Bacheloroppgåve:

BO22EB-71

Tyristorstyring med Arduino

Eli Laupsa

30. mai 2022

Kapittel 1

Dokumentkontroll

BO22E-71 Tyristorstyring med Arduino	
Dato :	30.mai.2022
Rapportnummerr :	BO22E-71
Forfattar :	Eli Laupsa
Sider :	86
Studieretning :	Elkraftteknikk
Veiledar :	Eirik Haustveit
Gradering :	Open
Merknadar :	Tillet at oppgåva kan publiserast

Oppdragsgjevar:	Høgskulen på vestlandet
Oppdragsgjevar sin referanse:	Eirik Haustveit
Oppdragsgjevar sin kontaktperson:	Eirik Haustveit

Kapittel 2

Forord

Med denne oppgåva avsluttar eg mitt studie innan Elkraftteknikk ved Høgskulen på Vestlandet. Det har vore lærerikt og kjekt å jobba med bacheloroppgåva. Eg har fått bruk for svært mange av dei emna som har inngått i studiet og det er motiverande, både for vidare studiar og arbeidslivet. Eg har lært svært mykje, både innfor fagfelta og det å jobba med eit slikt prosjekt.

Tusen takk til Eirik Haustveit for svært god rettleiing, og for ei spennande og kjekk oppgåve.

Takk til Pieter Johannes de Zwarte og Jan Vidar Årskaug for krinskort av firkantpulskrinsen.

Kapittel 3

Samandrag

Ved Institutt for datateknologi, elektroteknologi og realfag ved Høgskulen på Vestlandet i Bergen, nyttast det i dag ein halvstyrt tyristoromformar i forbindelse med labarbeid. Programkode og skildring av oppbygging er ikkje tilgjengeleg. Denne oppgåva har gått ut på å byggja ein fullstyrt tyristoromformar, og skriva programkode til denne for Arduino.

Resultatet skal brukast i framtidig labarbeid. Det skal då vera tilgjengeleg programkode for å forstå denne. I tillegg til skildring av korleis ein tyristoromformar er oppbygd, og kva berekningar og omsyn som er nødvendig.

Oppgåva består av ein krins som kan detektera nullgjennomgangane i nettspenninga, og generera eit signal av dette som kan lesast inn på ein Arduino mikrokontrollar. Det skal så skrivast program som gjev ut PWM signal. Det skal vera eit signal som kan trigga tyristorar.

Signalet lyt difor vera synkronisert med nullgjennomgangane, slik at tennvinkelen som er forsinkinga frå nullgjennomgang til trigging vert korrekt. I tillegg skal sjølve tyristorkrinsen og omformaren utformast og byggast. Nødvendige komponentar og berekningar skal skildrast.

Oppgåva er løyst ved ein krins som får inn nettspenninga og gjev ut ein firkantpuls, med positiv eller negativ flanke for kvar nullgjennomgang. Dette signalet vert lest av Arduino Uno, og integrerte timere i mikrokontrollaren genererer forsinking i form av tennvinkel, i tillegg til pulstog ut.

Det er eit LCD-display som viser tennvinkel i grader og prosent, i tilleg til om programmet er i start eller stopp, og kva kontrollmodus som er valt. Kontrollmodus kan veljast av brukar. Enten lokalt med eit potensiometer eller fjernstyring med mobiltelefon. Fjernstyringa er ikkje heilt optimal for bruk på skulen, då det er utfordringar knytt til tilkopling av nettverket. I tillegg er programmet utstyrt med to knappar, start og stopp, som alltid vil virka manuelt.

Det er gjort utrekningar og bygd ein tyristorkrins som er verifisert og verkar. Det er gjort halvbølge likeretting testing med denne, som bekreftar teorien godt. I tillegg er det gjort testar på den gamle omformaren for samanlikning. Det som

manglar er testing av fullbølge likeretting, men krinsar og utrekningar er klare for kopling og testing, og vil verta utført seinare.

Innhald

Figurar

Kapittel 4

Innleiing

4.1 Organisering av rapport

Prosjektet har blitt inndelt i fire delar. Nullgjennomgangsdeteksjon, programkode, tyristorkrins og desse sett saman som tyristoromformar-del. Alle delane er såpass omfattande og viktige at det er blitt oppretta eigne kapittel for kvar av dei, med unntak av tyristoromformar-del, grunna tidsmangel. For kvart av desse vert det gjennomgått planlegging, realisering og testing.

4.2 Oppdragsgjevar

For denne bacheloroppgåva er oppdragsgjevar Høgskulen på Vestlandet,(HVL), Western Norway University of Applied Sciences. Kontaktperson er Eirik Haustveit. HVL er ein av dei største utdanningsinstitusjonene i Noreg med ca. 17000 studentar. HVL vart oppretta 1.januar 2017, då Høgskolen i Bergen, Høgskulen i Sogn og Fjordane og Høgskulen i Stord/Haugesund slo seg saman [41].

Oppgåva vert utført for å kunna nyttast i laboratoriearbeid for elektrofaga på Institutt for datateknologi, elektroteknologi og realfag. Det eksisterer allereie ein tyristoromformar på lab som er i bruk. Det er ein halvstyrt tyristoromformar, medan det i dette prosjektet skal byggjast ein fullstyrt tyristoromformar. Grunnen til at det er ynskjeleg å nytta resultatet av denne oppgåva er for å få tilgang og forståing for software-programmet som vert skrive. I tillegg til ei tilgjengeleg skildring av korleis tyristoromformaren er oppbygd. Resultatet av denne oppgåva skal vera ein fullstyrt tyristoromformar som det kan kontrollerast tennvinkel på.

4.3 Problemstilling

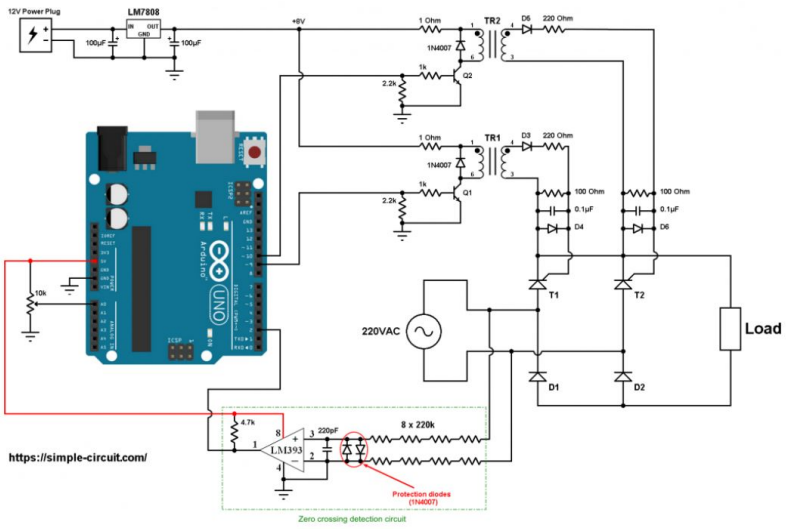
Oppgåva går ut på å byggja sjølve omformaren, i tillegg til å skriva program for kontroll av tyristorar. Det skal utviklast eit grensesnitt for styring av tyristorane med Arduino. Eit potensiometer skal lesast av mikrokontrollaren, som skal

nyttast til å justera tennvinkelen. Produktet skal nyttast i laboppgåver, så det skal vera mogleg å lesa av tennvinkel i grader og effekt.

Produktet skal koplast til nettspenninga. Her $230V_{RMS}$ AC 50Hz, men det skal takast høgd for at den kan brukast i t.d. USA på 60Hz. Nettspenninga skal inn på ein krins som kan detektera nullgjennomgangane på sinusspenninga. Det vil sei at denne krinsen skal gje ut eit signal som kan lesast av ein mikrokontrollar, som synkroniserer programmet med nettspenninga ved hjelp av nullgjennomgangane. Mikrokontrollaren skal så venta ei bestemt tid før den sender ut puls eller pulsar. Pulsane skal sendast inn på gate på tyristorane. Det vil seia trigga tyristorane. Ved å styra tida frå nullgjennomgang til trigging, kan ein styra spenninga ut frå omformaren.

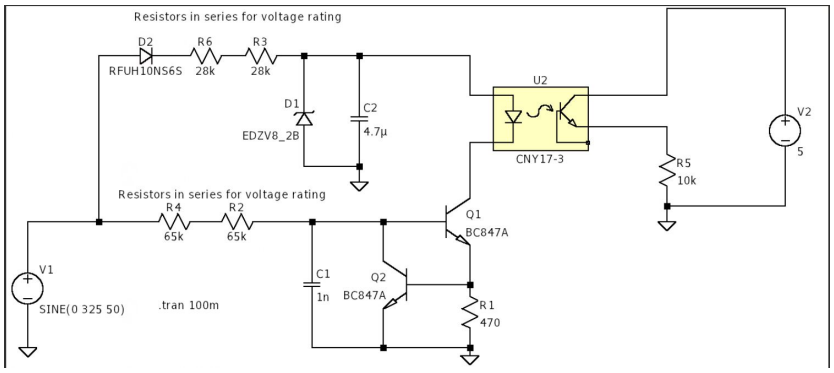
4.4 Hovudide for løysingsforslag

Oppdragsgjevar sin grunnide for kva løysinga skal innehalda er delvis skildra av fylgjande skisse:



Figur 4.1: Fullbølge omformar henta frå simple circuit nettside [9]

Firkantpulskrinsen, merka med grøn stipla linje, er bytta ut med fylgjande krins, som er utforma og designa av Eirik Haustveit, i figur 4.2.



Figur 4.2: Firkantpulskrins designa av Eirik Haustveit

Oppdragsgjevar ynskjer også at det skal nyttast eit LCD-display for å visa vald tennvinkel i grader og effekt. I tillegg skal det vera to trykknappar for start og stopp av programmet. Det er også ynskjeleg med moglegheit for fjernstyring av tennvinkel. Produktet skal monterast oppi ein boks, med display, knappar og potmeter montert utanpå, tilgjengeleg for justering. Det skal borast hol for å kopla til nettspenning inn og kopla til last ut.

Kapittel 5

Kravspesifikasjon

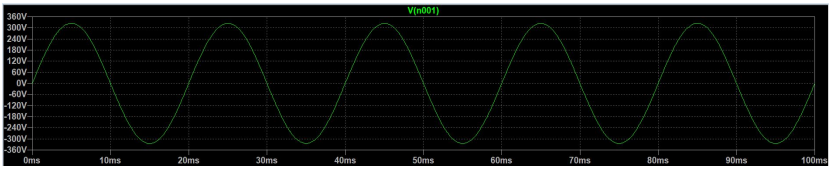
- Omformaren skal vera ein fullstyrt fullbølge likerettar.
- Den skal kunna driva tyristorar av typen SKKH26 og skal kunna drivast på eit spenningsnivå på $230V_{RMS}$.
- Programmet skal programmerast i C/C++. Koden skal gjerast tilgjengeleg på github.
- Pulstoget skal ha ein lågare frekvens enn pulstoget på eksisterande omformar. Det for å gjera det lettare synleg, for å forstå kva som skjer i labsamanheng.
- Programmet skal vera synkronisert med nettspenninga. Altså skal den tilførte sinusspenninga til tyristorane målast for gjenkjenning av nullgjennomgang.
- Det skal vera lokal styring av tennvinkel med potensiometer kor tennvinkel i grader og i prosent, i tillegg til kva modus programmet er i, skal visast på eit LCD-display.
- Det skal vera fjernstyring av tennvinkel med mobiltelefon. Tennvinkel i grader og i prosent skal då visast i applikasjon. Dersom det oppstår feil med kopling til mobil, skal programmet automatisk gå over til lokal kontroll, etter ei viss tid.
- Det skal vera start og stopp knappar for programmet og dermed heile omformaren. Slik at ein manuelt kan stoppa programmet kor tid som helst.

Kapittel 6

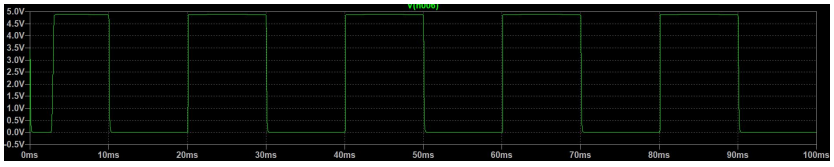
Nullgjennomgangsdeteksjon

6.1 Formål

For at omformaren skal virka, er den avhengig av deteksjon av nullgjennomgangane på nettspenninga. Det er ved nullgjennomgangen programmet vert synkronisert med nettspenninga. Ein elektrisk krins skal få inn nettspenninga og generera ut ein firkantpuls av den. Det vil sei at for kvar nullgjennomgang på nettspenninga, gjev krinsen ut ei endring, negativ eller positiv flanke. Sjå figur 6.1 for nettspenninga som skal inn i krinsen, simulert i LTspice, og figur6.2 for firkantpulssignalet som skal hentast ut frå krinsen og brukast vidare i programkoden.



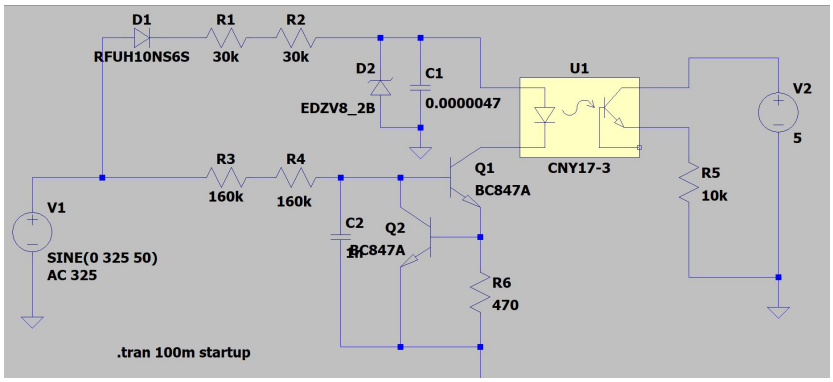
Figur 6.1: Nettspenning simulert i Ltspice, 230V$_{RMS}$, 50Hz



Figur 6.2: Firkantpuls simulert i Ltspice. Spenning over R5 i figur6.3

6.2 Krins

Krinsen er utvikla og designa av Eirik Haustveit.



Figur 6.3: Firkantpulskrins designa i Ltspice

For å oppfylla kravet om galvanisk skille, er det nytta ein optocoupler. Den består av ein LED og ein fototransistor, for å overføra elektriske signal mellom to isolerte krinsar [45]. Det er brukt typen HCPL-817 [23]. Inn på optocoupler på dioden har ein på katode ein straum-spegling krins, denne skal halda utgangsstraumen konstant uavhengig av kva last som er kopla til[25] [11].

6.3 Realisering

Komponentliste for krinsen:

Motstandar

Diode: RL14007-T[29]. Zenerdiode: BZX55[50]

Kondensator: WCAP 4.7μF [44]

Optocoupler: HCPL-817[23]

Transistorar: BC547[28]

Quadruple Input Positive-NAND Schmitt trigger: SN74S132N[32]

Det er i tillegg til krinsen i figur 6.3 sett inn ein Schmitt-trigger mellom utgangssignalet og inngangen på Arduino. Dette er ein logisk inngang som brukar hysteresen for å gje negativ eller positiv flanke. Altså har den eit terskelnivå kor signalet vert høgt eller lågt. Primært skal den modulera firkant eller sinussignal med støy, til fine firkantbølgjer [4]. Dette ser ein tydeleg at verkar ved å samanlikna figurA.4, kor den gule grafen er utan triggeren og figurA.5 som er same puls med trigger.

