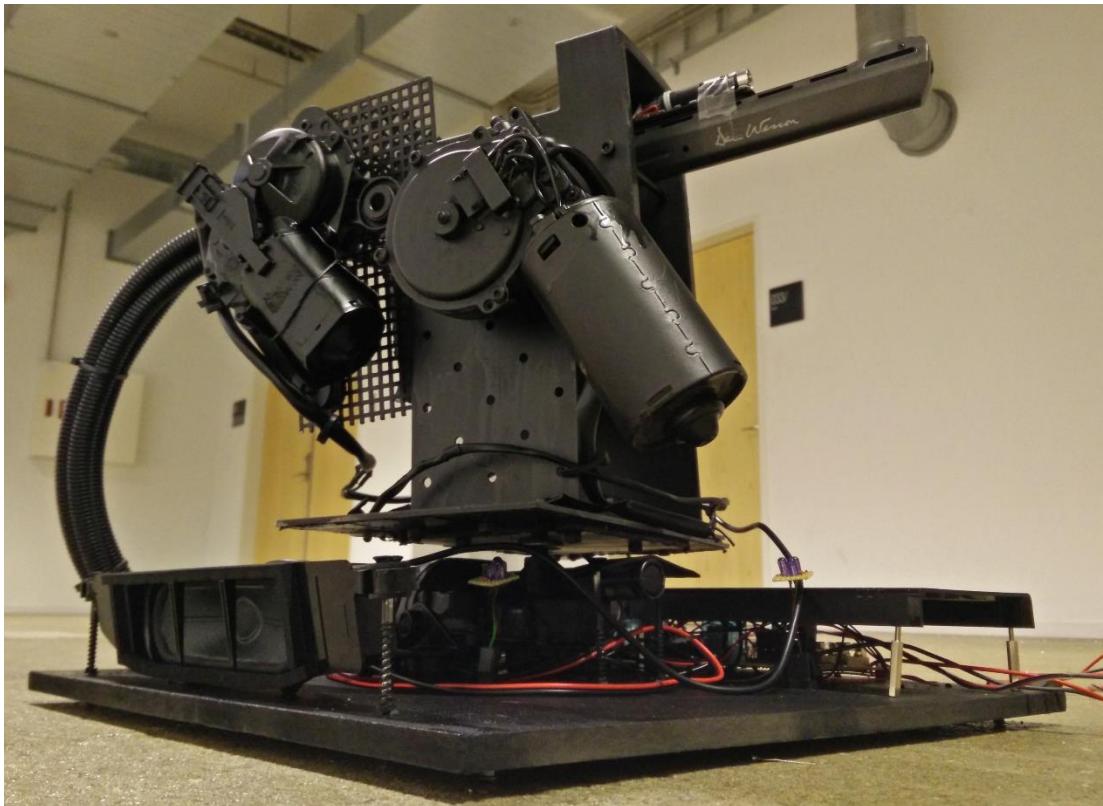


HOME DEFENSE TURRET

3. Semesterprojekt, Gruppe 7



Andreas Engelbrecht Larsen [AEL]	201370065	_____
Árni Þór Þorsteinsson [ATT]	201370318	_____
Dennis Tychsen [DT]	201311503	_____
Joachim Dam Andersen [JDA]	201370031	_____
Kasper Behrendt [KB]	20071526	_____
Lars Rudbæk Andersen [LRA]	201370796	_____
Steffen Fog [SF]	201370497	_____
Vejleder: Michael Alrøe		

Dato: 28/05-2015

Indholdsfortegnelse

Indholdsfortegnelse	2
1. Indledning [SF]	7
1.1. Indhold	7
1.2. Systembeskrivelse.....	7
2. Projektdefinition [Alle].....	8
2.1. MoSCoW	8
2.2. Ikke funktionelle krav.....	8
3. Funktionelle krav [Alle]	9
3.1. Aktørkontekst diagram	9
3.2. Aktør beskrivelse.....	9
3.3. Use case diagram	10
3.4. Use cases (fully dressed).....	11
3.4.1. UC1: Log ind	11
3.4.2. UC2: Log ud	12
3.4.3. UC3: Aktiver Alarm.....	12
3.4.4. UC4: Deaktiver Alarm.....	13
3.4.5. UC5: Udløs våben.....	14
3.4.6. UC6: Genlad	15
3.4.7. UC 7: Bevæg HDT horisontalt.....	16
3.4.8. UC 8: Bevæg HDT vertikalt.....	17
3.4.9. UC9: Udløs alarm	18
3.4.10. UC10: Advarsel.....	18
4. Systemarkitektur [Alle]	19
4.1. Overordnet arkitektur af Home Defense Turret [SF, KB, LRA, ATT].....	19
4.1.1. Blokbeskrivelse af Home Defense Turret.....	20
4.1.2. Internal Block Diagram af Home Defense Turret.....	22
4.1.3. Signalbeskrivelse af HDT	23
5. Systemarkitektur af Controller [SF, KB, LRA, ATT]	24
5.1.1. Block Definition Diagram af Controller	24
5.1.2. Blokbeskrivelse af Controller	25
5.1.3. Internal Block Diagram af Controller	28
5.1.4. Signalbeskrivelse af Controller.....	29

6.	Systemarkitektur af Platform [SF, KB, LRA, ATT].....	32
6.1.1.	Block Definition Diagram af Platform	32
6.1.2.	Blokbeskrivelse af Platform	32
6.1.3.	Internal block definition diagram af Platform.....	34
6.1.4.	Signalbeskrivelse af Platform	35
7.	Systemarkitektur af Motorstyring [SF, KB, LRA, ATT]	37
7.1.1.	Blokbeskrivelse af Motorstyring	37
7.1.2.	Internal block definition diagram af Motorstyring	40
7.1.3.	Signalbeskrivelse af Platform	41
7.2.	Ordliste.....	42
8.	Softwarearkitektur [JDA, DT, AEL]	43
8.1.	Domænemodel	43
8.2.	Beskrivelse af softwarens hovedmoduler.....	44
8.2.1.	Brugerinterface	44
8.2.2.	Indlejret Linux System.....	44
8.2.3.	Matrix keyboard.....	44
8.2.4.	PIR sensor.....	44
8.2.5.	Joystick	44
8.2.6.	Camerafeed.....	44
8.2.7.	Log.....	45
8.2.8.	AntalSkudFil	45
8.3.	Sekvensdiagrammer.....	45
9.	Design og implementering af hardware	51
9.1.	Platform Hardware [ATT]	51
9.1.1.	Motor	51
9.1.2.	Motorstyring Hardware	52
9.1.3.	Motorstyring Dødswitch.....	54
9.1.4.	SOMO-II Hardware.....	56
9.2.	Design Spændingsforsyningsprint [LRA]	57
9.2.1.	Implementering spændingsforsynings print.....	58
9.2.2.	5V forsyning lyssensor	59
9.2.3.	3.7V forsyning laser.....	59
9.3.	Platform Software [ATT]	61

9.3.1.	Software Beskrivelse	61
9.3.2.	UART.....	63
9.3.3.	Styring af horisontal og vertikal motor	64
9.3.4.	Styring af Dødswitch.....	64
9.3.5.	SOMO-II Platform Software [ATT].....	64
9.4.	Controller hardware og software [SF, KB]	66
9.4.1.	PIR sensor.....	66
9.4.2.	Trigger	66
9.4.3.	Joystick.....	67
9.4.4.	ADC.....	67
9.5.	Kommunikation mellem ADC og Indlejret Linux System	68
9.5.1.	Opsætning af SPI kommunikation.....	69
9.6.	Software til ADC og Indlejret Linux System	71
9.7.	ControllerHøjtalere	75
9.7.1.	Implementering af ControllerHøjtalere	76
9.7.2.	Software til ControllerHøjtalere og PIRsensor.....	77
10.	Test af hardware	78
10.1.	Test af Motorstyring Hardware [ATT].....	78
10.2.	Test af Uart på Platform [ATT]	79
10.3.	Test af Software på Platform [ATT].....	79
10.4.	Test af spændingsreguleringsprint [LRA]	80
10.5.	Kommunikation mellem ADC og Indlejret Linux System [SF, KB]	83
10.5.1.	Test af Joystick's x-akse.....	83
10.5.2.	Test af Joystick y-akse	84
10.5.3.	Test af Trigger	85
10.5.4.	Test af PIRsensor	86
10.5.5.	Test af ControllerHøjtalere	87
11.	Design og implementering af software [JDA, AEL, DT]	89
11.1.	Samlet klassediagram	89
11.2.	GUI [AEL]	91
11.2.1.	Login.....	92
11.2.2.	CameraFeed	93
11.2.3.	Genlad	98

11.3.	Backend funktionalitet [JDA, AEL, DT]	99
11.3.1.	Log [AEL].....	99
11.3.2.	Design af software til matrix tastatur.[JDA].....	101
11.3.3.	Design af kommunikationsprotokol [JDA]	103
11.3.4.	Implementering af matrixkeyboard [JDA].....	105
11.3.5.	Implementering af protokol mellem PSoC4 og raspberry pi [JDA].....	111
11.3.6.	Design og implementering af UART [DT]	115
11.3.7.	Implementering af UARTItem og UARTQueue	116
11.3.8.	Implementering af UART klassen.....	118
11.3.9.	Design af JoystickThread.....	122
11.3.10.	Implementering af JoystickThread.....	125
12.	Softwaretest.....	130
12.1.	Cross compiling til raspberry pi [JDA]	130
12.2.	Enhedstest af GUI [AEL]	131
12.2.1.	Login.....	131
12.2.2.	CameraFeed	133
12.2.3.	Genlad	137
12.3.	Enhedstest af Log [AEL].....	138
12.4.	Integrationstest af GUI [AEL]	139
12.4.1.	Login.....	139
12.4.2.	CameraFeed	141
12.4.3.	Genlad	145
12.5.	Integrationstest af Log [AEL].....	147
12.6.	Enhedstest af matrixkeyboard [JDA].....	148
12.7.	Integrationstest af matrixkeyboard	150
12.8.	Enhedstest af protokol-klassen [JDA]	154
12.9.	Integrationstest af protokollen [JDA].....	154
12.10.	Enhedstest UART og UARTQueue [DT]	156
12.11.	Enhedstest af Joystick [DT]	156
12.12.	Integrationstest af Joystick og motorstyring [DT].....	158
13.	Acceptttest [Alle].....	160
13.1.	UC1: Log ind	160
13.2.	UC1: Ext. 1a. Log ind	160

13.3.	UC2: Log ud	161
13.4.	UC3: Aktiver Alarm.....	161
13.5.	UC3: Ext. 1a. Aktiver Alarm	162
13.6.	UC4: Deaktiver Alarm.....	162
13.7.	UC5: Udløs våben.....	163
13.8.	UC5: Ext. 1a. Udløs våben	163
13.9.	UC6: Genlad	164
13.10.	UC6: Ext. 1a. Genlad.....	164
13.11.	UC7: Bevæg HDT horisontalt.....	165
13.12.	UC7: Ext. 1a. Bevæg HDT horisontalt.....	166
13.13.	UC7: Ext. 1b. Bevæg HDT horisontalt.....	166
13.14.	UC8: Bevæg HDT vertikalt.....	167
13.15.	UC8: Ext. 1a. Bevæg HDT vertikalt	167
13.16.	UC8: Ext. 1b. Bevæg HDT vertikalt	168
13.17.	UC9: Alarm	168
13.18.	UC10: Advarsel.....	169
14.	Bilag.....	170

1. Indledning [SF]

Dette afsnit specificerer kravene til systemet *Home Defense Turret* eller forkortet *HDT*.

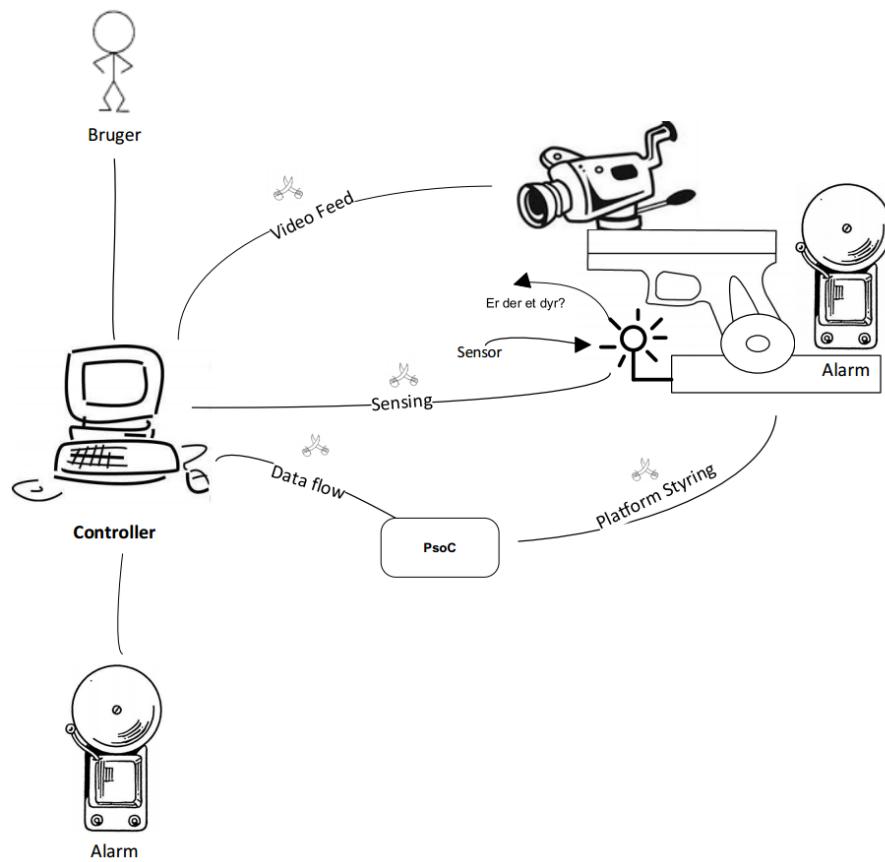
1.1. Indhold

Afsnittet indeholder

- Specifikation af funktionelle krav vha. Use Cases.
- Specifikation af ikke-funktionelle krav.

1.2. Systembeskrivelse

Systemet skal hjælpe ejeren med at beskytte sin ejendom ved at advare og jage rovdyr på flugt, ved at affyre ikke dødelige skud imod uønskede gæster.



FIGUR 1 - RIGT BILLEDE AF SYSTEMETS OPSÆTNING

2. Projektdefinition [Alle]

2.1. MoSCoW

Must

- Man skal gøre brug af PSoC og et Indlejret Linux System.
- Der skal anvendes et brugerinterface til systemkontrol - aktivering og deaktivering af system.
- Brugeren skal gennem et brugerinterface kunne styre HDT horisontalt og vertikalt, samt udløse våbnet.
- HDT skal kunne registrere at der er et mål i nærheden, og afgive besked (alarm) til brugerinterfacet.
- Brugergrænsefladen skal beskyttes med adgangskontrol i form af en adgangskode.
- Der skal kunne monteres et kamera på HDT, der leverer videofeed.

Should

- HDT bør føre en aktivitetslog der logger al aktivitet (antal skydninger, bevægelsesaktivitet).
- Der bør kunne monteres et lasersigte på våbnet, så det kan ses, hvor der sigtes hen.
- En højttaler giver en advarsels lyd på platformen.

Could

- En lyssensor kan registrere, når mørket falder på og aktivere night vision.
- HDT kan estimere afstanden til sit mål med ultralydssensorer.
- Man kan udskifte våbnet, så det passer til kundens behov.
- Flere brugere i systemet, med forskellig brugernavn/adgangskode.

Won't

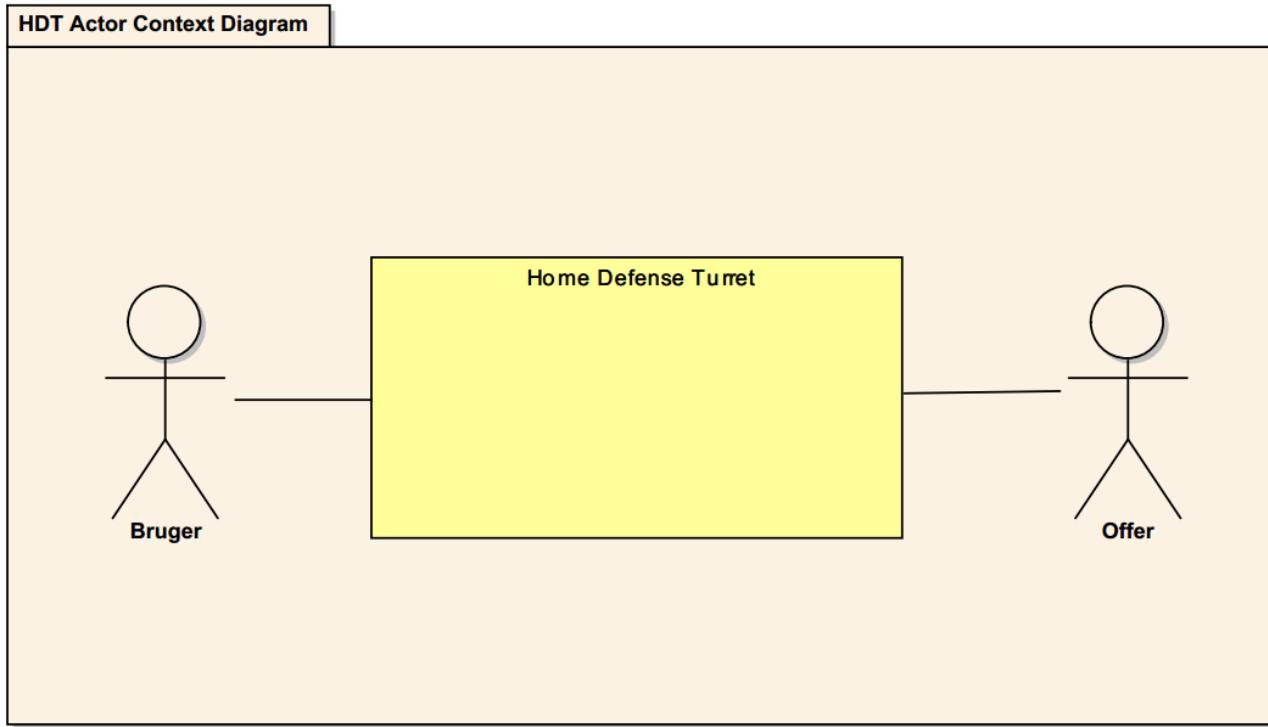
- Lave mulighed for udvide systemet med flere turrets.
- Benytte ansigtsgenkendelsesteknik.

2.2. Ikke funktionelle krav

- Motoren skal kunne dreje fra højre ydrepunkt til venstre ydrepunkt og omvendt på 10 sekunder.
- HDT'en skal kunne ramme 1 meter indenfor der hvor man sigter på en afstand af 10 meter.
- Alarm sensoren skal kunne opfange bevægelse på mindst 3 meter afstand.
- Motoren skal kunne dreje fra højeste ydrepunkt til laveste ydrepunkt og omvendt på 10 sekunder.
- Fra man bevæger joysticket til platformen reagerer, må der max gå 0.5 sekunder.
- Fra man slipper joysticket til platformen stopper, må der max gå 0.5 sekunder.
- HDT'en skal kunne affyre 10 skud/min.

3. Funktionelle krav [Alle]

3.1. Aktørkontekst diagram



FIGUR 2 - AKTØRKONTEKST DIAGRAM

3.2. Aktør beskrivelse

Bruger

Navn på aktør	Bruger
Type	Primær aktør
Beskrivelse	En person der er tilknyttet bopælen, hvor HDT er sat op.

Offer

Navn på aktør	Offer
Type	Sekundær
Beskrivelse	Dyr der ikke hører til på ejendommen.

3.3. Use case diagram



FIGUR 3 - USE CASE DIAGRAM

3.4. Use cases (fully dressed)

3.4.1. UC1: Log ind

Navn	UC1: Log ind.
Mål	Bruger får adgang til System.
Initiering	Bruger.
Aktører	Bruger (primær).
Referencer	Ingen.
Antal af samtidige hændelser	1
Forudsætning	System er tændt og brugerinterfacet er aktivt. System er logget ud.
Efterfølgende tilstand	Bruger er logget ind, videotransmission og menu ses på brugerinterfacet.
Hovedforløb	<ol style="list-style-type: none">Bruger indtaster adgangskode på tastatur og afslutter med firkant. [Ext. 1a: Bruger indtaster forkert adgangskode].System låses op og viser menu.Bruger har adgang til System.
Extensions	[Ext. 1a: Bruger indtaster forkert adgangskode] <ol style="list-style-type: none">Brugerinterfacet udskriver, at koden er forkert. Bruger forbliver på loginskærmen.Går tilbage til pkt. 1 i hovedscenariet.

TABEL 1 - UC1: LOG IND

3.4.2. UC2: Log ud

Navn	UC2: Log ud.
Mål	System låses.
Initiering	Bruger.
Aktører	Bruger (primær)
Referencer	UC1.
Antal af samtidige hændelser	1
Forudsætning	UC1 skal være gennemført.
Efterfølgende tilstand	Bruger er logget ud og log ind skærmen vises.
Hovedforløb	<ol style="list-style-type: none">Bruger trykker på log ud knap.Brugerinterfacet logger ud og viser log ind på brugerinterfacet.

TABEL 2 - UC2: LOG UD

3.4.3. UC3: Aktiver Alarm

Navn	UC3: Aktiver Alarm.
Mål	Bruger aktiverer ControllerHøjtaler alarm.
Initiering	Bruger.
Aktører	Bruger (primær).
Referencer	UC1.
Antal af samtidige hændelser	1
Forudsætning	UC1 skal være gennemført. Alarm er deaktivert.

Efterfølgende tilstand	Alarm er aktiveret.
Hovedforløb	<ol style="list-style-type: none"> 1. Bruger trykker på aktiveringsknappen i brugerinterface. 2. Brugerinterfacet viser en meddeelse om at Alarm er aktiveret. <p>[Ext. 2.a: Bruger logger ud af Brugerinterfacet]</p>
Extensions	<p>[Ext. 2.a: Bruger logger ud af Brugerinterfacet]</p> <ol style="list-style-type: none"> 1. Alarmen bliver stadig aktiveret.

TABEL 3 - UC3: AKTIVER SYSTEM

3.4.4. UC4: Deaktiver Alarm

Navn	UC4: Deaktiver Alarm.
Mål	Bruger deaktiverer ControllerHøjtaler alarm.
Initiering	Bruger.
Aktører	Bruger (primær).
Referencer	UC1.
Antal af samtidige hændelser	1
Forudsætning	UC3 er gennemført.
Efterfølgende tilstand	Alarm er deaktiveret.
Hovedforløb	<ol style="list-style-type: none"> 1. Bruger trykker på deaktiveringsknappen i brugerinterface. 2. Brugerinterfacet viser en meddeelse om at Alarm er deaktiveret.

TABEL 4 - UC4: DEAKTIVER SYSTEM

3.4.5. UC5: Udløs våben

Navn	UC5: Udløs våben.
Mål	Bruger udløser et skud fra våbnet.
Initiering	Bruger.
Aktører	Bruger (primær). Offer (sekundær).
Antal af samtidige hændelser	1
Referencer	UC1.
Forudsætning	UC1 er gennemført.
Efterfølgende tilstand	Våbnet er affyret og antal skud tilbage er talt ned.
Hovedforløb	<ol style="list-style-type: none">Bruger trykker på triggeren, for at affyre et skud. [Ext. 1.a: Våbnet er løbet tør for ammunition]Antal skud tilbage tælles ned.
Extensions	[Ext. 1.a: Våbnet er løbet tør for ammunition] <ol style="list-style-type: none">Brugerinterface giver meddelelse om, at våbnet er løbet tør for ammunition og affyrer ikke skud.Use case afsluttes.

TABEL 5 - UC5: UDLØS VÅBEN

3.4.6. UC6: Genlad

Navn	UC6: Genlad.
Mål	Våbnet er genladt.
Initiering	Bruger.
Aktører	Bruger (primær).
Referencer	UC1.
Antal af samtidige hændelser	1
Forudsætning	UC1 er gennemført.
Efterfølgende tilstand	Våbnet er genladt og System er aktiveret.
Hovedforløb	<ol style="list-style-type: none">1. Bruger trykker på "Genlad"-knappen i menuen.2. System deaktiverer automatisk. [Ext. 2a: Afbryd genladning]3. Bruger genlader våbnet manuelt.4. Bruger indtaster i brugerinterfacet hvor meget ammunition der er fyldt i våbnet og afslutter med Ok.5. System aktiverer automatisk.
Extensions	[Ext. 2a: Afbryd genladning] <ol style="list-style-type: none">1. Bruger vælger at annullere.2. System aktiverer automatisk.3. Use case afsluttes.

TABEL 6 - UC6: GENLAD

3.4.7. UC 7: Bevæg HDT horisontalt

Navn	UC7: Bevæg HDT horisontalt.
Mål	Bruger bevæger HDT til højre eller venstre.
Initiering	Bruger.
Altører	Bruger (primær).
Antal af samtidige hændelser	1
Refrencer	UC1.
Forudsætning	UC1 er gennemført.
Efterfølgende tilstand	HDT har bevæget sig horisontalt.
Hovedforløb	<ol style="list-style-type: none"> 1. Bruger bevæger joystick enten til højre eller venstre <ol style="list-style-type: none"> 1.a.1. Bruger bevæger joystick til venstre 1.a.2. System starter motor og styrer HDT til venstre <p>[Ext. 7.a: HDT kan ikke bevæges mere til venstre.]</p> <ol style="list-style-type: none"> 1.b.1. Bruger bevæger joystick til højre 1.b.2. System starter motor og styrer HDT til højre <p>[Ext. 7.b: HDT kan ikke bevæges mere til højre.]</p> 2. Bruger slipper joystick og system stopper motoren.
Extensions	<p>[Ext. 7.a: HDT kan ikke bevæges mere til venstre.]</p> <ol style="list-style-type: none"> 1. HDT rammer en dødswitch, og motoren stopper. 2. Use case afsluttes. <p>[Ext. 7.b: HDT kan ikke bevæges mere til højre.]</p> <ol style="list-style-type: none"> 1. HDT rammer en dødswitch, og motoren stopper. 2. Use case afsluttes.

TABEL 7 - UC7: BEVÆG HDT HORISONTALT

3.4.8. UC 8: Bevæg HDT vertikalt

Navn	UC8: Bevæg HDT vertikalt.
Mål	Bruger bevæger HDT op eller ned.
Initiering	Bruger.
Aktører	Bruger(primær).
Antal af samtidige hændelser	1
Referencer	UC1.
Forudsætning	UC1 er gennemført.
Efterfølgende tilstand	HDT har bevæget sig vertikalt
Hovedforløb	<ol style="list-style-type: none"> 1. Bruger bevæger joystick enten op eller ned <ol style="list-style-type: none"> 1.a.1. Bruger bevæger joystick op 1.a.2. System starter motor og styrer HDT op [Ext. 8.a: HDT kan ikke bevæges mere op.] 1.b.3. Bruger bevæger joystick ned 1.b.4. System starter motor og styrer HDT ned [Ext. 8.b: HDT kan ikke beværes mere ned.] 2. Bruger slipper joystick og HDT system stopper motoren.
Extensions	<p>[Ext. 8.a: HDT kan ikke bevæges mere op.]</p> <ol style="list-style-type: none"> 1. HDT rammer en dødswitch og motor stopper. 2. Use case afsluttes. <p>[Ext. 8.b: HDT kan ikke bevæges mere ned.]</p> <ol style="list-style-type: none"> 1. HDT rammer en dødswitch og motor stopper. 2. Use case afsluttes.

TABEL 8 - UC8: BEVÆG HDT VERTIKALT

3.4.9. UC9: Udløs alarm

Navn	UC9: Udløs alarm.
Mål	Sensor aktiverer alarmen.
Initiering	Sensor.
Aktører	Sensor(primær).
Referencer	UC3.
Antal af samtidige hændelser	1
Forudsætning	UC3 er gennemført. Ingen igangværende alarm.
Efterfølgende tilstand	Alarmen lyder.
Hovedforløb	<ol style="list-style-type: none">1. Sensor opfanger bevægelse.2. Brugerinterfacet viser en meddelelse om, at alarmen er gået.3. Alarmen lyder.

TABEL 9 - UC9: UDLØS ALARM

3.4.10. UC10: Advarsel

Navn	UC10: Advarsel.
Mål	Alarmen lyder
Initiering	Bruger
Aktører	Bruger(primær).
Referencer	UC1.
Antal af samtidige hændelser	1

Forudsætning	UC1 er gennemført.
Efterfølgende tilstand	Advarselstonen lyder.
Hovedforløb	<ol style="list-style-type: none">1. Bruger trykker på Advarsel knappen i menuen.2. Advarselstonen lyder.

TABEL 10 - UC10: ADVARSEL

4. Systemarkitektur [Alle]

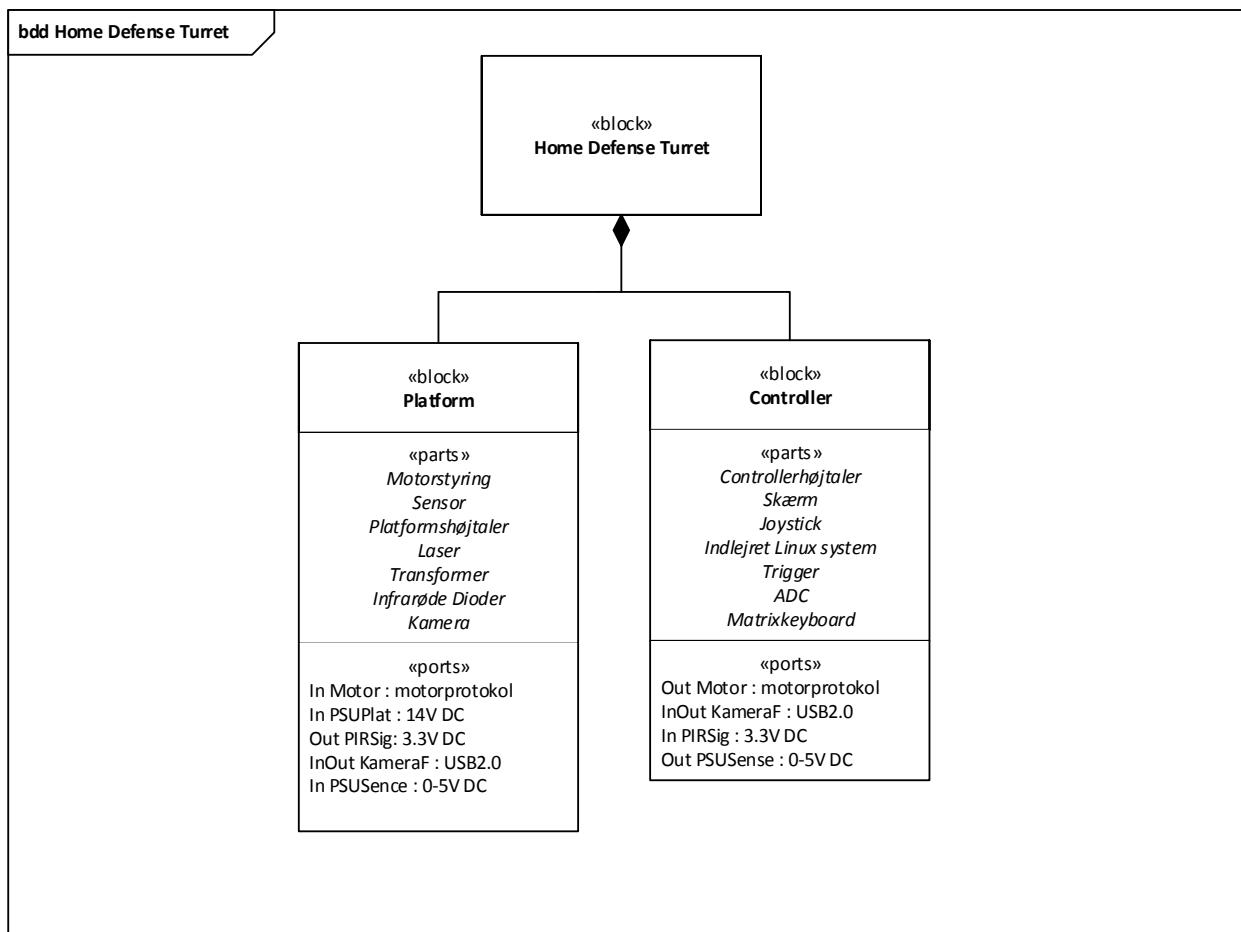
Følgende afsnit beskriver systemarkitekturen for "Home Defense Turret", som projektet er blevet beskrevet i kravspecifikationen og problemformuleringen.

Formålet med dette afsnit

- Identificere overordnede komponenter og fastlægge deres grænseflader
- identifikation og beskrivelse af eksterne komponenter, der anvendes i projektet
- identifikation af arbejdsopgaver for projektets desing- og implementeringsfase

4.1. Overordnet arkitektur af Home Defense Turret [SF, KB, LRA, ATT]

Det er valgt at dele Home Defense Turret op i to: Platform og Controller. Dettes skyldes, at selve opstillingen af projektet er delt op i to. Controller delen, er den del som Bruger integrerer med. Herfra kan Bruger styre pistolen op og ned, sigte samt skyde. Ude på Platform er pistolen opstillet. Her pistolen sat op, samt den hardware der skal til, for at styre den.



FIGUR 4 – BDD AF HOME DEFENSE TURRET

4.1.1. Blokbeskrivelse af Home Defense Turret

På baggrund af den overordnede systemarkitektur af Home Defense Turret, er der lavet en blokbeskrivelse

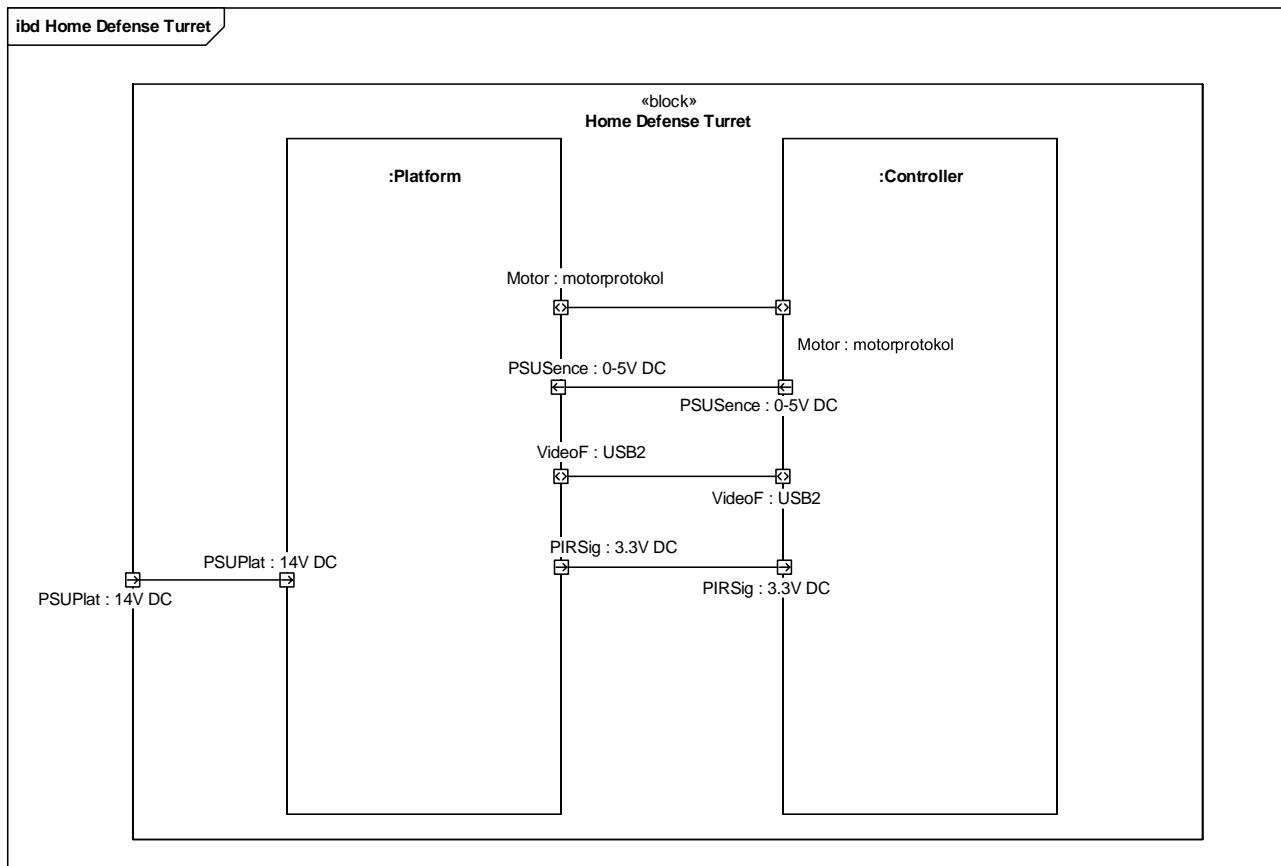
Bloknavn:	Systembeskrivelse:	Signal:	Kommentarer
Controller	Controller er den del af systemet der tillader Bruger at styre Platform ved brug af et joystik og et KameraFeed. Og en PIRsensor informerer om bevægelse.	Motor(motorprotokol)	Controllerens kommunikation med Motor styring + Forsyningsspænding.
		KameraF(USB2.0)	En videotransmission bliver sendt fra Platform til Controller via USB2.0.
		PIRSig(3.3V DC)	Et 3.3V DC signal går fra Platform til Controller når den opfanger bevægelse.

		PSUSence(0-5V DC)	Forsyningsspænding til PIRSensor
Platform	Platform er den overordnede konstruktion, hvor på pistolen er monteret. Der er tre aktuatorer, som gør at pistolen kan styres horisontalt, vertikalt og trykke på aftrækkeren. Der sidder et kamera, så man kan se sit mål og der sidder en laser så man kan sigte. Der sidder også en PIRsensor, der detekterer, om der er bevægelse i nærheden af HDT.	PSUPlat(14V DC)	Et 14V DC signal fra en strømforsyning går ind i Platform.
		PIRSig(3.3V DC)	Et 3,3V DC signal går fra Platform til Controller når den opfanger tilstrækkelig bevægelse.
		KameraF(USB2.0)	En videotransmission bliver sendt fra Platform til Controller via USB2.0.
		PSUSence(0-5V DC)	Forsyningsspænding til PIRsensor.

TABEL 10 - BLOKBESKRIVELSE AF HOME DEFENSE TURRET

4.1.2. Internal Block Diagram af Home Defense Turret

For at give et bedre overblik over grænsefladerne imellem Platform og Controller, er der udarbejdet et Internal Block Diagram af den overordnet system arkitektur



FIGUR 5 - IBD AF HDT

4.1.3. Signalbeskrivelse af HDT

I Internal Block Diagram af Home Defense Turret, er der defineret en række signaler. Disse er beskrevet i følgende tabel:

Bloknavn	Port	Signal	Type	Kommentar
Platform	In	LaserSig	5V DC	Signal der tænder laseren.
	In	Motor	motorprotokol	Control signaler fra Controller til motor. Dette signal indeholder også PSU til PsOC 4.
	InOut	KameraF	USB2.0	Signal der sendes igennem USBport, fra Kamera til controlleren for at opratte til et KameraFeed
	Out	PIRSig	3.3V DC	Signal der sendes, når sensoren opfanger et signal.
	In	PSUPlat	14V DC	Forsyningsspænding
	In	PSUSence	0-5V DC	Forsyningsspænding
Controller	Out	Motor	motorprotokol	Control signaler fra Controller til motor. Dette signal indeholder også PSU til PsOC 4.
	InOut	KameraF	USB2.0	Controller kommunikation med Kamera
	In	PIRSig	SPI	Signal modtages ved tilstrækkelig bevægelse
	Out	PSUSence	0-5V DC	Forsyningsspænding

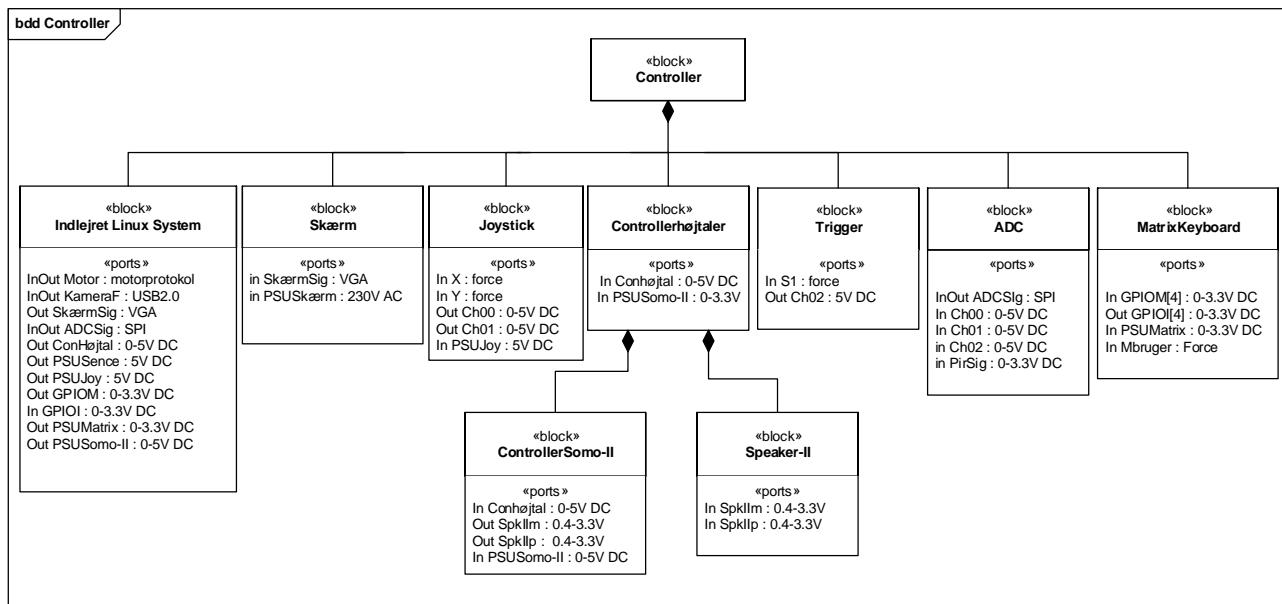
TABEL 11 - SIGNAL BESKRIVELSE AF HDT

5. Systemarkitektur af Controller [SF, KB, LRA, ATT]

5.1.1. Block Definition Diagram af Controller

Efter at have defineret Home Defense Turret, er systemet blevet yderligere nedbrudt. Følgende er et Block Definition Diagram af Controller. Formålet med Controller-blokken er at give Bruger mulighed for at kunne interagere med Platform. Bruger skal kunne styre pistolen op og ned, fra side til side, samt affyre pistolen. Herudover skal Bruger kunne logge ind i systemet, det vil sige Bruger skal have en form for tastatur, Bruger kan indtaste en kode, samt interagere med et brugerinterface. Der skal også kunne afspilles lyd og der skal kunne ses et KameraFeed for at opfylde use casene samt problemformuleringen.

Dette står til grund for, hvordan der er blevet valgt at opbygge Controlleren.



FIGUR 6 - CONTROLLER BDD

5.1.2. Blokbeskrivelse af Controller

Følgende tabel viser en blokbeskrivelse af Controller.

Bloknavn:	Systembeskrivelse:	Signal:	Kommentarer
Indlejret Linux System	Det Indlejrede Linux System er hovedenheden hvor alt kommunikation går igennem. Der er valgt en Raspberry Pi 2 Model B til dette formål.	Motor(motorprotokol)	Controllerens kommunikation med Motor styring + Forsyningsspænding.
		SkærmSig(VGA)	Visuelt video signal sendes fra Indlejret Linux System til Skærm.
		ADCSig(SPI)	Kommunikation vha spi bus i mellem ADC og Indlejret Linux System.
		ConHøjtal(0-5V DC)	Signal til ControllerSOMO-ii, der fortæller, om den skal afspille lyd.
		KameraF(USB2.0)	En videotransmission bliver sendt fra Kamera til Inlejret linux system.
		PSUSence(5V DC)	Forsyningsspænding til PIRsensoren.
		PSUJoy(5V DC)	Forsyningsspænding til Joystick
		GPIOI[4](0-3.3V DC)	0V fortolkes som Low, 3.3V fortolkes som High. "Spørger" om GPIO[4] er High eller Low
		GPIO[4](0-3.3V DC)	Er enten High, 3.3V eller Low 0V afhængig af om bruger har trykket på MatrixKeyboard
		PSUMatrix(0-3.3V DC)	Powersupply til MatrixKeyboard
		PSUSomo-II(0-5V DC)	Forsyningsspænding til ControllerSOMO-II
Joystick	Joystickket er bygget af to potentiometre. Når Bruger trykker joystickket	X (force)	Bruger kan bevæge joystickket til vertikalt vha tryk

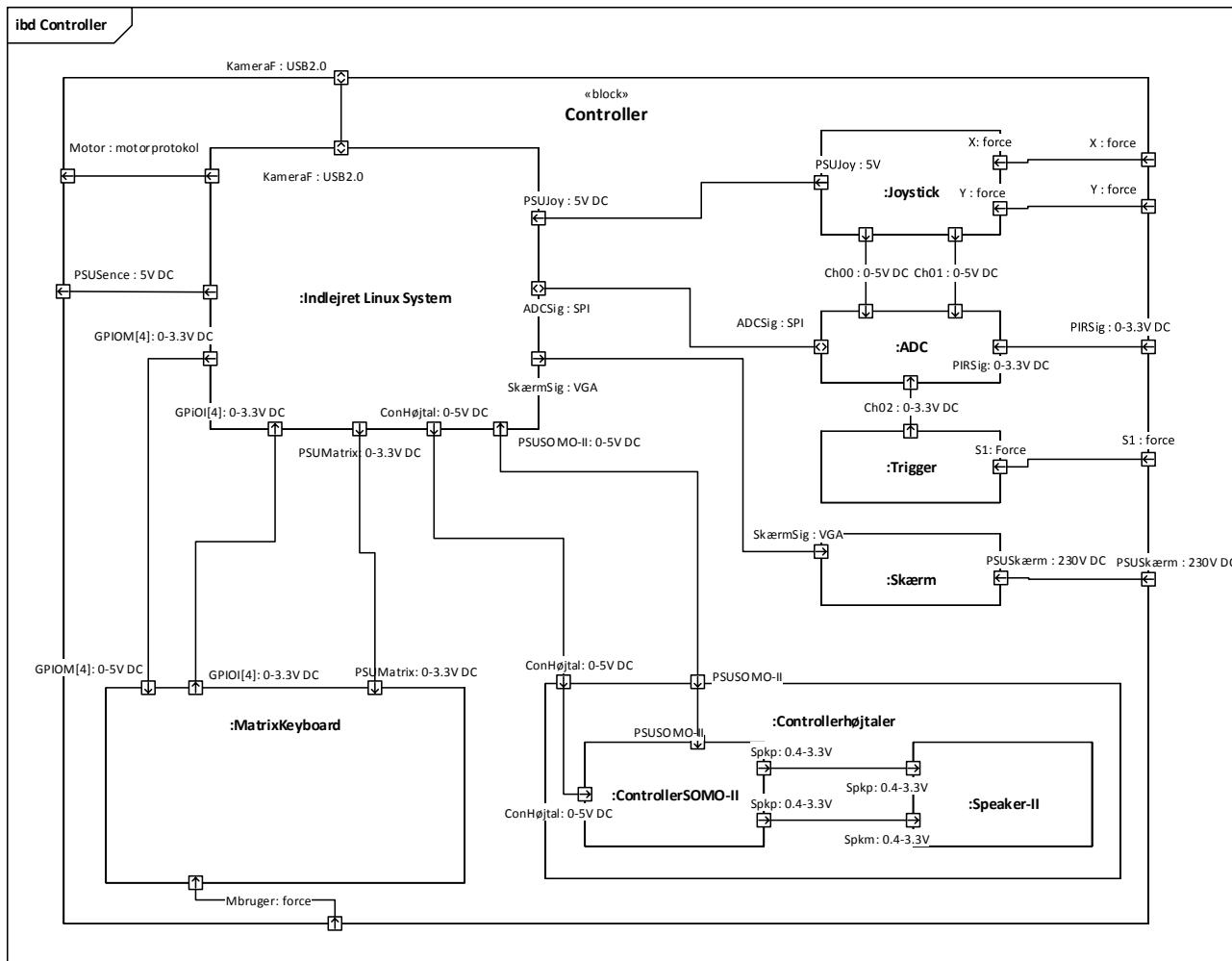
	horisontalt eller vertikalt, svarer det til at han skruer på et af potentiometrene og herved skruer op og ned for udgangs spændingen.	Y (force)	Bruger kan bevægte HDT til horisontalt vha tryk
		Ch00(0-5V)	Joystick sender et signal til ADC afhængigt af hvor meget Bruger trykker på joystickket vertikalt.
		Ch01(0-5V)	Joystick sender et signal til ADC afhængigt af hvor meget Bruger trykker på joystickket horisontalt.
ADC	ADC anvendes til at tolke signaler fra joystickket, Trigger og PIRsensoren digitalt for det Indlejret Linux System.	PSUJoy(5V)	PSU til joystick.
		ADCSig(SPI)	ADC kommunikerer med Indlejret Linux System vha SPI kommunikation
		Ch00(0-5V)	ADC modtager et signal fra joystickket afhængigt af, hvor meget Bruger trykker på joystickket vertikalt.
		Ch01(0-5V)	ADC modtager et signal fra joystickket afhængigt af, hvor meget Bruger trykker på joystickket horisontalt
		Ch02(0-3.3V)	Modtager en signal fra Trigger, når brugeren trykker for at skyde.
Trigger	Bruger anvender triggeren til at skyde, ved at trykke på den.	PIRSig(0-3.3V)	Modtager en signal fra PIRsensor, når PIRsensoren opfanger et signal
		S1(force)	Bruger trykker på triggeren.
		Ch02(0-3.3V)	Sender et signal til ADC på 3.3V, når der tryggers på Trigger.

