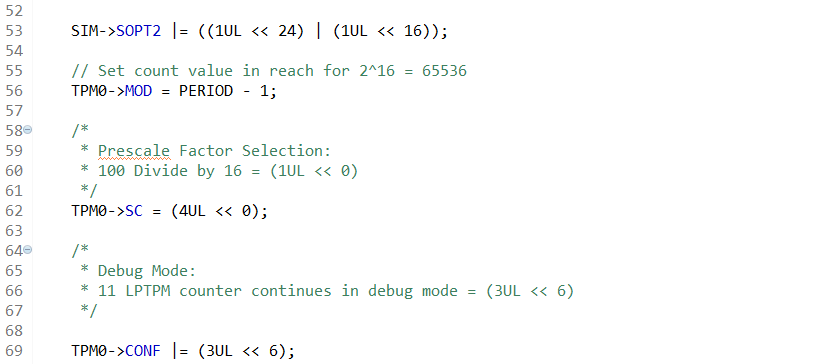
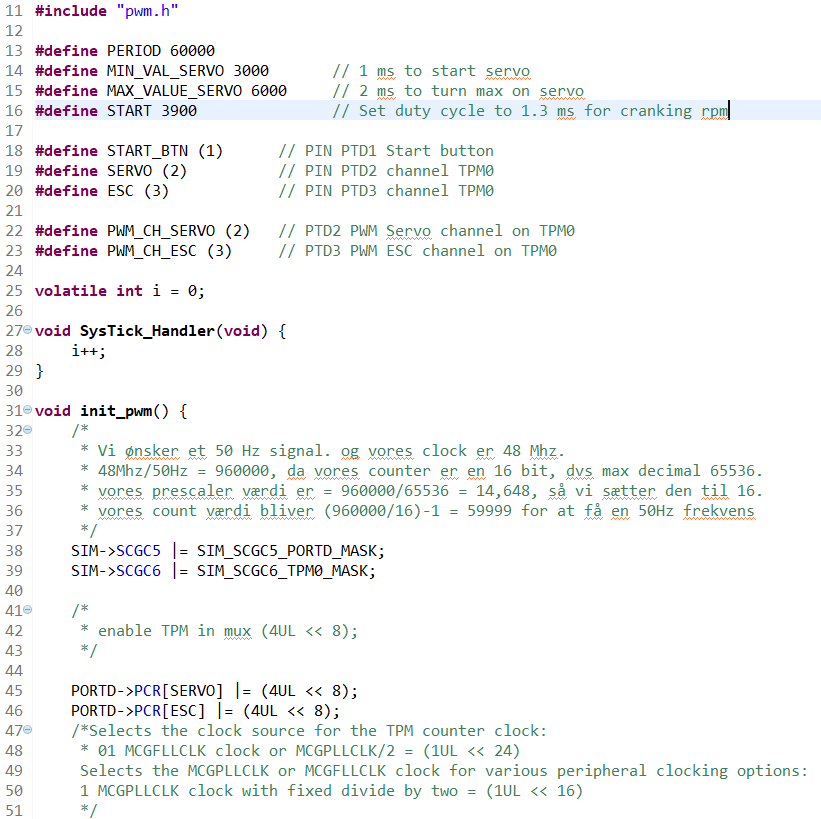
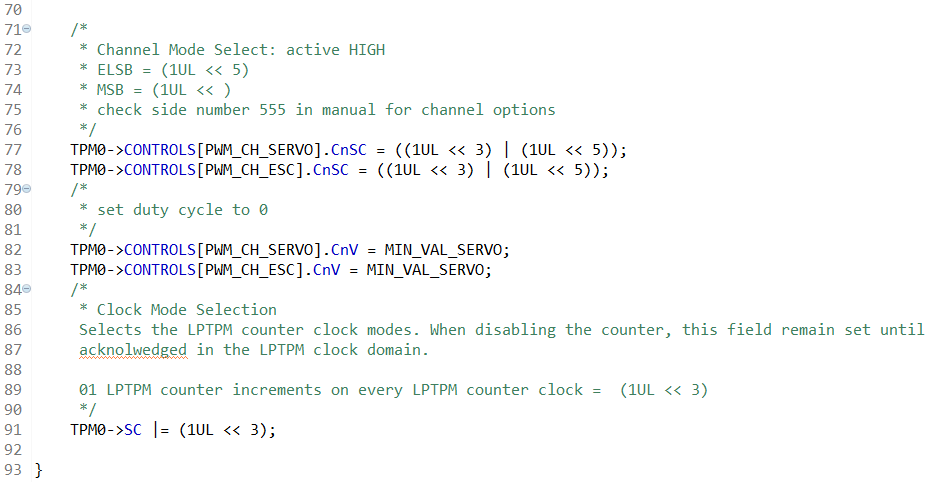
# Analyse/Design til servo motor og ESC driver

