	15.63	4096
10	31.25	8192
11	62.5	16384

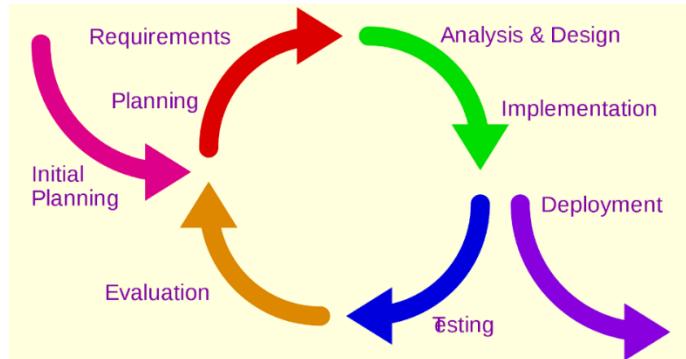
På tabellen herunder kan man se, at når vi kører med default setup, er vi indenfor de lovlige indstillinger på SPO₂ mode, da vi bruger både den røde og den infrarøde LED:

Table 11. SpO₂ Mode (Allowed Settings)

SAMPLES PER SECOND	PULSE WIDTH (μs)			
	69	118	215	411
50	O	O	O	O
100	O	O	O	O
200	O	O	O	O
400	O	O	O	O
800	O	O	O	
1000	O	O		
1600	O			
3200				
Resolution (bits)	15	16	17	18

UML

Når man skriver software, er det rigtig smart at planlægge, før man går i gang. Derfor arbejder vi med Iterativ programudvikling.



Figur 45: Iterativ programudvikling⁶²

Til planlægningen af vores software program, bruger vi Unified Modeling Language, til at planlægge vores system, før vi går i gang. Dette gør vi, fordi det skaber en bedre struktur, samt giver et overblik, over de ting der skal laves.

KRAVSPECIFIKATION

Vi skal lave et Java program til en opstartsvirksomhed, lavet af nogle lektorer fra KP og DTU.

Vi har fået givet en sensor fra dem, og har fået til opgave at undersøge, om denne sensor kan måle SpO2 og puls.

På baggrund af de krav vi er blevet stillet, har vi valgt at sætte nogle krav til vores program.

Vi skal have lavet et program, som kan udregne SpO2 og puls ud, med de data vi får gennem vores Arduino program. Derfor skal vi bruge I2C og RS232 kommunikation. Der skal derfor tages forbehold for de eventuelle kommunikationsfejl, der kan opstå.

For at kunne få noget inddata, skal vi lave et Java program, der kan læse fra en seriell port. Dernæst skal vi kunne opdele denne data, så vi kan differentiere mellem IR og R værdierne.

Når dette er sket, vil vi gerne kunne beregne SpO2 værdien og pulsen.

Efter at have målet SpO2 og puls, skal den også kunne være tilgængelig igen, når programmet lukkes. Derfor skal den også gemme målingerne, der bliver foretaget, samt resultaterne i en fil.

ANALYSE

Vores Arduino sender data til vores Java program. Vi ved at den minimum sender to data strenge, en IR og RED værdi. Yderligere ved vi, at hver af de værdier er 6 cifre.

Det vil sige at vi sender 2×6 cifre i bits. Hvor det 6 cifrede tal, hvis man gør brug af 2 kompliment, har en størrelse på 18 bits. Det vil sige at for hver værdi, så sender vi 18 bits.

Vi skal dog have en måde at skelne mellem IR og R værdierne, og der er jo nogle forskellige måder at gøre det på.

Først så kunne man adskille de to tal ved brug af substring, da vi ved at hvert tal er 6 cifre, så derfor kunne man tage de første 6 cifre som en måling, og de andre 6 cifre som en anden måling. Dette kommer ikke til at virke, da vi kan se i Serial Monitoren i vores Arduino program, at den kun sender et 6 cifret tal, når vi har en finger i sensoren. Dvs. at den vil kunne blande tallene sammen, hvis der ikke er en finger i, eller hvis den sender et tal som ikke er 6 cifret.

Derfor kan man gøre det, at man sender et skiltegn, mellem hver af værdierne.

f.eks. kunne man sende IR og så et vilkårligt tegn, derefter R og så et andet vilkårligt tegn.

Dette vil gøre, at man så vil kunne skelne mellem hver enkelte streng af data sendt, og hver enkelte IR og R værdi. Denne metode er mere pålidelig, da vi så er uafhængige af IR or R værdiernes størrelse, da vi kun leder efter vores skiltegns, som altid er de samme.

SpO₂:

SpO₂ kan vi regne ud, med formlen $SpO_2 = 104 - 17 \cdot R$

$$\text{hvor } R = (ACr/DCr)/(ACir/DCir)^{63}$$

Her kan man finde R ved brug af bubblesort, hvor man derefter har max på array[array.length] og min på array[0].

Der kan også bruges en metode som finder max, min samt average værdien af et array. Det gøres ved brug af arraystream.max(), min() eller average()

Der skal derfor vurderes hvilken metode er bedst. Dog regner vi med at average() er bedre, da det er en mere effektiv kode, og at DC er medianen, så det må være middelværdien af alle målinger.

Puls:

Algoritmen lyder således: en forklaring står under test kapitlet i elektronik

$$\Delta y_2 - y_1 \leq -30 \quad \& \quad \Delta y_3 - y_2 \leq \Delta y_2 - y_1 \quad \&& \text{Gemtpuls(1 og 2)} \leq -27$$

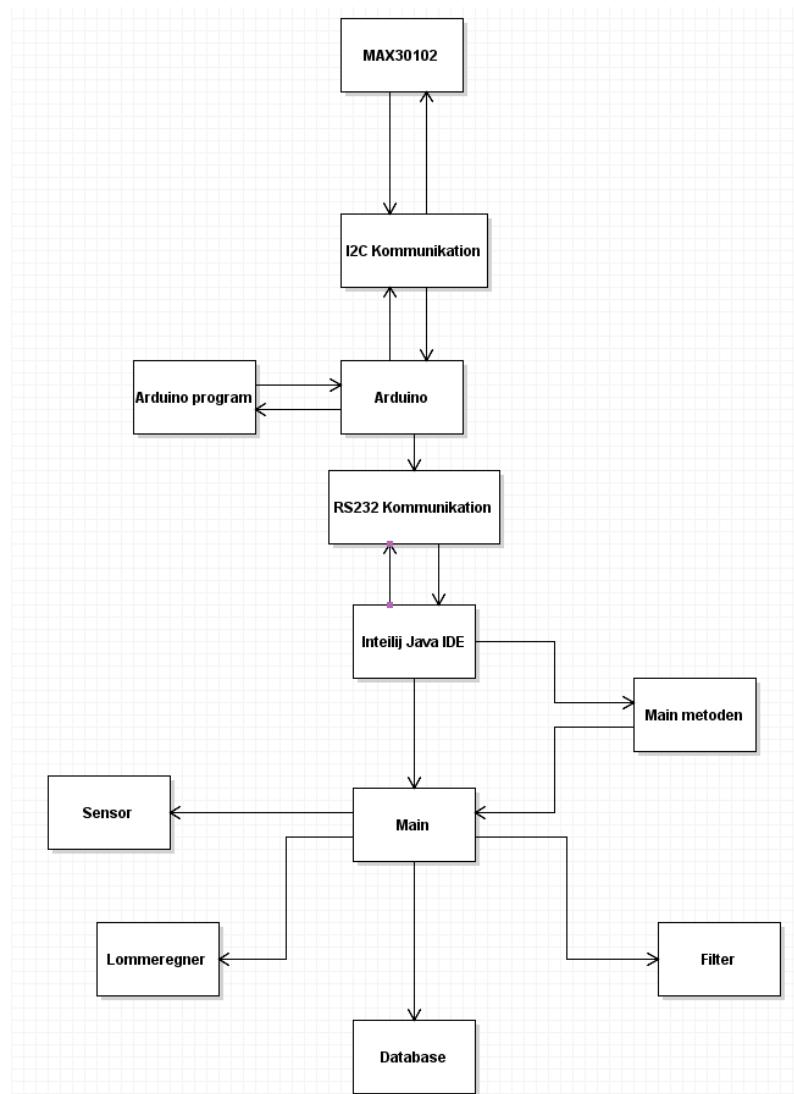
Hvis ovenstående er rigtigt tæller vi 1 op.