Skærm	Bruger kan se KameraFeed og menu på skærmen.	PSUSkærm(230V AC)	230V AC Forsyningsspænding
		SkærmSig(VGA)	Visuelt video signal sendes fra Indlejret Linux System til Skærm.
Controllerhøjtalere	Samlet betegnelse for ControllerSOMO-II og Speaker-II	PSUSOMO-II Conhøjtal(0-5V DC)	Forsyningsspænding Kommunikation i mellem Indlejret Linux System og Controllerhøjtalere.
ControllerSOMO-II	Embedded sound module, der indeholder et micro SD kort, der indeholder en mp3-lydfil. Hvis ControllerSOMO-II modtager besked fra Indlejret Linux System om at afspille lyd, sender ControllerSOMO-II lyd ud af Speaker-II.	Conhøjtal(0-5V DC) SpkIIIm(0.4-3.3V) SpkIIP(0.4-3.3V) PSUSOMO-II(0-5V DC)	Signalet fortæller, om ControllerSOMO-II skal spille en melodi. Hvis 0, ja, hvis 5V, nej. Differential lavt PWM- signal, lydsignal, fra ControllerSOMO-II til Speaker-II Differential lavt PWM+ signal, lydsignal fra ControllerSOMO-II til Speaker-II Forsyningsspænding til ControllerSOMO-II
Speaker-II	Her afspilles lyden der stammer fra ControllerSOMO-II	SpkIIIm(0.4-3.3V) SpkIIP(0.4-3.3V)	Differential lavt PWM- signal, lydsignal, fra ControllerSOMO-II til Speaker-II Differential lavt PWM+ signal, lydsignal fra ControllerSOMO-II til Speaker-II

TABEL 12 - BLOKBESKRIVELSE AF CONTROLLER

5.1.3. Internal Block Diagram af Controller

For at give et overblik over grænsefladerne defineret i Block Definition Diagram af Controller, er der udarbejdet et Internal Block Diagram af Controller.



FIGUR 7 - IBD AF CONTROLLER

5.1.4. Signalbeskrivelse af Controller

På baggrund af Internal Block Diagram af Controller, er der udarbejdet en signalbeskrivelse, der beskriver grænsefladerne.

Bloknavn	Port	Signalnavn	Type	Kommentar
Indlejret Linux System	Out	Motor	motorprotokol	Bestemt protokol der sender over UART til motoren afhængigt af, hvordan platformen skal bevæges, foruden forsyningsspænding.
	Out	PSUJoy	0-5V DC	Forsyningsspænding til joystick.
	InOut	KameraF	USB2.0	En videotransmission bliver sendt fra Kamera til Inlejret linux system.
	Out	SkærmSig	VGA	Signal der tilslutter skærmen, så der opstår et brugerinterface for brugeren.
	InOut	ADCSig	SPI	Kommunikation i mellem inlejret linux system og ADC vha SPI bus.
	Out	Conhøjtal	0-5V DC	Indlejret Linux System sender besked til Controllerhøjtaler om, at den skal afspille en lyd.
	In	PIRSig	0-5V DC	Signal opstår fra PIRsensoren til controlleren ved tilstrækkelig bevægelse.
	Out	PSUSense	5V DC	Forsyningsspænding til PIRsensor.
	In	GPIOI[4]	0-3.3V DC	Der er 4*GPIO på Indlejret Linux System, der modtager hvert sit signal. Det er enten Low, 0V, eller High, 3.3V, hvilket afhænger af, om brugeren har trykket på MatrixKeyboard.
	Out	GPIOM[4]	0-3.3V DC	Indlejret Linux System har fire porte med fire signaler, der er High, 3.3V eller Low, 0V, for at tjekke, om der er trykket på MatrixKeyboard.

	Out	PSUMatrix	0-3.3V DC	Forsyningsspænding til MatrixKeyboard
	Out	PSUSOMO-II	0-5V DC	Forsyningsspænding til ControllerSOMO-II
Joystick	In	X	Force	Bruger trykker på joystickket for at bevæge HDT vertikalt
	Out	Ch01	0-5V DC	Signal til ADC, der afhænger af, hvor hårdt bruger trykker for at bevæge HDT vertikalt.
	In	Y	Force	Bruger trykker på joystickket for at bevæge HDT horisontalt.
	In	PSUJoy	0-5V DC	Forsyningsspænding til Joystick.
	Out	Ch00	0-5V DC	Signal til ADC, der afhænger af, hvor hårdt bruger trykker for at bevæge HDT vertikalt.
	In	SkærmSig	VGA	Videotransmition bliver vist på skærmen fra Kamera.
Controllerhøjtalere	In	Conhøjtal	0-5V DC	Signal der fortæller, om der skal afspilles en lyd.
	In	PSUSOMO-II	0-5V DC	Forsyningsspænding til ControllerSOMO-II.
ControllerSOMO-II	In	Conhøjtal	0-5V DC	Når ControllerSOMO-II modtager 0V i et interval på 200ms, sender den en melodi ud af Speaker-II. Når signalet går på 5V, afspiller den ikke en melodi.
	In	PSUSOMO-II	0-5V DC	Forsyningsspænding til ControllerSOMO-II.
	In	Ch00	0-5V DC	Signal til ADC, der afhænger af, hvor hårdt bruger trykker for at bevæge HDT vertikalt.
ADC	In	Ch01	0-5V DC	Signal til ADC, der afhænger af, hvor hårdt bruger trykker for at bevæge HDT horisontalt.
	In	Ch02	0-5V DC	Signal til ADC, der påføres, når bruger har trykket på Trigger.

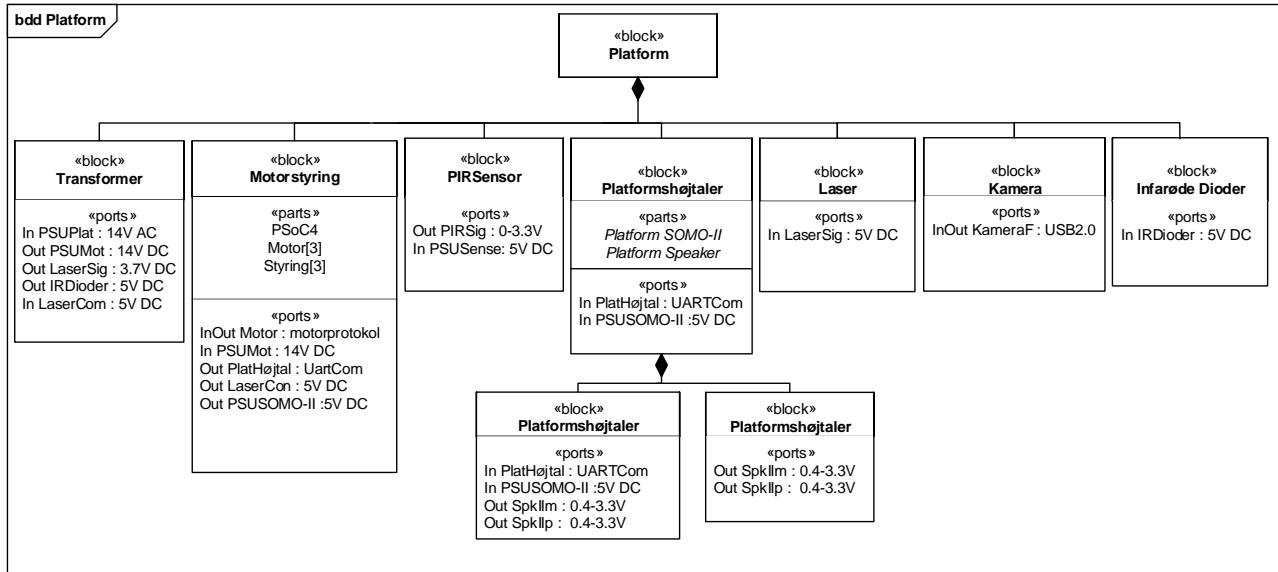
	In	PIRsig	0-5V DC	Signal opfanges når PIRsensoren detekterer bevægelse.
	Out	ADCsig	SPI	Kommunikation mellem Indlejret Linux System og ADC.
Trigger	In	S1	Force	Bruger trykker på skyd.
	Out	Ch02	0-5V DC	Trigger sender signal videre til ADC, hvis brugeren har trykket for at skyde.
MatrixKeyboard	In	GPIO[4]	0-3.3V DC	Indlejret Linux System har fire porte med fire signaler, der er High, 3.3V eller Low, 0V, for at tjekke, om der er trykket på MatrixKeyboardet.
	Out	GPIO[4]	0-3.3V DC	Der er 4*GPIO med hvert sit signal på Indlejret Linux System. De er enten Low, 0V, eller High, 3.3V, hvilket afhænger af, om brugeren har trykket på dem.
	In	PSUMatrix	0-3.3V DC	Forsyningsspænding til MatrixKeyboardet
	In	BrugerM	Force	Bruger trykker på MatrixKeyboardet
	In	SpkIIm	0.4-3.3V	Differentiel PWM+ signal der får Speakeren til at afspille en lyd.
Speaker-II	In	SpkIIP	0.4-3.3V	Differentiel PWM- signal der får speakeren til at afspille en lyd.

TABEL 13 - SIGNALBESKRIVELSE AF CONTROLLER

6. Systemarkitektur af Platform [SF, KB, LRA, ATT]

6.1.1. Block Definition Diagram af Platform

Følgende Block Definition Diagram beskriver opbygningen af Platform. Platforms formål er, at den skal indeholde komponenter der gør, at pistolen kan styres op og ned, man skal kunne se, hvor pistolen peget hen, en sensor skal kunne opfange bevægelse samt der skal være en højttaler, som afspiller en lyd, hvis Bruger ønsker dette. Herfra er der udarbejdet følgende Block Definition Diagram.



FIGUR 8 - PLATFORM BDD

6.1.2. Blokbeskrivelse af Platform

Udfra ovenstående Block Definition Diagram, er der udarbejdet en blokbeskrivelse af Platform

Bloknavn	Systembeskrivelse	Signal	Kommentarer
Transformer	En 14V DC transformer der forsyner platformen, og regulerer spændings-niveauer.	PSUPlat(14V DC)	Signal fra elnettet til Transformer.
		PSUMot(14V DC)	Forsyningsspænding til motor.
		IRDioder(5V DC)	Forsyningsspænding til Infrarøde Dioder.
		LaserSig(3.7V DC)	Forsyningsspænding til Laser.
		LaserCom(5V DC)	Kontrol signal til Laser.
Motorstyring	Modtager signal fra controller, styrer tre aktuatorer, en laser og en SOMO.	Motor(motorprotokol)	Controllerens kommunikation med motor og styring samt forsyningsspænding til PsOC4.
		PSUMot(14V DC)	Forsyningsspænding til motor.

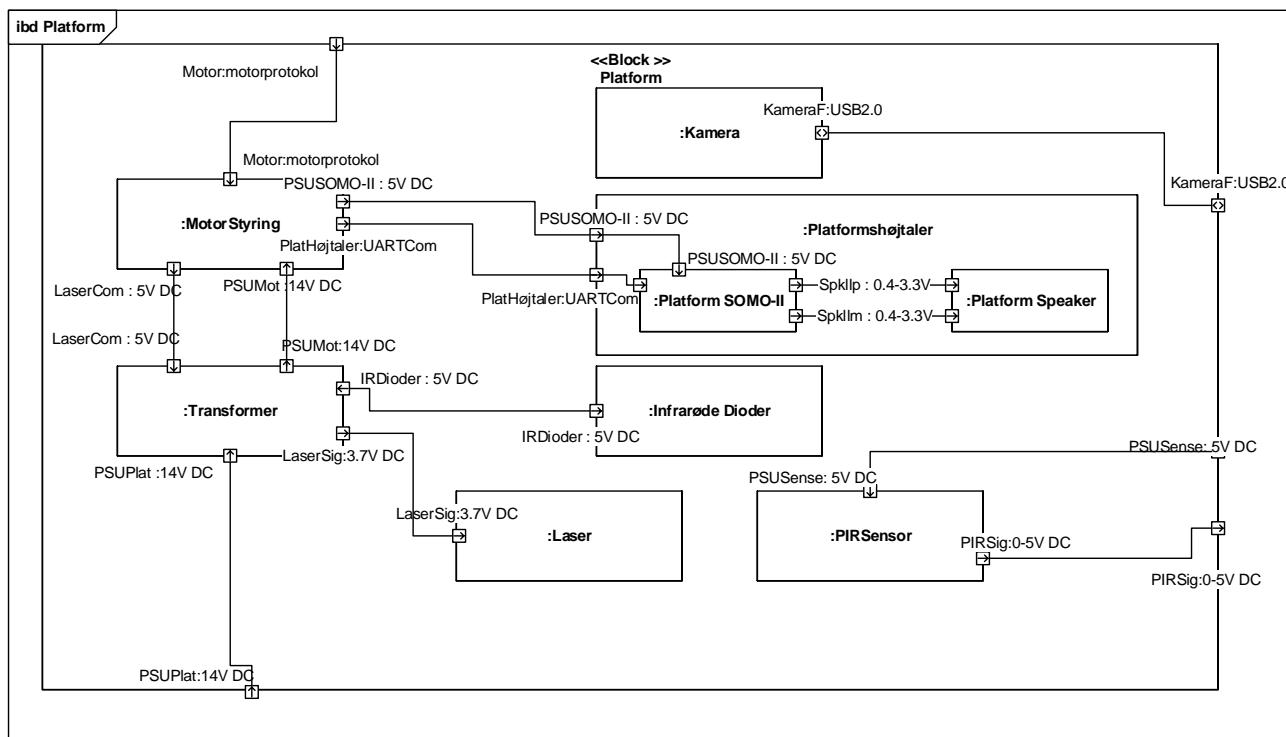
		PlatHøjtal(UARTCom)	UART kommunikation til at aktivere Platform Højtaler.
		LaserCom(5V DC)	Aktivering signal til Laser.
		PSUSOMO-II(5V DC)	Forsyningsspænding til SOMO-II.
PIRSensor	PIRsensor detekterer bevægelse og informerer Indlejret Linux System om dette via en ADC på controllerblokken.	PIRSig(0-3.3V)	Kommunikation imellem PIRsensor og Indlejret Linux System.
		PSUSence(5V DC)	Forsyningsspænding til sensorer.
Platformshøjtaler	Afspiller en lyd, når den får besked på dette.	Plathøjtal(UARTCom)	Advarelseskmando, når Bruger vælger at sende alarm.
		PSUSOMO-II(5V DC)	Forsyningsspænding til SOMO-II.
PlatformSOMO-II	Embedded sound module, der indeholder et micro SD kort, der indeholder en mp3-lydfil. Hvis PlatformSOMO-II modtager besked fra Motor Styring om at afspille lyd, sender PlatformSOMO-II lyd ud af Speaker-II.	PlathøjtalerUARTCom	UART sender 8 byte signal til PlatformSOMO-II.
		Spkllm(0.4-3.3V)	Differential lavt PWM-signal, lydsignal, fra PlatformSOMO-II til Speaker-II.
		Spkllp(0.4-3.3V)	Differential lavt PWM+ signal, lydsignal fra PlatformSOMO-II til Platform Speaker.
		PSUSOMO-II(0-5V DC)	Forsyningsspænding til PlatformSOMO-II.
Platform Speaker	Her afspilles lyden der stammer fra PlatformSOMO-II.	Spkllm(0.4-3.3V)	Differential lavt PWM-signal, lydsignal, fra PlatformSOMO-II til Platform Speaker.
		Spkllp(0.4-3.3V)	Differential lavt PWM+ signal, lydsignal fra PlatformSOMO-II til Platform Speaker.
Laser	Laseren lyser når den får forsyningsspænding.	LaserSig(5V DC)	Forsyningsspænding til Laser.
Kamera	Laver et KameraFeed, så brugeren kan se, hvor HDT peger hen.	KameraF(USB2.0)	KameraFeed fra Kamera.
Infrarøde Dioder	Infarød lys, hvis formål er at bremse motorene.	IRDioder(5V DC)	Infrarøde Dioder som lyser for infared

			modtager motorstyring.	i
--	--	--	---------------------------	---

TABEL 14 - BLOKBESKRIVELSE AF PLATFORM

6.1.3. Internal block definition diagram af Platform

For at give et bedre overblik over grænsefladerne i Platform, er der udarbejdet et Internal Block Diagram af Platform. Efterfølgende er der også udarbejdet en signalbeskrivelse over de enkelte interne grænseflader.



FIGUR 9 - IBD AF PLATFORM

6.1.4. Signalbeskrivelse af Platform

Bloknavn	Port	Signal	Type	Kommentar
MotorStyring	In	Motor	motorprotokol	Controllerens kommunikation med Motor Styring + Forsyningsspænding
	In	PSUMot	14V DC	Forsyningsspænding
	Out	PlatHøjtaler	UARTCom	Advareseskommando, når Bruger vælger at sende alarm.
	Out	LaserCom	5V DC	Aktivering signal til Laser.
	Out	PSUSOMO-II	5V DC	Forsyningsspænding til SOMO-II.
Transformer	In	PSUPlat	14V DC	Forsyningsspænding.
	Out	PSUMot	14V DC	Forsyningsspænding.
	Out	LaserSig	3.7V DC	Forsyningsspænding til Laser.
	Out	IRDioder	5V DC	Forsyningsspænding til Infrarøde Dioder.
	In	LaserCom	5V DC	Kontrol signal til Laser.
Laser	In	LaserSig	3.7V DC	Forsyningsspænding til Laser.
PIRSensor	In	PSUSense	0-5V DC	Forsyningsspænding til PIRSensor.
	Out	PIRSig	0-3.3V	Signal bliver 3.3V, når PIRSensoren detekterer bevægelse.
Platformshøjtaler	In	Plathøjtaler	UARTCom	Advareseskommando, når Bruger vælger at sende alarm.
	In	PSUSOMO-II	5V DC	Forsyningsspænding til SOMO-II.
Platform SOMO-II	In	Plathøjtaler	UARTCom	Advareseskommando, når Bruger vælger at sende alarm.
	In	PSUSOMO-II	5V DC	Forsyningsspænding til SOMO-II.
	Out	Spkllm	0.4-3.3V	Differential lavt PWM-signal, lydsignal, fra PlatformSOMO-II til Platform Speaker
	Out	Spkllp	0.4-3.3V	Differential lavt PWM-signal, lydsignal, fra

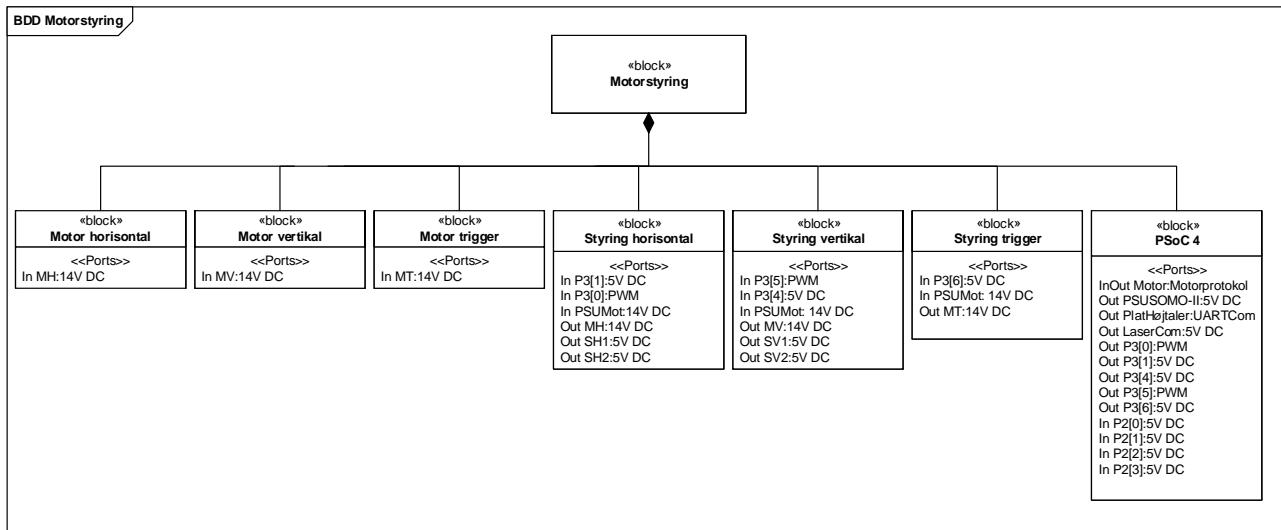
				PlatformSOMO-II til Platform Speaker
Platform speaker	In	Spkllm	0.4-3.3V	Differential lavt PWM-signal, lydsignal, fra PlatformSOMO-II til Platform Speaker
	In	Spkllp	0.4-3.3V	Differential lavt PWM-signal, lydsignal, fra PlatformSOMO-II til Platform Speaker.
Kamera	InOut	KameraF	USB2.0	KameraFeed som sendes til Controlleren.
Infrarøde Dioder	In	IRDioder	5V DC	Infrarøde Dioder som lyser for infared modtager i motorstyring.

TABEL 15 - SIGNALBESKRIVELSE AF PLATFORM

7. Systemarkitektur af Motorstyring [SF, KB, LRA, ATT]

Motorstyring skal bestå af en række komponenter for at opfylde sine formål. Formålet med motorstyring er at kunne styre motorene op og ned, fra side til side, samt at kunne affyre pistolens aftrækker. Udfra disse betegnelser er følgende Block Definition Diagram blevet udarbejdet.

Block Definition diagram af Motorstyring



FIGUR 10 - BDD AF MOTORSTRYRING

7.1.1. Blokbeskrivelse af Motorstyring

Bloknavn:	Systembeskrivelse:	Signal:	Kommentarer
Styring horisontal	Styrer om pistolen skal køre til højre eller venstre. Samt hastigheden den bevæger sig med. Den indeholder infarød stopsensor som sender stopsignal til PSoC 4 når den modtager infrarød lys.	P3[1](5V)	Signal fra PSoC 4 til H-bro Styrer retning.
		P3[0](PWM)	S Signal fra PSoC 4 til H-bro Styrer hastighed.
		PSUMot(14VDC)	Forsyningsspænding til motorbelastning.
		MH(14V)	Motorbelastning.
		SH1(5V)	Afbrydnings signal som afbryder når motoren bevæger sig for langt til højre.
		SH2(5V)	Afbrydnings signal som afbryder når motoren bevæger sig for langt til venstre.
Styring vertikalt	Styrer om pistolen skal køre til op eller ned. Samt	P3[5](PWM)	Signal fra PSoC 4 til H-bro Styrer hastighed.

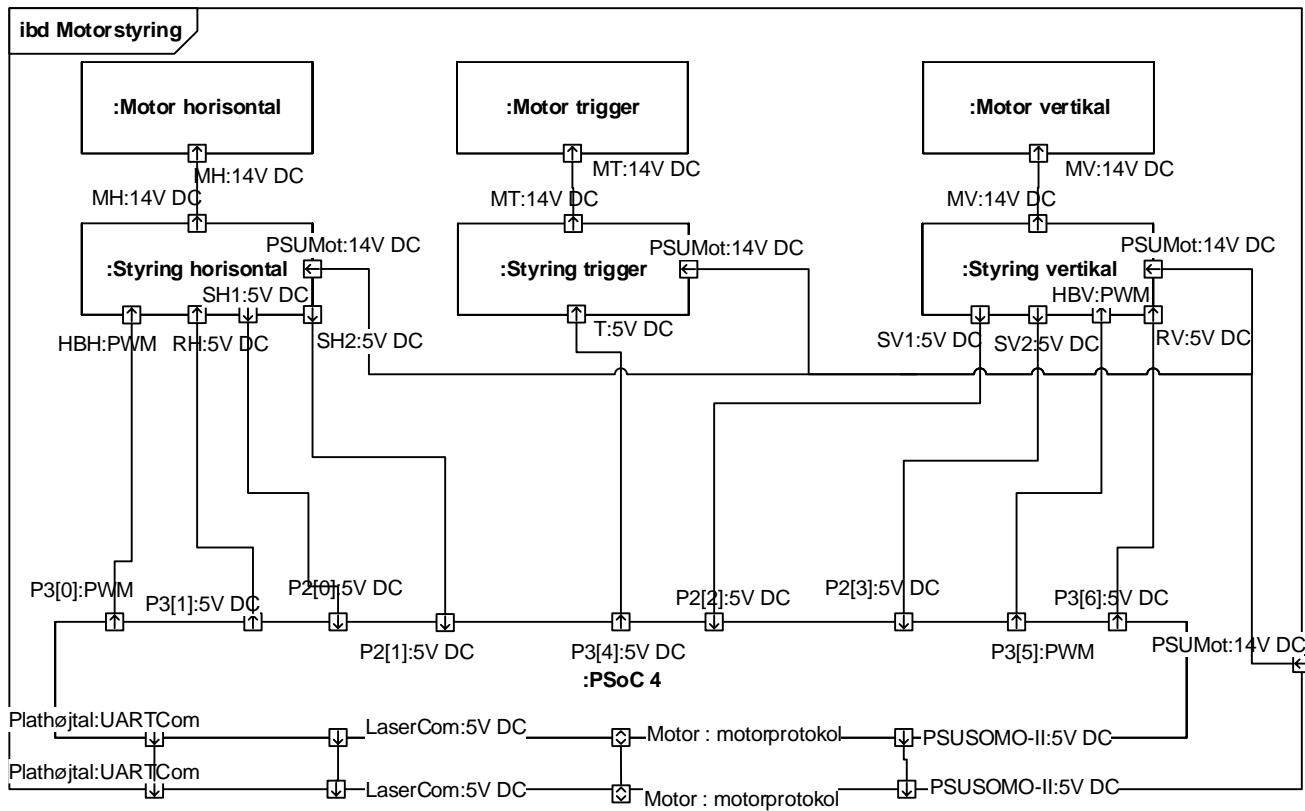
	hastigheden den bevæger sig med. Den indeholder infarød stopsensor som sender stopsignal til PSoC 4 når den modtager infrarød lys.	P3[4](5V)	Signal fra PSoC 4 til H-bro Styrer retning.
		PSUMot(14VDC)	Forsyningsspænding til motorbalastning.
		MV(14V)	Motorbalastning.
		SV1(5V)	Afbrydnings signal som afbryder når motoren bevæger sig for langt op.
		SV2(5V)	Afbrydnings signal som afbryder når motoren bevæger sig for langt ned.
Styring trigger	Styrer hvornår pistolen skal skyde.	P3[6](5V)	Signal fra PSoC 4 til relæ styrer affyring.
		PSUMot(14VDC)	Forsyningsspænding til motorbalastning.
		MT(14V)	Motorbalastning.
PSoC 4	Modtager signal fra indlejret linus system, og sender signaler ud til Styring horisontal, Styring vertikal og Styring trigger.	Motor(motorprotokol)	Signal fra Indlejret Linux System til .
		P3[0](PWM)	Signal fra PSoC 4 til H-bro Styrer hastighed.
		P3[1](5V)	Signal fra PSoC 4 til H-bro Styrer retning.
		P3[4](5V)	Signal fra PSoC 4 til H-bro Styrer retning.
		P3[5](PWM)	Signal fra PSoC 4 til H-bro Styrer hastighed.
		P3[6](5V)	Signal fra PSoC 4 til relæ styrer affyring.
		P2[0](5V)	Afbrydnings signal som afbryder når motoren bevæger sig for langt til højre.
		P2[1](5V)	Afbrydnings signal som afbryder når motoren bevæger sig for langt til venstre.
		P2[2](5V)	Afbrydnings signal som afbryder når motoren bevæger sig for langt op.
		P2[3](5V)	Afbrydnings signal som afbryder når motoren bevæger sig for langt ned.

		Plathøjtal(UARTCom)	Advarelseskommando, når Bruger vælger at sende alarm.
		LaserCom(5V DC)	Aktivering signal til Laser.
		PSUSOMO-II(5V DC)	Forsyningsspænding til SOMO-II.
Motor horisontal	Motoren bevæger sig til højre eller venstre alt efter modtaget signal.	MH(14V)	Motorbelastning.
Motor vertikal	Motoren bevæger sig til højre eller venstre alt efter modtaget signal.	MV(14V)	Motorbelastning.
Motor trigger	Motoren bevæger sig og affyreTrigger.	MT(14V)	Motorbelastning

TABEL 16 - BLOKBESKRIVELSE AF MOTORSTYRING

7.1.2. Internal block definition diagram af Motorstyring

Der er udarbejdet et Internal Block Diagram af Motorstyring for at give et overblik over de interne grænsefladerne i Motorstyring. En signalbeskrivelse kan findes efterfølgende af følgende diagram.



FIGUR 11 - BDD AF MOTORSTRYING

7.1.3. Signalbeskrivelse af Platform

Bloknavn	Port	Signal	Type	Kommentar
PSoC 4	In	Motor	motorprotokol	Controllerens kommunikation med Motor Styring + Forsyningsspænding.
	Out	Plathøjtal	UARTCom	Advarelseskommando, når Bruger vælger at sende alarm.
	Out	LaserCom	5V DC	Aktivering signal til Laser.
	Out	PSUSOMO-II	5V DC	Forsyningsspænding til SOMO-II.
	Out	P3[0]	PWM	Motorhastigheds kontrolsignal til horisontal bevægelse.
	Out	P3[1]	5V DC	Motoretnings kontrolsignal til horisontal bevægelse.
	Out	P3[4]	5V DC	Motor kontrolsignal til trigger bevægelse.
	Out	P3[5]	PWM	Motorhastigheds kontrol signal til vertikal bevægelse.
	Out	P3[6]	5V DC	Motoretnings kontrolsignal til horisontal bevægelse.
	In	P2[0]	5V DC	Afbrydnings signal som afbryder når motoren bevæger sig for langt til højre.
	In	P2[1]	5V DC	Afbrydnings signal som afbryder når motoren bevæger sig for langt til venstre.
	In	P2[2]	5V DC	Afbrydnings signal som afbryder når motoren bevæger sig for langt op.
	In	P2[3]	5V DC	Afbrydnings signal som afbryder når motoren bevæger sig for langt ned.
Styring horisontal	In	HBH	PWM	Motorhastigheds kontrolsignal til horisontal bevægelse.
	In	RH	5V DC	Motoretnings kontrolsignal til horisontal bevægelse.
	In	PSUMot	14V DC	Forsyningsspænding til motorbalastning.

	Out	MH	14V DC	Motorbelastning.
	Out	SH1	5V DC	Afbrydnings signal som afbryder når motoren bevæger sig for langt til højre.
	Out	SH2	5V DC	Afbrydnings signal som afbryder når motoren bevæger sig for langt til venstre.
Styring vertikal	In	HBV	PWM	Motorhastigheds kontrolsignal til vertikal bevægelse.
	In	RV	5V DC	Motoretnings kontrolsignal til vertikal bevægelse.
	In	PSUMot	14V DC	Forsyningsspænding til motorbalastning.
	Out	MV	14V DC	Motorbelastning.
	Out	SV1	5V DC	Afbrydnings signal som afbryder når motoren bevæger sig for langt op.
	Out	SV2	5V DC	Afbrydnings signal som afbryder når motoren bevæger sig for langt ned.
	In	T	5V DC	Motor kontrolsignal til trigger bevægelse.
Styring trigger	In	PSUMot	14V DC	Forsyningsspænding til motorbalastning.
	Out	MT	14V DC	Motorbelastning.
	In	MH	14V DC	Motorbelastning.
Motor horizontal	In	MV	14V DC	Motorbelastning.
Motor vertical	In	MT	14V DC	Motorbelastning.
Motor trigger	In	MT	14V DC	Motorbelastning.

TABEL 17 - SIGNALBESKRIVELSE AF MOTORSTYRING

7.2. Ordliste

MH	Motor horisontalt
MV	Motor vertikalt
MT	Motor trigger
PWM	Pulse Width Modulation
PSU	Power Source Unit
HDT	Home Defense Turret
BDD	Block definition diagram
IBD	Internal block definition diagram
SPI	Serial Peripheral Interface

8. Softwarearkitektur [JDA, DT, AEL]

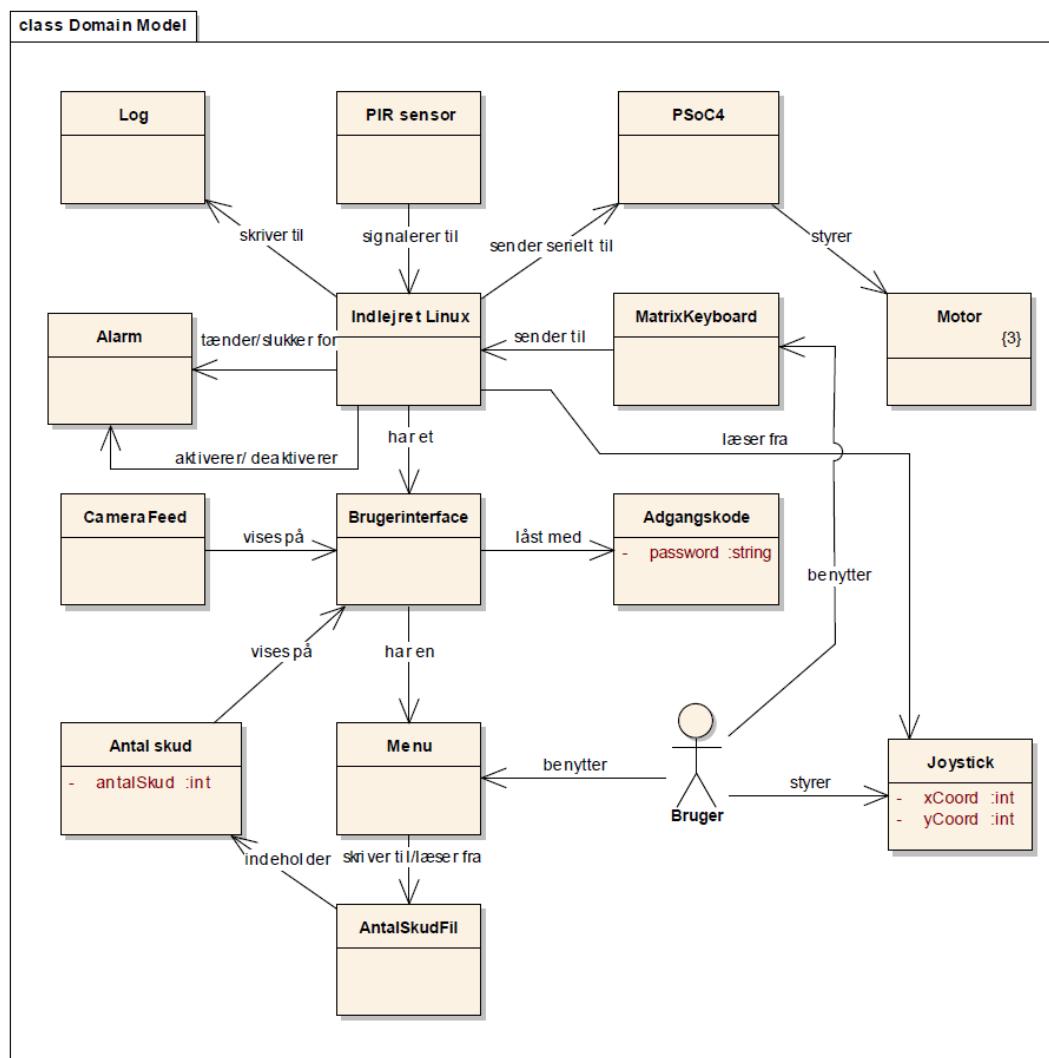
Dette afsnit beskriver den grundlæggende arkitektur for softwaren til dette system. Der vil være en forklaring af softwareens hovedblokke i systemet, samt overordnede sekvensdiagrammer.

Efterfølgende vil der gås i dybden med et mere detaljeret design af softwaren, så der kan fås et indblik i, hvordan softwaren fungerer, og hvordan systemets moduler kommunikerer med hinanden.

De overordnede sekvensdiagrammer giver et hurtigt overblik over, hvad der sker mellem system og bruger, hvor de detaljerede sekvensdiagrammer vil give et større indblik i, hvad der sker "behind-the-scenes".

8.1. Domænemodel

Der er udarbejdet en overordnet domænemodel der illustrerer systemets funktionalitet og software struktur. Den viser forbindelserne mellem delelementerne og er udarbejdet ud fra Use Casene, og på baggrund af den øvrige systemarkitektur.



FIGUR 12 - DOMÆNEMODEL FOR OVERORDNET SYSTEM

8.2. Beskrivelse af softwarens hovedmoduler

8.2.1. Brugerinterface

Brugerinterfacet vil være implementeret på et Indlejret Linux System. *Brugerinterfacet* er det primære interface mellem bruger og systemet. Brugeren af systemet kan tilgå flere menuer i brugerinterfacet, herunder en login-menu, som er det første brugeren møder i programforløbet. Systemet vil være beskyttet af en PIN-kode, som indtastes på et matrixkeyboard. Brugeren vil i brugerfladen blive præsenteret for fem muligheder:

1. Aktiver system
2. Deaktiver system
3. Genlad
4. Advarsel
5. Log ud

I hovedmenuen ses desuden et kamerafeed, der med levende billeder gengiver udsigten fra HDT's placering.

Knappen "Advarsel" giver brugeren mulighed for at udsende en advarselslyd fra HDT's højtalere.

8.2.2. Indlejret Linux System

Det Indlejrede Linux System er kernen i projektet, og skal stå for at læse fra en PIR sensor, et matrix keyboard og et joystick. *Brugerinterfacet* er implementeret på det Indlejrede Linux System. Ydermere vil dette system have til opgave at føre en log, som beskriver events, som skud, aktivering og deaktivering af HDT. Antal skud i systemet læses fra en fil herpå. Det Indlejrede Linux System har derudover også ansvaret for kommunikation med PSoC4.

8.2.3. Matrix keyboard

Al brugerinteraktion der foregår gennem *brugerinterfacet* vil ske via et matrix keyboard, der er tilkoblet det Indlejrede Linux System. Matrix keyboardet sender data med information om hvilken knap på tastaturet der er trykket.

8.2.4. PIR sensor

En PIR sensor vil være tilkoblet det Indlejrede Linux System. Denne har til formål at detektere bevægelse indenfor HDT's rækkevidde. Det Indlejrede Linux System læser fra PIR sensoren og behandler dataen. Når der detekteres bevægelse vil det Indlejrede Linux System aktivere en alarm.

8.2.5. Joystick

På det Indlejrede Linux System vil der være tilkoblet et analogt joystick, hvorigennem brugeren har mulighed for at styre HDT. Joysticket sender data til det Indlejrede Linux System, som behandler dataen, og videresender den til PSoC4 som styrer de elektriske motorer.

8.2.6. Camerafeed

Et kamera er fastsat på HDT, der gengiver udsigten fra HDT's placering. Dette giver mulighed for at brugeren kan sigte efter målet med HDT. Camerafeedet vises i *brugerinterfacet*.

8.2.7. Log

En klasse der skriver til en tekstfil når det Indlejrede Linux System udfører særlige handlinger.

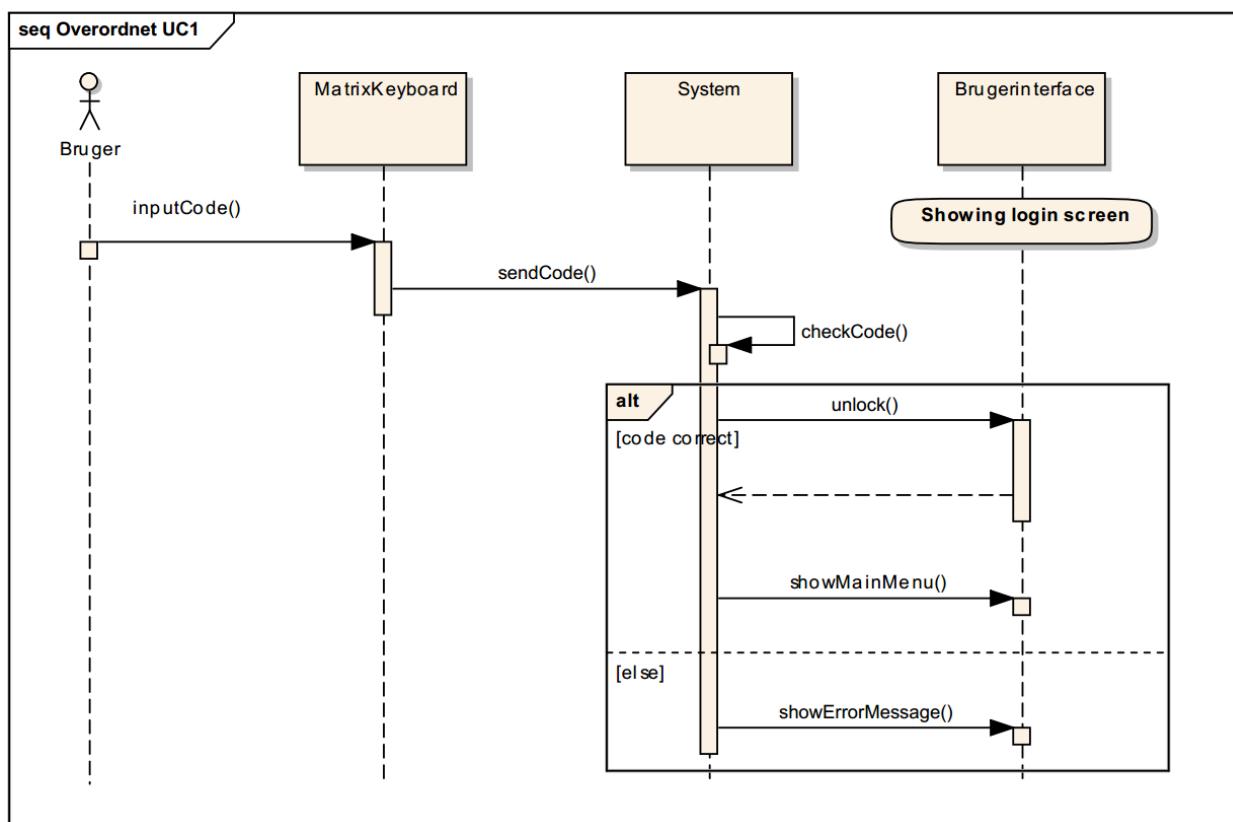
8.2.8. AntalSkudFil

En tekstfil som indeholder det antal skud der var i våbnet ved nedlukning af programmet og som det Indlejrede Linux System kan bruge ved opstart.

8.3. Sekvensdiagrammer

For at skabe et overblik over den programmæssige funktionalitet er der lavet sekvensdiagrammer for alle Use Cases. Sekvensdiagrammerne er meget overordnede, og beskriver mest den overordnede interaktion mellem bruger og grænseflader. Derudover er der også forsøgt at definere en grov skitse af programflowet, og hvornår de forskellige events forekommer og i hvilken rækkefølge.

De forskellige sekvenser i diagrammet er som udgangspunkt de moduler vi har fundet frem til i den overordnede domænemodel¹.

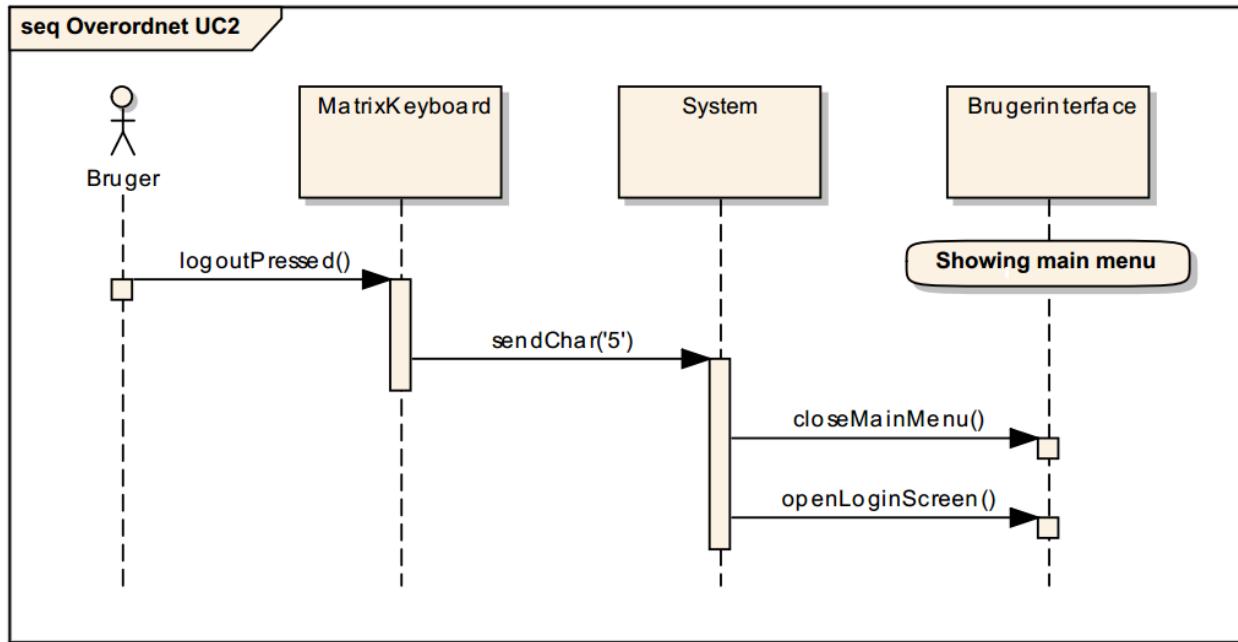


FIGUR 13 - SEKVENSDIAGRAM FOR USE CASE 1

Figur 13 viser funktionaliteten der udføres, når Use Case 1 (Log ind) gennemføres. Matrix keyboardet, det Indlejrede Linux System (system) og brugerinterfacet vil blive anvendt. Idéen er, at brugeren trykker på

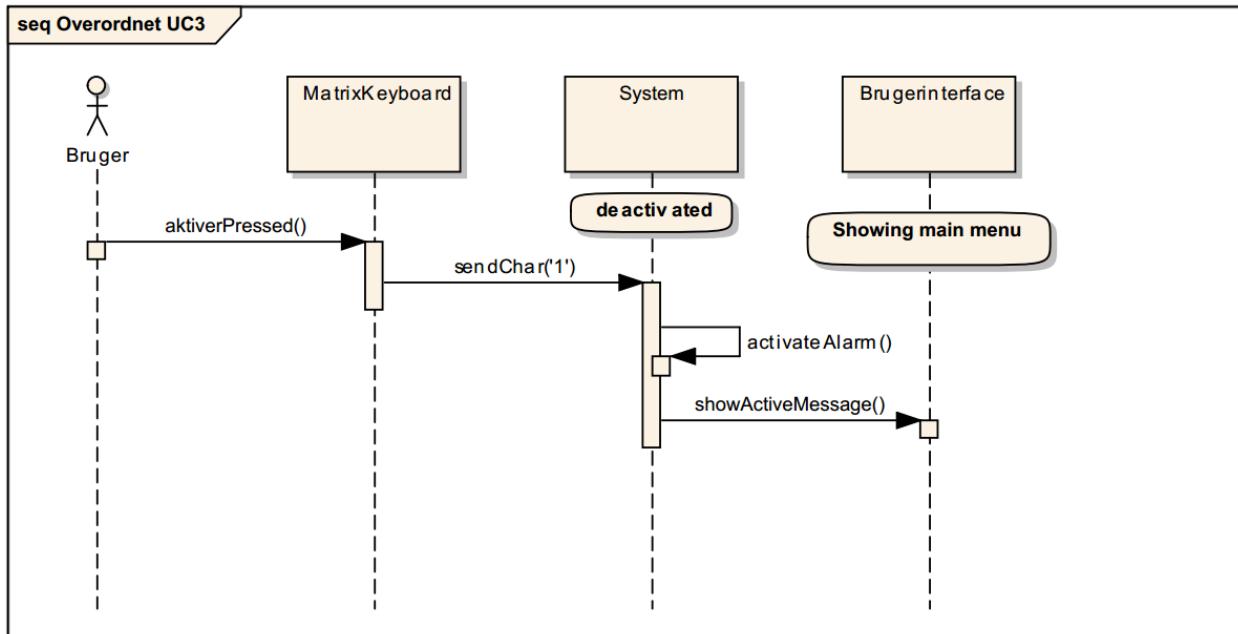
¹ Se domænemodel, afsnit 8.1

matrix keyboardet, som sender det registrerede input til system, som tjekker koden, og derefter låser op eller viser en "Din kode er forkert"-besked, alt efter hvad der blev skrevet.



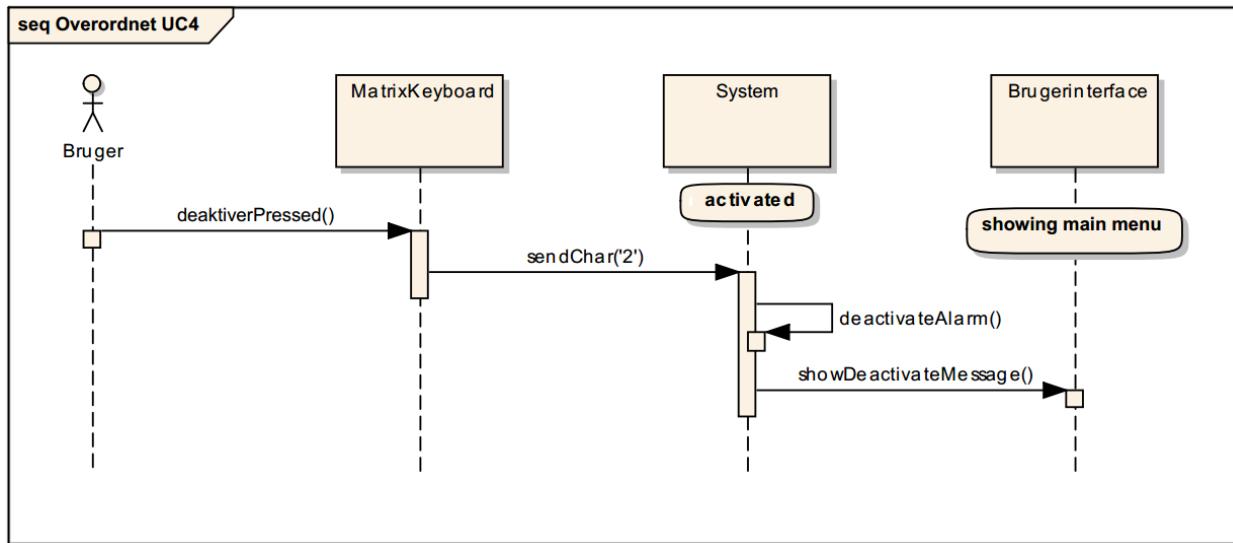
FIGUR 14 - SEKVENDIAGRAM FOR USE CASE 2

På Figur 14 vises funktionalitetet for Use Case 2 (log ud). Her trykker brugeren på en "log ud" knap på matrix keyboardet. Systemet logger blot ud og viser log ind skærmbilledet på *brugerinterface*.



