
Semesterprojekt

# Poleposition

*Alexander D. Larsen, Frederik N. Larsen, Mikkel L. Olsen, Phillip M. Kyndbøl og Rasmus Haugaard*

faglig vejleder
Jan EFTERNAVN HER

15. maj 2016

# Indhold

# 1 Indledning

SKRIV INDLEDNING

# 2 Problemformulering

Dette projekt indebærer at udvikle en microcontroller-baseret styring til en racerbil med henblik på at opnå den korteste omgangstid på en ukendt bane. På langsiderne vil bilen kunne køre med max. hastighed, mens et sving kræver en nedsat hastighed. Der vil blive fokuseret på to hovedområder, den fysiske del som består i optimering af tophastighed, acceleration, bremselængde og hastighed i sving. Det andet hovedområde er programmering, som består i at lære den ukendte bane så bilen selv ved hvornår den skal bremse og accelerere.

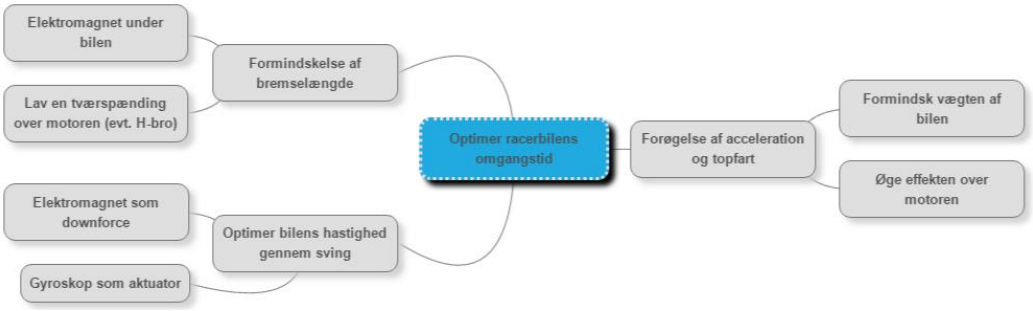
- Undersøg effekt af gyroskop som krængningsstabilsator til optimering af hastighed gennem sving.
- Undersøg elektromagnet som downforce til minimering af hjulspind og optimering af bremselængde.
- Udvikkel et optoelektronisk system der kan registrere en startlinje.
- Undersøg udvikling af mapning af banen vha. accelerometer, gyroskop og hallcensor.
- Undersøg effekten af fysiske ændringer på bilen f.eks. fastsættelse af affjedring i fht. til optimering af banetider.

# 3 Idégenerering

For at sætte den hurtigste omgangstid skal der være en kombination af den rigtig fysiske opsætning af bilen og en microprocessor der fortæller bilen hvordan den skal køre på banen. Det første der blev undersøgt var de fysiske optimeringer der kunne laves på bilen.

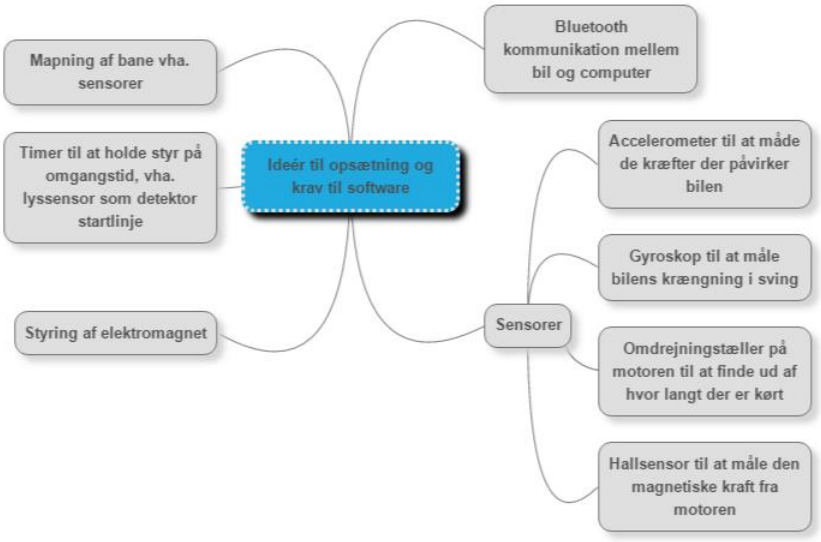
- Forøgelse af acceleration og tophastighed
- Formindskelse af bremselængde
- Optimering hastighed igennem sving

