# PAN/TILT System

Implementation in the PowerLeaf

To use this system in your project, it would require that a form of tracking was developed. This could be achieved through a sensor, though rather inefficient. The reason being, that a light sensor functions like a sensitive solar panel, meaning that the sun would be detected for a long time, without even being directly above the pan/tilt system.

Another and more practical solution, would be to just store data of the suns position in the system, and then adjust the angle of the pan/tilt system so that it is perpendicular with the sun's rays.

Below, I included the abstract and conclusion of our pan/tilt system project translated to English.

**Abstract**

The purpose of this project is to develop a user-friendly system for the regulation and control of a pan/tilt system. Programmable hardware has been designed and implemented for controlling and decoding of the system. The interaction between the user and the hardware is achieved using a microcontroller. Considerations for design, implementation, and user-friendliness have been made.

**Conclusion**

A control system for the pan/tilt system has been designed and then implemented. A pan/tilt system can be controlled by the means of a closed-loop controller, in the form of a proportional regulator with speed-control and position-control. Although simulation and testing do not match with each other, the controller functions as intended and provides a satisfactory result if there only the position control is observed.

A control-loop was designed and then implemented in C code on a microcontroller. In addition, a suitable method for scheduling and hard real-time operation has been chosen.

A SPI-communication have been established between the microcontroller and the FPGA, using microcontrollers built-in SPI peripheral, a proprietary SPI slave module, as well as a two-way communication support.

Control of the DC motors was achieved using a self-developed PWM module that interfaces with an H-bridge. In addition, a decoder module has been implemented, that allows for reading of the position, in conjunction with a closing loop regulation.

In addition, an intuitive user interface was developed to control the system.

All relevant aspects of the project, design, choices, simulations, tests, hardware and software are described and documented in this report.

I hope this can be of benefit to you in your project. If you have further questions, you are always welcome to contact me.

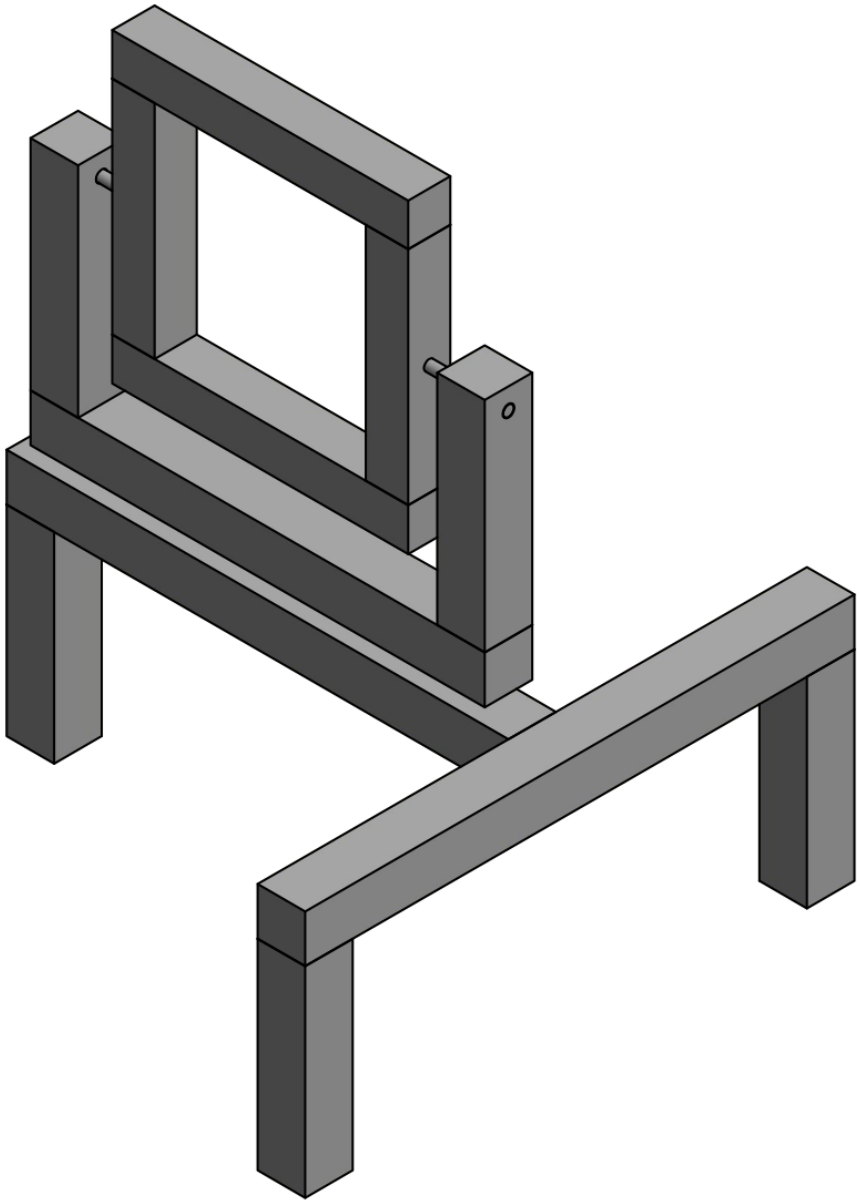
All the best,

Oliver Klinggaard
Stud.polyt
BSc in Engineering (Robot Systems)
University of Southern Denmark

Robotteknologi 2017 - Gruppe 5

Syddansk Universitet

# PAN/TILT System



Robotteknologi 2017 - Gruppe 5

Syddansk universitet

# Regulering og kontrol af robotsystemer


Anders Winter
Eksamens nr.: 426943
Mail: awint15@student.sdu.dk

Bjarke Ytting Hellden
Eksamens nr.: 84930262
Mail: bjhel15@student.sdu.dk

Christian Bach Andersen
Eksamens nr.: 425672
Mail: chan315@student.sdu.dk

Emil Seerup
Eksamens nr.:424342
Mail: emsee14@student.sdu.dk

John Tordur Kvlit Sevdal
Eksamens nr.: 425331
Mail: josev15@student.sdu.dk

Oliver Klinggaard
Eksamens nr.: 84929767
Mail: olkli15@student.sdu.dk

Rasmus Lundgaard Lange
Eksamens nr.: 426204
Mail: ralan15@student.sdu.dk


**Vejleder:**

Preben Holm

3. Marts 2017 - 29. Maj 2017

Forord

Denne rapport er skrevet af gruppe 5 på 4. semester Robotteknologi. Gruppen består af Anders Winter, Bjarke Ytting Hellden, Christian Bach Andersen, Emil Seerup, John Tordur Kvilt Sevdal, Rasmus Lundgaard Lange og Oliver Klinggaard. Formålet med dette projekt, er at udarbejde en regulering til et pan/tilt-system, ved at programmere en microcontroller, samt ved brug af en FPGA.

Der takkes vejleder Preben Holm.

**Resumé**

Formålet med dette projekt er udvikling af et brugervenligt system til regulering og kontrol af et pan/tilt-system. Der er udarbejdet en matematisk model for et pan/tilt-system. Der er designet og implementeret programmerbart hardware til styringen og dekodningen af pan/tilt-systemet. Til interaktion med brugeren og den programmerbare hardware er der udviklet et embeded software system til en microcontroller. Der er lavet overvejelser om kommunikation, design, implementering, og brugervenlighed.

# Indhold

# 1 Indledning

I den moderne verden er der et stadigt stigende behov, for at følge objekter eller mennesker, der er i bevægelse. Dette kan være i forbindelse med en produktion eller et overvågningssystem af en butik. Det grundliggende princip for sådanne systemer er, at de bliver styret af et pan/tilt-system.

# 2 Problemformulering

Projektes formål er at udarbejde en styring til et pan/tilt-system. Det skal styres via brugerinput, og der skal gives feedback i form af informationer fra systemet. For at kunne udarbejde en løsning til projektet, skal der anvendes kompetancer fra semestrets fag. Der skal programmeres en microcontroller til reguleringen, samt en FPGA til styring af motorer. Det er nødvendigt at modellere systemet matematisk for at kunne lave en regulering. I projektet vil der arbejdes på at svare på følgende formulerede spørgsmål:

- Hvordan kan et pan/tilt-system orienteringreguleres?
- Hvordan implementeres en regulerings sløjfe på en microcontroller?
- Hvordan kan der laves en SPI forbindelse mellem en microcontroller og en FPGA?
- Hvordan styres en DC-motor via en FPGA?

## 2.1 Kravspecifikation

Det prioriteres, at løsningen indeholder en systemanalyse og modellering af systemets enkelte elementer:

- Analyse og design af reguleringssløjferne i Matlab og Simulink.
- Dokumentation for FPGA design og implementering.
- Dokumentation for microcontrollerprogrammets design og implementering.
    - Herunder opdeling i Task's med lav kobling og valg af skedulering.
- Test og verifikation af systemet.

Der er følgende krav til systemet:

- Regulatorerne skal implementeres på én microcontroller.
- Der skal benyttes SPI kommunikationen imellem microcontroller og FPGA'en.
- FPGA'en skal styre PWM signalerne til motorerne.
- FPGA'en skal benyttes til at bestemme motorernes position via encoderværdierne.

### 2.2 Problemanalyse

Produktet skal indeholde en eller flere regulatorer, som skal styre pan/tilt-systemet. I reguleringen kan der indarbejdes forskellige input fra systemet samt fra brugere. Det er nødvendigt at modellere systemet matematisk for at designe en eller flere regulatorer. Der skal udvikles en kommunikationsprotokol mellem microcontroller og FPGA.

Hvilke krav er der til hastigheden på systemet, og hvilken indflydelse vil det have på kommunikationen mellem microcontroller og FPGA? Der skal undersøges, hvad der skal udarbejdes til FPGA'en for at kunne styre hastigheden på motorene via et PWM signal. Ligeledes skal der undersøges hvilke mulige reguleringer der er for at bestemme systemets position ud fra encoderværdier.

For at få et overblik over, hvordan systemet skal fungere og hvilken indflydelse de forskellige håndteringer af data har på systemet, skal der laves en analyse af systemet, samt et design af reguleringssløjferne i et Matlab og Simulink.

### 2.3 Projektafgrænsning

Der vil i dette projekt blive fokuseret på at lave en matematisk model af pan/tilt-systemet, og et softwaresystem der via FPGA'en kan regulere pan/tilt-systemet. Softwaresystemet skal tage brugerinputs og implementere regulatorene af pan/tilt-systemet. FPGA'en skal sende et PWM signal til motorene, og modtage data fra encoderne.

## 3 Læsevejledning

Denne læsevejledning gennemgår kort hvordan rapporten skal læses. Herefter vil der være en ordforklaring der udpensler relevante termer. I rapporten er der valgt at strukturere elementer, der er udviklet således:

- Teori
- Design
- Implementering
- Test

Ordforklaring kan ses i tabel 1.

| **Forkortelse** | **Fulde navn** |
|---|---|
| FPGA | Field Programmable Gate Array |
| microcontroller | En hel computer integreret i en chip. Et såkaldt SOC - System On a Chip. |
| PWM | Pulse Width Modulation |
| SPI | Serial Peripheral Interface |
| HDL | Hardware Definition Language |
| SCK | Serial Clock |
| MOSI | Master Out Slave In |
| MISO | Master in Slave Out |
| RTOS | Real Time Operating System |
| LCD | Liquid Crystal Display |
| Rise time | Den tid det tager at nå op til 90% |
| Overshoot | Den største positive procent afvigelse fra setpunktet |
| Steady state fejl | Forskellen mellem input og output for et system inden for et begrænset område |
| Settling time | Den tid det kræver for response kurven at forblive inden for 5% af målet |

Tabel 1: Ordforklring

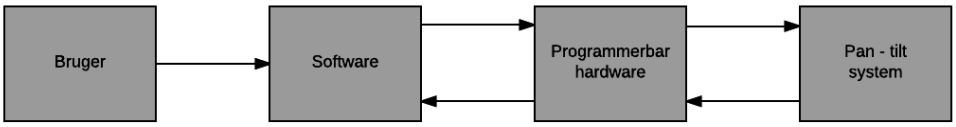
# 4 Løsningsforslag

For at løse problemstillingerne i problemformuleringen, skal der udvikles en applikation til pan/tilt-systemet, således at systemet kontrolleres ved brug af en regulering, hvor der anvendes en FPGA til styring af motoren, samt en SPI forbindelse mellem microcontrolleren og FPGA'en. Der er til problemstillingen forskellige aspekter der kan løses på flere måder:

- Regulering:
  - Reguleringen kan bestå af controllers fra den klassiske kontrol teori eller den moderne kontrol teori.
- Bruger input:
  - Bruger input der er overvejet til at betjene systemet, og user-interfacet er: Digi switch, Num-pad, knapper og computer gennem UART.
- Applikationer:
  - En applikation, hvor en bruger har mulighed for at styre systemet via et joystick eller lignende.
  - En applikation, hvor systemets placering bliver bestemt af værdier skrevet på num-pad'en, hvor reguleringen af både pan og tilt foregår synkront.

- En applikation, hvor systemet sættes i en bestemt position, hvor dataen omhandlende positionen gemmes. Dette kan gøres flere gange, hvor systemet efterfølgende kan flytte sig imellem disse positioner (inspireret af Universal Robots).
- En applikation hvor systemet følger en udefrakommende kilde, så som lys eller lyd.

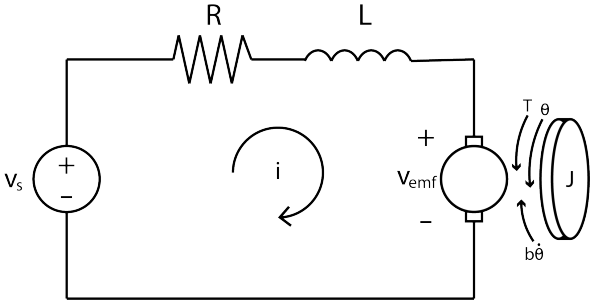
I dette projekt er det valgt at drage inspiration fra Universal Robots, og udvikle et system hvor brugeren fysisk kan ændre placeringen af pan/tilt-systemet og gemme denne position. Dertil skal det være muligt for brugeren at afspille disse positioner. Ydermere skal numpad'en tages i brug til at teste softwarens funktionalitet i de tidlige stadier af projektet. Flowet af information fra brugeren til pan/tilt-systemet kan ses i figur 1.




Figur 1: Figuren viser hvilke elementer data flowet fra brugeren, går igennem på vej til pan/tilt-systemet.

# 5 Modellering af system

I forbindelse med udvikling af regulerings kontrol af DC-motoren skal der udledes matematiske formler, som beskriver DC-motorens outputs, i forhold til input. Dette gælder for henholdsvis motorens position, hastighed og strøm i forhold til input spænding. Disse kan findes ud fra kredsløbssystemet set på figur 2. Figurens parametre er beskrevet i tabel 2.