Efter at vi har talt til 10 toppunkter, tjekker vi hvor lang tid dette tog, og bruger regnestykket $60.000 / (\text{tæller} - 1) / \text{tid} = \text{BPM}$

til at udregne pulsen. Dette er dog kun en approksimation af 10 pulsslag.

Vores Usecase

Der findes en person, der skal have overvåget sin SPO₂ og puls. Dette gør vi ved at sætte vores pulsoximeter fast på hans langfinger. Nu sender vi et infrarødt og rød LED lys på hans finger, og mäter på hvor meget der bliver reflekteret. Ud fra dette kan vi lave en approksimation af SPO₂ og puls. Disse værdier viser vi på skærmen, så patienten selv kan se hans SPO₂ og BPM. Vi viser altså ikke de rå målinger. Vi gemmer de rå data SPO₂ og BPM i en fil, således at de kan kontrolleres senere.



Figur 46: Analyse-klassediagram 0.1

Hvis man kigger på figur 46, ses et overordnet analyse-klassediagram for alle elementer.

I dette klassediagram ser vi på hele kredsløbet, og ikke kun vores JAVA program.

Derfor er Arduinoen og dets klasse også inkluderet.

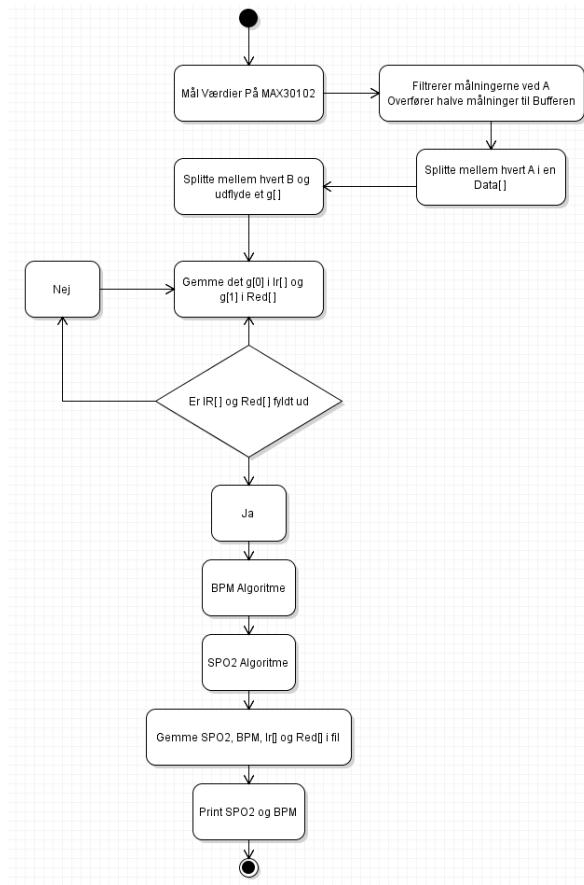
Det vi gerne vil komme frem til, ved at lave vores program med denne opstilling er, at vi har en Arduino, der sender data via RS232. Derefter har vi Main klassen, der kører main(), som kalder på de andre klasser. Så i Sensor klassen bliver Arduinoens data hentet, i Filter klassen bliver den data delt og filtreret, i Lommeregner klassen beregner vi SpO2 og puls, og i Database skriver vi målingerne i en fil.

På denne måde mener vi, at vi får delt programmet op i fornuftige klasser, hvor hver klasse har en funktion, der er væsentlig, for at vores program kan fungere.

DESIGN

Aktivitets diagram:

På nedenstående Aktivitetsdiagram ser man, at vi starter med at måle nogle værdier på MAX30102 komponenten, derefter filtrere vi målingerne ved A, således at hver måling bliver adskilt. Hvis der er nogle halve målinger, gemmer vi dem i bufferen. Dernæst deler vi hver måling i et array hver for sig. Dette gøres nu igen bare ved B i stedet for A, så målingerne bliver adskilt i infrarødt og rødt lys. Dem lægger vi i hvert deres array, og kontrollere om de er fyldt. Hvis de ikke er fyldt, fylder vi dem mere op. Derimod hvis de er fyldt, bruger vi vores algoritmer til at udregne BPM og SPO2, og derefter gemmer alt data i en. Denne sekvens gentager sig så uendeligt, eller indtil brugeren stopper programmet.

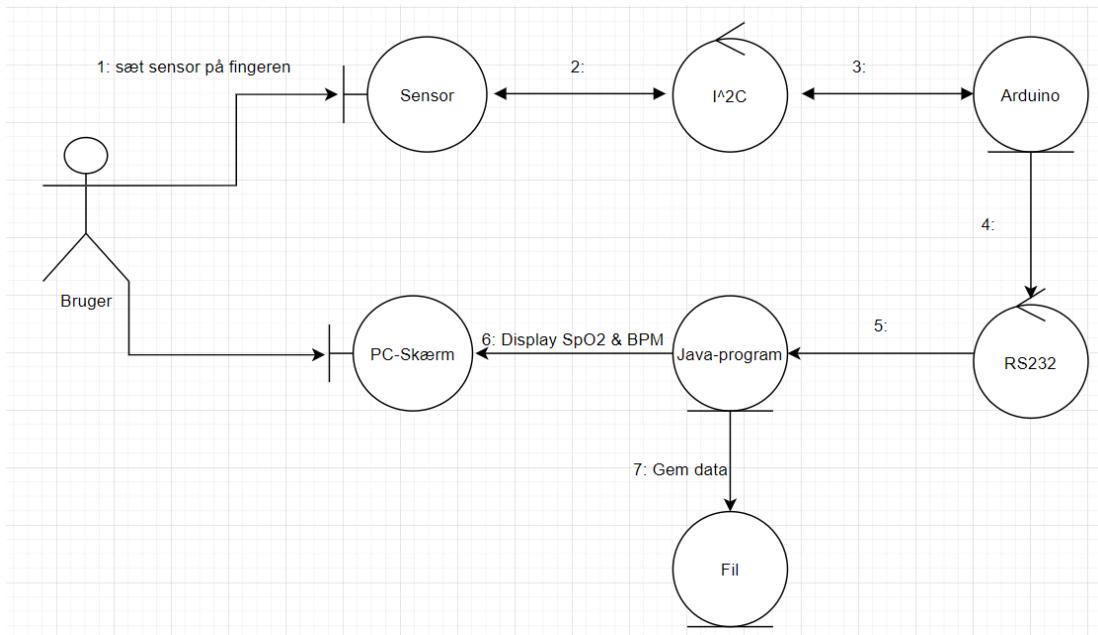


Figur 47: Aktivitets diagram iteration 0.1

Kommunikationsdiagram:

På nedenstående Kommunikationsdiagram ser man, hvordan tingene kommunikere med hinanden.

Der vises her, hvordan brugeren skal sætte fingeren på sensoren, også vil alt arbejdet ske inden bag i programmet, hvor brugeren derefter vil kunne læse værdierne på skærmen.



Figur 48: Kommunikationsdiagram iteration 0.1

Main

- Main()

Dette er hvor vi kører vores main metode. Da vi gerne vil have, at vi kan måle uendeligt

Sensor

- Sensor()

Dette vil være vores SerialPort konstruktør, som åbner, og sætter portens parametre.

- maaling()

I denne metode vil vi læse, hvad der kommer igennem serialporten, og derefter gemme det.

