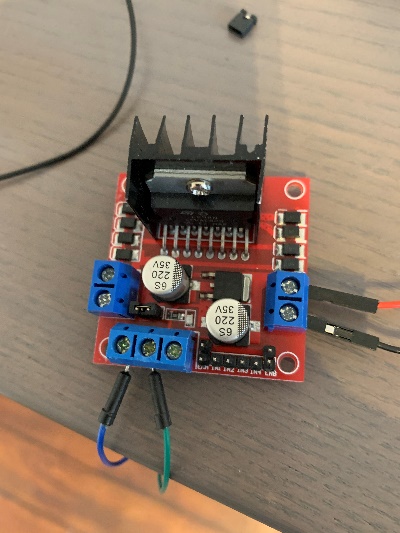
Denne vejledning beskriver hvor motorcontrol er blevet lavet til STM32.

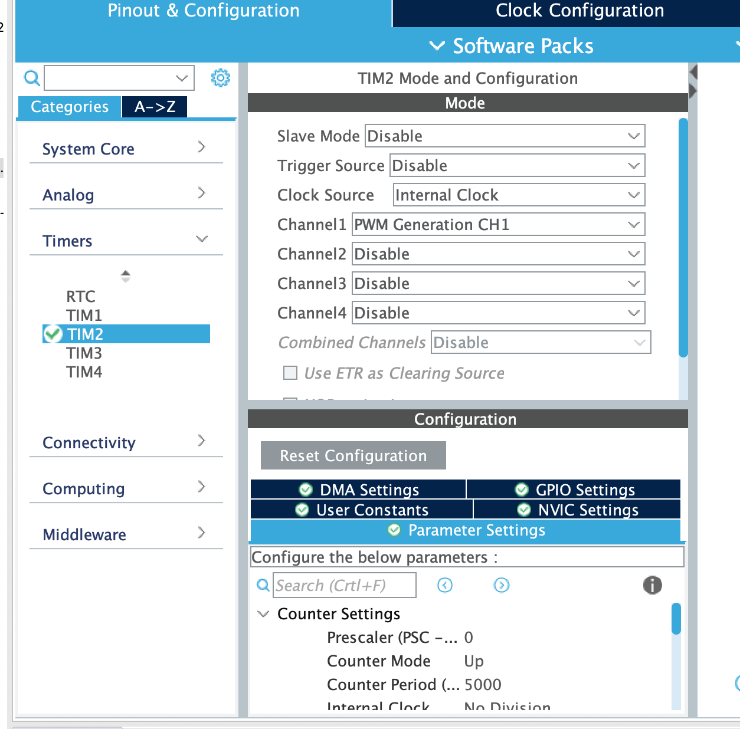
Vi startede med at bruge en H-bro til at styre retning og hastighed af motoren:



Herefter gik vi ind i STM32CubeIDE og opsatte pins. Vi skal bruge en timer i STM32 til at styre frekvensen af PWM signalet. PWM signalet bestemmer hastigheden af motoren. Frekvensen bestemmer hvor glat den kører så at sige. Den skal være høj nok til at sikre at motoren kører med en jævn hastighed.

STM32 har flere forskellige timere, som varierer i kompleksitet. Der er kun specielle pins, hvor du kan anvende de forskellige timere.

Vi har anvendt timer2, som hører til forudbestemte pins. Timer2 bruger vi til at styre PWM signalet. Timer2 har flere kanaler, som har mange funktioner, som kan bruges til andre ting samtidig. Billedet under illustrere de forskellige timere som kan bruges og timer2’s funktioner:



Figur TIM mode og configuration

Billedet illustrer at vi har valgt PWM på kanal 1 og at clock source er intern clock.

Nederst på billedet er counter perioden sat til 5000 og pre-scaleren til 0 (Husk 0 svarer til at dividere med 1, 2 svarer til 3 osv. Det samme er gældende for counterperioden, men her er det mindre vigtig da 5000 eller 5001 er næsten det samme). Det gør, at når vi i denne STM32 arbejder med en frekvens på 72 MHz, så får vi en frekvens af vores PWM signal på .