Figur 2: Model af kredsløb for EMG30 DC-motor

| | |
|---|---|
| R | Modstand |
| L | Induktans |
| e | Den elektromotoriske kraft |
| T | Motor trækkraft |
| $\theta$ | Motor position |
| b | Friktion i motor |
| $\dot{\theta}$ | Vinkelhastighed |
| J | Inertimoment |
| i | Strøm |
| $V_s$ | Inputspænding |

Tabel 2: EMG30 DC-motor parametre

Ud fra dette system kan det ses at motoren har en såkaldt "back emf", hvilket er den elektromotoriske kraft. Denne kraft kan beskrives som:

$$V_{emf} = K_{emf}\dot{\theta} \tag{1}$$

Hvor $K_{emf}$ kan beskrives som den elektromotoriske kraftkonstant. Her kan Kirchoff's spændingslov benyttes til at finde formlen:

$$L\dot{i} + Ri = V_s - V_{emf} \tag{2}$$

Hvis det antages at magnetfeltet i motoren er konstant, bliver den trækkraft som motoren genererer proportional med strømmen som løber igennem motoren. Derfor kan trækkraften beskrives ud fra formlen:

$$T = K_t i \tag{3}$$

Hvor $K_t$ kan beskrives som motor trækkraft konstanten og $i$ er den strøm, som løber gennem motoren. Der kan ud fra Newtons 2. lov findes frem til formlen:

$$J\ddot{\theta} + b\dot{\theta} = T \tag{4}$$

Her er $K_t = K_{emf}$ og disse kan derfor begge beskrives med K, hvilket defineres som motor konstanten. Ligning 2 og ligning 4 Laplace-transformeres fra tidsdomænet til frekvensdomænet, så overføringsfunktioner for positionen, hastigheden og strømmen kan findes.

$$I(s)(Ls + R) = V_s(s) - K\Theta(s)s \tag{5}$$

$$J\Theta(s)s^2 + b\Theta(s)s = KI(s) \tag{6}$$

Når overføringsfunktionen for positionen skal findes, skal der isoleres for I(s) i ligning 6, hvilket giver:

$$I(s) = \frac{s(Js + b)}{K}\Theta(s) \tag{7}$$

Ligning 7 skal herefter indsættes i ligning 5 hvilket giver:

$$\Theta(s)\frac{s(Js + b)}{K}(Ls + R) = V_s(s) - K\Theta(s)s \tag{8}$$

Dette kan omregnes til overføringsfunktionen for positionen i forhold til spændingen:

$$P_{position}(s) = \frac{\Theta(s)}{V_s(s)} = \frac{K}{s((Js+b)(Ls+R)+K)} \quad [\frac{rad}{V}] \tag{9}$$

Hastigheden i forhold til spændingens overføringsfunktion, kan findes ved at differentierer ligning 9, hvilket svare til at gange med s på højre side af lighedstegnet:

$$P_{Hastighed}(s) = \frac{\dot{\Theta}(s)}{V_s(s)} = \frac{K}{((Js+b)(Ls+R)+K)} \quad [\frac{rad/s}{V}] \tag{10}$$

Strømmen i forhold til spændingen, kan findes med samme fremgangsmåde. Der skal her isoleres for $\Theta$(s) i ligning 6:

$$\Theta(s) = \frac{KI(s)}{(Js^2+bs)} \tag{11}$$

Ligning 11 indsættes nu i ligning 5 og der isoleres for $\frac{I(s)}{V_s(s)}$. Overføringsfunktionen for strømmen i forhold til spændingen er givet ved:

$$P_{Strøm}(s) = \frac{I(s)}{V_s(s)} = \frac{1}{(Ls+R)+\frac{K^2}{Js^2+bs}} \tag{12}$$

## 5.1 Parametre til modelering

I forbindelse med den matematiske modellering af pan/tilt-systemets, er der opgivet parametre i [1]. Udover disse parametre skal modstanden, induktansen, friktionskoefficienten og inertimomentet for motoren er fundet. Disse kredsløbsparametre kan ses i tabel 3

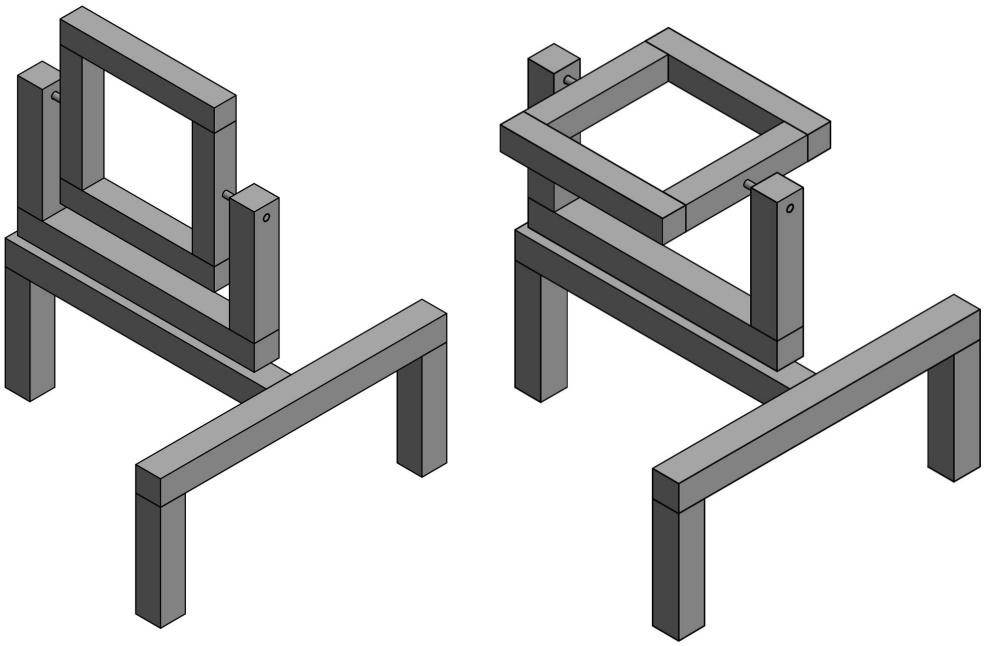
| | |
|---|---|
| Modstand | 6.8144 Ω |
| Induktans | 3.3475 mH |
| Motor friktionskoefficient | 0.000931 |
| Motorakslens intertimoment | 0.0019 $kg \cdot m^2$ |

Tabel 3: Model af motoren

Modstanden og induktansen er målt med et LCR-meter og motorens friktionskoefficient og rotorens inertimoment er fundet i en artikel [3]. En anden parameter er konstanten for den elektromotoriske kraft og motorens trækkraft. Denne motor konstant kan findes, ved at konverterer motorens hastighed uden belastning, til radianer pr. sekund. Dette giver 22,61946708 rad/s. For at finde motorkonstanten, skal den maksimale spænding divideres med denne hastighed.

$$K = \frac{12V}{22.61946708\frac{rad}{s}} = 0.530516 \tag{13}$$

Når der er tale om pan/tilt-systemets inertimoment kan denne variere, alt efter hvilken position systemet befinder sig i. Systemet ses i figur 3.



Figur 3: Venstre "best-case"og højre "worst-case"i forhold til inertimoment

Systemets inertimomenter er beregnet. Disse beregningerne kan ses i bilag **A.2**, og resultaterne er opsummeret i tabel 4.

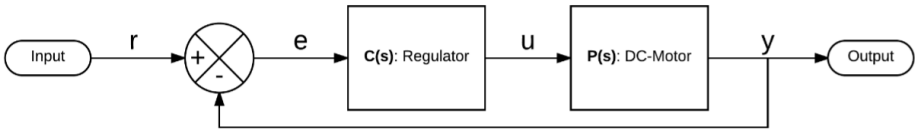
| | |
|---|---|
| Tilt-ramme | $0,0263kg \cdot m^2$ |
| Pan-ramme best case | $0,0889kg \cdot m^2$ |
| Pan-ramme worst case | $0,0946kg \cdot m^2$ |

Tabel 4: Model af motoren

De fundne parametre kan variere fra motor til motor og pan/tilt-system til pan/tilt-system da de systemer, som er stillet til rådighed, ikke alle er opbygget nøjagtigt på samme måde.

# 6 Design af regulering

Modellernes overføringsfunktioner er fundet, og kan ses i afsnit 5, og der kan ved hjælp af disse designes motorens regulatorer, som skal sørge for systemet er stabilt. Der findes flere forskellige former for regulatorer, men der vil i dette projekt tages udgangspunkt i PID regulering, som er en del af den klassiske kontrol teori. Dette er valgt da systemet er defineret som et lineært system og derfor er det oplagt at bruge klassisk kontrol teori. Et andet argument er at PID regulering stadig den dag i dag bruges meget i industrien.




Figur 4: Simpel regulator

Som det ses på figur 4 skal et system opsættes hvor motoren indgår i et feedback loop. I dette loop, skal der være en eller flere regulatorer, som skal regulerer for henholdsvis motorens position, hastighed og strøm. Det kan ses at **e** repræsenterer en fejl, som består af forskellen mellem inputtet **r** og outputtet **y**. Dette fejlsignal bliver sendt til PID regulatoren, som herefter beregner dennes integrale og differentiale. Mellem **C(s)** og **P(s)** ses kontrol signalet **u**, dette signal betegnes som:

$$u = K_p + \frac{K_i}{s} + K_d s \tag{14}$$

Signalet sendes herefter til motoren og et nyt output er nu beregnet, som igen sendes tilbage til regulatoren. Systemet bliver ved med at regulere uanset om fejlen er lig nul [11],[12].

## 6.1 PID regulator

PID står for Proportional, Integral og Derivative. Det ses i ligning 14 at de hver har en konstant, $K_p$, $K_i$ og $K_d$. Disse konstanter skal bruges til stabilisere systemet. Hvordan disse stabiliserer systemet, kan ses i tabel 5.

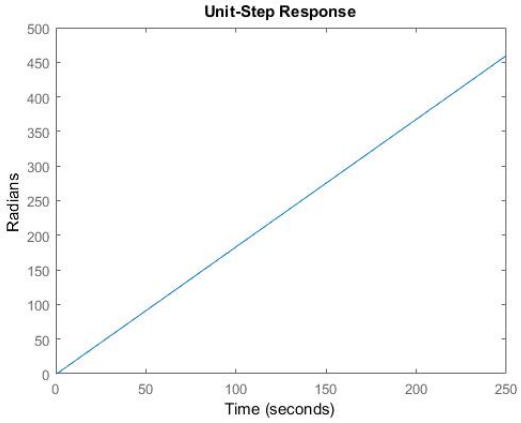
| **PID konstanter** | **Rise time** | **Overshoot** | **Settling time** | **Steady state Fejl** |
|---|---|---|---|---|
| Kp | Formindsker | Øger | Lille ændring | Formindsker |
| Ki | Formindsker | Øger | Øger | Eliminerer |
| Kd | Lille ændring | Formindsker | Formindsker | Ingen ændring |

Tabel 5: PID regulatorens forskellige leds effekt

Tages der udgangspunkt i overføringsfunktionen for motorens position, kan det ses ud fra et step respons på figur 5, at motoren uden en regulator vil fortsætte med at lede efter en position.

Der kan vha. Matlab designes en PID regulator, som stabiliserer systemet, så motoren vil nå en bestemt position. Det gælder her om at designe en regulator, som vil formindske **Rise time**, og så vidt muligt eliminere **Overshoot** og **Steady state fejl** og til sidst få sat **Settling time**, alt efter hvor hurtigt motoren skal finde den nye position. Inden konstanterne bestemmes er det som udgangspunkt en god ide at give konstanterne følgende værdier, som der kan ses i tabel 6:

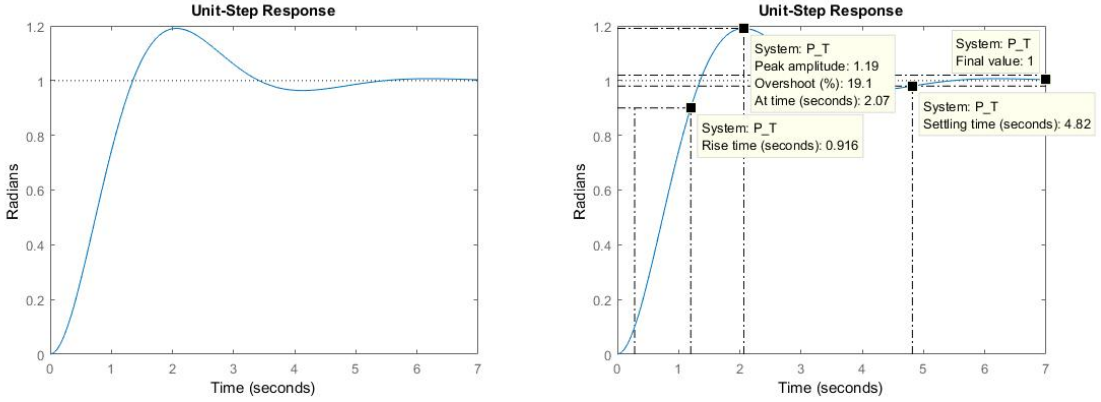
Dette giver et godt udgangspunkt, der herefter kan fin-tunes. Det kan ses på figur 6a og figur 6b, hvordan motoren vil opfører sig med en PID regulator, med forstærkningerne fra tabel 6.




Figur 5: Step response

| **Kp** | **Ki** | **Kd** |
|---|---|---|
| 1 | 0 | 0 |

Tabel 6: Startværdier for PID



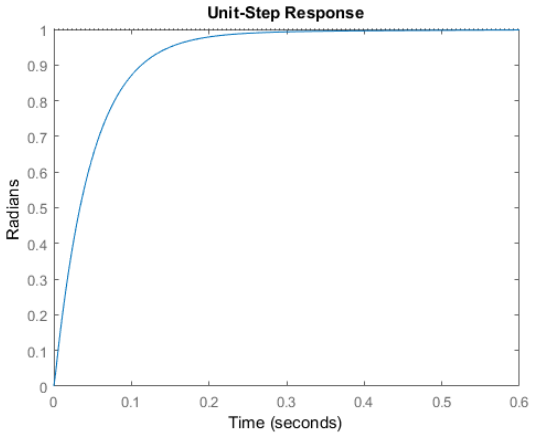

(a) Stepresponse uden markeringer

(b) Stepresponse med markeringer

Figur 6: Simulering af unit step response

Det kan ses at systemet vil tage omkring 7 sekunder at nå til den endelige position. Efter 2.07 sekunder sker der et overshoot på 19.1% hvilket betyder at systemet vil gå over den ønskede position, for derefter at rykke tilbage til den ønskede position. Dette skal undgås når der er tale om positions regulering, fordi det i praksis vil få pan/tilt-systemet til at svinge. Efter en tuning af PID konstanterne, hvilket gøres ved trial and error i Matlab's Simulink værktøj, er der blevet simuleret et passende step respons for motorens position, hvilket kan ses på figur 7.

PID konstanterne kan ses i tabel 7.




Figur 7: Stepresponse efter PID regulering

| Kp | Ki | Kd |
|---|---|---|
| 10 | 1 | 7 |

Tabel 7: PID værdier ud fra simulering

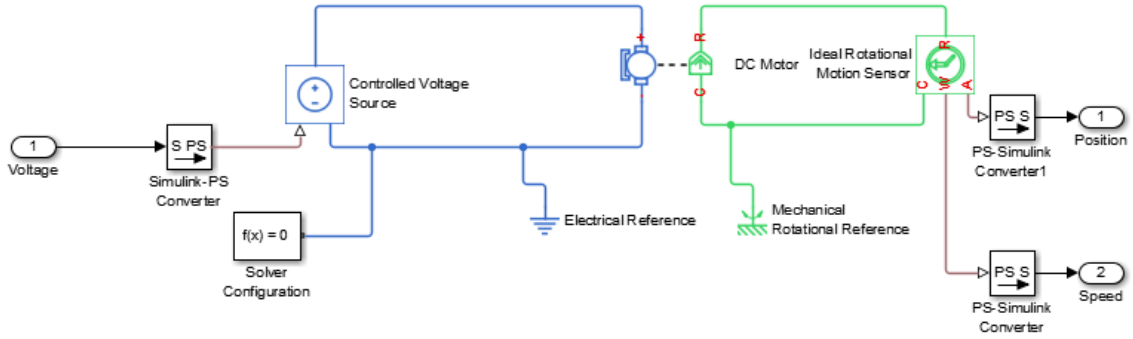
Der er her lavet en fuld PID regulator. Det er dog muligt at lave en regulator, som kun består af P, I, PI, PD eller ID konstanter. Disse skal vælges alt efter hvad der skal kontrolleres. I dette projekt undersøges det om det er muligt at implementere en regulering bestående af en P regulator til at regulere positionen, og en P regulator til at regulere hastigheden.