For at optimere ovenstående er der flere ting der kan overvejes, se figur 1.




Figur 1: SKRIV TEKST HER

Udover de fysiske ting i bilen skal bilens omgangstid også optimeres ved hjælp af software.




Figur 2: SKRIV TEKST HER

Ideén er ved hjælp af sensorerne skal softwaren kunne mappe den ukendte bane ved at kører et par omgange på den. Når banen er blevet mappet og gemt i bilens hukommelse kan den så sætte en hurtig omgangs tid da den ved hvornår der er sving hvor der skal bremse og hvor hårdt den kan accelerere de forskellige steder på banen.

# 4 Fysiske modifikationer

For at optimere bilens omgangstider mest muligt er der også fortaget fysiske modifikationer på bilen. Som beskrevet i opgaveformulering var der også krav til at der skulle bruges enten en elektromagnet som aktuator eller en hall-sensor. Der er blevet fortrukket en elektromagnet til at skabe yderligere downforce.

De ting der blandt andet er med til at give en optimeret omganstid er:

- Minimering af vægt
- Afstivelse af chassis
- Lavere tyngdepunkt
- Justerbar downforce

For at minimere vægten af bilen er alt unødvendigt plastik blevet fjernet fra bilen.

Det udleveret chassis på racerbilen er meget vakkelvornt, og kan vrides let. For at optimere dette er der blevet 3D-printet et nyt forstærket chassis. Det chassis er stærkere end det gamle og er samlet i et stykke.

Det vil sige at den affjedring der følger med bilen er blevet fjernet og erstartet med det stive chassis. Grunden til man har afjedring i virkelige biler er for at beskytte både kører og motor, da man aldrig er sikker på asfaltens tilstand. Da banen racerbilen skal køre på er flad og der er ingen kører der skal beskyttes, vil det optimere bilens hastighed igennem sving uden at den kæntre at fjerne affjedringen.

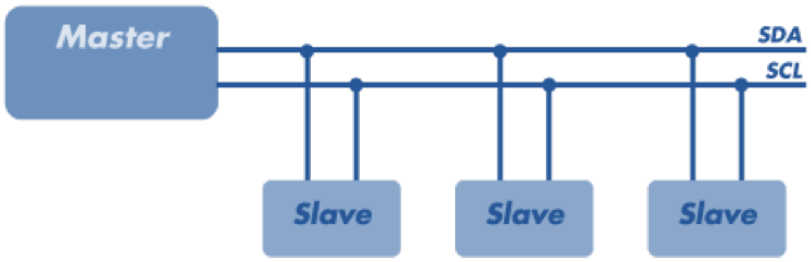
For at kunne hurtigt gennem sving kræver det et lavt tyngdepunkt i bilen, jo lavere tyndgdepunktet er jo hurtigere kan bilen klare at køre gennem et sving uden at kæntre. Ved at fjerne affjedringen optimeres tyngdepunktet også da karroseriet er i sin laveste tilstand hele tiden, og ikke kun når fjedrene er trykket sammen.

Downforce er ekstremt brugbart i sving da det hjælper med at holde bilen på banen ved højere fart. Downforce er mindre brugbart på lige strækninger da bilen let bliver på banen og downforce faktisk vil sænke bilens acceleration og topfart. Derfor skal der indføres justerbar downforce med elektromagneten. I det 3D-printet chassis er der en indbygget holder til elektromagneten. Elektromagnet er placeret så tæt på den split som går ned i banen som muligt

for at øge downforce på den da den holder bilen fast i banen, og der sidder en permamagnet bagesrt i bilen forvejen.