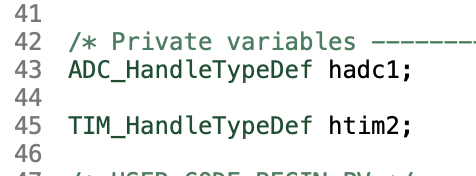
**OBS. Husk pre-load ellers tæller den kun en gang.**

Husk desuden under system-core -> RCC -> HSE -> Crystal/ceramic resonator. Det gør at timerne kan bruges op til 72 MHz i dette tilfælde. Havde vi valgt LSE så ville timerfrekvensen være lavere.

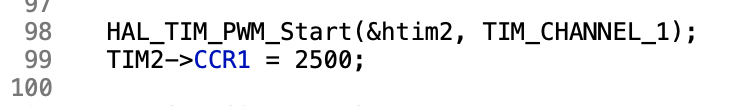
Det sidste step er at enable ADC til en pin. Her er der to ADC og de kan bruges på forudbestemte pins, men vælg blot en af dem og brug en af de passende pins.

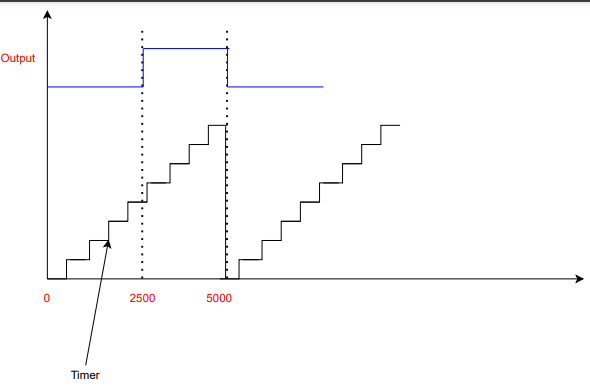
Nu er setup færdig, så tryk på gem og autogenere kode.

Nu bliver du ført ind i c-koden. Der er nu autogeneret to handlers til ADC og timeren. Dette er så vi tilgå de to egenskaber så at sige:

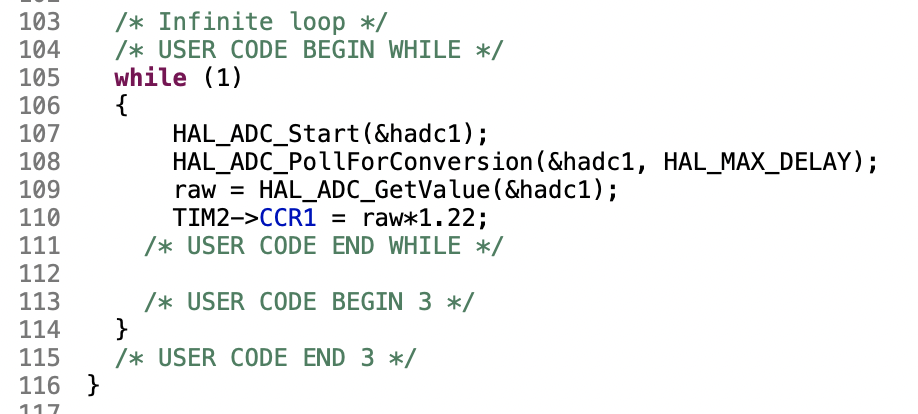


Herefter er timeren kun opsat, men ikke startede, det gør vi så med linje 98:

Linje 99 sætter registeret CCR1 (Capture-Compare-Register-1) til værdien 2500, som var den halve periode da vi definerede en periode på 5000 (se nederst på Figur 1 TIM mode og configuration ). Dette giver os 50% duty cycle, som startværdi. Det vil sige, at counteren tæller fra 0 til 5000, men når den kommer til 2500 så toggler værdien, som illustreret herunder:



Nu går vi så ind i vores while-løkke, som ser ud som følger:



Linje 107 er hvor vi starter ADC ved at refere til ADC-handleren, som beskrevet tidligere.

Linje 108 beder vi ADC konvertere værdien fra ADC, hvor vi venter ”HAL\_MAX\_DELAY”, som er den maksimale tid vi kan vente på en konvertering.

Linje 109 beder vi om den konverterede værdi og gemmer den i raw.

Linje 110 ændre vi værdien i CCR1, hvilket ændre duty-cyclen. (Vi ganger med 1,22 fordi ADC har en 12-bits opløsning, som svarer til 0-4095 værdier. 4095 \* 1,22 giver så 5000 og derved kan vi få den fulde duty-cycle.).

Nu kan du builde project og run programmet og motoren skulle kører under forudsætning af du har opstillet pins rigtig og du har flyttet jumperen og reset korrekt.