## 6.2 Implementering

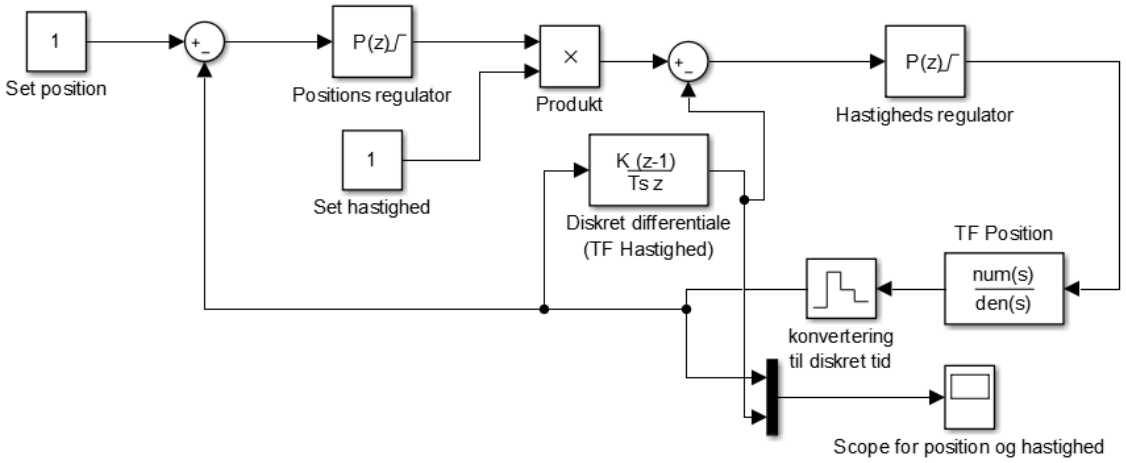
### 6.2.1 Simulering

For at simulere DC-motorens opførsel er der i Matlab og Simulink opstillet overføringsfunktioner for positionen og hastigheden. I Matlab kan disse simuleres uafhængigt af hinanden og i Simulink kan det simuleres, hvordan de arbejder afhængigt af hinanden. De endelige PID konstanter vil blive fundet ud fra simuleringer i Simulink. Der er vha. Simscape biblioteket i Simulink designet en kopi af motorsystemet, som indeholder de angivne parametre fra [1], som ses på figur 8. Denne skal bruges til at sammenligne det outputsignal, der bliver modtaget fra overføringsfunktionerne. Den endelige reguleringssløjfe, vil blive lavet ud fra en betragtning af virkeligheden og kan ses på figur 9.

I denne reguleringssløjfe sendes to set-points, i form af en position og en hastighed. Der laves et produkt af positions regulatorens output og hastighedens setpoint. Signalet bliver sendt videre til hastigheds regulatoren, hvorefter den sendes til motoren, i form af positionens overføringsfunktion. Signalet fra overføringsfunktionen, bliver efterfølgende konverteret til diskret tid, og gives som et feedback tilbage de to regulatorer.

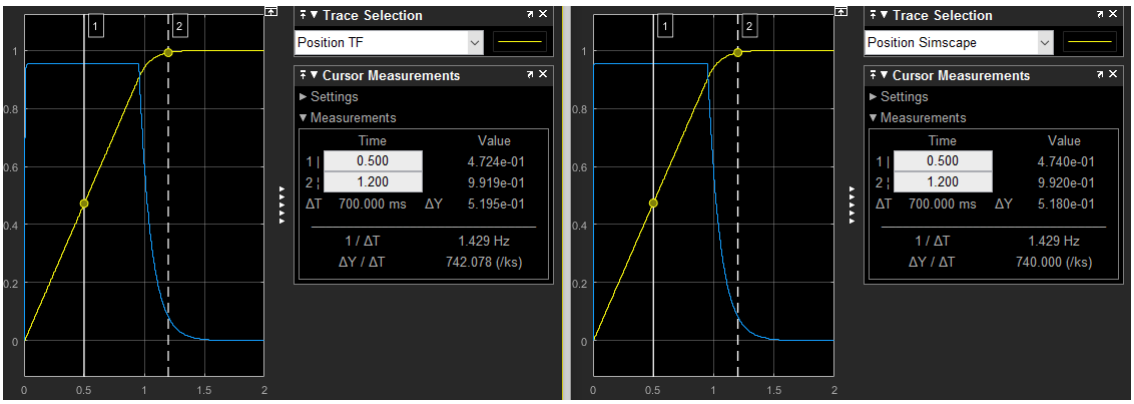



Figur 8: EMG30 DC-motor designet i Simscape




Figur 9: Endelig reguleringssløjfe designet i Simulink

Output signalerne fra Simscape motoren og reguleringssløjfen kan ses på figur 10.




Figur 10: Venstre: Output fra overføringsfunktionerne, Højre: Output fra Simscape motoren,

Det gule signal er positionen og det blå er hastigheden. Det ses at hastigheden stiger med det samme og kører med konstant hastighed, indtil positionen er ved at nå sin destination, hvorefter den falder proportionalt i takt med at systemet nærmer sig sin destination. Positions signalet

har en S-kurve, som ikke har noget overshoot. Dette design repræsenterer den reguleringssløjfe som skal implementeres i softwaren. Der er taget målinger efter 0.5 og 1.2 sekunder på begge figurer, hvor det ses at positionens værdier har en fejlprocent på 0.0007% efter 0.5 sekunder og 0.0004% efter 1.2 sekunder. Dette giver et godt udgangspunkt, til at lave en reguleringssløjfe ud fra overføringsfunktionerne.

### 6.2.2 Implementering i software

For at lave en implementering af reguleringssløjfen, skal der tages højde for at den matematiske model, som danner basis for at reguleringen, har radianer som positionsenheder. Da diskretiseringen af encoderen foretages til heltal med værdier fra 0 til 1079, kan der med fordel bruges en konstant som konverterer dette til radianer. Dette beregnes i ligning 15.

$$\frac{2\pi}{1080} = 0,005818 \tag{15}$$

Der er valgt at bruge PWM til spændingsregulering, derfor kræves det at der kan specificeres en duty cycle som er proportional med spænding. Denne kan angives med værdier fra 0 til 255. Dette beregnes i ligning 16.

$$\frac{255}{12V} = 21,25 \tag{16}$$

Da regulatordesignet specificerer en hastigheds regulering, er det nødvendig at danne et hastighedssignal. Da systemet ikke indeholder tachometer (omdrejningsmåler) beregnes encoderens frekvens ud fra encoderens position over tid. Dette er ikke en triviel sag og derfor er der undersøgt to forskellige metoder:

1. Kode køres ved faste tidsintervaller hvor $\Delta X$ måles.
2. Kode køres kun når X ændrer sig, derved opnås et variabelt tidsinterval.

Metode 1 virker bedst ved moderate-høje hastigheder, hvor der forekommer større ændringer i position end kodens frekvens. Metode 2 virker bedst ved lave hastigheder, hvor tidsintervallet i stedet noteres imellem hver positionsændring. Valget i projektet faldt på at bruge metode 2, da det eksperimentelt blev fundet at systemets maksimale rotationer ikke kan oversstige 1 Hz. Dette betyder, grundet den inkrementerende encoder, at positionen kun når at ændre sig 1080 gange pr. sekund. Ved lave hastigheder vil dette resultere i en meget forsinket respons fra det genererede frekvenssignal [4].

### 6.2.3 Delkonklusion af regulering

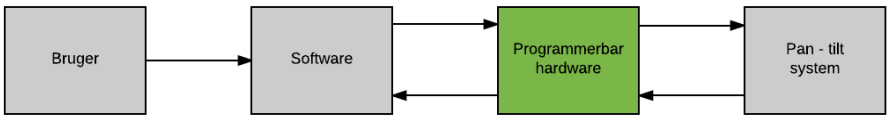
Matematiske modeller for DC-motoren er blevet udledt og testet i simulationer, hvor resultaterne er fundet tilfredsstillende.

# 7 Programmerbar hardware

Et af kravene til projektet er at services til kontrollering af pan/tilt-systemet, skal foregå igennem en FPGA, og der skal de anvendte peripherals implementers. Dette delsystem skal så kommunikere med en microcontroller, der agerer som master, igennem SPI-kommunkation. I dette projekt er der valgt at implementere følgende moduler med programmerbar hardware. Hver funktion er tilknyttet sit eget modul, som er implementeret i VHDL. Hvert modul forbindes ligeledes internt i FPGA'en:

- SPI.
- PWM-generering.
- Quadrature encoder til absolut decodermodul.

Den programmerbare hardware binder microcontrolleren sammen med pan/tilt-systemet. Se figur 11.




Figur 11: FPGA'ens placering i flowet af information fra brugeren til pan/tilt-systemet.

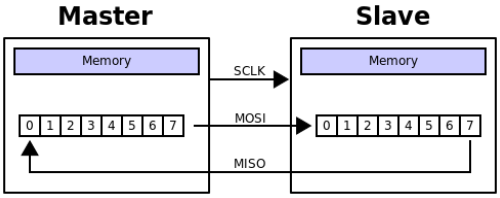
## 7.1 SPI

### 7.1.1 Protokol

I dette projekt er det et krav at der anvendes SPI. Der bruges ofte SPI til hurtig kommunikation med perifere enheder, som eksemplvis EEPROM, ADC, og andre systemer. SPI er en kommunikationsprotokol der har den fordel, at den kun skal bruge fire forbindelser, i stedet for en parallel bus, hvor der i dette tilfælde skulle bruge op til 16 signal linjer. Denne protokol består af fire forbindelser:

- SCK – Seriel Clock
- MOSI – Master Out Slave In.
- MISO – Master In Slave Out.
- $\overline{SS}$ – Slave Select

Et master-slave kommunikations system sættes op, hvor der er to enheder der forbindes sammen med to skifteregistre. Kommunikationen foregår ved at begge skifte registre clockes med samme clock signal, kendt som SCK. Således skiftes data rundt, fra register til register. Når antallet af clock perioder er blevet lig med registerlængden, er alt data blevet overført imellem de to enheder, betragt figur 12. I SPI styres denne transaktion af $\overline{SS}$ signalet. Når signalet bliver aktivt, informeres enheder i netværket om at kommunikation skal påbegyndes.




Figur 12: Master slave skifteregister

### 7.1.2 Valg af design

Seriel kommunkation via SPI kan løses på flere måder. I listen nedenunder ses løsningsforslag:

1. En SPI pr. modul: Hver enhed er sluttet til sit eget SPI modul.
   - \+ Simpel
   - \+ Hurtig
   - \+ Lav kompleksitet på både softwareside samt VHDL
   - − Kræver at microcontroller understøtter 16 signaler i alt
   - − Flere ledninger
   - − Øget hardwarekompleksitet og pris
   - − Ikke muligt med det anvendte hardware
2. Instruktion – datainterface: En besked sendes til enhed via SPI, som instruerer SPI-slaven om hvad den skal gøre med data.
   - \+ Intuitivt
   - \+ Simpelt at interface med microcontroller
   - \+ kræver kun en kanal
   - − Høj kompleksitet i VHDL
   - − Høj kobling af SPI modul med andre enheder i FPGA'en
3. Adresse-data interface: En besked sendes til en read eller write adresse hvorefter data skrives til FPGA'en eller data læses ud af FPGA'en. Dette gør det muligt at have to interne databusser, én til udgående data og én til indkommende data.
   - \+ Simpelt
   - \+ Nemt at implementere med VHDL
   - \+ Simpelt at interface med microcontroller
   - \+ Kræver ikke avanceret dekodning af instruktion

4. Multi slave bus: Flere slave select signaler føres til FPGA'en, men enheder deler SCK, MOSI og MISO.

   + Simpelt at implementere på FPGA'en

   – Ikke alle microcontrollere understøtter dette som standard

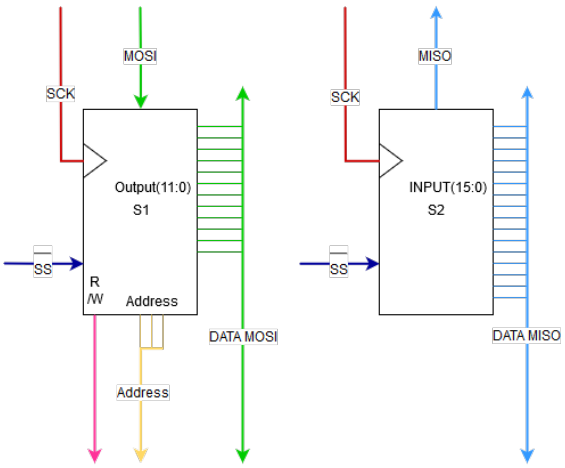
   – Øget hardwarekompleksitet ( Flere ledninger)

Løsning tre blev vurderet som den bedste løsning. Dette skyldes især ønsket om reduceret hardwarekompleksitet, og simpelt samspil med software. Der blev til formålet designet en simpel protokol, ovenpå SPI som tilbød en intuitiv metode for adressering, der blev hertil designet et 16-bit frame format, med ét bit der specificerer om det er en read- eller writeoperation. De næste tre bit er adresse bit, dette giver ni adresser. Derefter er der 12 data bit. Denne frame størrelse blev valgt således at der var muligt lave en write operation med én transaktion, og en read operation med to transaktioner. Systemets encodere har en opløsning på 1080 encoderticks på en hel systemomdrejning. Dette resulterer i at der minimum skal være 11 data bit til rådighed og derfor blev der valgt 12 bit data. Dette kan ses i tabel 8.

| **R / MSB** | **Addresse** | **Data** |
|---|---|---|
| MSB 15 | 14:12 | 11:0 |

Tabel 8: SPI modul og dens forbindelser

For at gøre systemet hurtigt er der simultan ud-/indlæsning af data. Derfor er der valgt at anvende to skifte registre. Indgående data skrives i register 1 (s1), og udgående data skrives ud af register 2 (s2). Dette gør det muligt at skrive data ud på to separate parallelle databusser, hvilket simplificerer interfacet imellem SPI modulet og andre moduler internt i FPGA'en. Dette kan ses i figur 13.




Figur 13: SPI adressering

Derudover er der designet funktionalitet således at et signal aktiverer systemet.