Filter

- Filtrering()

I denne metode, skal vi have splittet dataet for to skiltegn, samt dele den op i et array for infrarød og en array for Rød LED værdierne.

Lommeregner

- SpO2()

Dette vil være vores SpO2 algoritme

- Puls()

Dette vil være vores algoritme til at beregne pulsen.

DataBase

- DataBase

Dette er vores konstruktør, hvor vi laver en fil og en filewriter objekt.

- data()

Denne metode vil skrive vores IR og RED værdier i en fil

- resultat()

Denne metode vil skrive vores puls og SpO2 i en fil

Design-klassediagram:

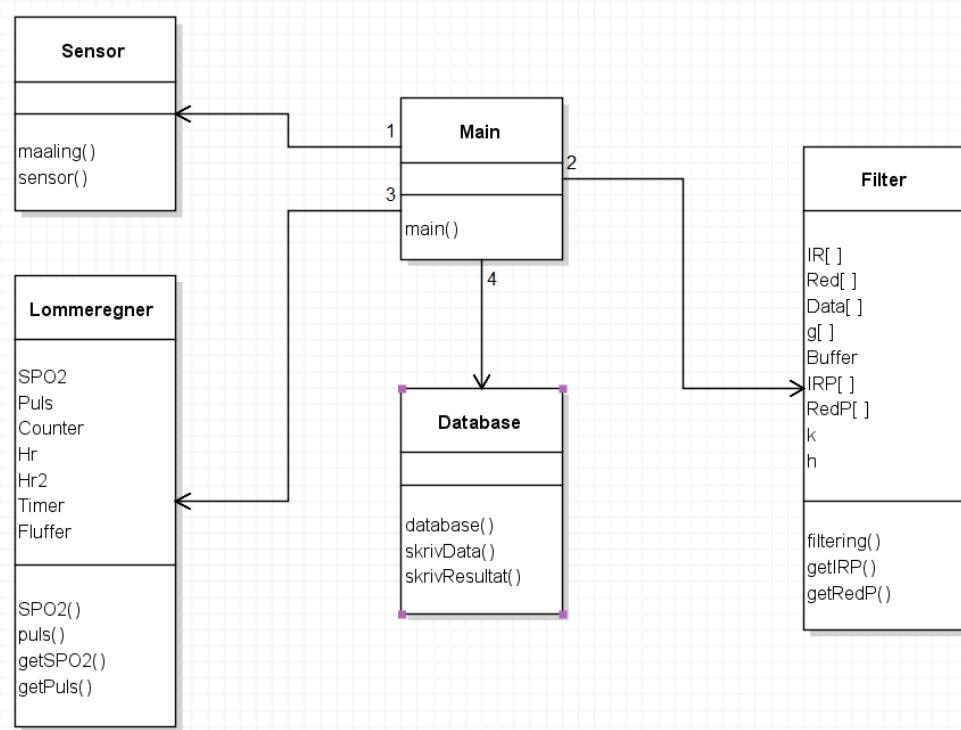
På nedenstående Design-klassediagram ser man, hvordan de forskellige klasser i vores Java program ser ud. Tager man først Sensor klassen, ser man at den er attribut løs, men indeholder metoder “maaling()” og “sensor()”, hvor sensor metoden er konstruktøren, og maaling metoden bruges til at læse målingerne fra Arduino.

Vi har Filter klassen som indeholder attributterne: IR[], Red[], Data[], g[], Buffer[], IRP[], RedP[], k og h. Alle array i denne klasse bruges til filtreringen/håndteringen af daten ud i separate array. IRP[]og RedP[] er de endelig arrays og indeholder henholdsvis IR og Red målinger fra komponenten i variabeltypen doubles. Variablerne k og h er integers, og bruges forskellige steder til optælling. Klassen indeholder også metoderne “filtering(), getIRP() og getRedP()”, hvor filtering metoden er den metode, der filtrerer dataen. De andre metoder såsom getIRP og getRedP er getters, som bruges til at hente værdien fra klassen.

Lommeregner klassen indeholder attributterne SPO2, puls, counter, hr, hr2, timer og fluffer. SPO2 og puls attributterne, er de endelige beregningsresultater, og resten bliver brugt til beregningen af pulsen. Klassen har metoderne “SPO2(), puls()”, getSPO2() og getPuls()”, hvor SPO2 og puls klasserne indeholder algoritmer til beregning af deres relevante værdi. Ingen bruges der get-metoderne getSPO2 og getPuls, til at hente værdierne fra klassen.

Database klassen indeholder ingen attributter, men indeholder metoderne “Database(), skrivDatabase() og skrivResultat()”. Her er Database() metoden konstruktøren, skrivData udskriver IR og Red værdierne til en fil, og skrivResultat skriver resultatet af SPO2 og puls beregningerne til filen.

Til slut har vi Main klassen der kun indeholder metoden main(), som kører hele programmet.

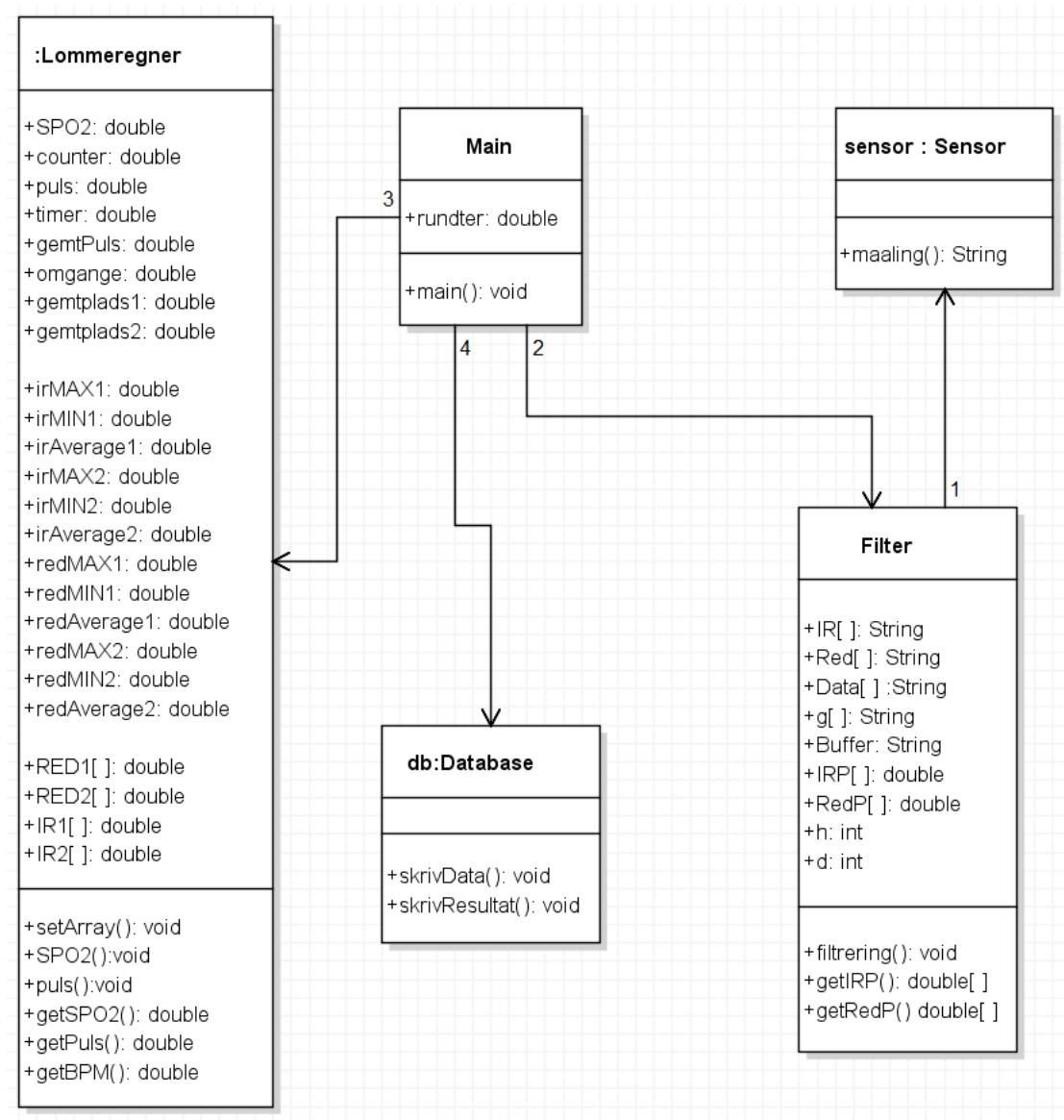


Figur 49: Design-klassediagram iteration 0.1

IMPLEMENTATION

I dette implementations-klassediagram kan vi se, hvordan vores endelige kode ser ud. Det kan ses med pilene, hvordan der behandles data. Der ses at vi læser fra sensoren, dernæst filtrere vi de data vi har læst. Hvor vi bagefter sender det til Lommeregner, regner på dem, og til sidst gemmer i vores Database.