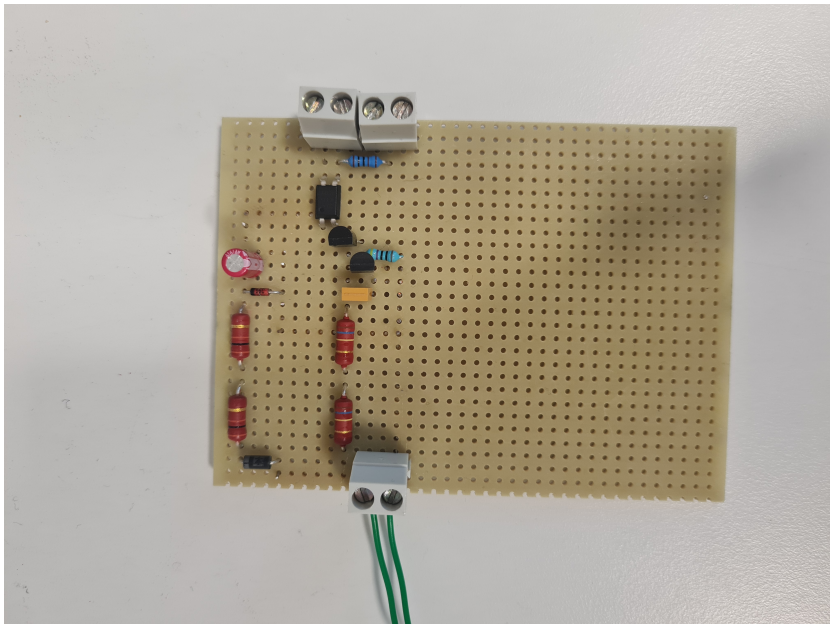
Ein ser konsekvensen ved å ikkje bruka triggeren. Utfordringa det fører til vidare, er at mikrokontrollaren ikkje klarar å detektera alle flankane. Eller detekterer dei for tidleg og for seint. Dermed kan det føra til feil i tidspunktet for pulstogsignal og tennvinkelen.

6.4 Testing

For alle testar utført med tilført spenning mindre enn 230V, er det brukt to kraftforsyningar. Ei for tilført spenning, $10V_{RMS}$ og $50V_{RMS}$, og ei tilkopla zenerdioden, då både 10V og 50V vert for lite i denne krinsen til at den når ledespenninga. Zenerdioden har ein zener-spenning på 8.2V [24]. Så kraftforsyninga vart sett til ca. 7V. I tillegg er det tilkopla 5V og jord frå Arduino på lågspenningssida av krinsen.

Krinsen vart kopla opp på koplingsbrett for å sikra at krinsen virka, før lodding. Bilete frå oppkoplinga ligg i vedlegg A.1.

Figur6.4 og figur6.5 viser dei to korta som er brukt under testing. Det lodda kortet er eige arbeid og det etsa krinskortet er laga av Pieter Johannes de Zwarte, og Jan Vidar Årskaug, som brukte same krins i prosjektarbeid. Dei har brukt Kicad for å teikna krinsen og deretter er krinskortet etsa. Planen var å gjera dette med alle krinsane i dette prosjektet også, men grunna dårleg tid vart det ikkje prioritert.



Figur 6.4: Firkantpulskrins lodda på koplingsbrett

Figur 6.6 viser utklipp frå oscilloskop under testing på 50V. Figur 6.7 viser forsinkinga mellom faktisk nullgjennomgang og registrert nullgjennomgang i firkantpulskrinsen. Forsinkinga er på 128μs som tilsvarer 2.3° på tennvinkelen.

Det etsa krinskortet er også testa på 50V. Sjå oppkopling i figur 6.9 og utklipp frå oscilloskop i figur6.10. Forsinkinga mellom nullgjennomgangen frå nettspenninga og den registrerte nullgjennomgangen ser ein i figur6.11, den er på 116μs som tilsvarer 2.088°. Krinskortet er og testa på $230V_{RMS}$, altså tilkopla nettspenning. Då vert forsinkinga endå mindre. På 230V er den på 76μs, altså 1.368° 6.12. Forsinkinga på 50V, samanlikna med 230V har ein differanse på 116-76= 40μs,

som tilsvarer 0.72°

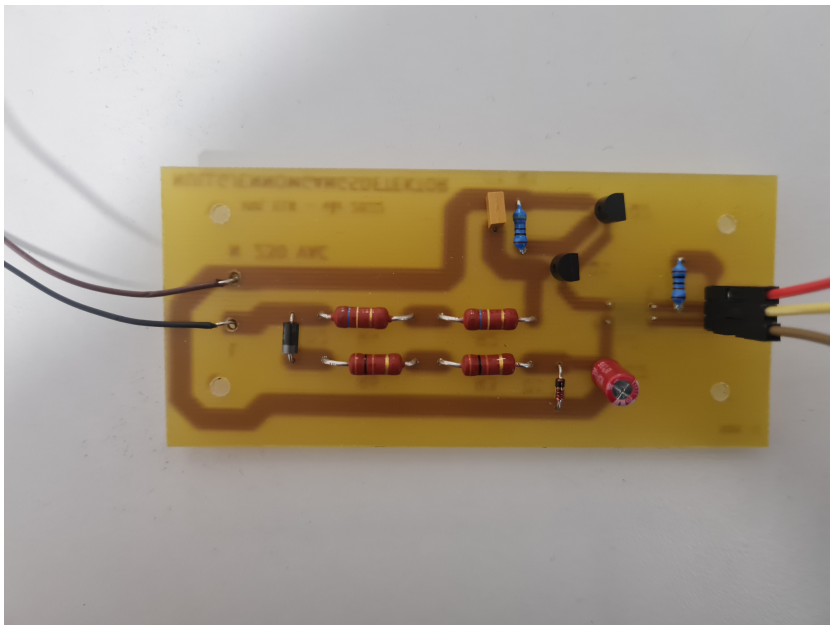
Utrekningar frå forsinking i μs til grader er gjort greie for i likninga 6.1. Der svaret er 0.018μs per grad.

$$\frac{180}{10000} = 0.018 \tag{6.1}$$

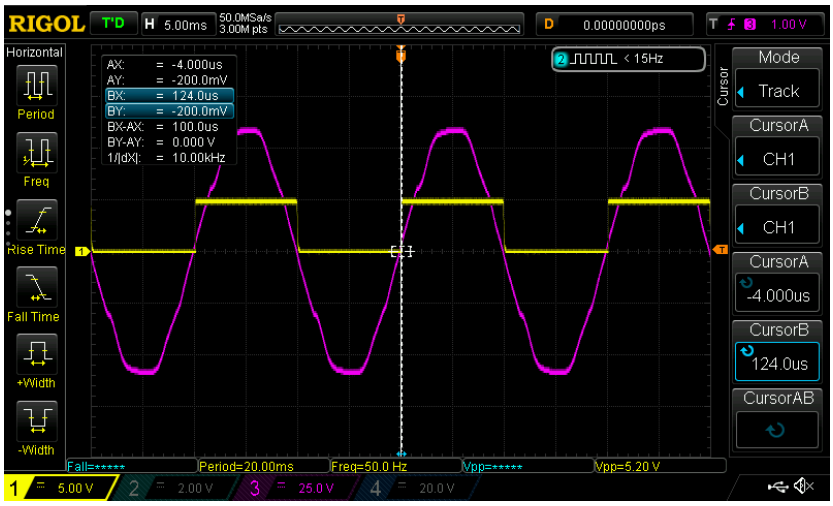
Det er også testa kor godt mikrokontrollaren registrerte signalet. Testen er gjort ved å senda signalet inn på pinne 2 og skapa eit avbrot som sat utgangar høge eller låge 6.1. Det takla mikrokontrollaren fint. I figur 6.8 ser ein den turkise grafen, kor programmet set utgangen høg og låg for kvar nullgjennomgang på sinuskurva. Den gule grafen er firkantpulssignalet frå krinsen, og Arduino sitt signal ligg fint oppå denne firkantpulsen. Altså er den korrekte firkantpulsen lest av mikrokontrollaren, og dette signalet er godkjent til å brukast vidare.

Listing 6.1: Test av firkantpuls inn på Arduino Uno

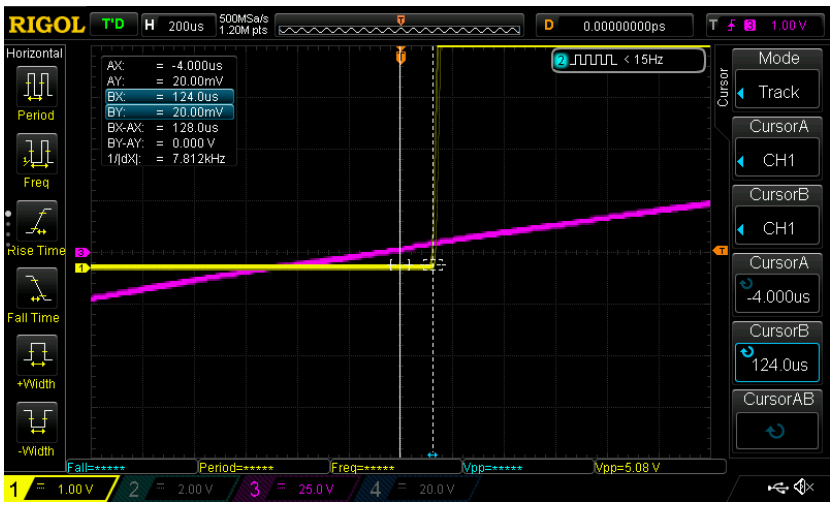
```
1  #include <Arduino.h>
2  void interruptSetup();
3  void ZC();
4
5  void setup() {
6      interruptSetup();
7      pinMode(10, OUTPUT);
8  }
9
10 void interruptSetup(){
11     attachInterrupt(digitalPinToInterrupt(2), ZC, CHANGE);
```



Figur 6.5: Firkantpulskrins på etsa krinskort

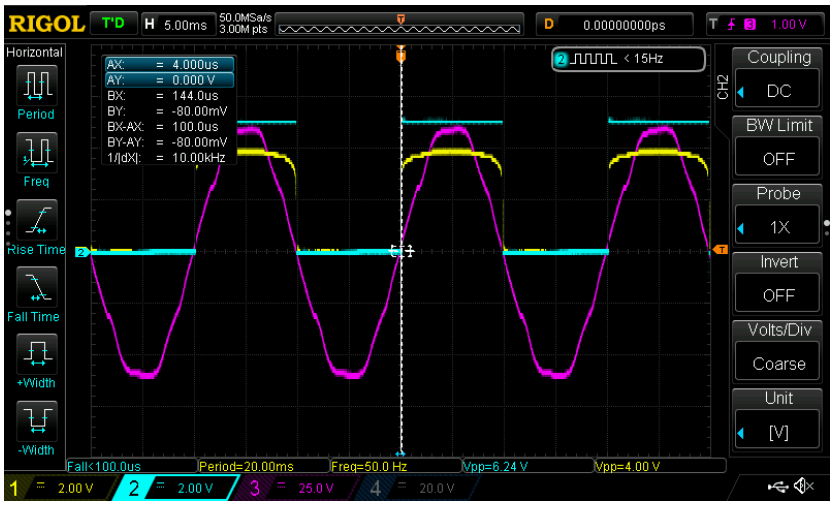


Figur 6.6: Utklipp frå Oscilloskop - firkantpulssignalet og innspenning



Figur 6.7: Forsinkinga her på lodda kort

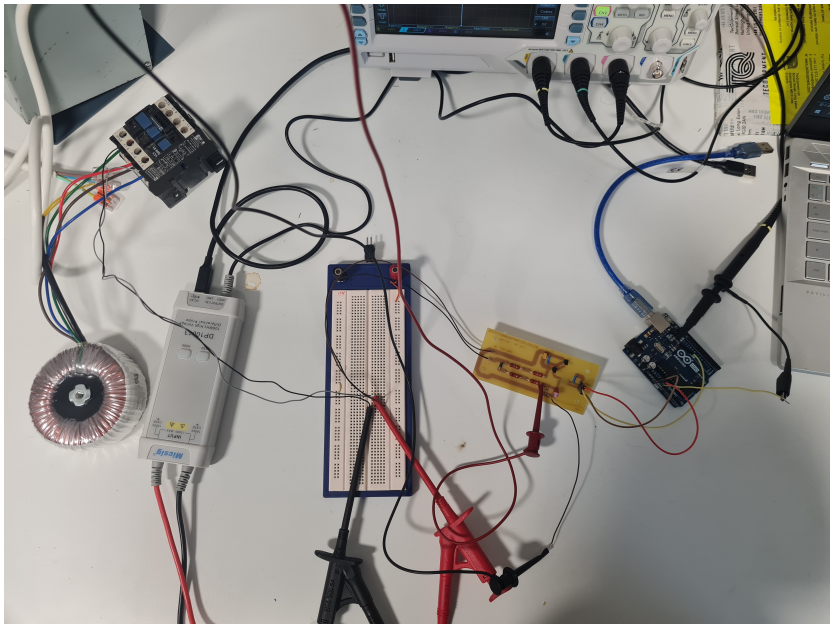
```
12  }
13
14  void ZC(){
15      digitalWrite(10, !digitalRead(10));
16  }
17
18  void loop() {
19      // put your main code here, to run repeatedly:
20  }
```



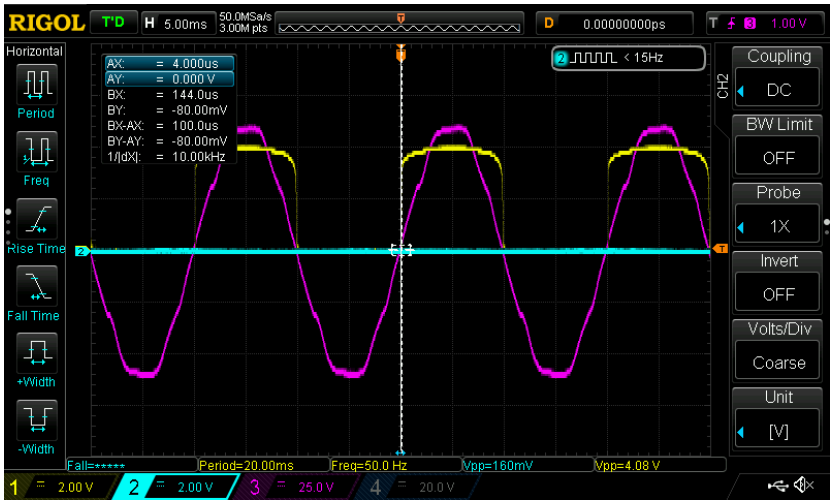
Figur 6.8: Test med Arduino

6.5 Resultat

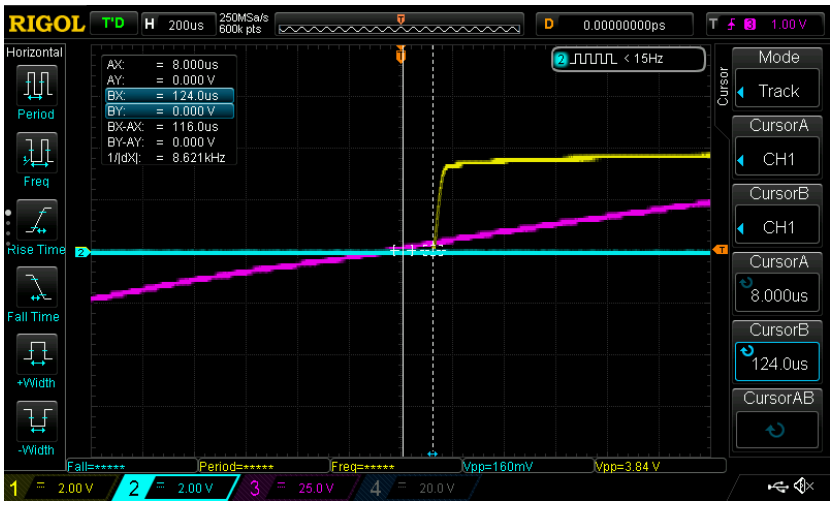
Etter testing er resultatet at krinskortet skal nyttast vidare. Det gav ein overraskande fin og rein firkantpuls som mikrokontrollaren fint klarar å lesa. Forsinkinga mellom nullgjennomgang og flankane er svært liten, sannsynlegvis ubetydelg. Som figur 6.12 viser, er forsinkinga endå mindre ved høgare spenning, som forventa. Det vert ettersjekka og testa på ferdig produkt, og dersom nødvendig vert det kalibrert i programkoden. Uansett vil det ikkje verta noko problem.