### 7.1.3 Implementering

SPI modulet er implementeret i VHDL. Denne implementering følger designet som beskrevet i sektion 7. Der er dog nogle implementations specifikke detaljer, der er værd at nævne:

#### 7.1.3.1 Clock Buffering

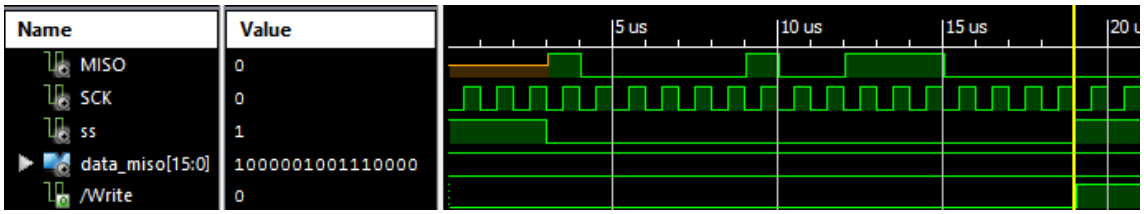
Ved brug af SPI til kommunikation er det den Serielle Clock (SCK), der sætter takten for systemet. SCK-signalet skal derfor synkroniseres med FPGA'ens egen clock. Dette gøres for at bringe SCK-signalet ind i FPGA'ens tidsdomæne, for derfor at undgå asynkron logik, hvilket ville besværliggøre implementeringen væsentligt. I kodeeksemplet i figur 14 vises det på linje to, samt tre, hvordan et signal kan bringes ind i FPGA'ens tidsdomæne. Eksemplet er et udsnit der både viser clock bufferen, og hvordan clock bufferen kan udnyttes, således der kan laves "event" baseret logik.

```
                if rising_edge(Clock) then
                ---------------------------
                SCLK_EDGE <= SCLK_EDGE(1 downto 0) & Shift_CLK;
                SS_EDGE <= SS_EDGE(1 downto 0) & SS;
                    ------------------------------------------
--                  Area of code that reacts to the SS Event.
                    ------------------------------------------
                    --- Falling Edge
                        if SS_EDGE = "110" then -- First falling edge SS EVENT
                            xWrite <= '0';
                            xRead <=  '0';
                            shift_register_MISO(15 downto 0) <= Data_MISO;
                        end if;
                end if;
```

Figur 14: VHDL kode til clock buffering

#### 7.1.3.2 Simulering/test

Systemet blev simuleret som vist i figur 15. Simuleringer viste at alt funktionalitet virkede efter hensigten. I simuleringen vises det at der laves en write operation. Signalet data_miso viser indholdet af MISO registret efter en succesfuld transaktion, med data sendt serielt på MISO porten.



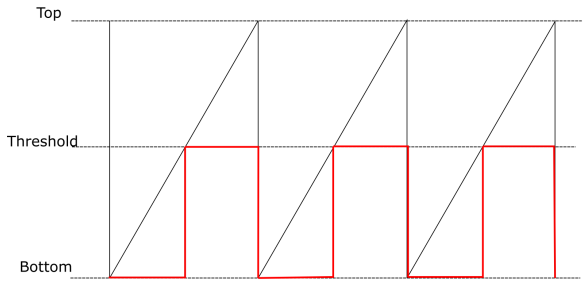

Figur 15: simulering af write operation

## 7.2 PWM

Et krav til projektet er at der til pan/tilt-opstillingen bliver drevet to motorer, én til hver akse. For at tilbyde funktionalitet til at drive motorene begge veje, er det derfor nødvendigt at bruge en H-bro. Til denne H-bro skal der skrives et hardware modul, som tilbyder PWM, samt retning og fra- og tilkobling.

### 7.2.1 Teori

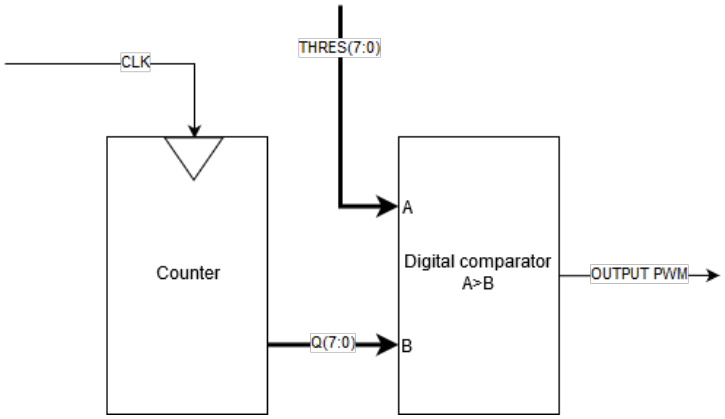
PWM står for Pulse-width modulation. Dette betyder at der genereres et periodisk digitalt signal, der tændes i en procentdel af periode tiden. Denne procentdel, kaldes duty cycle. Ved duty cycle på 100% er signalet altid højt, og ved 0% er signalet altid lavt. Et eksempel på 50% duty cycle kan ses på figur 16.




Figur 16: Visualisering af PWM

### 7.2.2 Design

Ved nogle applikationer er det relevant at specificere yderligere krav til PWM-signalet såsom fase-korrekthed, eller anvende Delta-Sigma modulation. Da applikationen i dette tilfælde ikke stiller nogle krav til typen af PWM, er der valgt PWM af typen ”fast PWM”. Denne type PWM fungerer ved at en tæller når en thresholdværdi. Her sættes et output til et være højt eller lavt alt efter polaritet. Imens tælles der stadig op indtil tælleren når sin maksimale værdi, hvorefter den overflower, og bliver nul igen. Her sættes et bit lavt eller højt alt efter polaritet. Hardwaremæssigt er det designet ved brug af et tælleregister, som tælles op ved hvert clock signal. Tællerens 8-bit output indsættes i et digitalt komparatorkredsløb med funktionen A>B, hvor A = tæller, B = threshold. Output fra den digitale komparatorfunktion er PWM-signalets output. Dette kan ses i figur 17.




Figur 17: PWM styring

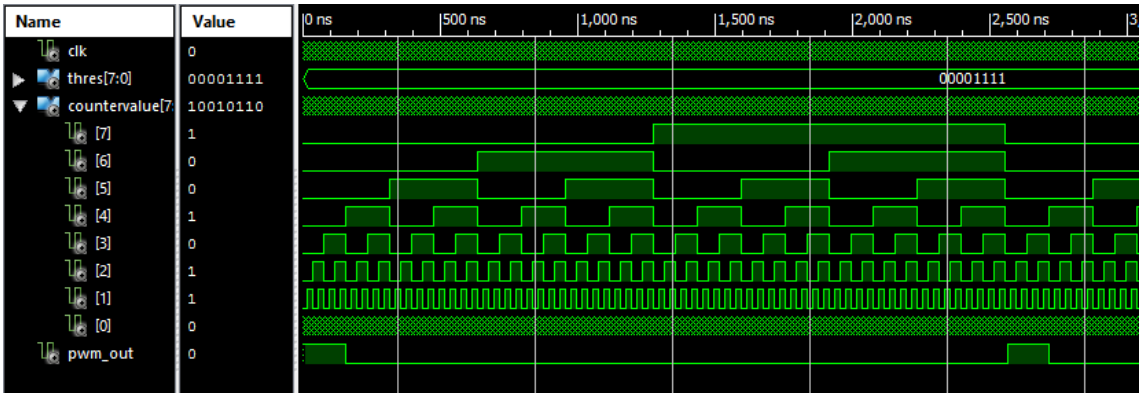
### 7.2.3 Implementering

På figur 18 ses et eksempel fra projektet kildekode.

```
begin
    PWM_OUT <= pulse;
    counterValue <= counter;
    process(Clk)
    Begin
        if rising_edge(Clk) then
            if counter = Thres then
                pulse <= '0';
            elsif counter = "00000000" then
                pulse <= '1';
            end if;
            counter <= counter + 1;
        end if;
end process;
```

Figur 18: Implementering af PWM signal

Simuleringen verificere at modulet virker korrekt, hvilket kan ses i figur 19.




Figur 19: Simulering af PWM signal

## 7.3 Dekodning af encoder

I dette projekt er der stillet det krav at FPGA'en skal omdanne et inkrementerende signal til et absoulut signal. Dette inkrementerende signal kommer fra en quadrature encoder, som er placeret

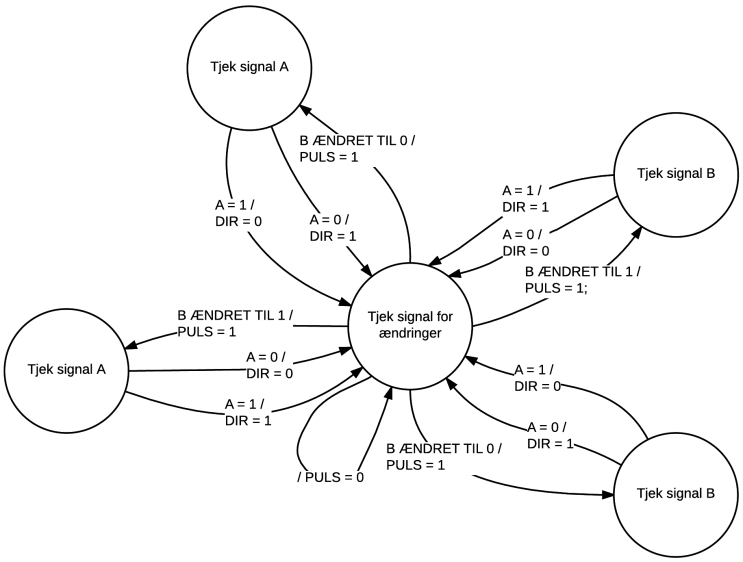
inde i motorens karrosseri.

### 7.3.1 Quadrature encoder

En roterende inkrementerende encoder er en encoder, som ofte bruges i forbindelse med måling af roterende bevægelse. Encoderene der er brugt i forbindelse med projektet, bruger to hall sensorer, som sammen med en magnetiseret skive generer to signaler der er 90 eller -90 grader faseforskudt.

### 7.3.2 Design af dekodermodul

Første del af dekoderen er lavet til at kunne genkende ændringer i begge encodersignaler og derefter genere en puls, som beskriver hvornår ændringerne i encodersignalet forekommer. Derudover skal der bestemmes, hvilken retning motoren roterer, ud fra hvordan signalerne ændrer sig i forhold til hinanden. Statediagremmet for dette kan ses på figur 20.

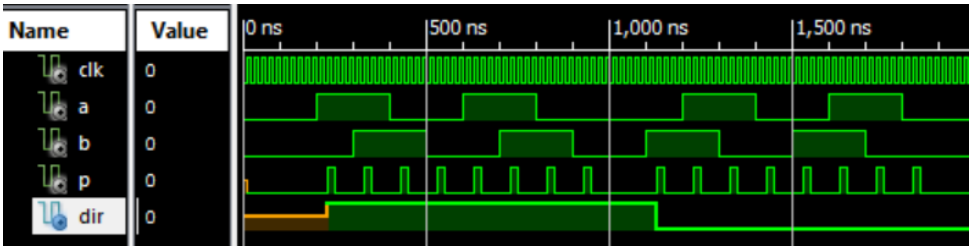



Figur 20: State diagram for en dekoder til quadrature encoder

### 7.3.3 Implementering af dekodermodul

Signalerne fra en encoder deles i et a og et b input. Disse input oversamples i hver sin vektor med to pladser. Oversamplingen foregår på samme måde som clock bufferen i sektion 7.1.3.1. Nu kontrolleres der, hvorvidt et af signalerne har ændret sig. Hvis et signal ændrer sig, sendes en puls ud fra modulet. Efterfølgende kontrolleres det modsvarende signals værdi, og retningen sættes.

Simuleringen af dette gengiver, hvad der sendes fra dette modul til det næste, og kan ses på figur 21.

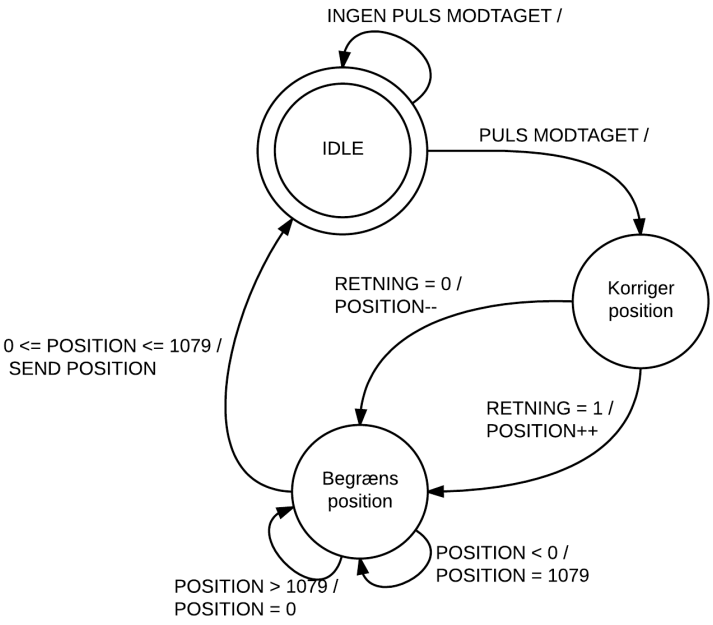



Figur 21: Output fra decoderen

Disse outputs bliver herefter videregivet til et andet modul, som tager sig af at udregne hvilken position motoren befinder sig i.

### 7.3.4 Design af positionsmodul

Dette modul har til opgave at generere noget konkret data omhandlende positionen, som en anden enhed kan bruge til styring af systemet. Positionsmodulet skal virke ved at analysere de to input, der kommer fra dekodermodulet. Ud fra disse input ændres der på en positionsværdi. Denne værdi skal bruges til outputtet fra modulet. Desuden skal dette modul tage højde for, hvorvidt pan/tilt-systemet er i en position, der kan kaldes for et nulpunkt. Dette skal gøres ved konstant opsamling af data fra en hallsensor på systemet. Denne data sendes sammen med positionsværdien til outputtet. Statediagrammet for dette kan ses på figur 22.



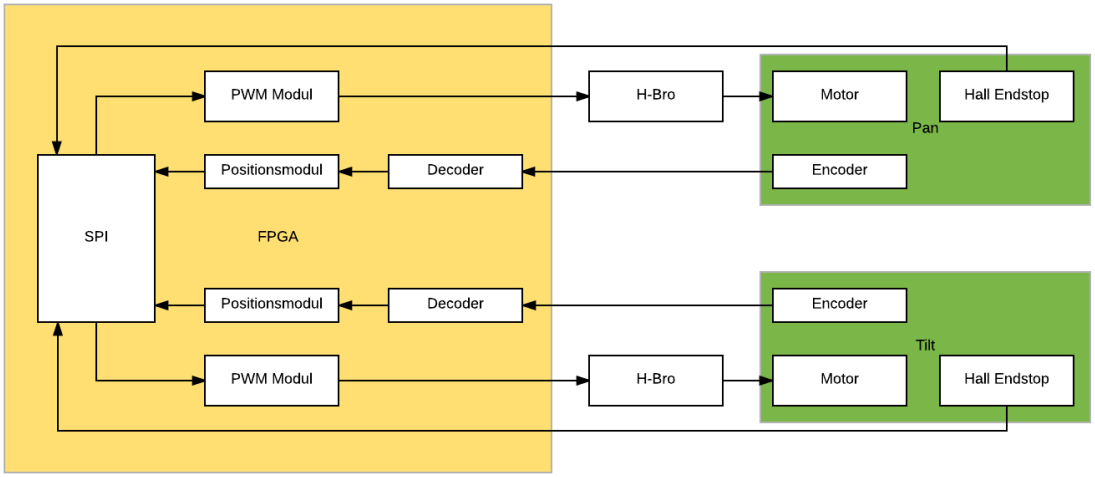

Figur 22: State diagram for positionsdekoderen

### 7.3.5 Implementering af positionsdekoder

Modulet er implementeret med tre indgangssignaler:

- Puls: Det indkommende signal, der indikerer en ændring i position.
- Direction: Signal der indikerer retning på puls signal. Ved '0' tælles der ned, ved '1' tælles puls signalet op.
- Hall Input: Signal fra hall elementer monteret på opstilling til homing/endstop.