Dette er en stærk repræsentation af vores aktivitetsdiagram, hvor vi gerne ville køre processen i samme rækkefølge.



Figur 50: Implementationsdiagram iteration 0.1

Main Klasse

Attributter

```

1 | public class Main {
2 |     static double rundter = 0;
  
```

- **rundter** bliver brugt som en tæller, så vi ikke bruger første måling, som ofte er forkert.

metoder

```

3
4     static Filter ft = new Filter();
5     static Lommeregner lr = new Lommeregner();
6     static DataBase db = new DataBase();
7
8 ▶   public static void main(String[] args) {
9       while (true) {
10         ft.Filtrering();
11         if (rundter >= 1) {
12             lr.setARRAY(ft.getREDP(), ft.getIRP());
13             lr.SP02();
14             lr.Puls();
15             System.out.println("SP02: " + lr.getSP02() + "%");
16             for (int i = 0; i < 500; i++) {
17                 db.skrivData(ft.IRP[i], ft.REDP[i]);
18             }
19             db.skrivResultat(lr.getSP02(), lr.getPuls(), lr.getBPM());
20         }
21         rundter++;
22     }
23 }
24

```

- I main metoden, bliver der kørt et uendeligt loop, while(true), hvor vi køre metoderne fra de andre klasser. Dette gør at vi i hvert loop, får kørt en måling, filtrering og beregning.

Sensor Klasse

metoder

I vores sensor klasse åbner vi serielparten i vores konstruktør, hvor vi har indstillet de parametre, vi bruger.

```

1  import jssc.SerialPort;
2  import jssc.SerialPortException;
3  // importere jssc librariet
4
5  public class Sensor {
6      private SerialPort sensor = new SerialPort( portName: "COM6"); // laver et sensor objekt
7      public Sensor() { /*laver en konstruktur, hvor vi indstiller vores parametre. Sensor konstruktur åbner porten
8          og sætter parameterne
9          JSSC bibliotek kommer fra https://github.com/java-native/jssc/releases\*/
10     try {
11         sensor.openPort(); //prøve at åbne porten();
12         sensor.setParams( baudRate: 115200, dataBits: 8, stopBits: 1, parity: 0); //sætter parametre
13         sensor.setFlowControlMode(SerialPort.FLOWCONTROL_NONE); // sætter ingen flowkontrolle, altså ingen begrænsning
14         // på data modtagelse.
15
16         sensor.purgePort( flags: SerialPort.PURGE_TXCLEAR | SerialPort.PURGE_RXCLEAR); //prøver at slette data, hvis
17         // der er data fra forrige læsning
18     } catch (SerialPortException ex) { // catcher vi serialportexception
19         System.out.println("FEJL SERIALPORTEXCEPTION");
20     }
21 }
22

```

Dernæst har vi maaling() metoden, som helt simpelt kontrollere, om der bliver læst noget fra serielparten. Læses en måling fra serielparten, returneres den i en string ellers returneres null, hvis der ikke bliver læst noget.

```

23     public String maaling() { //dette er vores metode til at læse fra porten
24
25         try {
26             if (sensor.getInputBufferBytesCount() > 0) {// her kontrollere vi at der læses noget.
27                 return sensor.readString(); //så returner vi den læste værdi.
28             } else {
29                 return null; // hvis ikke der læses returner vi null.
30             }
31         } catch (SerialPortException ex) {
32             System.out.println("fejl: " + ex);
33         }
34     }
35
36     return null; // her returnes null.
37
38 }
```

I denne klasse har vi to metoder, en konstruktør og en normal metode.

- Sensor(), dette er vores konstruktør, som åbner for serial kommunikationen, og sætter parametrene.
- Det maaling() gør er, at den læser hvad der er på porten, og returnere det.

Filter Klasse

Attributter

```

1  public class Filter {
2
3      String buffer = "";
4      String[] data;
5      String[] g;
6      String[] IR = new String[500];
7      String[] RED = new String[500];
8      double[] IRP = new double[500];
9      double[] REDP = new double[500];
10     int h = 0;
11     int d=0;
```

- buffer: dette er vores Strengvariabel, hvor vi gemmer målingen og filtrere den, hvor det der er tilbage, kobles til den næste måling.
- data: dette er et streng array, hvor vi splitter målingen mellem skilte tegnet “A”.
- g: dette er vores streng array, hvor vi splitter data mellem skilte tegnet “B”.
- IR og RED: disse er vores Streng array, hvor vi gemmer g[0] og g[1] på dem, så vi for splittet infrarød og rød LEDs målinger.
- IRP og REDP: dette er double arrays, hvor vi parser IR og RED, så vi kan bruge målingerne til at regne med.

Metoder

Filtering(): i denne metode, behandler vi den måling, vi får fra sensoren. Hvor vi splitter strengen ved brug af .split(), og derefter gemmer det som en double i IRP og REDP-arrays, ved at parse.

Vi splitter først ved A, da der nu vil være en hel måling fra A til A. Denne nye måling bliver gemt i data[]. Dvs. at der nu er en Ir og Red værdi i mellemrummet adskilt af A.

Dernæst laver vi et tjek, på at data[] er fyldt, og at data[0] indeholder et B. Nu laver vi et tjek på, at hvis det sidste symbol, i vores læste værdi på buffer, er forskellig fra A, skal den sidste udfyldte plads i data[] indsættes på buffer strengen, således at man ved næste måling, vil få resten af målingen med. Til slut laver vi et .split() igen, denne gang bare på B, således at vi nu kan dele IR og Red ud i hver sit array.

Vi kommer også udenom, at den sidste værdi på data[] engang imellem er ubrugelig, fordi der mangler noget. Til sidst tæller vi h og d op, sådan at vi kan udfylde 500 pladser i et array, og vi udfylder ikke flere, da vi har vores break kontrol struktur.

```

14     public void Filtrering() {
15         while (d < 500) {
16             String s = sensor.maaling(); // her sætter vi s til at være den maaling vi får, som vi bruger som et argument
17             if (s != null) { // her kontrollerer vi om der er læst noget.
18                 buffer = buffer + s; // her kobles rest, på den første nye værdi
19                 int i = buffer.indexOf("A"); // her finder vi pladsen A er på
20                 if (i > -1) { // her validere vi om der var et A i strengen
21                     data = buffer.split(regex: "#"); // her splitter vi for a
22                     if (data != null && data.length > 0) { // her kontrollerer vi den er splittet for 2 værdi
23                         if (data[0].indexOf("B") < 1) {
24                             data[0] = null; // her sætter vi den 1 plads til null hvis b ikke er en del af den array plads.
25                         }
26                         if (buffer.charAt(buffer.length() - 1) != 65) {
27                             buffer = data[data.length - 1];
28                         } else {
29                             buffer = "";
30                         }
31                         /* her har vi kontrolleret om der er noget rest, hvis ja så er det sat lig med buffer så vi kan
32                         bruge det til næste måling, hvis nej er buffer tom.*/
33                         while (hs < data.length - 1 && data.length > 1) { // her køre vi et loop for at splitte data array for h i en ny array g
34                             if (data[h] != null) {
35                                 g = data[h].split(regex: "B");
36                                 if (g.length > 1) { // her fylder vi IR og R arrays ud
37                                     if ((d+h) >= 500) { // vi har kun 500 pladser i array, så d og h må ikke blive 501
38                                         break;
39                                     }
40                                     IR[d + h] = g[0];
41                                     RED[d + h] = g[1];
42                                     IRP[d + h] = Double.parseDouble(IR[d + h]); // her laver vi arrays fra String til Double
43                                     REDP[d + h] = Double.parseDouble(RED[d + h]);
44                                 }
45                                 h++; // dette gøres så vi både spillet data[1] og data[0]
46                             }
47                             d=d+h;
48                         }
49                         h = 0; // efter at have splittet, så sætter vi h=0;
50                     }
51                 }
52             }
53         }
54         try {
55             Thread.sleep(millis: 100);
56         } catch (InterruptedException ex) {
57             System.out.println(ex);
58         }
59     }
60     d=0;
61 }
62 }
```