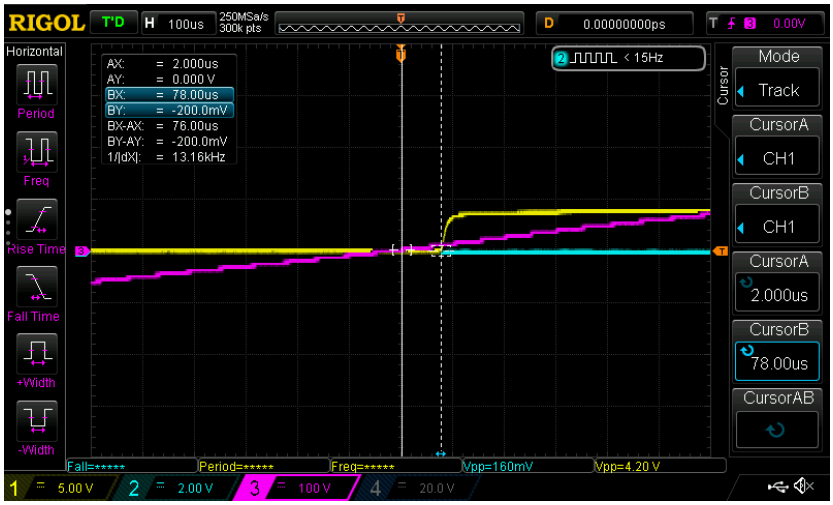
Figur 6.9: Oppsett med krinskort



Figur 6.10: Frå Oscilloskop under testing av krinskort på 50V



Figur 6.11: Forsinking her på krinskort



Figur 6.12: Forsinking på krinskort med 230V

Kapittel 7

Programkode

Programmet som skal skrivast skal få inn firkantpulssignal som er synkronisert med nettspenninga og senda ut pulstog som PWMsignal(pulse widt modulation). Det skal kunna styrast lokalt ved hjelp av eit potensiometer og fjernstyrast ved hjelp av mobiltelefon og applikasjon. For lokal kontroll skal data som tennvinkel i grader og prosent, og modus visast på eit LCD-display. For fjernstyring skal same data visast i mobilapplikasjon. Programmet skal skrivast i C/C++ og skal gjerast tilgjengeleg på github.

7.1 Komponentar og program

Mikrokontrollarar

Det er brukt Arduino Uno og ESP32 mikrokontrollarar. Hovudprogrammet er skrive til Arduino Uno, medan utviding som gjeld fjernstyring er skrive til ESP32.

Arduino Uno er ein mikrokontrollar, utvikla av Arduino.c.c. Den er bygd på ATmega328P microchip. Uno er enkel i bruk, fleksibel og rimeleg i pris. Det er det mest robuste brettet innfor Arduino familien. I tillegg det mest dokumenterte og mest brukte [19]. Difor eksisterar det store mengder med ressursar for programmering og design av krinsar [14].

ESP32 er ein mikrokontrollar med kombinert WiFi og Bluetooth, og eit fleksibelt bruksområde. WiFi-kortet gjer ESP32 godt eigna t.d. til IoT prosjekt. ESP32 fins i mange versjonar, i dette prosjektet nyttast ESP32DevKitC V4 [15]. Det er totalt 38 pinnar, kor 34 kan brukast som GPIO(General purpose input/output). Desse er generelle pinnar utan noko forhåndsdefinerte bruksområde, og brukaren bestemmer funskjonen. Sjølv om GPIO pinnar har fleksibelt bruksområde, har ulike pinnar nokre innebygde funksjonar som kan gje fordelar for nokre oppgåver. For I2C, som er bruksområdet for ESP32 i dette prosjektet, er IO21(SDA) og IO22(SCL) standard, men alle GPIO kan konfigurerast for I2C. Dette er dei einaste pinnane i prosjektet som vert brukt, då ESP32 er

tatt i bruk for å kommunisera med app. Den er difor sett opp som master, med Arduino Uno som slave [10].

Koplinga mellom Arduino Uno og ESP32 er på IO21(SDA) på ESP32 til SDA på Uno, og IO22(SCL) til SCL på Uno. Dette saman med Wire.h biblioteket av Arduino, er det som trengs for I2C koplinga.

7.2 Firkantpulssignal

Som sagt lyt programmet vera synkronisert med nettspenninga for å detektera nullgjennomgangane. For så å kunna skapa forsinkinga frå nullgjennomgang til pulstog, som då er tennvinkelen. Signalet er firkantpulsen, på 50Hz eller 60Hz. Det vil seia signalet gjev ei endring, positiv eller negativ flanke, 100 gonger i sekundet for 50Hz og 120 gonger i sekundet for 60Hz. Programmet skal registrera endringane og så utføra handlingar synkronisert med desse.

For å løysa dette er det vald å bruka eksternt avbrot. Det vil sei at prosessoren registrerer nullgjennomgangane umiddelbart, lagrar tilstanden, køyrer ein ISR funksjon, og går direkte tilbake til der programmet var før, utan å returnera noko [36].

Ein ISR (Interrupt Service Routine) stoppar det som skjer i programmet eller mikrokontrollaren, slik at noko anna kan skje. Altså vil hovudprogrammet/loop køyra, når ein ISR vert kalla vil hovudprogrammet stoppa og ISRen utfører det den skal. Etterpå fortset programmet der det var før ISRen vart kalla. Det fins hardware avbrot, som oppstår ved eksternt avbrot på pinne, og software avbrot som oppstår gjennom instruksjonar i programmet [18].

Avbrota bør vera korte, og det er kun ein ISR som kan køyra om gongen [21]. Dersom det skulle vore skrive eit program som skal registrera kvar nullgjennomgangspuls, hadde det vorte utfordrande å gjera andre ting i programmet fordi det heile tida måtte ha sjekka om det er ei endring i signalet. For nettopp slike problemstillingar er eksterne avbrot ei god løysing [37].

Innstilling av avbrot i kode 7.1.

Listing 7.1: Innstilling av avbrot i programmet

```
1 void interruptSetup()
2 {
3 EICRA = (1 << ISC01) | (0 << ISC00);
4 EIMSK = (1 << INT0);
5 }
```

Frå oversikt over pinnar til Arduino Uno i vedlegg D.1, ser ein at pinne 2 er kopla til INT0. Eksterne avbrot kan verta registrerte på INT0 og INT1 pinnane. Frå External Interrupt Mask Register (EIMSK) figur 7.1, ser ein at ved å setja INT0 til 1, så vert eksternt avbrot aktivert på pinne 2. Det vil sei at pinne 2 er aktivert for å registrera avbrot på det signalet som kjem inn her, altså firkantpulssignalet.

I External Interrupt Control Register A (EICRA) figur 7.2, veljast det kva del av signalet eller når avbrotet aktiverast. Frå tabell 12-2 i datablad, sjå figur 7.3,

er moglegheitene skildra. Det kan aktiverast avbrot når signalet er høgt, lågt, ved endring, har positiv flanke eller negativ flanke.

Frå kodedelen 7.1, er ISC01 sett til 1 og ISC00 sett til 0, altså vil avbrotet aktiverast for kvar negative flanke på firkantpulsen, "The falling edge of INT0 generates an interrupt request" [20][s.54]. Ein korresponderande avbrotsfunksjon frå eksternt avbrot på pinne 2, altså ein ISR-funksjon som er knytt til denne pinnen, vert køyrt frå avbrotsvektoren INT0.

Det tyder at med desse innstillingane i EICRA registeret, registrerer pinne 2 dei negative flankane i firkantpulssignalet. EIMSK registeret sørger for at det vert kalla ein ISR-funksjon kvar gong det skjer.

12.2.2 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 7.1: fig:Register EIMSK frå datablad[20]s.55

12.2.1 EICRA – External Interrupt Control Register A

The external interrupt control register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 7.2: fig:Register EICRA frå datablad[20]s.54

Table 12-2. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Figur 7.3: fig:Tabell Interrupt sense control frå datablad[20]s.54

Ei utfordring gjennom store delar av prosjektet var at pulstoga var ustabile, og enten var på heile tida, eller ikkje skrudde seg på når dei skulle. Det er testa at det skal aktiverast avbrot for kvar endring, altså både positiv og negativ flanke. For kvar flanke skulle eit av pulstoga skruast av, og det andre skruast på etter at forsinkingstida var gått. Problemet med dette er at den negative flanka til nullgjennomgangssignalet ikkje er optimal for å stoppa det eine pulstoget. Tidspunktet for denne flanka kan variera, avhengig av toleransen til komponentar eller anna støy frå nullgjennomgangskrinsen. Med denne innstillinga er pulsen mest ustabil med tennvinkel i nærleiken av 180 grader. Det som då skjer, er at den negative flanka, som skal skru av pulstoget, kjem før ISRen som skal skru på pulstoget. Konsekvensen er at pulstoget er på heile tida når tennvinkelen nærmar seg 180°.

Det ein veit er at den negative flanka alltid vil koma 10 millisekund etter positiv flanke, eller motsett. Så dersom ein berre nyttar den eine av flankane til å aktivera avbrot, vil den andre flanka uansett koma ein halv periode etter, altså 10ms for 50Hz og 8.33ms for 60Hz.

Difor er innstillinga, som vist i 7.1 vald. Mikrokontrollaren vil alltid registrera nullgjennomgangen på negativ flanke og aktivera avbrot, uavhengig av kva anna som skjer i programmet. Avbrotet kallar ISR(INT0_vect) som er kopla til same pinne som avbrotet skjer på, pinne 2. I denne funksjonen skjer fylgjande 7.2 :

Listing 7.2: ISR som køyrer for kvar neagtive flanke på firkantpulsen

```
1
2  ISR(INT0_vect){
3  zc_watchdog_counter = 0;
4      compA_counter = 0;
5      if (start){
6          period_count = TCNT1;
7          TCNT1 = 0;
8          OCR1A = get_comp_alpha();
9          OCR1B = get_comp_alpha() + get_comp_180();
10
11             #if defined(SQUARE_WAVE) && (SQUARE_WAVE == 0)
12                 T34pulseoff();
13             #else
14                 PORTD &= ~(1 << 3);
15             #endif
16             }
17     else {
18     }
```

Denne ISRen vil som sagt køyra kvar gong pinne 2 får inn negativ flanke frå firkantpulssignalet. Då sjekkar programmet om det er i start modus eller ikkje. I startmodus så nullstillast timer1 ved TCNT1 = 0, altså startar timeren telja frå null. I tillegg hentast det riktig alfaverdi til OCR1A og OCR1B. Her ser ein at ISRen som høyrer til OCR1A køyrer etter at tennvinkel-tida er gått, alpha. Medan ISRen til OCR1B vil køyra etter den same tida, alpha, pluss ein halv periode, 180, er gått. Utrekningar for get_comp_alpha() og get_comp_180() er gjort greie for i kapittel 7.6.

Det eine pulstoget vert også skrudd av ved den negative flanka, ved hjelp av funksjonen T34pulseoff(). Altså for kvar negative flanke i startmodus vert timer1 nullstilt, og det eine pulstoget stoppa. Dersom programmet ikkje er i startmodus, skjer ingenting og programmet sjekkar på ny om det er i startmodus.

I tillegg er det lagt til ein zc_watchdog som skal sjekka om det manglar firkantpulssignal. Denne ligg i loop som zc_watchdog++ og vert berre nullstilt ved negativ flanke. Dersom verdien vert større enn 15 får ein feilmelding på at det manglar firkantpulssignal. Dersom verdien vert 15, betyr det at loop har køyrt 15 gongar utan at ISR(INT0_vect) har køyrt, som tyder at ein ikkje får detektert negativ flanke. CompA_counter er ein teljar som skal brukast til å sjekka om ein er på negativ eller positiv flanke i timer1_compA_vector.

7.3 Knappar

For å kontrollera omformaren og programmet skal det vera tilkopla to knappar eller brytarar. Det er brukt type ZBE-101 og ZBE-102. Det er raud for stoppknapp, og grøn for startknapp. Desse skal overstyra alt anna som skjer i programmet, slik at dersom ein trykker på stoppknappen, stoppar mikrokontrollaren å senda ut signal til tyristorane og programmet er i stoppmodus. Knappane vil også overstyra alle kommandoar i applikasjonen når programmet er i fjernstyrings modus, slik at det alltid kan stoppast manuelt.

Knappane er programmert slik at for start knapp vil det ikkje registrera start før knappen er trykt og sluppet. Medan for stoppknappen vil den registrera stopp med det same knappen vert trykt inn. Det er for at stopp skal skje fortast mogleg.

I programmet nyttast analoge inngangar for å lesa av knappane. Dei er kopla i pulldown, med motstand til jord og 5V til knapp og A0. Slik at dersom ein trykkjer ned knappen les inngangane det som 1 og ellers som 0. For pulldown kopling, sjå vedlegg D.2.

Koden sjekkar kvart 100 millisekund om knappar er trykt ned. Dette er fort nok til at det oppfattast som kontinuerleg. Knappane vert sjekka i loop 7.3.

Listing 7.3: Kode for å sjekka om knappar er trykt

```
1  if ((currentTimeStamp_ms - prevTimeStamp_ms) >= interval_ms)
       {
2      SamplePushbuttons();
3      if ((button_state.changed & (1<<0)) != 0){
4          if ((button_state.pressed & (1<<0)) != 0){
5          }
6          else {
7
8          start= 1;
9          timer1on();
10         }
11     }
12
13     if ((button_state.changed & (1<<1)) != 0){
14         if ((button_state.pressed & (1<<1)) != 0){
15             start = 0;
16             timer1off();
17             T12pulseoff();
18             T34pulseoff();
19         }
20     }
21 }
```

Millis() funksjonen vert brukt for å skapa eit tidsintervall for å sjekka om knappar er trykt inn. Ved å velja verdi for interval_ms kan ein kontrollera kor ofte det skal sjekkast om knappane er trykt inn. Kvar gong den valde tida er gått, køyrer SamplePushbuttons() funksjonen 7.4, som er forklart under. Dersom statknappen er trykt inn vert start = 1. Start er flagget som vert sjekka

for å bestemma om programmet skal køyra. I tillegg vert timer1on() funksjonen kalla. Altså startar timer1 å telja frå null.

Dersom stoppknappen er trykt, vert start = 0, og programmet er ikkje i start modus lenger. Timer1off() køyrer og skrur av funksjonar som styrer forsinkinga til tennvinkelen. I tillegg vert pulstoga skrudd av. Det er strengt tatt ikkje nødvendig, men vert ei ekstra sikring for at programmet har stoppa senda ut pulsar.

Listing 7.4: Kode for debounce av knappar. Laga av Eirik Haustveit

```
1  struct {
2  uint8_t pressed;
3  uint8_t previous;
4  uint8_t changed;
5  } button_state;
6
7  void SamplePushbuttons (){
8
9      uint8_t currentState =  (!digitalRead(A4) << 1)|
           digitalRead(A3) | (digitalRead(A5) << 2);
10
11     button_state.changed = button_state.pressed;
12
13     button_state.pressed |= (button_state.previous &
           currentState);
14     button_state.pressed &= (button_state.previous |
           currentState);
15
16     button_state.changed ^= button_state.pressed;
17     button_state.previous = currentState;
18
19     }
```

Her er avlesing av dei analoge inngangane til knappane lagt inn i ei 8bit maske currentState. Grunnen til at avlesing av A4 er invertert er fordi denne knappen er av typen NC($NormallyClosed$), medan startknappen er NO($NormallyOpen$). Som tyder at stoppknapp les 1 for ikkje trykt og 0 for trykt.

Debouncefunksjonen er for å vera sikker på at knappen faktisk har vorte trykt, og at det vert registrert som eit trykk.