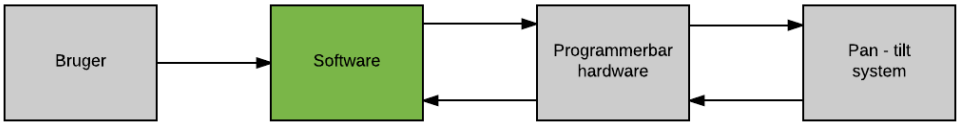
Da en omgang på systemet maksimalt kan være 1079, er der implementeret et overflow således at det implementerede tælleregister bliver sat til '0' ved værdier $> 1079$, og ved værdier $< 0$ sættes tælleren til 1079. Det signal der modtages fra end-stops, sendes som det mest betydende bit tilbage til SPI-modulet. Alle FPGA moduler og deres sammenspil kan ses i figur 23.




Figur 23: diagram over systemet

# 8 Software

I dette projekt bruges en Tiva C series Cortex-M4F microcontroller. Denne har til opgave at binde brugeren sammen med den hardware der styrer pan/tilt-systemet. Der er udviklet software til microcontrolleren. Denne gør brug af perifere moduler til at kommunikere med brugeren og FPGA'en. Se figur 24.




Figur 24: Softwarens position i flowet af data imellem brugeren og pan/tilt-systemet

## 8.1 Valg af Operativsystem

Det er et krav til projektet, at microcontrolleren skal have skedulering. Skedulering kan ske ved brug af styresystem. Der er forskellige typer styresystemer, som skedulerer på forskellige måder. Det blevet valgt at finde et styresystem, som er fuldt udviklet og opfylder alle disse opstillede krav:

- Styresystemet skal understøtte en Cortex-M4F Processor
- Systemet skal tilbyde følgende services:
  - Tasks
  - Semaphores
  - Queues
- Styresystemet skal tilbyde Hard Real-Time funktionalitet.
- Styresystemet skal kunne være i en Tiva tm4c123gh6pm's hukommelse.
- Styresystemet skal være veldokumenteret.

Der vælges at bruge et realtids operativsystem, da det ønskes at enkelte tasks, såsom reguleringen, skal køre med faste periodiske tidsintervaller. Der er besluttet at bruge preemtive skedulering, så en task kan afbrydes for at køre en anden task, som skal køre med et jævnt interval. Styresystemerne som blev overvejet kan findes i skemaet, over forskellige systemer, i bilag A.3. Det ses i tabel 9 at det er muligt, ud fra de opstillede krav, at bruge ChibiOS/RT, BRTOS, FreeRTOS, Nucleus RTOS og TI-RTOS Kernel (SYS/BIOS). Den fulde undersøgelse findes i bilag A.3. Disse fem systemer blev efterfølgende undersøgt nærmere. Under denne undersøgelse blev det konkluderet at der ikke ville være nogen fordel ved at vælge et system over et andet. Der blev valgt at bruge FreeRTOS som styresystem.

| **Disse operativ systemer opfyldte alle krav** | **Disse operativ systemer opfyldte ikke alle krav** |
|---|---|
| Nucleus RTOS | BeRTOS |
| TI-RTOS Kernel (SYS/BIOS) | Embox |
| ChibiOS/RT | FreeOSEK |
| FreeRTOS | Embkernel |
| BRTOS | Fusion RTOS |
| | eChronos |
| | distortos |

Tabel 9: Kompatibilitet af styresystem

## 8.2 Prioritering i FreeRTOS

I et embedded system arbejdes der med tasks. En task er et stykke af softwaren, som har sin egen opgave, design og implementering. FreeRTOS er et Preemptivt styresystem. Det betyder at

FreeRTOS bestemmer hvornår en task får lov at arbejde på CPU'en. I FreeRTOS kan en task være i 4 stadier:

- Running
- Ready
- Blocked
- Suspenderet

I FreeRTOS vil en højere prioriteret task, der kommer i ready stadiet altid sætte en lavere prioriteret task i blocked stadiet, for selv at kunne komme i running stadiet. Tasks af samme prioritet kan ikke skifte hinanden ud af running stadiet. Når en task bliver skabt, får den tildelt en prioritet.

## 8.3 FreeRTOS Application Programming Interface

FreeRTOS tilbyder en lang række funktionaliteter [2]. I dette projekt gøres der brug af følgenede funktionaliteter:

- Task kontrol: Disse systemkald bruges til at ændre stadiet af en task, samt initialisering.
- Queues / køer: Queues bruges til at kommunikere på tværs af tasks. I FreeRTOS har queues en specifik længde, og en specifik data type.
- Events: Disse er også kendt som flag. Deres funktion er at signalere events på tværs af tasks i systemet. Events inddeles i event grupper. En event gruppe indenholder 8 bit, og hvert bit er et flag.

## 8.4 Krav til system design

Softwarsystemet skal designes efter det løsningsforslag, som er beskrevet i sektion 4. For at kunne designe det specificerede system er der nogle krav der skal følges, software systemet skal:

- Gøre brug af perifere input moduler, sådan at systemet kan interagere med en bruger. Et eksempel på sådan et modul kunne være en knap.
- Kommunikere med FPGA'en, der forbinder microcontrolleren til pan/tilt-systemet.
- Regulere pan/tilt-systemet efter brugerens ønske.
- Nulstille pan/tilt-systemet, til en udgangs position.
- Interagere intuitivt med brugeren igennem et LCD, og de før nævnte perifere input moduler.

## 8.5 System design

I designet af softwaresystemet er der udviklet moduler som tager højde for in- og output, regulerer pan/tilt-systemet og binder softwaren sammen. Der er lavet overvejelser omkring kommunikationen

imellem tasks, og sammenspillet imellem modulerne. En liste over alle moduler kan findes i tabel 10.

| **Modul navn** | **Type** |
|---|---|
| Numpad | Driver |
| Button | Driver |
| Set position | Driver |
| LCD | Driver |
| SPI | Driver |
| Homing | Task |
| User interface | Task |
| Regulering | Task |

Tabel 10: Moduler der er blevet udviklet i dette projekt og deres typer.

### 8.5.1 Task diagram

Task diagrammet i bilag A.4 beskriver sammenspillet mellem de tasks og drivere, der er udviklet i dette projekt. Task diagrammet er printet på A3 og kan foldes ud. Dette task diagram kan være hjælpsomt til forståelsen af de kommende afsnit.

### 8.5.2 Task kommunikation

I et task orienteret system må task og drivere ikke kommunikere direkte med andre task og drivere. Der er behov for en kommunikationskanal. Her arbejdes der med queues, events og state buffere.

#### 8.5.2.1 Queue

En queue bruges til at kommunikere data mellem moduler. For queues tales der om en producer og consumer. Produceren skriver data til queuen og consumeren læser data fra queuen. Når der arbejdes med queues kan der opstå komplikationer i forhold til at have flere consumere og producere til den samme queue. En queue med multi-producer/single-consumer giver komplikationer med oprindelsen af den data consumeren henter. En queue med multi-consumer/single-producer giver komplikationer i forhold til tab af data. Derfor er der i dette projekt en specifik queue for hver kommunikations linje. Det betyder at et modul har en direkte kommunikation linje til et andet. Dette giver også en lavere kobling i systemet, da et modul kan blive skiftet ud, så længe det stadig overholder data strukturen for queuen.

#### 8.5.2.2 State buffer

En statisk værdi der skal være tilgængelig flere gange for specifikke moduler, bliver lagt i en state buffer. Specielt for denne type buffer er at ingen må konsumere dataen. En sådanne buffer sættes op i FreeRTOS med en queue af længden én. For at en queue kan fungere som en state buffer er der en række regler for consumere og producere.

- Der må kun være en producer til state bufferen.
- En consumer må kun tage en kopi af værdien.

- Produceren skal overskrive den nuværende værdi i state bufferen.

I dette projekt er der kun én state buffer. Denne indenholder data fra homing tasken. Regulerings tasken bruger værdien i denne state buffer.

#### 8.5.2.3 Events

Events bruges når der ikke er behov for at kommunikere information, men behov for at signalere et event. I FreeRTOS tilbyder systemkald at et modul kan vente på et event. Denne ventefunktionalitet bruges til suspendering og aktivering af moduler. Dette gør at flowet af programmet kan styres af UI tasken.

### 8.5.3 Perifer Drivere

Softwaresystemet gør brug af perifere moduler til at modtage inputs og give outputs. Der gøres brug af fem perifere moduler. En liste over perifere moduler og deres funktionalitet kan ses i tabel 11.

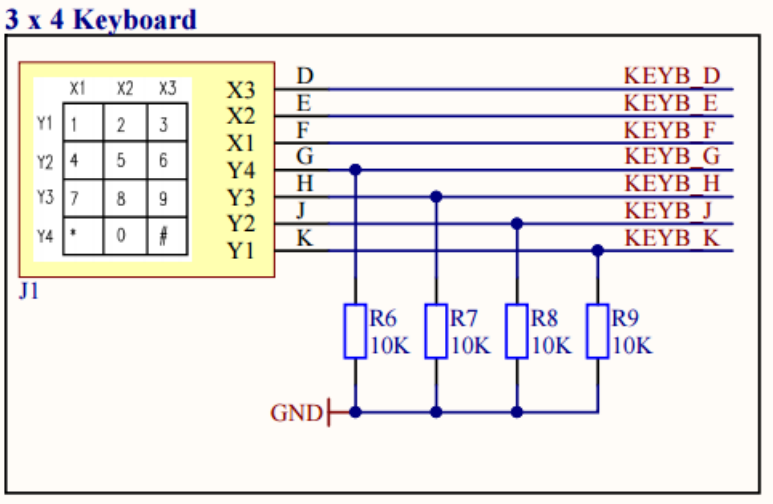
| **Perifere modul navn** | **Beskrivelse af funktionalitet** |
|---|---|
| Numpad | Dette modul tager et numerisk input fra brugeren. |
| Knapper | Dette modul består af to knapper som tager input fra brugeren. |
| LCD | LCD'en er systemets vindue til brugeren. Denne bruges i samarbejde med user interfacet, til at lade brugeren følge fremgangen af programmet. |
| Set position | Dette modul bruges af user interfacet. Den har til opgave at hente informationer om den nuværende position af pan/tilt-systemet. |
| SPI | Dette modul kommunikerer med FPGA'en. |

Tabel 11: Liste over de perifere moduler der giver input til microcontrolleren, og output til brugeren eller FPGA'en.

#### 8.5.3.1 Numpad

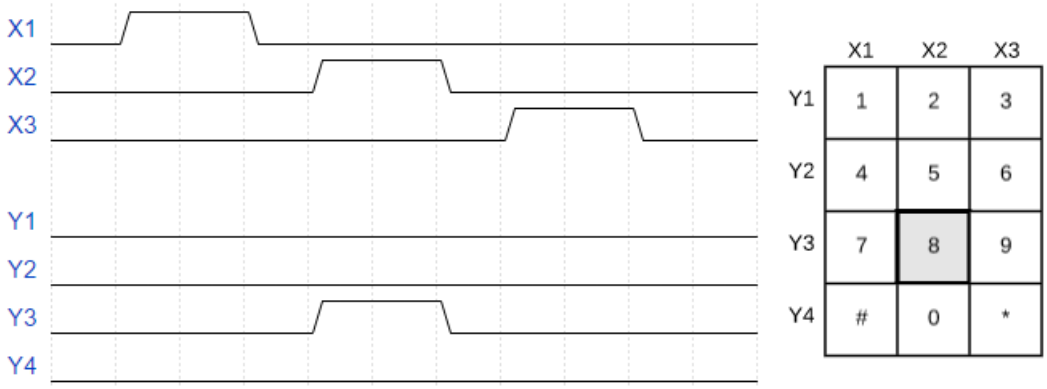
Numpad'en har sin egen hardware funktionalitet. Det er numpad driverens opgave at dekode denne funktionalitet. Numpad'en har et 4 x 3 indeks, bestående af henholdsvis en Y og X akse. Denne er indekseret Y1 til Y4 og X1 til X3. Betragt figur 25.

Y indekset er forbundet til GND. Numpad'en fungerer således, at når en knap bliver trykket ned vil der blive etableret en forbindelse imellem knappens X indeks, og knappens Y indeks.




Figur 25: Det skematiske layout for numpad'en [13]

Ved systematisk at skifte mellem hvilket X-indeks der går højt, og læse på hvilket Y-indeks der ligeledes går højt, kan der krydsrefereres for at bestemme hvilken knap der er blevet trykket. Eksemplet på figur 26 viser hvordan der systematisk skiftes mellem X-indekset, og måles på hvornår et Y-indeks går højt. I dette eksempel kan det konkluderes at knappen der er trykket er '8'.




Figur 26: Eksempel på krydsreferering af X og Y-indekseringerne

#### 8.5.3.2 Knapper

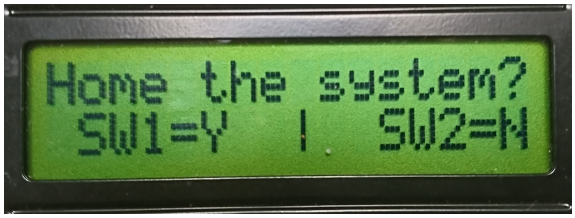
Denne task kontrollerer om portene forbundet til henholdsvis SW1 og SW2, går lav. For at undgå knapperne trigger flere gange pr. tryk, er der lavet debounce vha. et flag som fortæller om knappen er sluppet og en timer som begrænser hvor hurtigt knappen kan slå til og fra.

Når en knap trykkes og flaget er sat, nulstilles flaget og der sendes en karakter til en queue. 'A' for SW1 og 'X' for SW2.

Når en knap slippes igen og flaget ikke er sat, startes timeren og flaget bliver sat igen.

#### 8.5.3.3 LCD

Det er et krav til softwaren at kunne give brugeren af systemet feedback. Det er vurderet at tre lysdioder ikke kan give brugeren en tilstrækkelig mængde information på en intuitiv måde. Derfor benyttes et LCD med plads til 16x2 karakterer. Se figur 27.

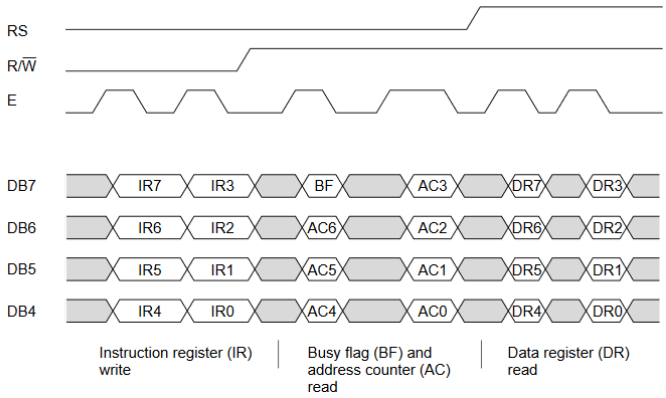



Figur 27: Eksempel på LCD

**Teori** Et standard 16x2 LCD kan interfaces parallelt med en 4-bit eller 8-bit data bus. Derudover er der tre signaler der styre transaktionstypen:

- R/W: Styrer om der foretages en skrivning eller en læsning
- Enable: På nedadgående flange af dette signal aflæses indholdet af databusen
- RS: Angiver om typen af information, der sendes er en instruktion eller data

Et diagram for dette er vist i Figur 28.