Getters:

getIRP og getREDP: disse to metoder bruges, til at referere til vores IRP- og REDP-arrays.

```

64      // her har vi get metode for IR og R arrays.
65      public double[] getIRP() {
66          return IRP;
67      }
68
69
70      public double[] getREDP() {
71          return REDP;
72      }
73
74 }
```

Lommeregner Klasse

Attributter

```

3  public class Lommeregner {
4      double SP02;
5      double counter;
6      double puls;
7      double fluffer;
8      double timer = 5000;
9      double gemtPuls;
10     double omgange=1;
11     double gemtplads1=-27;
12     double gemtplads2=-27;
```

- SPO2: dette er den variabel, vi gemmer vores SpO2 beregning i.
- counter: denne variabel bruges til at tælle pulsslag/toppunkter.
- puls: denne variabel gemmer vi vores akutte puls i.
- fluffer: denne variabel er forskellen mellem to målinger, som vi bruger til at kontrollere et toppunkt.
- timer: denne er sat ift. den tid det tager at tage 500 målinger, som vi bruger, når vi skal beregne pulsen.
- gemtPuls: denne variabel bliver brugt til at beregne BPM.
- omgange: denne variabel bruges til at gemme vores omgange, den bliver brugt til at udregne gennemsnittet af BPM.
- gemtplads1: bruges til puls algoritmen
- gemtplads2: bruges til puls algoritmen

```

13     double irMAX1; //burger alt imellem til SP02
14     double redMAX1;
15     double irMIN1;
16     double redMIN1;
17     double irAverage1;
18     double redAverage1;
19     double irMAX2;
20     double redMAX2;
21     double irMIN2;
22     double redMIN2;
23     double irAverage2;
24     double redAverage2; // Bruger alt imellem til SP02
```

- irMAX1 og redMAX1: dette er hvor vi gemmer RED1 og IR1 maksimumværdier.

- irMIN1 og redMIN1: dette er hvor vi gemmer RED1 og IR1 minimumsværdier.
- irAverage1 og redAverage1: her gemmer vi RED1 og IR1 gennemsnitsværdier.
- irMAX2 og redMAX2: dette er hvor vi gemmer RED2 og IR2 maksimumsværdier.
- irMIN2 og redMIN2: dette er hvor vi gemmer RED2 og IR2 minimumsværdier.
- irAverage2 og redAverage2: her gemmer vi RED2 og IR2 gennemsnitsværdier.

```

27     double[] RED1 = new double[250];
28     double[] IR1 = new double[250];
29     double[] RED2 = new double[250];
30     double[] IR2 = new double[250];
31
32

```

- RED1 og IR1: dette er hvor vi gemmer de første 250 målinger.
- RED2 og IR2: her gemmer vi så de resterende 250 målinger.

Metoder

set arraymetode:

I denne metode, køre vi to for-løkker, hvor vi får fyldt vores IR1-2 og RED1-2 arrays ud.

Derefter bruger vi Arrays.stream().max(), min() og average(), til at finde vores 4 arrays max, min og average værdier. Vi deler de 500 målinger ud i arrays på 250 størrelsen, for at undgå at en enkelt fejlmåling, kunne ødelægge hele vores beregning.

```

33     //Metode til at sætte ARRAY, delt op i 2 således at MAX MIN bliver mere præcise, hvis der kommer fejlmålinger.
34     public void setARRAY(double[] REDP, double[] IRP) {
35         for (int i = 0; i < 250; i++) {
36             RED1[i] = REDP[i];
37             IR1[i] = IRP[i];
38         }
39         for (int i = 0; i < 250; i++) {
40             RED2[i] = REDP[250 + i];
41             IR2[i] = IRP[250 + i];
42         }
43
44         irMAX1 = Arrays.stream(IR1).max().getAsDouble();
45         irMIN1 = Arrays.stream(IR1).min().getAsDouble();
46         irAverage1 = Arrays.stream(IR1).average().getAsDouble();
47         redMAX1 = Arrays.stream(RED1).max().getAsDouble();
48         redMIN1 = Arrays.stream(RED1).min().getAsDouble();
49         redAverage1 = Arrays.stream(RED1).average().getAsDouble();
50         irMAX2 = Arrays.stream(IR2).max().getAsDouble();
51         irMIN2 = Arrays.stream(IR2).min().getAsDouble();
52         irAverage2 = Arrays.stream(IR2).average().getAsDouble();
53         redMAX2 = Arrays.stream(RED2).max().getAsDouble();
54         redMIN2 = Arrays.stream(RED2).min().getAsDouble();
55         redAverage2 = Arrays.stream(RED2).average().getAsDouble();
56
57     }

```

SPO2 algoritme:

I denne metode beregner vi vores SpO2 værdi, hvor vi gør brug af max, min og average, som vi har sat i setArray().

```

58     //SP02 Algoritme: 104-17*R=SP02          R=AC660-DC660/AC900-DC900
59     public void SP02() {
60         SP02 =(104-(17* (((((redMAX1 + redMAX2) / 2) - ((redMIN1 + redMIN2) / 2)) / ((redAverage1 + redAverage2) / 2))/(((irMAX1 + irMAX2) / 2) - ((irMIN1 + irMIN2) / 2)) / ((irAverage1 + irAverage2) / 2)));
61         // SP02 formel fundet https://
62         // pdfserv.maximintegrated.com/en/an/AN6409.pdf?fbclid=IwAR1gm1NcIz4BQ-swpkue_Nj3YnJ39II-gwEV-pllvaRnPd_s3bHVjlg8KrA - 07/01/21
63     }
64
65
66

```

Puls algoritme:

- I vores puls() metode tjekker vi først for toppunkter vha. vores algoritme. Hvis vi finder nogle, tæller vi vores tæller op. Dette gøres for begge arrays. Dernæst udregner vi pulsen, som et regnestykke, der bliver mere og mere et gennemsnit op til 20 sekunder/4 målinger. Her gemmer vi værdien, som en BPM eller gennemsnitspuls, og genstiller tællerne til deres startværdier.