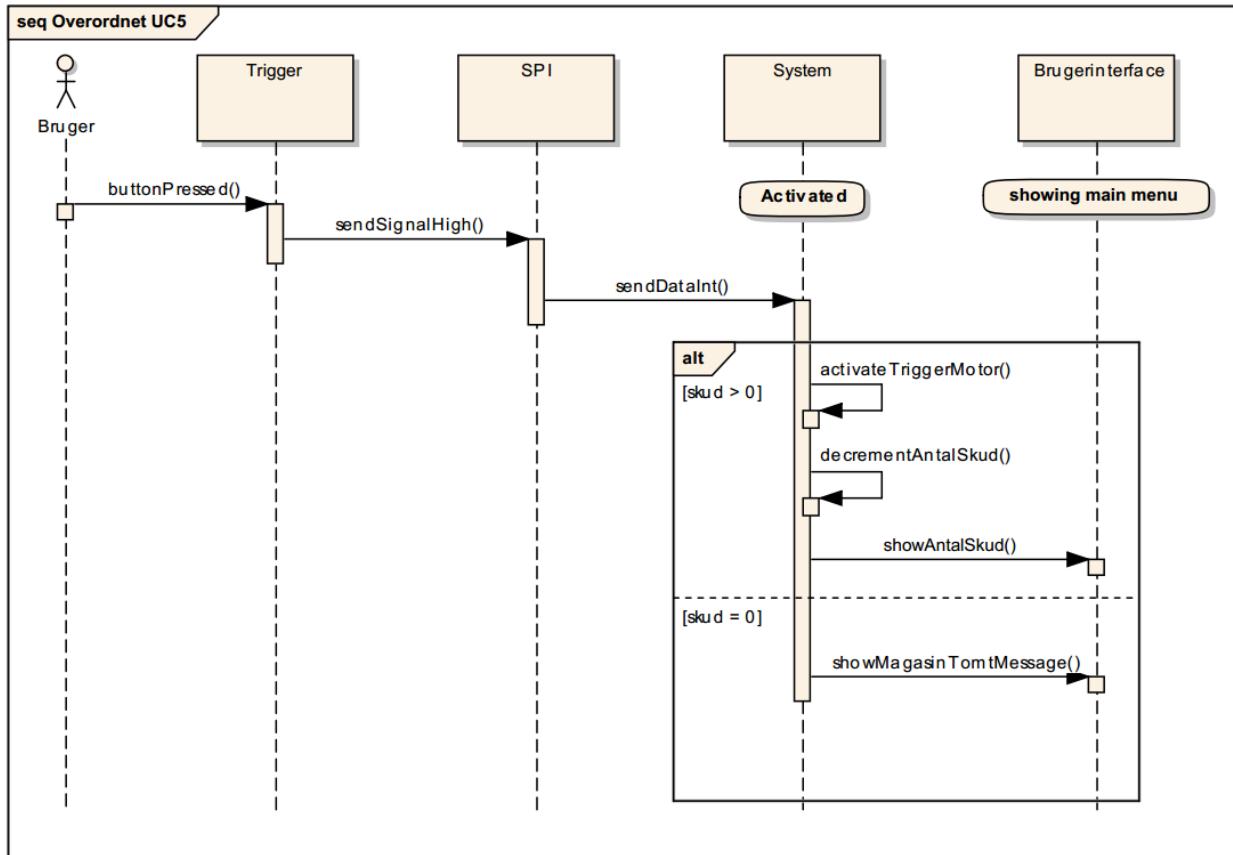
FIGUR 15 - SEKVENDIAGRAM FOR USE CASE 3

Sekvensdiagrammet for Use Case 3 (aktiver system) viser hvordan brugerens anmodning om "aktivering" behandles af systemet.



FIGUR 16 - SEKVENSDIAGRAM FOR USE CASE 4

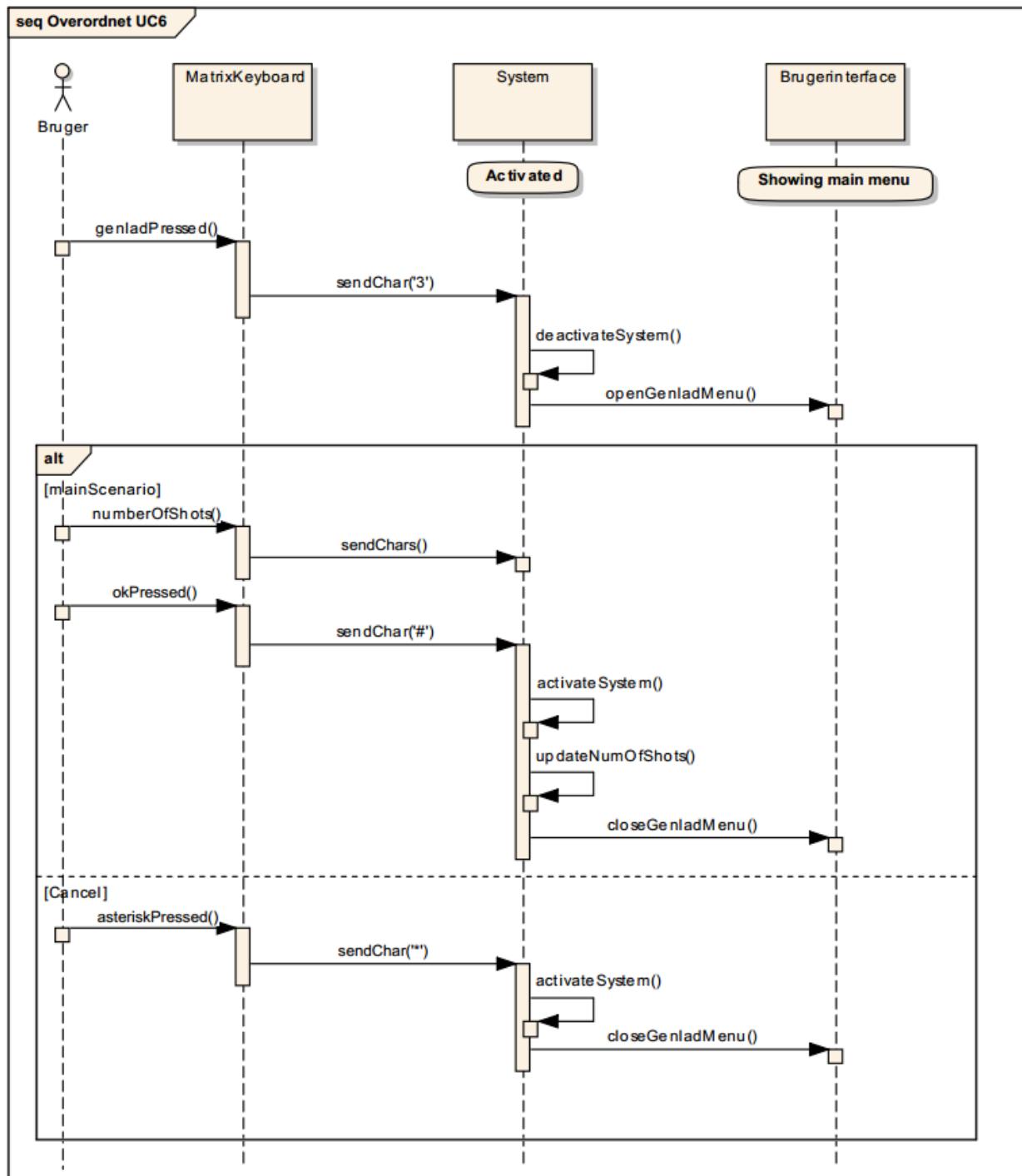
Use Case 4 (deaktivert system) sørger for at deaktivere systemet. Dette håndteres af det Indlejrede Linux System (system) og der vises en besked til brugeren på *brugerinterface*.



FIGUR 17 - SEKVENDIAGRAM FOR USE CASE 5

Her ses et sekvensdiagram for Use Case 5 (Udløs våben). Systemet skal være i "activated" state. Dette betyder blot, at systemet skal være aktivt, som i at det skal køre og være funktionelt og brugeren skal have logget ind. Det er dermed IKKE "Activated" fra Use case 2 (Aktiver system) der er tale om her.

Brugerinterfacet skal blot vise hovedmenuen.

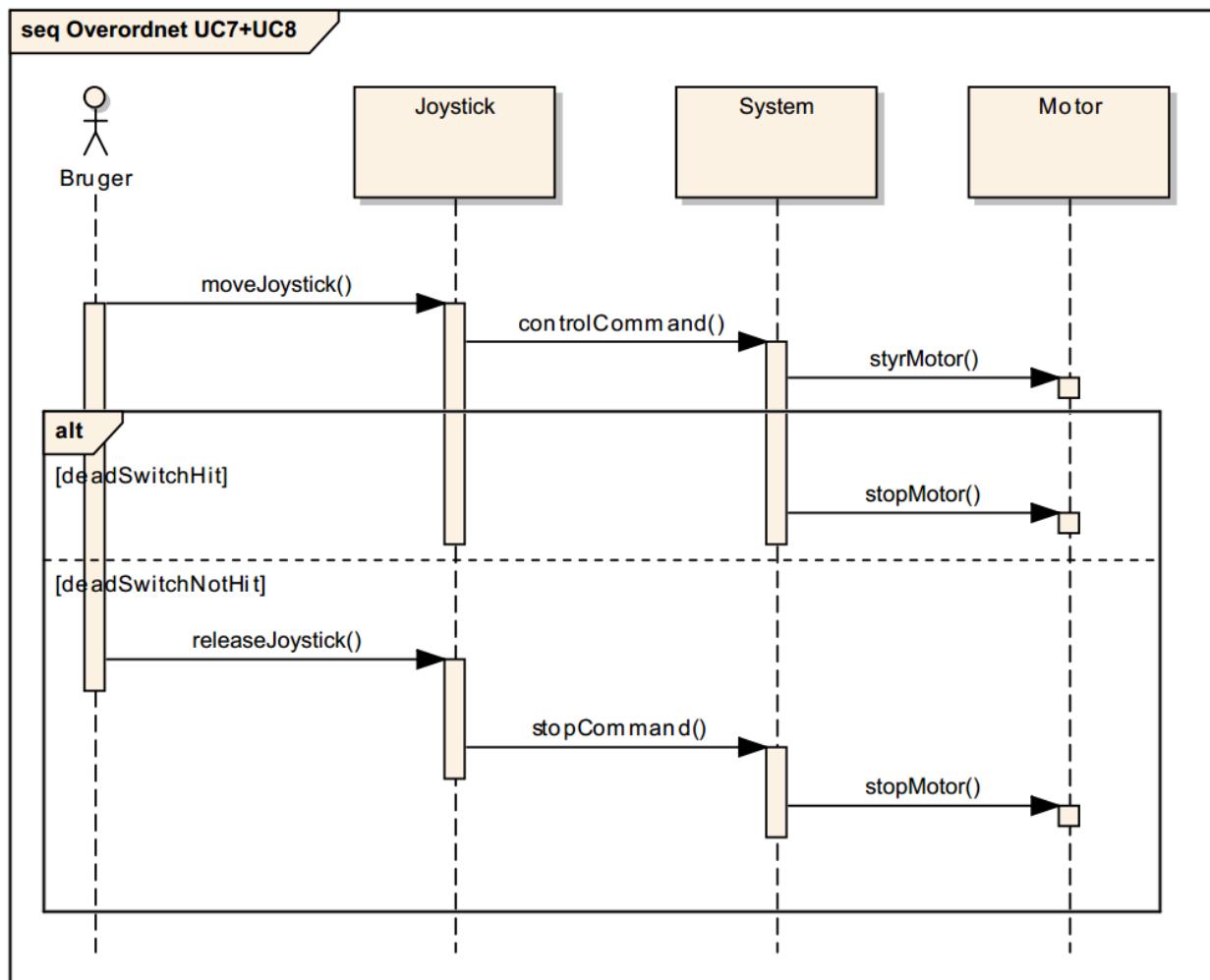


FIGUR 18 - SEKVENSDIAGRAM FOR USE CASE 6

Use case 6 (genlad) er en smule større en de foregående sekvensdiagrammer. Igen skal systemet være "Activated", som igen betyder at systemet skal være funktionelt og logget ind.

Alternative boksen har to muligheder: mainScenario og Cancel. MainScenario er den del der bliver gået igennem, når brugeren fuldfører en hel genladning, hvor han skriver et antal skud han har fyldt i og trykker godkend.

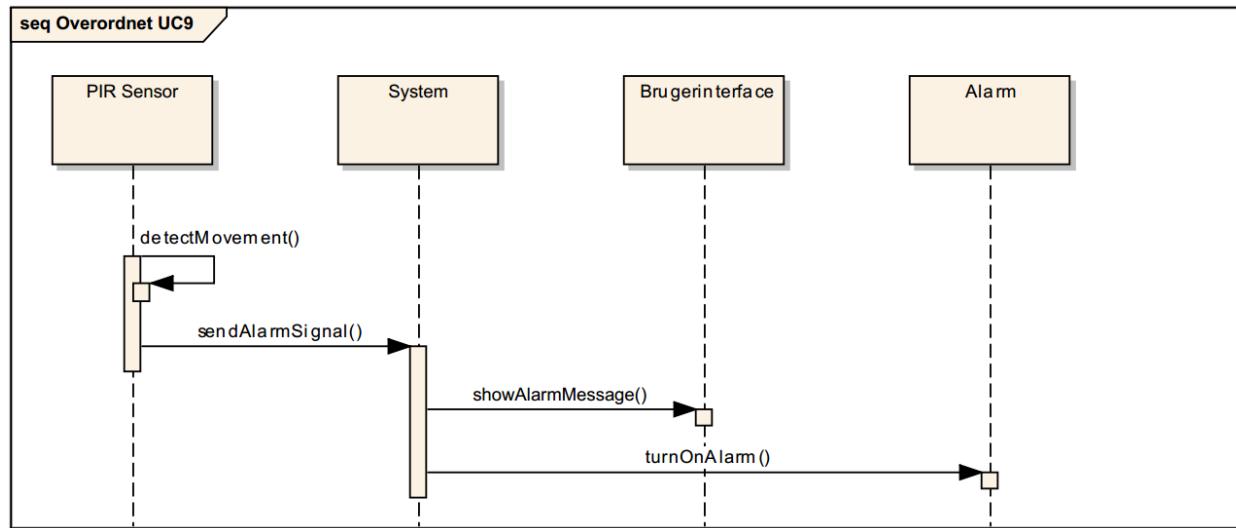
Cancel kører når bruger fortryder, og ønsker at annullere indtastningen og gå tilbage til hovedmenuen uden at foretage ændringer.

**FIGUR 19 - SEKVENSDIAGRAM FOR USE CASE 7 OG 8**

Use Cases 7 og 8 (hhv. "Bevæg HDT horisontalt" og "Bevæg HDT vertikalt") er slæjt sammen. Dette skyldes at den programmæssige funktionalitet er ens. Der er blot tale om, at der bliver tændt for forskellige motorer. Brugerens interaktion med systemet, og det der bliver udført af systemet er dog ens for de to Use Cases.

deadSwitchHit og deadSwitchNotHit er den extension der er i use case 7 og 8. Når motoren er kørt ud til et yderpunkt, så skal motoren automatisk stoppe, uanset om brugeren bliver ved med bevæge joysticket til den pågældende side.

Så længe længe dødswitchen ikke er ramt skal motoren dog blot køre som normalt, og der stoppes når brugeren slipper joysticket.



FIGUR 20 - SEKVENDIAGRAM FOR USE CASE 9

Use Case 9 (udløs alarm) er ganske simpel. Når PIR sensoren opfanger bevægelse, så viser systemet en besked til brugeren på *brugerinterface* og tænder for alarmen.

9. Design og implementering af hardware

Dette afsnit indeholder det samlede design og implementering af hardware modulerne.

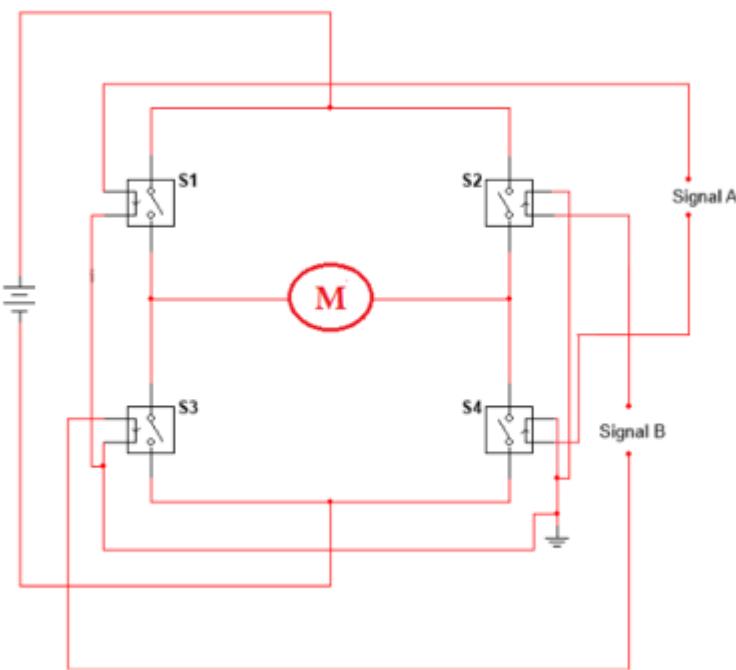
9.1. Platform Hardware [ATT]

9.1.1. Motor

Til triggeren samt horisontal og vertikal bevægelse, skal der bruges en form for motor. Der blev derfor erhvervet en viskemotor til denne opgave. Der er anvendt tre viskemotorer som er blevet brugt i HDT, to af dem er fra forruden og en er fra bagruden. De to fra forruden kører rundt og passer så til horisontal og vertikal bevægelse. Den tredje har en indbygget "frem og tilbage" funktion, og bliver brugt til triggeren. Den har også en indbygget funktion, således at hvis den sorte og lilla ledning får 12V, vil den køre tilbage til startposition. De er alle sammen tilsluttet 12V, og er kraftige nok til denne opgave. Forsyningsspændingen som bliver brugt er på 14V, men disse motorer kan godt klare denne spænding. Ved implementeringen af selve opbygningen af HDT, opstod der en fejl. Det ser ud som om, at motoren har en spændingsforbindelse fra selve motoren til akslen. Da rammen som holder pistolen var lavet af jern, løb der 14V over hele rammen. Derfor var det nødvendigt at adskille den horisontale motor fra den vertikale med plastik, ellers opstod der en korsløsning. Til fremtidig arbejde vil det være smart at have en elektriske sikring i form af dioder, i stedet for den mekaniske løsning på problemet.

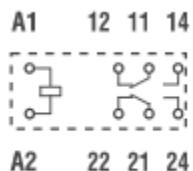
9.1.2. Motorstyring Hardware

For at Motoren kan bevæge sig horisontalt eller vertikalt, skal der være en motorstyring. Da motoren skal skifte retning hurtigt og motoren er en DC-motor, kan der bruges en H-bro til denne opgave. Derved kan der bestemmes en retning, ved at sende et signal til to af relæerne som vil lukkes, mens de andre forbliver åbne.



FIGUR 21 - H-BRO

Til H-bro'en bliver der brugt et Finder-relæ (40.525). Dette relæ kan ifølge databladet² klare op til 8A kontinuerligt, og peaks på 15A. Et signal fra microcontrolleren kan vende polariteten og dermed motorretningen. På Figur 22 ses der at signalet til A1 og A2 vil styre retningen og motorbelastningen går igennem pin 12 til 24.



FIGUR 22 - FINDER RELÆ DIAGRAM

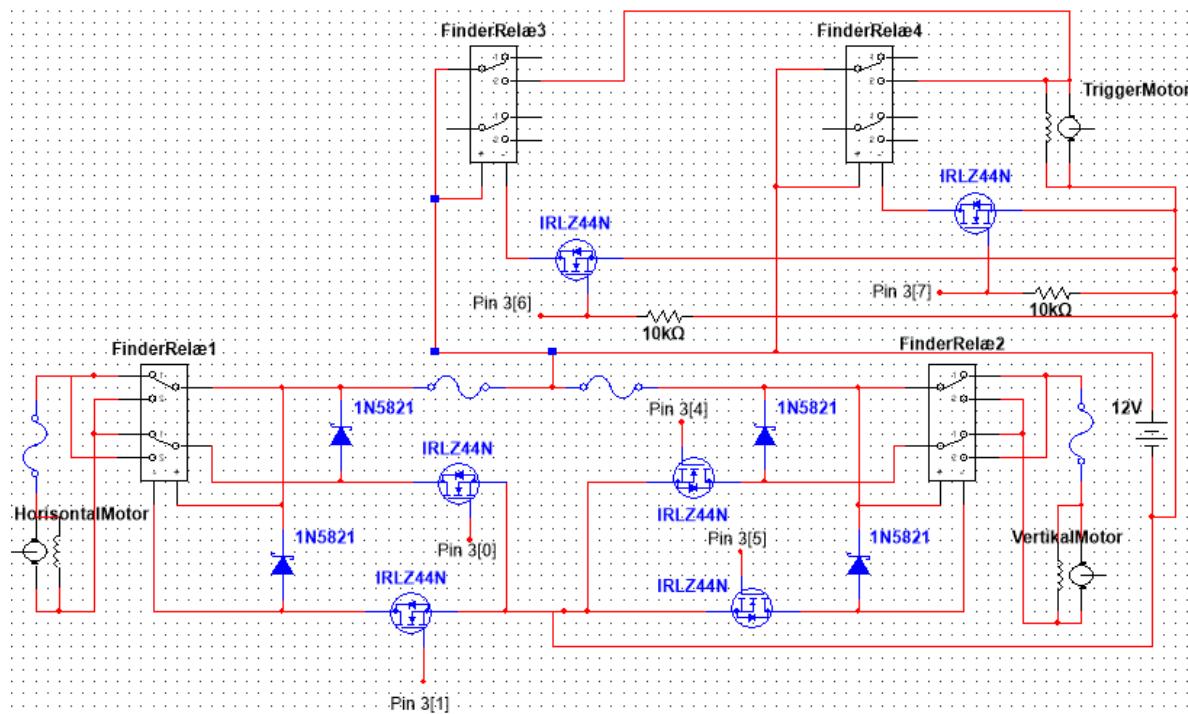
Dette signal åbner for en MOSFET som kan kontrollere om spændingen løber til relæet eller ej. På denne måde kan retningen styres med en microcontroller, og motorhastigheden styres med et PWM-signal fra

² Se bilag på CD-rom, under Datasheets. Navn: Finder Relay 40.525

microcontrolleren, som vil åbne en anden MOSFET. Der bruges MOSFET og dioder i motorkredsløbet, som kan holde til de høje strømme, der opstår ved retningsskift og motorstart. MOSFET-typen der benyttes er IRFZ44Z, som ifølge databladet³ holder til en Drain Current på 51A, og et spændingsfald fra Gate til Source på 20V. Dioderne er af typen 1N5821⁴ og kan holde til en kontinuerlig strøm på 3,0A med peaks på op til 80A.

Alle komponenterne blev valgt på grund af, at de er kraftige nok til at klare den høje strøm som motorene vil trække.

Til trigger-motoren er idéen den samme, men på grund af at motoren har en intern funktion (Læs Motor), skal man kun aktivere retningen, således at den belaster motoren med fuld kraft. Derfor bruges der to Finder-relæer. En af dem belaster motoren således at den kører frem, og den anden således at den kører til startposition.



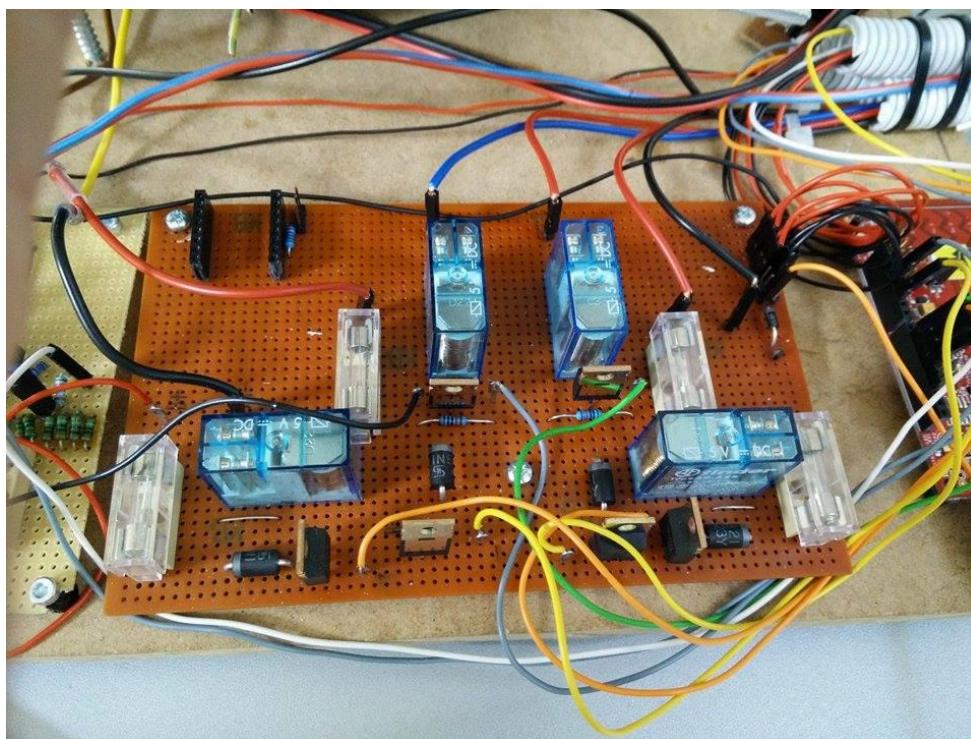
FIGUR 23 - DESIGN AF MOTORSTYRING I MULTISIM

Systemet fungerer på den måde, at Pin 3[0] vil give et PWM-signal til IRLZ44N og styre belastningen til HorizontalMotor. Når det ønskes at skifte retning, sendes der et logisk high fra Pin 3[1]. Derefter vil polariteten vendes og motoren kører den anden vej, med det samme PWM-signal fra Pin 3[0]. Det er samme princip til vertikal bevægelse, men her vil andre pins blive brugt. Dioderne bruges som sikkerhedsdioder. Når motoren kører og bliver slukket, vil den fungere som en generator, og derved vil give nogen strøm fra sig. Den strøm vil så løbe igennem dioden og tilbage til motoren. Systemet bliver derved mere sikkert. Sikringerne bliver brugt til at forhindre, at motoren trækker for meget strøm, så den ikke brænder af. Til triggeren er der to relæer til at give denne belastning, da den er nødt til at have en

³ Kan findes på bilag CD-rom, i mappen Datasheets. Navn: IRFZ44Z Motor-MOSFET

⁴ Datablad kan findes på bilag CD-rom, i mappen Datasheets. Navn: 1N5821 Motordiode

fast ground forbindelse. Pin 3[6] vil dermed styre "køre frem" funktionen. Den bliver høj i 1200ms, og derved har trigger-motoren trukket triggeren. Herefter skifter systemet over til det andet relæ ved at sende et logisk høj til "kør til start position" funktionen. Den kører i 2000ms til startpunkten. De pins som bliver brugt fra microkontrolleren afgiver noget støj og driller systemet lidt. Derfor bruges der en pull down modstand.

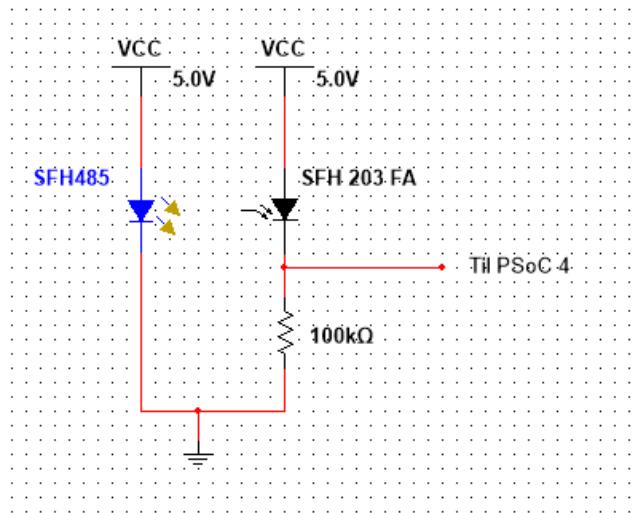


FIGUR 24 - OPBYGGET MOTORSTYRING

Den microcontroller som bliver brugt er en PSoC4. Den har indbygget PWM-signalstyring, som kan styres ved at programmere. Dette kan sættes til en udgangs-pin. Disse pins bliver sat direkte til MOSFET'en. Udgangs-pins kan også gives en logisk high, så den bliver høj i en ønsket tid. Den bruges til at skifte retning over til triggeren.

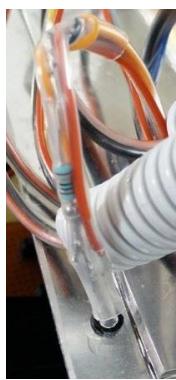
9.1.3. Motorstyring Dødsbricka

Systemet skal have en sikring for at HDT ikke kan bevæge sig for langt horisontalt eller vertikalt. Til dette bruges der en infrarød SFH485 sender og en SFH203 modtager. Når HDT'en for eksempel har kørt for langt til højre, vil en sender og modtager mødes, og dette vil give en logisk high til PSoC'en. Via kodning vil den aflæse den logiske high og stoppe motoren.



FIGUR 25 - OPBYGNING AF SENORKREDSSLØB I MULTISIM

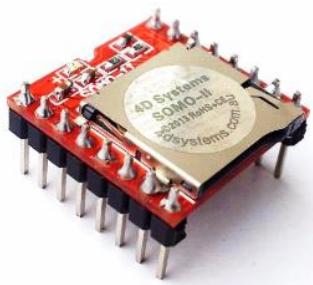
For at LED'erne lyser nok til at de infrarøde sensorer nemt kan opfange lyset, er der ingen modstand før LED'erne. Der bliver brugt tre LED'er for hver enkelt senderenhed. Når fotodioden opfanger infrarødt lys, åbner den for spændingen, som løber til PSoC'en og giver den en logisk high. PSoC'en bliver programmeret således, at en pin bliver brugt som en indgang. Ved test af denne pin, er der brugt en direkte VCC-forbindelse fra PSoC'en, til den pin som skal læse. Det var nødvendigt at sætte pinden til "Resistive Pulldown", ellers vil den give en high hele tiden, selvom VCC ikke er koblet til. Herefter kobles sensoren til PsoC'en, og systemet holder op med at virke. Det så ud til at der var et spændingsfald over fotodioden som tilsvarer 4.5V. Dette var ikke nok til at PSoC'en registrerede logisk high. Efter at have testet i lang tid, viste det sig at på grund af at modtagerenheden har en pull down modstand, og at PSoC-pinden var sat til "Resistiv Pulldown", opstår der en belastningsfejl og der sendes derved ikke 5V. Når læsepinden bliver sat til "High Impedance Digital" bliver fejlen ordnet, og hver gang fotodioden fanger infrarødt lys, giver den et stabilt 5V signal til PSoC'en.



FIGUR 27 - OPBYGGET FOTODIODE ENHED

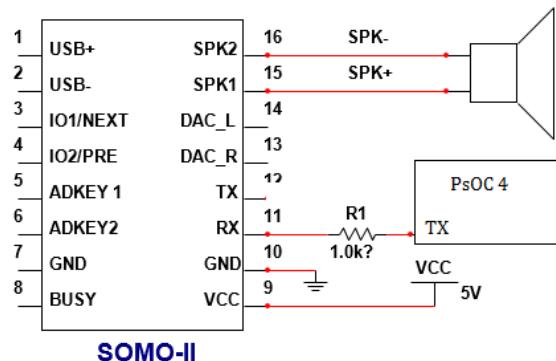
FIGUR 26 - OPBYGGET LED ENHED

9.1.4. SOMO-II Hardware



FIGUR 28 - SOMO-II

Til at afspille en alarm på platformen, bruges der en SOMO-II. SOMO-II kan læse mp3-filer direkte fra et micro SD-kort og afspille disse på en højttaler. SOMO-II'en skal have et VCC-signal på 5V fra PSoC'en. Dette går til ben 9 på SOMO-II'en. Ground forbindes til fælles ground på Platformen. TX på PSoC'en bliver koblet til RX på SOMO-II, altså ben 11. Der sættes en formodstand på $1\text{k}\Omega$ inden RX (aflæst fra databladet til SOMO-II⁵), da PSoC'en sender et 5V-signal, og SOMO-II's RX-ben skal have 3.3V. Der bliver målt 3.6V som løber in på RX, men det er ikke nok til at give en fejl. Derefter tilsluttes højtaleren med positiv indgang på ben 15 og negativ indgang på ben 16. Der blev erhvervet en højttaler fra et Philips-TV, som blev fundet i "Storskrald Århus". Det er en 8Ω , 10 Watts-højttaler.



FIGUR 29 - OPPBYGNING AF SOMO-II I MULTISIM

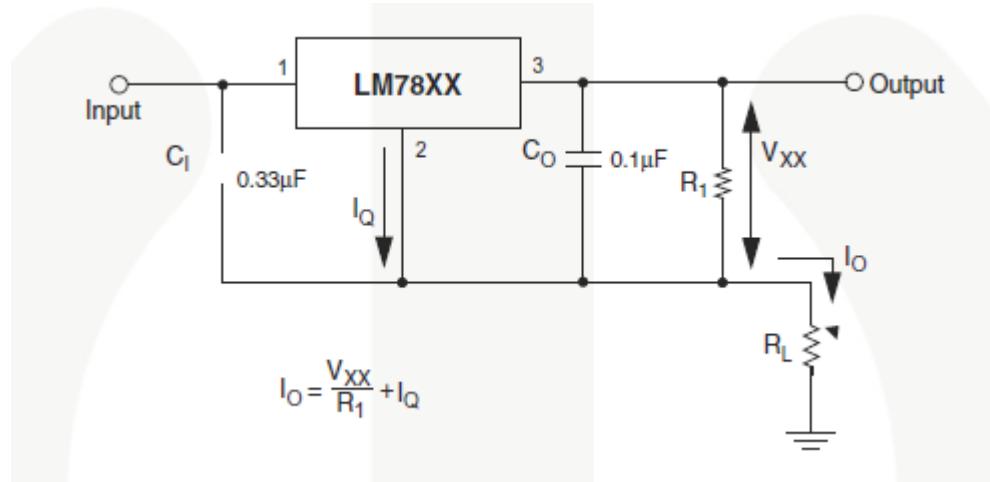
⁵ Se bilag på CD-rom, under Datasheets.

9.2. Design Spændingsforsyningssprint [LRA]

Spændingsforsyningssprintet skal forsyne tre DC motorer, en laser og 3*3 SFH485 dioder. Alle disse tre kredsløb har den samme strømforsyning, som er en 14VDC transformator, derfor ligger de på det samme print. Ud for 14V forsyningen er der tre sikringer som hver går ud til deres eget kredsløb.

Til de tre DC motorer, som er 12V viskermotorer, lader vi strømmen gå direkte ind fra transformeren da de sagtens kan klare en lidt højere spænding. Den sikring som sidder i inden motorene er på 8A, da der skal trækkes en stor strøm igennem.

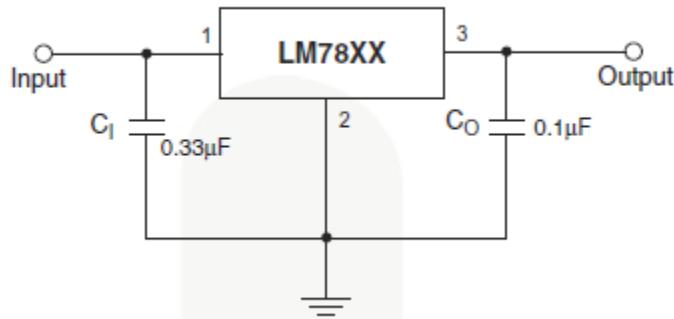
Laseren skal have 3,7V DC, dette klares ved at lade en modstand sænke spændingen fra 14V til 3,7V. En anden tanke er at bruge en LM7805 til at sænke spændingen til 5V og derefter lade en modstand sænke spændingen ned til 3,7V. Det er også muligt at bruge LM7805 i et bestemt kredsløb som kan findes i databladet for LM7805⁶ (se Figur 30) der bruger en modstand R_1 til at regulere spændingen. Den sikring som sidder i inden laseren er en på 500mA, så at undgå at laseren overopheder.



FIGUR 30 VARIABEL SPÆNDINGSREGULATOR

⁶ Databladet findes på CD-rom bilagene, under Datasheets.

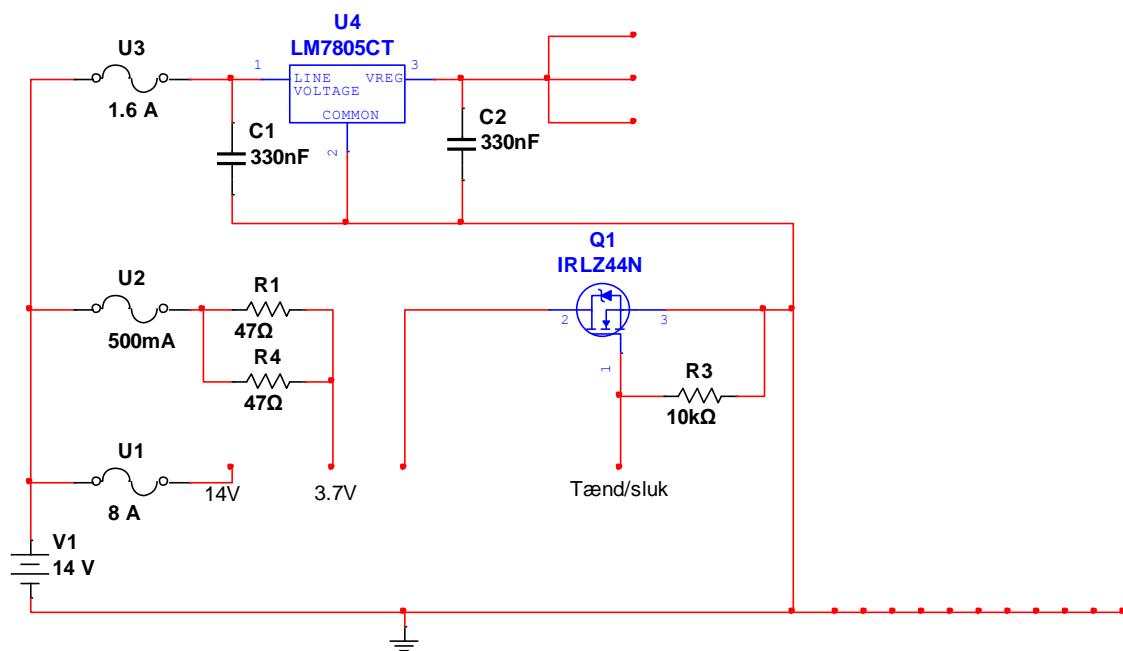
De 3*3 SFH485 dioder skal have 5V. Dette klareres med en LM7805. Kredsløbet findes i datasheetet for LM7805⁷, se Figur 31. Den sikring som sidder inden 3*3 dioderne er en 1,6A sikring.



FIGUR 31 FIXED SPÆNDINGSREGULATOR FOR LM7805

9.2.1. Implementering spændingsforsyningens print

Transformeren giver en spænding på 14V og da der skal bruges forskellige spændingsniveauer. Der skal 14V til de tre motorer, 3,7V til laseren og 5 volts spænding der forsyner lyssensorerne. Der sidder 3 sikringer fra 14V forsyningen i forskellige størrelser, der sikrer de forskellige komponenter.



FIGUR 32 MULTISIM DIAGRAM OVER SPÆNDINGSFORSYNINGSPRINT

⁷ Databladet findes på CD-rom bilagene, under Datasheets.

9.2.2. 5V forsyning lyssensor

Denne forsyning går til lyssensorerne, de skal have 5V og de trækker 0,9A. Den første plan var at lade PSOCen trække disse dioder, men det kan den slet ikke kan trække så stor en strøm, bliver de istedet trukket af et spændingsforsyningsprint.

Til at lave 5V bruges der en LM7805 spændingsregulator. Kredsløbet for 5V forsyningen er et standard kredsløb fundet i databladet for LM7805 se Figur 31. Sikringen der sidder inden dette kredsløb er på 1,6A, der blev først prøvet med en 1A sikring, men den sprang efter et stykke tid.

For at forhindre spændingsregulatoren fra at brænde sammen, er der påsat en køleplade.

9.2.3. 3.7V forsyning laser

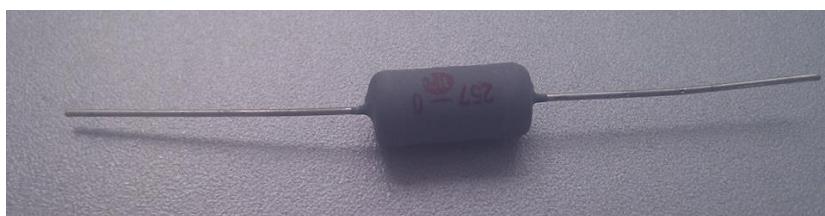
Denne forsyning går til laseren, og den skal have 3,7V og den trækker 0,4A.

Til 3,7V reguleringen er der benyttet 21 ohms modstand. Da vi har et spændingsfald på 10,3V og en 0,4A strøm, kan effekten afsat i modstanden udregnes til 4,12W, men det viser sig at MRS25 som er vores standart laboratoriemodstande, maximalt kan klare 600mW og der kommer til at være 4,12W over modstanden, så resultatet var at modstanden brændte sammen.

På det allerede producerede print var der plads til 7 parallelforbundne modstande, for at opnå en modstand på 21 ohm skulle de hvert være på 147 ohm se Ligning 2. Syv 150ohms modstande blev valgt som en tilnærmelse. De 7 modstande kan tilsammen klare 4,2W, hvilket er lige på grænsen af hvad de kan klare, men de kan ikke klare denne belastning over en længere periode, og de begyndte af ryge efter nogle minutter.

Efter at de 7 modstande brændte sammen, er to 257-0 modstande på 47 ohm blevet sat i parallel, dette giver en modstand på 23,5 ohm, hvilket er en rimelig tilnærmelse. Disse modstande kan klare op til 5W hver og kan tilsammen klare 10W hvilket er meget mere end nødvendigt.

Da laseren trækker 0,4A er der sat en 0,5A sikring mellem 14V forsyningen og modstandene.



FIGUR 33 BILLEDE AF 5W MODSTAND

$$10,3 \cdot 0,4 = 4,12$$

LIGNING 1 - UDREGNING AF EFFEKT AFSAT I MODSTAND

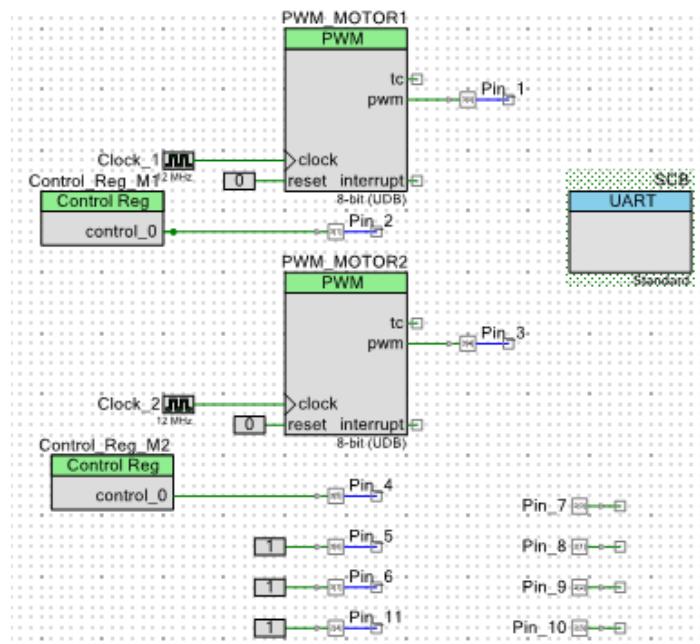
$$\frac{1}{21} = \frac{1}{x} \cdot 7 \quad \xrightarrow{\text{solutions for } x} \textcolor{blue}{147}$$

LIGNING 2 UDREGNING AF MODSTANDSSTØRRELSE TIL 7 PARRALLELE MODSTANDE

9.3. Platform Software [ATT]

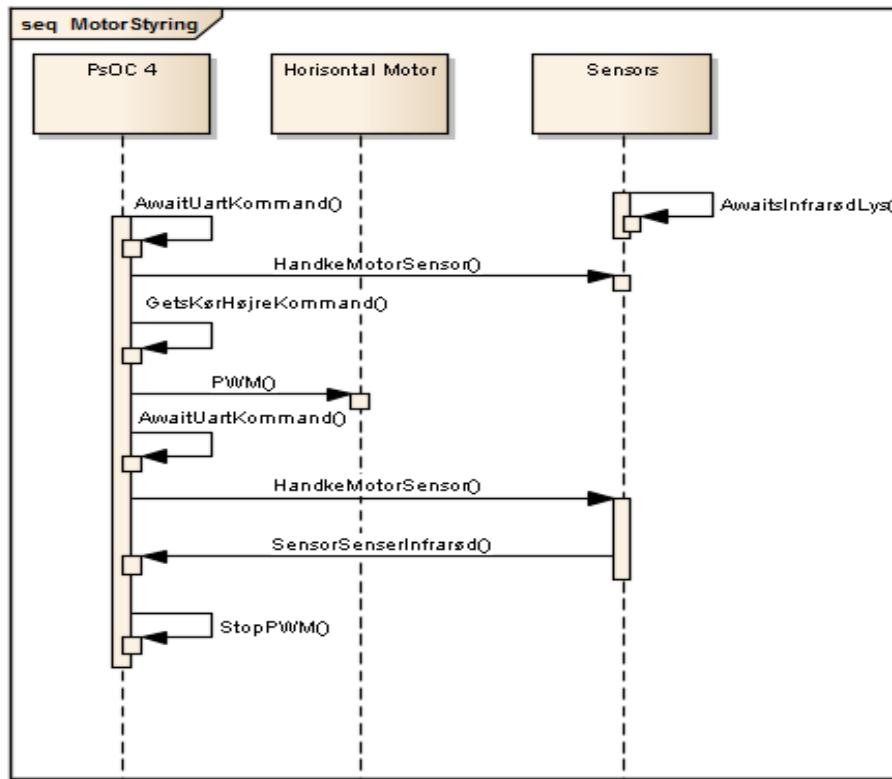
9.3.1. Software Beskrivelse

Til at styre hele hardwaresystemet, er det nødvendigt at have en microkontroller. Der benyttes en PSoC4 til dette projekt. Den har mange funktionaliteter som passer til de opgaverne i projektet. Disse funktionaliteter kan alle findes i TopDesign. Alle de funktionaliteter kan kontrolleres med indbyggede funktionskald, som kan findes i databladet, for hver af de enkelte funktionaliteter.



FIGUR 34 - TOPDESIGN AF PLATFORM SOFTWARE

Kommunikationen mellem Indlejreje Linux System og PSoC4 sker igennem UART TTL kommunikation. Idéen er at PSoC'en venter på en kommando fra Indlejret Linux System, men tjekker de infrarøde sensorer samtidigt, og når PSoC'en får en kommando, vil den udføre den tilsvarende kommando. Derefter venter den igen på den næste kommando.



FIGUR 35 - SEKVENS DIAGRAM. EKSEMPEL PÅ AT MOTOR KØRER TIL HØJRE INDTIL DØDSSWITCH

Handlingen på UART-dataen sker således, at de 4 bytes som ligger i RX-bufferen, bliver sat til bestemte variabler som har de samme navne og kommandoer i protokollen. Som sagt:

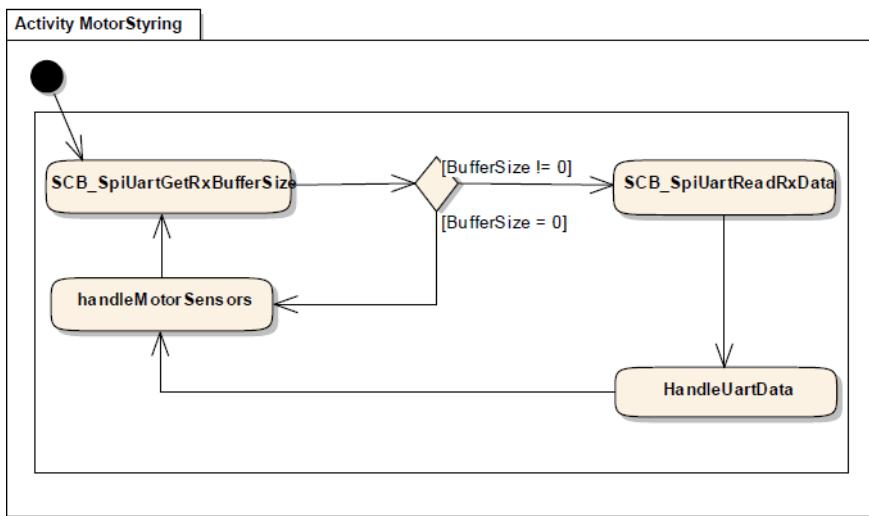
Start – Altid den samme, og angiver hvornår koden starter.

Kommand – Fortæller hvilken slags kommando der er, samt værdien til tjeksum.

Option – Bestemmer hastigheden til motorene, samt værdien til tjeksum.

Tjeksum – Tjekker hvis kommandoen har en fejl eller ej.

Når Start-variablen har fået en char som svarer til ASCII-værdien 1, vil den tjekke hvis Kommandoen XOR'et med Option opfylder den værdi som Tjeksum har. Derefter tjekkes hvilken kommando der skal gennemføres, fx hvis kommandoen er at køre horisontalt, vil Option afgive den hastighed som motoren skal køre med.



FIGUR 36 - ACTIVITY DIAGRAM AF MOTOR STYRING SOFTWARE

9.3.2. UART

Til UART-kommunikation bruges UART-blokken i PSoC'en (se Figur 34). Den kan tilsættes til en bestemt baud rate. Ved første forsøg blev der brugt en standard UART[v2.30] block, men når hele "TopDesign" var opbygget, opstod der et problem på grund af, at PSoC'en har ikke nok "Paths" til at klare dette. Derefter blev der prøvet med UART(SCB mode) [v2.0]. Herved var problemet ordnet. UART'en bliver delt op i to dele, hvor RX aflæser fra Indlejret Linux System, og TX bliver brugt til at aktivere SOMO-II(Læs SOMO-II). Ved aflæsning af UART-kommando fra Indlejret Linux System bruges funktionen SCB_SpiUartReadRxData(), som vil hente næste byte som ligger i RX-bufferen. Til testen af dette, blev der brugt Analog Discovery og tjekket om PWM går i gang når Indlejret Linux System sender denne kommando. Dette virkede ikke. Derefter blev dette testet ved hjælp af Silicon Labs CP210x USB til UART bridge. Derved blev PSoC'en programmeret til at sende det den aflæser fra UART til Silicon Labs CP210x's bridge. Det viste sig at det som PSoC'en aflæste, ikke passede helt. Problemet var at den prøvede at læse på forkerte tidspunkter. Løsningen på dette var at få funktionen til at vente til at RX-bufferen var forskellig fra 0. Der anvendes SCB_SpiUartGetRxBufferSize() funktionen, og derved startede UART'en altid at læse på rigtige tidspunkt. Til den anden del hvor UART'en skal sende en kommando til at aktivere SOMO-II, blev der brugt SCB_SpiUartWriteTxData() funktionen. Denne aktiverede ikke SOMO-II'en. Herfør blev der tjekket hvad PSoC'en sendte via UART til USB bridge. Det viste sig at PSoC'en sletter nogle af de bytes som den skal sende. Herefter blev der prøvet med SCB_UartPutChar() funktionen, og så slettede den ikke nogle bytes, og derved blev SOMO-II'en aktiveret.