For at få vores servo motoren til at fungere, kræves der et signal på 50 Hz og ved at justere PWM mellem 1 ms til 2 ms aktiv høj, hvilket fremgår fra databladet på servo motoren (SG90 micro servo). Kan vi justere positionen på servo motor fra 0 – 90 grader.

Dette signal bruges også til at justere hastigheden på vores BLDC-motor, via vores ESC. De værdier vi brugte, var 1 ms til stop af BLDC-motor og værdien 1.3 ms, for at kunne starte motoren. Denne værdi 1,3 ms, kom vi frem til via brug af et scope med et wavegen, hvor vi kom frem til at, at ved denne værdi, havde motoren nemt ved at starte. Følgende krav blev opsat, for at vi kunne verificere at det virkede efter hensigten

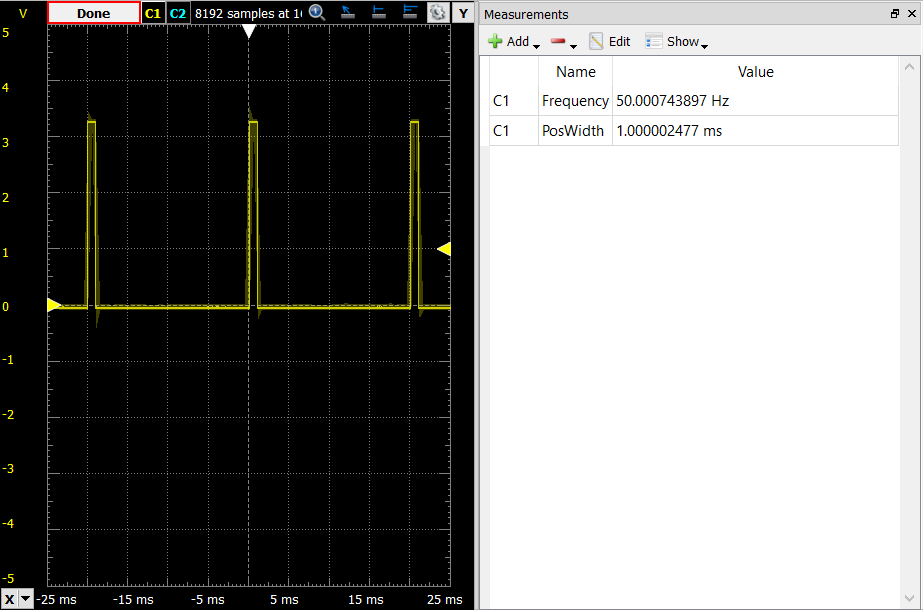
* 50 Hz signal, 20 ms periode (Krav nr. 2.1.3.8.1)
* Skal kunne justere puls mellem 1 – 2 ms. (Krav nr. 2.1.3.8.2)



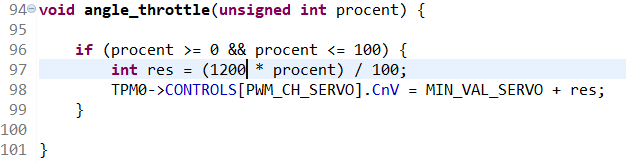


Figur 1 opsætning af servo og ESC driver

Ved at opsætte analog Discovery til på pin PTD3, vises at vores init funktion virker efter hensigten, med 50 Hz og 1 ms puls. Og hermed lever op til følgende krav 2.1.3.8.1 og 2.1.3.8.2, der er beskrevet på forrige side.