I vores pulsalgoritme udregner vi delta y, altså måling 1 trukket fra måling 2. Dernæst indsætter vi den på vores fluffer værdi og tjekker, om den er under eller lig med -30, og at der samtidigt ikke er blevet registreret en måling indenfor de sidste 27 målinger. 27 kommer fra regnestykket, at hvis man har en puls på 220, har man maksimalt 3,66667 pulsslag i sekundet. Det vil sige at der minimum vil være $1/3,66667=272$ ms sekunder mellem hvert pulsslag. Da vi har 10ms mellem hver måling, bliver dette til minimum 27 målinger mellem hver måling. Denne kontrol laver vi med gemtplads1 og gemtplads2, hvor vi gemmer det målingsnummer af a, hvor der blev registreret et toppunkt. Disse værdier bliver nulstillet til -27, efter algoritmen har kørt en gang.

```

66     //Puls algoritme:
67     public void Puls() {
68         for (int a = 0; a < 248; a++) { //PULS algoritme første array.
69             fluffer = (IR1[a + 1] - IR1[a]); //Hvis ydelta er lavere og der er gået minimum 27 målinger efter en optælling
70             if (fluffer <= -30 && ((gemtplads1 - a) <= -27)){
71                 if ((IR1[a + 2] - IR1[a + 1]) > fluffer) {
72                     counter++;
73                     fluffer = 0;
74                     gemtplads1=a; //gemt plads er ved start -27
75
76                 }
77             }
78         }
79
80         for (int a = 0; a < 248; a++) { //PULS algoritme andet array.
81             fluffer = (IR2[a + 1] - IR2[a]); //Hvis ydelta er lavere og der er gået minimum 27 målinger efter en optælling
82             if (fluffer <= -30 && ((gemtplads2 - a) <= -27)){
83                 if ((IR2[a + 2] - IR2[a + 1]) > fluffer) {
84                     counter++;
85                     fluffer = 0;
86                     gemtplads2=a; //gemt plads er ved start -27
87                 }
88             }
89         }
90
91         puls = (60000 / (timer / (counter - omgange))); //Puls beregning, tæller op til gennemsnit af 4 målinger
92         gemtplads1=-27;
93         gemtplads2=-27;
94         timer= timer+5000;
95         omgange++;
96         if (timer==25000) //BPM beregning
97         {
98             gemtPuls=puls; //gemmer
99             timer=5000; // nulstiller alt til startværdi efter
100            omgange=1;
101            counter=0;
102        }
103
104        System.out.println("Puls: " + puls+ " Gennemsnitpuls: "+gemtPuls);
105    }

```

Getter:

getSPO2, getPuls og getBPM: her returnerer vi SPO2, puls og gemitPlus, som er gennemsnitspulsen.

```
106     public double getSPO2() {
107         return SP02;
108     }
109
110     public double getPuls() {
111         return puls;
112     }
113
114     public double getBPM(){
115         return gemitPuls;
116     }
117
118 }
```

Database klasse

Metoder

DataBase(): Denne metode er en konstruktør, hvor vi opretter et objekt, bw, fra java.io* biblioteket.

skrivData(): denne metode bruger bw objektet og write() metoden, til at skrive IR og R værdierne ned i en fil, hvor IR og R er argumenter, som vi henter.

skivResultat(): denne metoder skriver vores SpO2 og puls, ved brug af write(), hvor puls og SpO2 er argumenter, vi henter fra Lommeregner klassen.

```
1 import java.io.*;
2
3 public class DataBase {
4
5     Writer bw;
6
7
8     public DataBase() {
9         try {
10             bw = new BufferedWriter(new FileWriter( fileName: "Data"));
11         } catch (IOException e) {
12             e.printStackTrace();
13         }
14     }
15
16     public void skrivData(double IR, double R) { //skriv Data til fil
17         try {
18             bw.write(str: "IR: " + IR + "      R: " + R + "\n");
19         } catch (IOException e) {
20             e.printStackTrace();
21         }
22         try {
23             bw.flush(); // der bruges flush, så vi ikke slette de forrige data i filien
24         } catch (IOException e) {
25             e.printStackTrace();
26         }
27     }
28 }
```

```
28
29     public void skrivResultat(double SP02, double HR, double BPM) { //skriv resultat til fil
30         try {
31             bw.write(str: "SP02    " + SP02 + "    HR :    " + HR + "    BPM :    "+ BPM +"\n");
32         } catch (IOException e) {
33             e.printStackTrace();
34         }
35         try {
36             bw.flush();
37         } catch (IOException e) {
38             e.printStackTrace();
39         }
40     }
41 }
42 }
```

AFPRØVNING

LINK TIL KODE

<https://drive.google.com/drive/folders/1UIUPAIIfg7XCDb6vusdWfGMFUR-pBvhdu?usp=sharing>

Black Box

Videoer af **blackbox** testing:

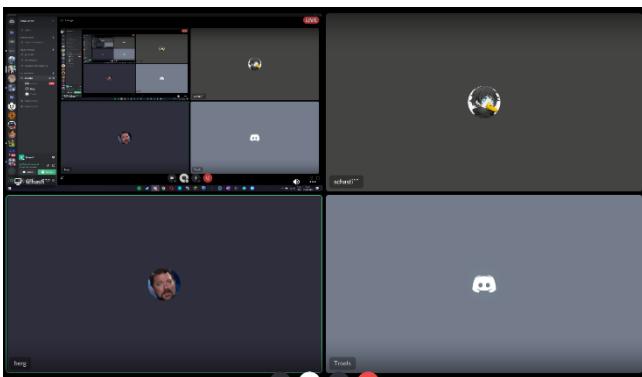
Sammenligning med måling af puls på a. Radialis og manuel måling.

<https://youtu.be/QDboVsiB8zs>

Sammenligning med finger i og uden finger i.

<https://youtu.be/4lNO1Y0DYls>

White Box



Vi fik Andreas Berg, til at kigge vores kode igennem, og finde eventuelle fejl, eller steder hvor der kunne være fejl. Vi prøvede forskellige dele af koden af, og har her billeder af resultaterne:

På dette billede printer vi s strengen ud i vores filter klasse. Her kan vi se at værdierne, er det vi regnede med.

```

int d=0;
Sensor sensor = new Sensor();
public void Filtrering() {
    while (d < 500) {
        String s = sensor.reading(); // her sætter vi s til at være den måling vi får, som vi bruger som et argument
        System.out.println(s);
        if (s != null) { // her kontrollerer vi om den er læst noget.
            buffer = buffer + s; // her kobles rest, på den første nye værdi
            int i = buffer.indexOf("A"); // her finder vi pladsen A er på
            if (i > -1) { // her validere vi om der var et A i strengen
                data = buffer.split("\\n"); // her splitter vi for a
                if (data[0] == null && data.length > 0) { // her kontrollerer vi den er splittet for 2 værdi
                    if (data[0].indexOf("0") < 1) {
                        data[0] = null; // her sætter vi den i plads til null hvis b ikke er en del af den array plads.
                    }
                }
                if (buffer.charAt(buffer.length - 1) == 65) {
                    buffer = data[data.length - 1];
                }
            }
        }
    }
}

```

Process finished with exit code 1

IntelliJ IDEA 202.4 available
Update...

På dette billede printer vi data array ud, her kan vi også se, at værdier er det vi regnede med. Disse værdier viser nu en måling ad gangen.

```

int d=0;
Sensor sensor = new Sensor();
public void Filtrering() {
    while (d < 500) {
        String s = sensor.reading(); // her sætter vi s til at være den måling vi får, som vi bruger som et argument
        if (s != null) { // her kontrollerer vi om den er læst noget.
            buffer = buffer + s; // her kobles rest, på den første nye værdi
            int i = buffer.indexOf("A"); // her finder vi pladsen A er på
            if (i > -1) { // her validere vi om der var et A i strengen
                data = buffer.split("\\n"); // her splitter vi for a
                for(int k=0;k<data.length-1;k++){
                    System.out.println(data[k]);
                }
            }
            if (data[0] == null && data.length > 0) { // her kontrollerer vi den er splittet for 2 værdi
                if (data[0].indexOf("0") < 1) {
                    data[0] = null; // her sætter vi den i plads til null hvis b ikke er en del af den array plads.
                }
            }
        }
        Thread.sleep(1000); // This line pauses the thread for 1 second
    }
}

```

Process finished with exit code 1

På dette billede tester vi Thread.sleep() indflydelse på vores program. Da Andreas ville se programmet køre uden Thread.sleep(). Det ødelægger ikke programmet, men gør, at vi læser null, når vi ikke får data fra Arduinoen. Derfor vælger vi at have Thread.sleep() på, da vi gerne vil mindske mængden af null målinger mest muligt, samt at vi har testet, at vi ikke mister målinger.

