Forår 2019

Laboratorieøvelse
1

Dato: 28/02/2019

#1 Stud.nr.: 201404212 Navn: Rasmus Vesterheim
#2 Stud.nr.: 201710712 Navn: Marcus Bech
#3 Stud.nr.: 201710685 Navn: Martin Krøjmand

Indledning

Formålet med denne øvelse er at blive bedre bekendt med PSoC igennem praktiske deløvelser, hvor vi først vil kontrollere PWM på en DC motor. Efterfølgende vil vi bruge en H-bro til at kontrollere retning og PWM på den samme DC motor. Til sidst vil vi bruge PSoC'en til at styre en stepper motor.

Experiment 1: PWM control of a DC motor

I denne del af øvelsen vil vi kontrollere PWM af en DC motor. Dette vil blive gjort i form af kommandoer i en terminal, der igennem PSoC'en varierer PWM.
På figur 1 nedenunder ses det, at udgangsstrømmen fra vores PSoC har en maksimal værdi på 41 mA. Da DC-motoren, vi bruger, trækker en strøm, der gennemsnitligt er 35 mA, har vi valgt at bruge et 9V batteri.

I_{GPIO}	GPIO current		–30	–	41	mA

Figure 1 Billede af udgangsstrøm fra PSoC hentet fra datablad[1]

Imellem PSoC og DC-motor bruger vi et udleveret print, hvor der er fire MOSFET, som har til opgave at drive motoren. I denne opgave har vi kun brug for en enkelt MOSFET. På figur 2 ses det udsnit af printet, som vi har brug for til at styre PWM på DC-motoren.

[1] https://www.cypress.com/file/45906/download, side 67

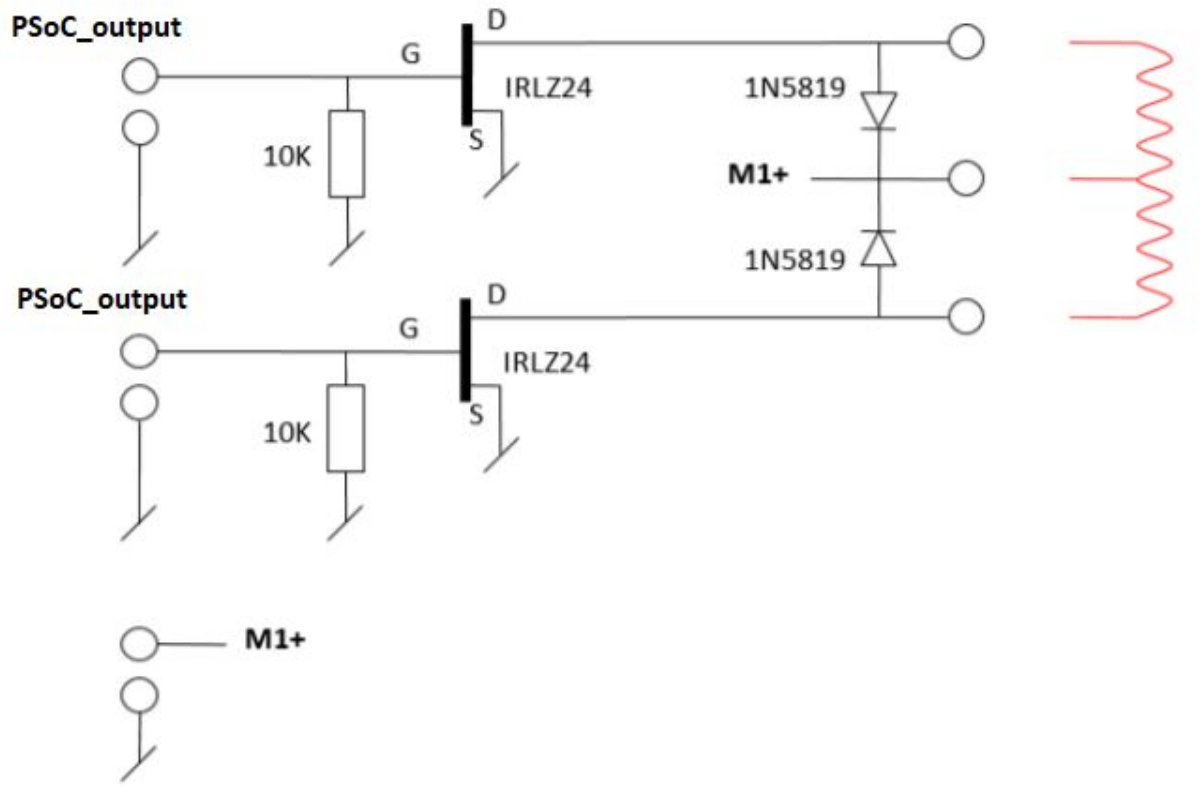

Figure 2 BIllede af kredsløbet, der styrer DC-motoren. Til venstre ses de to udgange fra PSoC, der løber igennem to transistorer

M1+-terminalen på figur 2 er indgangen fra spændingskilden, mens PSoC_output er udgangen fra PSoC, hvorigennem PWM bliver kontrolleret.
Koden til styring af PWM-signalet kan ses i bilaget.

Koden er lavet således, at man igennem terminalen kan styre duty-cycle i steps på 10%. Altså 0%, 10%, 20% osv. op til 100%. Dette ses på kodeudsnittet på figur 3.

```
106  void increaseSpeed()
107  {
108      UART_1_PutString("Increasing speed\r\n");
109      if(speed<10) {
110      speed=speed+1;
111      PWM_1_WriteCompare(speed);
112      }
113  }
```

Figure 3 Kodeudsnit af funktionen inceaseSpeed, hvor variablen speed inkrementeres med 1, hvorefter den nye værdi sendes til PWM-modulet.

Vi har implementeret hastighed ved at bruge en variabel (speed), som kan varieres fra 0 til 10. Denne værdi bruges som parameter til funktionen WriteCompare(int). Dette vil netop sørge for at variere duty-cycle fra 0-100% i trin af 10%.
Der er desuden også mulighed for at stoppe motoren helt, hvis man skriver '0' i terminalen. Koden for stop ses på figur 4.

```
132  void stop()
133  {
134      PWM_1_WriteCompare(0);
135  }
```

Figure 4 Kodeudsnit af funktionen stop. Stop er implementeret ved at sætte duty-cycle til 0%. Funktionen start er til gengæld lavet ved blot at sende speed variablen til PWM-modulet

Herefter kan man starte den op igen med den samme duty-cycle, motoren havde, før man stoppede den.
På figur 5 ses opstilling af kredsløbet, hvor printet er forbundet til forsyningskilde via bananstikkene. PSoC'en styrer duty-cycle og er desuden forbundet fælles ground.
På figuren er udgangssignalet til motoren først ført til fumlebræt, da vi på denne måde kan lave målinger af signalet på oscilloskop.