Herefter har vi implementeret en funktion til at styre vores servo motor. Den tager et argument, som er i procent, hvor meget den skal dreje, i forhold til om den skal give fuld gas (100 %) eller bare kører i tomgang (0 %). Men da vi fik monteret servo motor, fik vi testet at ved 1,4 ms signal til servo, at dette var fuldgas, dette blev derefter tilpasset i koden. Så vores max værdien der kunne sendes ud er 1,4 ms.



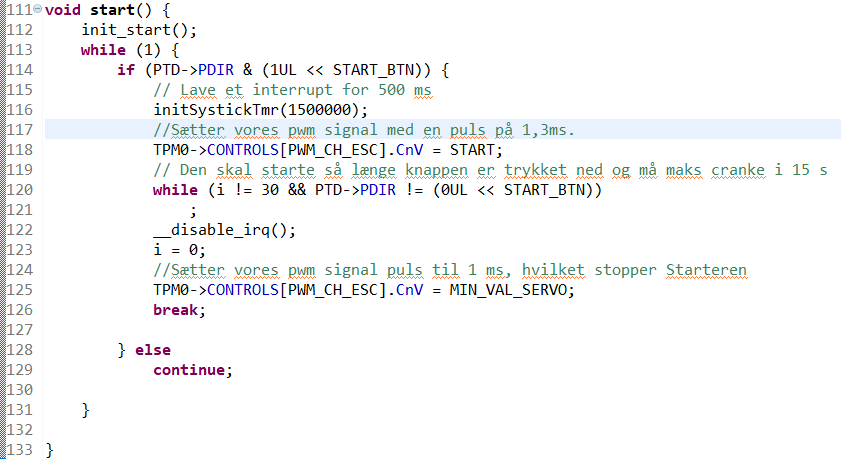
Figur 2 Kode til at styre gasspjældet

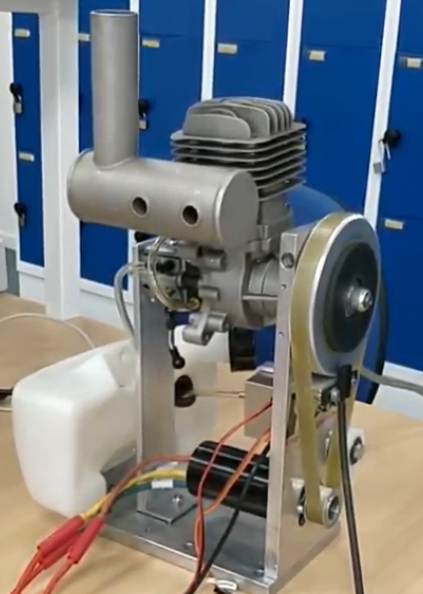
## Start af motoren.

Da den driver vi brugte til at styre servo motoren er den samme, som vi skal bruge for at kunne få vores BLDC-motor i gang, da den fungere som el-starter til motoren. Blev der implementeret en pin ekstra, i driveren, der er vist i koden se figur ??. Herefter lavede vi en start funktion, der skulle bruges til at starte motoren.

Flow for opstart af motor, ved funktionen start():

1. Startknap initialiseres
2. Afventer at brugeren trykker på knappen
3. En timer opsættes, som tæller vores ’i’ værdi op, når der er gået 500 ms.
4. Så prøver motoren at starte benzinmotoren, og det vil den gøre så længe start knappen er trykket ned (dog i maks. 15 sek)
5. Efter 15 sek vil opstartsprocessen afbrydes.