9.3.3. Styring af horisontal og vertikal motor

Til selve styringen af motoren, anvendes PSoC'ens indbyggede PWM-funktionalitet, som er en block som skal sættes til i TopDesign (se Figur 34). Denne kan sættes til en Digital Output pin. Derefter kan man kalde de indbyggede funktioner som PSoC'en har til PWM-blokken. Her kan der skrives et compare som vil sætte den ønskede PWM-værdi. Derefter skrives en startkommando til PWM-blokken. Hvis der skal skiftes retning på motoren, vil PSoC'en give en logisk high til et Controll Register, som er forbundet til en Digital Outout pin. Dette Control Register er valgt, da det giver et renere logisk low. Hvis den ikke bliver brugt når den er lav, opstår der en lille støj som driller hardwaresystemet.

```
Control_Req_M2_Write(1);
PWM_MOTOR2_WriteCompare(60);
PWM_MOTOR2_Start();
MotorKorerNed = 1;
```

FIGUR 37 - EKSEMPEL PÅ MOTOR AKTIVERINGS KODE

9.3.4. Styring af Dødswitch

Programmet har funktionen HandleMotorSensors(). Den håndterer dødswichen på platformen. Den skal altid være aktiv når programmet er inaktiv. Således vil det aflæse sensoren meget præcist. Til at modtage signalet fra infrarød-sensoren er der brugt en Digital Input pin. Den kan direkte aflæse med Pin_Read() funktionen. Når en af kommandoerne som starter en motor bliver udført, bliver der sat en variabel til high(MotorKorerHøjre), som bliver brugt til at compare med infrarødsensoren. Når de to krav bliver opfyldt, vil programmet stoppe denne motor med de samme.

9.3.5. SOMO-II Platform Software [ATT]

Ifølge databladet for SOMO-II, kan kommunikation med SOMO-II ske via UART. Her skal baud rate'en sættes til 9600. Dette kan tilføjes i TopDesign I PSoC'en (se Figur 34). PSoC'en bruger en internal clock til at give den bestemte baud rate, som er ikke helt præcis. Den giver en advarsel om at baud rate'en er 9615, men det er indenfor den grænse hvor det fungerer. Denne baud rate skal have følgende parametre:

Baud Rate: 9600 bps
Data bits: 1
Parity bit: none
Flow Control: none

SOMO-II'en har et indbygget serielt kommando-bibliotek, men alle kommandoerne er 8x8 bit kommando-strenge som skal sendes til RX på SOMO-II'en for at den udfører dem. Kommandoerne kan aflæses i databladet⁸, og er i hex-værdier som inddeltes i følgende dele:

Startbit – Altid den samme, og angiver hvornår koden starter

Kommandokode – Fortæller hvilken slags kommando det er

⁸ Datablad for SOMO-II. Kan findes på bilag CD-rom, under Datasheets.

Feedback – Fortæller om feedback er nødvendigt eller ej

Para1 – Første parameter af koden

Para2 – Anden parameter af koden

Checksum1 – Første byte af checksum

Checksum2 – Anden byte af checksum

End character – Altid den samme, og angiver hvornår koden slutter

Function	Serial Command
NEXT	7E 01 00 00 00 FF FF EF
PREVIOUS	7E 02 00 00 00 FF FE EF
SPECIFY TRACK #	7E 03 00 00 01 FF FC EF 7E 03 00 00 02 FF FB EF 7E 03 00 00 0A FF F3 EF
VOLUME +	7E 04 00 00 00 FF FC EF
VOLUME -	7E 05 00 00 00 FF FB EF
VOLUME #	7E 06 00 00 1E FF DC EF 7E 06 00 00 05 FF F5 EF

FIGUR 38 - EKSAMPLE PÅ KOMMANDO

Til dette projekt bruges der to kommandoer, NEXT og VOLUME #. NEXT kommandoen vil afspille den næste fil på SD-kortet. VOLUME # sætter lyden til højeste niveau. Disse kommandoer bliver sendt via en string som indeholder hex-værdier som skal sendes. Der kan også kaldes en bestemt fil i SOMO-II'en. Eventuelle forbedringer kan være at lave en advarselslyd for hver type af dyr, som brugereren ønsker at skræmme væk. Der kan også være en advarsel til folk om at det er privat område.

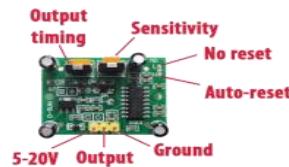
9.4. Controller hardware og software [SF, KB]

9.4.1. PIR sensor

Dette afsnit giver et overblik over, hvordan løsningen til PIR sensor modulet er løst. PIR sensoren, der er blevet valgt, er en Pyroelectric Infrared PIR Motion Sensor Detector Module HC-SR501



FIGUR 40 - PIRESENSOR HC-SR501



FIGUR 39 - PIRESORENS SPECIFIKATIONER

Datasheet for PIRsensoren kan findes under bilag⁹. Denne PIRsensor er valgt, fordi den kan operere på 5V, og den var billig. Ifølge databladet kan PIRsensoren detektere op til 7 meter. PIRsensoren har tre pinde:

- 5V fra Indlejret Linux System
- Output er sat til ADC
- Ground er sat til Indlejret Linux System

Hvordan PIRsensoren er koblet til ADC, kan ses på Figur 44.

9.4.2. Trigger

Følgende afsnit giver et overblik over, hvilke komponenter, der er blevet valgt til dannelsen af Trigger. Til modulet Trigger, er der blevet fundet en tryktast. Tryktasten er en 1-pol ON/OFF tryktast og danner herved baggrund for Trigger.



FIGUR 41 - TRIGGER

Der er blevet valgt to outputs på tryktasten. Når man trykker på Trigger, skal den sende en højspænding ud af den ene udgang. Der skal kobles to signaler på Trigger.

- 5V fra Indlejret Linux System
- Output går til ADC. Her findes en forbindelse til GND

⁹ Kan findes på bilag CD-rom, i mappen Datasheets.

9.4.3. Joystick

I projektet skal der tilkobles et Joystick. Dette skal kommunikere med ADC, og brugeren skal kunne intragere med Joystick, for at kunne styre motorerne.



FIGUR 42 - JOYSTICK

Joystick skal være med til at kunne bevæge platformen vertikalt og horisontalt. Ovenstående Joystick har også en knap, der blev overvejet som trigger. Denne er dog ikke blevet valgt, da det ville kunne forstyrre ens sigte, når man trykker på den. Joystickket består af to potentiometre. En der giver en spænding i forhold til Joystickkets horisontale stilling, og et andet der giver en spænding i forhold til den vertikale stilling.

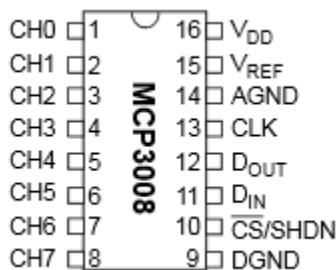
Følgende pins er blevet forbundet til systemet.

- vRX: Output til ADC.
- vRY: Output til ADC.
- GND: Reference til Indlejret Linux system
- Vcc: Forsyningsspænding fra Indlejret Linux System

9.4.4. ADC

ADC har som formål at fortolke signalerne fra PIRSensor, Joystick og Trigger, for derefter at sende dem videre til Indlejret Linux System. Dette skal gøres med SPIbus, der skal anvendes til at kommunikere med Indlejret Linux System. For at opnå dette, er der valgt en MCP3008 som ADC. MCP3008 konverterer et analogt signal til et digitalt signal. Både til valg af Joystick og ADC'en MCP3008, er der hentet inspiration fra dette projekt.¹⁰

¹⁰ <http://www.raspberrypi-spy.co.uk/2014/04/using-a-joystick-on-the-raspberry-pi-using-an-mcp3008/>

**FIGUR 43 - MCP3008**

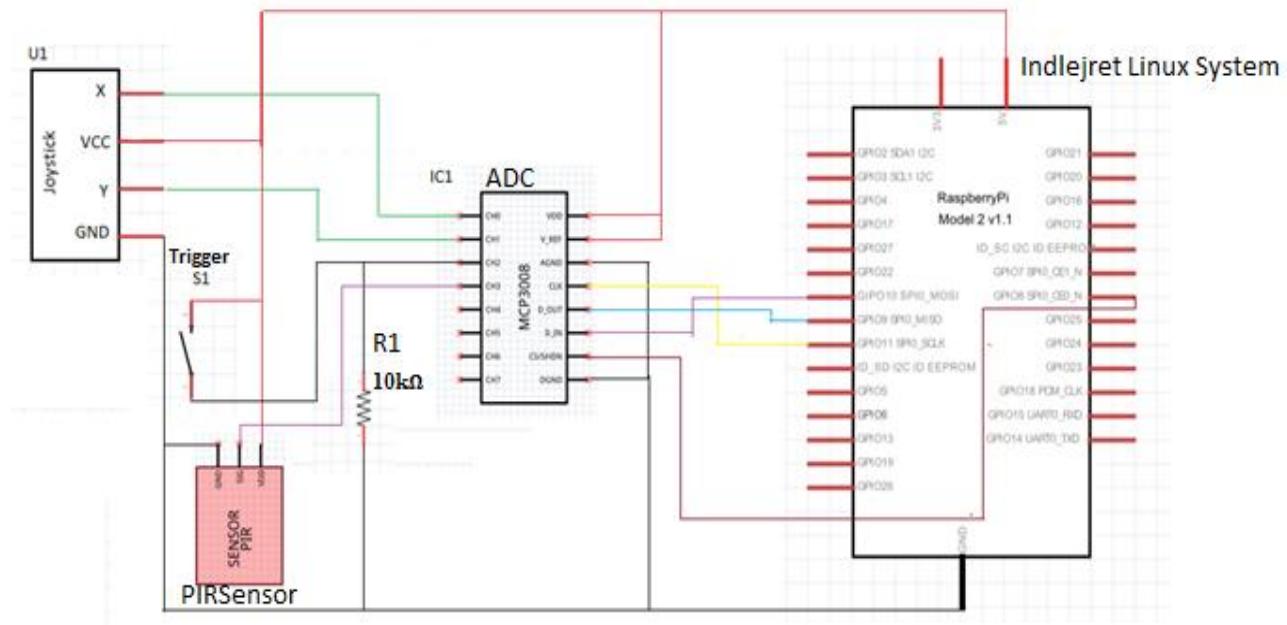
ADC'en er en 8-channel 10-Bit A/D converter. ADC'en er også valgt, fordi den har 8-channels til input af fx sensorer. Der anvendes 4 channels for input fra henholdsvis Trigger, to fra Joystick og en fra PIRsensor. ADC'en har et Serielt Perifiskt Interface, der skal anvendes for at kommunikere med Indlejret Linux System. Til SPI kommunikation med Indlejret Linux System skal der anvendes fire forbindelser foruden Vcc og GND.

9.5. Kommunikation mellem ADC og Indlejret Linux System

Kommunikationen i mellem ADC og Indlejret Linux System er implementeret med SPI.

- MOSI: Data bevæges fra Indlejret Linux System til ADC.
- MISO: Data bevæges fra ADC til Indlejret Linux System.
- SCLK: Clocken fra Indlejret Linux System. Anvendes til at synkronisere transmissionen af data imellem ADC og Indlejret Linux System.
- SS: Dette er chip select benet, her tilsluttes slave.
- Gnd og Vcc.

Formålet med ADC'en er, at den skal fungere som slave imens Indlejret Linux System skal fungere som master. Selve opstillingen imellem ADC og Indlejret Linux System kan ses på Figur 44. I diagrammet er medtaget Joystick, Trigger og PIRsensor. Der er tilføjet en enkelt modstand i diagrammet, dette skyldes, at der kan være støj på ADC'ens indgangskanal, når der ikke trykkes på Trigger.



FIGUR 44 - OPSÆTNING AF ADC

9.5.1. Opsætning af SPI kommunikation

ADC'en kan sample op til 200k samples per sekund. Det afhænger dog af reference spændingen, der dog max skal være 5V. Både forsyningsspændingen og referencespændingen sættes til 5V, da der ønskes hurtigst mulig sampling, da ADC'en skal anvedes til "real time operations". Både AGND og DGND sættes til ground. Alle forbindelser kan findes i figur Figur 44.

Referencespændingen vil herved have den betydning, at hvis ADC sampler 5V, så vil dette blive fortolket som den største digitale værdi, og den vil blive repræsenteret som et 10-bit tal. 5V, den højeste analoge værdi, skal blive fortolket digitalt som $2^{10} - 1 = 1023$. Den laveste analoge værdi vil herved blive VREF/1024. Dette giver en analog værdi på $5/1023 = 2.08\text{mV}$ og dette vil repræsentere en digital værdi på 1. Den formel der skal anvendes til at fortolke den analoge værdi til den digitale værdi bliver $digi = 1024 * \text{VIN}/\text{VREF}$, hvor Vin, er det output, der fx kan komme fra Joystick. De digitale til analoge formler, kan alle findes i datasheet for ADC'en¹¹.

Når Indlejret linux system sender data til ADC'en igennem MOSI porten, skal den sende tre bytes. Følgende illustration stammer fra ADC'ens datasheet.

¹¹ Kan findes på bilag CD-rom, i mappen Datasheets. Navn: MCP3008

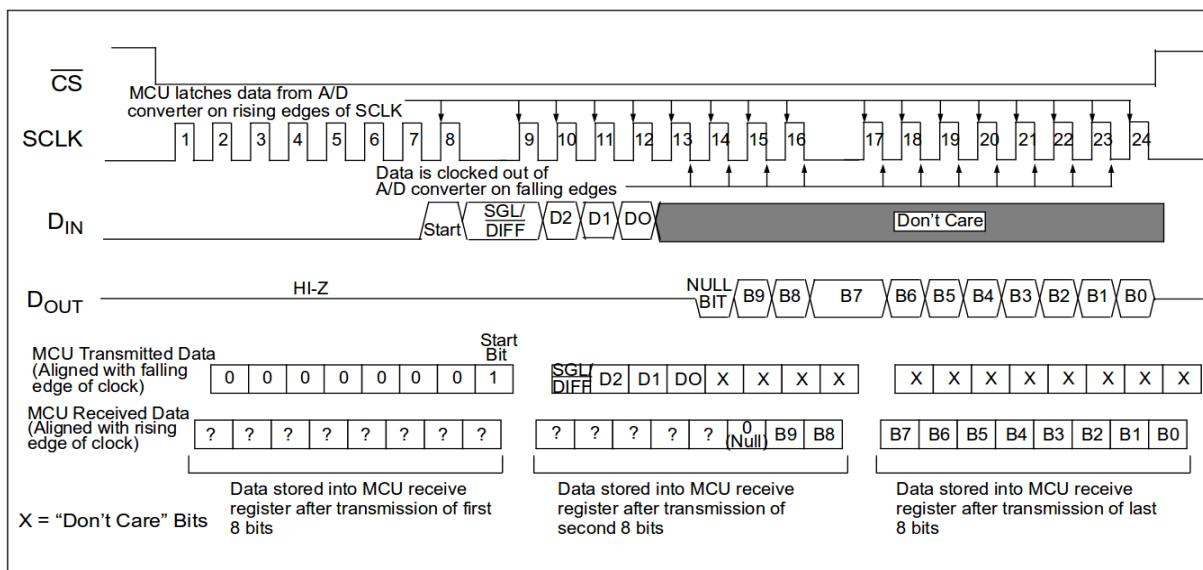


FIGURE 6-1: SPI Communication with the MCP3004/3008 using 8-bit segments
(Mode 0,0: SCLK idles low).

FIGUR 45 - SPI KOMMUNIKATION MED MCP3008

Ovenstående illustration viser en SPI transaktion af ADC'en. Her kan det ses, hvordan ADC'en opererer.

Indlejret Linux System sætter sin forbindelse til chip select benet til ADC'en til 0V. Dette skal ske, når den rette funktion kaldes af Indlejret Linux System.

Herfra sendes den første byte. Denne inkluderer kun en værdi på 1, hvilket er en start bit. Ved anden byte skal bit 7, 6, 5 og 4 sættes. I tabellen på Figur 46 kan ses, hvordan disse skal sættes alt afhængigt af, hvordan man ønsker ADC'en skal operere. Det er her man bestemmer, hvilken channel man vil læse fra på ADC'en og her kan man bestemme om det skal være single ended eller differential ended.

Tredje byte, der sendes, er en byte fyldt med "Don't Care", denne sendes samtidigt med, at ADC'en sender en byte tilbage til Indlejret Linux System, der repræsenterer den samplede værdi digitalt.

Indlejret Linux System fortolker 10-bit digital værdi. Dette gøres ved at sammensætte bit 8 og 9 fra byte nummer to sammen med bit 7 og 0, modtaget fra den tredje byte. Nu har konverteringen resulteret i et 10-bit tal, der skal kunne aflæses på Indlejret Linux system.

Da ovenstående er på plads, kan softwaren til selve ADC'en skrives, så den kan fortolke analoge signaler fra Joystick, Trigger og PIRsensor.

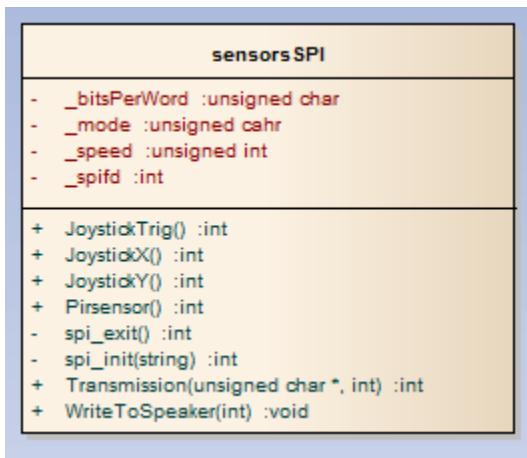
TABLE 5-2: CONFIGURE BITS FOR THE MCP3008

Control Bit Selections				Input Configuration	Channel Selection
Single /Diff	D2	D1	D0		
1	0	0	0	single-ended	CH0
1	0	0	1	single-ended	CH1
1	0	1	0	single-ended	CH2
1	0	1	1	single-ended	CH3
1	1	0	0	single-ended	CH4
1	1	0	1	single-ended	CH5
1	1	1	0	single-ended	CH6
1	1	1	1	single-ended	CH7
0	0	0	0	differential	CH0 = IN+ CH1 = IN-

FIGUR 46 - KONFIGURATIONSBITS FOR MCP3008 - UDSNIT FRA MCP3008 DATASHEET

9.6. Software til ADC og Indlejret Linux System

Formålet med softwaren til systemet er, at det skal retunere integers værdier, for hvilke værdier ADC'en aflæser på Joystick, Trigger og PIRsensor. Da dette foregår med SPI kommunikation, skal dette sættes korrekt op på Indlejret Linux System. Til denne software er oprettet følgende klasse:



FIGUR 47 - SENSORSPI KLASSE

Softwaren til SPIkommunikation er opbygget i userspace. Dette kan lade sig gøre med de rette ioctl()
reqursts¹².

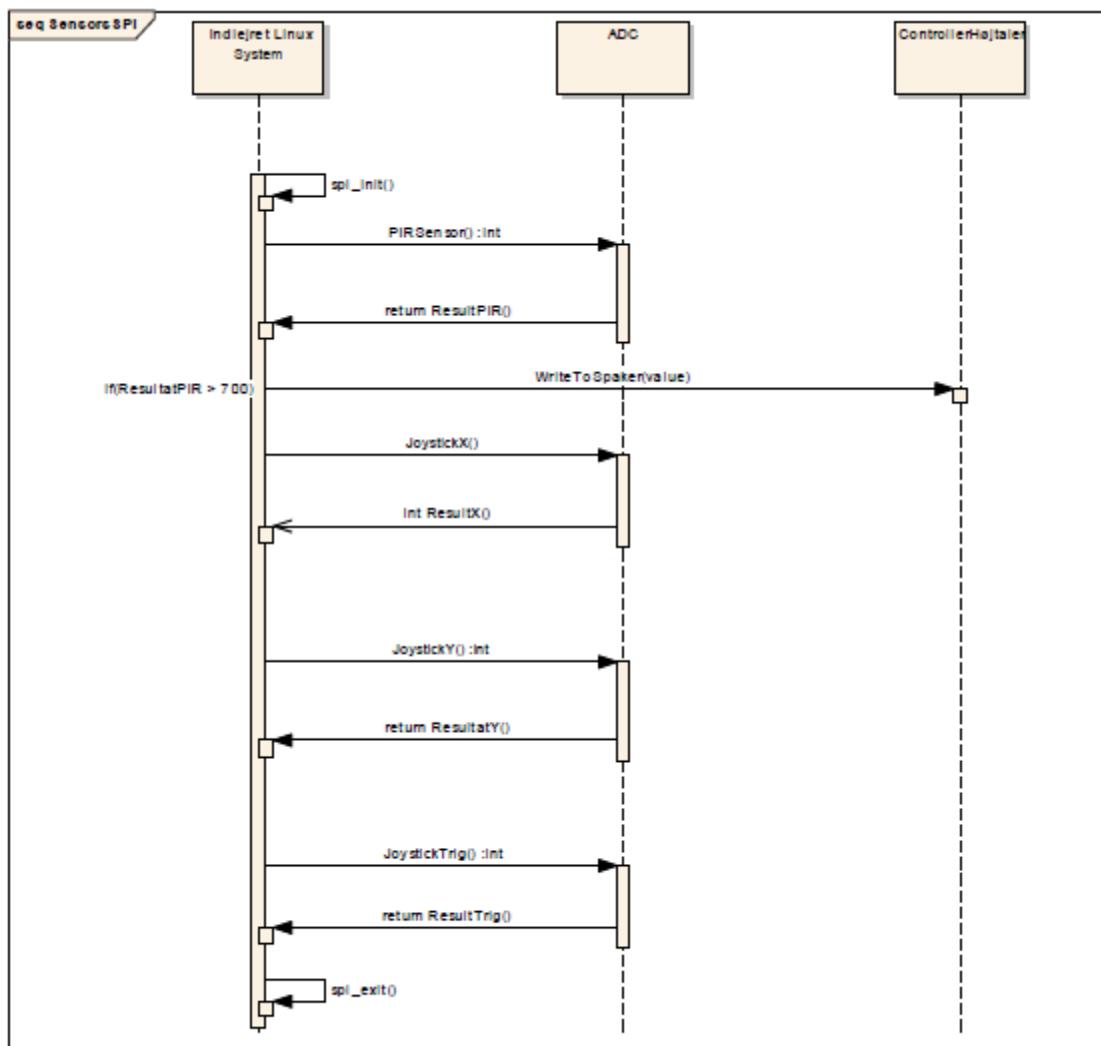
Private attributter

Det er blevet valgt, at det er nogle bestemte private attributter, der er tilføjet til SensorsSPIklassen:

`_spifd`: er en filedescriptor, denne skal åbne for interfacet med SPIdev.
`_mode`: Hvilken SPI mode der anvendes.
`_speed`: clock speed. Jo højere, jo bedre sampling ratio med ADC
`_bitsPerWord`, det antal af bits, der sendes af gangen til ADC.

Da ADC, Indlejret Linux System og Controllerhøjtaler alle arbejder sammen, er der lavet et sekvensdiagram, til beskrivelse og overskueligørelse af netop dette. Sekvensdiagrammet er lavet ud fra, hvordan SensorsSPI kommunikerer med ADC og Controllerhøjtaler. Herefter beskrives de enkelte funktioner i klassen SensorsSPI.

¹² <https://www.kernel.org/doc/Documentation/spi/spidev>



FIGUR 48 - SEQUENCE DIAGRAM

Det antages, at der er kommet et input på ADC'en fra JoystickX() aksen. Indlejret Linux System har oprettet et objekt, og kalder funktionen for JoystickX(). Internt i JoystickX() kaldes Transmission(), hvorfra selve overførelsen af data i mellem ADC og Indlejret Linux system hænder. Indlejret Linux System kalder funktionerne, og ADC retunerer et resultat.

Klassen består af to constructors. En default constructor og en overloaded constructor. Her kan man ændre clockhastigheden, SPIMODE, hvilken spichannel der anvendes på Indlejret Linux System og det antal bits man sender.

Klassen består af en funktion for hver channel, som der er aktiv på ADC'en. Det vil sige fire, da der er to outputs fra Joystick, et output fra PIRsensor og et output fra Trigger. Disse funktioner returnerer alle sammen en integer for hver funktion, for at de bliver lettere at implementere med andet software.

Spi_init står for initieringen af spidev/device på Indlejret Linux System. Det er vigtigt her, at private attributter sættes korrekt. Følgende kode er et udsnit af spi_init funktionen. Spi_init funktionen er blevet inspireret af¹³

```

int SensorsSPI::spi_init(std::string devspi)
{
    _spifd = open(devspi.c_str(), O_RDWR);
    int mod = _mode;
    int bits = _bitsPerWord;
    int speed = _speed;

    if(_spifd < 0)
    {
        printf("The SPIDevice was not able to be loaded\n");
        exit(1);
    }

    int check = -1;
    check = ioctl (_spifd, SPI_IOC_WR_MODE, &mod);
    if(check < 0)
    {
        printf("The mode was not able to be written\n");
        exit(1);
    }

    check = ioctl (_spifd, SPI_IOC_RD_MODE, &mod);
    if(check < 0)
    {
        printf("The mode was not able to be read\n");
        exit(1);
    }
}

```

FIGUR 49 - KODEUDSNIT AF SPI_INIT

_spifd er en fildescriptor, der gør at man kan tilgå </dev/spidev0.0> vha en fil pointer. Det er her spidev devicet opfamges. Der udprintes en fejlmeddelse, hvis dette ikke kan lade sig gøre. Herfra checkkes der for aflæsningen af diverse ioctl-funktioner, hvor den fx tager SPI_IOC_RD_MODE som parameter, samt filedescriptoren foruden en reference til mode. I ioctl(_spifd, SPI_IOC_RD_MODE, &mode) er dette for eksempel aflæsningen af, hvilken mode der er blevet anvendt, og om dette er godkendt. Kan dette ikke aflæses returneres en fejlmeddelse. Dette gentager sig i resten af koden¹⁴, hvor der også tjekkes for antal bits og clokhastiged, hvilket henholdsvis er _speed og _bitsPerWord.

Funktionerne for PIRsensor, Trigger og begge Joystick akser, er næsten ens. Følgende er et eksemel på, hvordan første del af sourcekoden til **JoystickY()** er blevet implementeret.

```

int SensorsSPI::JoystickY()
{
    int ResultY = 0;
    unsigned char Y[3];

    Y[0] = 0b00000001;
    Y[1] = 0b10010000;
    Y[2] = 0b00000000;
}

```

¹³ <http://mitchtech.net/raspberry-pi-arduino-spi/>

¹⁴ Se source koden i bilag. I mappen Sourcekode/SPI/SensorsSPI.cpp

FIGUR 50 - KODEUDSNIT AF JOYSTICKY()

Der oprettes et unsigned char array med tre pladser, der hedder Y[3]. For at initiere SPI kommunikationen skal Y[0] sættes til 0b0000001 som indeholder en start bit. Herefter skal der bestemmes, hvilken channel, der skal læses fra. Outputtet fra JoystickY channel er sat til ADC'ens channel 1, se Figur 44. Herfra kigges der i databladet, og man kan se at Y[1] skal sættes til 0b10000000, se Figur 46. Byte tre er don't care så Y[2] sættes til 0. Herfra kaldes Transmission funktionen.

Det er Y[1]-byten, der varierer fra de andre funktioner, da inputsene fra JoystickY(), PIRsensor og Trigger, går ind på andre channels på ADC'en. JoystickY går ind på channel 1, Trigger på channel 2 og PIRsensor på channel 3. Hvordan bitsene sættes til de andre channels kan ses på Figur 46. Følgende er en implementering af Transmission().

Transmission() står for transmissionen af data imellem Indlejret Linux System og ADC. Den tager som parameter, transmit. Dette er data fra fx JoystickY()'s char array Y[]. Derudover tager den bytes som parameter, da den skal vide, hvor mange bytes, der afsendes. Dette bliver til tre, da der afsendes tre bytes.

Transmission() sender Y[] til spidev device. Strukturen for, hvordan transmission skal foregå er blevet fundet her. Følgende link ligger til grund for selve transmission funktionen.¹⁵ Dette står også til grundlag for valgene af private members _speed og _bitsPerWord, da disse har med transmissionen at gøre - der skal fortælles clockspeed og hvor mange bits der sendes. Følgende kode, er et udsnit af Transmission(). Der henvises til bilaget, for den fulde source kode.

```
|int SensorsSPI::Transmission(unsigned char *transmit, int bytes)
{
    struct spi_ioc_transfer spi[bytes];
    int check = -1;
    int i = 0;

    bzero(spi, sizeof(spi));

    for (i = 0; i < bytes; i++)
    {
        spi[i].tx_buf      = (unsigned long)(transmit + i);
        spi[i].rx_buf      = (unsigned long)(transmit + i);
        spi[i].len         = sizeof(*(transmit + i));
        spi[i].delay_usecs = 0 ;
        spi[i].speed_hz    = this->_speed;
        spi[i].bits_per_word = this->_bitsPerWord;
        spi[i].cs_change   = 0;
    }
}
```

FIGUR 51 - KODEUDSNIT AF TRANSMISSION()

¹⁵ <http://www.cs.fsu.edu/~baker/devices/lxr/http/source/linux/include/linux/spi/spidev.h>

Her bliver transmit sendt til spidev device og transmit bliver sendt fra spidev device. Her angives også hvilke parametre der er blevet givet med hensyn til, hvilken gpio port, der er blevet valgt på indlejret linux system, hvor mange bits der sendes og clock hastighed. Herfra returneres transmit, der er blevet sendt fra ADC'en. Ovenstående kode sker for hver byte der sendes, det vil sige tre gange.

- tx_buf: Indholder en pointer til en userspace buffer, med det data, der skal overføres fra Indlejret Linux System.
- rx_buf: Indholder en pointer til en userspace buffer, med det data der skal modtages fra ADC.
- .len: Dette er længden af overførelsens mængde i bytes altså 3.
- .delay_uses: Delay efter sidste bit transfer. Denne sættes her til 0.
- .speed_hz: Clock hastighed.
- .bits_per_word: Hvor mange bits, der endes af gangen. Her tre bytes af otte bits.
- .Cs_change: Her kan device fravælges før næste transmission.

Fra SensorsSPI returneres resultatet fra transmission. Følgende kode er andet udsnit af JoystickY() funktionen.

```
Transmission(Y, sizeof(Y));
ResultY = ((Y[1] & 3) << 8 ) + Y[2];
cout << "the result for Y is" << ResultY << endl;

return ResultY;
```

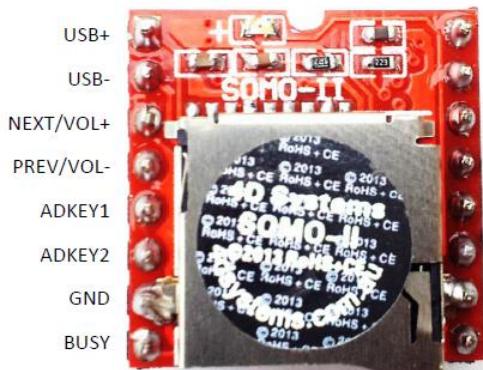
FIGUR 52 - ANDET UDSNIT AF JOYSTICKY()

Ovenstående kode viser det sidste udsnit af JoystickY funktionen. For at få det endelige resultat anvendes der lidt bit manipulation. Den første byte og den anden byte lægges sammen, hvorefter variablen ResultY sættes lig med det endelige resultat i decimaltal.

9.7. ControllerHøjtalere

For **Controllerhøjtalere** er der benyttet en **SOMO-II** (audio Sound Module 2). Enheden er forbundet til Indlejret Linux System samt til en højtalere (her en fjernsynshøjtalere) med to inputs. Formålet med SOMO-II er at afspille en lyd (mp3-fil på et microSD-kort eller UBS 2.0 flash drive med ekstra komponenter). For dette projekt er der valgt, at afspille en selvvalgt mp3-fil lagt ind på et tilkøbt microSD-kort.

SOMO-II er forbundet fra NEXT/VOL+ til Indlejret Linux System, GND og VCC er forbundet med Indlejret Linux System. SPK- og SPK+ er forbundet til Speaker-II.



FIGUR 53 - CONTROLLERSOMO-II



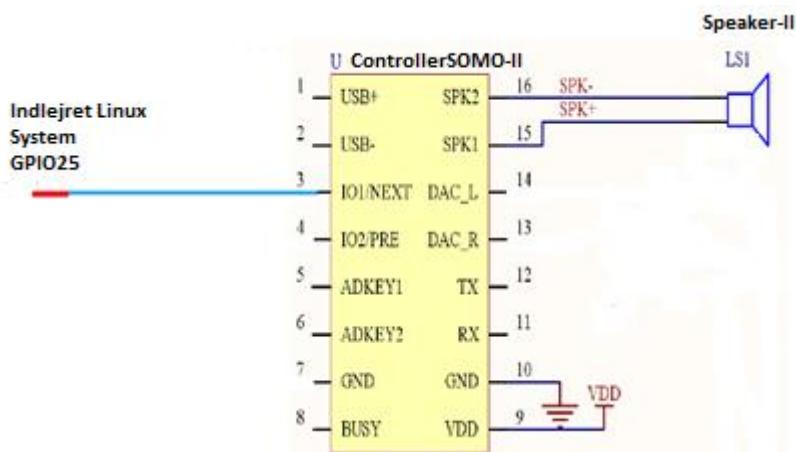
FIGUR 54 – SPEAKER II

Følgende billeder viser udseendet på SOMO-II og dens 16 ben samt den benyttede højtalere (bilhøjtalere):

Controllerhøjtalere benyttes indenfor til at oplyse brugeren om, at et dyr er registeret i PIRsensorens sigte indenfor en afstand på 7 meter. Brugeren kan derefter med Joystick navigere Home Defense Turret i retning af dyret, og derefter med affyre skud fra pistolen mod Offer ved at trykke på Trigger. Der blev overvejet kort at tilslutte et forstærkende kredsløb, men efter at have testet, hvor højt lyden blev afspillet ud af højtaleren, blev dette droppet.

9.7.1. Implementering af ControllerHøjtaller

Der er flere måder, hvorpå man kan fortælle, at SOMO-II skal afspille en lyd. Da UART pindene går til kommunikation med PSOC 4 i dette projekt, er der blevet valgt pin NEXT/VOL+. Når der sendes OV til ben 3 på controller-SOMO-II som vist på Figur 55. afspiller SOMO-II en lyd. Når der ikke afspilles en lyd, sættes GPIO25 fra Indlejret Linux System til 5V. Hvis man sender 0 til pin 3 i længere tid IO/NEXT på SOMO-II skrues der op for lyden. Derefter skal der kun sendes 0 i et kort interval, for ikke at skrue op for lyden. Bliver man ved med at sende 0, bliver den ved med at skrue op, og den afspiller ikke nogen lyd.



FIGUR 55 - CONTROLLERHØJTALER PINLAYOUT

9.7.2. Software til ControllerHøjtalere og PIRsensor

ControllerHøjtaleren skal afspille en lyd, når PIRsensoren opfanger en bevægelse. Derfor er kode for ControllerHøjtalere modulet inkapslet i SensorsSPI. Når PIRsensor opfanger bevægelse sættes GPIO25 på Inlejret Linux System til 0 i 200ms. Herefter sættes den høj. Dette får ControllerHøjtalere til at afspille en lyd.

I SensorsSPI klassen, er der implementeret en funktion, der sørger for, at der afspilles en lyd til Controllerhøjtaleren. Funktionen tager value som parameter. Når value sættes til nul afspilles der en lyd.

```
void SensorsSPI::WriteToSpeaker(int value)
{
    digitalWrite(26, value);
}
```

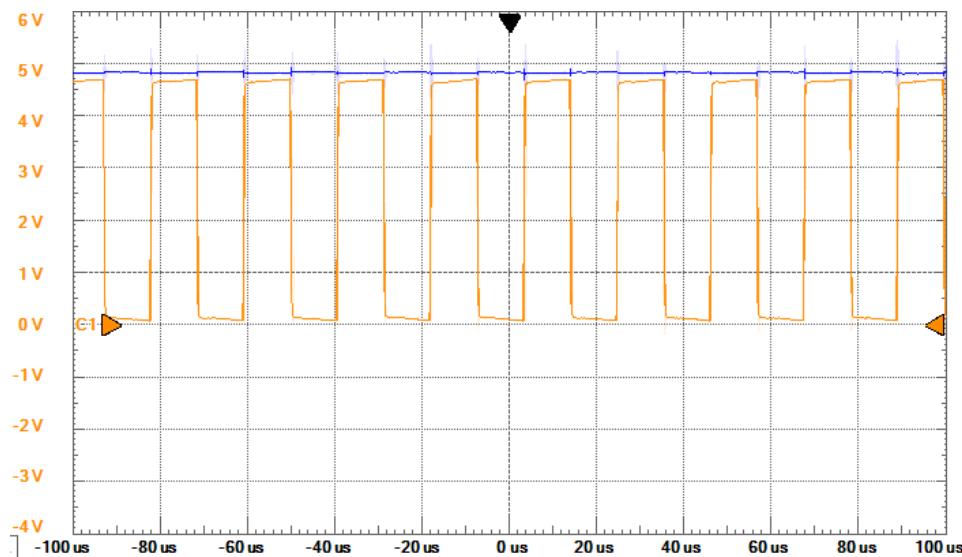
FIGUR 56 - WIDGETSPEAKER()

I Joystick klassen skrives der til denne højtalere afhængigt af, om der er blevet informeret, at der har været bevægelse ved PIRsensoren. Der henvises til afsnit 11.3.9 og 11.3.10 om implementering af Joystick klassen.

10. Test af hardware

10.1. Test af Motorstyring Hardware [ATT]

Når idéen bag motorstyringen var blevet implementeret på et fumlebræt, var det klart til at blive testet. Så var det bedst at koble det til en spændingsgenerator og starte med at teste ved en lav spænding. Det virkede ved første forsøg. Det eneste problem var at man skal skru op for strømmen for at skifte retningen i Finder relæ'et til cirka 60mA til at få den skifte. Der viste det sig at PsOC ikke kan klare det at skifte retningen ved direkte kobling. Så det var nødventligt at bruge en MOSFET, som lader forsyningens spændingen gå til retningskiftet på Finder relæ'et. Der blev testet med 14V på indgangen og når MOSFET'en fik 5V på gate, kunne der måles 14V på motor indgangen med et multimeter. Til at teste PWM signalet og retningsskiftets signal, blev der brugt et Analog Discovery scope og målt på udgangen af PsOC'en. Derefter var PWM signalet programmeret til den halve periode tid, nu måles der med scop'et om det passer med det ønskede PWM signal, samt med det logiske high til retningsskifts pinden, som det gjorde. Dette kan ses på Figur 57.



FIGUR 57 - MÅLING AF PWM OG RETNINGS SKIFT PÅ ANALOG DISCOVERY.

Ved testen af den infrarøde LED og fotodiode, var idéen den samme. Systemet var opbygget på et fumlebræt og derefter målt med et multimeter. Når man brugte en formodstand foran LED'en, fungerede det fint, men afstanden mellem LED'en og fotodioden før det gav et udslag, var meget lille. Da modstanden blev fjernet, var der en meget bedre rækkevidde, og LED'en kunne godt klare dette. Når sensoren var koblet til PSoC'en var der et belastningsproblem, men så blev der målt et højt spændingsfald over fotodioden. Det blive ordnet ved at ændre funktionen af den input pin som PSoC'en har. Det var som sagt "Learn by doing" methoden.

10.2. Test af Uart på Platform [ATT]

Testen af UART'en blev relativ simpelt. For eksempel når kommunikationen til SOMO-II'en blev testet, var den koblet til og det blev sendt en kommando. Det viste sig så ikke at fungere. Til at finde ud af hvad problemet var, blev der brugt en "Silicon Labs CP210x USB to UART bridge".



FIGUR 58- SILICON LABS CP210x USB TO UART BRIDGE

Den kan kobles til en computer og aflæse UART kommandoer via Tera Term. Tera Term viser kun ASCII værdier og de hex værdier som skal sændes til SOMO-II'en har char værdier som kan ikke vises. For eksempel 0x06 som svarer til ASCII værdien ACK, som ikke vises i Tera Term. Så blev der forsøgt med tilfældige ASCII værdier og der viste det sig at den funktion som blev brugt, sprang nogle af de hex værdier som blev sendt ud over. Det der bør skrives ud er "what is up", som vist på Figur 59 og Figur 60.



FIGUR 60 – STRING SENDES MED FORKERT FUNKTION



FIGUR 59 - STRING SENDES MED RIGTIGE FUNKTION

Samme fremgangsmåde blev brugt til teste kommunikationen mellem PSoC4 og det Indlejerede Linux System. Igen, så blev de to systemer koblet sammen og det blev forsøgt at sende en af de ønskede kommandoer. Igen, virkede det ikke. Det var derfor bedst at starte ved source'en og aflæse hvad der kommer ud fra det Indlejerede Linux System. Det blev gjort med den samme UART til USB bridge og aflæst i Tera Term. Nu var problemet der ikke. Nu var det nødvendigt at programmere PSoC4'en til at sende alt hvad den aflæser ud. Det var meget ustabilt og rækkefølgen var meget forkert, når en string blev sent over UART. Det blev ordnet med en funktion som PSoC4'en har (læs UART under Platform Software). Se afsnit 9.3 for yderligere information.

10.3. Test af Software på Platform [ATT]

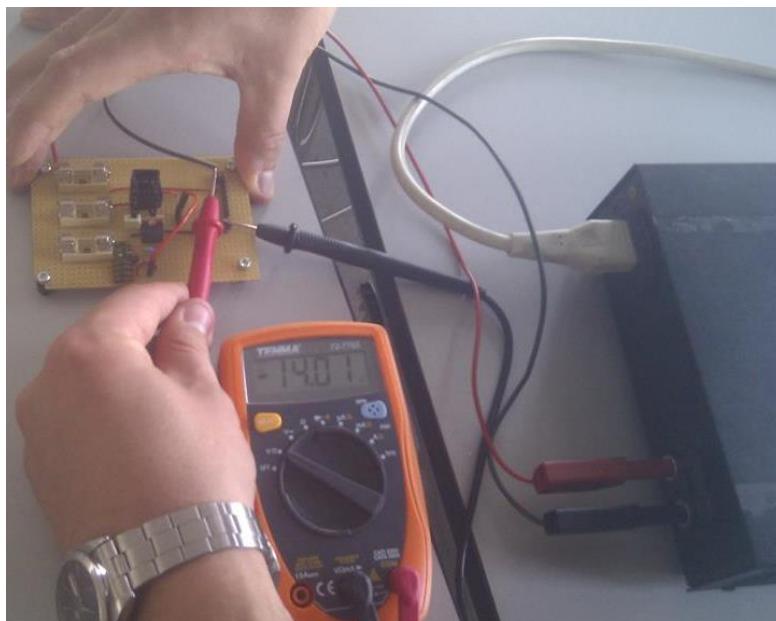
Ved test af softwaren blev der testet meget igennem et scope, da de fleste af funktionerne som bliver brugt skulle sendes ud på en pin, som kan aflæses direkte. På grund af at softwaren er meget afhængige af hardwaren, var det nødvenligt at scopet var koblet til PSoC4'en mens man testede det. Efter at have ordnet problemet med UART'en var softwaren klar til test. Det var udført på en sådan måde at det Indlejerede Linux System sente den ønskede kommando, for eksempel kommandoen V, til at få horisontal motoren til at køre til venstre. Når PSoC4'en har modtaget kommandoen, var forventningen at pinden som skal give et PWM signal, giver det ønskede signal. Dette fungerede i første forsøg. Da det ikke var alt hardwaren, som var helt klart, blev der kørt Debug mode og hver enkelt kommando er blevet trukket hvert for sig, og koden blev tjekket for om den gjorde som forventet. Når Dødschich'en blev testet, blev den koblet til "Digital Input" pin'en, som skal aflæse den. Herefter blev Debug mode afviklet. Når fotodioden

ikke var placeret over LED'erne, viste det sig at den aflæste værdi blev logisk low, og så snart fotodioden var over LED'erne fik den en logisk high.

10.4. Test af spændingsreguleringsprint [LRA]

Til at teste spændingsreguleringsprintet er der lavet to opstillinger.

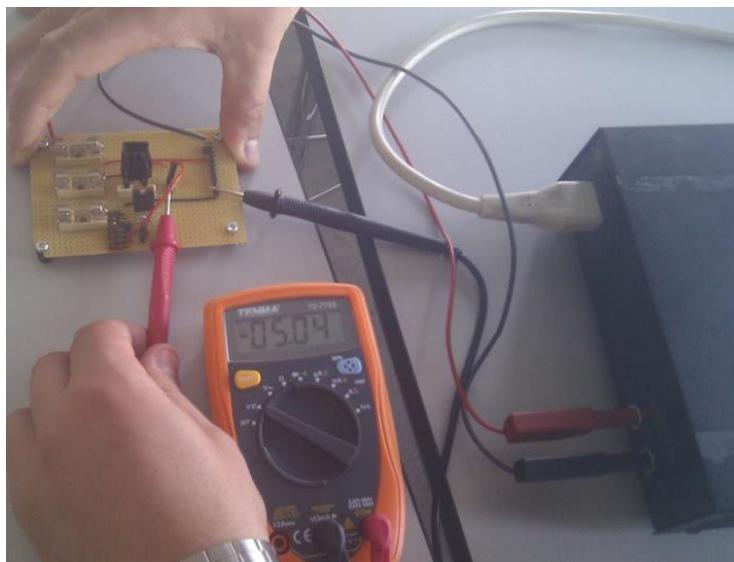
På den første test måles 14V forsyningen. Tilsæt 14V transformeren til en 230AC forsyning(stikkontakt) og isæt det røde kabel fra printet i transformerenes positive udgang og det sorte kabel i den negative udgang. Placer et voltmeter på printets 14V ben og grund.



FIGUR 61 MÅLING AF 14V SPÆNDINGSFORSYNING

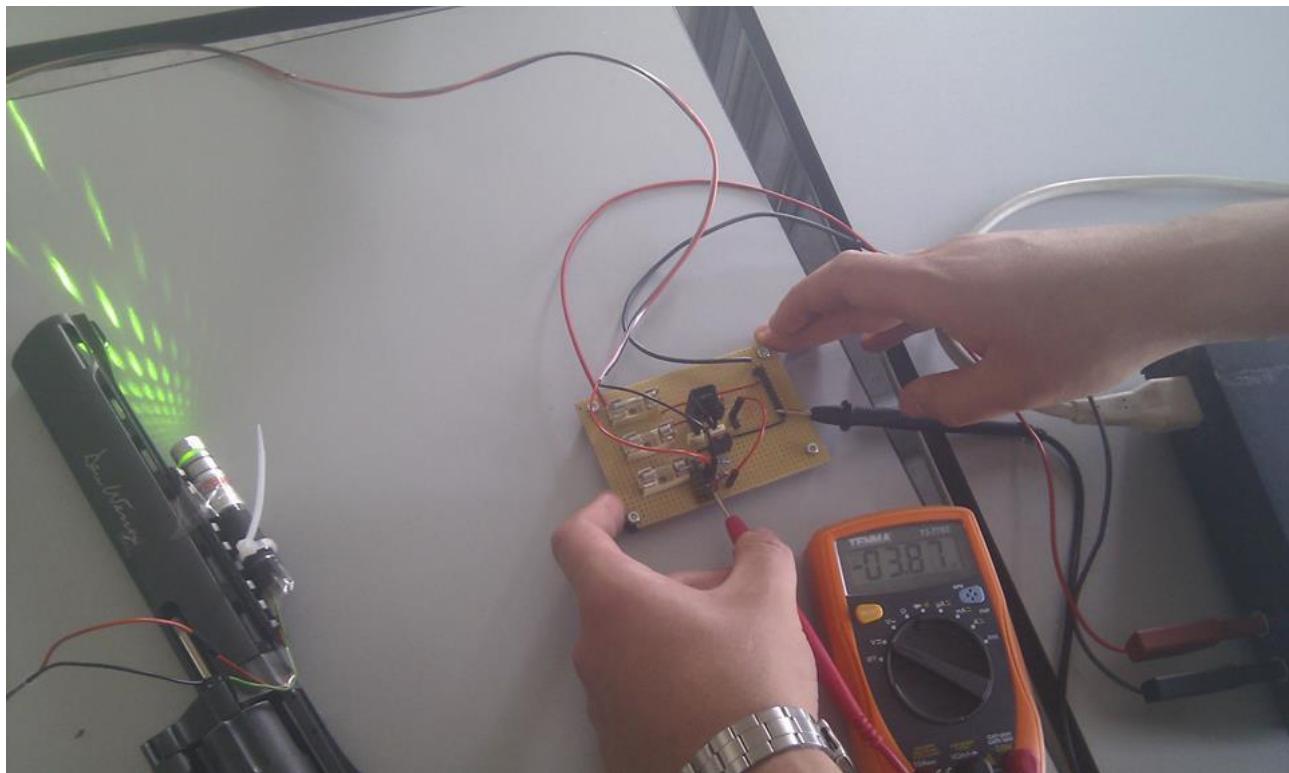
Som det målte resultat er 14V som forventet. Se Figur 61.

Til den anden test bruges samme opstilling som ved 14V, men voltmeteret bliver flyttet over på de tre 5V ben. Se Figur 62.



FIGUR 62 MÅLING AF 5V SPÆNDINGSFORSYNINGER

Til den tredje test bruges samme opstilling som ved de andre tests, men der bliver også tilføjet en ledning fra et af de tre 5V ben, til at aktivere transistoren på dens ben 1. Laseren placeres på de to ben rød ledning på benet længst til venstre sort ledning på benet til højre (se Figur 63). Et voltmeter placeres nu mellem laserens indgangsben og ground.



FIGUR 63 MÅLING AF 3,7V SPÆNDINGSFORSYNING.