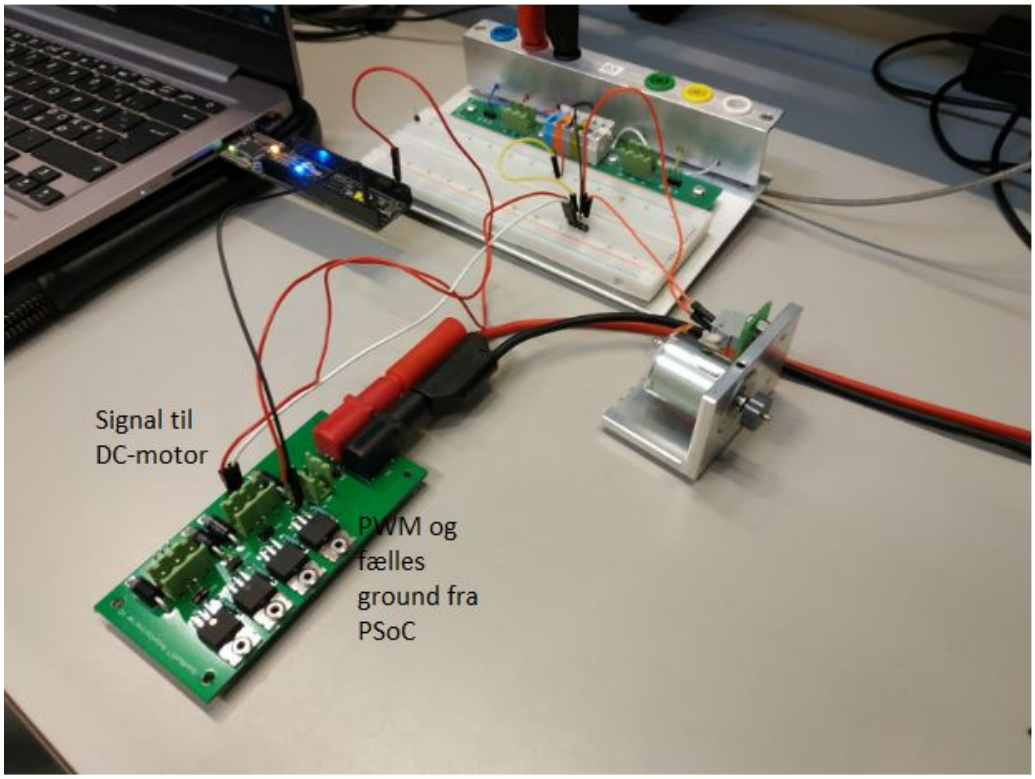

Figure 5 Billede af opstilling af kredsløbet. De to bananstik i enden af printet går til forsyningskilde. Den røde og sorte ledning til højre på printet er PWM-signal og ground fra PSoC, mens den hvide og orange ledning er ført til fumlebræt, hvorfra de går til

På figur 6 og 7 ses måling af PWM på udgangen fra PSoC.

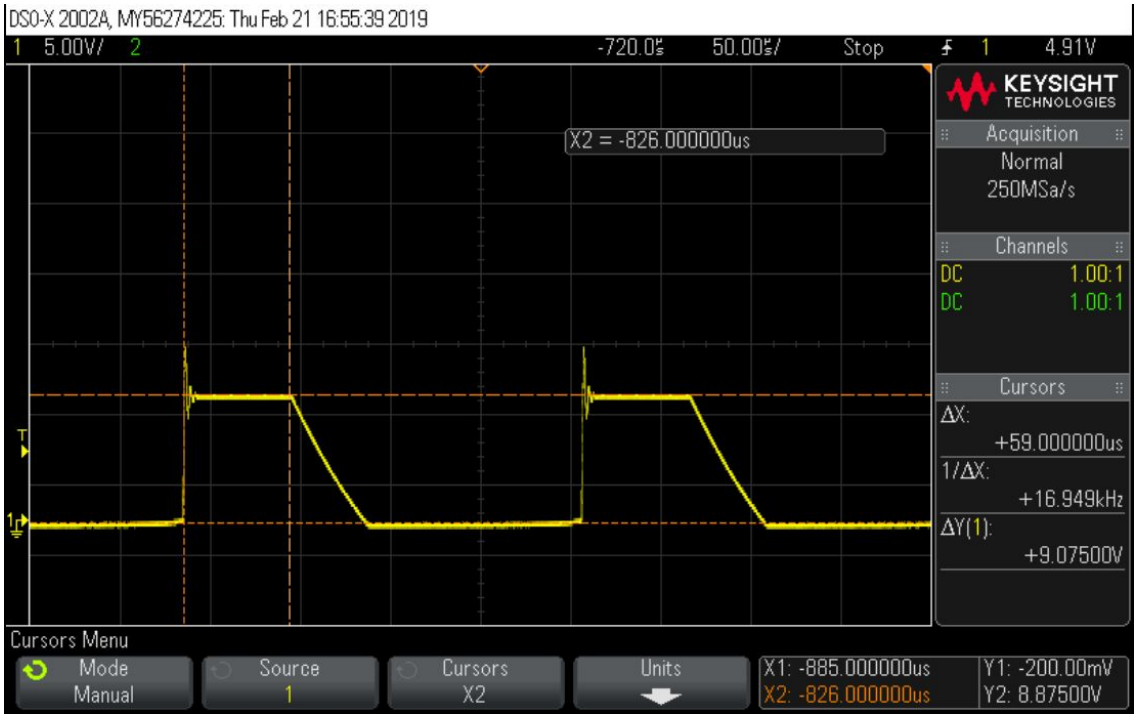

Figure 6 Billede af måling af duty-cycle på udgangssignalet fra PSoC. Her er målingen lavet over den tid, hvor signalet er højt. Den er målt til at være 59 mikrosekunder.