Figur 3 BLDC-motor forbundet til motoren med en tandrem

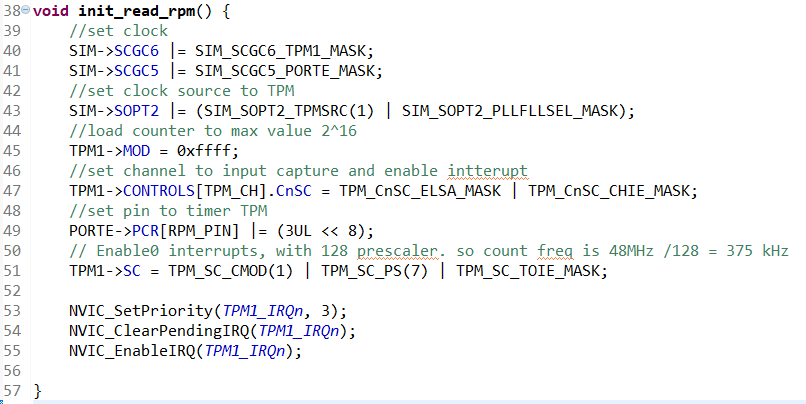
## RPM-detektering

Implementering af koden til at aflæse rpm, der skal bruges til vores PID-regulering. Har vi taget udgangspunkt i at aflæse et tacho signal som kommer fra vores tændspole. Dette signal lave en høj spænding ved hver omdr, og går lav igen.

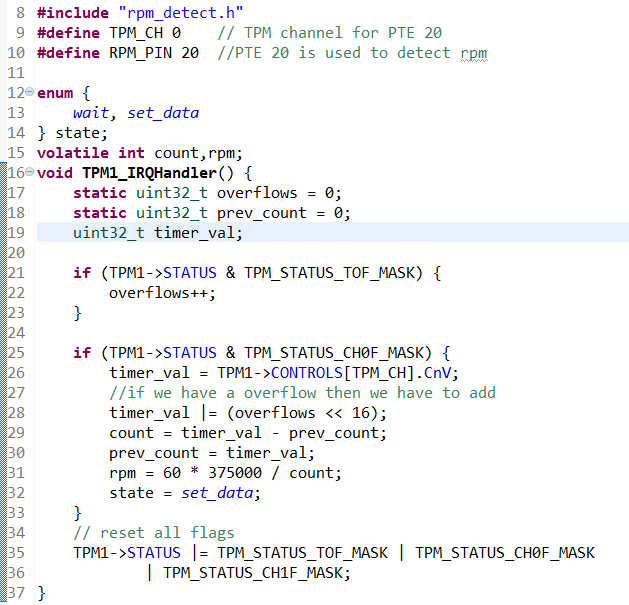
For at afkode dette signal, bruges der input capture, der er implementeret i TPM periferenheden på kl25. Funktionen input capture, fungere ved at den tælle op til hver ’rising edge’ og gemmer denne værdi. Og når vi kender værdien og frekvensen vi tæller op med, kan vi omregne dette til RPM.

Ved formlen: *60 sek \* frekvens /input capture værdien = RPM*

Selve koden er vist her forneden, hvor vi bruger interrupts, til at opfange overflows og når vi får et rising edge, gemmer den værdi i CNV-registeret.



Her initialiseres vores TPM1 modul. Vores frekvens der bruges til at tælle op med, bliver nedskaleret, til 375 kHz. Dvs. imellem hver overflows går der 5,75 Hz, hvilket passer med vores måle område (17 - 167 Hz) vi arbejder i.



Her ses vores handler, hvori RPM bliver beregnet ud fra værdierne fra TPM1 perifer enhed. Og videre i koden kan man detektere om der er kommet ny RPM-data, ved bruge enum ’state’. Da det tager minimum en omgang på krumtappen, før den har beregnet en ny værdi.