The screenshot shows an IDE interface with the following details:

- Project Structure:** Semesterprojekt 1 [Semesterprojekt 1 opdelt] contains .idea, Billeder, Database, Kode Arduino, out, and src. src contains DataBase, Filter, Lommeregner, Main (selected), Sensor, Data, GruppeKontrakt.pdf, GruppeKontrakt (1).pdf, Semesterprojekt 1 opdelt.iml, SemesterProjekt1.zip, and StatusNotat 1.pdf.
- Main.java Content:**

```
50     h = 0; // efter at have splittet, så sætter vi h=0;
51 }
52 }
53 }
54 }
55 }
56 try {
57     Thread.sleep( millis: 50 );
58 } catch (InterruptedException ex) {
59     System.out.println(ex);
60 }
61 }
62 }
63 d=0;
64 }
65 }
66 // her har vi get metode for IR og R arrays.
67 public double[] getIRP() {
68     return IRP;
69 }
70 }
71 public double[] getREDP() {
72     return REDP;
73 }
```
- Run History:** Shows a list of recent runs:
 - 116959B
 - 111131A
 - 116923B111109A
 - 116
 - 932B111123A
 - 116928B
 - 111112A
 - 116950B
 - 111121A
 - 116960B
 - 111125A
 - 116956B111130A
 - null

Her printer vi fluffer værdierne ud, og ser om de stemmer overens med, hvad vi gerne vil se.

```

    for (int a = 0; a < 248; a++) { //PULS algoritme andet array.
        fluffer = (IR2[a + 1] - IR2[a]); //Hvis ydelta en lavere og der er gået minimum 15 målinger efter en optælling
        System.out.println(fluffer);
        if (fluffer <= -30 && (gentplads2 - a) <= -27){
            if ((IR2[a + 2] - IR2[a + 1]) > fluffer) {
                counter++;
                fluffer = 0;
                gentplads2=a; //gent plads er ved start -15
            }
        }
    }
    puls = (60000 / (timer / (counter - omgange))); //Puls beregning, tæller op til gennemsnit af 4 målinger
}
    
```

Run:

- 5.0
- 36.0
- 3.0
- 35.0
- 18.0
- 0.0
- 35.0
- 3.0
- 18.0
- 23.0
- 37.0
- 12.0
- 39.0
- 28.0
- 7.0
- 43.0
- 1.0
- 25.0
- 17.0
- 31.0
- 15.0
- 29.0
- 28.0

Her kan man se vores data udprint til filen, Andreas kommenterede på, at man kunne undlade at printe resultatet, da det ville gøre det nemmere at lave en graf over senere. Vi valgte at holde det som det var, fordi vi føler, at man skal have noget at sammenligne det resultat, man selv ville kunne udregne fra værdierne, med hvad vi udregnede fra værdierne.

480	IR: 110995.0	Red: 107900.0
481	IR: 110991.0	Red: 107889.0
482	IR: 110956.0	Red: 107885.0
483	IR: 110892.0	Red: 107862.0
484	IR: 110835.0	Red: 107825.0
485	IR: 110729.0	Red: 107789.0
486	IR: 110643.0	Red: 107749.0
487	IR: 110525.0	Red: 107701.0
488	IR: 110428.0	Red: 107643.0
489	IR: 110298.0	Red: 107689.0
490	IR: 110204.0	Red: 107582.0
491	IR: 110110.0	Red: 107560.0
492	IR: 110059.0	Red: 107537.0
493	IR: 109995.0	Red: 107514.0
494	IR: 109966.0	Red: 107497.0
495	IR: 109924.0	Red: 107471.0
496	IR: 109908.0	Red: 107470.0
497	IR: 109877.0	Red: 107474.0
498	IR: 109876.0	Red: 107471.0
499	IR: 109855.0	Red: 107478.0
500	IR: 109857.0	Red: 107455.0
501	SP02 97.14682736223637	HR : 72.0 BPM : 0.0
502	IR: 109822.0	Red: 107450.0
	TOT: 109876.0	Red: 107471.0

Gruppernes indsats

I dette afsnit laver vi en gennemgang af byrdefordelingen, hvor vi detaljeret beskriver hvor lang tid, der er blevet brugt på visse ting. Dette indebærer, hvor mange timer vedrørende har sat af til sit arbejde, samt hvilke emner personen har skrevet om.

Oversigt over møder

Her er en link til vores møder:

<https://docs.google.com/document/d/16AzL4BjpNzamzC9CPe6tOKTH2CVAkZfMI-buh84t7bEc/edit>

Angivelse af hvem laver hvad

Her er et link til tidsregnskabet og angivelse af hvem der har lavet hvad:

<https://docs.google.com/document/d/1wyB-4GH4mHENmGZ3xM-o6DZlG00zYjXQ61a0ZciG4MA/edit#>

Diskussion

Vi må diskutere de eventuelle fejlkilder der findes. Det betyder, at vi bliver nødt til at snakke om fejlmålinger i komponenten. Komponenten kan give fejlmålinger, hvis den bliver flyttet på fingeren midt i en måling, hvis den optiske detektor registrerer udefrakommende lys osv. Disse kan vi selvfølgelig ikke komme udenom, da de kommer fra fabrikanten og ikke os. Udover må man også vurdere, at visse faktorer ændrer den reflekterede lys, såsom melaninniveauet og vævstykkelsen. Dette påvirkede os under indsættelse af en grænseværdi til pulsberetningen, da Naveeds delta y toppunkter, lå omkring -150, hvorimod Troels' lå omkring -50. Ud fra denne observation, valgte vi vores på -30 og testede, at den virkede på Naveed, hvor den stadig virkede, samt prøvede, at lave kontrol for sådanne fejl.

Man skal ikke fastlægge, at man har en forhøjet puls eller for lav puls lige efter en pulsmåling. Man kan have en forhøjet puls hvis man har været fysisk aktiv, eller en for lav puls, hvis man har en god kondition. Man skal derfor have denne viden i baghovedet, inden man kan diagnosticere brugeren med tradycardia eller bradycardia.

Puls er og vil altid være en approksimation, medmindre man måler direkte på hjertet med elektroder. Dog kan det godt være et kvalificeret gæt. Vores pulsoximeter er også kun en approksimation, der finder ens puls over 4 gange 5 sekunder. Man skal undgå at bruge vores pulsoximeter som ens endelige svar, da komponenten vi brugte, ikke er FDA, CE og FCC godkendt. Men den kan godt give dig en approksimation.⁶⁴

Konklusion

Vi har givet en grundlæggende teori om puls, hjertet, iltmætningen og generelt om alt det der er relevant, at have som baggrundsviden, når vi måler data fra sensoren. Dernæst har vi fået lavet algoritmer, til både at kunne beregne puls og SpO₂ ud, med de Ir og Red målinger vi får fra MAX30102 sensoren. Derfor kan vi konkludere, at man godt kan bruge MAX30102 til at måle SpO₂ og pulsen.