Figur 28

LCD'et operer på 8-bit data. Som vist i Figur 28 betyder det, at ved en busbredde på 4-bit skal der sendes data to gange for at modtage 8-bit data.

**Design** Der er i forbindelse med projektet designet en LCD driver. Driveren er designet ud fra følgende ønsker:

- Lavt CPU forbrug
- Minimere mængden af data der skal skrives til LCD'en.
- Nemt at bruge

- Hurtig

I løsningen er der lagt vægt på brugen af en display buffer, samt en "dirty" buffer. Driveren bliver således instrueret i at opdatere LCD'et, når der er et bit der er sat i dirty bufferen. Derefter bliver dirty bufferen gennemgået. Dirty bittet afspejler de element i display bufferen der skal opdateres. Driveren modtager således en hel display buffer fra en bruger applikation, igennem en queue. Driveren sammenholder den indkommende buffer med den interne display buffer, og ændrer de karaktere, som er forskellige. Dette forsager at bits bliver sat på de tilhørende positioner inden i dirty bufferen. Hvilket forsager at der kun opdateres de karaktere der er blevet ændret.

**Test** LCD driveren er blevet testet i forbindelse med UI uden fejl.

#### 8.5.3.4 SPI

Microcontrolleren der er brugt i dette projekt tilbyder en SPI funktionalitet. I opsætningen af denne SPI er der en række muligheder for funktionaliteter i protokollen:

- Mode

  SPI'en på microcontrolleren kan blive konfigureret til master eller slave mode [5]. I master mode er det microcontrolleren der styrer clocken og Slave select pinden. Fordi at hele flowet af programmet udspringer fra microcontrolleren, og UI tasken, er det microcontrolleren der skal være master i kommunikationen. Microcontrolleren er konfigureret til at være i master mode.

- Båndbredde

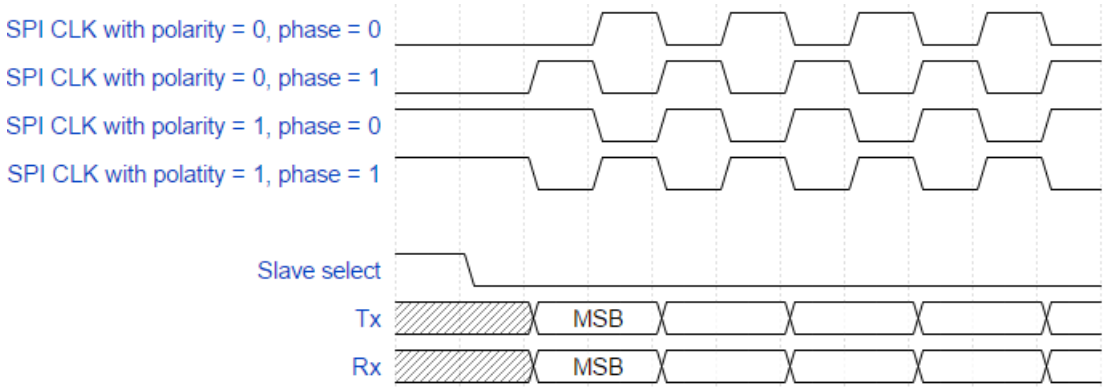
  Båndbredden på SPI'en er bestemt af 2 værdier. En clock prescaler, og en serial clock rate. Sammenhængen mellem båndbreden, og disse to værdier er givet ved formel 17 [6].

$$Baudrate = \frac{system \quad clk}{(clk \quad prescaler * (1 + serial \quad clock \quad rate))} \tag{17}$$

  Ved master mode er minimum værdien for *clk prescaler* to [7]. SPI kommunikationen kan altså ikke gå hurtigere end det halve af system clocken. System clocken på microcontrolleren er 16 MHz.

- Clock polaritet og clock fase

  Clock fasen bestemmer om data skal læses på første eller anden flanke. Clock polariteten bestemmer om clock'en starter med at være høj eller lav. Figur 29 viser hvordan konfigurationen af clock fasen og polariteten påvirker datatransmissionen [8].




Figur 29: Illustration af clock fasen og clock polaritetens betydning for data transmisionen

Da SPI kommunikationen på FPGA'en er designet efter opsætningen på microcontrolleren, er der ingen begrundelse for en specifik opsætning. Clock fasen og clock polariteten er sat til at være 0.

- Data størrelse
  SPI'en på microcontrolleren giver mulighed for valg af data størrelse. Denne data størrelse kan være 4 til 16 bit [9]. SPI'en vil blive brugt til 2 funktioner.
  - Skrive PWM til FPGA'en
    Der sendes PWM til FPGA'en. Denne PWM værdi går fra 0 til 254. Dette er 8 bit. Når der sendes PWM-værdi stiller kommunikationsprotokollen krav for FPGA'en, om en adresse og en retning. Dette er 6 ekstra bits.
  - Hente encoderværdier fra FPGA'en
    Opløsningen på encoderværdierne, med 0 indeksering, er 1079. For at tælle til 1079 er der behov for 11 bit. Ved hent af encoderværdier stilles der også krav til en adressering, men ikke en retning. Dette er 4 ekstra bit.

  Det betyder at der ved PWM er behov for $(8 + 6) = 14$ bit, og ved hent af encoderværdier $(11 + 4) = 15$ bit. Data størrelsen for SPI'en er sat til 16 bit.

Når der sendes og læses data fra den indbyggede SPI i microcontroleren, ventes der på et hardware flag, som microcontrolleren selv sætter. Dette gøres for at sikre at der ikke bliver læst korrupt data [10].

### 8.5.4 Intuitiv Robotstyring

Til dette projekt er der valgt, som applikation, en funktionalitet, som faciliteter intuitiv robotstyring. Idéen er at det skal være muligt fysisk at flytte systemet til nogle ønskede positioner, gemme positionerne og derefter gengive disse positioner i korrekt rækkefølge.

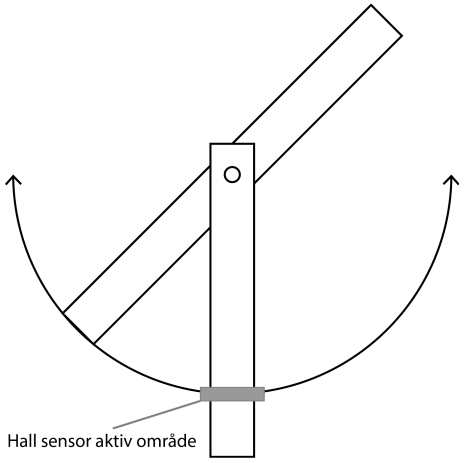
Denne intuitive robotstyring virker ved at gemme de pågældende positionsværdier for de to akser. Herefter, bliver der givet mulighed for indtastning af endnu en position eller afspille de gemte positioner.

### 8.5.5 Homing

Når pan/tilt-systemet bliver initialiseret, er der ingen garanti for at systemet er placeret i nulpunktet. Der er derfor udviklet en homing funktionalitet i software systemet.

#### 8.5.5.1 Homing problemet

Som tidligere nævnt detekteres nulpunktet af en hall-sensor på pan/tilt-systemet. I et ideelt scenarie registrerer hall sensoren kun magneten på rammen i ét encodertick, når den står nøjagtigt over hall-sensoren. Dette er ikke tilfældet. Hall sensoren måler magnetfelt. I pan/tilt-rammen er der monteret en magnet. Denne magnet aktiverer hall-sensoren i et område nær sensoren. Dette område kan ses som det markerede område på figur 30. Da størrelsen på dette område ikke er kendt er det svært at indstille systemet. Der skal derfor tages højde for dette i udviklingen af homing modulet.




Figur 30: Område hvor hall-sensoren er aktiveret

#### 8.5.5.2 Homing af Tilt

Som nævnt i sektion 7.3.5 vil bit 16 i kommunikationen med FPGA'en indikere at rammen befinder sig inden for dette område af hall-sensoren. Hvis det antages at området med zero flag er lige stort på hver side af hall-sensoren, kan det antages at det reelle nulpunkt befinder præcist i midten af de værdier. Ved at bevæge tilt-rammen med en minimum hastighed, for at reducere overshoot, og måle hvornår der kommer en encoderværdi med zero flag, kan det bestemmes hvornår området starter. Ved herefter at oversample encoderværdierne, er det muligt at se hver eneste ændring i encoderværdien, og gemme værdierne indtil der læses en encoderværdi uden zero flag. Såfremt der ikke er ændret retning i dette område, indikerer denne encoderværdi slutningen på området. Alle de læste encoderværdier inden for området tælles, og den midterste værdi må derfor være

nulpunktet. Tilt-rammen kan nu flyttes til denne encoderværdi. Dette er det nye nulpunkt for tilt-rammen. Hvis tilt-rammen starter inden for zero området, forsætter rammen til den har forladt zero området, og starter herefter forfra med homingen.

#### 8.5.5.3 Homing af Pan

Fremgangsmåden for pan-rammen er næsten magen til fremgangsmåden for tilt-rammen, med den undtagelse at tilt-rammen kan bevæge sig ubegrænset, mens pan-rammen har et fysisk endepunkt. Dette fysiske endepunkt begrænser pan-rammen til kun at kunne flytte sig under 540 encoderticks. Da hall-sensoren befinder sig i midten af dette område, og da det ikke kan bestemmes hvilken side af hall-sensoren pan-rammen er placeret, er det nødvendigt at implementere et retnings skift. Det kan antages at når pan-rammen står stille, vil encoderværdien ikke ændre sig. Først igangsættes pan-rammen. Ved at læse et antal samples og vurdere om de allesammen har den samme værdi, kan det konkluderes at pan-rammen står stille. Hvis pan-rammen står stille, må det antages at den har ramt det fysiske endepunkt og pan-rammen skifter retning. Resten af homingen forgår på samme måde som på tilt-rammen. Det nye nulpunkt for pan- og tilt-rammen gemmes og lægges i state bufferen omtalt i sektion 8.5.2.2.

### 8.5.6 User interface design

Der er udviklet to kontrol tasks. En regulering, og en homing. Der er udviklet 6 perifere drivere. Alle disse moduler har deres egne funktionaliteter og ansvar for deres egne opgaver. En user interface task er designet til at binde hele systemet sammen. For at opnå så lav kobling som muligt er det kun user interfacet der ved noget om hele systemet. User interfacet skal ved hjælp af drivere interagere intuitivt med brugeren. Det antages at når systemet startes, er det første brugeren vil, at nulstille systemet. Herefter skal der vælges mellem de to input typer, enten numpad driveren, eller set position driveren.

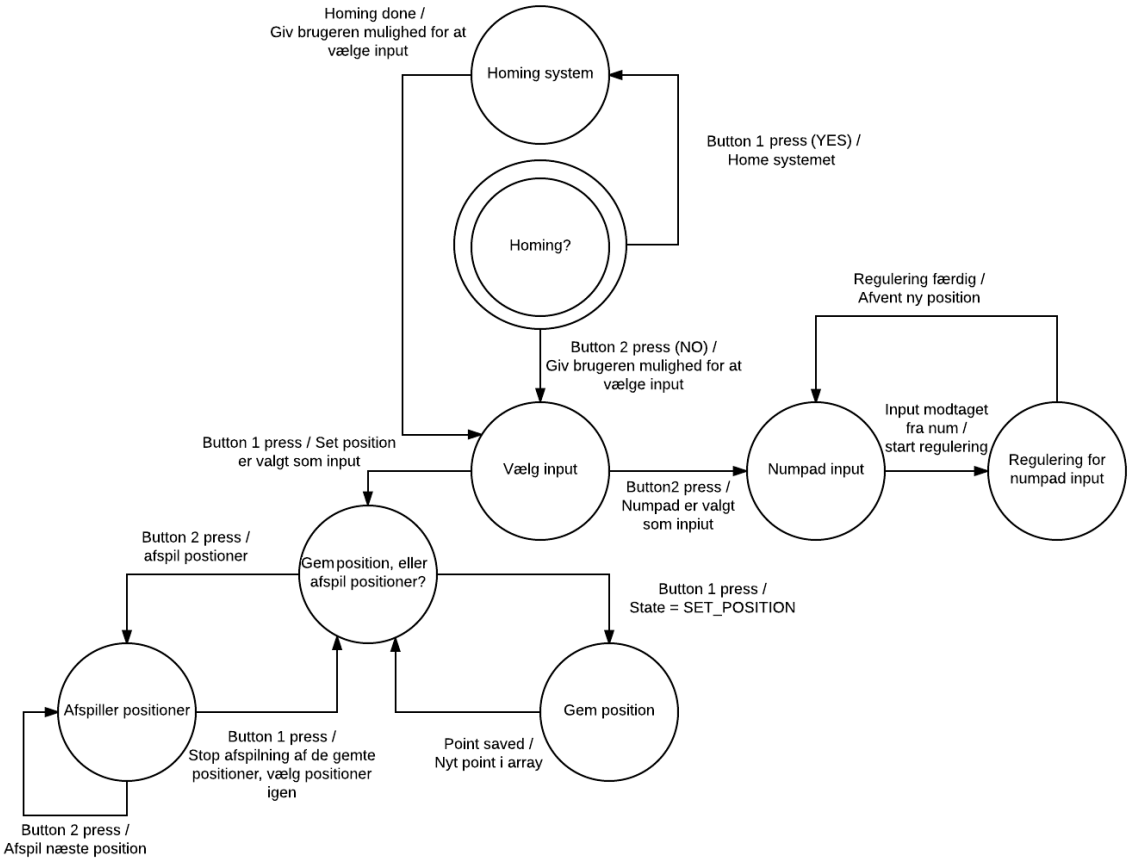
- Numpad som input

  Brugeren skal have mulighed for at taste en pan- og tilt-position, hvor efter pan/tilt-systemet flytter sig til den indtastede position.

- Intuitiv robotstyring som input

  Brugeren skal have mulighed for at gemme et antal positioner, ved fysisk at flytte systemet til de ønskede positioner og efterfølgende afspille disse positioner.

Figur 31 illustrerer de stadier som user interface tasken går igennem, og hvilke handlinger der kan forekomme for at ændre stadie.




Figur 31: Stadier for bruger interface tasken, og interaktioner der ændrer stadierne

## 8.6 Implementering

Interaktionen mellem moduler er implementeret i UI-tasken. Her er der taget brug af events og queues til at styre fremgangen af tasken. De følgende afsnit gennemgår de forskellige stadier af UI tasken. Fremgangen kan følges i figur 31.

### 8.6.1 User interface tasken

Alle stadier af UI tasken gør brug af LCD'en til at skabe et intuitivt flow af programmet.

- Homing?

  User interface tasken starter med at give brugeren mulighed for at home systemet. Input fra knapperne angiver brugerens valg.

- Homing system

  Pan/tilt systemet-homer. Når homingen er færdig, signaleres UI tasken igennem et event.

- Vælg input

  Brugeren bliver tilbudt to input muligheder. Enten numpad, eller set position. Input fra

knapperne angiver brugerens valg.

- Numpad input

  Når numpad er valgt som input skal brugeren først indtaste fire numeriske værdier for tilt-positionen. Herefter skal brugeren indtaste fire numeriske værdier for pan-positionen. Der skal indtastes fire numeriske værdier fordi den højeste encoderværdi er 1079. Ui tasken forventer altid fire indputs pr. ramme i dette stadie. Hvis der ønskes en værdi på mindre end fire karaktere skal der tastes '0' på de højest betydende positioner.