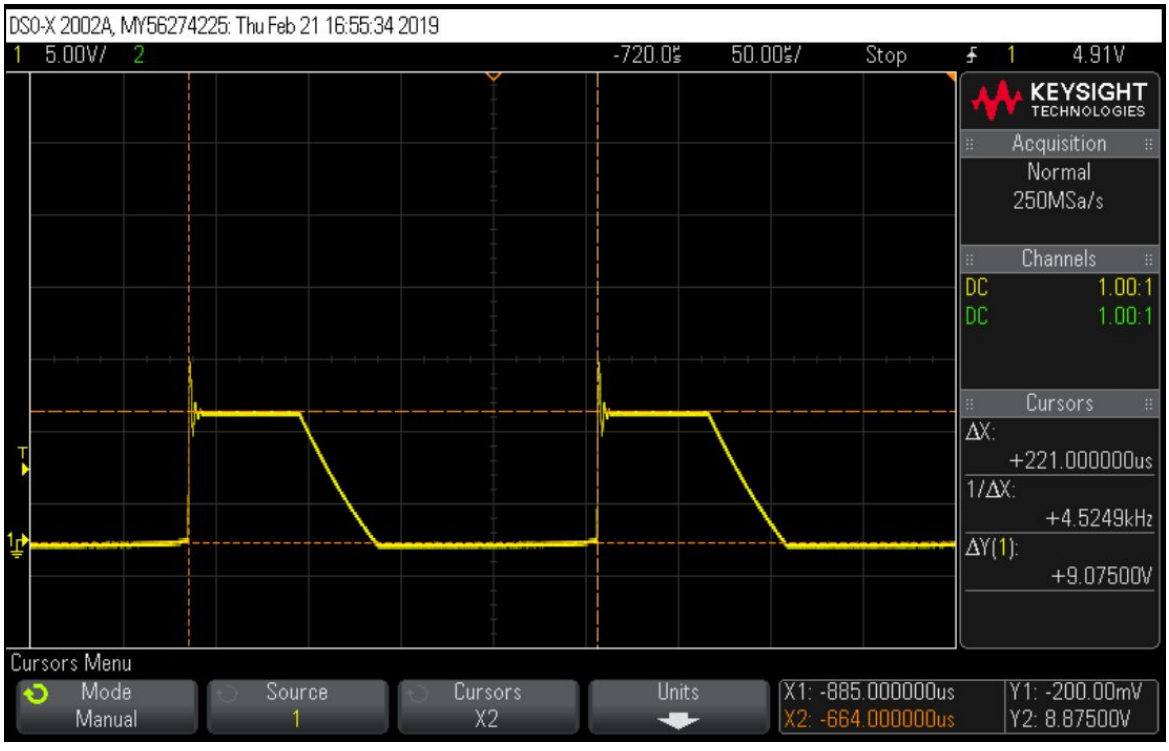

Figure 7 Billede af måling af duty-cycle på udgangssignalet fra PSoC. Her er målingen lavet over en hel periodetid Den er målt til at være 221 mikrosekunder.

Ud fra disse målinger er der målt en duty-cycle på godt 27%, hvilket stemmer ret godt overens med de 30%, der var valgt igennem terminalen. Forskellen på de 3% skyldes bl.a., at firkantssignalet fader ud ved falling edge, da den bliver belastet med MOSFET printet og motoren. Det gør, at oscilloscopet måler en lidt anerledes periodetid for dette firkantsignalet. Jævnfør fallingtime er lang.

Motoren får kun en hvis procent dutycycle, så når vi måler henover motoren ser vi at den aflades og at hastigheden øges igen, altså en dutycycle. På figur 8 ses det at kurven svinger lidt og det er motoren der har denne indvirkning på signalet. Denne måling er lavet med en frekvens på 4.5k Hz.

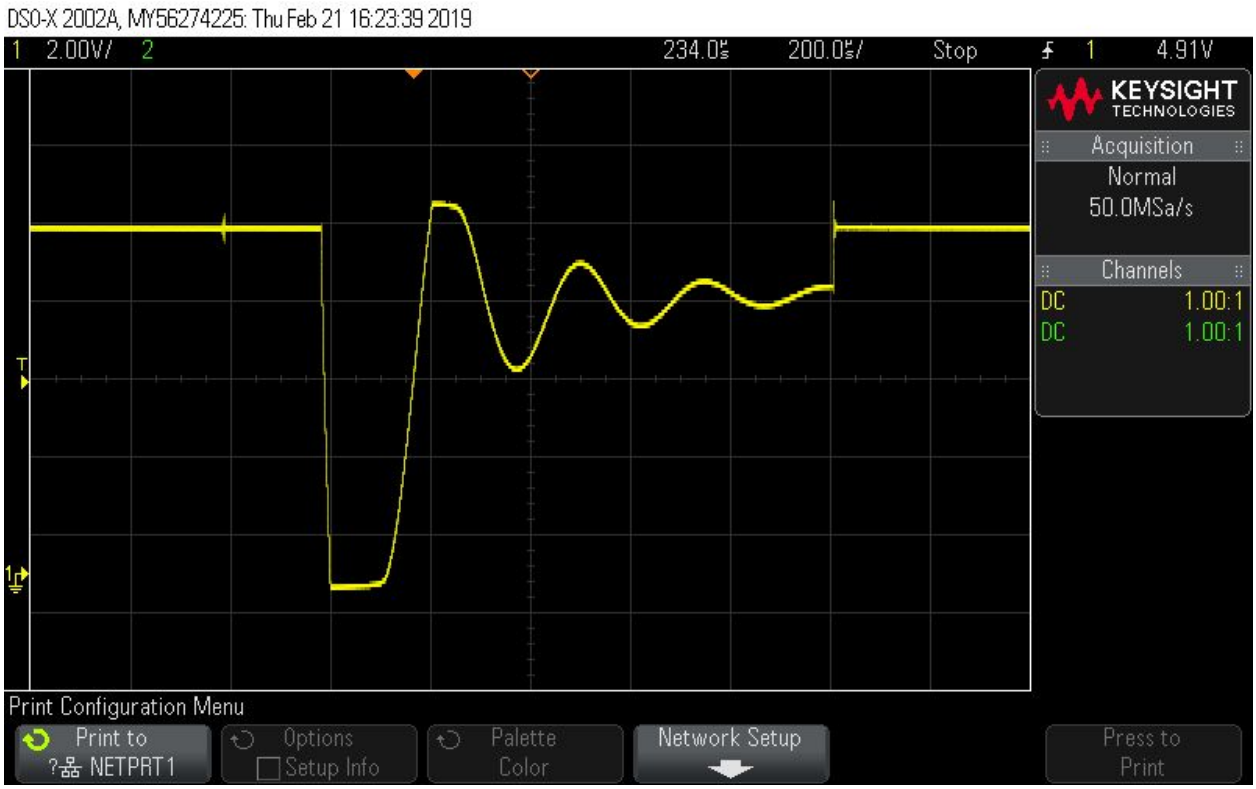

Figure 8 Billede af måling over motoren, når der bliver sendt et signal ud fra PSoC'en med en frekvens på 4.5 kHz

På figur 9 ses en måling med en frekvens på 1k Hz. Her er det tydeligt, at der er mere tid til, at de 9V bliver afladet gennem motoren. Det underdæmpet system, der opstår, når firkantsignalet går lavt tilnærmer sig en stationær værdi. Det kan ses, at en længere periode for firkantsignalet vil give afladningen af "spolen" mere tid til at tilnærme sig en stationær værdi. Det vil altså sige, at strømmen gennem "spolen" vil nå at blive højere, når der er en lang periode, end hvis der er en kort periode.