# 5 I2C kommunikation

Til at kommunikere med de digitale komponenter, herunder vores udleveret accelerometer og gyroskop, anvendes den kommunikationsprotokol kaldet I2C eller TWI ("two-wire interface"). Navnet "two-wire interface" kommer af, at hele kommunikationen foregår over kun 2 ledninger, men hvor der samtidig er mulighed for at kommunikere med mange komponenter.




Figur 3: Viser en skitse af hvordan opsætningen kan laves mellem én "master" og flere "slaver", kun ved brug af to forbindelser.

De to forbindelser der bruges til vores TWI kommunikation hedder SCL ("Serial Clock Line") og SDA ("Serial Data Line"). SCL er en clock der sættes af master, for at både master og slaven snakker og lytter ved en fælles frekvens. SDA er den forbindelse, hvor alt data bliver overført både fra slaven til master, men også fra master til slaven. Denne form for kommunikation kaldes for Half-Duplex, og fungere ligesom en walkie talkie, hvor der kun er én der snakker af gangen, imens den anden lytter.

## 5.1 Kommunikationsprotokol

Kommunikationen mellem master og slave, foregår ud fra en bestemt protokol, som er opgivet ved den enkelte slaves datablad. Denne protokol er den samme for accelerometeret og gyroskopet, og er illustreret på figur 4.

| Master | ST | SAD + W | | SUB | | SR | SAD + R | | | NMAK | SP |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Slave | | | SAK | | SAK | | | SAK | DATA | | |

Figur 4: SKRIV TEKST HER

Figur 4 viser skridt for skridt, hvordan kommunikationsprotokollen er opbygget, hvis man vil læse data én gang fra én af vores to digital komponenter.

Protokollen indeholder følgende instruktioner for at modtage én data byte:

1. ST ("Start Bit"): Master sender et start bit til alle slaverne, for at starte kommunikationen.

2. SAD+W ("Slave Adress + Write"): Master sender en slave adresse på 7 bits, samt ét write bit. Adressen er specifik for hver komponent og fortæller hvilken slave, som resten af kommunikationen kommer til at foregå med. Det sidste bit indikere, at masteren vil skrive noget til slaven.

3. SAK ("Slave Acknowledge"): Slaven med den valgte adresse sender er ACK bit tilbage, som fortæller at den har hørt hvad masteren sagde, og gør klar til at læse næste instruktions.

4. SUB ("Sub adress"): Master sender herefter en underadresse. Denne underadresse er en 7 bit adresse, som fortæller hvilken register inde i den valgte komponent, der gerne vil adresseres.

5. SAK ("Slave Acknowledge"): Slaven sender endnu et SAK bit, og gør klar til næste instruktion.

6. SR ("Repeated Start"): Master sender herefter et gentagende start bit. Der skal altid sendes en start/repeated start instruktion, hver gang der skiftes mellem at læse eller skrive. Ved repeated start vedholdes forbindelsen til slaven, og slaven ved derfor allerede hvilken underadresse, der herefter skal læses fra.

7. SAD+R ("Slave Adresse + Read"): Master sender herefter igen slave adressen, samt én read bit, for at ændre at vi nu gerne vil læse på slaven, modsat at vores tidligere write, som skrev til slaven.

8. SAK ("Slave Acknowledge"): Slaven sender en ACK bit.

9. DATA ("Data Byte"): Slaven sender efterfølgende en data byte til master.

10. NMAK ("Master Not Acknowledge"): Herefter sendes én NMAK fra master til slaven, for at indikere at der ikke skal læses mere data.