- Regulering for numpad

  Dette stadie kommunikerer den indtastede position af pan/tilt-systemet til regulerings tasken. Et event signalerer når regulerings tasken er færdig.

- Gem eller afspil positioner?

  Dette stadie spørger brugeren om der skal gemmes en ny position, eller om de gemte positioner skal afspilles. Se figur 32.


Figur 32: LCD Menu: Gem ny position eller afspil positioner

Input fra knapperne bruges til at indikere brugerens valg.

- Gem position

  Dette stadie gemmer de aktuelle tilt- og pan-encoderværdier. Derefter bliver der sendt et start-event til numpad-tasken så der via. numpad'en kan indtastes en "hold position-tid" for positionen. Denne tid bliver gemt sammen med positionen. LCD outputtet ses på figur 33.


Figur 33: LCD Menu: Indtast antal millisekunder positionen skal holdes

Derefter bliver en tæller-variabel incrementeret så der er klar til at blive gemt en ny position.

- Afspil positioner

  Dette stadie sender de gemte positioner til reguleringstasken gennem en queue og derefter sendes et start-event til regulerings-tasken. Derudover sendes positionerne også til LCD-tasken, sådan at den aktuelle position vises på LCD'en. Se figur 34. Et event signalerer

når regulerings-tasken er færdig. Den næste position bliver sendt efter den indtastede "hold position-tid" er gået. Knapperne kan også bruges til at gå gennem listen af gemte positioner, henholdsvis SW2 for at gå til næste position, eller SW1 for at gå til forrige position.


Figur 34: SW2 = næste position, SW1 = forrige position. Nederst vises den aktuelle position

## 8.7 Delkonklusion

Et krav til projektet var at bruge tasks, tage højde for hvordan disse skal skeduleres, og hvordan koblingen imellem dem kan minimeres. Alt software, perifere drivere og øvrige tasks, er udviklet med lav kobling. Alle moduler dekoder deres eget input, og kommunikerer det til en specifik queue i systemet. Denne lave kobling fungerer fordi datakommunikationen imellem modulerne er sat til en fast data struktur.

Microcontrolleren kommunikerer som ønsket med FPGA'en, henholdsvis master og slave, gennem SPI. Microcontrolleren kan sende og modtage alle de relevante data til og fra pan/tilt-systemet. Da kommunikationen mellem microcontrolleren og FPGA'en ligger til grund for resten af projektet, vurderes det dermed at SPI-kommunikationen opfylder kravene.

Det er lykkedes at opfylde kravet om et intuitivt brugerinterface som gør det let at betjene pan/tilt-systemet. Brugeren kan betjene systemet ved fysisk at tage fat i rammerne og flytte dem til de ønskede positioner. Efter positionerne er gemt, kan brugeren skifte mellem de gemte positioner. Brugeren kan hele tiden se de relevante informationer via. LCD'et. Det vurderes at pan/tilt-systemet kan betjenes uden større kendskab til de tekniske detaljer.

# 9 Test

## 9.1 Test af præcision

Systemet skal være i stand til at flytte sig hen i en given position. Her ønskes en høj præcision. For at undersøge hvor nøjagtig systemet er, blev der lavet følgende test.

### 9.1.1 Fremgangsmåde:

Intuitiv robotstyring blev brugt til at vælge to positioner som systemet skulle bevæge sig mellem. Encoderværdierne fra både pan og tilt for de valgte positioner blev aflæst på LCD'et. Systemet bevægede sig mellem de to positioner 20 gange, henholdsvis 10 gange til hver position. Ved hver

position blev de aktuelle encoderværdier sendt til en PC gennem UART og sammenlignet med encoderværdierne for den ønskede position. Konstanter og resultater kan ses på tabel 12.

### 9.1.2 Resultater:

| **Tilt-Test** | **Position proposional konstant = 9** | | | | | | | **Speed proposional konstant = 10** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ønsket position:** | **Målinger:** | | | | | | | | | | **Gennemsnit:** |
| 277 | 277 | 277 | 277 | 277 | 277 | 277 | 277 | 277 | 277 | 277 | 277 |
| **afvigelse:** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 813 | 813 | 813 | 813 | 813 | 813 | 813 | 813 | 813 | 813 | 813 | 813 |
| **afvigelse:** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Pan-Test** | **Position proposional konstant = 10** | | | | | | | **Speed proposional konstant = 40** | | | |
| **Ønsket position:** | **Målinger:** | | | | | | | | | | **Gennemsnit:** |
| 115 | 115 | 115 | 115 | 115 | 115 | 115 | 115 | 115 | 115 | 115 | 115 |
| **afvigelse:** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 391 | 391 | 391 | 391 | 391 | 391 | 391 | 391 | 391 | 391 | 391 | 391 |
| **afvigelse:** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Tabel 12: Tabellen viser positioner, og afvigelser i form af encoderticks

Det ses at både pan- og tilt-reguleringen finder den ønskede position med en afvigelse på under en encodertick, hvilket svarer til 0.3 grader.

### 9.1.3 Delkonklusion:

Testen af præcisionen reguleringen for både TILT og PAN har en afvigelse på under 1 encodertick, hvilket svarer til opløsningen på encoderen. En større nøjagtighed vil kræve en højere opløsning på encoderen end den nuværende 1080 ticks pr. omgang. Det vurderes at denne præcision er acceptabel.

## 9.2 Sammenligning af test og simulering

For at vurdere om simuleringerne har været gode til at beskrive det fysiske system, blev der lavet følgende test.

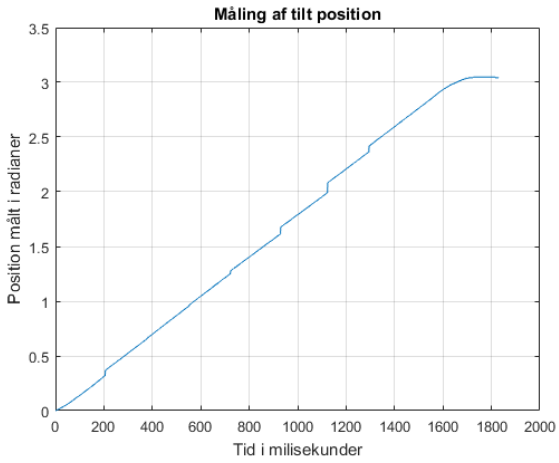
### 9.2.1 Fremgangsmåde:

Systemet blev sat til at bevæge sig mellem to positioner. Under hele reguleringen blev der sendt positions data fra microcontrolleren til en PC via. UART-tasken. Der blev sendt data hvert millisekund. Dette blev gjort for både pan- og tilt-reguleringen. For pan-reguleringen blev der samlet data for både best case, når tilt-rammen står lodret så inertimomentet er mindst, og worst case,

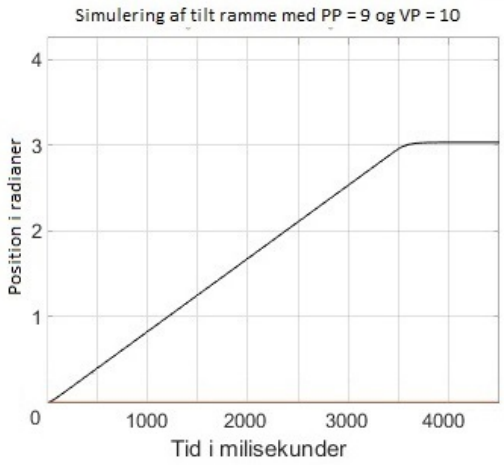
når tilt-rammen står vandret så inertimomentet er størst. Dette blev illustreret på figur 3. Af det opsamlede data blev der lavet grafer som viser position over tid.

### 9.2.2 Resultater:

Grafen for position af tilt over tid fra testen, figur 35, ved siden af grafen for position af tilt over tid fra simuleringen, figur 36.

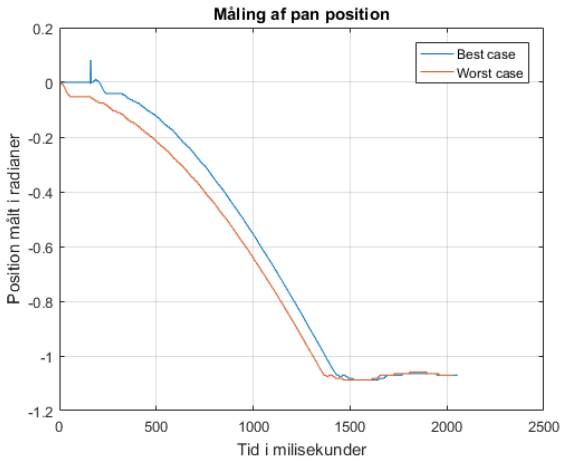



Figur 35: Tilt-test

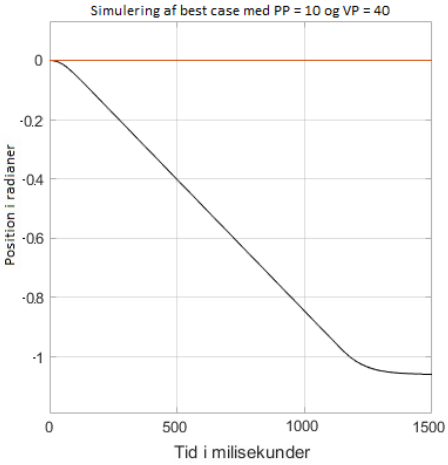



Figur 36: Tilt-simulering

Grafen for position af pan over tid fra testen, med både best case og worst case, figur 37, ved siden af grafen for position af pan over tid fra simuleringen, figur 38.




Figur 37: Pan test med best og worst case




Figur 38: Pan simulering

Det kan ses i figur 35 at rise time på testen af tilt-systemet er dobbelt så hurtig som den simulerede figur 36.

### 9.2.3 Delkonklusion:

På trods af at settling time for tilt er hurtigere end simuleringen, så har simuleringen hjulpet til at lave en tilfredsstillende tilt-regulering.

På grafen for pan-reguleringen fra testen ses det at der er et overshoot efterfulgt af et undershoot. På trods af dette vurderes det at pan-reguleringen er tilfredstillende.

# 10 Perspektivering

Regulering er essentielt i blandt andet droner og fly-controllere, selvbalancerende systemer som Segways og endda styreraketterne i en rumraket. Et to-akset reguleringssystem som dette kan perspektiveres til en gimbal, som er et system, der holder et kamera stabilt selvom brugeren bevæger sig. En gimbal består typisk af to eller tre børsteløse motorer som styres via., input fra gyroskoper og PID regulering. 2-akset gimbals bruges typisk på kameradroner mens håndholdte gimbals typisk har en tredje akse. Se figur 39.

Intuitiv robotstyring konceptet bruges flere steder til intuitivt at betjene diverse typer robotsystemer. Universal Robots bruger f.eks. et lignende koncept til at betjene deres robotarm. Se figur 40. Konceptet øger tilgængeligheden og gør at disse robotter kan betjenes af almindelige mennesker, frem for udelukkende at eksperter kan betjene dem.

Figur 39: FeiyuTech 3-akset kamera gimbal. [14]

Figur 40: Universal Robots kan betjenes med hænderne. [15]

# 11 Diskussion

Der er blevet udledt matematiske formler for motoren, som beskriver motorens position, hastighed og strøm i forhold til spændingen. Der er designet en reguleringssøjfe uden anvendelse af strømmen. Ved simulering blev det undersøgt om det var muligt at undlade strømreguleringen. Simuleringerne viste at en reguleringenssløjfe med kun positionen og hastigheden gav et tilfredsstillende resultat.

Det har efter implementeringen af reguleringssløjfen vist sig at selvom der i simuleringen ses en graf uden overshoot, kan der i praksis stadig ske overshoot. Dette kan skyldes forkerte parametre i form af inertimomenter, friktion i systemet, eller slip på remmen mellem gearene.

FPGA'ens rolle i systemet er at tilbyde eksterne perifere moduler igennem en SPI kanal. Dette kan dog skabe problemer med forsinkelse. Dette skaber også unødvendig kompleksitet grundet den ekstra kommunkation nødvendig ved brug af FPGA. Under ideelle omstændigheder, ville der som udgangspunkt blive brugt periphals inde i en microcontroller hvis tilgængelige. Der blev også valgt, at estimere hastighed i software. Dette er ikke en optimal løsning, da det optager unødigt CPU-tid. Alternativt kan dette gøres på en FPGA. Der er ingen begrænsing, da der ikke bruges en CPU. Der direkte adgang til signalerne fra encoderen, hvor det simpelt kan bruges til hastighedsmåling. Dog ville dette kræve at SPI modulet på microcontrolleren er i brug oftere.
Derudover ville der kunne opnås bedre perfomance i hele systemet hvis der benyttes et multislave system, med flere slave select linjer. På den måde kan man dedikere en Slave select linje pr. modul på FPGA'en. Dette ville mindske data der skulle sendes, da dette betyder at der ikke skal sendes en tom frame med en addresse. Men denne metode vil kræve flere forbindelser mellem microcontroller og FPGA. Det ville også kræve at softwaren understøtter dette. Hvis det er et krav at have kompliceret programmerbar logik, vil en optimal løsning være at bruge en FPGA plus en microcontroller på samme chip. Dette kan f.eks. være en chip fra Xilinx kaldet ZYNQ.

FreeRTOS blev valgt til at skedulere modulerne i softwaresystemet. Softwaresystemet kører sekventielt, så behovet for et preemtivt styresystem er ikke tilstede. I forhold til task-kommunikationen har brugen af queues og events givet systemet en hel del overhead. Dette har givet problemer med stack-overflow. Problemet er løst ved passende indstilling af allokeret hukommelse. Da softwaresystemet er grundigt designet, er der ikke behov for en queue størrelse på mere end én, og en øget størrelse vil kun bidrage til dårlig udnyttelse af ressourcer. Queues har dog haft den positive indvirkning, at de har kunnet tilbyde et fleksibel dataflow. Der har i projektet være anvendt events til at styre aktiveringen af tasks. Dette har fungeret efter hensigten, og har givet et godt flow i systemet. Det har virket efter hensigten at implementere perifere moduler i driver tasks. Det giver mulighed for nemt at genanvende driverene forskellige steder i programmet. SPI driveren er sat op med en data størrelse på 16 bit. Denne er valgt i starten af projektet. Herefter blev implementeringen af SPI på FPGA'en sat op i forhold til denne størrelse. Data størrelsen på frameformatet for SPI'en i dette projekt skal minimum være 15 bit. Den er beholdt til 16 bit. Fordelene ved at ændre data størrelsen til 15 bit var ikke betydelige nok til at fortage ændringer i både VHDL og C koden. Homing tasken der er udviklet har fungeret efter hensigten i test, men efter implementering med regulerings tasken er der opstået problemer. Dette er grundet modstridende versioner af SPI og FreeRTOS konfigurationen. Ydermere skriver homing tasken et offset til en state buffer. Det var tiltænkt, at der skulle ligge et abstraktionlag imellem denne state buffer og reguleringen, således

at systemet ikke skulle tage højde for offset i encoderværdierne. Grundet sen implementering af homing task'en er abstraktionslaget ikke færdigdesignet. Reguleringen virker derfor kun med en manuel nulstilling af systemet. Implementeringen af en UI-task har virket efter hensigten. Det har været let at implementere alle de ønskede funktionaliteter i UI-tasken grundet strukturen af task-kommunikationen.