7.4 Potensiometer

For å bestemma tennvinkelen skal det brukast eit potensiometer. Det er typen B10K som er brukt her. Dette vert kopla til analog inngang, A0, 5V og jord. Sjå figur i D.2. Ved å nytta analogRead av denne verdien les det av analog verdi mellom 0-1023. Dette skal tilsvara 0-180 grader i programmet. Dette vert gjort ved fylgjande rekning 7.1, 7.2 :

$$alpha_{degrees} = \frac{potmeterverdi \cdot (180)}{1023} \tag{7.1}$$

$$alpha_{percentage} = 100 - \frac{alpha_{degrees} \cdot (100)}{180} \tag{7.2}$$

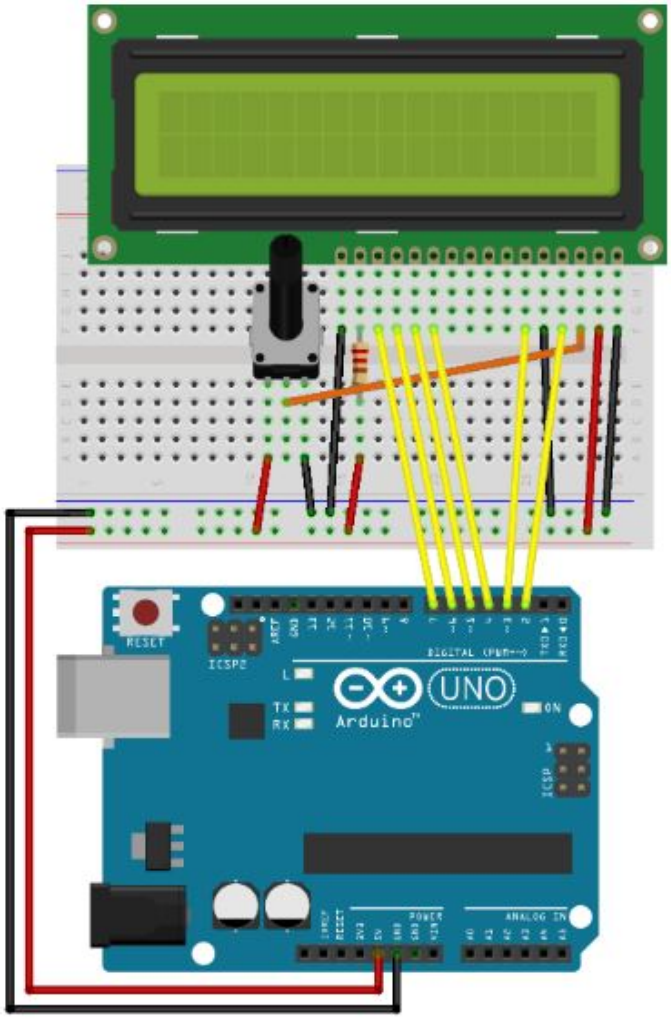
Utrekningane i programmet ligg i vedlegg E.1. Det tyder at alfa i koden alltid er mellom 0 og 180. På denne måten blir det ikkje meir nøyaktig enn ein grad, då desimalar ikkje vil bli tatt med. Dette er gjort vurdering for, og ein grad nøyaktighet for bruk i labsamanheng er godt nok.

Potmeteret vert lest av i loop, kvart 100ms. Altså vil det for oss oppfattast som at det kan endrast kontinuerleg. Potmeteret vil kunna justerast og verdien vil visast på LCD displayet uavhengig av start flagget. Slik at ein kan sjå kva tennvinkel som er vald før ein startar programmet.

7.5 LCD

Verdiane som skal lesast inn og visast på display er tennvinkel i grader og prosent, start og stopp, og kva modus programmet er i når det gjeld lokal styring eller fjernstyring. Det er også mogleg å visa kva spenning omformaren gjev ut, når spenninga inn og komponentar er kjende. Det er brukt LiquidCrystal biblioteket av F Malpartida. Biblioteket tillet kommunikasjon med alfanumeriske flytande krystallskjermar, altså LCD-display[38]. Displayet er kopla til pin 4-9 på Arduino, og er ellers identisk til bilete under7.4, med unntak av at det er vald å ikkje bruka potmeter for å justera bakgrunnslysstyrken på displayet, så denne er kopla tl jord i staden for. Koplinga er lodda til eit loddekort som skal settast ned i Arduino Uno. Det er laga ein funksjon for skriving til lcd display, sjå vedlegg E.2.

I funksjonen er det brukt kommandoane som ligg i LiquidCrystal biblioteket. Det er også laga eit eige alfa symbolE.3. Dette alfa symbolet er plassert meir symmetrisk i displayet enn det som ligg førehandslaga. Funksjonen vert kalla i loop, kvart 100ms, altså vert displayet oppdatert kontinuerlig for oss.



Figur 7.4: fig:Figur er henta frå nettside, ELE102 Microcontroller course[33]

7.6 Alfa verdiar

Kode for å rekna ut kva alfaverdiar som skal brukast visast i kodedel 7.5. Under settings i programmet, sjå vedlegg F.1, kan ein velja kva frekvens på nettspenninga omformaren er tilkopla. Deretter reknast det ut kva $\alpha = 180°$ er for vald frekvens 7.3, omrekna til OCRxy verdiar. Likning 7.4 reknar ut kva 1° for vald frekvens er, og likning 7.5 reknar ut OCRxy verdi for den faktiske alfaverdien. Dette ligg i kodedel 7.5. Her er også lagt inn grense på at alfa ikkje vert 0 eller over 178. Dette er for å unngå forsinkingar eller feil på timinga av pulstog.

$$comp_{180} = \frac{16MHz}{P \cdot C \cdot f} = \frac{16MHz}{8 \cdot 2 \cdot f} \tag{7.3}$$

$$x = \frac{180 \cdot 1000000}{comp_{180}} \tag{7.4}$$

$$comp_{alpha} = \frac{alpha \cdot 1000000}{x} \tag{7.5}$$

Listing 7.5: Kode for å bestemma alfa

```
1         void timer1setalpha(uint16_t alpha, uint8_t
              grid_freq){
2
3             comp_180 = (16000000/(8*2*grid_freq));
4             uint32_t x = ((180*1000000)/comp_180);
5
6             if (alpha > 178){
7                 alpha = 178;
8             }
9
10            comp_alpha = ((alpha*1000000)/x);
11
12            if(0 == comp_alpha){
13                comp_alpha = 1;
14            }
15        }
```

7.7 Pulstog

For å trigga tyristorane skal eit pulstog sendast inn på gate. Det kunne vore nytta eit enkelt signal også, eller ein kontinuerleg gate-straum. Problemet med eit enkelt signal er at det kan henda ein ikkje når I_L straumen, som er den straumverdien tyristoren treng på anode for å vera leiande. Eller det kan vera anna støy i krinsen som hindrar tyristoren frå å registrera det eine signalet[49]. Ein kontinuerleg straum på gate er ofte ei dyr løysing og gjev store effekttap i tyristoren [49][s.95]. Difor er det vald å nytta eit pulstog kor ein kan justera arbeidssyklus og frekvens.

Eit av krava for pulstoga var at det skal vera låg nok frekvens til at det er mogleg å sjå signalet på oscilloskop, og forstå kva som skjer. Det pulstoget som

vert brukt på eksisterande tyristoromformar på lab har ein frekvens på 25kHz, som ein såg under testing av denne, sjå figur i vedlegg C.1.

For å generera pulstoga og forsinkinga som er tennvinkelen, vert det brukt Arduino Uno sine integrerte timere. Desse ligg i kontrollaren Atmel AVR Atmega328. Uno har tre timere, timer0 og timer2 som er 8bit, og timer1 som er 16bit . Det er Atmega328 si klokke som er grunnlaget, og timerane har register som kan programmerast rundt denne [3]. Klokkefrekvensen til Arduinoen er 16MHz. Nokre av Arduino sine ferdigdefinerte funksjonar nyttar timer0, t.d. funksjonane delay(), millis() og micros() [35]. Om registeret til timer0 vert endra, vil det kunna påverka desse funksjonane. Både delay og millis er i bruk i programmet, så det er berre timer1 og timer2 sine register det vert skrive til.

7.7.1 Timere

Alle register for timerane og deira forklaring ligg i databladet for Atmega328P[20]. For å generalisera namna på timerane vil det vidare brukast X for tala 0-2 etter kva timer det gjeld, altså timer0, timer1 eller timer2. Y står for bokstaven A eller B. Det er same forkortingar for alle timerane. T.d. OCRxy for timer1 vert OCR1y. For utgang som høyrer til pinne 11 vert det OCR1A, då A er knytt til pinne 11D.1.

I mangel på god norsk oversetting lyt ordet bitane vidare lesast som fleire bit i bestemt form.

Forkortingar:

TCNTx: Timer/Counter x. Dette er verdien til timeren, altså det den har talt til, for 8bit timere mellom 0-255, og 16bit mellom 0-65535. Dersom TCNTx = 0, tyder det at timeren vert nullstilt, og startar telja frå null.

Det å skriva til TCNTx i kva som helst modus, vil blokkera alle compare matchane for ein heil klokkesyklus. Difor er det ein risiko å endra TCNTx når ein brukar Output Compare. For å unngå dette i fylgje datablad, skal TCNTx ikkje skrivast lik toppverdi, dersom toppverdien variera og heller ikkje lik botnverdien, dersom det skal teljast nedover.

COMxy: Compare Output Mode for kanal y. Desse bitane ligg i kontrollregistera for timerane. Dei styrer OCxy pinnane og korleis dei skal oppføra seg, etter kva modus som er vald.

OCxy: Output Compare, er ikkje oversatt i denne rapporten og er det ordet som vert brukt vidare. Dette er tilstanden og verdien til output compare eininga. Det kan vera både interne verdiar og verdien på pinnen eller utgangen.

OCRxy: Output Compare Register. Desse registera inneheld 16 eller 8 bit verdi som kontinuerleg vert samanlikna med TCNTx. Dersom det er match eller treff mellom desse, vil eit flagg OCF1x bli sett. Dersom ein i tillegg aktiverer OCIE1x bit, vil flagget generera eit avbrot.

OCFxy: Timer x Output Compare y Match Flag. For timer x på utgang/pinne y, vil flagget bli sett til 1 idet TCNTx = OCFxy. Idet avbrotsvektoren køyrer blir den sett til 0 igjen, eller ein kan skriva den til 0 i registeret.

OCIExy: Output Compare y Match Interrupt Enable. For timer x, når OCIExy er satt til 1 og globalt avbrot er aktivert vil korresponderande vektor køyra, når OCFxy vert 1. Denne heng saman med OCFxy og dersom denne er sett til 0, vil ikkje korresponderande vektor køyra.

TOVx: Flagg som vert sett til 1 når TCNTx når toppverdien. Det vert sett til 0 att når korresponderande vektor har køyrt.

TIMERx_COMPy_vector: Dette er dei korresponderande vektorane som køyrer når det er treff mellom TCNT1 og OCRxy. Altså t.d. Timer1_compA_vector vil køyra når TCNT1 = OCR1A. Desse vektorane er ISR-funksjonar.

Dobbel buffring: OCRxy registera er dobbel buffra så lenge dei er i rett modus. Dobbel buffringa synkroniserer oppdatering av OCR1x til topp eller botn, for å hindra usymmetriske pulsar. Dersom det nyttast dobbel buffring, har kontrollaren direkte tilgang til buffer registeret, og nyttast det ikkje buffring har den direkte tilgang til OCRxy rgisteret.

Modusar

For alle timerane kan det veljast modus. Det vil sei korleis timeren skal telja og korleis utgangane skal oppføra seg. Dette definerast gjennom å kombinera WGMx3:0 bitane som står for Waveform Generation Mode. Tar som eksempel tabell 15.5 frå datablad, Waveform Generation Mode Bit Description, for timer 1, figur7.5. Her fins 16 ulike modusar. For timer2 vil det vera 8 modusar.

For timer1 ved å skru på eller av, WGM13, WGM12, WGM11 og WGM10 vel ein mellom modus 0 til 15. Ein ser kva toppverdien vil vera, og når OCR1y vert oppdatert, i tilleg til når TOV1 flagget vert sett, altså TOV1 = 1.

Table 15-5. Waveform Generation Mode Bit Description(1)

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, phase and frequency correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, phase and frequency correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, phase correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, phase correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Figur 7.5: Ved å skru på eller av, WGM13, WGM12, WGM11 og WGM10 vel ein mellom mode 0 til 15.[20]

For å bestemma korleis pinnane for Output Compare skal oppføra seg, set ein COMxy0:1 bitane til 1 eller 0, etter kva modus ein er i. T.d for timer2 som skal vera i Phase Correct PWM modus, brukast tabell 17-4 i datablad, sjå figur 7.6.

Table 17-4. Compare Output Mode, Phase Correct PWM Mode(1)

COM2A1	COM2A0	Description
0	0	Normal port operation, OC2A disconnected.
0	1	WGM22 = 0: Normal port operation, OC2A disconnected. WGM22 = 1: Toggle OC2A on compare match.
1	0	Clear OC2A on compare match when up-counting. Set OC2A on compare match when down-counting.
1	1	Set OC2A on compare match when up-counting. Clear OC2A on compare match when down-counting.

Figur 7.6: Tabell for Compare Output Mode i Phase Correct PWM Modus

For å nå desse bitane lyt ein skriva til kontrollregistera for timeren. T.d. for timer2, ligg bitane i TCCR2A register, figur 7.7. Ved å fyrst bruka tabell 17-4, for å finna ut kva kombinasjon av bitane ein skal ha, og deretter finna desse bitane i TCCR2A, altså bit 7 og 6 i dette tilfelle, og velja om desse skal setjast til 1 eller 0, kontrollerer ein desse.

17.11.1 TCCR2A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 7.7: TCCR2A register

15.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 7.8: TCCR1B

Normal modus: Dette er den enklaste innstillinga, og timeren vil alltid telja oppover. Teljaren går frå null til den når maks(0xFFFF for timer1), og startar deretter frå botn igjen(0x0000). Output compare einingane kan i denne modusen brukast til å skapa avbrot. I fylgje datablad bør dei ikkje nyttast til å generera bølgjesignal, fordi det vil okkupera for mykje tid [20][s.100].

Clear Timer on Compare Match (CTC): I denne modusen vil timeren telja på same måte som for normal modus, men teljaren blir sett til null når teljeverdien(TCNTx) vert lik enten OCRxy eller ICRx. Desse verdiane definerer toppverdien for teljaren, altså kor langt den tel før den vert sett til null. Det kan for eksempel genererast avbrot kvar gong TCNT1 vert lik toppverdien.

Fast PWM modus: I denne modusen startar teljaren frå botn, tel til toppverdi og tel frå botn igjen. Utan å telja nedover. Oppløysinga for fast PWM lyt vera minimum 2-bit(0x0003) og opp til maksimum 16bit. Dette definerast med ICR1 eller OCRxy. Teljaren kan telja opp til TCNTx vert lik ein av forhåndsverdiane(0x01FF, 0x03FF, 0x00FF, med WGMx3). Eller til TCNTx = ICR1 eller OCR1A. Så vert teljaren nullstilt.

TOV1 flagget blir sett til 1 kvar gong timeren når toppverdien. Flagga OC1A og ICF1 vert også sett lik 1 på toppverdien, når ein nyttar OCR1A eller ICR1 for å setja toppverdi. Dersom eit av avbrota vert brukt, kan det i avbrotsfunksjonen oppdaterast topp og compare verdiar. For at dette skal gå an, lyt den nye toppverdien vera lik eller høgare enn verdien til eksisterande compare verdi.Ellers kan det vera at compare match aldri oppstår.

Grunna OCR1A er dobbelbuffra, så kan den skrivast ny verdi til kor som helst i programmet. Den verdien som ligg i buffer vil bli brukt til OCR1A neste gong TCNT1 matchar toppverdi, altså på neste compare match. Oppdateringa vert gjort samstundes som TCNTx er nullstilt.

ICR1 registeret derimot er ikkje dobbelbuffra. Så dersom ICR1 vert oppdatert til ein lågare verdi medan teljaren går, er det ein risiko for at ICR1 vert lågare enn TCNT1 og compare match aldri oppstår. Då lyt teljaren telja heilt til toppverdi og deretter til den nye verdien igjen, før compare match kan oppstå.

I fylgje databladet er det på bakgrunn av dette eit klart betre alternativ å bruka OCR1A som toppverdi, dersom den skal endrast i programmet.

Phase Correct PWM modus: I denne modusen går teljaren kontinuerleg frå botn til topp og til botn. Ellers gjeld dei same alternativa og moglegheitane som for fast PWM.

Phase and Frequency Correct PWM modus: Denne modusen fungerer likt som Phase Correct PWM, berre det er ulikt tidspunkt for når OCR1y registeret og bufferregisteret vert oppdatert.

Register:

I programkode vert det skrive til fleire register. Det vert gjort med bit manipulering. Altså setja bit i registera til 1 eller 0, skru dei på eller av.

I kontrollregister A og B, TCCR1A og TCCR1B, stillast det inn kva modus timeren er i, val av compare output og val av skalering. Ved å t.d. kombinera bit 4(WGM13) og bit 3(WGM12) i TCCR1B, og bit 0(WGM10) og bit1(WGM11) i TCCR1A, vel ein modus for timer1.