11. SP ("Stop Bit"): Til sidst sender master et stop bit, for at afslutte kommunikationen.

# 6 Boost converter

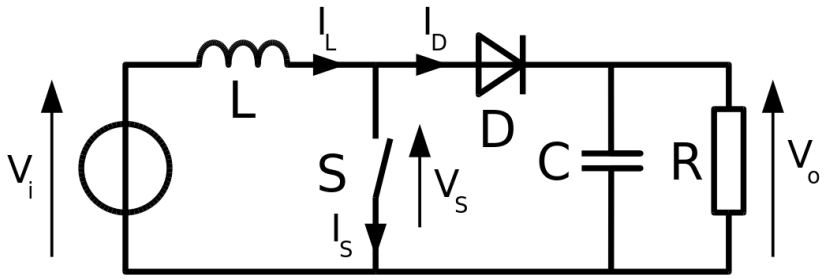
En boost converter er et analogt DC-DC kredsløb som har en højere udgangsspænding end indgangsspænding. Ideén med en boost converter er at øge effekten hen over motoren og derved få højere acceleration og tophastighed. Ved 15V trækker motoren kun 0,5A og derved bruger den kun 7.5 Watt. Da vi har en strømforsyning der kan levere 30 Watt kan vi udnytte det meget mere ved at øge spændingen. Dette vil dog gøre vi ikke helt kan trække 30 Watt da man aldrig kan lave en 100 procent effektiv boost converter. Ideén er at have en udgansspænding på 30v, og hvis boost converteren kun er 85 procent effektiv giver det en strøm, I på:

$$15V \cdot 2A = 30W \tag{1}$$

$$30W \cdot 0,85 = 25,5W \tag{2}$$

$$I = \frac{25,5W}{30V} = 0,85A \tag{3}$$

Hvilket burde være mere end rigeligt til at drive motoren langt hurtigere end med 15v og 0,5A




Figur 5: SKRIV TEKST HER

På ovenstående figur ses et forsimplet kredsløb af en boost converter. Ideén er at når switchen S, sluttes bliver der en magnetisk kraft opladet i spolen L. Når switchen bliver brudt vil kraften fra spolen oplade kondensatoren, men da strømmen i en spole ikke kan ændres momentant vil der også løbe strøm fra spændingskilden igennem spolen og derved vil der opnås en højere spænding over kondensatoren C.

Da dette kredsløb gerne skulle være selvkørende bruges der ikke en switch, men en MOSFET som skal aktiveres af et højfrevent signal. Det er vigtig at

frekvensen er så høj at spolen ikke når at gå i mætning, som vil sige at det ikke når det punkt hvor virker som en kortslutning, da kredsløbet så vile være en direkte kortslutning fra vores spændingskilde til ground.

Det er vigtigt at udvælge de rigtige komponenter til dette kredsløb da det er et højfrekvent kredsløb hvor der løber store strømme. En schottky-diode benyttes da den kan switche ved langt højere frekvenser end en almindelig diode og har et lavere spændingstab henover den og dermed får vi større effektivitet. Kondensator skal have så stor kapitans som mulig(selvfølglig inden for rimelighedens grænser) og kunne tåle høje spændinger hen over den. MOSFET'en som bliver brugt som switch skal kunne switche hurtigt og kunne klare at der løber stor strøm igennem den uden at blive alt for varm.

En af de problemer ved at bygge en boost converter som fast skal kunne levere 30V er at an på motorens hastighed ændre motorens indre modstand sig, og derved ændrer spændingenen sig også. Det er derfor nødvendigt at have en form for feedback der fortæller frekvensgenratoren om den skal øge eller sinke frekvensen for at outputtet bliver som forventet. Der findes IC-kredse der gør dette automatisk, men da vi ikke kan stå indefor hvordan de IC-kredse virker valgte vi ikke at bruge dem i vores bil, og derved mente vi at bygge boost converteren ville blive en for stor udfordring i forhold til den tid vi havde til projektet.