Bilag filer

Excel fil:

https://drive.google.com/file/d/1PFfLrEQgO7JyegiiZkJ_bIbOIzoUtg82/view?usp=sharing

Test Arduino fil:

https://drive.google.com/file/d/12-Cai6_qio3BksGuKGWuuYuQUAwrkoDo/view?usp=sharing

Arduino fil:

<https://drive.google.com/file/d/1xJXXwrdR1CqR3peLnB0PyLhAlOViKNN2/view?usp=sharing>

Java Kode:

<https://drive.google.com/drive/folders/1UlUPAI8g7XCDb6vusdWfGMFURpBvhdu?usp=sharing>

Litteraturliste

¹ <https://anatomifysiologi.digi.munksgaard.dk/?id=531> d.19-01-20201

² <https://www.sundhed.dk/borger/patienthaandbogen/hjerte-og-blodkar/illustrationer/tegning/hjerteskamre-set-forfra/> d.19-01-20201

³ <http://claus-lm.blogspot.com/2014/12/jeg-undres-over-guds-forunderlige.html> d.19-01-20201

⁴ <https://anatomifysiologi.digi.munksgaard.dk/?id=531> d.19-01-20201

⁵ <https://anatomifysiologi.digi.munksgaard.dk/?id=538> d.19-01-20201

⁶ <https://anatomifysiologi.digi.munksgaard.dk/?id=527> d.19-01-20201

⁷ <https://www.studocu.com/da/document/kobenhavns-universitet/arbejdsfysiologi-1-bevaegelgeapparatets-fysiologi-og-funktionelle-anatomi-t-afysl/foredragsnoter/basal-neurofysiologi-foredragsnotater-6/2155836/view> d.19-01-20201

⁸ <https://anatomifysiologi.digi.munksgaard.dk/?id=538> d.19-01-20201

⁹ <https://bookanaut.com/dk/puls-hvilepuls-normal-puls/> dato: 6 januar

¹⁰ <https://sygdomslaere.digi.munksgaard.dk/index.php?id=209#c954> d.19-01-20201

¹¹ <https://bookanaut.com/dk/puls-hvilepuls-normal-puls/> dato: 6 januar

¹² <https://anatomifysiologi.digi.munksgaard.dk/?id=535> d.19-01-20201

¹³ Nielsen F. Oluf & Bojsen-Møller J. Mette, Anatomi og fysiologi, hånden på hjertet 1. udgave, side 82

¹⁴ Nielsen F. Oluf & Bojsen-Møller J. Mette, Anatomi og fysiologi, hånden på hjertet 1. udgave, side 83 og 85

¹⁵ <https://anatomifysiologi.digi.munksgaard.dk/?id=532> d.19-01-20201

¹⁶ <https://anatomifysiologi.digi.munksgaard.dk/?id=532> d.19-01-20201

¹⁷ <https://anafys.dk/Mobil/ParaKapHigh/index.html> d.19-01-20201

¹⁸ <https://anatomifysiologi.digi.munksgaard.dk/?id=532> d.19-01-20201

¹⁹ <https://anatomifysiologi.digi.munksgaard.dk/?id=532> d.19-01-20201

²⁰ Nielsen F. Oluf & Bojsen-Møller J. Mette, Anatomi og fysiologi, hånden på hjertet 1. udgave, side 94-95

²¹ Nielsen F. Oluf & Bojsen-Møller J. Mette, Anatomi og fysiologi, hånden på hjertet 1. udgave, side 91, 94-95, 104-105

²² <https://anatomifysiologi.digi.munksgaard.dk/?id=545> d.19-01-20201

²³ <https://anatomifysiologi.digi.munksgaard.dk/?id=475> d.19-01-20201

²⁴ Nielsen F. Oluf & Bojsen-Møller J. Mette Anatomi og fysiologi, hånden på hjertet 2 udgave s.132.

²⁵ <https://www.sundhed.dk/borger/patienthaandbogen/lunger/illustrationer/tegning/trakea-bronkier-alveoler/> d.19-01-20201

-
- ²⁶ Nielsen F. Oluf & Bojsen-Møller J. Mette, Anatomi og fysiologi, hånden på hjertet 1. udgave, side 146-147
- ²⁷ Nielsen F. Oluf & Bojsen-Møller J. Mette, Anatomi og fysiologi, hånden på hjertet 1. udgave, side 145-146
- ²⁸ <https://www.sciencedirect.com/topics/neuroscience/pulse-oximetry>. d.15-01-20201
- ²⁹ <https://learn.sparkfun.com/tutorials/max30105-particle-and-pulse-ox-sensor-hookup-guide/using-the-sparkfun-max30105-arduino-library> 08-01-2021 d.19-01-20201
- ³⁰ <https://drive.google.com/file/d/1VCtWxW6ll-fZjU1cymkv6JStq19PWxbk/view?usp=sharing> d.07-01-2021
- ³¹ <https://drive.google.com/file/d/1VCtWxW6ll-fZjU1cymkv6JStq19PWxbk/view?usp=sharing> d.07-01-2021
- ³² <https://drive.google.com/file/d/1VCtWxW6ll-fZjU1cymkv6JStq19PWxbk/view?usp=sharing> d.19-01-20201
<https://drive.google.com/file/d/1VCtWxW6ll-fZjU1cymkv6JStq19PWxbk/view?usp=sharing> d.07-01-2021.
- ³⁴ <https://da.wikipedia.org/wiki/Arduino> d.19-01-20201
- ³⁵ <https://www.arduino.cc/en/Main/ArduinoBoardUnoSMD> d.10/01/21
- ³⁶ <https://components101.com/microcontrollers/arduino-uno> d.10/01/21
- ³⁷ <http://www.computerdk.com/Hardware/computer-peripherals/14919.html> d.7/01/21
- ³⁸ <https://www.elecparts101.com/atmega328p-datasheet-and-pinout-low-power-8-bit-atmel-microcontroller/> d.7/01/21
- ³⁹ https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf d.7/01/21
- ⁴⁰ <https://king.kapook.com/adminfooter/cache/up-me-izjao/cwj9qxw.php?cd0413=atmega328p-eeprom-write-cycles>) d.7/01/21
- ⁴¹ https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf d.7/01/21
- ⁴² https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf d.7/01/21
- ⁴³ <https://components101.com/microcontrollers/arduino-uno> d.7/01/21
- ⁴⁴ <https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf> d.06-01-20201
- ⁴⁵ <https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf> d.06-01-2021.
- ⁴⁶ <https://theengineeringmindset.com/the-basics-of-diodes-explained/> d.06-01-2021
- ⁴⁷ <https://theengineeringmindset.com/the-basics-of-diodes-explained/> d.06-01-2021
- ⁴⁸ <https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf> d.08-01-2021
- ⁴⁹ <https://www.nxp.com/docs/en/application-note/AN4327.pdf> d.08-01-2021
- ⁵⁰ <https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf> d.09-01-2021
- ⁵¹ <https://en.wikipedia.org/wiki/Photodetector> d.09-01-2021
- ⁵² [https://en.wikipedia.org/wiki/Sampling_\(signal_processing\)](https://en.wikipedia.org/wiki/Sampling_(signal_processing)) d.06-01-2021
- ⁵³ <https://www.electronics-tutorials.ws/combination/analogue-to-digital-converter.html> d.06-01-2021
- ⁵⁴ <https://www.electronics-tutorials.ws/opamp/op-amp-comparator.html> d.06/01/21
- ⁵⁵ <https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf> d.07-01-2021
- ⁵⁶ <https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf> d.07-01-2021

⁵⁷ 3 min inde, screenshot: https://www.youtube.com/watch?v=6IAkYpmA1DQ&ab_channel=HowToMechatronics d.06-01-2021

⁵⁸ [https://wiki.mcselec.com/bavr/Using the I2C protocol](https://wiki.mcselec.com/bavr/Using_the_I2C_protocol) d.07-01-2021

⁵⁹ <https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf> d.07-01-2021

⁶⁰ [https://wiki.mcselec.com/bavr/Using the I2C protocol](https://wiki.mcselec.com/bavr/Using_the_I2C_protocol) d.07-01-2021

⁶¹ <https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf> d.07-01-2021

⁶² https://en.wikipedia.org/wiki/Iterative_and_incremental_development d.09-01-2021

⁶³ <https://pdfserv.maximintegrated.com/en/an/AN6409.pdf?fbclid=IwAR24uEhOqVBj9fx3lR5wh-jomiGle0qP9hL3No9RyG3c48wzYgxgffMIAk> d.11-01-2021

⁶⁴ <https://protocentral.com/product/pulse-3-pulse-ox-heart-rate-sensor-based-on-max30102-qwiic-compatible/> d.06-01-2021