Handling	Forventning	resultat
Sæt ledningen mellem transistor ben 1 og 5V ben i.	Voltmeter måler 3,7V	Voltmeter måler 3,8V
Fjern ledningen mellem transistor ben 1 og 5V ben.	Voltmeter måler 14V	Voltmeteret måler 14V

TABEL 18 - SPÆNDINGSFORSYNING, 3,7V

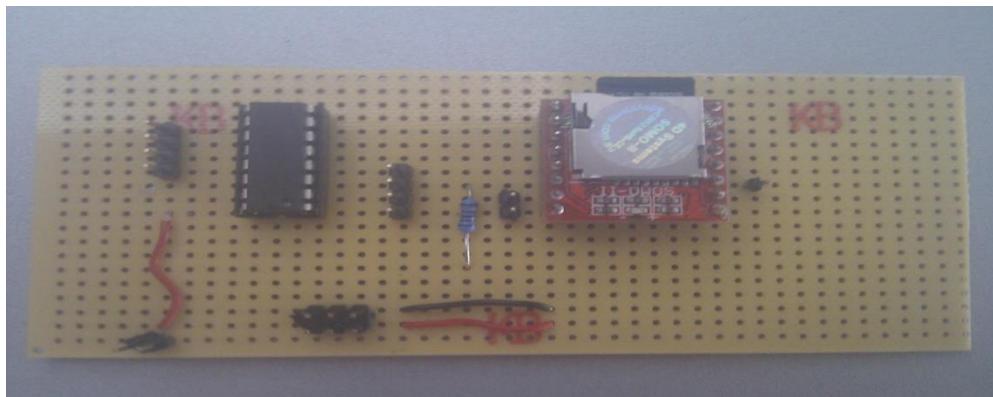
Til sidst bliver der lavet en holdbarhedstest hvor alle ting er tændt samtidigt.

Handling	Forventning	resultat
Alle systemer er tændt i 5 minutter	Forsyningsprintet fortsætter med at virke som det skal.	Forsyningsprintet fortsatte med at virke som forventet, men modstandene for 3,7V forsyningen er blevet lune.
Alle systemer er tændt i 15 minutter	Forsyningsprintet fortsætter med at virke som det skal.	Forsyningsprintet fortsatte med at virke som forventet, men modstandene for 3,7V forsyningen er blevet varme.
Alle systemer er tændt i 30 minutter	Forsyningsprintet fortsætter med at virke som det skal.	Forsyningsprintet fortsatte med at virke som forventet, men modstandene for 3,7V spændingsforsyningen fortsætter med at have ca samme temperatur.

TABEL 19 – HOLDBARHEDSTEST

10.5. Kommunikation mellem ADC og Indlejret Linux System [SF, KB]

PIRSensor, ControllerHøjtaler, Trigger og Joystick er tilsluttet følgende kredsløb. Enhedstest for følgende afsnit er alle lavet på dette kredsløb. For at se, hvordan der er forbundet med Indlejret Linux System henvises til Figur 44.

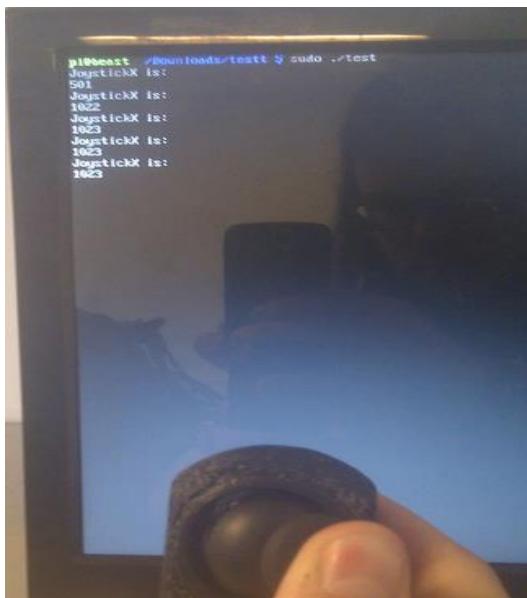


FIGUR 64 - KREDSLØB FOR CONTROLLERHØJTALER OG ADC

10.5.1. Test af Joystick's x-akse

Test	JoystickX()
Klasse	SensorsSPI
Test beskrivelse	Funktionen JoystickX() testes med et testprogram. Testprogrammet printer resultatet ud på skærmen, så testen kan valideres. Formålet med funktionen er, at den skal kunne udskrive de rigtige værdier, alt afhængigt af hvor langt Joystick bevæges horisontalt. I denne test testes der både for den maksimale og minimale værdi, Joystick's x-akse kan udskrive.
Input	Fysisk input på Joystick, der går over på channel 0 på ADC, der sender ud til Indlejret Linux System.
Output	Der modtages en integer fra 0 til 1023.
Forventet Resultat	Der udprintes 0 ved den laveste værdi og der modtages 1023 ved den højeste værdi.

TABEL 20 - TEST AF JOYSTICK X-AKSE



FIGUR 66 - JOYSTICKX() MAKSIMAL VÆRDI



FIGUR 65 - JOYSTICKX() MINIMAL VÆRDI

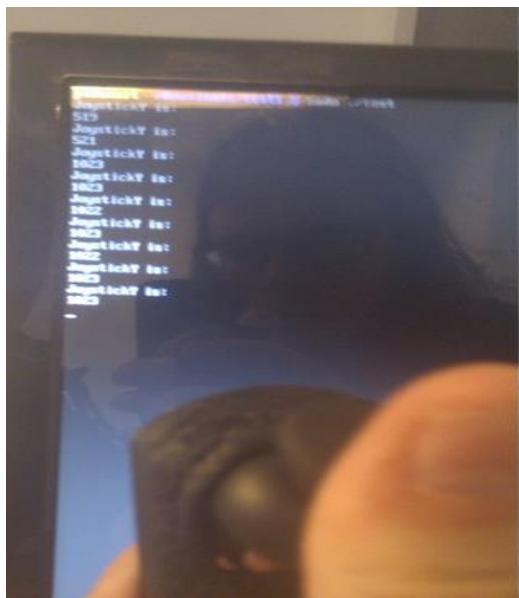
Resultat	Som der kan ses på Figur 66, vises der en værdi på 1023, når Joystick er kørt længst mod højre – OK! Som der kan ses på Figur 65, vises der en værdi på 0, når Joystick er kørt længst mod venstre – OK!
----------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

TABEL 21 - RESULTAT AF TEST AF JOYSTICK'S X-AKSE

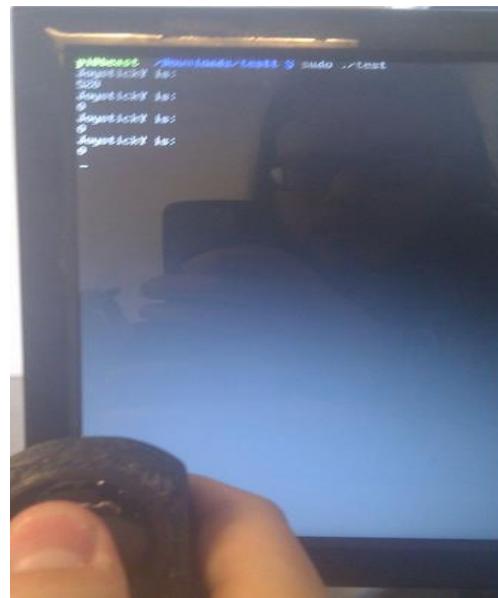
10.5.2. Test af Joystick y-akse

Test	JoystickY()
Klasse	SensorsSPI
Test beskrivelse	Funktionen JoystickY() testes med et testprogram. Testprogrammet printer resultatet ud på skærmen, så testen kan valideres. Formålet med funktionen er, at den skal kunne udskrive de rigtige værdier, alt afhængigt af hvor langt Joystick bevæges vertikalt. I denne test testes der både for den maksimale og minimale værdi, Joystick's y-akse kan udskrive.
Input	Fysisk input på Joystick, der går over på channel 1 på ADC, der sender ud til Indlejret Linux System.
Output	Der modtages en integer fra 0 til 1023.
Forventet Resultat	Der udprintes 0 ved den laveste værdi og der modtages 1023 ved den højeste værdi.

TABEL 22 - TEST AF JOYSTICK'S Y-AKSE



FIGUR 68 - JOYSTICKY() MAKSIMAL VÆRDI



FIGUR 67 - JOYSTICKY() MINIMAL VÆRDI

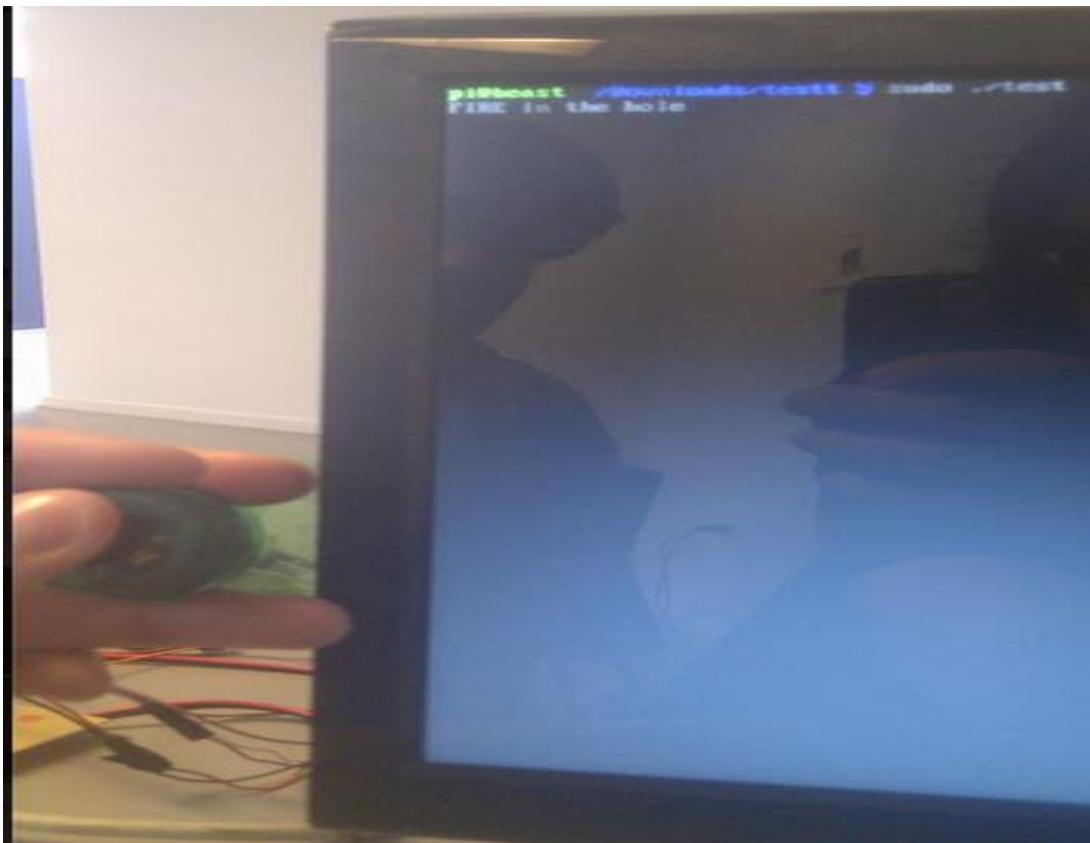
Resultat	Som der kan ses på Figur 68, vises der en værdi på 1023, når Joystick er kørt længst mod højre – OK! Som der kan ses på Figur 67, vises der en værdi på 1023, når Joystick er kørt længst mod venstre – OK!
----------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

TABEL 23 - RESULTAT AF TEST AF JOYSTICK'S Y-AKSE

10.5.3. Test af Trigger

Test	JoystickTrig()
Klasse	SensorsSPI
Test beskrivelse	Funktionen JoystickTrig testes med et testprogram. Testprogrammet printer resultatet ud på skærmen, så testen kan valideres. Formålet med funktionen er, at den skal kunne udskrive de rigtige værdier, alt afhængigt af om der er blevet trykket på Trigger. Test programmet er sat til at udskrive "Fire in the hole", hvis Indlejet Linux System modtager en værdi fra JoystickTrig() over 1000.
Input	Fysisk input på Trigger, der går over på channel 2 på ADC, der sender ud til Indlejet Linux System.
Output	Der modtages en værdi på 1023.
Forventet Resultat	Der udprintes en linje tekst "Fire in the hole", da der er samplet en værdi på 1023.

TABEL 24 - TEST AF TRIGGER



FIGUR 69 - TEST AF TRIGGER

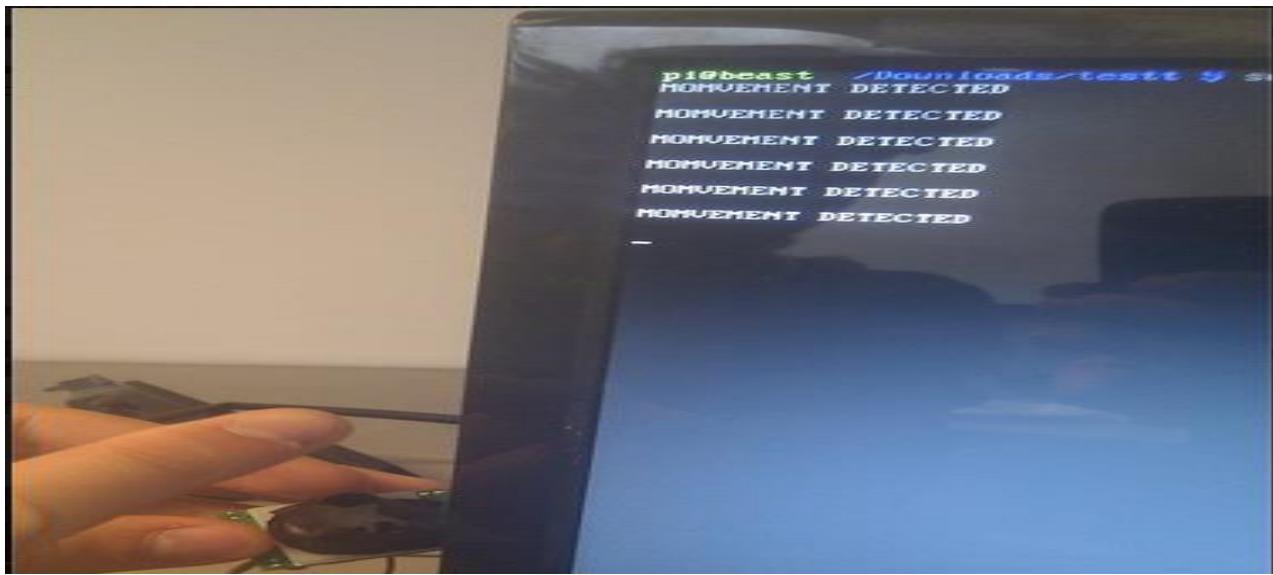
Resultat	Som der kan ses på Figur 69, udskrives der "Fire in the hole" på skærmen. – OK!
----------	---------------------------------------------------------------------------------

TABEL 25 - RESULTAT AF TRIGGER TEST

10.5.4. Test af PIRsensor

Test	Pirsensor()
Klasse	SensorsSPI
Test beskrivelse	Funktionen Pirsensor() testes med et testprogram. Testprogrammet printer resultatet ud på skærmen, så testen kan valideres. Formålet med funktionen er, at den skal kunne fortælle om der er bevægelse afhængigt af, om PIRsensor opfatter dette. Test programmet er sat til at udskrive "Movement detected", hvis Indlejret Linux System modtager en værdi fra Pirsensor over 700.
Input	PIRsensor opfanger bevægelse. Herfra sender den et signal på channel 3 af ADC.
Output	Signal fra ADC til Indlejret Linux System.
Forventet resultat	Der udprintes en linje tekst "Movement Detected", da der er samplet en værdi over 700.

TABEL 26 - TEST AF PIRSENSOR



FIGUR 70 - TEST AF PIRSENSOR

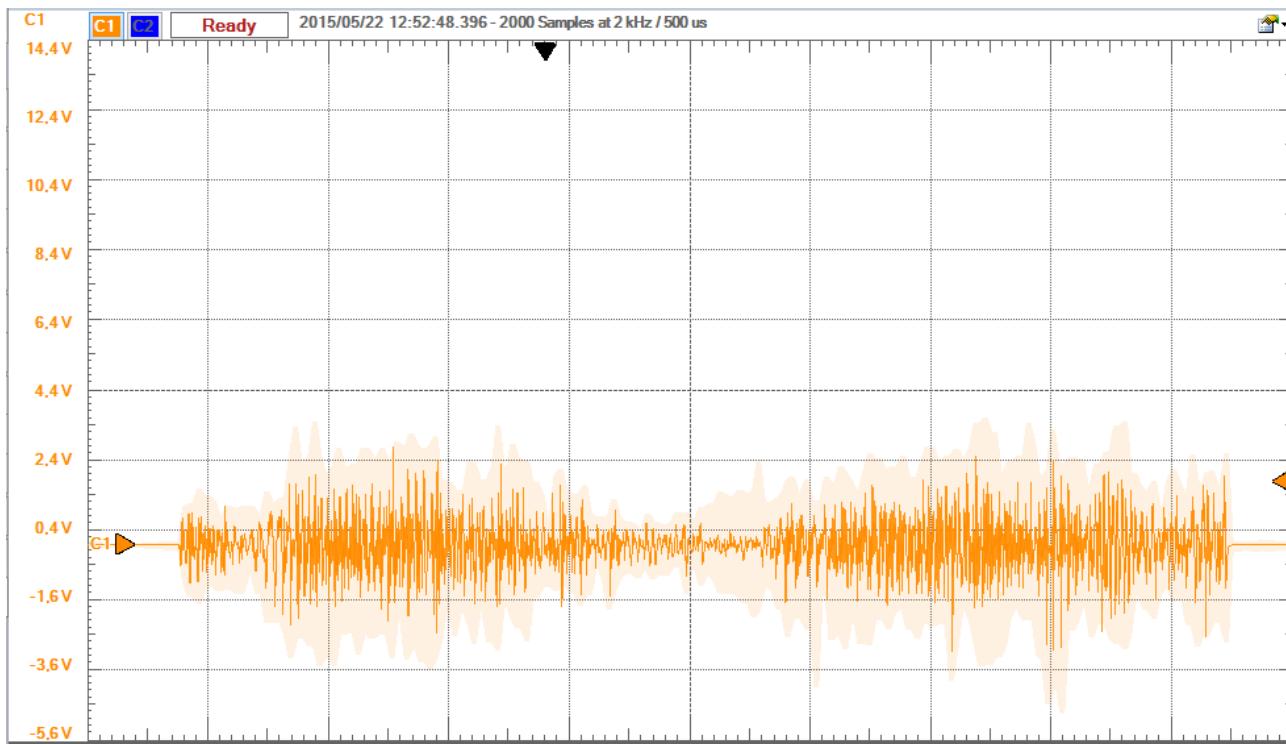
Resultat	Som der kan ses på Figur 70, udskrives "Movement Detected", når PIRSensor opfanger bevægelse. – OK!
----------	-----------------------------------------------------------------------------------------------------

TABEL 27 - RESULTAT AF PIRSENSOR TEST

10.5.5. Test af ControllerHøjtalere

Test	ControllerHøjtalere
Test beskrivelse	ControllerHøjtaleren, skal afspille en lyd, når PIRsensoren opfanger en bevægelse. Derfor testes der for, om ControllerHøjtaleren afspiller en lyd, når fornævnte er hændt. Der testes med et oscilloscope, på udgangen af ControllerSOMO-II på signalerne Spkm & spkp. Se Figur 71.
Input	PIRSensor opfanger bevægelse. Der sendes 0 i 200ms fra Indlejret Linux System til ControllerSOMO-II.
Output	Der sendes et lydsignal til Speaker-I I, og der afspilles lyd.
Forventet resultat	På oscilloscopbilledet kan der ses et udfald, der skal forestille at være et lydsignal – Eller rettere sagt et PWM signal.

TABEL 28 - TEST AF CONTROLLERHØJTALER



FIGUR 71 - OUTPUT FRA CONTROLLERSOMO-II TIL SPEAKER-II

Resultat	Som der kan ses på Figur 71, er der blevet afspillet en lyd –OK
----------	-----------------------------------------------------------------

TABEL 29 - RESULTAT AF CONTROLLERHØJTALER TEST

11. Design og implementering af software [JDA, AEL, DT]

Dette afsnit indeholder design og implementering for systemets software. Alt softwaren beskrevet i dette afsnit, køres på det Indlejrede Linux System.

11.1. Samlet klassediagram

På næste side, kan ses et samlet klassediagram for alle de klasser, der er beskrevet i dette afsnit. Klassediagrammet er så stort, at det er printet på et A3-ark og tilskåret. Det har derfor sin egen dedikerede side, så der er plads til alle klasserne.

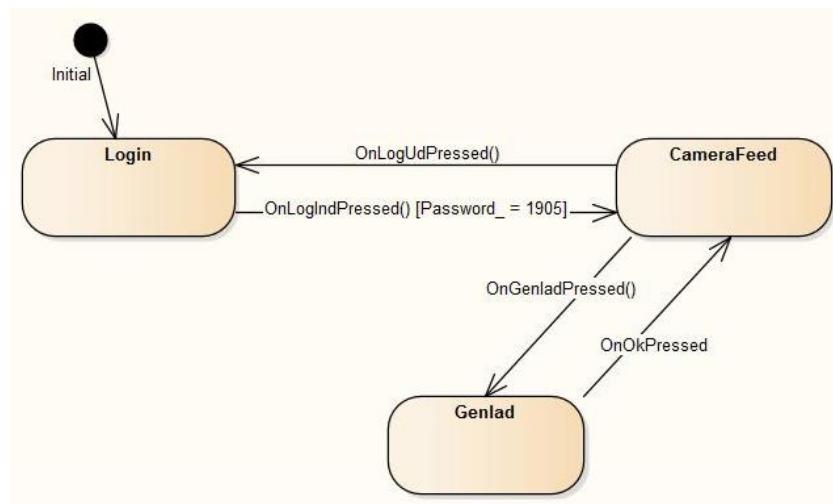
11.2. GUI [AEL]

Til dette projekt blev det fremsat som et krav at der skulle være et *brugerinterface*. Dette viste sig at blive mere omfattende som tingene skred frem og tog mange former fra start til slut. Det blev rimeligt tidligt vedtaget at der skulle bruges Qt da dette efter sigende skulle virke på de fleste enheder og da der heller ikke var nogen der havde de store erfaringer med en raspberry pi som var blevet brugt som projektets Indlejrede Linux System. Der blev hurtigt lavet udkast til brugerinterfacet men det skulle vise sig at en raspberry pi ikke understøtter versioner af Qt senere end Qt4, hvor den nyeste var Qt5. Dette var et stort tilbageslag i og med selve måden der kodes og hvordan vinduer implementeres er vidt forskellig i disse to versioner og derfor skulle alt laves om. Det endte dog med at et nyt brugerinterface blev lavet og dette endte også i det endelige system.

Lykken var dog ikke gjort da der igen opstod problemer da kamerabillederne, der skulle implementeres i Qt4 vinduet, ikke var lige til at få indsats. Dette var der ikke taget højde for da det havde forekommet nemt med Qt's egen QCamera funktion men den blev først understøttet i Qt5. Derfor skulle der findes et API til at håndterer kamerabillederne og dette blev OpenCV. Dette var langt fra simpelt, men langt bedre end alternativet. Der opstod utallige problemer med at få linket til OpenCV's biblioteker både på vores host, men også senere på det Indlejrede Linux System. Der gik derfor lang tid før *brugerinterfacet* overhovedet kunne compiles og ingen vidste derfor hvor langt projektets *brugerinterface* var nået.

Da der endelig blev linket korrekt til de forskellige biblioteker virkede *brugerinterfacet* forholdsvis hurtigt og de første funktioner kunne implementeres. Der havde på dette tidspunkt været så mange problemer med kamerabillederne at det blev vurderet at den bedste løsning ville være at der herfra blev bygget et *brugerinterface* "omkring" kamerabilledet i stedet for at lade kamerabillederne være en klasse for sig selv som først antaget. Dette fungerede dog ganske udemærket og funktionelt var der ingen forskel og dermed kunne dette hurtigt implementeres.

Nedenfor på Figur 72 ses et simplificeret statemachine diagram der viser brugerinterfacets vinduer og hvordan systemet styrer rundt imellem disse.



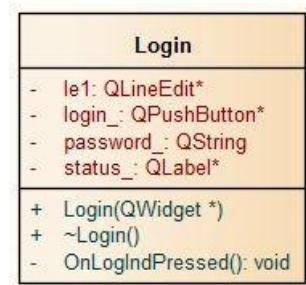
FIGUR 72 - STATE MACHINE DIAGRAM OVER DE FORSKELLIGE VINDUER

11.2.1. Login

Denne klasse er den første man møder når programmet starter. Login klassen fungerer som en lås for resten af systemets funktionalitet og er derfor udstyret med en adgangskode for at man kan skifte vindue. Der var fra start tale om at implementere en database indeholdende brugerinformationer og separate adgangskoder. Det blev dog vurderet at systemet som prototype ikke skulle have adskillige brugere og det blev derfor ikke set som nødvendigt at lave en database. Adgangskoden blev i stedet hardcoded ind i sourcekoden for at skabe funktionaliteten ved en adgangskode og samtidig kunne have større fokus på andre aspekter af projektet.

Adgangskoden blev "1905" og for at trykke på "Log Ind" knappen skal man bruge hashtag/firkant på matrix keyboardet.

FIGUR 73 - LOGIN KLASSE



Metoder:

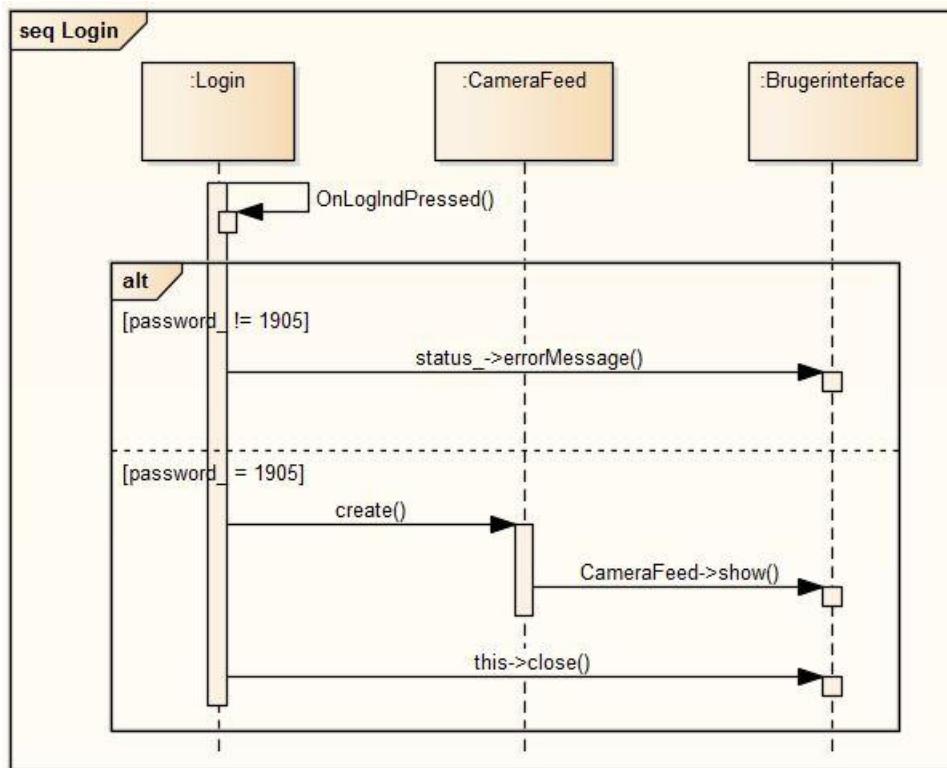
Constructor:

Her oprettes en QPushButton ved navn "Log Ind" og der skabes en forbindelse fra denne til funktionen OnLogIndPressed(). Derudover oprettes en QLineEdit som skal indeholde det data der bliver indtastet.

void OnLogIndPressed():

Denne funktion vurderer om den indtastede data matcher med variablen password_ som indeholder den korrekte kode. Hvis dette er tilfældet oprettes et objekt af klassen CameraFeed, der åbner et nyt vindue, hvorefter Login vinduet lukkes og objektet destrueres.

Hvis ikke den indtastede data matcher koden vil der blive sendt en string til variablen status_ med en fejlmeldelse.



FIGUR 74 - SEKVENDIAGRAM OVER ONLOGINDPRESSED()

11.2.2. CameraFeed

CameraFeed klassen var i første omgang skabt til at håndtere billeder fra kameraet og få sendt disse ud til *brugerinterface*. Dette viste sig dog at være en større opgave i og med at der var store problemer, med kinectens biblioteker som blev oplevet som ringe dokumenterede og ellers generelt indforståede og overkomplicerede. Der blev besluttet at bruge et Microsoft Lifecam, som var et ganske almindeligt usb baseret webcam. Dette muliggjorde at der kunne bruges OpenCV til at håndtere kameraets billeder og derefter få disse omdannet til images som Qt4 kunne håndtere.

Dette blev dog ikke enden for CameraFeed klassen. Der viste sig igen at opstå problemer når Qt4 skulle indkorporere OpenCV vinduet. Løsningen blev at CameraFeed blev en hovedklasse og indeholder nu det meste af funktionaliteten fra projektets visuelle brugerinterface som knapper og lignende.

Efter CameraFeed klassen også blev fundament for "hovedvinduet" blev der tilføjet fem knapper: Aktiver, Deaktiver, Genlad, Advarsel og Log ud. Derudover en statustekst samt et tekstlabel der fortæller om antallet af skud våbnet indeholder.

CameraFeed	
-	advarsel_: QPushButton*
-	aktiver_: QPushButton*
-	capture: CvCapture*
-	deaktiver_: QPushButton*
-	feed_: QLabel*
-	genlad_: QPushButton*
-	logud_: QPushButton*
-	msg_: QLabel*
-	sstat_: QLabel*
-	text_: QLabel*
-	timer_: QTimer*
+	cameraFeed(QWidget*)
+	~cameraFeed()
#	keyPressEvent(QKeyEvent*): void
-	OnAdvarselPressed(): void
-	OnAktiverPressed(): void
-	OnDeaktiverPressed(): void
-	OnGenladPressed(): void
-	OnLogUdPressed(): void
+	putImage(const Mat&): QImage
-	updatePicture(): void

FIGUR 75 - CAMERAFEED KLASSE

Der kommer løbende et input fra systemets PIR sensor som vil igangsætte en alarm hvis den opfanger bevægelse. ”Aktiver”/”Deaktiver” knappen sørger for henholdsvis at aktivere og deaktivere dette input så der på denne måde f.eks. ikke kan lyde en alarm hvis knappen ”Deaktiver” er blevet trykket.

Klassen holder også styr på hvor mange skud der er tilbage i våbnet og viser, som tidligere nævnt, dette antal på brugerinterfacet. Når våbnet bliver affyret vil dette tal tælle ned. Ved nedlæggelse af CameraFeed objektet skrives antallet af skud til en .txt fil, med det nuværende antal skud. Dette betyder derfor at ved oprettelse kan et nyt objekt læse fra denne fil og dermed huske hvor mange skud våbnet stadig indeholder. Når ”Genlad” knappen trykkes vil en klasse af dette navn oprettes, hvilket man også kan læse mere om senere i dette afsnit.

Hvis brugeren vælger at trykke på knappen ”Advarsel” vil klassen sende et signal ud til platformhøjtaleren der er placeret ude ved våbnet. Denne vil så afspille en advarselstone og dermed afskræmme eventuelle vilde dyr og lignende for på den måde at undgå at affyre våbnet så vidt muligt.

Når brugeren har udført sit ærinde ved systemet kan han trykke på ”Log ud” knappen og dette vil medføre at ”hovedvinduet” med CameraFeed objektet lukkes ned og Login vinduet vises igen og dermed låser systemet.

Metoder:

Constructor:

Opretter Qlabels og Qpushbuttons og forbinder disse til deres respektive slots og funktionaliteter. Opretter derudover forbindelse til et hvilket som helst camera i systemet og har til dette en timer der initierer en ny læsning af billeder herfra hvert 50. millisekund.

Destructor:

Frigiver cameraet, joystikket og sletter uartQueue. Åbner derefter ”AntalSkud.txt” filen og noterer hvor mange skud der er i våbnet og lukker derefter filen igen.

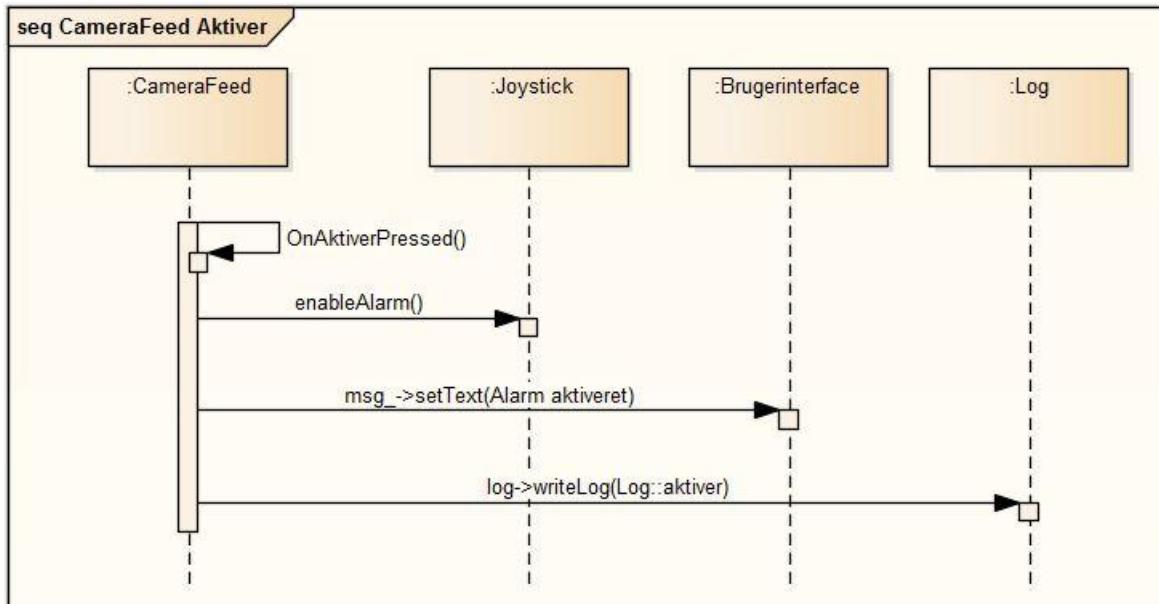
Qimage putImage(const Mat& mat):

Denne funktion er hentet fra nettet¹⁶ da det at omdanne en OpenCV IplImage type til Qt4’s QImage var en større udfordring end først antaget.

¹⁶ Se <http://stackoverflow.com/questions/11543298/qt-opencv-displaying-images-on-qlabel>

void OnAktiverPressed():

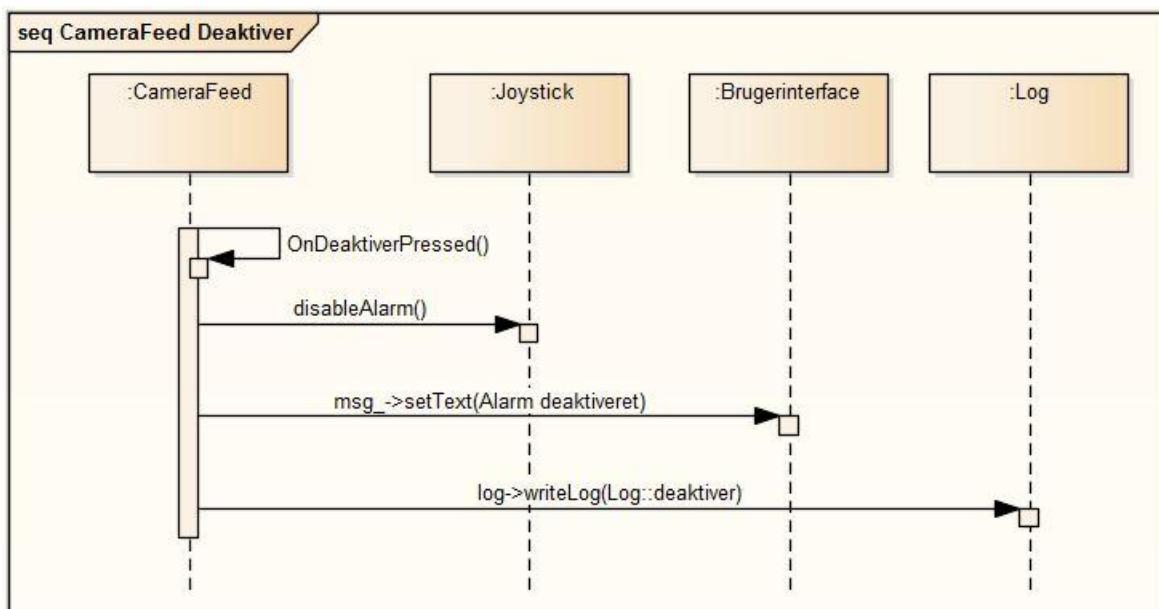
Denne funktion er forbundet til knappen "Aktiver" i brugerinterfacet og ved et klik på denne vil systemet aktivere inputs fra PIR sensoren og dermed muliggøre at alarmen kan lyde hvis sensoren opfanger bevægelse. Dette noteres desuden i log filen.



FIGUR 76 - SEKVENDIAGRAM OVER ONAKTIVERPRESSED()

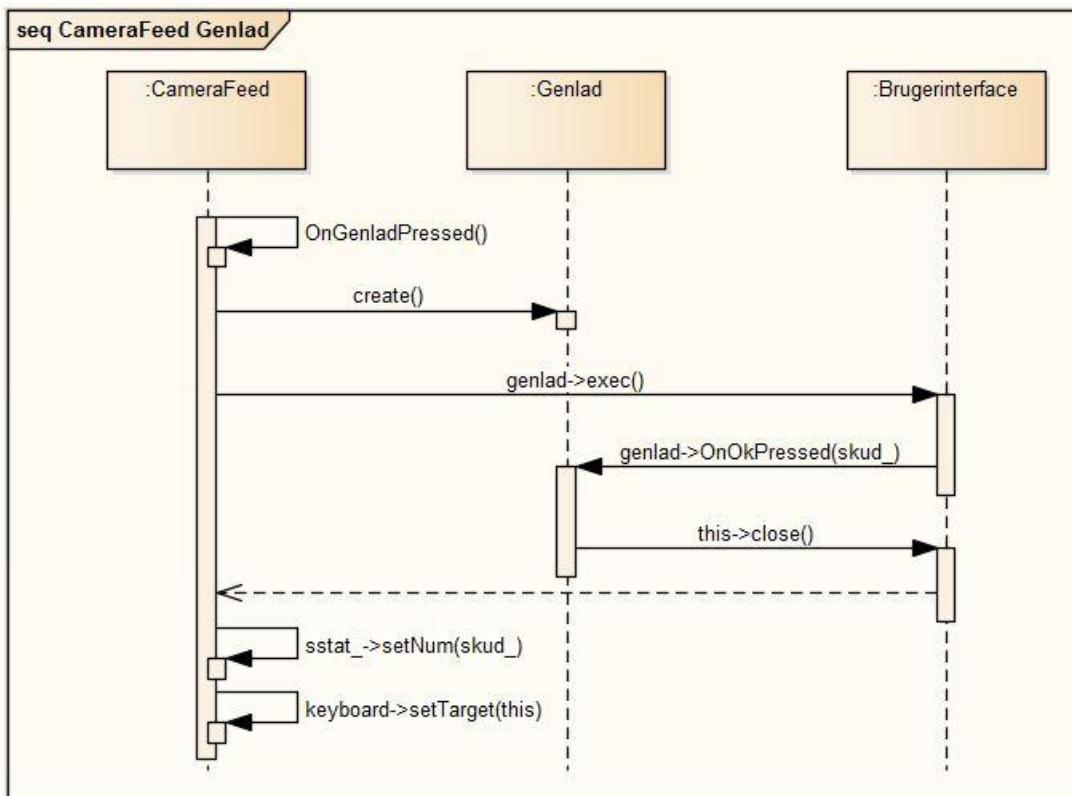
void OnDeaktiverPressed():

Denne funktion er på samme måde som OnAktiverPressed() forbundet til knappen "Deaktiver" i brugerinterfacet, et klik på denne vil dog deaktivere inputs fra PIR sensoren og dermed sørge for at alarmen ikke kan lyde. Dette noteres også i log filen.

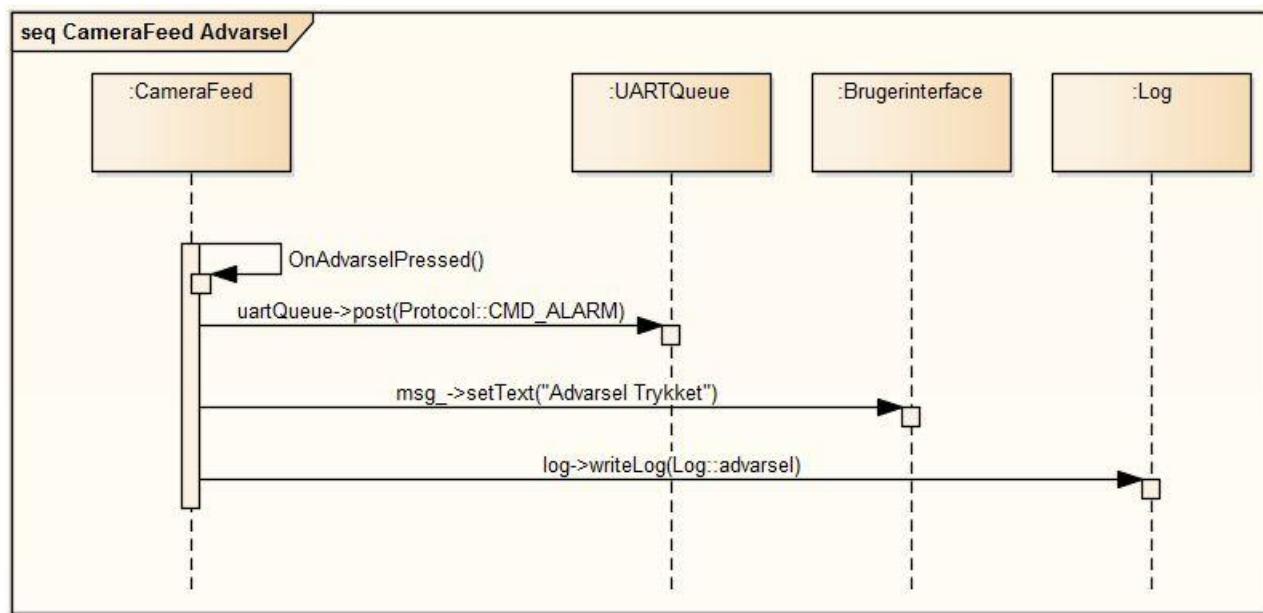


FIGUR 77 - SEKVENSDIAGRAM OVER ONDEAKTIVERPRESSED()**void OnGenladPressed():**

Ved tryk på denne knap oprettes et nyt objekt af Genlad klassen og dette initieres. Ved afslutning af det nye vindue sættes variablen sstat_ til det nye antal skud der blev indtastet i genlad vinduet.

**FIGUR 78 - SEKVENSDIAGRAM OVER ONGENLADPRESSED()****void OnAdvarselPressed():**

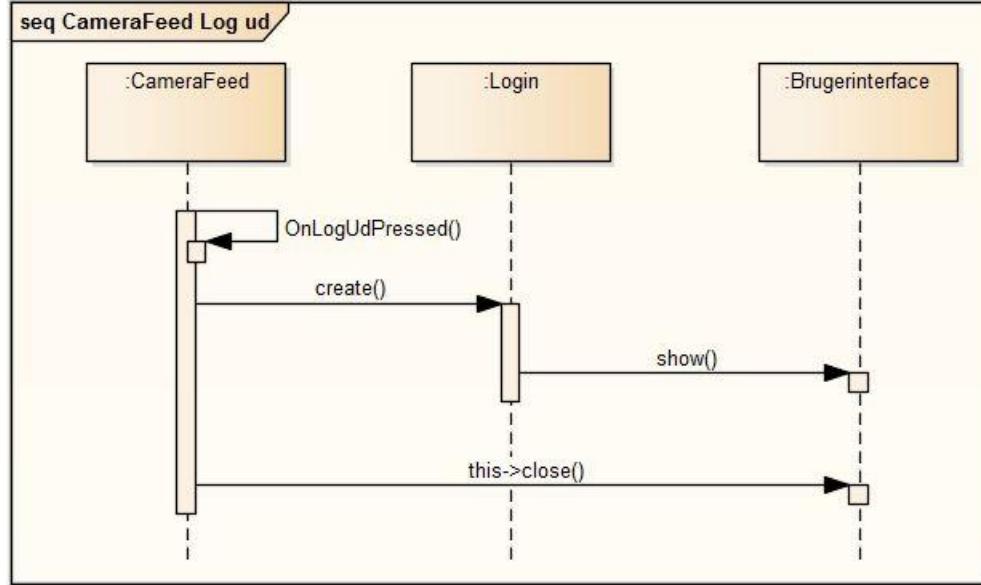
Denne funktion sender en advarsels-kommando til UARTQueue klassen, gennem Protocol klassen. Dette starter en advarselstone ude ved HDT. Derefter vises en meddeelse på brugerinterfacet om at en advarsel er givet og til sidst noteres dette i log filen.



FIGUR 79 - SEKVENDIAGRAM OVER ONADVARSELRESSED()

void OnLogUdPressed():

Når denne funktion kaldes oprettes et nyt objekt af Login klassen, dette vindue tilpasses og vises på brugerinterfacet, hvorefter CameraFeed objektet lukkes.



FIGUR 80 - SEKVENDIAGRAM OVER ONLOGUDPRESSED()

void updatePicture():

Denne funktion bruges til at opdatere billederne fra kameraet. Først laves capture variablen via OpenCV's bibliotek om til et `IplImage` som derefter kan indsættes i `putImage()` funktionen for derefter at blive sat ind i et `QPixmap` der muliggør at billedet kan vises i et Qt4 vindue.

void keyPressEvent(QKeyEvent *k):

Denne funktion er dybere beskrevet under emnet matrixkeyboard i afsnit 11.3.2 og 11.3.4 om design og implementering af matrixkeyboardet.

11.2.3. Genlad

Denne klasse bliver oprettet når brugeren trykker på ”Genlad” knappen i *brugerinterfacet*. Den sørger for at åbne et nyt vindue af typen QDialog, dette gør at brugeren er nødt til at forholde sig til dette vindue og enten indtaste hvor mange skud våbnet er blevet ladt med eller annulerer genladningen.

Klassen har indbyggede restriktioner der gør at man kun kan indtaste tal værende mellem 0 og 100, disse inkluderede.

Genlad	
-	antal_ : QLabel*
-	info_ : QLabel*
-	les_ : QLineEdit*
-	msg_ : QLabel*
-	ok_ : QPushButton*
+	Genlad(QWidget*)
+	~Genlad()
-	OnOkPressed(): void

FIGUR 81 - GENLAD KLASSE

Metoder:

Constructor:

Her oprettes en QPushButton ”Ok”, og der skabes en forbindelse fra denne og til funktionen OnOkPressed(), derudover oprettes labels med uddybende info til brugeren samt en QLineEdit som muliggør indtastning af data.

void OnOkPressed():

Når knappen ”Ok” bliver klikket på, bliver en midlertidig variabel oprettet, sammen med et for loop. For loopet bruges til at tælle igennem den QString af data, der er blevet indtastet i klassens QLineEdit. Først vil variablen temp blive ganget med 10, for at tælle korrekt op. Derefter vil det data der ligger på den gældende plads i for loopet blive lagt til temp variablen. Alt dette for at ændre datatypen til integers, som indsættes i variablen skud_. Efter dette lukkes genlad vinduet og CameraFeed vinduet vil være funktionsdygtigt igen.

```
int temp = 0;

for (int var = 0; var < les_->text().length(); ++var)
{
    int digit = les_->text().at(var).digitValue();
    temp=temp*10;
    temp+=digit;
}

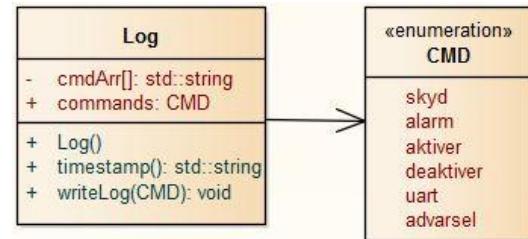
skud_ = temp;
```

FIGUR 82 – KODE UDSNIT FRA GENLAD KLASSEN

11.3. Backend funktionalitet [JDA, AEL, DT]

11.3.1. Log [AEL]

Denne klasse har til opgave at notere hver gang der sker en, for systemet, interessant handling. Dette bliver noteret i en .txt fil med et tidsstempel, så man på senere tidspunkt vil kunne gå ind og analysere et handlingsforløb. De interessante handler for systemer er vurderet til at være: Affyret skud, Aktiveret Alarm, Aktiver/Deaktiver system og Fejl i UART'en.



FIGUR 83 - LOG KLASSE

Metoder:

Constructor:

Opretter de strings der skal formuleres og indsætter dem på de passende pladser i kommando arrayet

string timestamp():

Bruger time.h biblioteket til at oprette et tidsstempel i lokal tid, få dette formateret på en korrekt måde og derefter at returnere dette.

Return:

Her returneres char arrayet "buf", som indeholder det korrekt formaterede tidsstempel.

void writeLog(Log::CMD cmd):

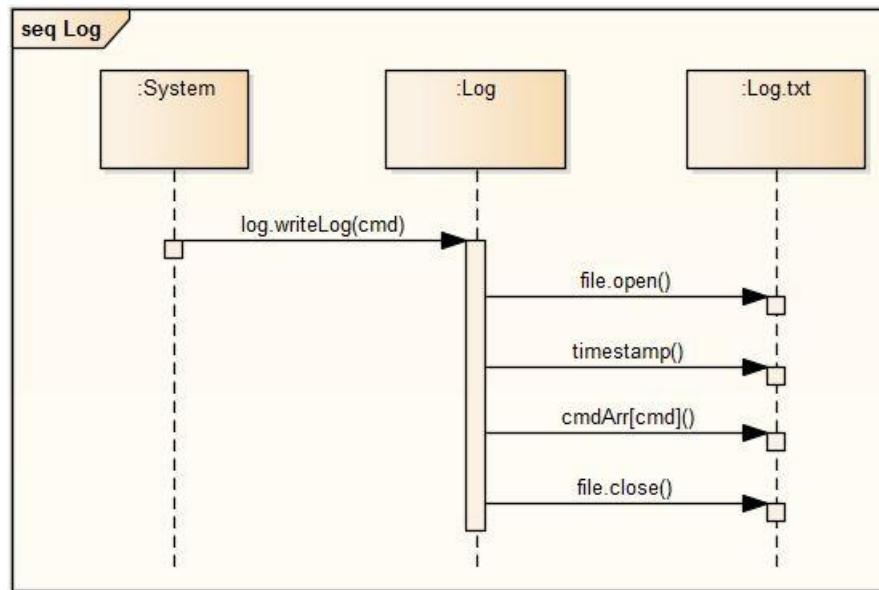
Opretter forbindelse til "Log.txt" filen, åbner den, skriver først timestamp() funktionen til denne efterfulgt af den relevante besked fra kommando arrayet cmdArr[cmd], hvorefter den lukker filen igen.