Der er lavet en præcisionstest samt en sammenligning af systemets regulering og den simulerede regulering. Disse test er med til at give et indtryk af om den udviklede regulering fungerer efter hensigten. Præcisionstesten viste at der ikke var steady-state fejl i positionssetpunktet. I testen der sammenligner systemets regulering med det simulerede, er der en dobbelt så stor rise time på reguleringen i forhold til simuleringen. Dette kan tyde på at der er en steady state fejl på hastighedsregulering, der forsager at responsen ikke er som designet. Dette kan behjælpes ved anvendelse af et integral led i hastighedsregulatoren. Ved pan-systemet kan det ses, at settling time er væsentligt bedre, og at positionsregulatoren virker efter hensigten. Hastighedsreguleringen har et undershoot, der forsager en buet form på grafen. Dette kan også forsages grundet softwarefejl.

# 12 Konklusion

Et kontrolsystem til pan/tilt-systemet er designet og derefter implementeret. Et pan/tilt-system kan reguleres ved hjælp af en lukketsløjfe regulering, med formen af en proportional regulator, med hastighedsregulering samt positionskontrol. Selvom simulation og test ikke stemmer overens med hinanden fungerer reguleringen efter hensigten, og giver et tilfredsstillende resultat, hvis der udelukkende ses på positionskontrol.

En reguleringsløjfe blev designet, og derefter implementeret i C-kode på en microcontroller. Der er desuden valgt en passende metode til skedulering, og hård realtidsoperation.

En SPI kommunikation er blevet etableret imellem en microcontroller og FPGA, ved brug af microcontrollerens indbyggede SPI peripheral, samt et eget udviklet SPI-slave modul, samt med understøttelse af tovejskommunikation.

Styring af DC-motorene blev opnået ved hjælp af egenudviklet PWM-modul, som interfaces med en H-bro. Derudover er der implementeret et dekoder-modul som giver mulighed for aflæsning af position, i forbindelse med en lukketsløjfe regulering.

Udover dette blev der udviklet et intuitivt bruger interface til styring af systemet.

Alle relevante aspekter af projektet, design, valg, simuleringer, tests, hardware og software er beskrevet og dokumenteret i denne rapport.

# Litteratur

[1] Data bladet for EMG30, <https://www.robot-electronics.co.uk/htm/emg30.htm>, 28-05-2017

[2] FreeRTOS API, <http://www.freertos.org/static_menu.html#API_reference>, 28-05-2017.

[3] Modeling and simulation of the EMG30, Se PDF dokument på vedlagte CD, 28-05-2017.

[4] How to Estimate Encoder Velocity Without Making Stupid Mistakes: Part I, https://www.embeddedrelated.com/showarticle/158.php, 28-05-2017.

[5] Tm4c123gh6pm datasheet, afsnit 15. 28-05-2017

[6] Tm4c123gh6pm datasheet, Side 966 SSI control. 28-05-2017

[7] Tm4c123gh6pm datasheet, afsnit 15.3.1 Bit rate Generator. 28-05-2017

[8] Tm4c123gh6pm datasheet, afsnit 15.3.4.3 til 15.3.4.6, Freescale SPI frame format. 28-05-2017

[9] Tm4c123gh6pm datasheet, Side 967 SSI control. 28-05-2017

[10] Tm4c123gh6pm datasheet, Side 971 SSI status. 28-05-2017

[11] Modern control engineering , Katsuhiko Ogata , 5'th edition

[12] Control tutorials, <http://ctms.engin.umich.edu/CTMS/index.php?example=MotorPosition&section=SystemModeling>

[13] EMP board schematic, vedlagt PDF på CD.

[14] Gimbal, <www.bhphotovideo.com>, 28-05-2017.

[15] Universal robots, <www.universal-robots.com>, 28-05-2017.

# A Bilag

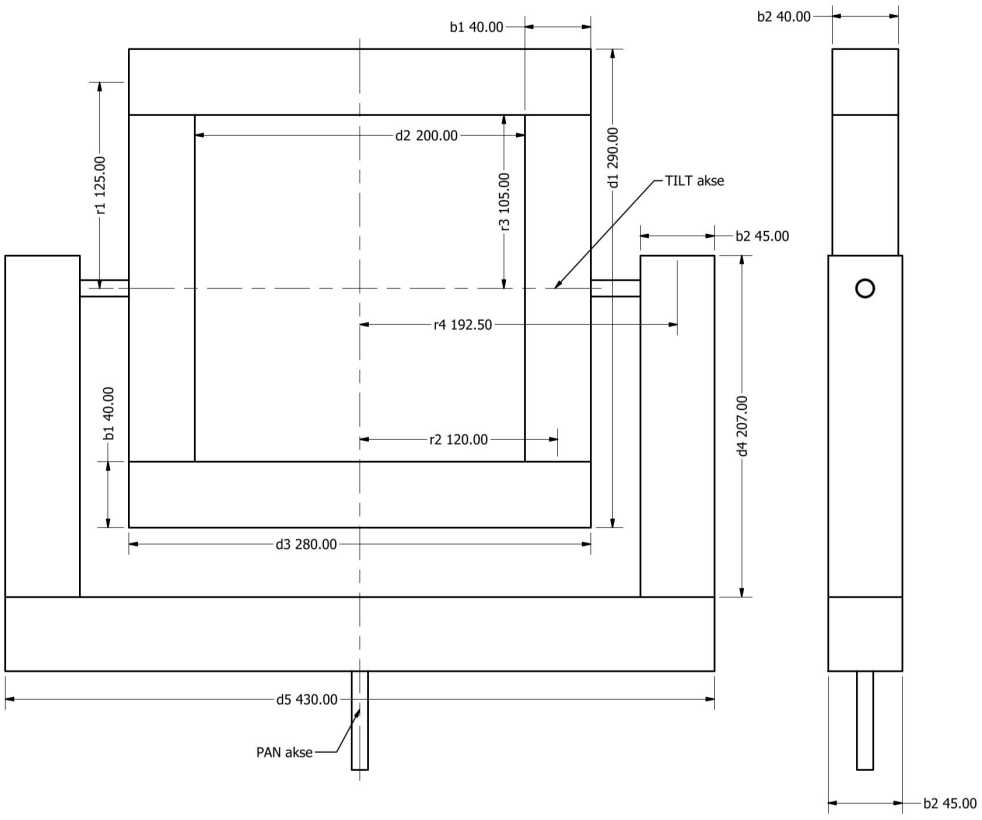
## A.1 CD

### A.1.1 Indhold

- C-Kode
- Video
- VHDL-kode
- Litteratur
- Matlab
- Simulink
- Digital udgave af rapport

## A.2 Beregning af inertimoment

Til at finde inertimomentet for de forskellige akser, er systemet blevet opmål, og disse mål er anvendt til at bestemme vægten af de forskellige dele. Denne Opmåling kan ses i figur 41.




Figur 41: Opmålning af systemets rammer

For at finde massen af stængerne er der udregnet volumen af den pågældende stang, og ganget med aluminiums densitet. Beregning af inerti om tilt-aksen er delt op i to, en for de to vandrette stænger, og en hvor de to lodrette stænger er beregnet som punkt masser. Inertimoment af bund og top stang er udregnet ved massen $m_{d1}$ af stangen med længde d1:

$$I_{tilt\_1} = \left(\frac{1}{12} m_{d1} {d_1}^2\right) \cdot 2 \tag{18}$$

Inertimoment for punktmasserne $m_{d2}$ af de lodrette stænger med længden d2:

$$I_{tilt\_2} = \left(m_{d2} {r_1}^2\right) \cdot 2 \tag{19}$$

Total inerti for tilt-rammen om tilt-aksen:

$$I_{tilt\_total} = I_{tilt\_1} + I_{tilt\_2} \tag{20}$$

Inertimomentet for pan-rammen regnes først for den nederste ramme, og dertil tilføjes tilt-rammen i et best-case og et worst-case.

Inertimoment af bund stangen er udregnet ved massen $m_{d5}$ af stangen med længde d5:

$$I_{base\_1} = \left(\frac{1}{12} m_{d5} {d_5}^2\right) \cdot 2 \tag{21}$$

Inertimoment for punktmasserne $m_{d6}$ af de lodrette stænger med længden d6:

$$I_{base\_2} = \left(m_{d6} {r_4}^2\right) \cdot 2 \tag{22}$$

$$I_{base\_total} = I_{base\_1} + I_{base\_2} \tag{23}$$

Til basens inertimoment ligges et inertimoment for et best og worst case som udregens i formel 26 og **??**

Best case regnes for top og bund masse $m_{d3}$ af stangen med længde d3:

$$I_{pan\_best1} = \left(\frac{1}{12} m_{d3} {d_3}^2\right) \cdot 2 \tag{24}$$

Best case for punktmasserne $m_{d4}$ af de lodrette stænger med længden d4:

$$I_{pan\_best2} = \left(m_{d4} {r_4}^2\right) \cdot 2 \tag{25}$$

Det totale inertimoment for best case:

$$I_{pan\_best\_total} = I_{base\_total} + I_{pan\_best1} + I_{pan\_best2} \tag{26}$$

Worst case regnes som fire punkt masser med masserne $m_{d1}$ og $m_{d2}$ af stængerne længderne d1 og d2:

$$I_{worst\_1} = \left(m_{d1} {r_4}^2\right) \cdot 2 \tag{27}$$

$$I_{worst\_2} = \left(m_{d2} {r_3}^2\right) \cdot 2 \tag{28}$$

Worst case inertimomentet:

$$I_{worst\_total} = I_{base\_total} + I_{worst1} + I_{worst2} \tag{29}$$

| **Inertimoment:** | $kg \cdot m^2$ |
|---|---|
| I_tilt_1 = ((1/12)*(M1*d1)^2)*2 | 0,0096 |
| I_tilt_2 = ((M2*R1)^2)*2 | 0,0148 |
| I_Samlings_punkter | 0,0019 |
| Inerti_Tilt_Ramme total | 0,0263 |
| | |
| I_base_total | 0,0612 |
| I_Pan_best 1 = ((1/12)*M2*d3^2)*2 | 0,0062 |
| I_Pan_best 2 = (M1*r1^2)*2 | 0,0214 |
| I_Pan_best_total | 0,0889 |
| | |
| I_base_total | 0,0612 |
| I_Pan_worst_1 = (M1*r2^2)*2 | 0,0198 |
| I_Pan_worst_2 = (M2*r1^2)*2 | 0,0136 |
| I_Pan_worst_total | 0,0946 |

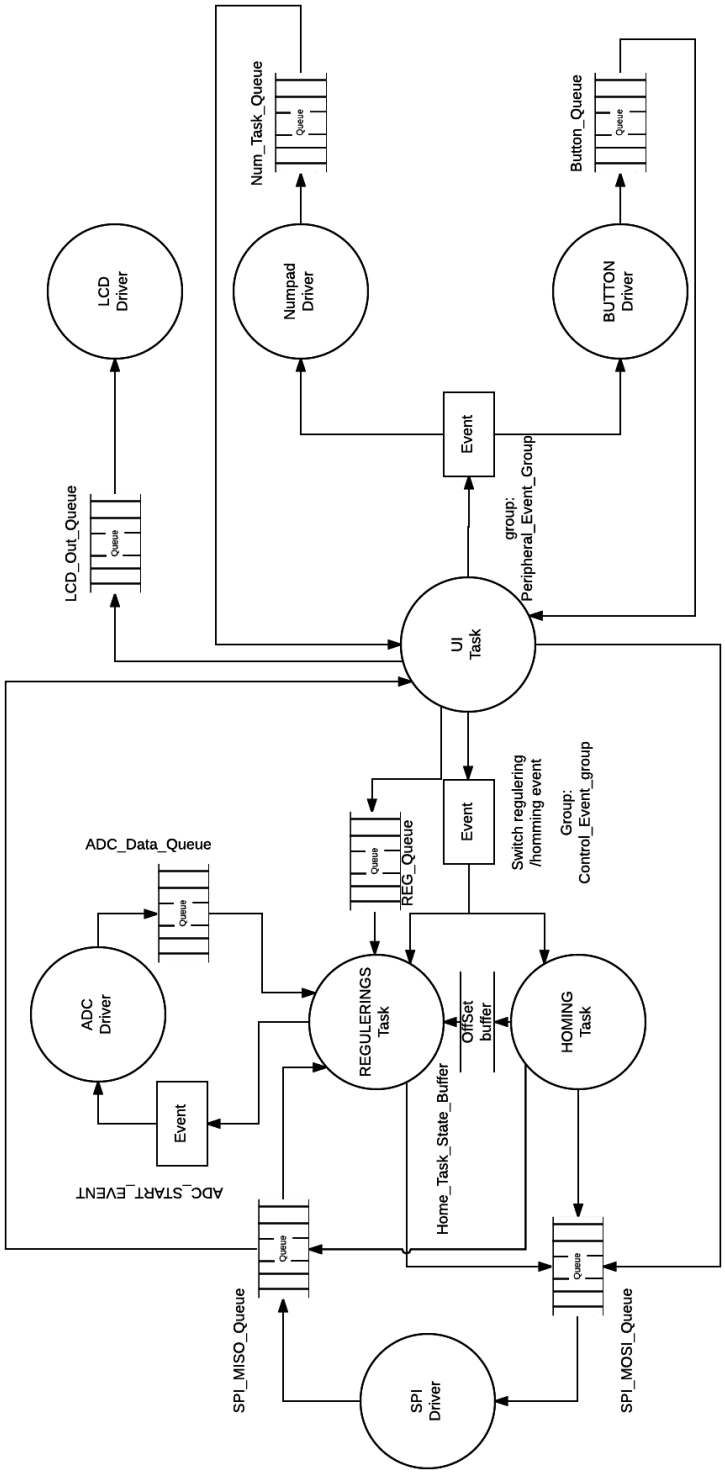
Tabel 13: Inertimoment for systemet

## A.3 Valg af Operativsystem

| **Name** | **Source model** | **Target uses** | **Cortex-M4F kompatibel** | **Tasks** | **Semaphores** | **Kan vaere på Tiva'en** | **Queues** | **Real-Time** |
|---|---|---|---|---|---|---|---|---|
| BRTOS | open source | embedded | Ja | Ja | Ja | Ja | Ja | Ja |
| distortos | open source | embedded | Ja | Ja | Ja | Manglende info | Ja | Ja |
| FreeOSEK | open source | embedded | Ja | Manglende info | Manglende info | Manglende info | Manglende info | Manglende info |
| BeRTOS | open source | embedded | Nej | Ikke relevant | Ikke relevant | Ikke relevant | Ikke relevant | Ikke relevant |
| ChibiOS/RT | open source | embedded | Ja | Ja | Ja | Ja | Ja | Ja |
| eChronos | open source | embedded | Ja | Ja | Ja | Manglende info | Ja | Manglende info |
| Embox | open source | embedded | Muligvis | Ja | Ja | Manglende info | Ja | Manglende info |
| Embkernel | open source | embedded | Ja | Manglende info | Manglende info | Manglende info | Manglende info | Manglende info |
| FreeRTOS | open source | embedded | Ja | Ja | Ja | Ja | Ja | Ja |
| Fusion RTOS | | semi-general purpose | Manglende info | Manglende info | Manglende info | Manglende info | Manglende info | Ja |
| Nucleus RTOS | source code provided | embedded | Ja | Ja | Ja | Ja | Ja | Ja |
| TI-RTOS Kernel | open source | embedded | Ja | Ja | Ja | Ja | Ja | Ja |

Tabel 14: Oversigt over forskellige Operativ systemer

## A.4 Task diagram




Figur 42: Taks diagrammet der beskriver hvordan alle task og drivere i software systemet på microcontrolleren er forbundet