Bit 4-7 i TCCR1A styrer output compare mode for kanal A og B. Etter kva modus ein har vald, lyt ein fylgja den rette tabellen. Desse bitane for timer1 ligg i TCCR1A, som vist i figur 7.9.

Bitane CS12, CS11, CS10 ligg i kontrollregister B, figur 7.10 og er skaleringsinnstillingane.

Kontrollregister B har også bit 6 og 7, IGNC1 og ICES1. Desse er ikkje i bruk i dette prosjektet, men gjev tilgang til filter for støykansellering.

15.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 7.9: Tabell frå datablad[20]s.188

15.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 7.10: fig:Tabell frå datablad[20]s.

7.7.2 Timer1

Med timer1 kan ein få meir nøyaktige verdiar enn med timer2, då det er 16bit timer. Den vert i programmet brukt til å justera tennvinkelen. Altså skapar den forsinkinga etter nullgjennomgang, før pulstoga vert skrudd på.

TCNT1 er teljeverdien til timer1, altså der den er komt frå 0-65535. OCR1A er ein verdi ein kan bestemma og som kontinuerleg vert samanlikna med TCNT1.

Timer1 i CTC modus

Timer1 er testa i CTC modus. Då vil timeren verta nullstilt i det TCNT1 = OCR1y. OCR1y vil då vera tennvinkelverdien, omrekna frå alfa til OCR1y verdi. Frå tabell 15-2 i datablad vart COM1A1 og COM1B0 bit satt til 1. Det tyder at OC1y vil bli sett til 1 idet TCNT1 = OCR1y. Slik vil timeren telja til OCR1y, som då er alfaverdien mellom 0-10ms, og setja utgangen høg.

Med denne modusen lyt ein i tillegg ha ein sjekk, for å sjekka om utgangen er høg og i det augeblikk lyt pulstoga verta skrudd på. Dette virka under testing, men med noko feil på timing av pulstoga. Det som kan vera ei utfordring er at sjekken tek litt for lang tid og programmet mistar nokre av compare matchane som oppstår. For å koma rundt denne sjekken, er bruk av TIMSK1(Timer/Counter1 Interrupt Mask) eit alternativ.

I TIMSK1 registeret setjast bit1, OCI1A til 1. Då vil avbrot for output compare match A vera aktivert, dersom I-flagget er sett. I-flagget vert sett idet TCNT1 = OCR1A, det skjer automatisk utan noko bit manipulering. Ein vektor, Timer1_CompA_vect, vil køyra på dette treffet.

Med desse innstillinga, vil det vera ei god løysing å skru av og på pulstog i den korresponderande vektoren til OCI1A. Då må det i vektoren nullstillast TCNT1, slik at ein får med neste pulstog også.

Timer1 i Normal modus

Timer1 skal kun telja frå null og opp til tennvinkel verdien. Dette kan også gjerast i normal modus. Det skal ikkje brukast nokre utgangar, så compare

output mode trengs heller ikkje brukast. Difor vert heile kontroll A register sett til null. TCNT1 vil uansett kontinuerlig samanliknast med OCR1A verdien, utan at noko nødvendigvis skjer med utgangane. Det tyder at det kun er skaleringa som treng brukast. Det gjer ein i kontroll B registeret. I TCCR1B kan ein bruka ulike skaleringar for å gjera utrekning til OCR1A enklast mogleg.

Med desse innstillingane er det ein timer som tel frå null og oppover, heilt til TCNT1 vert nullstilt. I tillegg samanliknar den kontinuerleg TCNT1 med OCR1A verdien.

Frå datablad finn ein i register TIFR1 for bit1, kalla OCF1A, at dette flagget vert sett idet TCNT1 = OCR1A. Frå datablad finn ein også i register TIMSK1, bit1 kalla OCIE1A. Dersom denne vert sett til 1 og OCF1A flagget er sett, som det er automatisk, vil avbrotsvektoren bli kalla idet TCNT1 = OCR1A. Denne vektoren vil køyra, men TCNT1 vil fortsetja å telja oppover heilt til den vert nullstilt. Samtidig vil OCF1A bli sett til 0 att når vektoren køyrer. TCNT1 lyt nullstillast synkronisert med neste nullgjennomgang.

Timer1 vil køyra heila tida, uavhengig av resten av koden. Sidan koden skal innehalda start og stopp knapp, trengst det funksjonar for å skru av og på timer1. Dette vert løyst med å skru av og på TIMSK1 bit. Dersom OCIE1A vert sett til 0, vil ikkje vektoren køyra og ingenting i programmet knytt til timer1 vil skje.

Listing 7.6: Funksjonar for å skru av og på timer1

```
1         void timer1on(){
2             TCNT1 = 0;
3             TIMSK1 |= (1<<OCIE1A) | (1<<OCIE1B);
4         }
5         void timer1off(){
6             TCNT1 = 0;
7             TIMSK1 &= ~((1<<OCIE1A)| (1<<OCIE1B));
8         }
```

Timer1_CompA_vector skal skru på pulstoga, medan i ISR($INT0\_vect$) skruast pulstog av etter kva flanke programmet er på. Det må difor vera eit flagg i funksjonen som vert lest i timer1_CompA_vector for å finna ut kva for eit pulstog som skal skruast på, etter kva flanke ein er på. Dette kan føra til mindre effektivt program, då sjekk av flagg skapar undøvendig forsinking i timer1_CompA_vector.

Ei anna moglegheit er at det kan nyttast to ulike vektorar med to ulike OCR1x verdiar.CompA_vector og CompB_vector for timer1. Difor kan CompA_vector skru på det eine pulstoget og CompB_vector skru på det andre. Vektorane skal ha same forsinking etter nullgjennomgang, etter kvar sin flanke. Det vert løyst ved å skru av timer1 i vektorfunksjonane etter at pulstoga vert skrudd på. Så vert timer1 skrudd på att i ISR(INT0_vect). Difor vert det i timer1off funksjonen lagt inn at TCNT1 = 0. Å skru av timer1 i begge vektorfunksjonane virka, men det kan gjerast meir effektivt.

For ei enklare og meir effektiv løysing skal CompB_vector alltid køyra ein halvperiode etter at CompA_vector køyrer. Det vil sei at OCR1B kan reknast ut til

å vera nøyaktig ein halvperiode meir enn OCR1A. Då treng TCNT1 nullstillast berre ein gong på ein periode. Slik vert pulstoga skrudd på i kvar sin vektor.

Gjennom alle testane, har det vore eit problem at idet tennvinkel nærmar seg 180 grader, vert pulstoget ustabilt og skrudde seg av og på tilfeldig. Det same gjaldt idet tennvinkel vart sett til 0. Dette vart løyst ved at lågaste verdi er 1, altså dersom tennvinkel = 0, vert den i programmet lik 1.

Resultat: Timer1 vil i programmet telja frå null og oppover til TCNT1 = 0. Som vist i kodedel 7.7, er det berre skaleringa som vert stilt inn.

Listing 7.7: Innstilling av timer1

```
1 void timer1setup(){
2 TCCR1A =0;
3 TCCR1B = (0 << WGM12) | (0 << CS12) | (1 << CS11) | (0 <<
      CS10);
4 }
```

7.7.3 Timer2

Timer2 skal generera pulstoga. Dette vert gjort med fylgjande innstilling i koden 7.8 :

Listing 7.8: Timer2 innstilling

```
1 void timer2setup(){
2 TCCR2A = _BV(COM2A0) | _BV(WGM20);
3 TCCR2B = _BV(CS22)|  _BV(CS21) | _BV(CS20);
4 OCR2A = 130; //pin 11
5 OCR2B = 130; //pin 3
6 }
```

Ved å velja WGM20 er timeren i Phase correct PWM modus, med 0xFF som toppverdi. Det ser ein i figur 7.7 . Grunnen til at denne modusen vert brukt i staden for Fast PWM modus, er at frekvensen vert halvert ved dei same innstillingane, då den tel både opp og ned. Difor kan ein i tillegg tl å justera skalering, få dobbelt så mange val av frekvens ved å byta mellom Fast PWM og Phase Correct PWM.

Det er funksjonar som brukar timer2 til å skru av og på pulstoga 7.9 . Det er to par for kvar funksjon, som styrer kvar sin utgang. For utgang 11, gjeld T12pulseon() og T12pulseoff(). For utgang 3, er det identisk, men A er erstatta med B for å gjelda utgang på pinne 3.

Ved å setja COM2A0 og COM2A1 så vel ein at utgangen vert høg på compare match og låg når teljaren tel nedover igjen. Slik vert det generert eit PWM signal. Ved å setja dei til 0, vert utgangen fråkopla, og dei gjev ikkje ut noko signal. Det er desse bitane som skrur av og på pulstoga.

OCR2y verdiane bestemmer toppverdien, og dermed arbeidssyklusen til pulstoget. Dersom det skal reknast i prosent, kan ein setja OCR2y = $\frac{255}{100}$·(ynskt prosent).

Listing 7.9: Funksjonar for av og på pulstog

```
1 void T12pulseon() {
2 TCNT2 = 0;
3 TCCR2A |=  (1<<COM2A0) | (1<<COM2A1);
4 TCCR2B |= ((0<<CS22) | (0<<CS21)|(1<<CS20));
5 }
6 void T12pulseoff(){
7 TCCR2A &= ~( (1<<COM2A0) | (1<<COM2A1));
8 TCCR2B &= ~((1<<CS22) | (1<<CS21)|(1<<CS20));
9 }
```

Det er modus og skalering som kan justera frekvensen. Frekvensen lyt veljast ut frå kva pulstrafoar som skal brukast.

Med dei innstillingane som er vald, trengs det i programmet funksjonar som sjekkar om pulstoga er på. Dei returnere enten true eller false, for på eller av, sjå vedleggE.4.

7.8 I2C

I dette prosjektet er I2C nytta for kommunikasjon mellom ESP32 som kommuniserer med applikasjon, og Arduino Uno som styrer tyristorane.

I2C, også kalla TWI, står for Inter-Integrated-Circuit og Two-Wire-Interface. Det er ein synkron kommunikasjonsprotokoll. Altså er det einingar som kommuniserer og lyt ha eitt felles klokkesignal. Her er to ledningar, ein for klokkesignal og ein for distribusjon av data. Desse koplast mellom slave og master. Kommunikasjonen vil vera kontinuerleg for desse to einingane og ein fordel med I2C er at det kan vera fleire slavar på ein master [17] [10].

Dei to ledningane kallast SCL(Serial Clock), og SDA(Serial Data). Klokkesignalet vert generert av masteren, her ESP32. Det er master som initierer til kommunikasjon. Hadde det vore fleire slavar måtte dei hatt kvar si individuelle adresse som master kan referera til. Her er det berre ein slave som er i bruk. Master kan både senda til og motta data frå slaven. Her skal kun master senda til slave [17].

I prosjektet er Wire.h bibliotek av Arduino nytta for I2C.

7.9 Blynk

Blynk er ein applikasjon og ein IoT-plattofrm. Det verkar ved at t.d. ein mobil og ein mikrokontrollar, som ESP32, kan kommunisera gjennom ei sky. Blynk støttar hardware plattformar som Arduino, ESP32, Raspberry PI og andre. Den verkar med ethernet, wifi, bluetooth, cellular og serial kopling. Blynk brukar ein applikasjon, ein server og blynk sine bibliotek. Når du utfører ei handling i applikasjonen, t.d. trykker ein knapp, så sender appen dataen til Blynk si sky. Data vert lagra på server og skya sender så instruksar/data vidare til mikrokontrollaren [10] [22].

Blynk server: Serveren står for kommunikasjon mellom hardware og mobil. Her kan ein også velja om ein vil bruka Blynk si eiga sky eller setja opp ein eigen lokal Blynk server.

Blynk bibliotek: Verkar for mange maskinvarer, det er desse som står for kommunikasjon med server, tek imot kommandoar som kjem inn og sender ut kommandoar som skal vidare.

7.9.1 Oppsett av Blynk

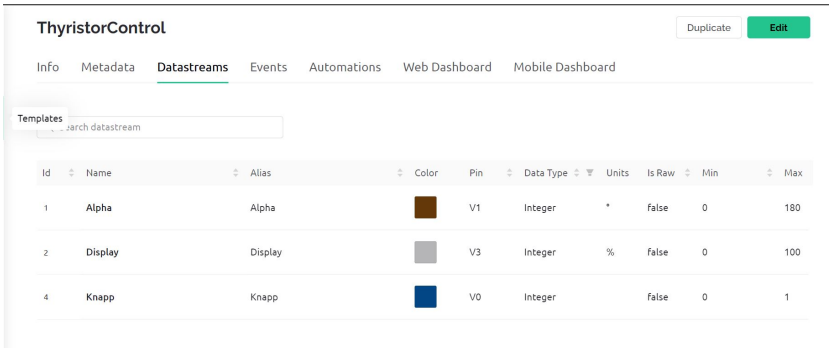
Template: Konfigurasjon av einingar vert lagra i såkalla templates. Kan samanliknast med eit oppretta prosjekt for eininga, t.d. for ESP32 [22].

Widgets: Forhåndsdesigna delar av det grafiske brukargrensesnittet, slik som knappar og slider [46].

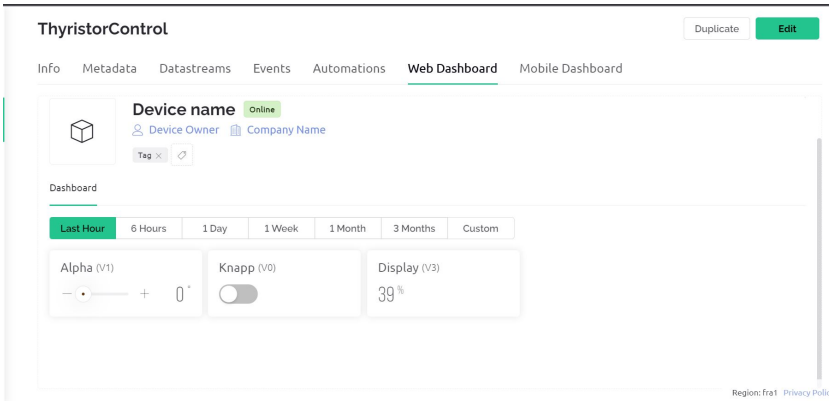
Datastreams: Datastreams er kanalar som vert sett opp for å senda data mellom applikasjon og Blynk skya [43].

I dette prosjektet er den nye versjonen av Blynk tatt i bruk, Blynk IoT. Det vert gjort ved å fyrst oppretta eit template på web-dashboard7.12. Her lyt ein bestemma kva widgets ein skal bruka og kva datastreams som skal høyra til 7.11.

Template, kalla ThyristorControl, inneheld ein on-off-knapp for å bestemma om ein skal bruka app som kontroll, altså fjernstyring av tennvinkel eller ikkje. I tillegg er det ein slider for å justera alfa, og eit display som viser alfa i prosent. Av-og-på knappen sender ein "integer datastream", altså 1 eller 0 til ESP32. Programmet i ESP32 les 1 som R for remotecontrol modus og 0 som L for localcontrol modus. Slideren stillast inn for å gå frå 0 til 180, altså alfa i grader og sender talverdien til ESP32. I ESP32 vert denne verdien rekna om til prosent og sender tilbake til applikasjon for å visast på display.



Figur 7.11: Utklipp frå datastream i blynk web dashboard



Figur 7.12: Utklipp frå Dashboard i Blynk Web

Applikasjonen lyt lastast ned. Figur D.3 viser startside i applikasjon. Ein må oppretta brukar og ved hjelp av BLYNK AUTH TOKEN og at mobiltelefon er tilkopla same nett som mikrontrollaren, kan ein kommunisera mellom dei. I applikasjonen legg ein til widgetsD.4 i forbindelse med dei datastreams som er valt i web dashboard. Så nyttar ein Blynk biblioteket BlynkSimpleESP32 av Volodymyr Shymanskyy.

7.10 Funksjonar i programkode

Tabell 7.1 er ein oversikt over alle funksjonane i programmet som er skrive.