Parametre:

CMD: Et enum der finder den korrekte hændelse at notere i Log'en.

Return:

funktionen returnere intet.

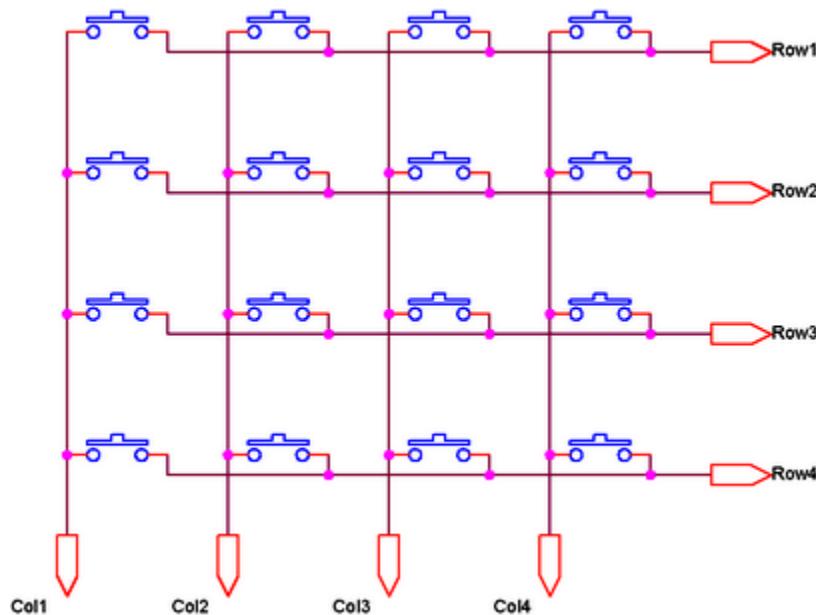


FIGUR 84 - SEKVENSDIAGRAM OVER WRITELOG()

11.3.2. Design af software til matrix tastatur.[JDA]

Kommunikation mellem brugeren og systemet sker via et matrix tastatur. Måden dette ønskes implementeret på, er gennem det Indlejrede Linux Systems GPIO porte. Det valgte matrixkeyboard har 16 knapper, og otte output-pins som kan læses fra. Knapperne repræsenterer tallene 0-9, samt karaktererne A, B, C, D, # og *.

Der læses fra tastaturet i loops ved skiftevis at sende et logisk LOW til tastaturets fire "række pins", hvor der for hvert loop, skiftevis læses fra tastaturets kolonner. Som det ses på Figur 85 kortsluttes en række med en kolonne når der trykkes på en knap. Der kan da aflæses hvilken kolonne og række der er kortsluttet.



FIGUR 85 - OPBYGNING AF MATRIXKEYBOARD

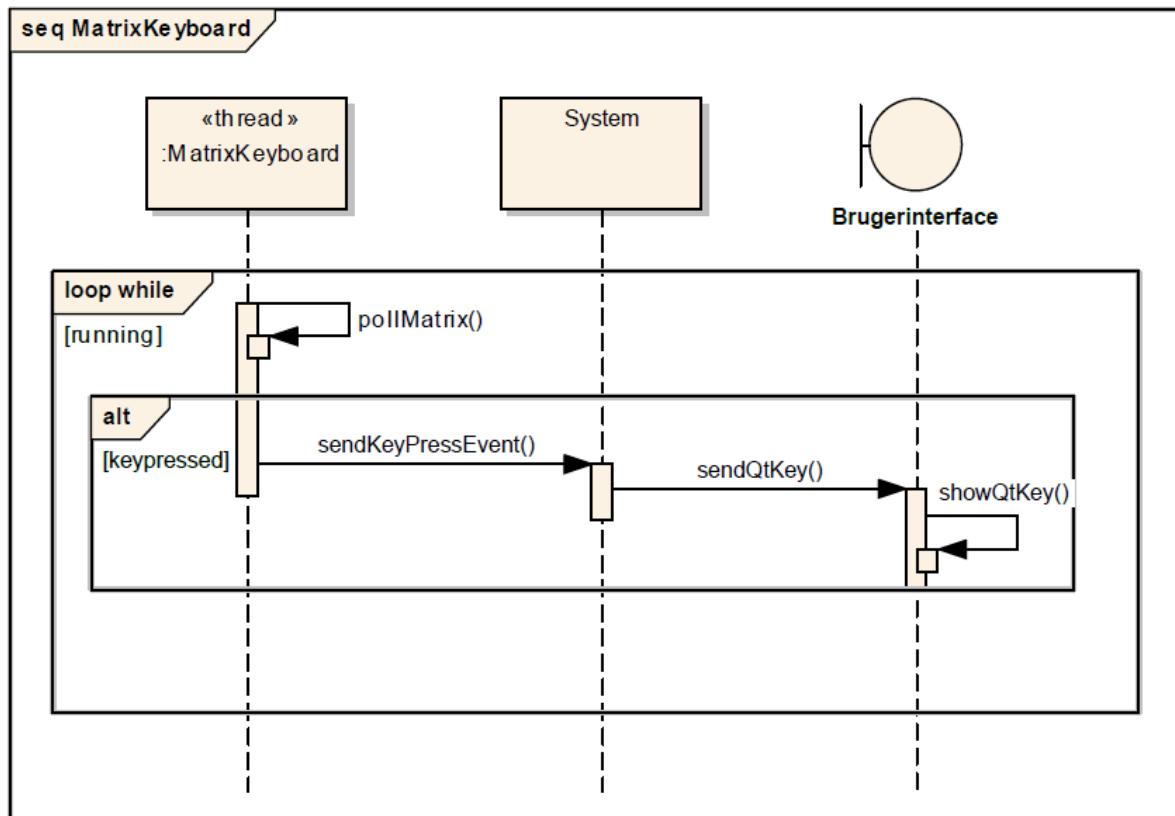
I arbejdet med design af software til matrixkeyboardet har pseudokode spillet en større rolle. Der var brug for en algoritme til aflæsning af matrixkeyboardet på det Indlejrede Linux System. Nedenfor ses et af de første pseudokode eksempler.

```
int columnCount=1
    for(;;) {
        setGPIO(columnCount++)
        for(rowCount=1; rowCount==4; rowCount++) {
            if((readGPIO(rowCount))==true)
                return cmd={columnCount,rowCount};
        }
    }
```

FIGUR 86 - PSEUDOKODE - FØRSTE UDKAST TIL MATRIXKEYBOARD ALGORITMEN

Til aflæsning og skrivning til det Indlejrede Linux Systems GPIO pins bruges biblioteket wiringPi – Se mere om dette under implementeringen af matrixkeyboardet.

Matrixkeyboard klassen skal indeholde en polling funktion der kan aflæse GPIO input, sammensætte dette med den output pin der er blevet sat logisk LAV. Herudover skal matrixkeyboard klassen indeholde en metode der oversætter pin-aflæsningen, til værdien på den tilsvarende tast, der er blevet trykket. Der ønskes concurrency i hovedprogrammet og matrixkeyboardets funktionalitet skal derfor køre i sin egen tråd. På Figur 87 ses et overordnet sekvensdiagram for matrixkeyboard klassen.



FIGUR 87 – OVERORDNET SEKVENDIAGRAM FOR MATRIXKEYBOARD

11.3.2.1. Matrixkeyboardet i brugerinterfacet.

Tanken at bruge matrixkeyboardet til navigation og indtastning i *brugerinterfacet*. I *brugerinterfacet* skal matrixkeyboardets funktionalitet kaldes flere steder, idet det skal bruges til forskellige formål. Matrixkeyboardets funktionalitet pakkes derfor ind i en klasse, så et objekt af denne kan bruges, hvor det findes nødvendigt. Matrixkeyboardets funktionalitet skal bruges både i login menuen, genlad menuen, samt hovedmenuen.

11.3.3. Design af kommunikationsprotokol [JDA]

Til kommunikationen mellem det Indlejrede Linux System og PSoC4, udvikles en protokol-klasse til manipulation af kommandoer med henblik på at give dem en konsistent form. Protokollen skal generere et array af fire chars, som kan sendes over det Indlejrede Linux Systems UART. Den første char er en start-byte der indeholder '1'. Den næste char er en kommando-char der fortæller hvilken kommando der ønskes udført af den modtagende enhed. Da designet af motorstyringen, kræver en hastigheds-variabel, der beskriver motorernes ønskede hastighed, bruges en "option-char" som nummer tre. Den sidste char der sendes, er en checksum af de to forrige. Checksummen skal være den XOR'ede værdi af "kommando" og "option". Denne checksum metode er lånt fra NMEA 183¹⁷ protokollen.

Alle kendte kommandoer der sendes over UART, kendes af protokolklassen. Af kommandoer kan der nævnes følgende:

- fullstop – bringer HDT's motorer til et fuldt stop.
- up – beder PSoC4 om at styre HDT's vertikale motor i opadgående retning
- down - beder PSoC4 om at styre HDT's vertikale motor i nedadgående retning
- left – beder PSoC4 om at styre HDT's horisontale motor mod venstre
- right – beder PSoC4 om at styre HDT's horisontale motor mod højre
- shoot – beder PSoC4 om at starte HDT's trigger motor (udløser våbnet)
- alarm – beder PSoC4 om at afspille en advarselslyd fra HDT's højtalere
- laser – beder PSoC4 om at tænde for HDT's lasersigte
- laseroff – beder PSoC4 om at slukke for HDT's lasersigte

Der blev oprettet et dokument til at holde styr på samtlige kommandoer, og tilhørende karakterer. Se et udsnit her på Figur 88.

¹⁷ http://en.wikipedia.org/wiki/NMEA_0183#Application_layer_protocol_rules – NMEA183 protokollens application layer rules.

Start	Kommando	Option	Tjeksum	
char '1'	1 byte (char)	1 byte (char)	Kommando XOR Option	
Kommandoer				
Bits	Kommando	Bemærk:	Store/små bogstaver er vigtige at skelne imellem	
F	Fuldt stop			
V	Venstre			
H	Højre			
O	Op			
N	Ned			
S	Skyd			
R	Reset			
A	Alarm			
L	Laser			
Option				
0-256	0 = stop, 255 = fuld fart			
Tjeksum				
Der udføres en XOR på kommando og option (i C: kommando ^ hastighed)				

FIGUR 88 - PROTOKOL TIL KOMMUNIKATION MELLEM PSoC OG DET INDELJREDE SYSTEM

Bemærk at R for RESET ikke benyttes i den endelige implementering.

11.3.3.1. Protokollen i brugerinterfacet

Protokollen bruges hver gang der sendes data fra det Indiejrede Linux System til PSoC4. Kommunikationen sker serielt over UART. Da der er flere klasser der skal sende information til PSoC4, vil der hertil udvikles en UART kø-klasse¹⁸ som har til formål at holde styr på, samt videreformidle, den data der skal sendes. Protokolklassen skal udvikles således, at de klasser som har et objekt af denne, kan omdanne en evt. "kommando" og "option" til et array af fire chars, på formen beskrevet ovenfor.

¹⁸ Se afsnit 11.3.6 og 11.3.7 om UART og UARTQueue

11.3.4. Implementering af matrixkeyboard [JDA]

Dette afsnit indeholder en dybdegående forklaring matrixtastatur-klassens implementering samt dens implementering i brugerinterfacet. Herudover indeholder afsnittet en beskrivelse af funktionaliteten fra GPIO biblioteket wiringPi¹⁹

11.3.4.1. Indlejret linux system – Raspberry Pi og wiringPi

Som det Indlejrede Linux System benyttes en raspberry pi 2²⁰. Raspberry Pi er en single-board computer, udstyret med 40 GPIO²¹ pins hvoraf 10 bruges til matrixkeyboardet (4 input, 4 output samt 3.3V og GND). For at benytte Raspberry Pi's GPIO pins bruges wiringPi biblioteket. Dette bibliotek gør det muligt at inititere, skrive og læse fra GPIO.

11.3.4.2. Identifikation og nummering af Raspberry Pi's GPIO pins

Raspberry Pi's 40 GPIO pins er nummeret på flere måder. wiringPi giver en "gpio readall" kommando, hvis output er vist på Figur 89.

i@raspberrypi ~ \$ gpio readall											
						Pi 2					
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
		3.3v			1	2		5v			
2	8	SDA.1	IN	1	3	4		5V			
3	9	SCL.1	IN	1	5	6		0v			
4	7	GPIO. 7	IN	1	7	8	1	ALTO	TxD	15	14
		0v			9	10	1	ALTO	RxD	16	15
17	0	GPIO. 0	IN	0	11	12	0	IN	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13	14			0v		
22	3	GPIO. 3	IN	0	15	16	0	IN	GPIO. 4	4	23
		3.3v			17	18	0	IN	GPIO. 5	5	24
10	12	MOSI	IN	0	19	20			0v		
9	13	MISO	IN	0	21	22	0	IN	GPIO. 6	6	25
11	14	SCLK	IN	0	23	24	1	IN	CEO	10	8
		0v			25	26	1	IN	CE1	11	7
0	30	SDA.0	IN	1	27	28	1	IN	SCL.0	31	1
5	21	GPIO.21	IN	1	29	30			0v		
6	22	GPIO.22	IN	1	31	32	0	IN	GPIO.26	26	12
13	23	GPIO.23	IN	0	33	34			0v		
19	24	GPIO.24	IN	0	35	36	0	IN	GPIO.27	27	16
26	25	GPIO.25	IN	0	37	38	0	IN	GPIO.28	28	20
		0v			39	40	0	IN	GPIO.29	29	21
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
					Pi 2						

FIGUR 89 - RASPBERRY PI2 GPIO NUMMERERING OG FUNKTIONALITET

På Figur 89 ses alle tre forskellige nummeringsmetoder: Fysiske, wiringPi numre og BCM numre, som er defineret af Broadcom, som er udvikleren af mikroprocessoren²². Før GPIO'erne kan bruges, kaldes funktionen **wiringPiSetupGpio(void)**. Herved kan GPIO'erne tilgås direkte med funktionerne **digitalWrite(int pin, int value)** og **digitalRead(int pin, int value)** med broadcoms BCM numre, som pinnummer. Dette er praktisk da der ikke er brug for at remappe til GPIO'ernes fysiske addresser, dog bruges dette på bekostning af portabilitet.

¹⁹ <http://wiringpi.com/>

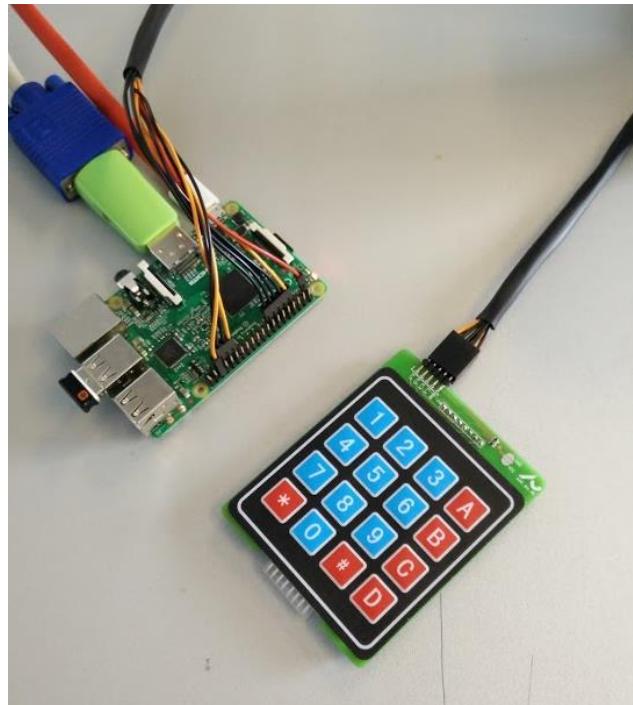
²⁰ <https://www.raspberrypi.org/>

²¹ General-purpose input/output

²² Mikroprocessoren er af typen BCM2836 ARM Cortex-A7

11.3.4.3. Implementering af matrixkeyboard klassen

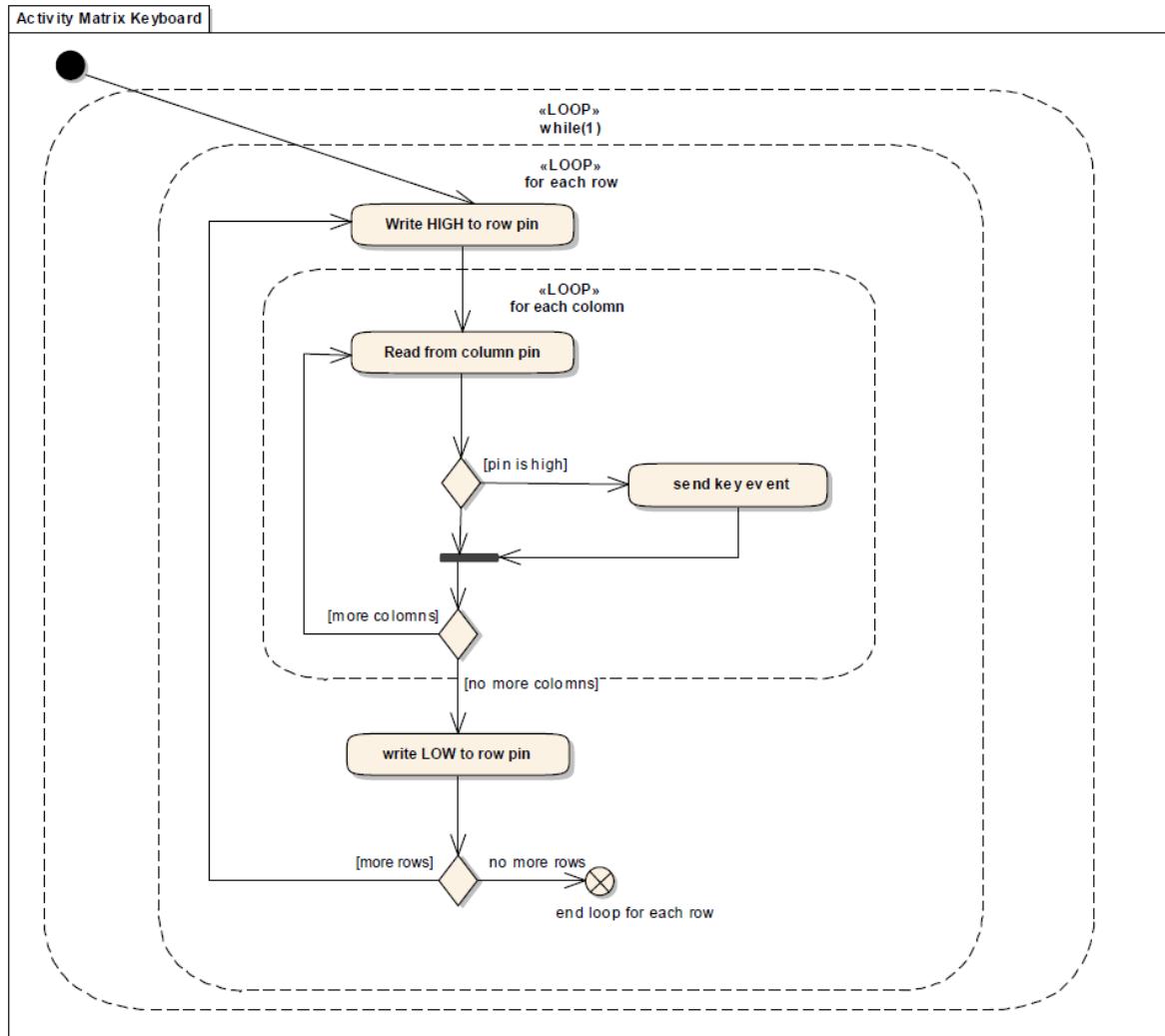
Matrixkeybordet er opbygget af kolonner og rækker²³, hvor hver har en pin loddet på printet. De fire rækker tilsluttes hhv. til GPIO pin[4, 17, 27, 22] og kolonneerne til GPIO pin[5, 6, 13, 16] VCC og GND tilsluttes de fysiske pins 9 og 1. Se Figur 90 for fysisk opstilling.



FIGUR 90 - FYSISK OPSTILLING AF MATRIXKEYBOARD

²³ Se afsnit 11.3.2 om design af Matrixkeyboard.

Matrixkeyboard klassen blev identificeret i designfasen, og i forbindelse med implementeringen heraf er der udviklet et detaljeret klassediagram, se Figur 92 der beskriver klassens metoder og attributter. I forbindelse med implementeringen af matrixkeyboard klassen er der udarbejdet et aktivitetsdiagram, se Figur 91.



FIGUR 91 - AKTIVITETSDIAGRAM FOR MATRIXKEYBOARD

11.3.4.4. Beskrivelser af attributter og metoder for MatrixKeyboard klassen

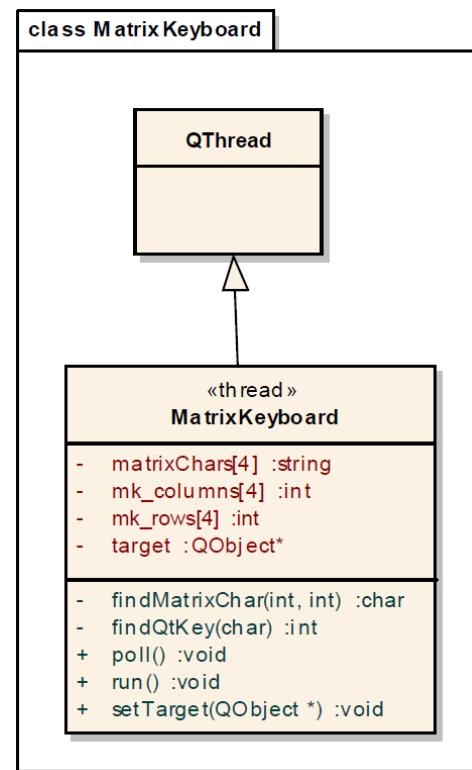
I dette afsnit findes detaljerede beskrivelser af MatrixKeyboard klassens medlemsfunktioner samt attributter.

For at gøre koden letlæselig og lettere at rette, defineres Raspberry Pi's GPIO BCM numre til genkendelige navne.

```
// Rows
#define MK1_0 4
#define MK1_1 17
#define MK1_2 27
#define MK1_3 22

// Columns
#define MK1_4 5
#define MK1_5 6
#define MK1_6 13
#define MK1_7 16
```

**FIGUR 93 - KODEUDSNIT,
#DEFINE AF RASPBERRY
Pi's BCM NUMRE**



Constructor – MatrixKeyboard(QObject *target)

Constructoren tager en pointer til et **QObject target** som eneste parameter. Se beskrivelse af klassens attributter for yderligere information.

I constructoren initieres Raspberry Pi's GPIO pins med **wiringPiSetupGpio()**, og matrixkeyboardets rækker[1;4] sættes til output med wiringPi funktionen **pinMode(int pin, int mode)**.

```

// Setup output pins (rows)
pinMode(MK1_0, OUTPUT); // Row 1
pinMode(MK1_1, OUTPUT); // Row 2
pinMode(MK1_2, OUTPUT); // Row 3
pinMode(MK1_3, OUTPUT); // Row 4

// Setup input pins (columns)
pinMode(MK1_4, INPUT); // Column 1
pinMode(MK1_5, INPUT); // Column 2
pinMode(MK1_6, INPUT); // Column 3
pinMode(MK1_7, INPUT); // Column 4

```

FIGUR 94 - KODEUDSNIT FRA MATRIXKEYBOARD.CPP "PINMODE SETUP"

På same måde sættes matrixKeyboardets kolonner[1;4] til input.

For at sikre at rækkerne er sat logisk HIGH når et objekt af matrixtastaturet oprettes, køres funktionen **digitalWrite(int pin, int value)** med **HIGH** som anden parameter.

Attributten **matrixChars** initialiseres også i constructoren, se beskrivelse af attributter.

void poll()

Metoden **poll()** er kernen i matrixkeyboard klassen. Det er her algoritmen for aflæsning af input kommer til udtryk. Grundstenen i algoritmen, fra pseudokoden i designfasen, benyttes stadig i algoritmen. Dog har det været nødvendigt at tilføje yderligere funktionalitet til **poll()** for at få et fornuftigt resultat. Ved første udkast til implementeringen blev der blot læst fra kolonnerne i et loop for hver række. Dette medførte at programmet læste flere gange ved bare et enkelt tryk. For at løse problemet blev der i **poll()** introduceret et nyt integer array, **previousRead[]** med 2 pladser, samt en bool variabel, **dirtyBit**. Arrayet bruges til at holde den sidst læste værdi (række og kolonne), hvor **dirtyBit** sættes hver gang der registreres input på matrixkeyboardet. **poll** metoden opretter kun et QKeyEvent når **previousRead** ikke indeholder de samme værdier som den nyeste læsning fra rækkerne og kolonnerne. Dette medfører at hvis brugeren holder en knap på matrixkeyboardet nede, vil der kun registreres ét input i *brugerinterfacet*. Se Figur 95 for eksempel.

```

digitalWrite(mk_rows[row], LOW);
for (column = 0; column < 4; column++) {
    if (digitalRead(mk_columns[column]) == 0) {
        dirtyBit = true;
        if (previousRead[0] != row || previousRead[1] != column) {
            std::cout << "Read: " << findMatrixChar(row, column) << std::endl;
            char read_char = findMatrixChar(row, column);
        }
    }
}

```

FIGUR 95 - POLL METODENS AFLÆSNING AF KOLONNER

For at brugeren har mulighed for at trykke på samme knap flere gange i træk, bruges **dirtyBit**. Denne variabel/flag sættes kun når der registreres input på matrixkeyboardet. Når brugeren slipper en knap, kører loopen indtil det igen når til den første række. Her sættes **dirtyBit** til false eller **previousRead** sættes til -1, hvis **dirtyBit** er false. Figur 96 illustrerer dette.

```

while (1) {
    for (row = 0; row < 4; row++) {
        if (row == 0) {
            if (dirtyBit) {
                dirtyBit = false;
            }
            else {
                previousRead[0] = -1;
                previousRead[1] = -1;
            }
        }
    }
}

```

FIGUR 96 - KODEDSNIT - RÆKKERNE FOR LOOP. HER CHECKES OM DIRTYBIT ER SAT

std::string matrixChars[4] og metoden findMatrixChars

Attributten **matrixChars** er et string array med fire pladser – en til hver række på matrixkeyboardet. I constructoren sættes pladserne [0;4] til de fire karakterer der står på det fysiske matrixkeyboard. Se Figur 97

```

// Create array of the matrix chars
matrixChars[0] = "123A";
matrixChars[1] = "456B";
matrixChars[2] = "789C";
matrixChars[3] = "*0#D";

```

FIGUR 97 - KODEUDSNIT - MATRIXCHAR TILDELES TRINGS TILSVAREnde LAYOUTET PÅ DET FYSISKE TASTATUR

Når poll funktion oplever at der bliver trykket på en knap på matrixkeyboardet, kaldes funktionen **findMatrixChar** med pågældende række og kolonne som parametre. **FindMatrixChars** returnerer da karakteren tilsvarende til den på det fysiske matrixkeyboard. Se Figur 98.

```
char MatrixKeyboard::findMatrixChar(int row, int column) {
    return matrixChars[row][column];
}
```

FIGUR 98 - KODEUDSNIT - IMPLEMENTERING AF FUNKTIONEN MATRIXCHARS

int findQtKey(char key)

I Qt kan karakterer defineres med en enumeration, Qt::key. Dette udnyttes i metoden **findQtKey** som indeholder en switch med 16 cases – en for hver karakter på det fysiske matrixKeyboard. De 16 cases returnerer en Qt::key enum værdi tilsvarende den char, **findQtKey** fik som parameter. Se Figur 99 for eksempel.

```
int MatrixKeyboard::findQtKey(char key) {
    switch (key) {
        case '0':
            return Qt::Key_0;
            break;
        case '1':
            return Qt::Key_1;
            break;
        case '2':
            return Qt::Key_2;
            break;
```

FIGUR 99 - KODEUDSNIT - DE FØRSTE TRE CASES I METODEN FINDQTKEY

int mk_rows[] og int mk_columns[]

Disse integer arrays bruges til at holde Raspberry Pi'ens pin BCM numre, hvorpå hhv. matrixkeyboardets række- og kolonne pins er påsat. **mk_rows** og **mk_columns** tildeles værdierne i constructoren. Disse værdier er prædefineret med #define, som set på Figur 93.

void setTarget(QObject *target) og matrixkeyboardets implementering i Qt frameworket

Matrixkeyboardet bruges af flere vinduer i hovedprogrammet. Klasserne cameraFeed, Genlad og Login, se afsnittet om GUI, har derfor en pointer til et objekt, af klassen MatrixKeyboard kaldet keyboard. MatrixKeyboard objektets metode **setTarget** kaldes i de tre vinduers constructor. Dette gøres på følgende måde: **this->keyboard->setTarget(this)**, hvor **this** er pointeren til det aktuelle objekt (objektet hvorfra funktionskaldet sker). MatrixKeyboardets **setTarget** metode sætter attributten **target** til det objekt metoden får som parameter. Ved at bruge en pointer til matrixkeyboard-objektet undgås det at bruge flere objekter af MatrixKeyboard klassen, som vil kunne føre til problemer, når nye vinduer åbnes gennem brugerinterfacet.

Oprettelse og passing af QKeyEvent

Når brugeren trykker på en knap opretter **poll** funktionen et QKeyEvent, med **findQtKey(read_char)**, som anden parameter. Se **read_char** variablen på sidste linje på Figur 95. Dette event sendes til target-objektet med metoden **postEvent** fra Qt klassen, QApplication. Se eksempel på Figur 100. **postEvent** tager et QObject og et QEvent som parametre, hvor QObjectet er det objekt som modtager QEventet.

```
char read_char = findMatrixChar(row, column);
QKeyEvent *event = new QKeyEvent(QEvent::KeyPress, findQtKey(read_char), Qt::NoModifier, (QString)read_char);
QApplication::postEvent(target, event);
```

FIGUR 100 - KODEUDSNIT, MATRIXKEYBOARD.CPP - OPRETTELSE AF QKEYEVENT SAMT SEND EVENT TIL TARGET MED POSTEVENT

I koden til brugerinterfacet implementeres matrixkeyboardets funktionalitet ved at benytte det QKeyEvent der blev passed til **postEvent**. Dette gøres vha. switch-cases, der indeholder cases til de QKeyEvents der kommer fra matrixkeyboardet. På Figur 101 ses et eksempel af koden fra brugerinterfacets hovedmenu.

```
void cameraFeed::keyPressEvent(QKeyEvent *k)
{
    switch (k->key()) {
        case Qt::Key_1:
            OnAktiverPressed();
            break;
        case Qt::Key_2:
            OnDeaktiverPressed();
            break;
        case Qt::Key_3:
            OnGenladPressed();
            break;
        case Qt::Key_4:
            OnAdvarselPressed();
            break;
        case Qt::Key_5:
            OnLogUdPressed();
            break;
        default:
            break;
    }
}
```

FIGUR 101 - KODEUDSNIT FRA CAMERAFEED.CPP'S KEYPRESSED FUNKTION - MATRIXKEYBOARD FUNKTIONALITET MED EVENTS

Som det ses på Figur 101, laves der funktionskald baseret på hvilket QKeyEvent der bliver registreret. Metoden som dette er implementeret på, er den samme for både login menuen og genlad menuen.

11.3.4.5. Implementering af matrixkeyboardet med QThread og metoden run

Da matrixkeyboardets funktionalitet skal bruges samtidig med andet, køres **poll** metoden i sin egen tråd.

Virtual void run()

Matrixkeyboard klassen arver fra Qt klassen, QThread. Metoden **run** er et override af QThread's egen run-funktion. En QThread-tråd startes i metoden **run**. **run** metodens eneste opgave er da at køre **poll** metoden.

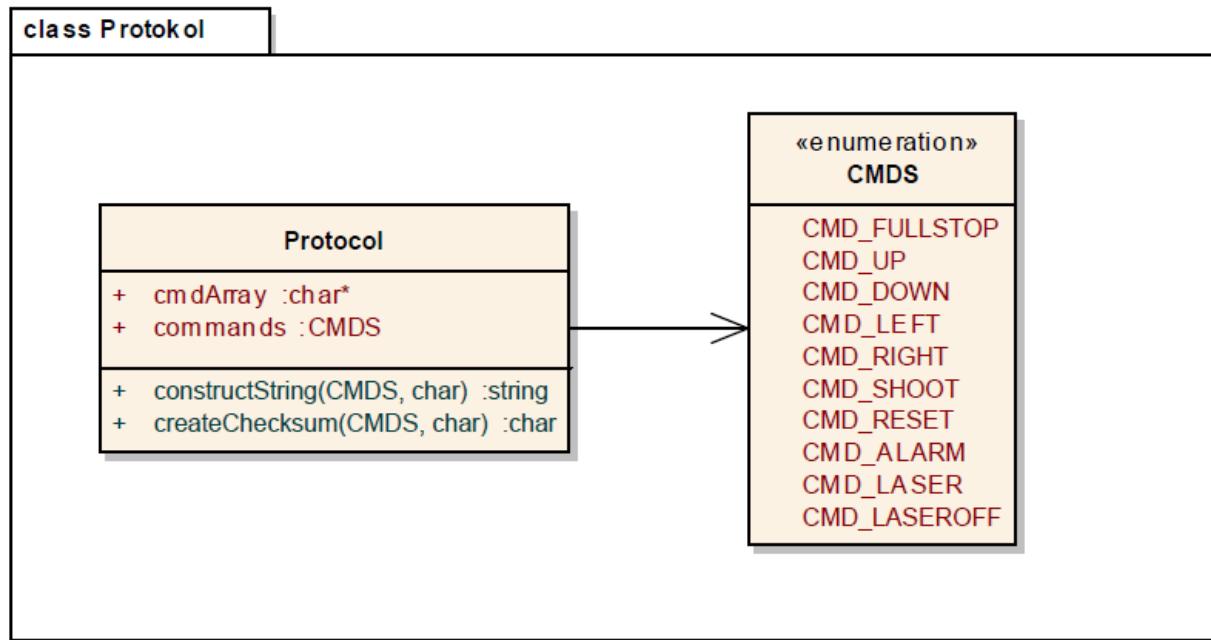
11.3.5. Implementering af protokol mellem PSoC4 og raspberry pi [JDA]

Kommunikation mellem det Indlejrede Linux System og PSoC4 foregår serielt med UART²⁴ hvor data sendes én char (8 bit) af gangen. Som beskrevet i designfasen er protokollen designet så der sendes 4

²⁴ Universal Asynchronous Receiver/Transmitter

bytes per dataudveksling. Kommunikationen mellem PSoC4 og det Indlejrede Linux System er ensrettet mod PSoC4, og der modtages ikke en acknowledge besked efter endt sendesekvens. Protokol klassen er designet til at skulle modtage en kommando samt en tilhørende option, og derefter konstruere en string, bestående af af fire chars. En start-char '1', en kommando, en option og en checksum, som defineret i protokoldesignet.

Option char'en benyttes til fortælle PSoC4 hvor hurtigt den horisontale og/eller vertikale motor skal køre. På denne måde kan hastigheden hvormed HDT bevæger sig bestemmes. Hvis en kommando ikke henvender sig til bevægelse af HDT, sendes '0' som option.



FIGUR 102 - KLASSEDIAGRAM FOR PROTOCOL

Protokollens implementering er relativt simpel. Se Figur 102 for klassediagram.

For at gøre kommandoerne lettere at huske på, samt for at forberede letlæseligheden i koden, bruges en enum, **CMDS** og et char array, **cmdArray** til at holde styr på samtlige mulige kommandoer. Når protokollen skal bruges et sted i hovedprogrammet, kan man blot bruge kommandoens enum-navn, i stedet for at huske hvilken char der bruges, til at udføre hver kommando. Se Figur 103, som er et kodeudsnit fra protokollens constructor, hvor det kan ses, hvilke chars, der binder sig til de forskellige kommandoer. Som det også ses på Figur 103, er det kommandoens enum navn, der bestemmer hvilken plads i **cmdArray**'et der tildeles en bestemt char.

11.3.5.1. Beskrivelse af protokol klassens metoder og attributter

I dette afsnit forefindes dybdegående beskrivelser af protokolklassens metoder og attributter

Constructor og destructor

I konstruktoren initialiseres char arrayet **cmdArray** dynamisk med ni pladser til at holde en char for hver kommando. Her bruges enum'en **CMDs** til at definere karakterernes plads i arrayet. Hvis der er brug for udvidelse af systemet med flere kommandoer skal kommandoen skrives ind i konstruktoren og enum'ens erklæring i Protocol.h filen.

Destructoren bruges udelukkende til at frigøre den hukommelse der er allokeret af **cmdArray**'et.

char createChecksum(CMDs command, char option)

Protokollens checksum genereres ved at XOR're commando-char'en og option-char'en. I C++ bruges den logiske operator `^`. Metoden **createChecksum** returnerer den XOR'ede værdi som en char.

String constructString(CMDs command, char option)

Metoden **constructString** har til formål at bygge en std::string ud fra dens parametre **CMDs command** og **char option**. Til dette oprettes en lokal string, i metodens scope. String klassens metode, **push_back** bruges da til indsættelse af karakterer bagerst i strengen. Først indsættes karakteren der svarer til **CMDs** kommandoen, derefter **option** og til sidst checksummen, der genereres af metoden **createChecksum**. Se Figur 104 for kodeudsnit fra **constructString** metoden.

11.3.5.2. Brug af protokol klassen

Protokollen bruges hver gang der skal sendes data fra det Indlejrede Linux System til PSoC4. Protokol-klassen bruges derfor både i klassen **JoystickThread** og i klassen **cameraFeed**. Joystick klassen har et objekt af protokol-klassen som private member, hvilket gør det muligt at kalde protokol-klassens **constructString** metode fra joystick-klassens koordinat-handler metoder, se afsnit for design og implementering af joystick. Figur 105 viser brugen af Protokol klassens **constructString** metode i **Joystick.cpp**

```
string out;
out.push_back('1');
out.push_back(cmdArray[command]);
out.push_back(option);
out.push_back(createChecksum(command, option));
return out;
```

FIGUR 104 - KODEUDSNIT, CONSTRUCTSTRING FRA PROTOCOL.CPP

```
else {
    yDelta *= (-1);
    direction = Protocol::CMD_DOWN;
}

if (yDelta > 200 && lastY != 1) {
    lastY = 1;
    uartQueue->post(protocol.constructString(direction, '1'), 4);
```

FIGUR 105 - KODEUDSNIT, JOYSTICK.CPP - METODEN HANDLEYCORD

Klasserelationen mellem cameraFeed og Protocol er på samme måde som med Joystick en komposition, hvor cameraFeed har et objekt af Protocol som private member. I cameraFeed vinduet (hovedmenuen) er der mulighed for at tænde/slukke for HDT's lasersigte samt at afgive advarselslyd. Da funktionaliteten

til at udføre disse opgaver ligger på PSoC4, sendes disse kommandoer med UART'en. På Figur 106 kan det ses hvorledes protokollens funktionalitet bruges af **cameraFeed**.

```
void cameraFeed::OnAdvarselPressed()
{
    uartQueue->post(protocol.constructString(Protocol::CMD_ALARM, '0'), 4);
    msg_->setText("Advarsel Trykket");
    log->writeLog(Log::advarsel);

}
```

FIGUR 106 - KODEUDSNIT, CAMERAFEED.CPP - METODEN ONADVARSELPRESSED

11.3.6. Design og implementering af UART [DT]

Kommunikationen mellem PSoC og Raspberry foregår ved hjælp af seriel kommunikation, nærmere bestemt UART. Til dette formål er der på raspberry'en lavet en UART klasse, som udelukkende står for at sende kommando-strenge, som bliver lavet med Protocol klassen²⁵.

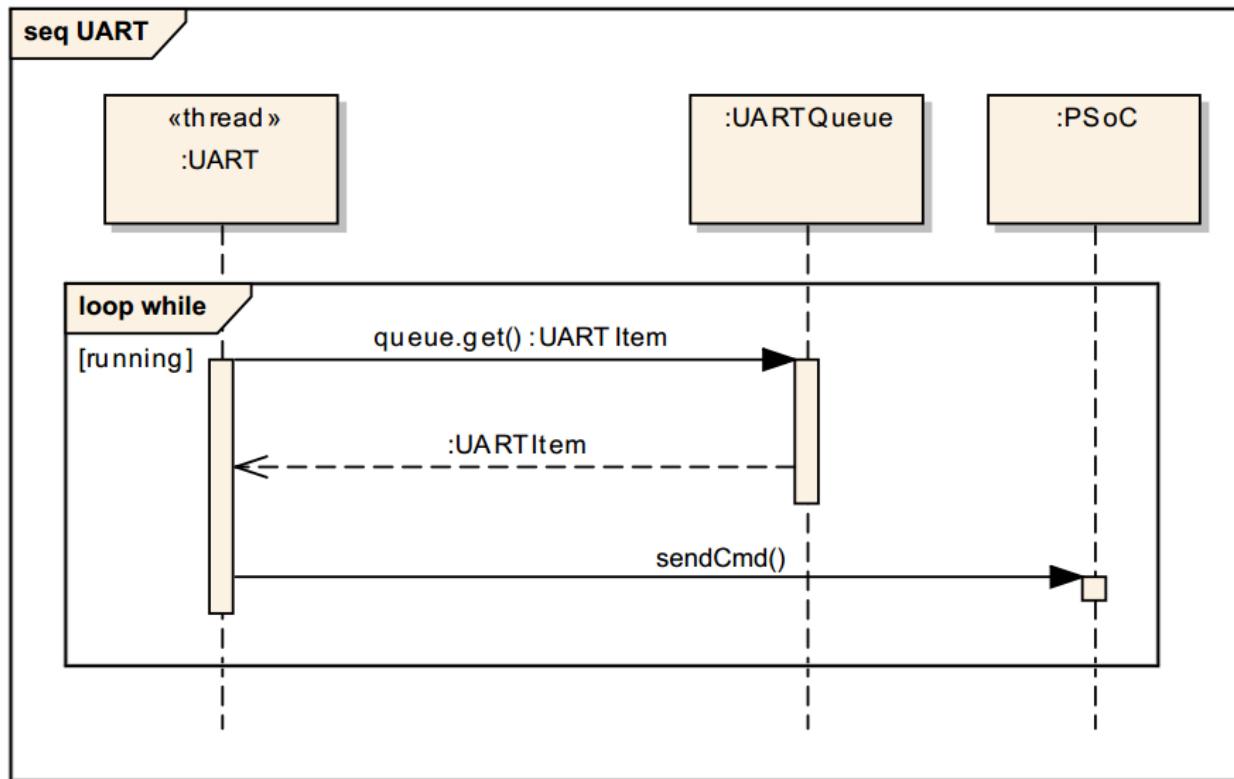
For at sende beskeder over UART bruges Raspberry'ens GPIO pinde dedikeret til dette formål. Raspberry'en har to pins til dette formål: En Rx (receive) pin og en Tx (transmit) pin. Kommunikationen mellem PSoC og Raspberry er envejs. Der bliver udelukkende sendt data fra Raspberry og til PSoC, men PSoC'en sender ikke noget tilbage.

UART klassen er designet så den bliver kørt i sin egen tråd, som kører uafhængigt og asynkront i forhold til main programmet (som styrer vinduerne i GUI'en). Der bliver oprettet én instans af UART klassen, og denne vil efterfølgende blive passeret rundt i mellem de forskellige vinduer.

11.3.6.1. Modtage data der skal sendes til PSoC

Da UART klassen kører i en asynkron tråd skal der implementeres en form for kommunikationsvej mellem hovedprogrammet og UART tråden. For at opnå dette bruges en kø (som kaldes UARTQueue).

UART klassens funktionalitet kan ses på følgende diagram.



FIGUR 107 - SEKVENSDIAGRAM FOR UART KLASSEN

²⁵ Se design af protocol I afsnit 11.3.3

Som det ses på sekvensdiagrammet (Figur 107) kører UART klassen i et while loop. Det eneste UART klassen gør er at læse fra UARTQueue klassen, vha. En get funktion. UARTQueue er en besked kø, som har en post- og en get-funktion. Post sætter noget ind i køen, og skal kaldes fra de klasser, hvor man ønsker at sende noget til UART klassen, og get kaldes udelukkende af UART klassen, for at hente det næste data der skal sendes.

Så længe UARTQueue er tom, så vil get() metoden blokere, indtil der sættes noget ind i køen. Når UART klassen modtager data fra UARTQueue, så sender den det pågældende data med det samme, som modtages af PSoC'en.

UART klassen laver ikke tjek af det data den modtager, og indsætter ikke ny data eller ændrer på det modtagne data. Al denne funktionalitet sørges for i Protocol klassen. Det eneste job UART klassen har er blot at sende.

11.3.6.2. [UARTQueue og UARTItem](#)

For at UART klassen kan modtage data, når den køres i en separat tråd er der som sagt en UARTQueue.

UARTQueue skal udelukkende stå for at modtage og videre sende beskeder, som skal fra en klasse til UART klassen. Dette gøres ved brug af en **std::queue** fra C++'s STL bibliotek. Std::queue er en FIFO (First In First Out) container, med tilhørende algoritmer til at indsætte i enden (push) og fjerne fra fronten (pop).

Køen gemmer på UARTItems. UARTItem vil indeholde den *string* der skal sendes til PSoC'en (4 chars). UARTItem indeholder også hvor mange bytes der er i stringen. Dette vil altid være 4 bytes, når der sendes til PSoC'en, da der altid sendes 4 chars, som er defineret i protokollen for kommunikation mellem Raspberry og PSoC.

Da UARTQueue kommunikerer med flere forskellige asynkrone instanser (UART tråden, main programmet og joysticket) skal der være tråd-sikkerhed, så UARTQueues interne data ikke bliver korrupt/fejlfyldt. Dette gøres ved hjælp af en mutex i UARTQueue klassen. Denne mutex bliver brugt, så der altid kun er én instans der modifierer køen af gangen. Dette betyder, at mutexen skal tages i brug og låses, hver gang der enten skrives til køen (post) eller trækkes data ud af køen (get). Uden tråd-sikkerhed vil to instanser der modifierer samme objekt kunne gøre den data der står i køen korrupt og ubrugbar.

11.3.7. [Implementering af UARTItem og UARTQueue](#)

UARTItem er en simpel struct, der har to public attributter og en constructor funktion, uden nogen betydelig implementering.

```
struct UARTItem {
    // Constructor for parsing values when creating the struct object.
    UARTItem(std::string data, int bytes) : data(data), bytes(bytes) {}
    std::string data;
    int bytes;
};
```

FIGUR 108 - UARTITEM IMPLEMENTERING

Som det ses på Figur 108, så har UARTItem en medlemsvariabel kaldet *data*, af typen *std::string*. Denne indeholder den char-sekvens der skal sendes over UART. Den anden medlemsvariabel *bytes* er blot en *int*, der indeholder det antal chars der er i *data* variablen.

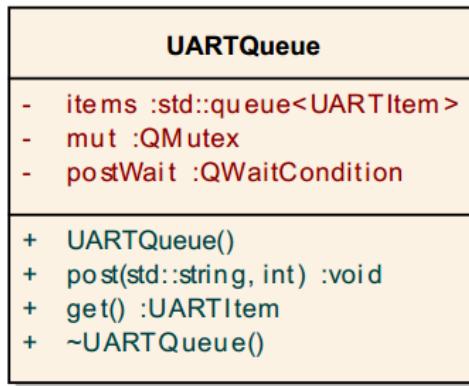
Constructoren er der, så det er hurtigt at oprette en instans af UARTItem og indsætte værdier i de to medlemsattributter.

Det kan ses, at constructoren udelukkende sætter de to medlemsattributters værdi til de værdien der er givet i de to parametre til constructoren. Grundet constructoren er det muligt at oprette et object på én linje:

```
UARTItem item("string", 6);
```

Ovenstående vil oprette et UARTItem objekt og sætte *data* og *bytes* til henholdsvis "string" og 6. Denne UARTItem kan efterfølgende indsættes i UARTQueue køen.

UARTQueue klassens opbygning kan ses nedenfor, på Figur 109.



FIGUR 109 – KLASSEDIAGRAM FOR UARTQUEUE

Klassen indeholder 3 private medlemsattributter, samt 4 public metoder.

11.3.7.1. Klassebeskrivelse

UARTQueue()

Beskrivelse: Dette er constructoren for klassen. I denne udskrives der blot, at objektet er oprettet til konsollen. Da der ikke bruges dynamisk memory behøves der ikke gøres yderligere i constructoren.

Returnerer: Ingenting, da det er en constructor.

Parametre: Ingen.

void post(std::string, int)

Beskrivelse: Denne funktion bruges, når der ønskes at indsættes data i køen. Funktionen tager to parametre, som er tilsvarende til dem der er i UARTItem klassen. Som det kan ses på klassediagram (Figur 109) modtager selve køen (*items*) kun UARTItems. Dette betyder, at post funktionen sørger for at oprette en instans af UARTItem klassen og pushe denne til køen. For at brugeren af klassen ikke behøver gøre dette, sker dette automatisk.

Returnerer: void.

Parametre: *std::string data*. Modtager den streng af chars der skal sendes. *Int bytes*. Antallet af bytes der skal sendes fra den givne streng.

UARTItem get()

Beskrivelse: Denne metode skal udelukkende kaldes fra UART klassen, som køres i egen tråd. Get metoden fjerner det første UARTItem i køen og returnerer dette. Derefter er det den klasse der har kaldet get-metoden der ejer beskeden, og som står for at deallokere den.

Da UARTQueue kun indeholder UARTItem objekter der er gemt på stacken vil dette være nemt, da de automatisk bliver slettet, når UARTItem objektet går ud af scope.

Returnerer: Første UARTItem i køen.

Parametre: Ingen.

~UARTQueue()

Beskrivelse: Dette er constructoren. Ligesom constructoren udskriver denne funktion blot en besked til konsollen, for at der kan foretages enhedstest af klassen. I den færdige version af programmet vil denne besked ikke være synlig for end-user.

Returnerer: Ingenting.

Parametre: Ingen.

11.3.7.2. Medlemsattributter

Der er tre medlemsattributter i UARTQueue klassen:

- QMutex
- QWaitCondition
- Std::Queue<UARTItem>

QMutex er en klasse fra Qt-frameworket. QMutex klassen fungerer som en cross-platform mutex wrapper. Dvs. QMutex klassen kan bruges både på Windows og Linux i stedet for at bruge en platform-bestemt mutex.