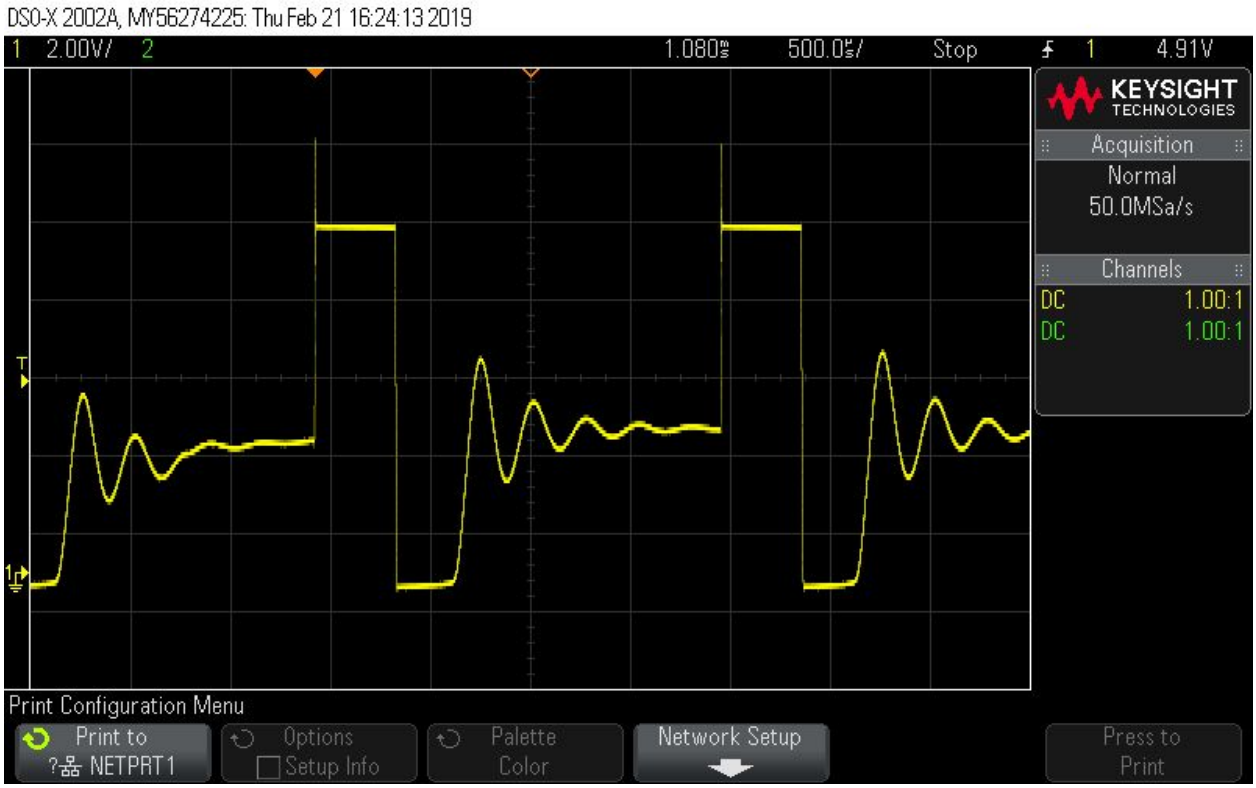

Figure 9 Billede af måling over motoren, når der bliver sendt et signal ud fra PSoC'en med en frekvens på 1 kHz

Vi har altså i denne del af øvelsen kunnet styre PWM af DC-motoren. Vi har desuden lavet funktioner til at stoppe og starte motoren igen.
Der kunne ikke mærkes en signifikant effekt ved at ændre frekvensen af PWM til en lavere. Dette har formentlig noget at gøre med, at frekvensen kun blev ændret fra 4,5 kHz til 1 kHz. Herudover var rotationer pr. tid i forvejen så høj, at forskellen ikke vil være synlig. Vi kan kun måle effekten på oscilloskopet.

Experiment 2: DC motor rotation direction

I denne del af øvelsen vil vi tilføje to nye funktioner til styring af DC-motoren. Vi vil bruge en H-bro til at kontrollere retning af motoren. Styringen vil foregå ved at skrive til tre forskellige pins på PSoC, som føres ind i det udleverede print med en H-bro. Herefter vil signalet løbe ind i den samme DC-motor, vi brugte til eksperiment 1.
Opstillingen af øvelsen kan ses på figur 10.

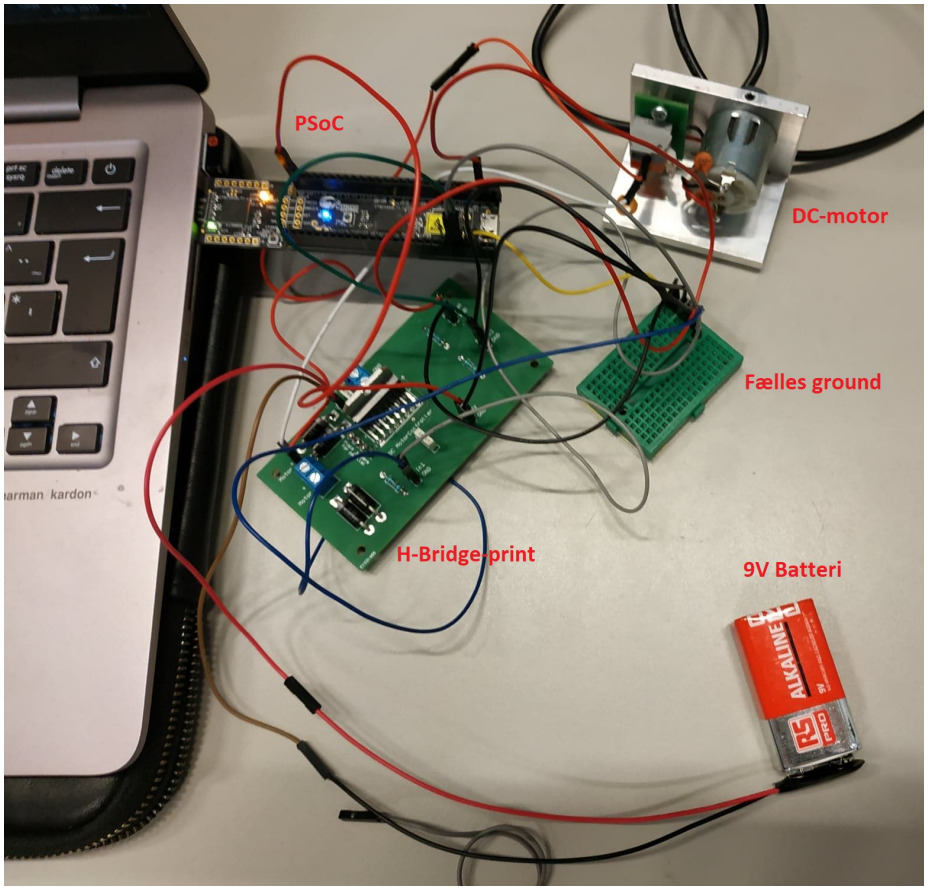

Figure 10 Billede af opstilling af eksperimentet. PSoC sender EN, IN1 og IN2 ud til printet med H-bro, som så føres ind i DC-motoren

Vi har i programmet sat de tre pins op til funktionerne, der er taget vilkårligt frie pins, som ikke var sat op til noget. Disse pins er, i programmet, sat op uden tilslutning til hardware. Der er tilføjet “IN1” og “IN2”, hvor at disse pins bestemmer retningen af motoren ud fra H-broens opsætning.
Programmet er lavet ud fra opgave 1, hvor vi har tilføjet de ekstra funktioner, men ellers er alt andet ens. Vi vil med det udgangspunkt implementere funktion for rotationsretningen.
PWM signalet er forbundet med enable signalet, så kan vi med de tre signaler styre hastigheden og retningen på DC motoren.

På figur 11 nedenunder ses et kodeudsnit af funktionen driveForwards(), hvor vi sætter EN og IN1 høje, mens IN2 bliver sat til 0.
Funktionen for at køre baglæns er lavet på tilsvarende måde, men hvor In1 og In2 har omvendte værdier.

```
115  void driveForwards()
116  {
117      UART_1_PutString("Set direction: forwards\r\n");
118      Pin_EN_Write(1);
119      Pin_LN1_Write(1);
120      Pin_LN2_Write(0);
121
122  }
```

Figure 11 Kodeudsnit af funktionen driveForwards

Disse funktioner er ligeledes testet med denne opstilling.