Tabell 7.1: Forklaring av funksjonar

Funksjon	Forklaring
$slaverecievefrommaster()$	Les inn verdiar frå ESP32, dersom det er data å lesa. Lagrar verdiane, modus og alfa for å brukast vidare
$lc()$	Set opp og skriv ynskt data i lcd display.
$SamplePushbuttons()$	Funksjon for å detektera om knappar er trykt, og hindra debounce
$timer2setup()$	Innstillingar for timer2
$T12pulseon()$	Skrur på pulstog på pinne 11
$T12pulseoff()$	Skrur av pulstog på pinne 11
$T34pulseon()$	Skrur på pulstog på pinne 3
$T34pulseoff()$	Skrur av pulstog på pinne 3
$is\_pulse12\_on()$	Sjekkar om pulstog er på eller av på pinne 11. Returnerer true for på og false for av
$is\_pulse34\_on()$	Sjekkar om pulstog er på aller av på pinne 3. Returnerer true for på og false for av
$timer1setup()$	Innstillingar for timer1
$timer1setalpha()$	Omrekning frå alfaverdi enten frå potmeter eller ESP, til OCR1Y-verdiar
$timer1on()$	Nullstiller timer1, og aktiverer at avbrot skal oppstå på compare match i TIMSK1 registeret
$timer1off()$	Nullstiller timer1, og deaktiverer avbrot på compare match
$get\_comp\_alpha()$	Funksjon som returnerer korrekt alfaverdi
$get\_comp\_180()$	Funksjon som returnerer korrekt verdi for alfa lik 180
$interruptSetup()$	Innstillingar for når avbrot skal aktiverast og kva ISR som skal køyra
$ISR(INT0\_vect)$	ISR som køyrer for avbrot, i tillegg skrur av pulstog. Sjekkar modus og set OCR1Y verdiar
$ISR(Timer1\_COMPB\_vect)$	ISR som køyrer når TCNT1 = OCR1B, som skrur på eit pulstog
$ISR(Timer2\_COMPA\_vect)$	ISR som køyrer når TCNT1 = OCR1A, som sjekkar kva flanke ein er på, skrur av eller på pulstog, i tillegg oppdaterer OCR1A

Kapittel 8

Tyristorkrins

Tyristorkrins omfattar alle komponentar og krinsar frå pulstoget til Arduino til gate på tyristorane. Det er tatt utgangspunkt i krins frå simple-circuit nettside [9] som vart presentert av veiledar under fyrste oppstartsmøte. Krinsen består av pulstoget, som skal sendast inn på gate på tyristoren, transistorar, kraftforsyning, pulstransformator, motstandar, diodar og kondensator. For kravet om galvanisk skilje er det nytta pulstransformatorar. Desse lyt drivast av ei kraftforsyning ut ifrå kva frekvens som skal overførast. Vidare er det gjort greie for utrekningar og val av komponentar.

8.1 Komponentar:

Motstandar

Kondensator: CK05BX472K [31]

Transistorar: IRF520[48], IRLZ34 [30].

Diodar:RL14007-T[29]

Pulstransformatorar: 1003C[27], IT370[39].

Gate drivar chip: IR2110[47], HCPL-3120[34].

Tyristorar: SKKH26[42], TYN20[8].

Kjøleribbe: TO-220[26].

8.2 Oppbygging av krins:

Ein pulstransformator har som oppgåva å overføra spenningspulsar mellom krinsar som har behov for galvanisk isolasjon.[12]. Det er for det meste som isolasjonstransformatorar dei vert brukt. [13]. Storleiken er avhengig av spenning-tid produktet, altså E·t, [V/μs]. Dette kan ikkje overskridast då trafoen, eller det magnetiske materialet vil gå i metning. Det tyder at den magnetiske fluksen vert større enn kva jernkjerna kan takla, som igjen fører til at påtrykt straum på

primærside ikkje fører til tilsvarande straum på sekundærsida[16]. Denne verdien er oppgitt i datablad, eller det er ein viktig parameter som lyt reknast ut dersom ein bygger trafoen sjølv [12].

T.d. har IT370 pulstrafoen eit spenning-tid produkt på 4000V/μs. Det vil sei U·t ikkje kan verta større enn 4000. T finn ein ut frå frekvensen. T.d. hadde frekvensen vore 1000Hz med 50% arbeidssyklus, ville t ha vore 50% av 1ms, sjå utrekning 8.1, altså 500μs og spenningen kunne ha vore maks 8V, fordi U·t = 8·500 = 4000.

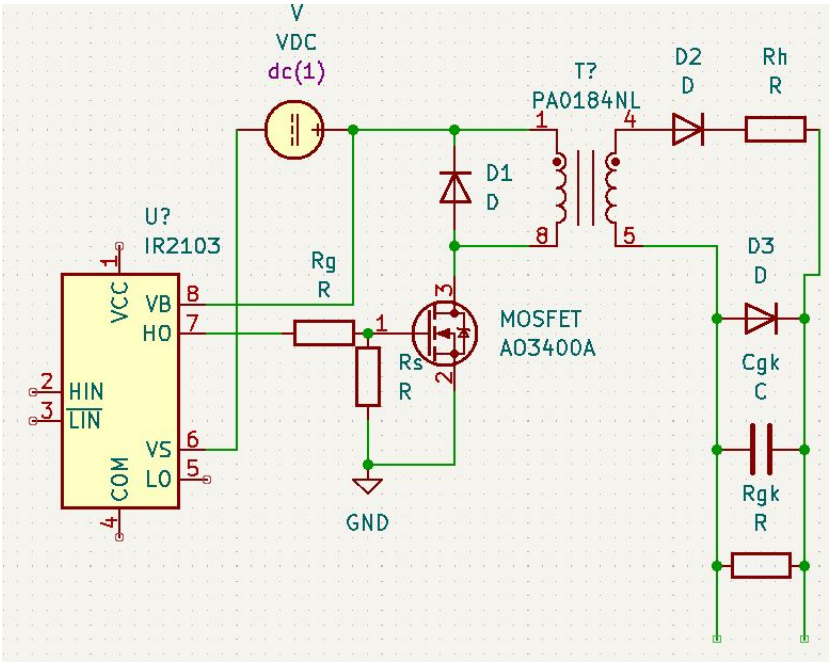
$$T = (\frac{1}{1000}) = 1ms \tag{8.1}$$

Frekvensen som skal brukast på pulstoget kan justerast, som gjort greie for i kapittel 8. Pulstoget direkte ut frå Arduino har maksimum spenning på 5V og maksimum straum på 40mA. For å levera stor nok straum til tyristorane, medrekna spenningsfallet over diodar og motstandar, vert i mange tilfeller Arduino sine verdiar for låge. Difor er det sett inn ein MOSFET transistor og ei kraftforsyning. Transistoren verkar som ein brytar synkronisert med pulstoget. Det vil sei at det pulstoget som går inn på pulstrafoen har ein amplitude på den spenningsverdien som kjem frå kraftforsyninga, men same frekvens som pulstog ut frå Arduino.

Motstand mellom gate og source på transistor Rs er for å unngå å øydeleggja transistoren dersom noko skulle vera feil eller at den skrur seg på uynskt grunna støy i krinsen. Motstanden som ligg i serie med gate er for å unngå å trekka for stor straum frå Arduino, som berre kan levera 40mA. Ved bruk av gate drivar chip, så vert denne motstanden sett inn i serie med anode, og inn på gate til transistor vert ein motstand Rg sett inn. Rg har oppgitt anbefalt verdi i datablad og har som formål å hindra lekasjestraum [7] [6].

På sekundærsida er det ein diode D2 og motstand Rh kopla i serie. Dioden er for å sikra at det ikkje går negativ straum inn på sekundærsida. Motstanden kontrollerer kva straum som går inn på gate. Filterkomponentane, altså motstanden Rgk og kondensatoren Cgk som ligg i parallell med dioden, sitt formål er å filtrera vekk støy på linja inn til gate.[40].

Gjennom alle desse komponentane vil ein få eit signal inn på gate med frekvens og form som pulstoget, med korrekt straum i forhold til I_{GT}, 8.1.



Figur 8.1: Tyristorkrins designa i Kicad

8.3 Utrekningar

.

8.3.1 Transistor IRLZ34:

Omsett effekt i transistoren kan uttrykkast ved hjelp av likning8.2 henta frå [1]:

$$P_d = \frac{T_J - T_A}{R_{TH,JC} + R_{TH,CS} + R_{TH,S}} \tag{8.2}$$

Der T_J er maksimal indre temperatur i transistoren, T_A er omgjevnadstemperatur, $R_{TH,JC}$ er termisk resistans til monteringsflata for transistoren, $R_{TH,CS}$ er termisk resistans frå monteringsflata til kjøleribba og $R_{TH,S}$ er termisk resistans til kjøleribba. Alle verdiar er henta frå datablad.

Kjøleribba som var tilgjengeleg var på 8°C/W. Ein kan også snu på likninga, dersom ein skal finna ut kva kjøleribbe ein treng. Då lyt det fyrst vera kjend kva effekt som skal verta omsett i transistoren og fylla inn i likning 8.3. Resultatet gjev ein termisk resistans kor ein lyt velgja kjøleribbe med lågare verdi enn denne.

$$\frac{T_J - T_A}{P_d - R_{TH,JC} - R_{TH,CS}} \tag{8.3}$$

T.d. for transistor IRLZ34 er $R_{TH,JC} = 2.2$°C/W og $R_{TH,CS} = 0.50$°C/W. $T_J = 175$°C og $T_A = 40$°C, då det ikkje vert gjennomført noko lab dersom omgjevnadstemperaturen overstig 40°C. Desse verdiane finn ein i Thermal resistancectabell i datablad[30], sjå figur 8.2 . Utrekning i likning 8.4, viser at den kan ha ein omsett effekt på 12.61W, med denne kjøleribba montert.[1][s.711]

$$P_d = \frac{175 - 40}{R_{2.2+0.5+8}} = 12.61W \tag{8.4}$$

Thermal Resistance

	Parameter	Min.	Typ.	Max.	Units
$R_{\theta JC}$	Junction-to-Case	—	—	2.2	°C/W
$R_{\theta CS}$	Case-to-Sink, Flat, Greased Surface	—	0.50	—	
$R_{\theta JA}$	Junction-to-Ambient	—	—	62	

Figur 8.2: Thermal Resistance for IRLZ34

I tillegg kan det reknast leietap og svitsjetap for transistorane. For å rekna ut leietap 8.5 [5][s.23] P_{on} lyt I_d og $R_{DS,on}$ vera kjend. I_d er straumen som går i krinsen og inn på gate. Det vil sei at tyristor må vera vald og rekna ut kva triggestraum den skal ha. $R_{DS,on}$ er motstandsverdi mellom drain og source, som variera med U_{DS}, som ein finn i datablad. Verdien kan og lesast ut frå tabellar, som Fig2. i datablad, sjå figur 8.3.

For svitsjetap 8.6 [5][s.24] P_{sw} , lyt ein i tillegg vita U_d, som i denne krinsen er spenninga frå kraftforsyninga minus spenningsfall over dioden. Det trengs også frekvensen til pulstoget F_{sw} og rise-time t_{on} og fall-time t_{off}, som hentast frå datablad.

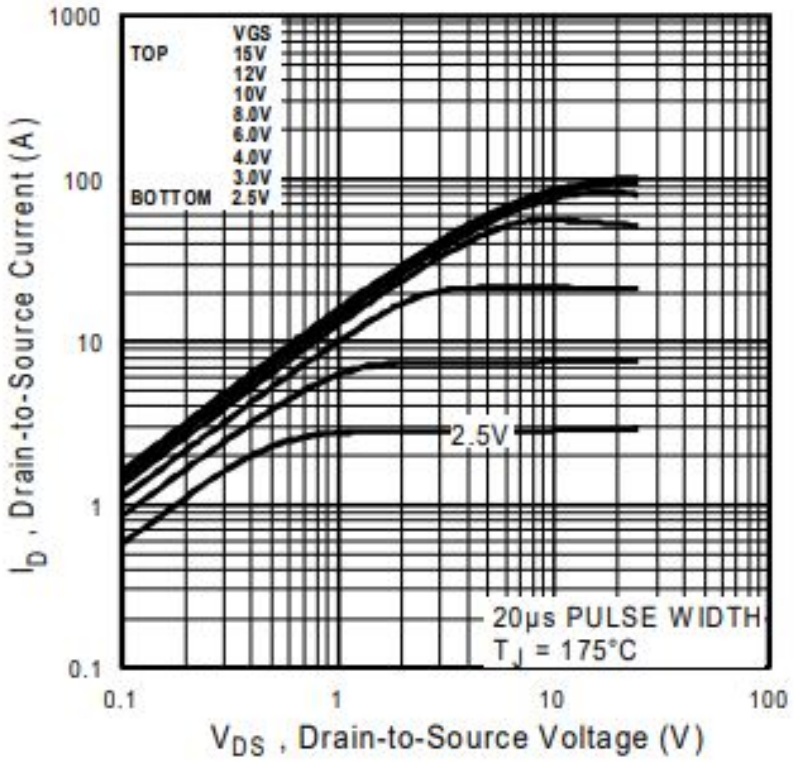


Fig 2. Typical Output Characteristics

Figur 8.3: Fig 2. i datablad[30][s.3]

$$P_{on} = I_d^2 R_{DS,on} \tag{8.5}$$

$$P_{sw} = \frac{1}{2} U_d I_d f_{sw} (t_{on} + t_{off}) \tag{8.6}$$

8.3.2 Transistor IRF520:

Dei same utrekningane er gjort for IRF520 transistoren, med valde verdiar for $I_d = 0.5$A, $U_d = 15.3$V og $F_{sw} = 1960$Hz. Som er verdiar som vart brukt under nokre av testane.

$$P_{on} = I_d^2 R_{DS,on} = 0.5^2 0.8 = 0.2W \tag{8.7}$$

$$P_{sw} = \frac{1}{2} U_d I_d f_{sw} (t_{on} + t_{off}) = \frac{1}{2}(15.3)(0.5)(1960)((30^{-9}) + (20^{-9})) = 3.7^{-4}W \tag{8.8}$$

$$P_d = \frac{T_J - T_A}{R_{TH,JC} + R_{TH,CS} + R_{TH,S}} = \frac{175 - 40}{2.5 + 0.5 + 8} = 12.27W \tag{8.9}$$

8.3.3 Tyristor TYN20:

$I_{GT} = 32$mA, $U_{GT} = 1.3$V, $I_H = 40$mA, $I_L = 60$mA. Det betyr at for å trigga tyristoren treng den i verste fall eit signal på gate som er 32mA og 1.3V. Straumen inn på anode lyt vera over 60mA for at tyristoren skal vera leiande og under 40mA for å ikkje leia.

For tyristoren har ein som for transistor, leietap og svitsjetap. Det er dei same formlane som gjeld 8.10, 8.11 henta frå [2][s.23].

$$P_{on} = I_d^2 R_{DS,on} \tag{8.10}$$

$$P_{sw} = \frac{1}{2} U_d I_d f_{sw} (t_{on} + t_{off}) \tag{8.11}$$

8.3.4 Tyristor SKKH26:

$I_{GT} = 150$mA, $U_{GT} = 3$V frå datablad. For å gjera krinsen meir robust vert det dimensjonert for ein triggestraum på 500mA. I fig.10 i datablad 8.4 ser ein at dette er godt innfor.

For å få ein straum på 0.5A lyt det veljast korrekt motstand på sekundærsida av trafo. For testing kor $I_{GT} = 500$mA, er det rekna ut at kraftforsyninga setjast til 16V med pulstrafoar IT370. Då har ein 16V inn på diode, eit spenningsfall over den på 0.7V, og det skal vera 3V inn på gate. Ved Ohms lov får ein då R $= \frac{U}{I} = \frac{16-0.7-3}{0.5} = 24.6$ og vel R $= 25\Omega$. Her også gjeld dei same tapa som for tyristor TYN20, altså formlane for leietap og svitsjetap.