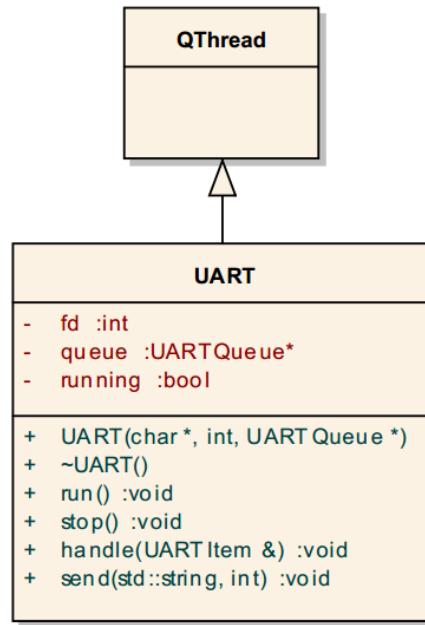
Mutexes bruges til at udføre "mutually exclusive" operationer. Dette er nødvendigt i UARTQueue klassen, for at sikre tråd-sikkerheden. I et flertrådet miljø kan der være to tråde der udfører kommandoer på samme objekt, hvilket er uønsket opførsel. Mutexen bliver brugt i **post** og **get** metoderne, da disse udfører operationer på queue'en.

QWaitCondition er igen en klasse fra Qt-frameworket. QWaitCondition er en conditional wrapper, og bruges til at give et signal fra **post** metoden til **get** metoden, når der indsættes data i køen. Når køen er tom og **get** metoden kaldes, vil denne blokere og vente, indtil der modtages et signal på QWaitCondition objektet. Når der fås et signal, betyder dette, at der er indsættet data i køen, og at denne derfor ikke længere er tom. **Get** metoden vil derfor køre videre og trække objektet ud af køen og returnere det.

Queue<UARTItem> er kernen i UARTQueue klassen. Denne attribut er selve køen, hvor beskederne bliver gemt og opbevaret, indtil de igen trækkes ud. Det er denne attribut der er beskyttet af den førnævnte mutex.

11.3.8. Implementering af UART klassen

UART klassen er opbygget efter følgende klassediagram.



FIGUR 110 - KLASSEDIAGRAM FOR UART KLASSEN

UART klassens funktion er at afvente at der bliver sendt en besked til UARTQueue køen. Når der er en besked i denne kø sendes denne til PSoC over UART. For at sende data over UART bruges Raspberry Pi's GPIO pins. På denne er to GPIO pins dedikeret til UART transmission. UART klassen skal udelukkende sende kommandoer til PSoC, og derfor bruges kun Tx GPIO pinnen.

11.3.8.1. Klassebeskrivelse

UART(char *, int, UARTQueue *)

Beskrivelse: Constructor. Denne åbner forbindelsen til Raspberry'ens serielle interface. Constructoren modtager tre parametre. Den første er navnet på den /dev/ fil som skal bruges. Raspberry'ens GPIO UART pins kan skrives til ved hjælp af "/dev/ttyAMA0"-devicen. Parameter nummer to er en int, der indeholder baud raten på UART transmissionerne. Den sidste parameter er UARTQueue, hvor alle de beskeder der skal sende hentes fra.

Constructoren opretter også en instans af Log klassen, til at skrive til loggen ved forbindelsesfejl.

Returnerer: Ingenting.

Parametre: const char *device, int baudRate, UARTQueue *queue.

~UART()

Beskrivelse: Destructor. Nedlægger forbindelsen til /dev/ device filen.

Returnerer: Ingenting.

Parametre: Ingen.

Void run()

Beskrivelse: Klassens kerne. Når tråden er oprettet vil denne funktion køre, indtil tråden anmodes om at stoppe. Run er en overskrivelse af den virtuelle metode "run" fra klassen QThread. Når der oprettes et UART objekt, og **start()** kaldes på dette objekt, så vil **start()** oprette en tråd og kalde **run()**. **Start()** er en medlemsfunktion i QThread, og er altså en funktion der er arvet derfra.

Det eneste run funktionen står for at gøre er: receive, handle, delete. Dette er en design-form der bruges, når der arbejdes med beskedkøer. Først modtages næste besked i køen, derefter sendes beskeden til en handlefunktion (handler), og sidst slettes beskeden. Da beskedkøen indeholder beskeder gemt på stacken er "delete" delen automatisk, i det objektet destrueres, når det går ud af scope, hvilket sker efter beskeden er blevet håndteret af handleren.

Returnerer: void.

Parametre: Ingen.

Void stop()

Beskrivelse: Anmoder tråden om at afslutte. Funktionen sætter "running" variablen til "false", hvilket vil stoppe while loopet i **run()** metoden.

Returnerer: void.

Parametre: Ingen.

Void handle(UARTItem &)

Beskrivelse: Håndterer beskederne der kommer til UART klassen. Denne klasse sørger for at splitte besked-objektet op, og kalde **send()**-metoden med de korrekte parametre.

Returnerer: void.

Parametre: UARTItem &item: Beskeden der skal håndteres.

Void send(std::string, int)

Beskrivelse: Sender kommandoen på UART Tx GPIO pinnen. Dette gøres ved hjælp af et eksternt bibliotek, kaldet *wiringPi*, som vil være beskrevet længere nede i dokumentationen.

Send tager to parametre. En string med data der skal sendes, og en int der fortæller hvor mange chars fra den førnævnte string der skal sendes.

[11.3.8.2. Brug af wiringPi i UART](#)

For at oprette forbindelse til GPIO pins på Raspberry Pi kan der entes skrives til en /dev/ fil. Denne er koblet sammen med et kerne modul, som sørger for at sætte pinnen HIGH eller LOW, alt efter hvad der skrives til filen. I stedet for at bruge denne low-level fremgangsmåde kan der bruges et library der giver

adgang til et C++ interface til Raspberry'ens GPIO pins, på et langt højere abstraktionsniveau. Dette library er kaldet "*wiringPi*".

På Raspberry'en kan *wiringPi's "gpio utility"* bruges. Denne giver en funktionalitet til at læse tilstanden på alle GPIO pins på Raspberry'en.

På Figur 111 er der vist et screendump af outputtet fra denne kommando.

B Plus											
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
		3.3v			1	2		5v			
2	8	SDA.1	IN	1	3	4		5V			
3	9	SCL.1	IN	1	5	6		0v			
4	7	GPIO. 7	IN	0	7	8	1	ALT0	TxD	15	14
		0v			9	10	1	ALT0	RxD	16	15
17	0	GPIO. 0	IN	0	11	12	0	IN	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13	14			0v		
22	3	GPIO. 3	IN	0	15	16	0	IN	GPIO. 4	4	23
		3.3v			17	18	0	IN	GPIO. 5	5	24
10	12	MOSI	IN	0	19	20			0v		
9	13	MISO	IN	0	21	22	0	IN	GPIO. 6	6	25
11	14	SCLK	IN	0	23	24	0	IN	CE0	10	8
		0v			25	26	0	IN	CE1	11	7
0	30	SDA.0	IN	0	27	28	0	IN	SCL.0	31	1
5	21	GPIO.21	IN	0	29	30			0v		
6	22	GPIO.22	IN	0	31	32	0	IN	GPIO.26	26	12
13	23	GPIO.23	IN	0	33	34			0v		
19	24	GPIO.24	IN	0	35	36	0	IN	GPIO.27	27	16
26	25	GPIO.25	IN	0	37	38	0	IN	GPIO.28	28	20
		0v			39	40	0	IN	GPIO.29	29	21

BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
B Plus										

FIGUR 111 - SCREENDUMP AF GPIO READALL KOMMANDOEN

Ved hurtig inspektion af listen kan det ses, at der er en Tx pin, som står ud for "Physical 8" på højre handel af tabellen. For at koble denne til *wiringPi* C++ interfacet skal der bruges et andet nummer, end det fysiske. Ved første øjekast vil det give mening at bruge "wPi" nummeret. Dette er dog forkert, da *wiringPi* biblioteket af den ene eller anden årsag har valgt at bruge "BCM" numrene i stedet.

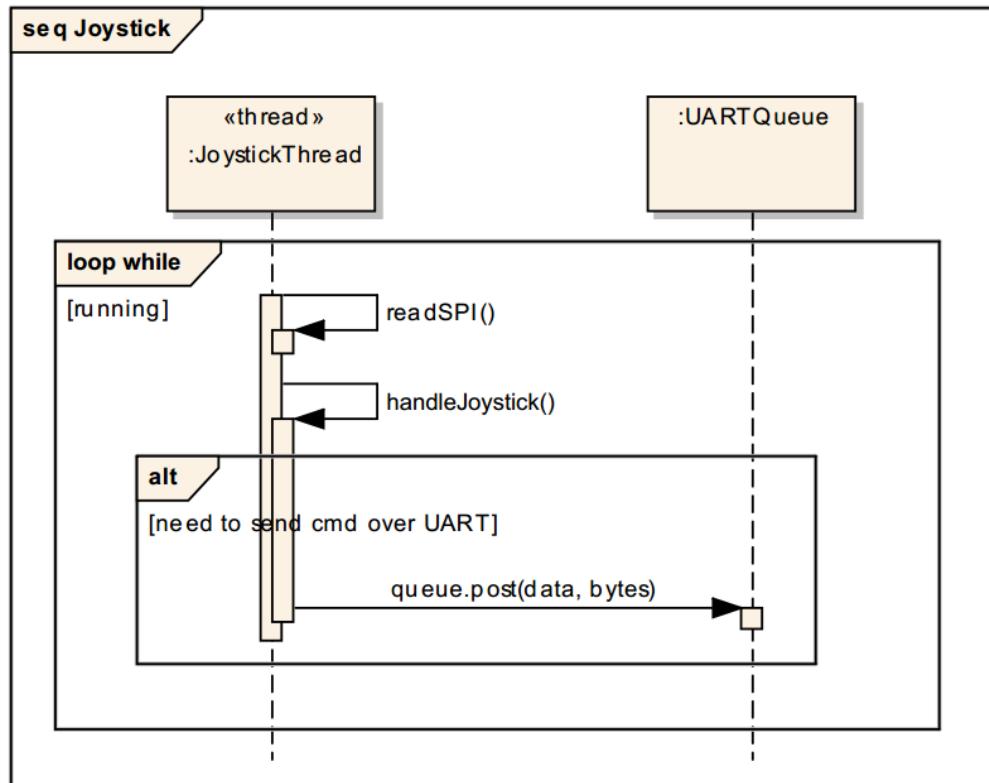
wiringPi giver en række funktioner der kan bruges, så det bliver nemmere at sende og modtage data fra UART. Der kaldes en *wiringPiSetupGpio()* funktion, og efterfølgende kan det serielle interface bruges.

11.3.9. Design af JoystickThread

Joysticket bruges til at styre pistolens bevægelse. Sammenkoblingen mellem SPI-sensoren og hovedprogrammet håndteres af JoystickThread-klassen.

JoystickThread klassen skal være en tråd, så den ikke blokerer hovedprogrammet, når der polles for joystick-værdier fra ADC-sensoren (via SPI). Idéen i JoystickThread-klassen er at lave en klasse, der står for at hente data fra ADC'en til programmet, evaluere de aflæste værdier, og hvis joysticket er bevæget nok til den ene eller anden side, så sendes der en kommando til PSoC'en, som styrer motorerne. Alle kommandoer til PSoC'en sendes gennem UART-klassen²⁶.

Joystick-klassens overordnede funktionalitet er beskrevet i følgende sekvensdiagram.



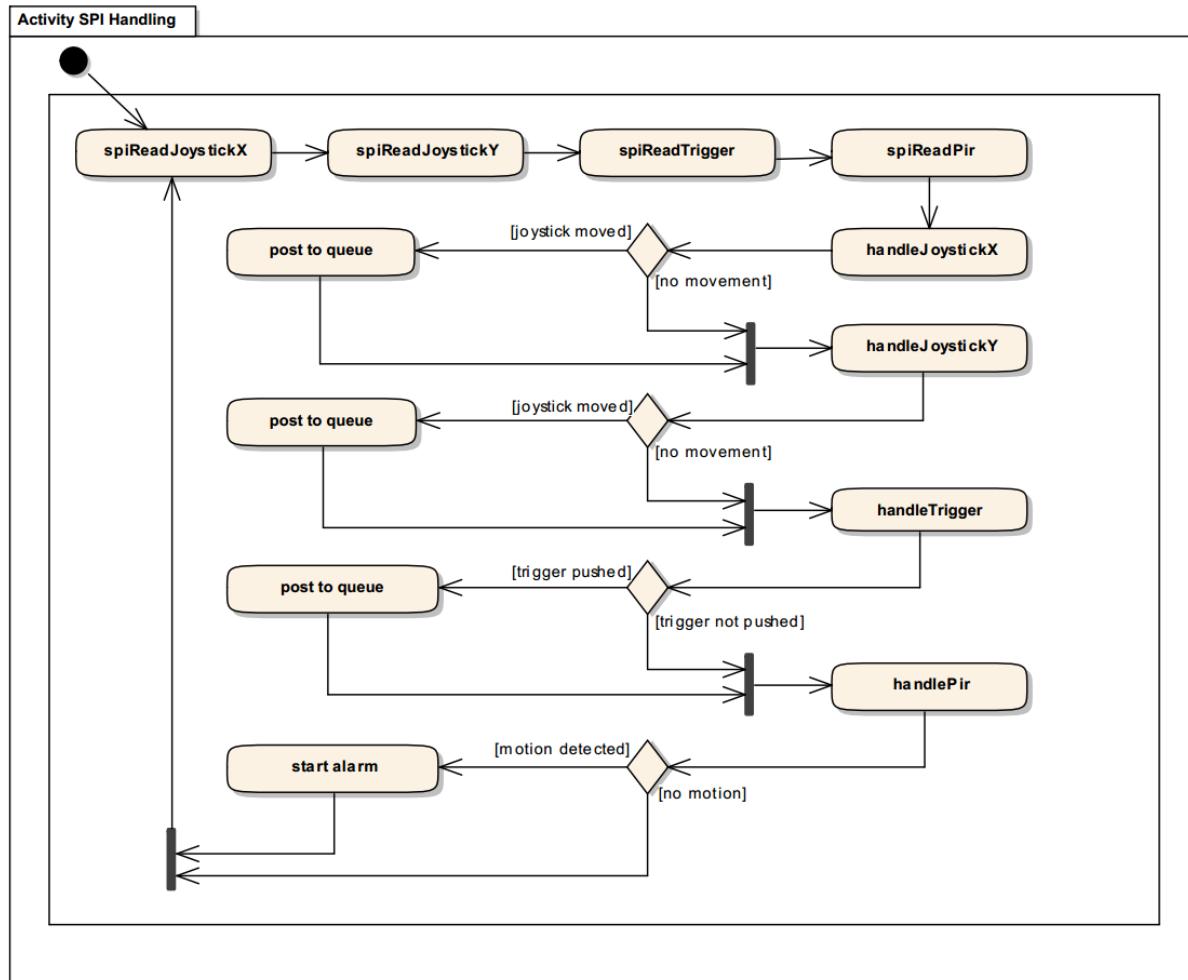
FIGUR 112 - SEKVENSDIAGRAM FOR JOYSTICK FUNKTIONALITETEN

På Figur 112 ses det, at JoystickThread kommunikerer med UARTQueue, og at kommunikationen er envejs. JoystickThread klassen står for konstant at aflæse SPI-værdierne (joysticks X og Y værdier, trigger-værdien og PIR-senorens værdi). Når værdierne er aflæst tjekkes de, og hvis nogle bestemte kriterier bliver opfyldt, så sendes der en kommando til PSoC'en, for at styre motorerne. Dette er lige med undtagelse af PIR-sensoren. Når der modtages et højt signal fra denne, så sættes højtaleren på Raspberry Pi'en til at afspille en advarselslyd til brugeren.

²⁶ Se afsnit 11.3.7 og 11.3.8 om design og implementering af UART klassen

Joystick-tråden implementeres ved at arve fra Qt-frameworkets "QThread" klasse. Funktionaliteten for programmet placeres i en run-metode, som er en virtuel funktion fra QThread, som overskrives ved hjælp af polymorfi.

Nedenstående figur viser et aktivitetsdiagram der beskriver den interne funktionalitet i run-metoden.



FIGUR 113 - AKTIVITETSDIAGRAM FOR JOYSTICKTHREAD RUN-METODE

Run-metoden er klassens kerne. Det er i denne metode, at trådens funktionalitet befinder sig. Klassen har en række handler funktioner. Efter alle SPI-dataene er læst, bliver de sendt til deres repektive handler funktioner, som tager stilling til værdierne, og udfører kommandoer, hvis de korrekte kriterier er mødt.

Som det ses på aktivitetsdiagrammet (Figur 113), så læses der fra de 4 SPI-slaver. Herefter bliver JoystickX dataen sendt til **handleJoystickX()** metoden. Hvis Joysticket er blevet bevæget, så vil handleren poste en besked til UARTQueue klassen. Det samme sker for **handleJoystickY()**. **handleTrigger()** tjekker i stedet for høj eller lav på triggeren. Hvis triggeren er høj, så skal der postes en kommando om at skyde i UARTQueue klassen. Sidst er der **handlePir()**, som håndterer Pir-dataen. I realiteten skal PIR-sensoren sende et højt signal ud, som bliver oversat til 1023 og sendt over SPI, når master (Raspberry'en) anmoder om det.

Når det hele er kørt igennem, går funktionen tilbage til udgangspunktet og starter forfra.

JoystickThread klassen benytter ikke "mutual exclusive" låse, eller conditionals, da der ikke er flere tråde der retter i noget af det data om JoystickThread klassen benytter sig af.

Alle fire ting der skal aflæses går gennem en ADC²⁷. Dette betyder, at et højt signal returneres som 1023, og et lavt signal som 0. Joysticket har to akser, en X-akse og en Y-akse. X-aksen er den horisontale akse, højre og venstre. Y-aksen er den vertikale akse, altså op og ned.

Joysticks neutrale tilstand er når selve "sticken" står direkte op, og er altså i midten af de to yderpositioner på hver akse. Se Figur 114 for illustration af den neutrale udgangsposition.



FIGUR 114 - JOYSTICK I NEUTRAL POSITION

Når joysticket står som vist på Figur 114 vil begge akser returnerer omkring 512. Da joysticket sender et analogt signal ind i ADC'en, kan det egentlige tal variere en smule.

Bevæges joysticket mod højre stiger X-aksens værdi, indtil den når 1023. Til venstre vil X-aksens værdi falde, indtil den rammer 0. Det samme sker for Y-aksen, når joysticket føres henholdsvis op og ned.

Triggeren er en ganske normal knap, og returnerer høj eller lav. Da denne også går gennem ADC'en, vil LAV returnere 0, mens HØJ vil returnere 1023.

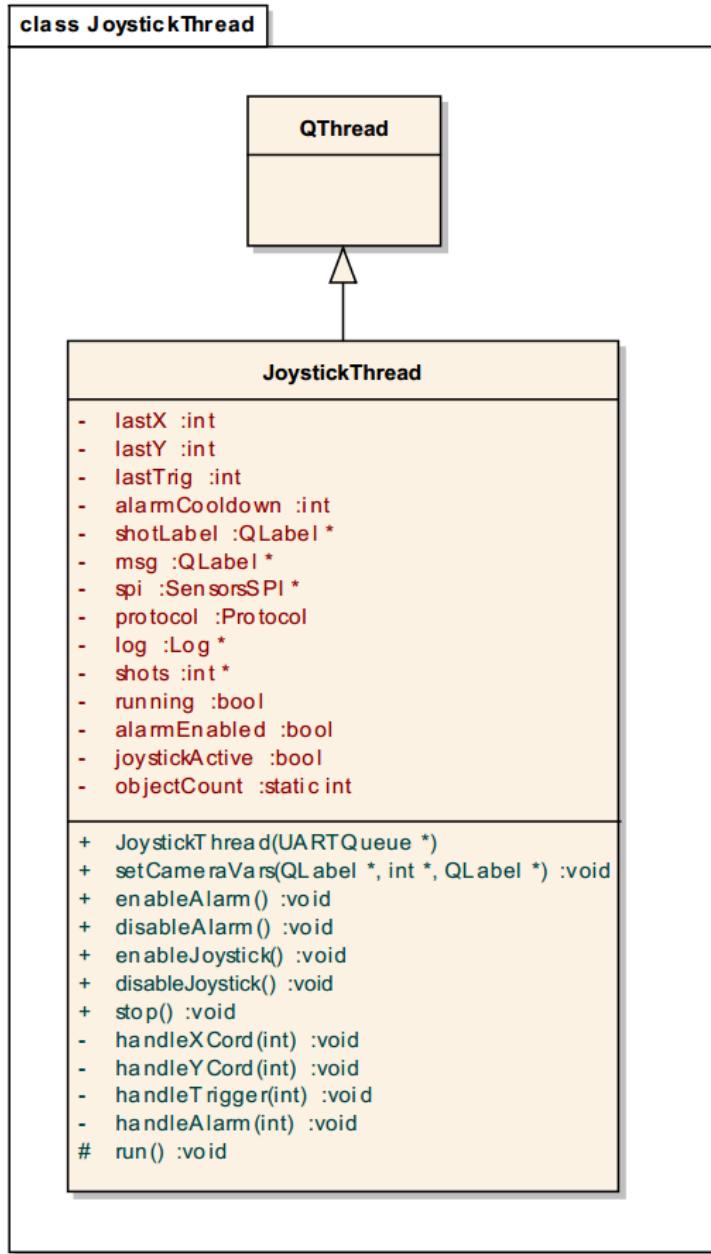
PIR sensoren returnerer ligeledes et højt eller lavt signal, og vil derfor returnere omkring 700-800, da den udsender en spænding på 3.3V.

²⁷ Analog-to-digital converter

11.3.10. Implementering af JoystickThread

Implementeringen af JoystickThread klassen er lavet ud fra sekvensdiagrammet (Figur 112) og aktivitetsdiagrammet (Figur 113). Den endelige implementering afviger en smule fra aktivitetsdiagrammet, i det der er flere tjek.

Det samlede klassediagram for JoystickThread klassen kan ses på Figur 115



FIGUR 115 - KLASSEDIAGRAM FOR JOYSTICKTHREAD

Alle data læses fra SensorsSPI objektet (*spi*). Herefter sendes den aflæste data til *handleXCord*, *handleYCord*, *handleTrigger* og *handleAlarm*. Alt dette sker i *run()* funktionen, som kører, når tråden startes. Som for nævnt er *run()*-metoden en virtuel funktion fra **QThread** klassen, som JoystickThread arver

fra. Da det er en virtuel funktion kan der laves en ny implementering af denne, som passer til JoystickThread klassen.

Ud fra aktivitetsdiagrammet for *run()*-metoden (Figur 113) har ét tjek på handleJoystick-metoderne. Dette tjek er beskrevet som "joystickMoved". Dette er funktionalitet der er i selve handle metoderne. I implementeringen af JoystickThread er der dog indsat tjek af, om Joystick, Trigger og Alarm er aktiveret. Dette gøres for at kunne deaktivere de respektive dele af klassen, når disse ikke skal bruges. Når systemet er logget ud (og viser log ind skærmen), så vil Pir-sensoren stadig kunne være aktiv, men Joystick og Trigger er deaktiveret.

Implementeringen i C++ ser således ud:

```

50     if (joystickActive) {
51         xCord = spi->JoystickX();
52         yCord = spi->JoystickY();
53         trigger = spi->JoystickTrig();
54     }

```

FIGUR 116 - UDSNIT FRA IMPLEMENTERING AF JOYSTICKTHREAD::RUN()

På denne måde er alle SPI-læsninger og handlers pakket ind. Til de to bool attributter (*joystickActive* og *alarmEnabled*) er der tilsvarende setter-metoder. *enableAlarm()*, *disableAlarm()* og tilsvarende for Joysticket. Da triggeren er en del af styringen af HDT er denne koblet sammen med *joystickActive* tjekket. Hvis *joystickActive* er *false*, så virker triggeren heller ikke.

Handlerne til de forskellige aflæsninger fungerer stort set også som specificeret på aktivitetsdiagrammet, men der er mindre ændringer, så der ikke bliver sendt flere UART-kommandoer end højst nødvendigt. Klassen er forsøgt implementeret, så der kun sendes én kommando til PSoC'en, hver gang joysticket bliver bevæget fra ét "område" til det næste. For at opnå denne funktionalitet er der i Joystick-klassens handlerfunktioner defineret nogle intervaller for, hvor Joysticket skal være, før der sker noget nyt.

JoystickThread'ens *handleXCord()* metode er taget med som eksempel, for at vise, hvordan de forskellige handlers er implementeret. Alle handler metoderne er bygget op på stort set samme måde. De tjekker alle, hvad der blev registreret fra tidligere, og hvis dette input har ændret sig tilstrækkelig meget, så sker der en event. Dette kan enten være en kommando der postes til UARTQueue'en, eller at alarmen går i gang.

```

79 void JoystickThread::handleXcord(int xCord) {
80     Protocol::CMDs direction;
81     int xDelta = xCord - 512;
82
83     if (xDelta > 0) {
84         direction = Protocol::CMD_RIGHT;
85     }
86     else {
87         xDelta *= (-1);
88         direction = Protocol::CMD_LEFT;
89     }
90
91     if (xDelta < 100 && lastX != 0) {
92         lastX = 0;
93         uartQueue->post(protocol.constructString(Protocol::CMD_FULLSTOP, '0'), 4);
94     }
95     else if (xDelta < 400 && xDelta > 100 && lastX != 1) {
96         lastX = 1;
97         uartQueue->post(protocol.constructString(direction, '1'), 4);
98     }
99     else if (xDelta > 400 && lastX != 2) {
100        lastX = 2;
101        uartQueue->post(protocol.constructString(direction, '2'), 4);
102    }
103 }
```

FIGUR 117 - JOYSTICKTHREAD::HANDLEXCORD()

På Figur 117 ses den egentlige implementering af JoystickThread klassens *handleXcord()* metode. Fra linje 80-89 tjekkes der blot om der er modtaget en negativ eller positiv værdi. Hvis værdien der er modtaget er negativ, så konverteres den til positiv (ved at gange med minus en), og retningen motoren skal køre sættes til venstre. Hvis værdien er positiv, så sættes retningen blot til højre.

Den spændende del kommer fra linje 91 og ned. Der er en if-else if sætning med 3 muligheder. Første if tjekker om joysticket står i "idle"-position, altså den position joysticket står i, når det er sluppet. Hvis dette er tilfældet og *lastX* er lig med 0, så sendes der en "FULLSTOP" kommando til PSoC'en. *lastX* er en variabel der kan have tre mulige værdier. 0 (stoppet), 1 (hastighed 1) og 2 (hastighed 2). Denne er implementeret, for at gøre det nemmere at tjekke, hvad sidste kommando afsendt er. De to andre else-if statements tjekker blot om joysticket er mellem 100 og 400 eller over 400, for henholdsvis "hastighed 1" og "hastighed 2".

Hvis joysticket bevæges i højre retning og står så der aflæses en X-værdi mellem 100 og 400, så vil der blive afsendt en kommando til PSoC'en, og *lastX* vil blive sat til 1. Alle aflæsninger herefter vil ikke sende kommandoer, da *lastX* stadig er lig med 1. Herved vil næste kommando først blive afsendt, når joysticket bliver bevæget så meget, at det går ind i et andet interval.

11.3.10.1. Klassebeskrivelse

Dette er en kort beskrivelse af JoystickThread klassens metoder, og deres funktion i klassen.

JoystickThread()

Parametre: UARTQueue *uartQueue.

Returnerer: Ingenting. Constructor.

Beskrivelse:

Opretter nyt SPI-objekt, så der kan læses fra sensorerne. Opretter et nyt Log-objekt, som bruges til at logge beskeder ved forskellige events (f.eks. når der affyres skud). Opretter også relation til *uartQueue*, som bruges til at poste kommando-beskeder.

Alle variabler bliver også initialiseret i constructoren, så der ikke forekommer fejl i programmet, når disse læses fra.

setCameraVars()

Parametre: QLabel *shotLabel, int *shots, QLabel *msg.

Returnerer: void.

Beskrevelse:

Sætter variabler der bruges af CameraFeed klassen. Grunden til dette er, at hver gang CameraFeed (hovedmenu vinduet) lukkes, så bliver objektet slettet. Når CameraFeed menuen åbner på ny, skal der derfor relateres til de nye objekter der er oprettet af klassen.

enableAlarm() og disableAlarm()

Parametre: Ingen.

Returnerer: void.

Beskrevelse:

Sætter *alarmEnabled* til henholdsvis true eller false. True starter læsningen af Pir-sensoren, mens false stopper læsningen og disable alarmen.

enableJoystick() og disableJoystick()

Parametre: Ingen.

Returnerer: void.

Beskrevelse:

Sætter *joystickActive* til henholdsvis true eller false. True vil starte læsningen fra joystick og trigger og aktivere joystick funktionaliteten. False stopper læsningen og deaktiverer ligeledes joystick- trigger funktionalitet.

Stop()

Parametre: Ingen.

Returnerer: void

Beskrivelse:

Sætter *running* variablen til false. Dette får *run()*-metodens while løkke til at afslutte, når den er kørt igennem, hvorefter tråden afslutter.

12. Softwaretest

12.1. Cross compiling til raspberry pi [JDA]

Kompilering på Raspberry Pi kan blive en langvarig proces. Det var derfor oplagt at cross-compile til Raspberry Pi fra en hurtigere enhed, som fx en PC. Crosscompiling har specielt været brugt ved enheds, og integrationstests. Gennem research på internettet er der fundet adskillige guides til dette. I projektet er brugt en guide²⁸ der viser hvordan Qt programmer kan kryds-kompileres til Raspberry Pi. Forudsætningen for at kunne kompilere til Raspberry Pi's arkitektur²⁹ er compiler-toolchain bestående af en compiler³⁰ samt Raspberry Pi's egne biblioteker og headers.

Raspberry Pi's egne biblioteker samt headers kan downloades vha. QtCrossTool³¹ softwaren, direkte gennem en SSH forbindelse til Raspberry Pi.

I projektet bruges Qt version 4.8.5, da nyere versioner ikke understøttes på Raspberry Pi. For at kryds kompilere Qt programmer til Raspberry Pi er det da nødvendigt at have en fungerende version af Qt 4.8.5 eller ældre på PC'en hvorfra kompileringen sker. Når forudsætningerne for krydskompilering er opfyldt, kan der med qmake³² laves en Makefile, som "eksekveres" med kommandoen *make*. Dette foregår i Windows' kommando prompt, **cmd.exe**. Se Figur 118 for eksempel på krydskompilering med qmake.

```
C:\Users\Joachim\Desktop\cross_compile_test>C:\SysGCC\Raspberry\bin\qmake.bat test.pro

C:\Users\Joachim\Desktop\cross_compile_test>make
C:/SysGCC/Raspberry/bin/arm-linux-gnueabihf-g++.exe -c -pipe -Wno-psabi -g -Wall
-W -D_REENTRANT -D_LARGEFILE64_SOURCE -D_LARGEFILE_SOURCE -DQT_GUI_LIB -DQT_CORE_LIB -DQT_HAVE_MMX -DQT_HAVE_3DNOW -DQT_HAVE_SSE -DQT_HAVE_MMXEXT -DQT_HAVE_SSE2 -I../../../../SysGCC/Raspberry/arm-linux-gnueabihf/sysroot/usr/share/qt4/mkspecs/arm-linux-gnueabihf -I. -I../../../../SysGCC/Raspberry/arm-linux-gnueabihf/sysroot/usr/include/qt4/QtCore -I../../../../SysGCC/Raspberry/arm-linux-gnueabihf/sysroot/usr/include/qt4/QtGui -I../../../../SysGCC/Raspberry/arm-linux-gnueabihf/sysroot/usr/include/qt4 -I../../../../SysGCC/Raspberry/include -IDebug -IDebug -o Debug/main.o main.cpp
C:/SysGCC/Raspberry/bin/arm-linux-gnueabihf-g++.exe -o Debug/cross_compile_test
Debug/MainWindow.o Debug/MatrixKeyboard.o Debug/main.o Debug/moc_MainWindow.o
-LC:/SysGCC/Raspberry/arm-linux-gnueabihf/sysroot/usr/lib -LC:/SysGCC/Raspberry/lib -lwiringPi -lQtGui -lQtCore -lpthread

C:\Users\Joachim\Desktop\cross_compile_test>
```

FIGUR 118 - EKSEKVERING AF AUTOGENERERET MAKEFILE FOR ET QT TEST-PROJEKT

Den kompilerede, eksekverbare fil ligger nu i projekt-mappens Debug-folder, og kan kopieres til Raspberry Pi. Projektet der vises kompilert på Figur 118 og bilag er at finde på CD-rom³³.

²⁸ Se guiden her: <http://visualgdb.com/tools/QtCrossTool/>

²⁹ ARM Cortex A7

³⁰ Til kompilering af programmer til Raspberry Pi benyttes GCC compileren

³¹ QtCrossTool kan hentes fra <http://visualgdb.com/tools/QtCrossTool/QtCrossTool.zip>

³² Qt værktøj til at oprette en Makefile ud fra et Qt projekts .pro fil

³³ Se CD-rom, under "Cross compile test".

12.2. Enhedstest af GUI [AEL]

De forskellige enhedstest blev udført ved at compile og køre de forskellige vinduer. Herefter indtastes input, for at se, om systemet reagerer korrekt. Log klassen er testet ved et specifikt test-program.

12.2.1. Login



FIGUR 119 - LOGIN VINDUE VED OPSTART

Dette er skærmen man ser når programmet bliver executed.



FIGUR 120 - LOGIN VINDUE VED PASSWORD_ != 1905

På Figur 120 bliver der testet hvad der sker hvis der indtastes en forkert kode. Resultatet blev som antaget, altså der bliver vist en fejlmeldelse på brugerinterfacet.



FIGUR 121 - TEST AF LOGIN MED PASSWORD_ = 1905

I Figur 121 testes der hvad der sker hvis der indtastes den korrekte kode. Resultatet blev at Login skærmen blev nedlagt og et CameraFeed vindue blev åbnet, altså som forventet.

12.2.2. CameraFeed



FIGUR 122 - CAMERAFEED VINDUE VED OPSTART

På Figur 122 ses CameraFeed vinduet som det ser ud fra opstart.



Antal skud: 0

FIGUR 123 - TEST AF FUNKTIONEN ONAKTIVERPRESSED()

På Figur 123 testes funktionen OnAktiverPressed() og som forventet ses en statusmeddeelse i nederste højre hjørne af skærmen, der fortæller at systemet er aktiveret.



Antal skud: 0

FIGUR 124 - TEST AF FUNKTIONEN ONDEAKTIVERPRESSED()

På Figur 124 testes funktionen OnDeaktiverPressed() og igen som forventet ses en statusmeddelelse i bunden af skærmen der dog her fortæller at systemet er deaktiveret.



FIGUR 125 - TEST AF FUNKTIONEN ONGENLADPRESSED()

På Figur 125 testes funktionen OnGenladPressed(), som det ses ud fra figuren åbnes der et nyt genlad vindue klar til indtastning, og resultatet er dermed som forventet.



Antal skud: 0

FIGUR 126 - TEST AF FUNKTIONEN ONADVARSELRESSED()

På Figur 126 testes funktionen OnAdvarselPressed(), Denne viser en statusmeddeelse i bunden af skærmen der fortæller at den pågældene knap er blevet trykket på.



FIGUR 127 - TEST AF FUNKTIONEN ONLOGUDPRESSED()

Til sidst på Figur 127 testes funktionen OnLogUdpPressed() der skulle nedlægge CameraFeed vinduet og derefter åbne et login vindue. Dette er også her tilfældet og resultatet er dermed som forventet.

12.2.3. Genlad



FIGUR 128 - GENLAD VINDUE VED OPSTART

På Figur 128 ses Genlad vinduet som det ser ud ved opstart, bemærk at her allerede er indtastet 100 skud for at gøre klar til næste test.



FIGUR 129 - TEST AF FUNKTIONEN ONOKPRESSED()

På Figur 129 tester vi OnOkPressed() funktionen som skulle tage det indtastede antal skud og indsætte dette i skud_ variablen som vises i CameraFeed vinduet. Dette kan også ud fra figuren ses at være tilfældet.

12.3. Enhedstest af Log [AEL]

```

1 #include "Log.h"
2
3 void main()
4 {
5     Log log;
6     log.writeLog(Log::skyd);
7 }
```

FIGUR 130 - TEST PROGRAM FOR LOG KLASSEN

På Figur 130 ses Testprogrammet for Log klassen.

```

20150525 01:08:38 - Der blev affyret skud
20150525 01:09:01 - Der opstod en fejl ved oprettelse af forbindelse til UART
20150525 01:09:22 - Systemet blev deaktiveret
20150525 01:09:37 - Systemet blev aktiveret
20150525 01:09:51 - Alarmen blev aktiveret
```

FIGUR 131 - LOG.TXT FØR TEST

```

20150525 01:08:38 - Der blev affyret skud
20150525 01:09:01 - Der opstod en fejl ved oprettelse af forbindelse til UART
20150525 01:09:22 - Systemet blev deaktiveret
20150525 01:09:37 - Systemet blev aktiveret
20150525 01:09:51 - Alarmen blev aktiveret
20150525 01:11:31 - Der blev affyret skud
```

FIGUR 132 - LOG.TXT EFTER TEST

På Figur 131 og Figur 132 ses Log.txt filen før og efter at testprogrammet er blevet kørt og ud fra dette kan det ses at alle de forskellige kommandoer udskriver tekst som de skal og der tilføjes korrekt formaterede tidsstemplere. Derudover kan det ses at der efter programmet er kørt er blevet tilføjet det ekstra notat som der burde, altså er resultatet som forventet.

12.4. Integrationstest af GUI [AEL]

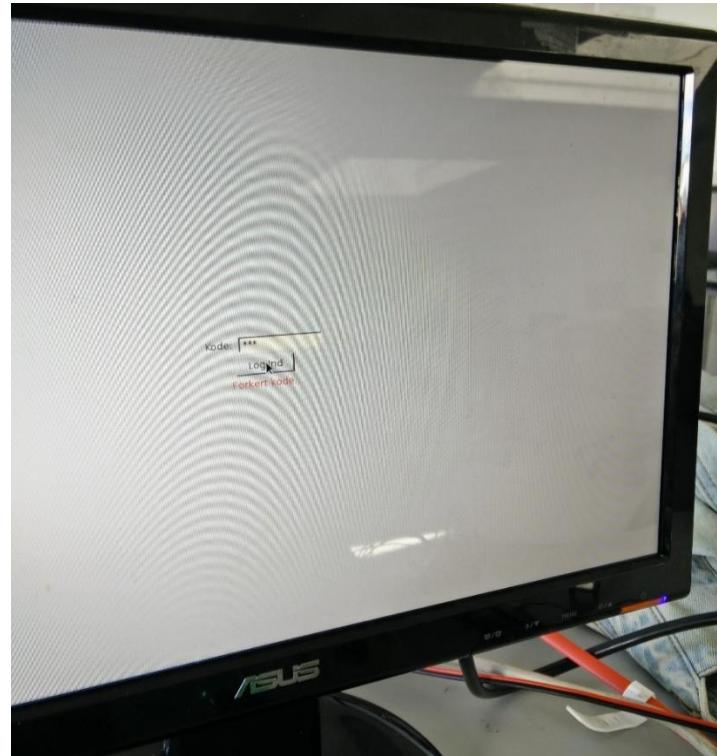
Koden fra enhedstesten er blevet tilpasset og compilet til projektets raspberry pi og derefter testet igen for at se om funktionaliteten stadig var intakt og kunne køre på det endelige system.

12.4.1. Login



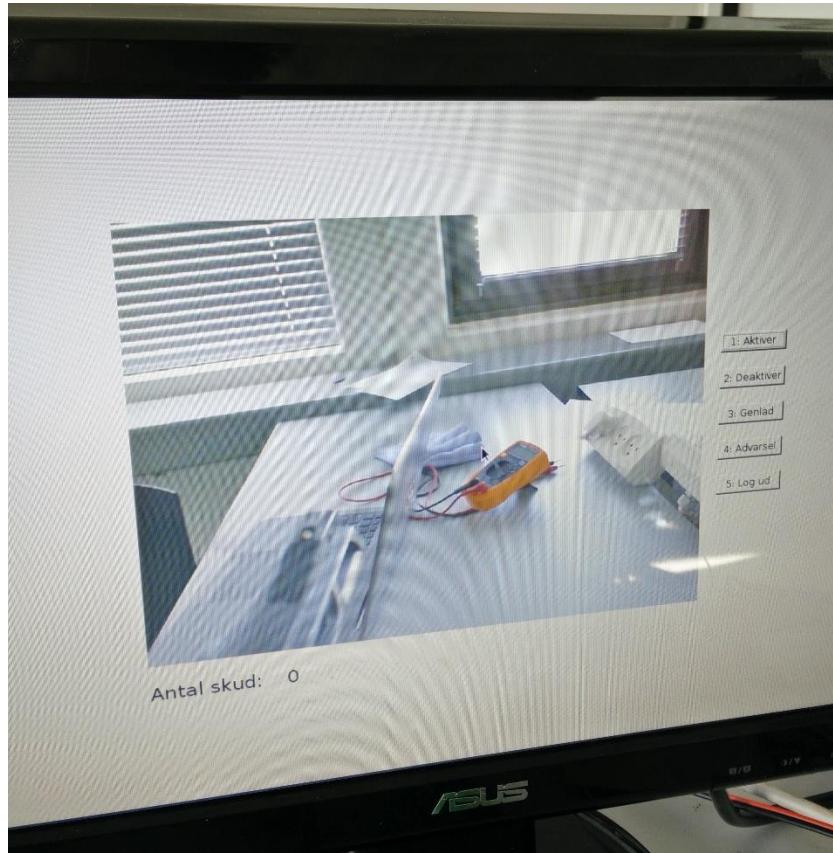
FIGUR 133 - LOGIN VINDUET VED OPSTART

Som det ses på Figur 133 starter programmet som det skal.



FIGUR 134 – LOGIN VINDUE VED PASSWORD_ != 1905

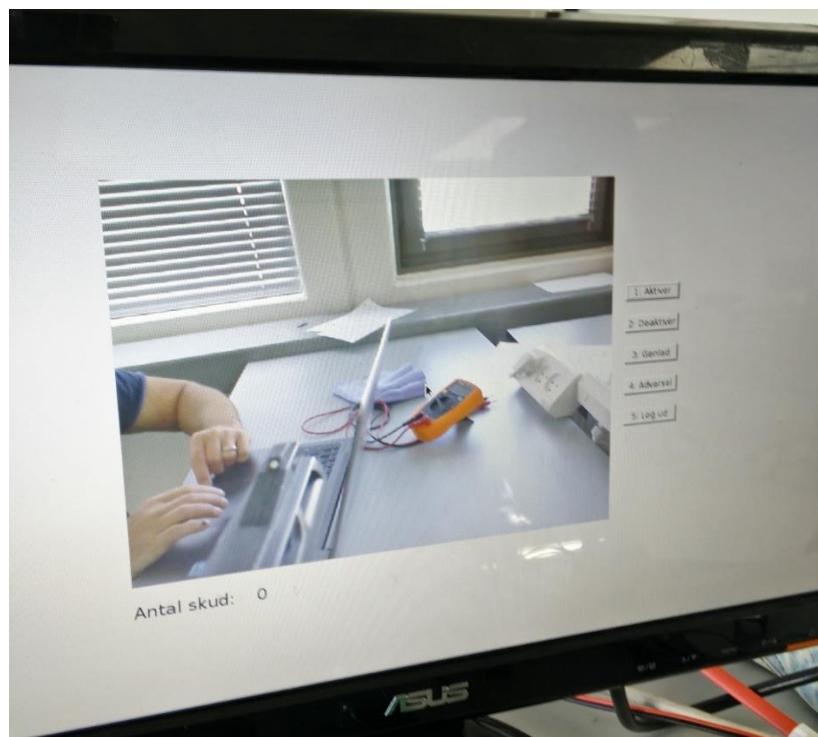
På Figur 134 ses det at ved en forkert adgangskode vises en fejlbesked, hvilket betyder at resultatet er som forventet.



FIGUR 135 - LOGIN VINDUE VED PASSWORD_ = 1905

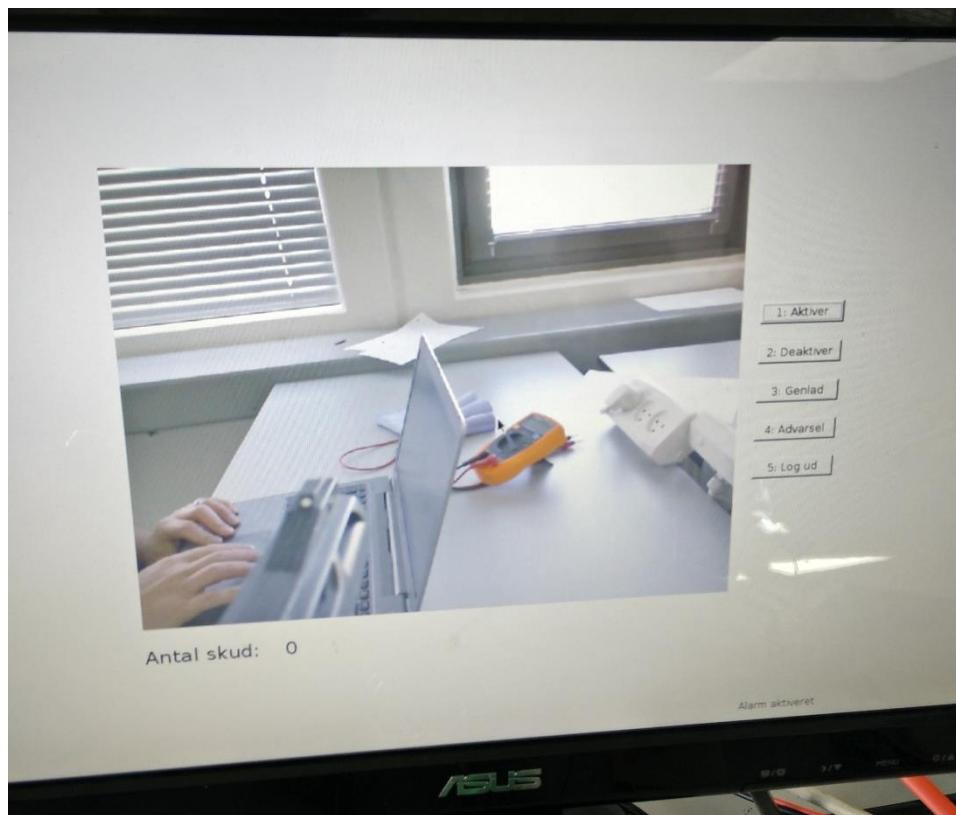
Som det ses på Figur 135 vil det at trykke på Log ind mens password_ = 1905, give adgang til systemet og dermed vise CameraFeed vinduet.

12.4.2. CameraFeed



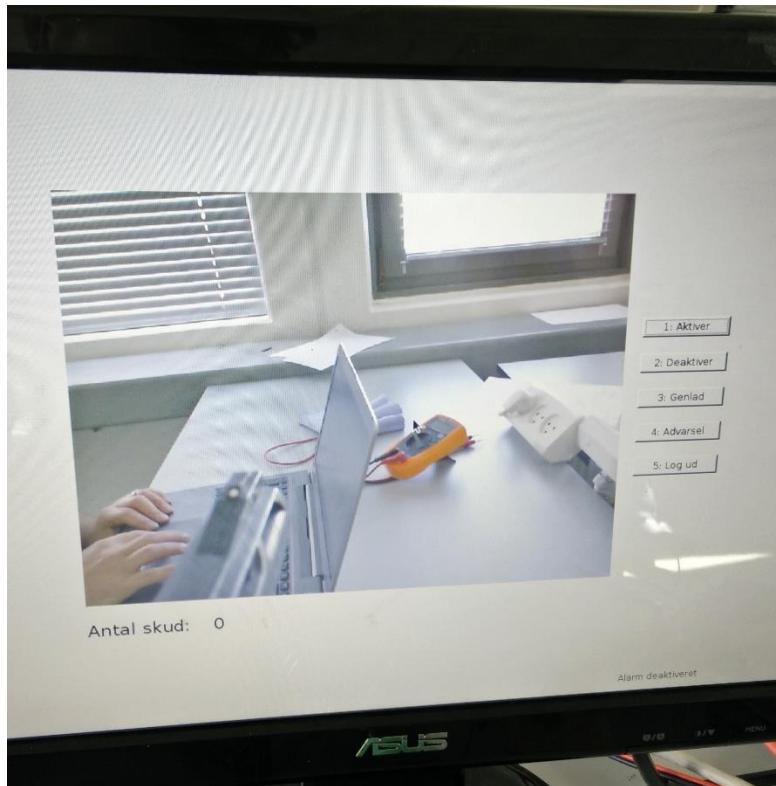
FIGUR 136 - CAMERAFEED VINDUE VED OPSTART

På Figur 136 ses CameraFeed vinduet ved opstart.



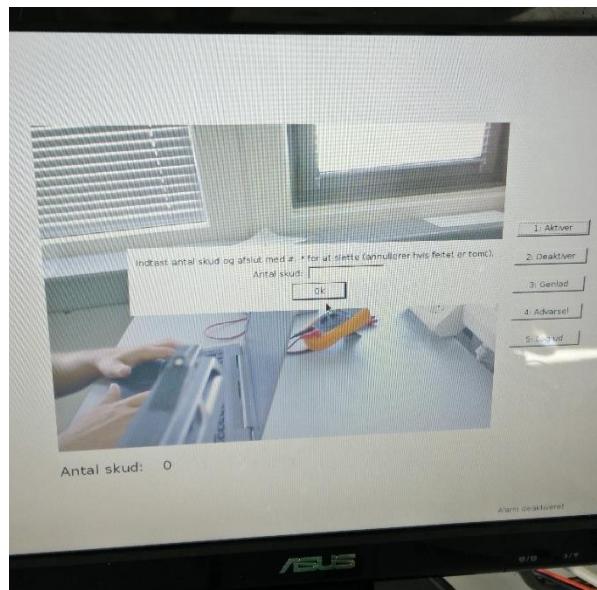
FIGUR 137 - TEST AF FUNKTIONEN ONAKTIVERPRESSED()

På Figur 137 ses en test af funktionen OnAktiverPressed(), og det ses at der her ligesom ved enhedstesten kommer en statusbesked nede i højre hjørne der fortæller at Alarmen er aktiveret.