Så har vi lavet stop funktionen som skriver ud til de tre pins, her sættes alle tre signaler højt (1). Så får vi break funktionen som også ses i tabel1.

```
void stop()
{
    UART_1_PutString("Stop\r\n");
    Pin_EN_Write(1);
    Pin_LN1_Write(1);
    Pin_LN2_Write(1);
}
```

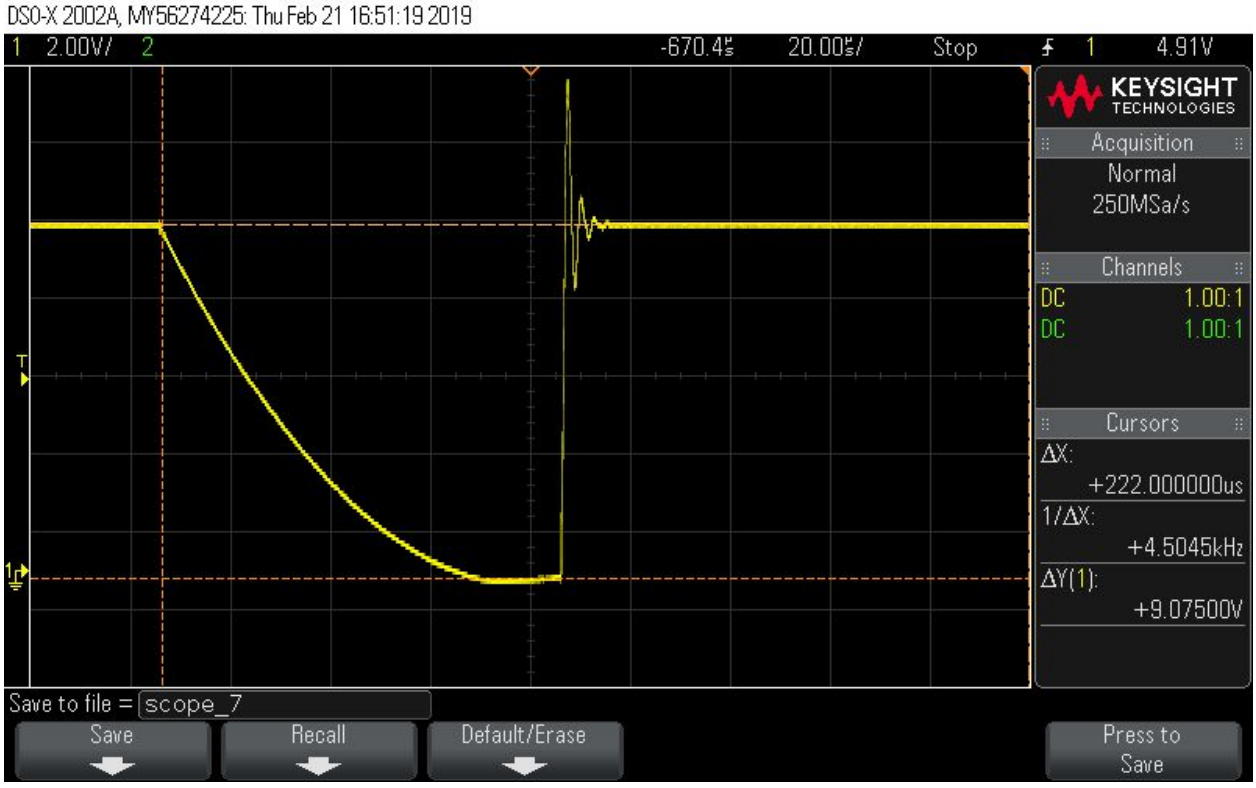

Figure 12 Billede af måling, hvor det ses det, at vi tilfører motoren 9V

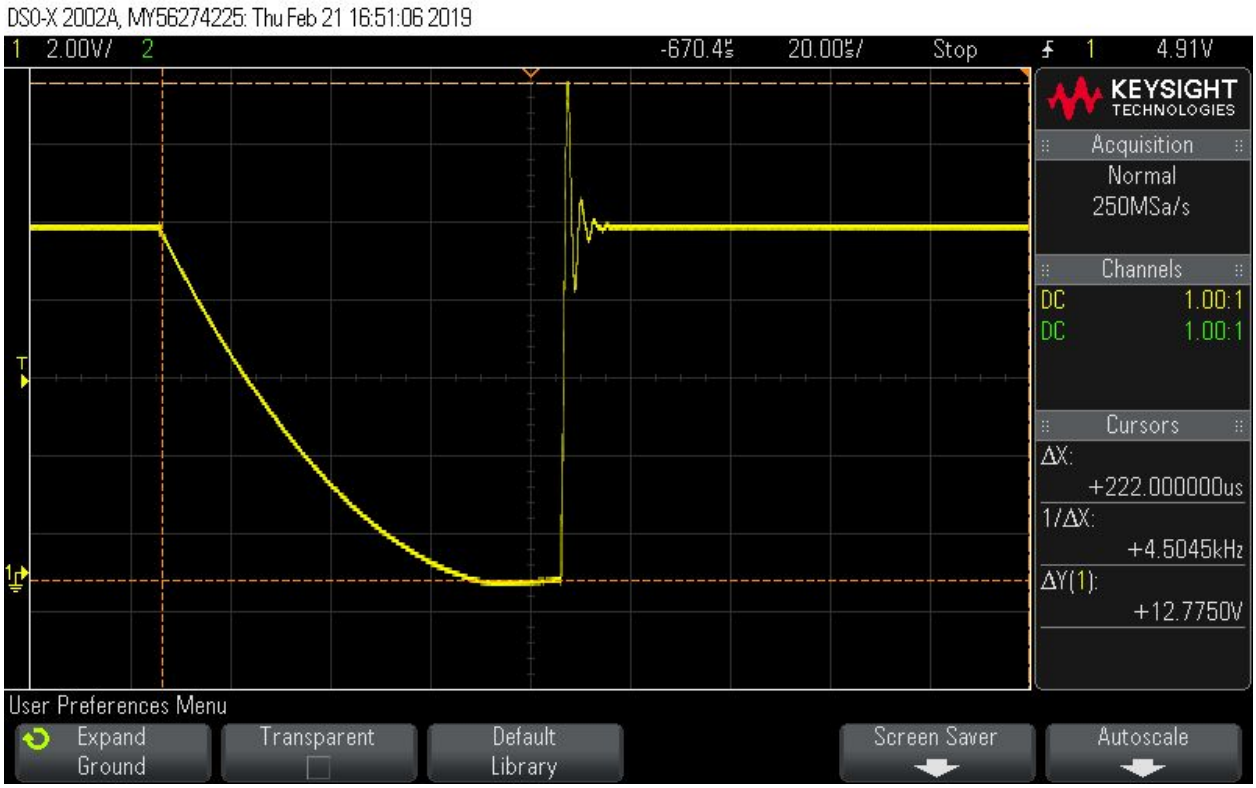

Figure 13 På dette billede ses det, at motoren svinger helt op til 12V

På tabellen nedenunder ses forholdet mellem værdierne af IN1 og IN2 og den tilsvarende virkning på motoren. Tabellen er opstillet ud fra databladet for L298[2].