# 7 Lap timer

Formålet med Lap timeren er at have et stykke kode som måler og returnere omgangstiden, for hver omgang bilen køre. Lap timeren er baseret på "timer1"som er den største timer. Lap timeren bliver udover omgangstid, også brugt af andre kode moduler, til at måle tid.

## 7.1 Opsætning af Lap timer

Som tidligere nævnt, er det "timer1"som bliver benyttet i dette kodemodul. Inden timer1 kan bruges, skal følgende siddes op:

1. Control register
2. Nulstille timer/counter register
3. Global interrupt
4. Timer1 overflow interrupt
5. Extern interrupt 2

### 7.1.1 Prescaler

En stor del af timerens precsitionen afhænger af hvor mange counts den kan nå at tælle inden den bliver afbrudt igen. Det vil sige at jo lavere prescaler jo større præsition. Problemet ved en lav prescaler er dog, at man mindsker den tid timeren kan tælle, inden den overflower. Timer1 bliver også brugt af andre dele af programmet, til bla. at måle hastighed på motoren, hvilket sidder et forholdsvis stort krav om nøjagtighed. For både at kunne bevare en høj nøjagtighed, men samtidig have en lang tælletid, har vi valgt at tildele timer1 et ekstra 8-bit register. Alene kan timer1 max indeholde decimalværdien 65536(16-bit), hvor derimod med det ekstra 8-bit registre, kan timeren indeholde decimalværdien 16777216 (24-bit).
Hver gang der kommer et impuls fra motor enkoderen (**Indsæt reference motor sensor kapitel**)refereres der til dette som et "tik".
For at bestemme den rette prescaler bruges tabel 7.1.1. Tabellen er baseret på udregningerne fra billag 9.1.

| Prescaler | Δt pr. count | Counts pr. "tik" | Overflow efter |
|---|---|---|---|
| 0 | 0.0000000625 s | 8000.00 | 1.05 s |
| 8 | 0.0000005 s | 1000.00 | 8.39 s |
| 64 | 0.000004 s | 125.00 | 67.10 s |
| **256** | **0.000016 s** | **31.25** | **268.44 s** |
| 1024 | 0.000064 s | 7.81 | 1073.74 s |

*Tabel 7.1.1: Værdierne i tabellen er baseret på en motor hastighed på 2KHz, Intern clock på 16MHz og 24-bit timer register.*

Ved en fysisk test af encoderen, blev den maksimale motor hastighed målt til 2KHz. Microcontroleren's interne clock er sat op til at køre med 16MHz. Ved at kaste et blik på tabel 7.1.1 ses det at en prescaler på 256, giver en lang tælletid og en acceptabel præcision. Umiddelbart ser det ud som om 64 ville være den bedre prescaler.. Men 31 count's pr. motor tik (max hastighed) er acceptabelt præcision, derfor er der ingen grund til at vælge en lavere prescaler. Hver gang timer1 overflower (16-bit), skal et interrupt inkrimentere det ekstra register som timeren fik tildelt. Jo lavere prescaler, jo flere gange bliver programmet afbrudt af interrupt. 256 bruges som prescaler.

## 7.2 Kode

Lap-timer koden består primært af 2 interrupt rutiner. Den første rutine bliver aktiveret når bilen passere målstregen. Når målstregen passeres, sker der 4 ting: (Figur nr. 6 fra bilag viser et flowchart over den første interrupt rutine.)

1. Pushe værdierne fra general purpose registrene, samt status registret ind i stakpointeren

2. Timeren stoppes, hvor ved værdien i timer registret + det ekstra register bliver sendt til computeren. Hele dette step bruges under test perioden og bliver derfor "kommenteres ud"ved det endelige løb, da der ikke må kommunikeres med computeren.

3. Det 16-bit timer register og det 8-bit ekstra register bliver nulstillet, så timeren er klar til næste omgang.

4. Inden der returneres popes de oprindelige værdier tilhørende status register og generalpurpurs registrene.

Figur nr. 6 fra bilag viser et flowchart over den første interrupt rutine.