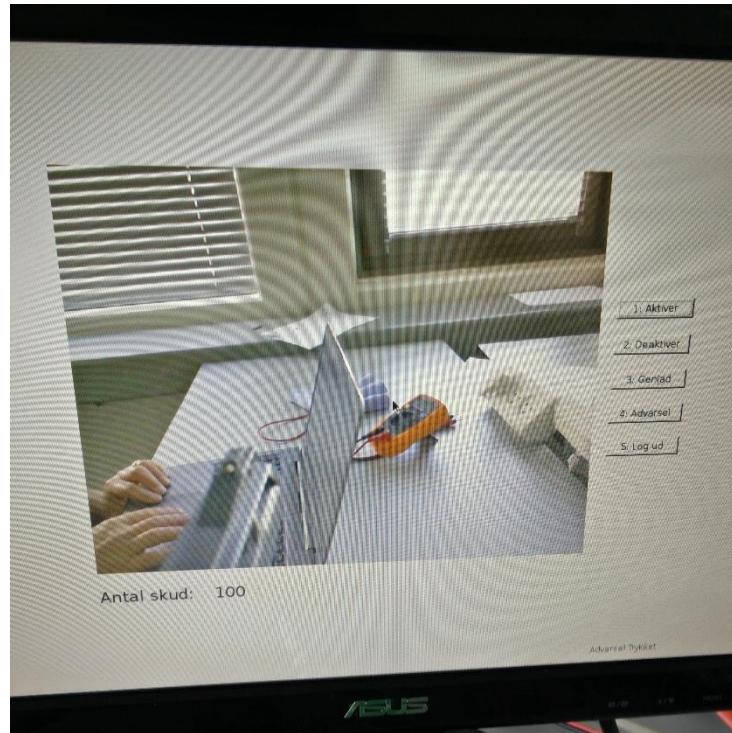
FIGUR 138 - TEST AF FUNKTIONEN ONDEAKTIVERPRESSED()

På Figur 138 ses at ved test af funktionen OnDeaktiverPressed() kommer der også her en statusmeddeelse nede i højre hjørne der fortæller at denne er trykke. Resultatet er dermed som forventet.



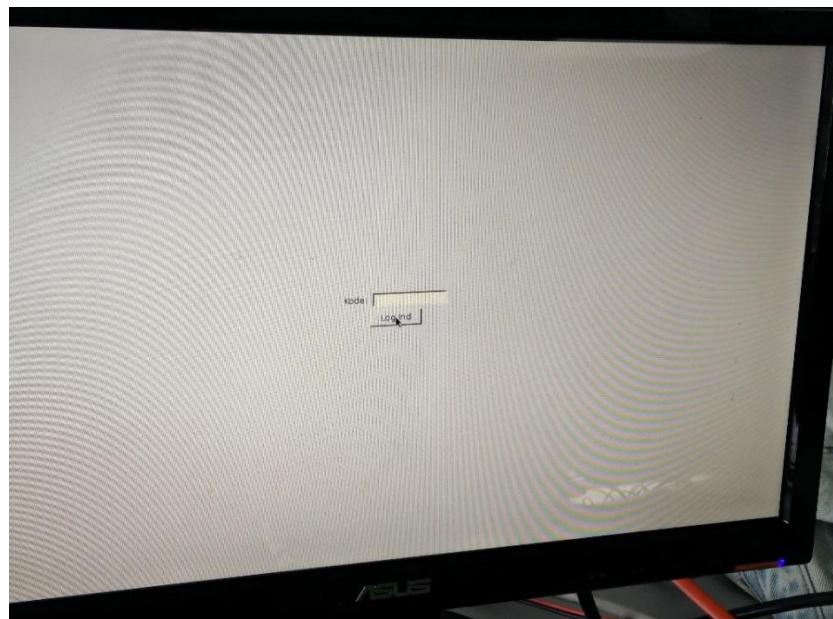
FIGUR 139 - TEST AF FUNKTIONEN ONGENLADPRESSED()

På Figur 139 ses det her at systemet korrekt åbner det nye genlad vindue, bemærk dog at den forklarende statustekst over indtastningsfeltet er blevet ændret en smule så den passer til matrixKeyboardets inputs, Resultatet er altså som forventet.



FIGUR 140 - TEST AF FUNKTIONEN ONADVARSELPRESSED()

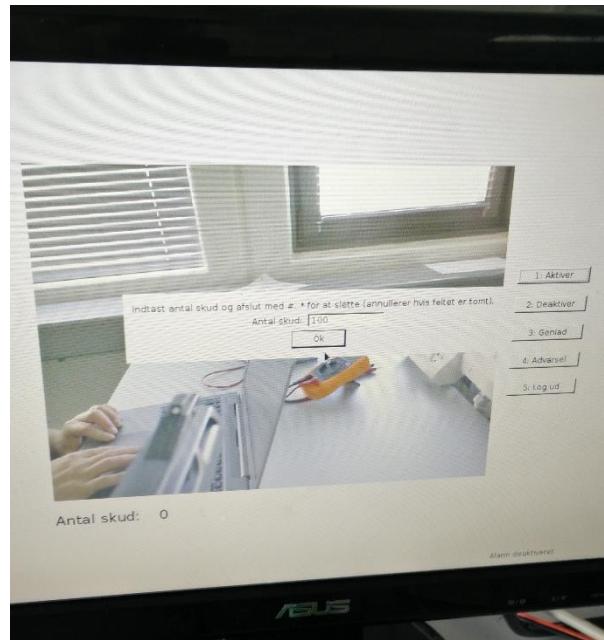
Her ses det på Figur 140 at testen af funktionen OnAdvarselPressed() ganske som forventet viser dette i en opdateret statustekst nede i højre hjørne.



FIGUR 141 - TEST AF FUNKTIONEN ONLOGUPPRESSED()

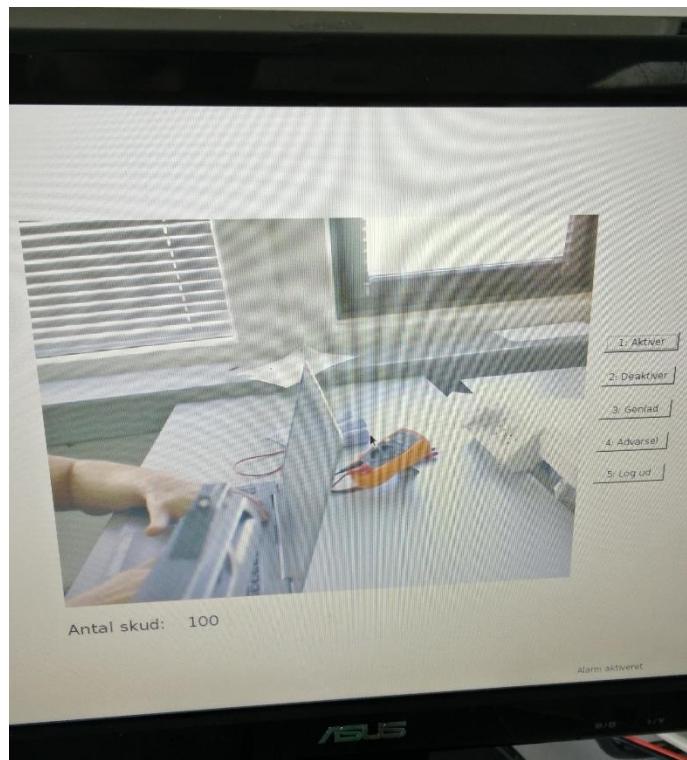
På Figur 141 ses det at ved et tryk på "Log ud" knappen, så lukkes CameraFeed vinduet og et nyt Login vindue oprettes. Resultatet stemmer altså overens med forventningerne.

12.4.3. Genlad



FIGUR 142 - GENLAD VINDUET SOM DET SER UD VED OPSTART

På Figur 142 ses hvordan Genlad vinduet vises ved opstart, bemærk at her er indtastet 100 skud ind i indtastningsfeltet for at gøre klar til næste test, hvilket også blev gjort ved enhedstesten.

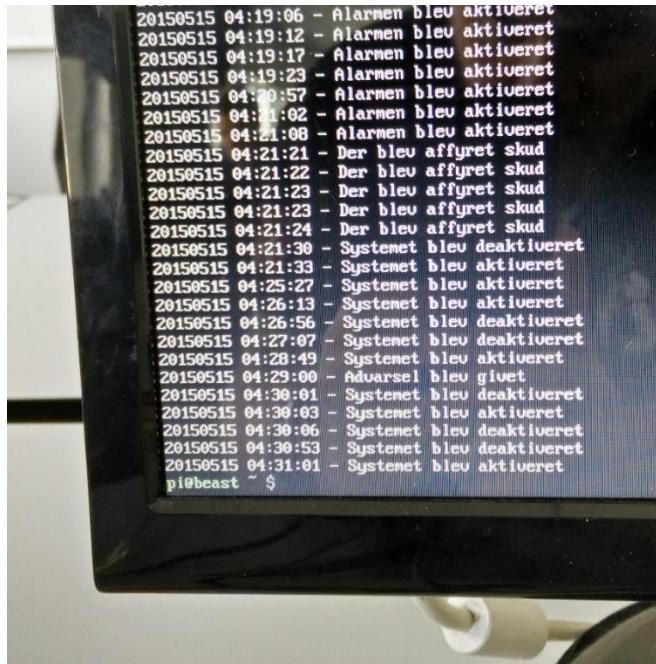


FIGUR 143 - TEST AF FUNKTIONEN ONOKPRESSED()

På Figur 143 ses en test af funktionen OnOkPressed(), her er altså lige blevet klikket på Ok i genlad vinduet, dette er lukket, og det tal der blev skrevet i genlad vinduet er nu blevet indsat og opdateret i CameraFeed vinduet, som det også ses nede i venstre hjørne. Resultatet stemmer her altså godt overens med forventningerne.

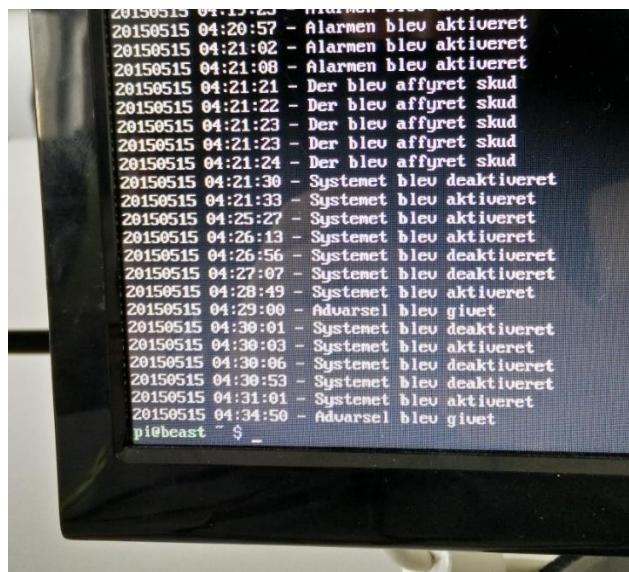
12.5. Integrationstest af Log [AEL]

Denne test blev udført ved at åbne CameraFeed vinduet og trykke på advarselsknappen, dermed skulle der noteres i Log.txt filen at en advarsel er blevet givet.



FIGUR 144 - LOG.TXT FØR TEST

Som det ses på Figur 144 har der været mange noteringer mens systemet er blevet testet, bemærk dog at tiden er lidt ved siden af, og dette skyldes at programmet er kodet til at notere den lokale tid på det Indlejrede Linux System, altså hvad dette er indstillet til, og den tror i dette tilfælde at klokken er godt halv fem om morgenen.



FIGUR 145 - LOG.TXT EFTER TEST

Det kan ses på Figur 145 at på trods af den lidt skæve tid, så er funktionaliteten korrekt, med korret formatering og den korrekte tekst bliver noteret. Derfor må resultatet konstateres at være som forventet.

12.6. Enhedstest af matrixkeyboard [JDA]

I forbindelse med implementeringen af matrixkeyboardet, udarbejdes en enhedstest, til at sikre klassens funktionalitet. Klassen testet ved at oprette et nyt Qt projekt, inkludere matrixKeyboardets header og source filer i projektet. Til testen skrives et lille test program, der kalder klassen **poll** funktion. Da klassen er designet til at køre i sin egen tråd, startes **poll** metoden ved at kalde **QThread::start** gennem et objekt af matrixKeyboardet. Da der i test applikationen ikke oprettes nogen grafiske vinduer, erstattes matrixKeyboard klassens contructors parameter, **QObject *target**, med **NULL**. Koden til test af programmet kan ses på Figur 146.

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    std::cout << "Starting test environment for MatrixKeyboard class... \n";
    wiringPiSetupGpio();
    //Create MK object - No windows, target = NULL
    MatrixKeyboard mk(NULL);

    mk.start();

    return a.exec();
}
```

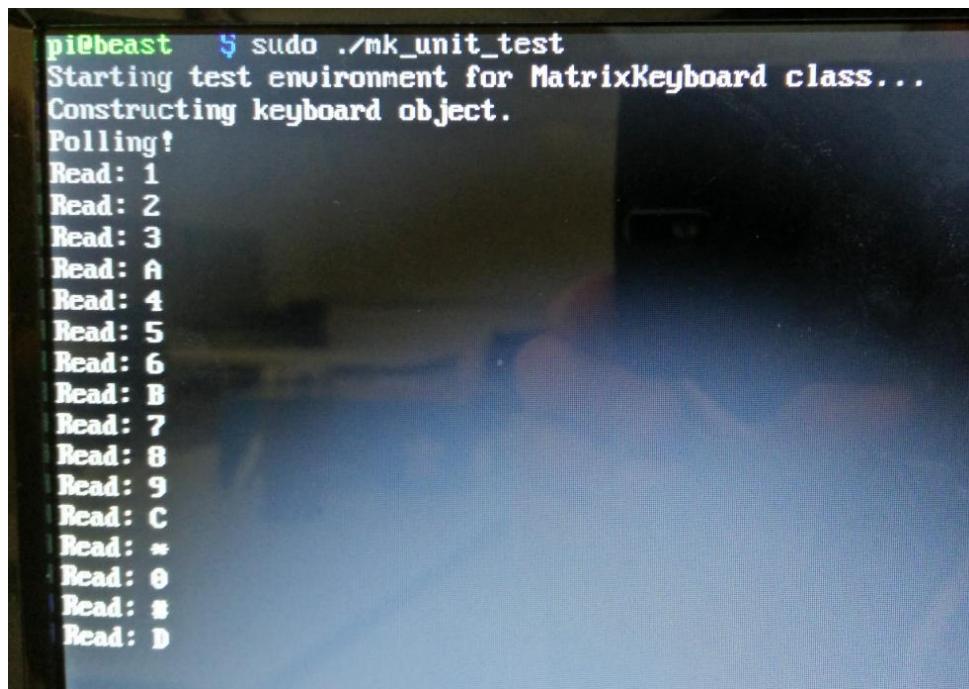
FIGUR 146 - KODEUDSNIT, MAIN FUNKTION TIL ENHEDSTEST AF MATRIXKEYBOARD

Enhedstesten foregår på det Indlejrede Linux System, da der benyttes GPIO pins.

MatrixKeyboard klassen er designet til at sende et QtKeyEvent til et **target** vindue³⁴ i brugerinterfacet. Denne del af kode udkommenteres, og der skrives blot ud til konsollen, hvad der læses fra matrixKeyboardet. Ellers er al kodden fra matrixKeyboard klassen, med i denne test. I integrationstesten³⁵, testes matrixKeyboard klassens evne til at sende QKeyEvents til brugerinterfacets vinduer. På Figur 147 ses et billede af test applikationens output. Billedet er taget efter samtlige knapper er trykket. Under testen blev der lagt vægt på hvorvidt der kun blev udskrevet en enkelt karakter per tryk.

³⁴ Se nøjagtig beskrivelse i afsnit 11.3.2 og 11.3.4, om design og implementering af matrixKeyboardet

³⁵ Se afsnit 12.7 for integrationstest af matrixKeyboard.

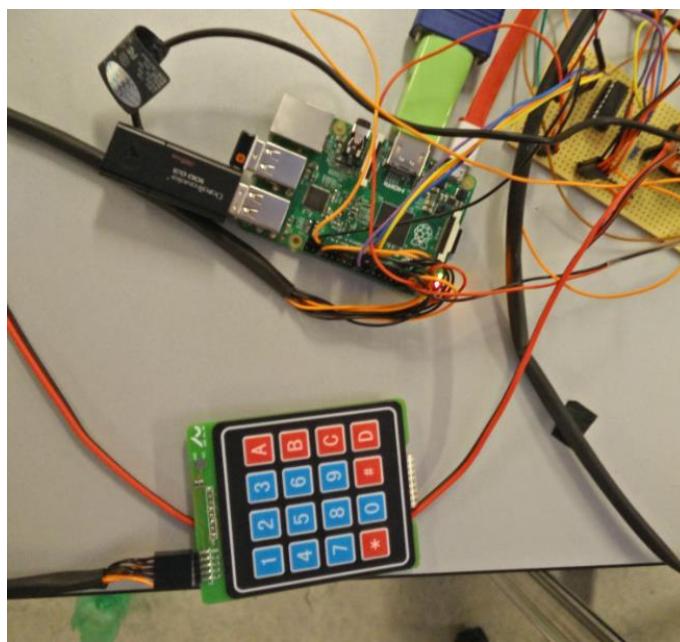


```
pi@beast: ~ $ sudo ./mk_unit_test
Starting test environment for MatrixKeyboard class...
Constructing keyboard object.
Polling!
Read: 1
Read: 2
Read: 3
Read: A
Read: 4
Read: 5
Read: 6
Read: B
Read: 7
Read: 8
Read: 9
Read: C
Read: *
Read: 0
Read: #
Read: D
```

FIGUR 147 - BILLEDE AF TEST-APPLIKATIONENS OUTPUT

Samtlige filer til testen kan findes i bilag³⁶.

Se billede af opstillingen på Figur 148.



FIGUR 148 - OPSTILLING AF TEST MED MATRIXKEYBOARDDET

³⁶ Se CD-rom, under "Matrixkeyboard test application".

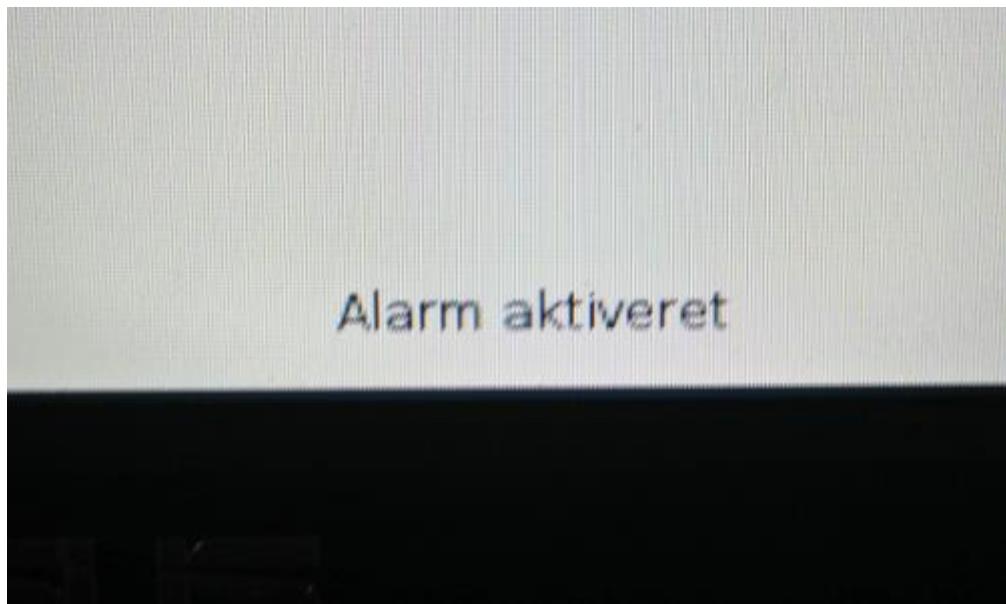
12.7. Integrationstest af matrixkeyboard

For at sikre at matrixkeyboardets fungerer sammen med samtlige elementer af systemet udarbejdes en integratoionstest. I modsætning til enhedstesten, som brugerinterfacet ikke var en del, vil denne blive brugt i integrationstesten. Dette er nødvendigt for at teste Matrikeyboard klassens evne til at sende QKeyEvents til et target vindue i brugerinterfacet. Integrations testen vil foregå ved at se om der sendes de korrekte karakterer til vinderne i brugerinterfacet. Da testen tager udgangspunkt i hvad der vises på brugerinterfacet, er billede af udfaldet at se på Figur 149 til Figur 153.



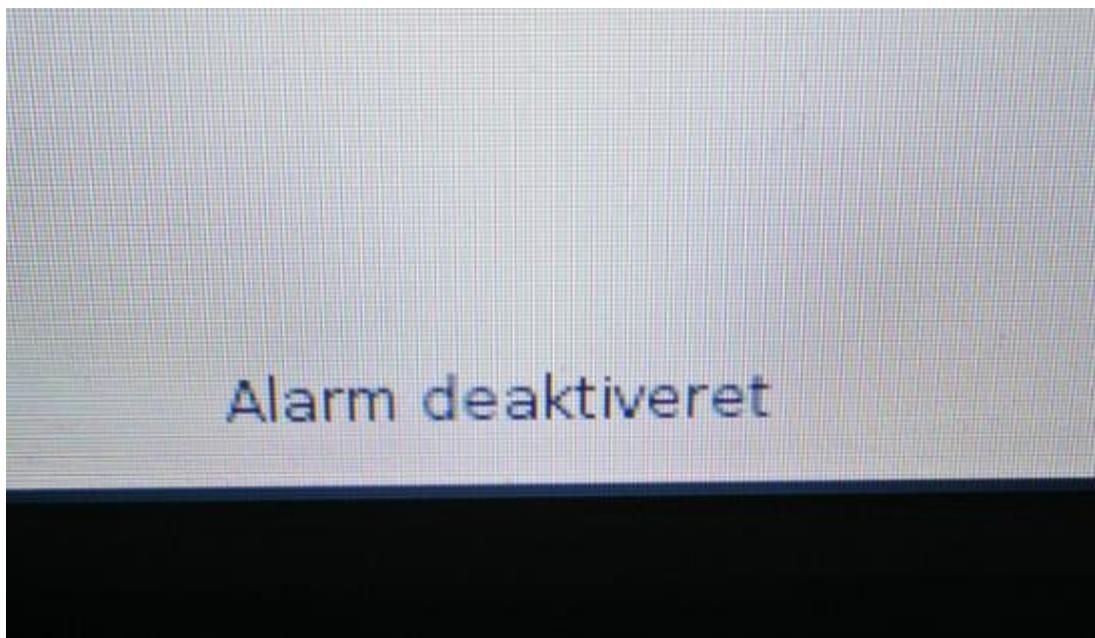
FIGUR 149 - BILLEDE AF BRUGERINTERFACE, EFTER INDTASTNING AF PÅ KODE PÅ MATRIXKEYBOARDDET

På Figur 149 ses, udfaldet af indtastning af kode på matrixkeyboardet. Indtastningen vises på skærmen



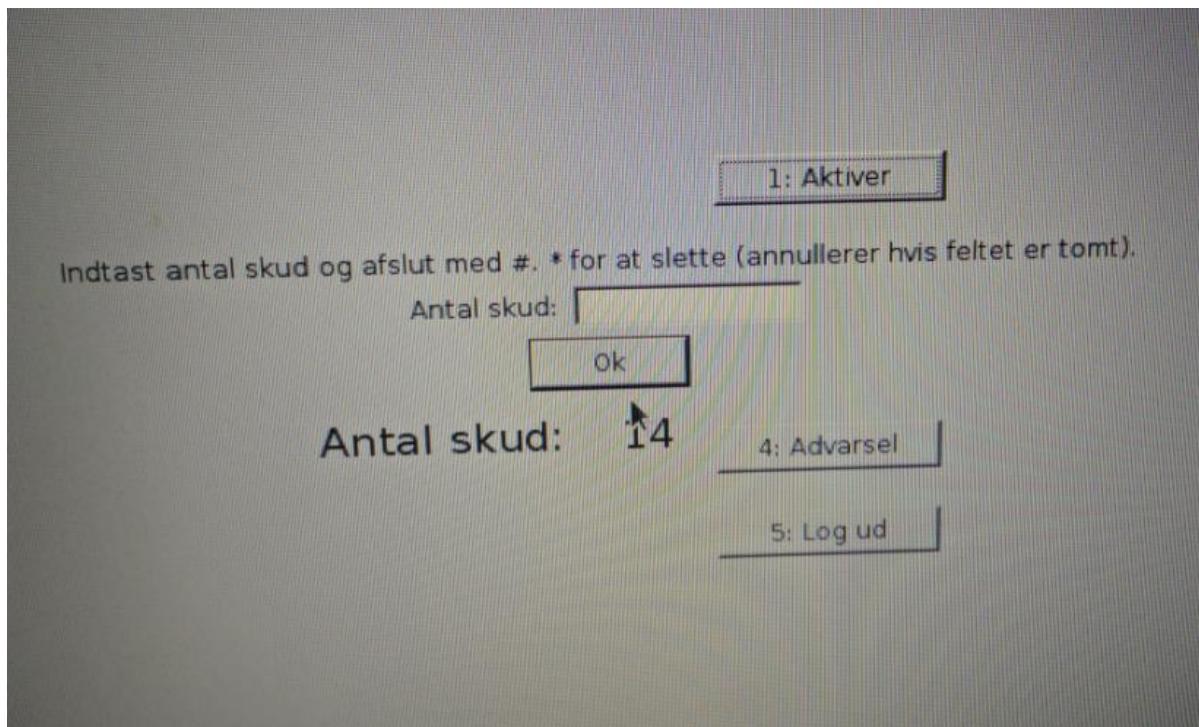
FIGUR 150 - TAST '1' TRYKKET - VISER BESKED MED "ALARM AKTIVERET"

På Figur 150 og Figur 151 ses hhv. udfaldet når knapperne '1' og '2' trykkes. Dette resulterer i alarmen, aktiveres/deaktiveres.

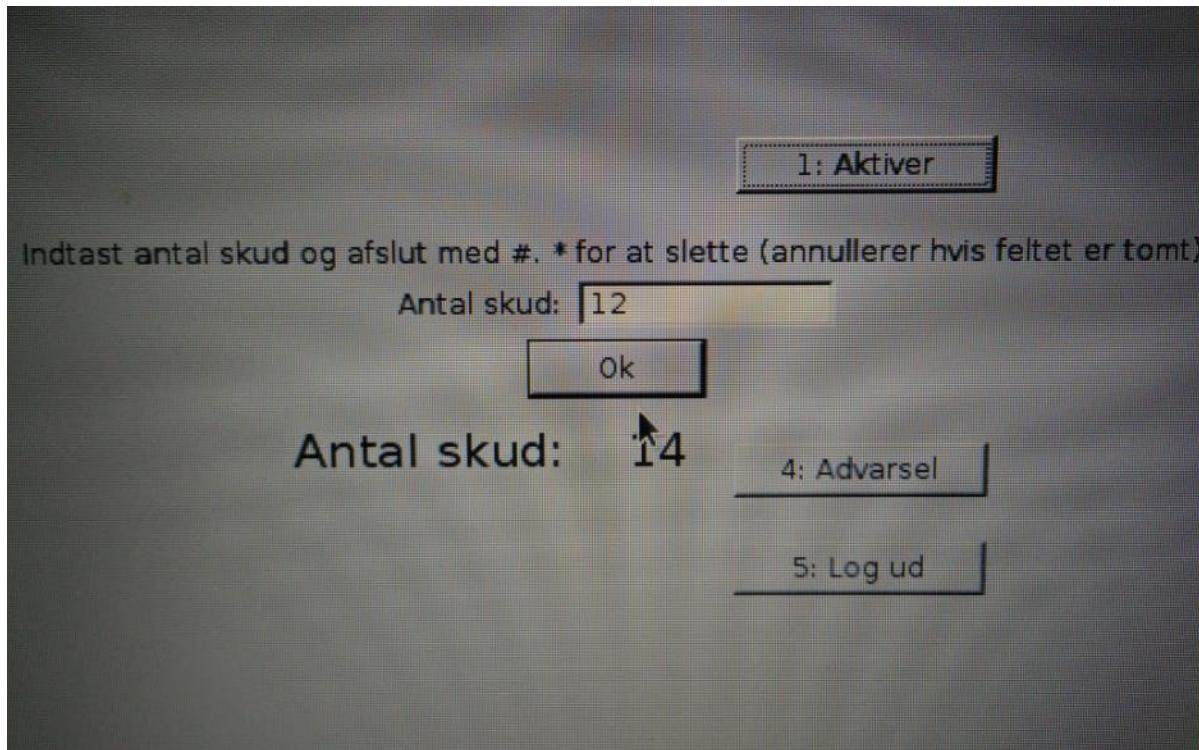


FIGUR 151 - TAST '2' TRYKKET - VISER BESKED MED "ALARM DEAKTIVERET"

På Figur 152 og Figur 153 kan det ses at matrixkeyboardet også virker i genlad menuen, og det kan derfor konkluderes, at **target** har ændret sig som det skal.



FIGUR 152 - TAST '3' TRYKKET - GENLAD MENUEN ÅBNES



FIGUR 153 - INTASTNING MED MATRIXKEYBOADET I GENLAD MENUEN

Som det ses på billederne modtager brugerinterfacet, de korrekte karakterer.

På tabellerne vist nedenfor ses udfaldet af hele integrationstesten ved indtastning på matrixkeyboardet i menerne i brugerinterfacet.

Login menuen

Handling foretaget	Udfald
Indtast kode '1905' og herefter '#'	Fire '*' dukker op, hovedmenuen vises
Tast '*'	Én karakter slettes, hvis der er nogen

TABEL 30 - UDFALD AF INTEGRATIONSTEST I LOGIN MENUEN

Hovedmenuen

Handling foretaget	Udfald
Tryk '1'	Der vises "alarm aktiveret" besked
Tryk '2'	Der vises "alarm deaktiveret" besked
Tryk '3'	Genlad menuen åbnes
Tryk '4'	Der vises "Advarsel trykket"
Tryk '5'	Hoved menuen lukkes, og Login menuen vises

TABEL 31 - UDFALD AF INTEGRATIONSTEST I HOVEDMENUEN

Genlad menuen

Handling foretaget	Udfald
Indtastning af et tal	Tallet vises i Genlad menuen
Indtastning af '*'	Et tal slettes. Hvis ingen tal, luk Genlad menuen
Indtastning af '#'	Vis det indtastede tal, i "Antal Skud" og Genlad menuen lukkes

TABEL 32 - UDFALD AF INTEGRATIONSTEST I GENLAD MENUEN

12.8. Enhedstest af protokol-klassen [JDA]

Dette afsnit indeholder:

- Beskrivelse af implementeringen af test programmet til protokol-klassen
- Resultater af enhedstest

I forbindelse med test af softwaren til protokolklassen, er der udviklet et test miljø. Her testes klassens metoder, **constructString** og **createChecksum**. Med dette værktøj kan det bekræftes eller afkræftes om metoderne gør hvad de er designet til.

Testprogrammet er skrevet i Microsoft Visual Studio Ultimate 2013.

constructString testes ved at give metoden parametrene **CMD_S command** og **option**. Den resulterende **string** skrives da til konsolvinduet.

På samme måde testes funktionen **createChecksum** ved at kalde funktionen og udskrive den karakter som funktionen returnerer.

```
std::cout << "Testing createChecksum.. \n";
std::cout << "Checksum of CMD_LASER ^ option = '0' is " << p.createChecksum(Protocol::CMD_LASER, 0);

std::cout << "Testing constructString with all commands.. \n";
std::cout << "CMD_FULLSTOP = 'F'\n";
std::cout << p.constructString(Protocol::CMD_FULLSTOP, '0') << endl;
std::cout << "CMD_UP = '0'\n";
std::cout << p.constructString(Protocol::CMD_UP, '1') << endl;
```

FIGUR 154 - KODEUDSNIT, PROTOKOLLENS TEST ENVIRONMENT

På Figur 154 ses et kodeudsnit fra **protocolTestEnvironment.cpp**. Programmets output kan ses på Figur 155.

```
Constructing protocol object..
Testing createChecksum..
Checksum of CMD_LASER ^ option = '0' is LTesting constructString with all commands..
CMD_FULLSTOP = 'F'
1F0v
CMD_UP = '0'
101~
```

FIGUR 155 - TEST PROGRAM TIL PROTOCOL-KLASSEN

12.9. Integrationstest af protokollen [JDA]

En integrationstest af protokollen foretaget, for at sikre at klassen fungerer efter hensigten, når den bruges sammen med resten af systemet.

Da det er UARTqueue der modtager kommandoer, der skal videresendes til det Indlejrede Linux System, er det oplagt at teste om protokollens funktionalitet virker, ved at se om UARTQueue modtager den rette data. Det er klassen **UART** der står for at hente dataen fra **UARTQueue**. Det kan da uden større problemer, testes om **UART** klassen sender de korrekte karakterer idet, **UART** klassen i forvejen udskriver den data, som sendes ud til konsollen. Se billede af testen herunder på Figur 156.

```
|STARTING HOME DEFENSE TURRET SOFTWARE!
Creating UARTQueue.
Joystick running!
Constructing keyboard object.
Polling!
Read: 1
Read: 9
Read: 0
Read: 5
Read: #
Setting new target.
Sending string over UART: 1L0|
Read: 4
Sending string over UART: 1A0q
Sending string over UART: 1V1g
Sending string over UART: 1V2d
Sending string over UART: 1V1g
Sending string over UART: 1F0v
Sending string over UART: 1H1y
Sending string over UART: 1H2z
Sending string over UART: 1H1y
Sending string over UART: 1F0v
Sending string over UART: 101~
Sending string over UART: 1F0v
Sending string over UART: 1N1
Sending string over UART: 1F0v
Read: 5
Setting new target.
Sending string over UART: 1K0{
```

FIGUR 156 - UART OUTPUT TIL PSoC4 SAMT LÆSNING AF KNAP TRYKKET PÅ MATRIXKEYBOARDDET

På Figur 156 ses outputtet fra hovedprogrammet.

Først ses koden blive indlæst fra Matrixkeyboardet. Den første string der sendes over UART er kommandoen "1L0|", som tænder for laseren (laseren tændes automatisk efter login). Herefter trykkes på '4' på matrixkeyboardet for at teste afendelse af advarsel, og kommandoen "1A0q" sendes. Herefter bevæges joysticket. Det kan ses på de læste kommandoer, at de ændrer sig korrekt.

Udskriften "Setting new Target" udskrives når der i matrixkeyboardet "passes" et nyt **target** vindue.

12.10. Enhedstest UART og UARTQueue [DT]

For at teste, om UART og UARTQueue klasserne fungerer som de skal, er der foretaget enhedstest på disse. I enhedstesten er der skrevet et test program, som har til opgave at oprette et UARTQueue objekt, oprette et UART objekt og derefter give en besked til UARTQueue. Når beskeden bliver givet til UARTQueue testes der, om denne besked sendes ud fra det Indlejrede Linux Systems serielle Tx pin.

Test programmet kan ses nedenfor.

```
1 #include "../UART/UART.h"
2
3 int main() {
4     UARTQueue queue;
5
6     UART uart("/dev/ttyAMA0", 9600, &queue);
7
8     queue.post("string", 6);
9
10    uart.wait();
11 }
```

FIGUR 157 - UART TESTPROGRAM

På Figur 157 er testprogrammet for UART klassen og UARTQueue klassen. Det er et meget kort program, som blot tester, om UART modtager beskederne fra køen, og sender dem ud over seriel kommunikation.

For at teste om strengen bliver sendt ud, er Tx-benet og GND sat til en USB og dette sat i en PC. På PC'en bruges et program der kan aflæse UART modtagelse fra USB-porten (COM).

Når programmet køres, modtages følgende på PC'en:

```
string
```

FIGUR 158 - MODTAGET DATA PÅ USB COM-PORT

Herved kan det ses, at UART klassen modtager beskeden fra UARTQueue klassen og sender denne ud, så snart beskeden er modtaget.

12.11. Enhedstest af Joystick [DT]

Der testes, om Joystick klassen kan aflæse værdier fra joysticket, triggeren og PIR sensoren.

For at teste dette oprettes et Joystick objekt og der aflæses hvilke værdier aflæsningerne returnerer:

- Joysticket bevæges fra side til side
- Joysticket bevæges op og ned
- Triggeren trykkes ned
- PIR sensoren opfanger bevægelse

Testens resultat er sat op i en tabel, for at gøre det nemt at se, hvilke værdier der er aflæst. Testen er foretaget sammen med SensorsSPI klassens funktionalitet, da Joystick klassen er fuldstændig afhængig af denne klasse, for overhovedet at have noget funktionalitet.

Test af joysticks X-akse

Handling foretaget	Aflæst værdi
Joystick i neutral position	504-510
Joystick bevæget fra neutral til maks højre	Fra ca. 504 til 1023
Joystick bevæget fra neutral til maks venstre	Fra 504 til 0

TABEL 33 - TEST AF AFLÆSNING FOR JOYSTICK X-AKSE

Test af joysticks Y-akse

Handling foretaget	Aflæst værdi
Joystick i neutral position	523
Joystick bevæget fra neutral til maks op	Fra 523 til 1023
Joystick bevæget fra neutral til maks ned	Fra 523 til 0

TABEL 34 - TEST AF AFLÆSNING FOR JOYSTICK Y-AKSE

Test af triggeren

Handling foretaget	Aflæst værdi
Trigger ikke trykket ned	0
Trigger trykket ned	1023

TABEL 35 - TEST AF AFLÆSNING FOR TRIGGER

Test af PIR sensoren

Handling foretaget	Aflæst værdi
Ingen bevægelse foran sensoren	0-100
Bevægelse foran sensoren	750-800

TABEL 36 - TEST AF AFLÆSNING FOR PIR SENSOR

Som det kan ses på ovenstående tests, så kan klassen aflæse alle komponenter fra SPI-modulet (vha. SensorsSPI klassen). Efterfølgende testes handler-funktionernes funktionalitet. Dette gøres ved at få Joystick-klassen til at udskrive en besked, hver gang en kondition er opfyldt i de forskellige handler funktioner.

Test af handlerfunktionerne

For at teste disse funktioner er følgende beskeder sat til at udskrives, i Joystick klassen.

Kondition der er opfyldt	Besked der udskrives
Joystick er i neutral position	Joystick er neutralt
Joystick er i intervallet for hastighed 1 for højre	Hastighed 1 – H
Joystick er i intervallet for hastighed 2 for højre	Hastighed 2 – H
Joystick er i intervallet for hastighed 1 for venstre	Hastighed 1 – V
Joystick er i intervallet for hastighed 2 for venstre	Hastighed 2 – V
Joystick er ført opad	Hastighed 1 - O
Joystick er ført nedad	Hastighed 1 - N

Trigger er ikke trykket ned	Trigger ikke trykket
Trigger er trykket ned	Trigger TRYKKET
PIR sensor ingen bevægelse	
PIR sensor registrerer bevægelse	ALARM!

TABEL 37 - HANDLER KONDITIONER OG UDSKREVNE BESKEDER

Implementeringen af handlermetoderne for joysticket X- og Y-akse, samt trigger betyder, at hver besked kun skal udskrives én gang efter hinanden, for at testen kan markeres som en succes.

For PIR sensoren skal der gå 5 sekunder, før den igen udskriver "ALARM!".

Testens resultater

Her er beskrevet handlingens forløb, skridt for skridt, og det observerede resultat.

Foretaget handling	Observeret besked
Joystick er i neutral position	Joystick er neutral
Joystick føres en smule til højre	Hastighed 1 – H
Joystick føres til maks højre	Hastighed 2 – H
Joystick føres tilbage, så det igen er en smule til højre	Hastighed 1 – H
Joystick føres tilbage til neutral tilstand	Joystick er neutral
Joystick føres en smule til venstre	Hastighed 1 – V
Joystick føres så langt til venstre det kan	Hastighed 2 – V
Joystick slippes, så det hurtigt går til neutral position (og springer hastighed 1 over)	Joystick er neutral
Joystick føres op	Hastighed 1 – O
Joystick føres ned (der køres hurtigt hen over neutral position, men joysticket slippes ikke)	Joystick er neutral Hastighed 1 – N
Triggeren trykkes ned	Trigger TRYKKET
Trigger slippes igen	Trigger ikke trykket
Der viftes foran PIR sensoren	ALARM!

TABEL 38 - RESULTATER AF GENNEMFØRT TEST

Efter testen er overstået kan det konkluderes, at Joystick klassens funktionalitet opfylder kravene og at klassen er fuld funktionsdygtig.

12.12. Integrationstest af Joystick og motorstyring [DT]

Der er foretaget en integrationstest, hvor følgende moduler spiller sammen:

- UART klassen
- UARTQueue klassen
- Joystick klassen
- PSoC4 motorstyring

Testen foregår således:

Joysticket bevæges, hvilket registreres. En kommando sendes derfor til UARTQueue. UART klassen modtager kommandoen fra UARTQueue klassen, og UART klassen sender derfor kommandoen til PSoC4. PSoC4 registrerer kommandoen og styrer motoren i korrekt retning.

Foretaget handling	Ønsket resultat	Resultat OK/fejl
Joystick bevæget lidt til højre (i hastighed 1 interval).	HDT bevæger sig langsomt til højre.	OK
Joystick bevæget maks til højre.	HDT bevæger sig hurtigt til højre.	OK
Joystick neutral.	HDT stopper.	OK
Joystick bevæget lidt til venstre (hastighed 1 interval).	HDT bevæger sig langsomt til venstre.	OK
Joystick bevæget maks til venstre.	HDT bevæger sig hurtigt til venstre.	OK
Joystick bevæget op	HDT bevæger sig op	OK
Joystick bevæget ned	HDT bevæger sig ned	OK
Trigger trykket	HDT affyrer skud	OK
Bevægelse foran PIR sensor	Alarm ved det indlejrede system lyder	OK

TABEL 39 - RESULTATER AF INTEGRATIONSTEST

Som det ses på Tabel 39 er integrationstesten vellykket. For et billede af UART klassens udsendinger, når joysticket bevæges, se Figur 156, i afsnit 12.8.

13. Accepttest [Alle]

13.1. UC1: Log ind

Use Case under test: Log ind.			
Scenarie: Hovedscenarie.			
Prækondition: System er tændt og brugerinterfacet er aktivt. System er logget ud.			
Step	Handling	Forventet observation/ resultat	Vurdering ok/fejl
1	Bruger indtaster adgangskode på tastatur og afslutter med firkant.	System åbnes, brugerinterface vises på skærmen.	OK
2	System låses op.	System låses op og viser menu.	OK
3	Bruger har adgang til System.	Bruger har adgang til System.	OK

TABEL 40 - UC1

13.2. UC1: Ext. 1a. Log ind

Use Case under test: Log ind.			
Scenarie: Ext. 1a: Bruger indtaster forkert kode.			
Prækondition: System er tændt og brugerinterfacet er aktivt. System er logget ud.			
Step	Handling	Forventet observation/ resultat	Vurdering ok/fejl
1	Bruger indtaster forkert adgangskode.	Brugerinterfacet udskriver fejlmeldelse og bliver på log ind skærm.	OK

TABEL 41 - UC1 EXT 1A

13.3. UC2: Log ud

Use Case under test: Log ud.			
Scenarie: Hovedscenarie.			
Prækondition: UC1 er gennemført.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
1	Bruger trykker på "Log ud" knap.	Brugerinterfacet viser log ind skærmen.	OK

TABEL 42 - UC2

13.4. UC3: Aktiver Alarm

Use Case under test: Aktiver Alarm.			
Scenarie: Hovedscenarie.			
Prækondition: UC1 er gennemført. Alarm er deaktiveret.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
1	Bruger trykker på aktiverings-knappen i brugerinterfacet.	Brugers valg registreres.	OK
2	Brugerinterfacet viser en meddelelse om at System er aktiveret.	Der udskrives en besked på brugerinterfacet.	OK

TABEL 43 - UC3

13.5. UC3: Ext. 1a. Aktiver Alarm

Use Case under test: Aktiver Alarm.			
Scenarie: Ext. 1a: Bruger logger ud af brugerinterfacet			
Prækondition: UC1 er gennemført. Alarm er aktiveret.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
1	Bruger logger ud af brugerinterfacet.	Log ind skærmen vises.	OK
2	PIRsensor registrer offer.	Alarm lyder.	OK

TABEL 44 - UC3

13.6. UC4: Deaktiver Alarm

Use Case under test: Deaktiver Alarm.			
Scenarie: Hovedscenarie.			
Prækondition: UC3 er gennemført. Alarm er aktiveret.			
Step	Handling	Forventet observation/ resultat	Vurdering ok/fejl
1	Bruger trykker på deaktivérings-knappen i brugerinterfacet.	Brugers valg registreres.	OK
2	Brugerinterfacet viser en meddelelse om at Alarm er deaktiveret.	Der udskrives en besked på brugerinterfacet.	OK

TABEL 45 - UC4

13.7. UC5: Udløs våben

Use Case under test: Udløs våben			
Scenarie: Hovedscenarie			
Prækondition: UC1 er gennemført.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
1	Bruger trykker på triggeren, for at affyre et skud.	Våbnet affyrer skud.	OK
2	Antal skud tilbage tælles ned.	Antal skud viser én mindre end før.	OK

TABEL 46 - UC5

13.8. UC5: Ext. 1a. Udløs våben

Use Case under test: Udløs våben.			
Scenarie: Ext. 1.a: Våbnet er løbet tør for ammunition.			
Prækondition: UC1 er gennemført.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
1	Bruger trykker på udløserknappen, når magasinet er tomt.	Brugerinterfacet meddeler, at våbnet er løbet tør for ammunition og affyrer ikke skud.	OK

TABEL 47 - UC5 EXT 1.A

13.9. UC6: Genlad

Use Case under test: Genlad.			
Scenarie: Hovedscenarie.			
Prækondition: UC1 er gennemført			
Step	Handling	Forventet observation/ resultat	Vurdering ok/fejl
1	Bruger trykker på ”Genlad”-knappen i brugerinterfacet.	Et vindue popper op med et genladnings-interface.	OK
2	System deaktiverer automatisk.	System deaktiverer.	OK
3	Bruger genlader våbnet med 6 skud.	magasinet fyldes op.	OK
4	Bruger indtaster, at 6 skud er indsat i våbnets magasin.	Brugerinterfacet registrerer, hvor meget ammunition Bruger indtaster.	OK
5	System aktiverer automatisk.	System er aktivt.	OK

TABEL 48 - UC6

13.10. UC6: Ext. 1a. Genlad

Use Case under test: Genlad.			
Scenarie: Ext. 2a: Afbryd genladning.			
Prækondition: System er tændt, alle komponenter tilsluttet korrekt, og System er oplæst. Brugerinterfacet viser genladningsvinduet.			
Step	Handling	Forventet observation/ resultat	Vurdering ok/fejl
1	Bruger trykker på Annuler knappen	Annuler knappen bliver presset og genladningsvinduet lukkes.	OK
2	System aktiverer automatisk.	System er aktiveret.	OK

TABEL 49 - UC6 EXT 2.A

13.11. UC7: Bevæg HDT horisontalt

Use Case under test: UC7: Bevæg HDT horisontalt.			
Scenarie: Hovedscenarie.			
Prækondition: UC1 er gennemført.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
1	Bruger bevæger joysticket til højre.	System starter motor og HDT bevæger sig til højre.	OK
2	Bruger slipper joystick.	System stopper den horisontale motor.	OK
3	Bruger bevæger joystick til venstre.	System starter motor og HDT bevæger sig til venstre.	OK
4	Bruger slipper joystick.	System stopper den horisontale motor.	OK

TABEL 50 - UC7

13.12. UC7: Ext. 1a. Bevæg HDT horisontalt

Use Case under test: UC7: Bevæg HDT horisontalt.			
Scenarie: Ext. 1.a: HDT kan ikke styres mere til venstre.			
Prækondition: UC1 er gennemført.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
1	Bruger bevæger joysticket til venstre, og HDT kører henover dødsswitch.	HDT rammer dødswitch og motor stopper.	OK

TABEL 51 - UC7 EXT 1.A

13.13. UC7: Ext. 1b. Bevæg HDT horisontalt

Use Case under test: UC7: Bevæg HDT horisontalt.			
Scenarie: Ext. 1.b: HDT kan ikke styres mere til højre.			
Prækondition: UC1 er gennemført.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
3	Bruger bevæger joysticket til højre og HDT Kører henover dødsswitch.	HDT rammer dødswitch og motor stopper.	OK

TABEL 52 - UC7 EXT 1.B

13.14. UC8: Bevæg HDT vertikalt

Use Case under test: UC8: Bevæg HDT vertikalt.			
Scenarie: Hovedscenarie.			
Prækondition: UC1 er gennemført.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
1	Bruger bevæger joysticket op.	System starter motor og HDT bevæger sig op.	OK
2	Bruger slipper joystick.	System stopper den vertikale motor.	OK
3	Bruger bevæger joysticket ned.	System starter motor og HDT bevæger sig ned.	OK
4	Bruger slipper joystick.	System stopper den vertikale motor.	OK

TABEL 53 - UC8

13.15. UC8: Ext. 1a. Bevæg HDT vertikalt

Use Case under test: UC8: Bevæg HDT vertikalt.			
Scenarie: Ext. 1.a: HDT kan ikke styres mere op.			
Prækondition: UC1 er gennemført.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
1	Bruger bevæger joysticket op og HDT Kører henover dødsswitch.	HDT rammer dødswitch og motor stopper.	OK

TABEL 54 - UC8 EXT 1.A

13.16. UC8: Ext. 1b. Bevæg HDT vertikalt

Use Case under test: UC8: Bevæg HDT vertikalt.			
Scenarie: Ext. 1.b: HDT kan ikke styres mere ned.			
Prækondition: UC1 er gennemført.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
3	Bruger bevæger joysticket ned og HDT kører henover dødsswitch.	HDT rammer dødswitch og motor stopper.	OK

TABEL 55 - UC8 EXT 1.B

13.17. UC9: Alarm

Use Case under test: Alarm.			
Scenarie: Hovedscenarie.			
Prækondition: UC3 er gennemført. Ingen igangværende alarm.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
1	PIRsensor opfanger bevægelse. Brugerinterfacet viser en meddelelse om at det registreres et mål.	Der udskrives en besked på skærmen.	OK
2	Alarmen lyder.	Alarmen går i gang og afspiller lyd.	OK

TABEL 56 - UC9

13.18. UC10: Advarsel

Use Case under test: Advarsel.			
Scenarie: Hovedscenarie.			
Prækondition: UC1 er gennemført.			
Step	Handling	Forventet observation/resultat	Vurdering ok/fejl
1	Bruger trykker på Advarsels knappen i menuen	Der udskrives en besked på skærmen.	OK
2	Advarselstonen lyder.	Advarselstonen går i gang og afspiller lyd.	OK

TABEL 19 – UC10

14. Bilag

Bilag forefindes på den vedlagte CD-rom.