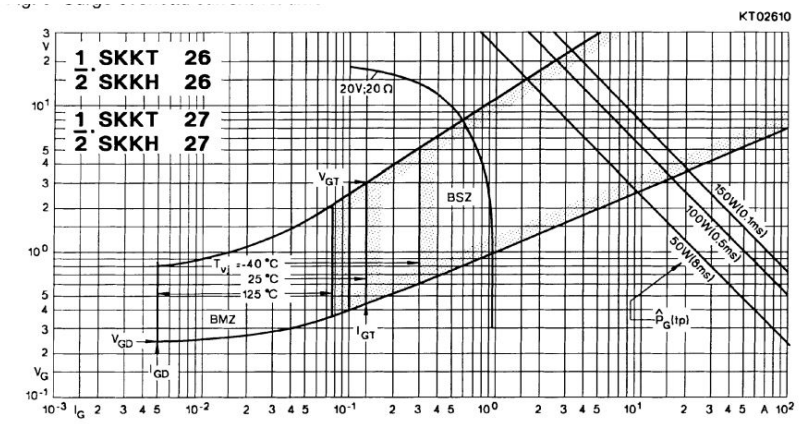


Fig. 10 Gate trigger characteristics

Figur 8.4: Fig.10 frå datablad[42][s.B1-38]

8.3.5 Pulstransformator 1003C:

For pulstransformatorane er E·t-produktet viktig. Som forklart tidlegare kan ikkje dette overstigast. Verdien er henta frå datablad, E·t= 400V/μs. Det er ikkje oppgitt i datablad kva maksimal straum den tåler. Under testing, dersom straumen vart auka til 0.2A og over, vart den svært varm.

Ein fekk samanlikna med pulstrafoar som tidlegare har vorte brukt på lab, desse var omtrent fysisk lika store, og tålte ein straum på 0.25A, så ein kan anta det gjeld 1003C trafoane også.

8.3.6 Pulstransformator IT370:

For å kunna driva dei store tyristorane, trengs ein gatestraum på 150mA i fylgje datablad. Med nødvendig spenning på kraftforsyning for å få denne straumen til gate på tyristor, lyt frekvensen på pulstrafo 1003C vera svært høg. Difor vart det bestilt inn større pulstrafoar IT370. Desse er mykje overdimensjonert for prosjektet, men ein kan køyra ein mykje lågare frekvens, som gjer det meir optimalt for bruk i lab.

Dei store trafoane har E·t = 4000V/μs. T.d. med frekvens på 1960Hz og arbeidssyklus 50% gjeld fylgjande utrekningar: t $= \frac{\frac{1}{1960}}{2} = 255.1\mu$s.

$$U = \frac{Et}{t} = \frac{4000}{255.1} = 15.7V \tag{8.12}$$

Då hadde det vore nødvendig med ei kraftforsyning på 15.7V.

8.3.7 Filterkomponentar:

Verdiane for desse er vald ut frå anbefalte verdiar i Semikron Application Note[40]. Det er anbefalt med kondensator på mellom 10-47nF og motstand på 220-2200Ω. Her er vald C_{GK}=47nF og R_{GK}=2200Ω.

For å finna effekttap over R_{GK} og C_{GK}, lyt V_{GK} og T_{rep} som er frekvensen på pulstoget, vera kjend. Under er formlane sett opp 8.13,8.14 henta frå [40], og det er rekna ut med V_{GK}=3V, og T_{rep}=1960Hz, som er realistiske verdiar ved bruk av dei store pulstrafoane og tyristorane.

$$P_{R_{GK}} = \frac{V_{GK}^2}{2R_{GK}} = \frac{3^2}{2(2200)} = 2.05^{-3}W. \tag{8.13}$$

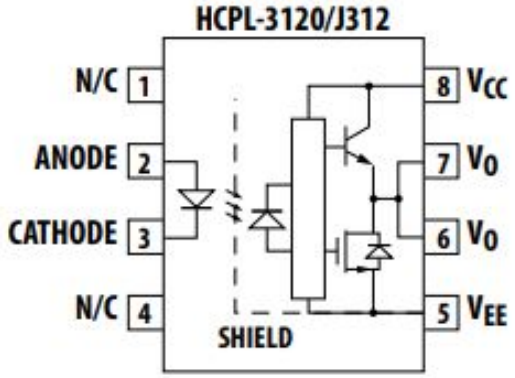
$$P_{C_{GK}} = \frac{C_{GK}}{4} V_{GK}^2 \frac{1}{T_{rep}} = \frac{47nF}{4} 3^2 \frac{1}{1960} = 5.4^{-11}W. \tag{8.14}$$

8.3.8 Gate drivar chip:

Ein lyt også ta omsyn til total gate charge på transistoren. Sjølv etter termiske berekningar og montert kjøleribbe, kan transistoren bli svært varm. Total gate charge er gjeve ved Q = I·t. Så ved låg straum vert tida transistoren brukar på å skru seg på lengre, som gjev større tap.

Ved å auka straumen kan ein minka tapet, og transistoren vert mindre varm. Arduinoen kan ikkje gje større straum enn 40mA. Difor er det sett inn ein gate

drivar chip av typen HCPL-3120[34]. Figur 8.5 viser funksjonsdiagram. Her skal pulstog og jord frfå arduino inn på anode og katode. For V_{CC} og V_{EE}, skal kraftforsyninga koplast på. V_0 gjev ut pulstoget med den ynskte amplitudeverdien, altså spenningsverdien frå kraftforsyninga. Chipen har ein anbefalt $I_{F,ON}$ på 16mA[34][s.13]. For å oppnå dette, er det sett inn ein motstand mellom anode og arduino. Denne er på 270Ω, som gjev $\frac{5}{270} = 18.5$mA. Dette er over anbefalt verdi, men under maksimalverdien som er på 25mA.



Figur 8.5: Funksjonsdiagram frå datablad[34][s.2]

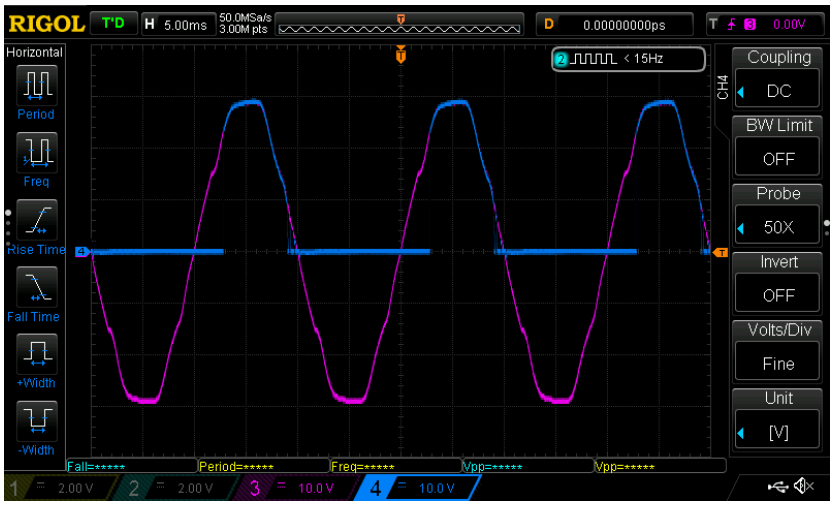
8.4 Halvbølgje testing

Dei ulike komponentane er testa. Etter verifisering av at desse virka som dei skal, vart det kopla opp for testing av halvbølgje likerettar. Oppkopling av test ligg i vedlegg B.1. Testinga er utført med pulstrafo 1003C, tyristor TYN20 og transistor irf520. Bilete av lodda krins ligg i figur 8.11.

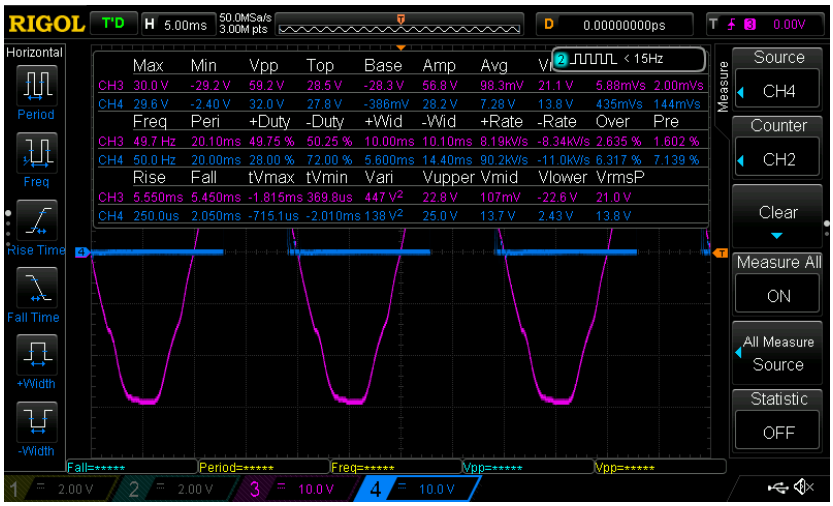
Ved halvbølgje likeretting gjeld, for spenninga U_d og snittspenninga $\bar{U_d}$, fylgjande formlar8.15 8.16 henta frå [2]. Verdiane er rekna ut for utvald alfa-verdi og sett inn i tabell 8.1. Deretter er alfa stilt inn med potensiometer og verdiar lest av frå oscilloskop. Verdiane stemmer godt, då noko tap frå komponentar lyt medreknast. Utklipp frå oscilloskop ligg for $\alpha = 45^\circ$ under tabell 8.6 8.7, og for resten av alfaverdiane i vedlegg B.

$$\bar{U_d} = \frac{U_s}{\sqrt{2}\pi}(1 + \cos(\alpha)) \tag{8.15}$$

$$U_d = \frac{U_s}{2\sqrt{\pi}}\sqrt{\pi - \alpha + \frac{1}{2}\sin(2\alpha)} \tag{8.16}$$



Figur 8.6: Testing med $\alpha = 45^\circ$



Figur 8.7: Verdiar frå oscilloskop med $\alpha = 45^\circ$

Tabell 8.1: Samanlikning mellom berekning og måling

α	Utrekna U_d	Avlest U_d	Utrekna $\bar{U_d}$	Avlest $\bar{U_d}$	Utklipp frå oscilloskop
0°	15.2V	14.6V	9.6V	8.7V	B.2 B.3
45°	14.5V	13.7	8.2V	7.3V	8.6 8.7
90°	10.7V	9.8V	4.8V	3.8V	B.4 B.5
135°	4.6V	3.5V	1.4V	0.5V	B.6 B.7
180°	0V	0V	0V	0V	B.8 B.9

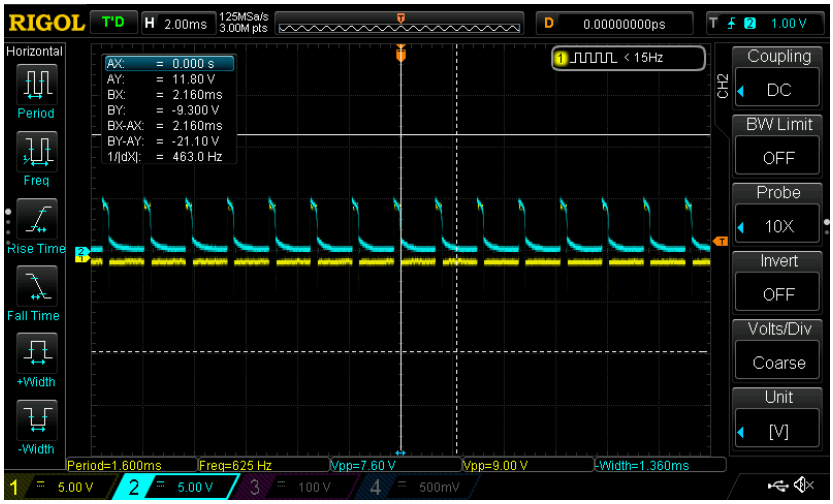
8.5 Hovudkrins:

For hovudkrinsen som skal dimensjonerast for dei store tyristorane SKKH26, er krinsen designa med pulstrafoar IT370. Figur 8.10 viser lodda kort med to tyristorkrinsar. Totalt skal det vera 4 slike krinsar, eller 6 for trefase. Grunna tidsmangel er det ikkje gjennomført testar på desse. Det som er testa er pulstoget inn og ut av pulstransformatoren, sjå figur 8.8. I tillegg til pulstog ut frå gate drivar chipen med ei kraftforsyning på 15V, figur 8.9. Desse ser fine ut.

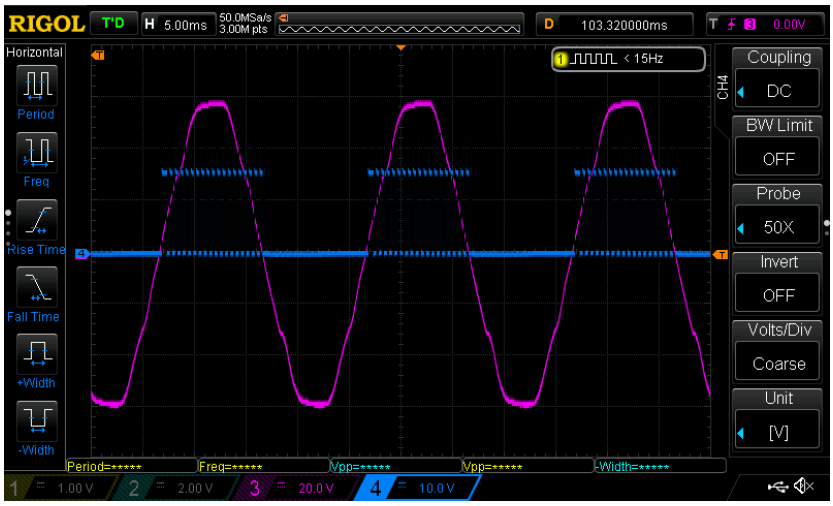
For fullbølge tyristoromformar gjeld fylgjande formlar henta frå [2] :

$$P_{max} = U_{dmax} * I_{dmax} \tag{8.17}$$

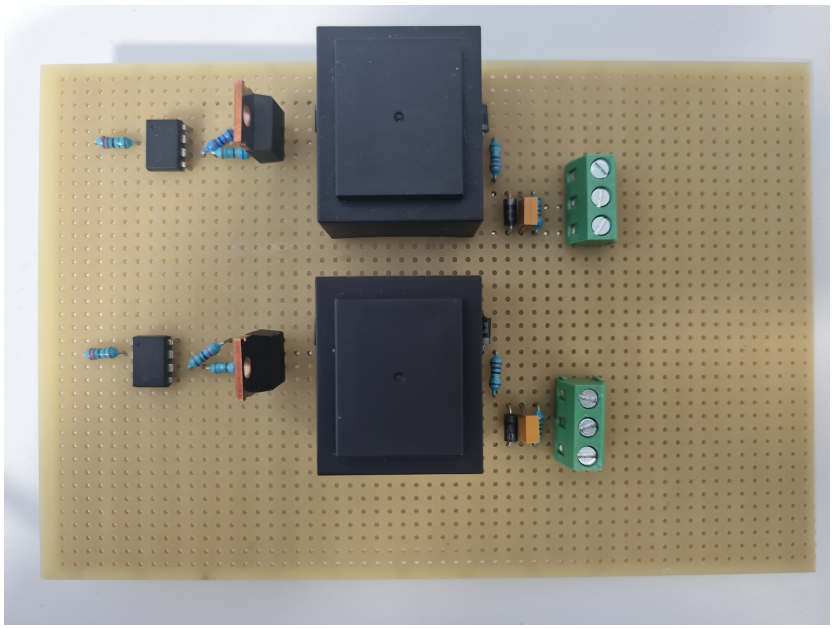
$$U_{dmax} = \frac{U_s 2\sqrt{2}}{\pi} \cos(\alpha) \tag{8.18}$$



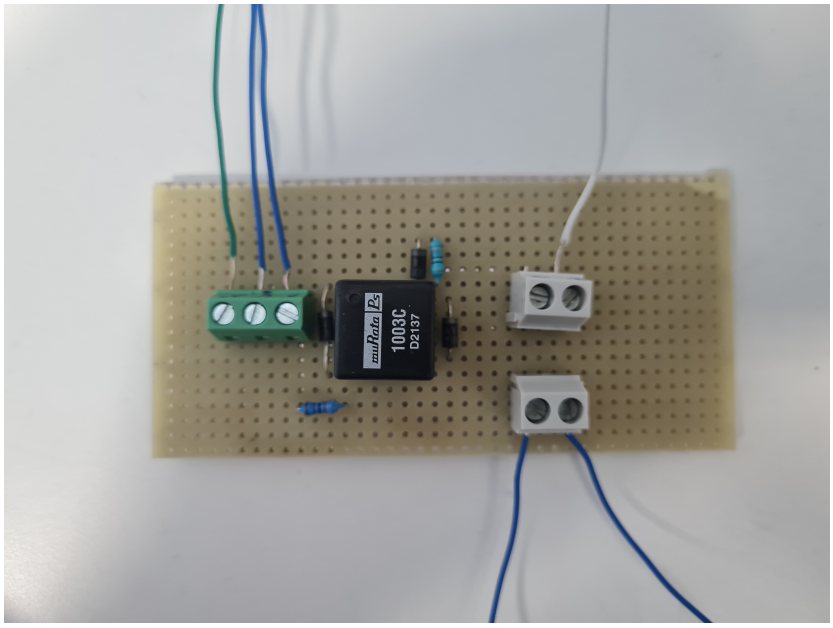
Figur 8.8: Pulstog inn og ut av pulstransformator IT370. Den turkise grafen er pulstoget ut frå pulstrafo.