IN1	**IN2**	**Retning**
0	0	Break
0	1	Forward
1	0	Backward
1	1	Break

Tabel 1 - oversigt over motor funktioner samt værdi af pins. Dette er forudsætning af, at Enable er 1.

2 https://blackboard.au.dk/bbcswebdav/pid-2111831-dt-content-rid-6126033_1/courses/BB-Cou-UUVA-83105/BB-Cou-UUVA-69270_ImportedContent_20170609013941/L298.pdf

Experiment 3: Stepper motor control

I dette eksperiment skal der styres en steppermotor. Der er valgt at bruge en 28BYJ48-5V motor, som selvsagt kører på en 5 V forsyning.
Der skal styres noget forskelligt i henhold til motoren:

- Stop motor
- Forøg/sænk fart
- Skift retning
- Skift mellem Wave-drive, full-step og halfstep
- Roter en omgang frem/tilbage

Der er brugt det MOSFET driver circuit fra laboratoriet. Opstillingen af forsøget kan ses herunder:

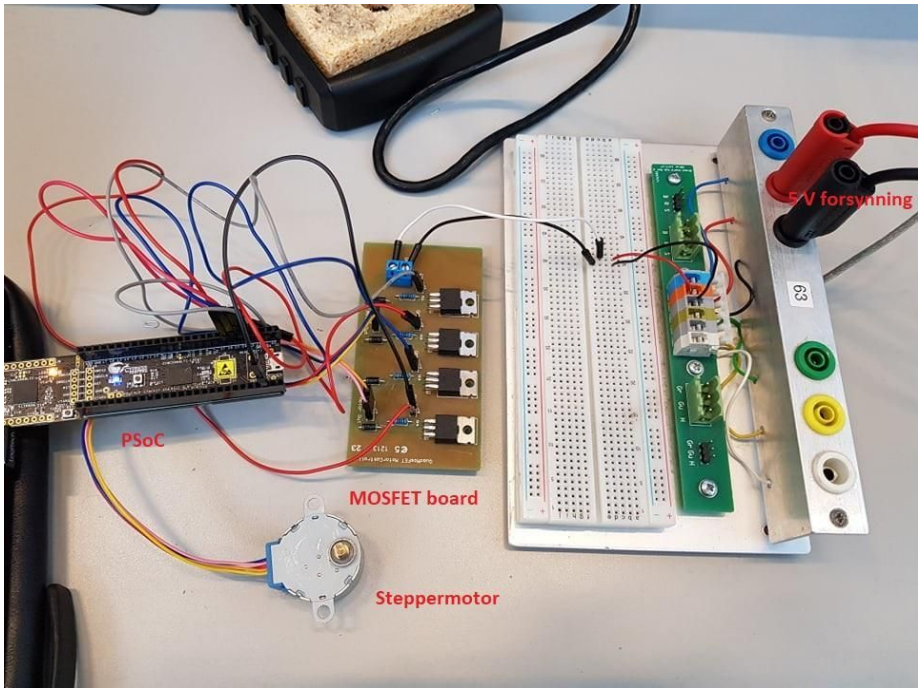

Figure 14 På billedet ses opstillingen med en ekstern 5V strømforsyning til steppermotoren, så der ikke trækkes for meget strøm ud af PSoC'en

Strømforsyningen er sat til 5 V, der tages fra en funktionsgenerator, så der ikke trækkes for meget strøm ud af PSoC'en, selvom det også virkede ganske fint. Det er dog en mere holdbar løsning at trække fra en funktionsgenerator, der er sikker på at kunne forsyne motoren.

Modes

Der bruges forskellige states til at udtrykke hvilken mode, motoren er sat til og hvilken retning, stepper motoren kører. Det kan ses i koden, som er vist herunder:

```
typedef enum
{
    FORWARD,
    BACKWARD
} DIRECTION;

typedef enum
{
    WAVE,
    FULL,
    HALF
} MODE;
```

Figure 15 Kodeudsnit af definitioner af enum, som bliver brugt til at implementere statemachines

Steppermotoren består af 4 spoler, der via elektromagnetisme kan rykke rotoren i midten rundt ved at aktivere de forskellige spoler. Dette kan gøres på overordnet 3 måder.
Der er Wave-drive. Her er der 4 steps. Hver spole bliver aktiveret en efter en. Når alle 4 spoler har været aktiveret startes der forfra, og sådan fortsætter det. Billedet herunder fra rs-online.com illustrerer det godt:

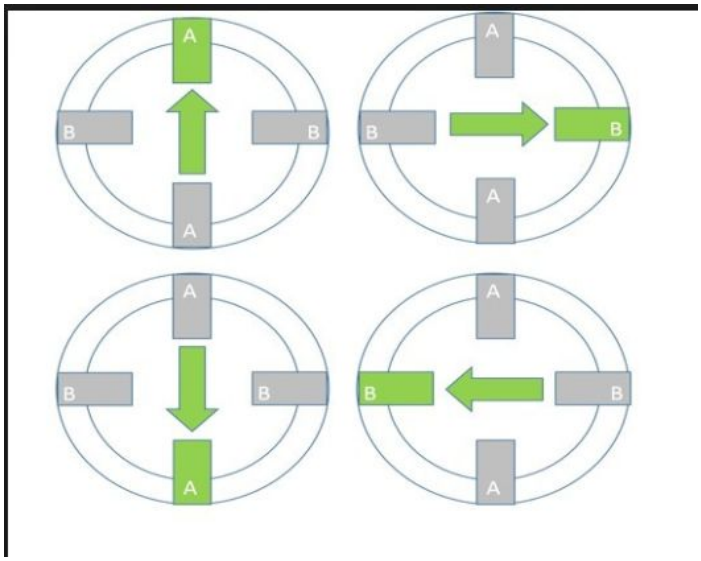

Figure 16 Billede af, hvordan wave-drive fungerer inde i motoren

Det ses, at rotoren drejes direkte hen ud for spolen i alle 4 tilfælde:
I koden er det gjort ved at bruge en switch, der tjekker en variabel state, der inkrementeres en gang i en periode eller dekrementeres en gang per periode for henholdsvis forward og backward direction:

```
if(mode==WAVE) {
switch(state) {
    case 0:
    {
        Pin_1A_Write(1);
Pin_2A_Write(0);
Pin_1B_Write(0);
Pin_2B_Write(0);
    }
    break;
    case 1:
    {
        Pin_2A_Write(1);
        Pin_1A_Write(0);
Pin_1B_Write(0);
Pin_2B_Write(0);
    }
    break;
    case 2:
    {
        Pin_1B_Write(1);
        Pin_1A_Write(0);
Pin_2A_Write(0);
Pin_2B_Write(0);
    }
    break;
```

Figure 17 Udsnit af koden, der implementerer wave-drive. Her ses de første tre states, motoren skal føres igennem

De rigtige pins for spolerne bliver aktiveret via PSoC'ens digitale pins.
Den anden mode er full-step. Den har også 4 steps, men her placerer rotoren sig mellem 2 af spolerne, som er vist på illustrationen fra rs-online.com:

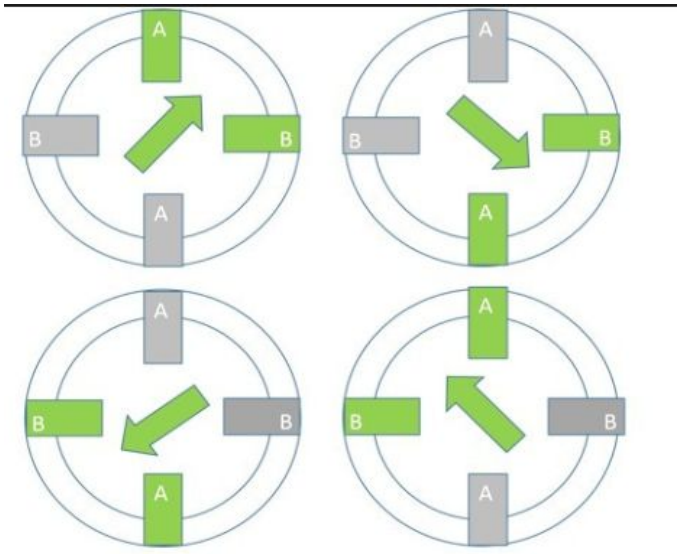

Figure 18 Billede af, hvordan full-step fungerer inde i motoren

Som det kan ses på illustrationen aktiveres 2 spoler ved siden af hinanden for at få rotoren til at stå imellem dem som følge af interferens af 2 elektromagnetiske felter. I koden er det implementeret således:

```
if(mode==FULL) {
    switch(state) {
    case 0:
    {
        Pin_1A_Write(1);
Pin_2A_Write(1);
Pin_1B_Write(0);
Pin_2B_Write(0);
    }
    break;
    case 1:
    {
        Pin_2A_Write(1);
        Pin_1A_Write(0);
Pin_1B_Write(1);
Pin_2B_Write(0);
    }
    break;
    case 2:
    {
        Pin_1B_Write(1);
        Pin_1A_Write(0);
Pin_2A_Write(0);
Pin_2B_Write(1);
    }
    break;
```

Figure 19 Udsnit af koden, der implementerer full-step. Her ses de første tre states, motoren skal føres igennem

Det minder om koden for wavedrive, men nu er der 2 digitale pins aktiveret ved hver case, fordi rotoren skal stå imellem 2 spoler.

Den sidste mode i dette eksperiment er half-step. Den har 8 steps til forskel for de 2 andre. Man kan sige, at denne mode er en kombination af de 2 forrige modes. Både de 4 stillinger for wave-drive og de 4 stillinger for full-drive er brugt i denne mode, således at rotoren drejer 45 grader pr. Omgang. Det skal ses i illustrationen nedenfor fra rs-online.com:

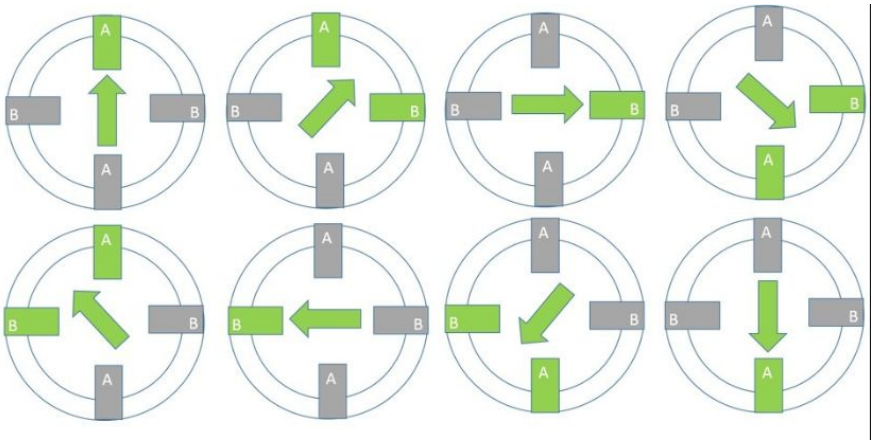

Figure 20 Billede af, hvordan half-step fungerer inde i motoren

Rotoren placerer sig altså både lige ude for en spole og imellem. Det giver som sagt 8 steps i alt. I koden er det implementeret med 8 cases fra 0 til 7. 3 af disse cases vises herunder:

```
if(mode==HALF) {
switch(state) {
  case 0:
    {
        Pin_1A_Write(1);
        Pin_2A_Write(0);
        Pin_1B_Write(0);
        Pin_2B_Write(0);
    }
    break;
  case 1:
    {
        Pin_1A_Write(1);
        Pin_2A_Write(1);
        Pin_1B_Write(0);
        Pin_2B_Write(0);
    }
  break;
  case 2:
    {
        Pin_1A_Write(0);
        Pin_2A_Write(1);
        Pin_1B_Write(0);
        Pin_2B_Write(0);
    }
    break;
```

Figure 21 Udsnit af koden, der implementerer half-step. Her ses de første tre ud af otte states, motoren skal føres igennem

Her er der altså nogle cases, hvor 2 digitale pins er høj, og andre cases hvor kun en digital pin er høj for de 4 spoler.

Stop motoren

Til at stoppe motoren er der lavet en global, statisk variabel ved navn turnOn. Den er fra start initialiseret til 1. Når stop funktionen bliver kaldt fra terminalen, så sættes turnOn til 0.

```
void stop()
{
    UART_1_PutString("Stop\r\n");
    turnOn=0;
}
```

Figure 22 Implementering af funktionen stop

Hele for-løkken er omkranset af en if, der tjekker turnOn.
Når turnOn bliver sat til en, bliver hele blokken i den if ikke udført, og motoren kører derfor ikke hen til næste step. Altså er motoren stoppet.

```
if(turnOn) {

switch (mode) {
case WAVE:
{

    switch(direction) {
     case FORWARD:
```

Figure 23 Udsnit af if-sætningen, der styrer, om motoren overhovedet skal dreje

Skift retning

Til at skifte retning for motoren er der implementeret 2 funktioner: driveForwards() og driveBackwards(). Her bliver direction sat til henholdsvis state FORWARD og state BACKWARD:

```
void driveForwards()
{
    UART_1_PutString("Set direction: forwards\r\n");
    direction = FORWARD;

}

void driveBackwards()
{
    UART_1_PutString("Set direction: backwards\r\n");
    direction = BACKWARD;
}
```

Figure 24 Kodeudsnit, der viser implementeringerne af funktionerne driveForwards og driveBackwards for stepper-motoren

I main er der sat en switch for hver mode, som tjekker hvilken direction programmet er sat til. Herunder ses en switch for moden WAVE:

```
switch(direction) {
 case FORWARD:
    {
        state++;
        if(state>=4) {
        state=0;
        }
    }
break;
 case BACKWARD:
    {
        state--;
        if(state<0) {
        state=3;
        }
    }
break;
```

Figure 25 Implementering af retningsstyring for stepper-motor

Som det ses tælles state en op ved forward og en ned ved backward. På den måde går motoren et step frem ved forward og et step bagud ved backward.