Den anden rutine bliver aktiveret, når timer1 overflower. Denne rutine har til opgave at inkrimentere det ekstra 8-bits register som timeren er blev tildelt. Ved at gøre dette svare det til at timer1 havde været baseret på et 24-bit register. Figur nr. 7 fra bilag viser et flowchart over rutinen
Igen starter rutinen med at pushe værdierne fra general purpose registrene, samt status registret ind i stakpointeren. Det ekstra timer register TCNT1HH loades, inkimenteres og ligges tilbage i sram igen. Herefter clears overflow flaget. Normalt bliver dette flag automatisk clear'et når man returnere fra et interrupt, men Denne rutine er lavet på den måde at andre kodemoduler som ikke interrupt's kan kalde overflow rutinen. Igen inden der returneres popes de oprindelige værdier tilhørende status register og generalpurpurs registrene.

## 7.3 Macro's

Hvis andre kodemoduler skal måle en tid, bruger de også timer1. Tiden måles ved at gemme timer1's værdi ved et given tidspunkt og så senere sammenligne men en ny værdi fra timer1. Forskellen mellem de to timer værdier, repræsentere den tis som er gået imellem de to "måle"tidspunkter. Der er lavet nogle macro's som kan implementeres i andre kodemoduler, for at gøre brug af timeren let tilgængeligt. Macro'erne returnere henholdsvis: hele timer registret (24-bit), timer1's register (16-bit) eller det ekstra timer register (8-bit).

# 8 Motor omdrejninger (kode)

Dette kodemodulet er lavet til at måle tiden mellem pulsene fra motor encoderen. Tiden gemmes i sram, hvorved andre kodemoduler kan bruge værdien. Ved både at kende tiden imellem pulsene og antal af pulse på en motor omgang, kan motorens omdrejnings hastigheden beregnes. Til at måle tiden, benyttes en macro fra afsnit 7. Ud over at måle tiden imellem hvert puls, så bliver antallet af impulser fra motor encoderen også talt. Pulsene fra motor encoder vil altid repræsentere samme fysiske distance, uafhængigt af bilens hastighed. Antallet af pulser vil derfor repræsentere hvor stor en strækning bilen har tilbagelagt.

## 8.1 Kode

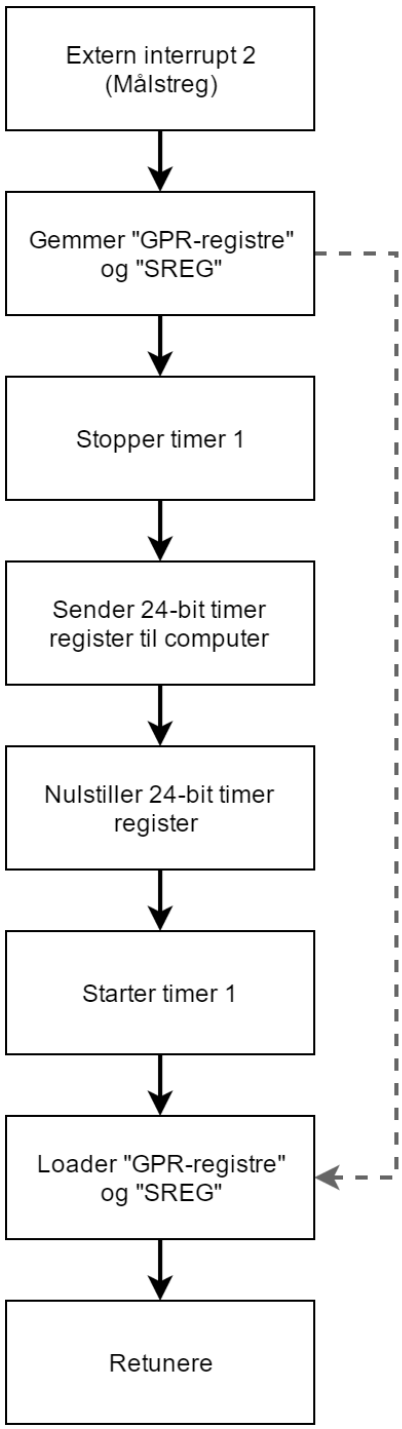
Koden består af en interrupt rutine, som bliver kørt når "Ekstern interrupt 1"trigges. Motor encoderen er tilsluttet det eksterne interrupt, så hver gang der kommer et puls fra encoderen, bliver rutinen kørt. Figur 8 fra bilaget, viser et flowchart over koden.