Forøg/sænk fart

Der er også en funktion for højere fart og en funktion for lavere fart, der hedder henholdsvis increaseSpeed() og decreaseSpeed(). Implementeringen af dem ser således ud:

```
void decreaseSpeed()
{
    UART_1_PutString("Increasing speed\r\n");
    if(speed<20) {
    speed+=1;
    }


}

void increaseSpeed()
{
    UART_1_PutString("Decreasing speed\r\n");
    if(speed>1) {
    speed-=1;
    }
}
```

Figure 26 Implementering af hastighedsstyring for stepper-motor

Her bliver speed henholdvis inkrementeret eller dekrementeret. speed er sat som et delay, der bruges i main i slutningen af for-loppen. Det er tiden mellem hvert step. Det ser således ud:

```
        CyDelay(speed);

        }
    }
}
```

Figure 27 Billede af CyDelay kaldet, der er måden, hvorpå duty-cycle bliver kontrolleret

Hvis der er et højt delay (når speed er stor), så er farten lille, og når delayet er lille (når speed er lille), så er farten stor.
Der er dog en begrænsning på, hvor hurtigt det kan gå. Det tager tid for rotoren i steppermotoren at dreje. Derfor må der ikke gå for kort tid mellem hvert step. Ellers vibrerer motoren sig bare meget uden at blive roteret.

Roter en omgang frem/tilbage

Der laves også en funktion, så der kan roteres en gang rundt. Både den ene vej og tilbage. Funktionen til dette hedder oneRotation(), og ser således ud:

```
void oneRotation()
{
    UART_1_PutString("Rotate one time\r\n");
    //counter sættes til 0 og flagene turnOn og rotate sættes
    turnOn=1;
    counter=0;
    rotate=1;
}
```

Figure 28 Billede af implementeringen af oneRotation. 2 flag bliver sat, og counter sættes til 0

Den sætter et flag for rotation sammen med flaget for turnOn for at sikre sig, at motorstyringen er aktivt lige nu. Herudover sættes counter til 0, så den kan tælles op. Counteren er sat som global variabel udenfor main. Counteren ses også i den uendelig for-løkke i main, hvor counter tælles op for hvert step, når rotate flaget er højt.
Der blev eksperimenteret med forskellige tal for, hvor mange steps en rotation er. Eftersom denne stepper-motor er 64 gearet, vidste vi, at der skulle man steps til. Der blev blot forsøgt med forskellige værdier, og 2000 viste sig at komme rigtig tæt på en rotation. Når counter er på 2000 sættes turnOn flaget til lavt, og motoren stopper efterfølgende. Det kan ses i følgende kodeudsnit:

```
for(;;)
{
    if(rotate) {
     counter++;
    //hvis der skal roteres, skal denne forløkke køres 2000 gange. Herefter stop
    if(counter>2000) {
    turnOn=0;
    }
    }
    if(turnOn) {
```

Figure 29 Billede af main, hvor rotate flaget tjekkes. I det tilfælde bliver counter talt op, og motorstyringen slukkes efter, den er nået 2000 via flaget turnOn

Hvis man vil have den til at rotere fremad, skal man kalde funktionen for “forward” i terminalen ved at trykke ‘1’. Hvis man vil have motoren til at rotere den anden vej, skal man kalde funktionen for “backward” ved at trykke ‘2’ i terminalen. Dette gøres inden oneRotation() er kaldt ved at trykke ‘r’ i terminalen.

Diskussion

Da der blev testet med at skifte mode for stepper motoren, så kunne der mærkes en lille vibration ved skift fra en mode til en anden. Det er formentlig fordi, at der skiftes til spole “1a”, når der sættes en mode. Det forstyrrer den nuværende rotation en anelse. Derudover kunne der mærkes en mindre vibration af motoren omkring rotoreren ved half-step end ved de 2 andre modes. Derudover var hastigheden en anelse hurtigere ved half-step. Det er formentlig, fordi half-step har 8 steps i stedet for 4. Ved 8 steps er afstanden fra rotoreren til den største magnetisk flux værdi kortere end ved 4 steps. Det gør, at den magnetiske kraft er større for roteren. Derfor bliver der ikke afsat så meget energi til det omkringliggende af motorens kerne, som bliver påvirket af det magnetiske felt. Netop fordi rotoreren bliver påvirket mest af det magnetiske felt.

Fordelen ved steppermotoren er, at den er bedre, hvis man skal have motoren til at ramme bestemte positioner. Det kan f.eks. være en robotarm, der skal løfte noget i en bestemt vinkel. Det kommer af, at vi selv vælger, hvor rotoreren i steppermotoren skal sættes ved at aktivere spoler i motoren.
Til gengæld gør dette, at der kan komme noget spildenergi, fordi denne operation forårsager vibrationer i motoren. Det sker ikke i samme grad ved DC-motoren.

Stepper motoren har til gengæld et meget simpelt design med 4 spoler, hvis man skulle lave en selv. Herudover ses, at steppermotorer i højere grad end DC motorer laves, så de kan trække en højere belastning.
Ulempen for steppermotorer er, at der skal bruges mange digitale pins, hvorimod DC-motoren kun kræver 1.
DC motoren har også flere rotationer pr. minut end ved steppermotoren.
DC-motoren er som følge af færre vibrationer mere energieffektivt end steppermotoren.
Duty cycle er nemmere at sætte ved DC-motoren, da man blot ændrer den gennemsnitlige spænding. Ved stepper-motoren ændrer man forsinkelsen mellem 2 steps.
Om man vil bruge den ene motor frem for den anden afhængig af, hvad den skal bruges til. Hvis man skal lave en stor robotarm, vil det være bedre med en steppermotor, der kan give bedre positionering, og det er nemmere at lave end med stor trækkraft end DC-motor.
Hvis man skal lave en drone, er det bedre at bruge en DC-motor, da den er mere energieffektiv, som er en fordel for batteriet på dronen. Herudover er der flere rotationer pr. Minut, som er vigtigt for at få opdrift på dronen.

Det ses også, at steppermotoren har en begrænsning på farten. Hvis det går for hurtigt, når rotoren ikke at dreje, inden der skiftes til et nyt step. Derfor kommer motoren ikke rigtig nogen vegne. På den anden side, kan det gå så langsomt med at skifte step, at man næsten ikke kan se rotoren bevæge sig, eftersom vores valgte steppermotor er 64 gearet. Det kunne også ses under testen. Da delayet mellem hvert step blev for lavt, så vibrerede motoren sig meget, uden der forekom nogen rotationer.

Konklusion

Der kan konkluderes, at man kan ændre hastigheden af en DC-motor i form af flere eller færre rotationer pr. tid med PWM. Det ses også en lille effekt ved at ændre frekvensen, selvom der bruges den samme duty cycle. Herudover kunne man bruge en H-bridge til at styre retningen af motorens rotationer. Til sidst kunne der ses, man kunne styre en steppermotor med 4 MOSFETS. Der kunne både ændre retning og hastighed. Der kunne også bruges forskellige drive modes til steppermotoren. Der ses, at der er en begrænsning for hastigheden for steppermotoren, da rotoren tager en anelse tid om at dreje.