Først pushes værdierne fra general purpose registrene, samt status registret ind i stakpointeren. Dernæst loades den nuværende timer værdi. Timer værdien trækkes fra en "gammel"timer værdi, fra forrige encoder impuls, for at finde forskellen mellem de to timer værdier. Forskellen mellem de 2 værdier, repræsentere den tid der er gået mellem de 2 pulse. Tiden mellem de 2 pulse ligges ind i SRAM, så andre kodemoduler let kan benytte værdien. Herefter inkrementeres distance registret, for at holde styr på hvor langt bilen har kørt. For at have den rigtige timer reference værdi, næste gang denne rutine kaldes, overskrives den "gmale"timer værdi med den "nuværende"timer værdi. Inden der returneres fra rutinen, popes de oprindelige værdier for henholdsvis general purpus registrene og status registret.

## 8.2 Macro's

For at gøre de 2 værdier lettilgængeligt for andre kodemoduler, er der lavet 2 macro'er. Den første makro returnere tiden mellem pulsene og den anden macro returnere den tilbagelagte distance i for af et antallet af encoder pulser, siden start linjen blev passeret.

# 9 Bilag-phil




Figur 6: Flowchart over første rutine i lap-timer

## 9.1 Prescaler udregninger

Formlerne er de samme for de forskellige prescalere, det eneste som ændre sig er prescaler værdien. Som eksempel bliver prescaler værdien 256 brugt. For at beregne tiden mellem hvert clock count bruges følgendene formel:

$$\Delta t = \frac{1}{16000000/prescaler}$$

$$\Delta t = \frac{1}{16000000/256} = 1.6 \cdot 10^{-5}$$

Clock count's pr. puls fra motor encoder, beregnes ved:

$$cpp = \frac{1}{Motorhastighed}/\Delta t$$

$$cpp = \frac{1}{2000}/1.6 \cdot 10^{-5} = 31.25$$

For at kunne beregne hvor lang tid timeren kan være aktiv inden den overflower, skal man først vide hvor mange count's timeren tæller på et sekund. Dette beregnes ved formlen:

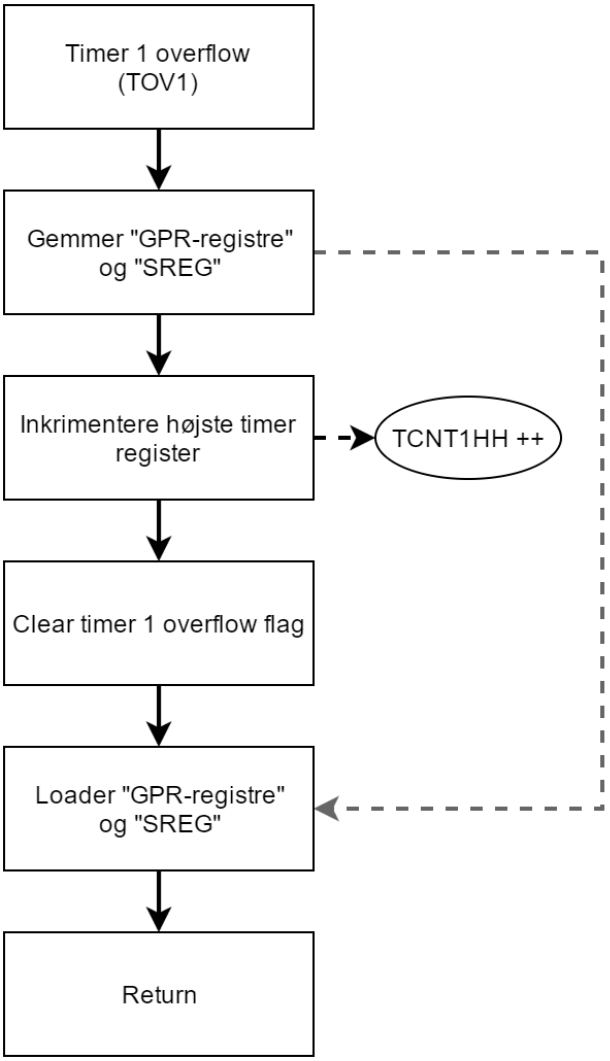
$$cps = \frac{1}{\Delta t}$$

$$cps = \frac{1}{\Delta t} = 62500$$

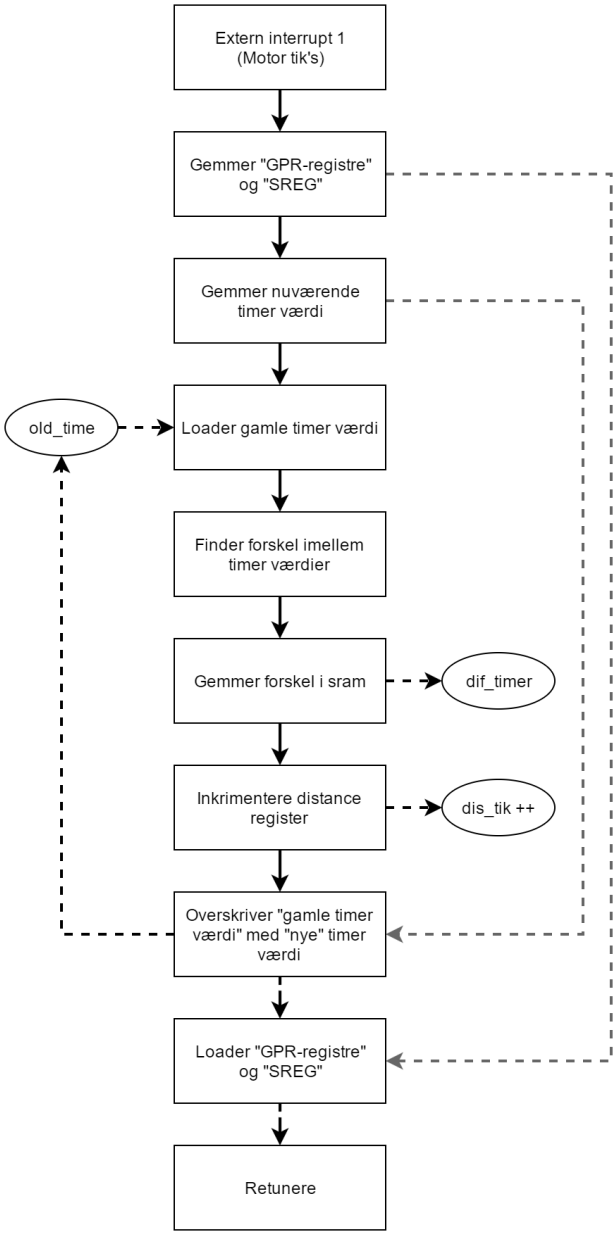
Ved at dividere timer registrets max værdi med antal counts pr. sekund, fås hvor langtid timeren kan køre inden overflow:

$$os = \frac{2^{24}}{cps}$$

$$os = \frac{2^{24}}{62500} = 268.44$$




Figur 7: Flowchart over overflow rutine i lap-timer




Figur 8: Flowchart over motor hastighed's kode