Figur 8.9: Pulstog ut frå HCPL. Pulstoget har rett frekvens og spenning på 15V



Figur 8.10: To lodda tyristorkrinsar på eit loddekort, med dei store pulstrafoane IT370. To slike kort sett saman er det som manglar av testing



Figur 8.11: Lodda tyristorkrins med den vesle pulstrafoen 1003C

Kapittel 9

Diskusjon og Konklusjon

Prosjektet har vorte jobba med stykkvis. Fyrste del som omfatta nullgjennomgangsdeteksjon vart fyrst løyst, deretter programkode skrive og ferdiggjort, og til slutt er det jobba med tyristorkrinsen og testing av denne. Siste del kom ein ikkje heilt i mål med, og det manglar testing under denne. Det har undervegs oppstått faglige og tekniske utfordringar, men ikkje meir enn det som lyt medreknast. Andre uforutsette problem har det vore minimalt med. Ei utfordring har vore at prosjektet starta som eit gruppearbeid på 3 studentar og over midtvegs vart gruppa splitta. Det kan nok medreknast som ein forklarande del til kvifor ein ikkje kom heilt i mål med oppgåva.

Målet med oppgåva har vore å bygga ein tyristoromformar som ein kan styra tennvinkelen til, og skriva sjølve programmet til Arduino Uno. Saman med dette også å få ei god forståing og innsikt i korleis ein tyristoromformar er bygd opp og kva element ein lyt ta omsyn til. Inkludert kva berekningar og komponentval som er nødvendig. I tillegg oppnå ei djupare forståing av korleis Arduino og C/C++ språket verkar, og kunna bruka dette.

Programmet er komplett og virkar som det skal. Det er eit program som klarar å detektera nullgjennomgangane og justera når den sender ut pulstoga. I tillegg vert ynskt data vist i lcd display og ein kan også fjernstyra tennvinkelen ved hjelp av mobiltelefon. Dei nødvendige delane til omformaren er også testa og bygd. Det som manglar er å setja saman desse delane og få testa som ein fullbølge omformar. Halvbølge likerettar er verfifisert at verkar, saman med firkantpulkrins og programkode.

Kapittel 10

Attståande arbeid og forslag til utvidingar

Grunna mangel på tid er ikkje alle delar fullført i dette prosjektet. Det attståande arbeidet i tillegg til utvidingar vert her presentert som forslag til vidare arbeid med prosjektet.

Det skulle vore lodda ferdig tyristorkrinsar og sett saman i ein fullstyrt, fullbølge likerettar. Dette skulle vore testa og sett kva avvik ein får i praksis, samanlikna med teorien og dei tapa som vert rekna ut. I tillegg er ein viktig del av desse testane å sjå på tennvinkelen og forsinkinga mellom faktisk nullgjennomgang og det programmet detekterer. Dette skulle vore gjort greie for, og deretter kalibrert i programkoden.

Det skulle også vore laga for å virka på trefase. Då lyt det loddast og koplast til to ekstra tyristorkrinsar, totalt seks. I tillegg trengs ein Arduino Mega, då det ikkje er nok timera på Arduino Uno. Med ein Mega har ein også moglegheit til å bruka ein ekstra timer til å justera frekvensen på pulstoga meir enn det ein kan med skaleringen.

Alle delane skulle også vore sett saman og montert i ein lukka boks. Med tilkopling for nettspenning inn, og kopling til last ut. Her kan også leggjast til krinsar for måling av straum og spenning ut. Desse verdiane kunne vore vist i LCD display. Det er også mange andre verdiar som kunne vore vist i LCD display, som effekt og spenning. Det kan då skrivast eit program saman med ein trykknapp, som vel kva verdi som skal visast.

Slik ESP32 er sett opp no, er det eit problem for studentar på lab å bruka. Mobiltelefon lyt vera kopla til same nettverk som ESP32, og med det internettet som er på skulen, vil det bli mykje styr å endra i programmet, då kvar student har eige brukarnamn og passord på nettet. For å koma rundt dette, kan ESP32 setjast opp med bluetooth i staden for. I tillegg er applikasjonen no tilgjengeleg gjennom applikasjonen Blynk. I staden for kan ein publisera ein eigen app, enten gjennom Blynk eller noko anna tilsvarande. Det kan også leggast til ekstra eller fleire val og innstillingar i appen, t.d. at ein kan velja kva spenning ein vil ha ut.

Bibliografi

[1] R. L. Boylestad og L. Nashelsky, *Electronic Devices and Circuit Theory*, 7th edition. Reading, Mass: Prentice Hall, 14. aug. 1998, 926 s., ISBN: 978-0-673-98053-3.

[2] N. Mohan, T. M. Undeland og W. P. Robbins, *Power Electronics: Converters, Applications, and Design*, 3rd edition. Hoboken, NJ: Wiley, 10. okt. 2002, 832 s., ISBN: 978-0-471-22693-2.

[3] "Arduino 101: Timers and Interrupts - Let's Make Robots / Tutorials," RobotShop Community. (7. aug. 2011), adresse: http://www.robotshop.com/community/forum/community/forum/t/arduino-101-timers-and-interrupts/13072 (sjekka 29.05.2022).

[4] "What is Schmitt Trigger | How It Works," How To Mechatronics. (24. aug. 2015), adresse: https://howtomechatronics.com/how-it-works/electrical-engineering/schmitt-trigger/ (sjekka 28.05.2022).

[5] G. Lakkas, "MOSFET power losses and how they affect power-supply efficiency," s. 7, 2016.

[6] Bimpelrekkie. "Answer to "Why Is the Feedback or Gate Resistance RG Necessary?"," Electrical Engineering Stack Exchange. (5. jul. 2018), adresse: https://electronics.stackexchange.com/a/383291 (sjekka 29.05.2022).

[7] S. Pefhany. "Answer to "Why Is the Feedback or Gate Resistance RG Necessary?"," Electrical Engineering Stack Exchange. (5. jul. 2018), adresse: https://electronics.stackexchange.com/a/383294 (sjekka 29.05.2022).

[8] Semicron, "Datablad Tyristor TYN20-800T," WeEn Semiconductors, 23. jul. 2018. adresse: https://no.mouser.com/datasheet/2/848/TYN20-800T-1846749.pdf.

[9] S. Projects. "Arduino 220V Full Wave Controlled Bridge Rectifier," Simple Projects. (18. jul. 2019), adresse: https://simple-circuit.com/arduino-220v-full-wave-controlled-bridge-rectifier/ (sjekka 21.05.2022).

[10] E. Laupsa, "Lys, Temperatur Og Trykkmåling Med Arduino," NTNU, 2020.

[11] "Current Mirror : Circuit, Working, Specifications & Its Limitations," ElProCus - Electronic Projects for Engineering Students. (3. apr. 2021), adresse: https://www.elprocus.com/current-mirror-circuit-working/ (sjekka 29.05.2022).

[12] E. Haustveit, "Pulstransformator-Forelesning-ELE119-Kraftelektronikk," Høgskulen på Vestlandet, Bergen, 16. sep. 2021, s. 97–100.

[13] "Pulse Transformer : Design, Types, Working and Its Applications," ElProCus - Electronic Projects for Engineering Students. (11. jul. 2021),

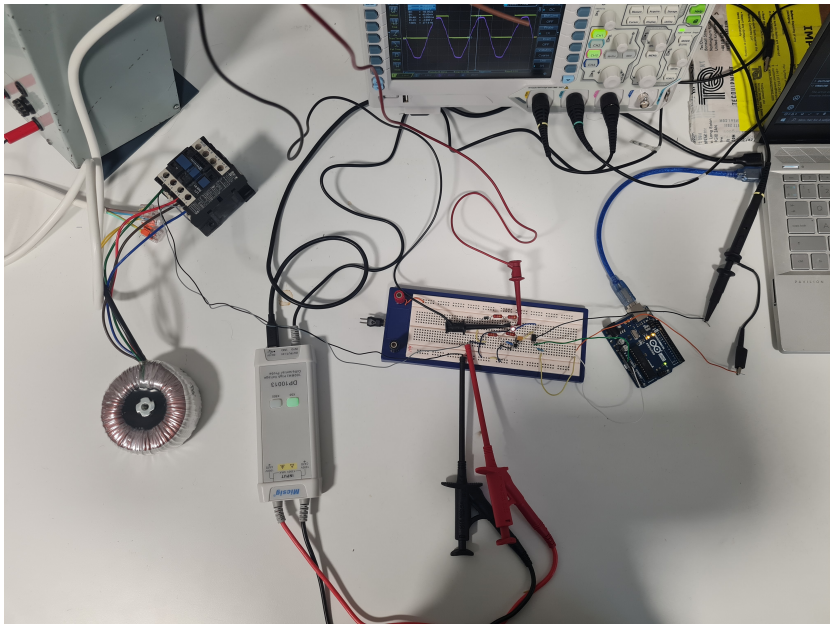
adresse: https://www.elprocus.com/pulse-transformer/ (sjekka 22.05.2022).
[14] *Arduino Uno*, i *Wikipedia*, 8. mai 2022. adresse: https://en.wikipedia.org/w/index.php?title=Arduino_Uno&oldid=1086736316 (sjekka 29.05.2022).
[15] *ESP32*, i *Wikipedia*, 6. mai 2022. adresse: https://en.wikipedia.org/w/index.php?title=ESP32&oldid=1086524212 (sjekka 29.05.2022).
[16] *Transformator*, i *Wikipedia*, 21. apr. 2022. adresse: https://no.wikipedia.org/w/index.php?title=Transformator&oldid=22510885 (sjekka 27.05.2022).
[17] "A Guide to Arduino & the I2C Protocol (Two Wire) | Arduino Documentation." (), adresse: https://docs.arduino.cc/learn/communication/wire (sjekka 29.05.2022).
[18] "Arduino - Interrupts." (), adresse: https://www.tutorialspoint.com/arduino/arduino_interrupts.htm (sjekka 27.05.2022).
[19] "Arduino Uno Rev3," Arduino Official Store. (), adresse: http://store.arduino.cc/products/arduino-uno-rev3 (sjekka 29.05.2022).
[20] "Atmega328p Datasheet," Atmel, Microchip. adresse: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.
[21] "attachInterrupt() - Arduino Reference." (), adresse: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/ (sjekka 06.02.2022).
[22] "Blynk." (), adresse: https://blynk.io (sjekka 29.05.2022).
[23] Broadcom, "HCPL-817: Phototransistor Optocoupler High-Density Mounting Type Data Sheet," s. 9, adresse: https://no.mouser.com/datasheet/2/678/AV02_0265EN_2021_04_26-2935641.pdf.
[24] "BZX55C8V2-TAP Vishay Semiconductors | Mouser," Mouser Electronics. (), adresse: https://no.mouser.com/ProductDetail/78-BZX55C8V2-TAP (sjekka 28.05.2022).
[25] "Chapter 11: The Current Mirror [Analog Devices Wiki]." (), adresse: https://wiki.analog.com/university/courses/electronics/text/chapter-11 (sjekka 29.05.2022).
[26] "Datablad-Kjøleribbe," AAVIDboyd. adresse: https://www.boydcorp.com/aavid-datasheets/Board-Level-Cooling-Low-Profile-Channel.pdf.
[27] "Datasheet-1003C," muRataPs. adresse: https://no.mouser.com/datasheet/2/281/1/kmp_1000-2936239.pdf.
[28] "Datasheet-Bc547," onsemi. adresse: https://no.mouser.com/datasheet/2/308/1/BC550_D-2310266.pdf.
[29] "Datasheet-Diode," REctron Semiconductor. adresse: https://no.mouser.com/datasheet/2/345/rl1n4001-rl1n4007-35989.pdf.
[30] "Datasheet-IRLZ34," International Rectifier. adresse: https://logosfoundation.org/instrum_gwr/tubo/irlz34n.pdf.
[31] "Datasheet-Kondensator," KEMET, Yageo company. adresse: https://api.kemet.com/component-edge/download/specsheet/C052K472K1X5CA.pdf.
[32] "Datasheet-SN74S132N," Texas Instruments. adresse: https://www.ti.com/lit/ds/sdls047/sdls047.pdf?HQS=dis-mous-null-mousermode-

dsf-pf-null-wwe&DCM=yes&ref_url=https%3A%2F%2Fno.mouser.com%2F&distId=26.
[33] E. Haustveit. "LC-Display — ELE102 Microcontroller Course v1.1.0 Documentation." (), adresse: https://gbthreepwood.github.io/ele102-elk-v2022/texts/Lessons/L8_LC_display.html (sjekka 24.05.2022).
[34] "HCPL-3120/J312, HCNW3120: 2.5 Amp Output Current IGBT Gate Drive Optocoupler Data Sheet," s. 29,
[35] "Internal Timers of Arduino," Arduino Project Hub. (), adresse: https://create.arduino.cc/projecthub/Marcazzan_M/internal-timers-of-arduino-58f6c9 (sjekka 29.05.2022).
[36] "Interrupts basics," Arduino Project Hub. (), adresse: https://create.arduino.cc/projecthub/rafitc/interrupts-basics-f475d5 (sjekka 04.05.2022).
[37] "Interrupts basics," Arduino Project Hub. (), adresse: https://create.arduino.cc/projecthub/rafitc/interrupts-basics-f475d5 (sjekka 27.05.2022).
[38] "LiquidCrystal - Arduino Reference." (), adresse: https://www.arduino.cc/reference/en/libraries/liquidcrystal/ (sjekka 26.05.2022).
[39] E. Mouse, "Datasheet-IT370," Schaffner. adresse: https://no.mouser.com/datasheet/2/355/IT_Series_single-2488601.pdf.
[40] D. U. Nicolai, "Application Note- AN 18-002, Thyristor Triggering and Protectionof Diodes and Thyristors." adresse: file:///C:/Users/elila/Downloads/SEMIKRON_Application-Note_Thyristor_Triggering_and_Protection_of_Diodes_and_Thyristors_EN_2018-11-19_Rev-02%20(2).pdf.
[41] "Om Høgskulen på Vestlandet." (), adresse: https://www.hvl.no/om/ (sjekka 28.05.2022).
[42] Semikron, "SKKH26-datasheet," Semikron. adresse: https://www.farnell.com/datasheets/16397.pdf.
[43] "Set Up Datastreams." (), adresse: https://docs.blynk.io/en/getting-started/template-quick-setup/set-up-datastreams (sjekka 29.05.2022).
[44] "WCAP-47," Wurth Electronic. adresse: https://no.mouser.com/datasheet/2/445/860010772008-1725435.pdf.
[45] "What Is an Optocoupler and How It Works." (), adresse: https://www.jameco.com/Jameco/workshop/Howitworks/what-is-an-optocoupler-and-how-it-works.html (sjekka 28.05.2022).
[46] "Widgets (App)." (), adresse: https://docs.blynk.io/en/blynk.apps/widgets-app (sjekka 29.05.2022).
[47] "Datasheet-IR2110," International Rectifier, 6.03.19. adresse: https://www.infineon.com/dgdl/Infineon-IR2110-DataSheet-v01_00-EN.pdf?fileId=5546d462533600a4015355c80333167e.
[48] "Datasheet-Irf520," Vishay Siliconix, 1.01.22. adresse: https://www.vishay.com/docs/91017/irf520.pdf.
[49] E. Haustveit, "Tyristor Og Tyristoromformar, Trigging Av Tyristoren," Høgskulen på Vestlandet, 16.09.21.
[50] E. Mouser, "Datasheet-Zenerdiode," Vishay semiconductors, 9.07.21. adresse: https://no.mouser.com/datasheet/2/427/bzx55-1767760.pdf.

Tillegg A

Oppkoplingar

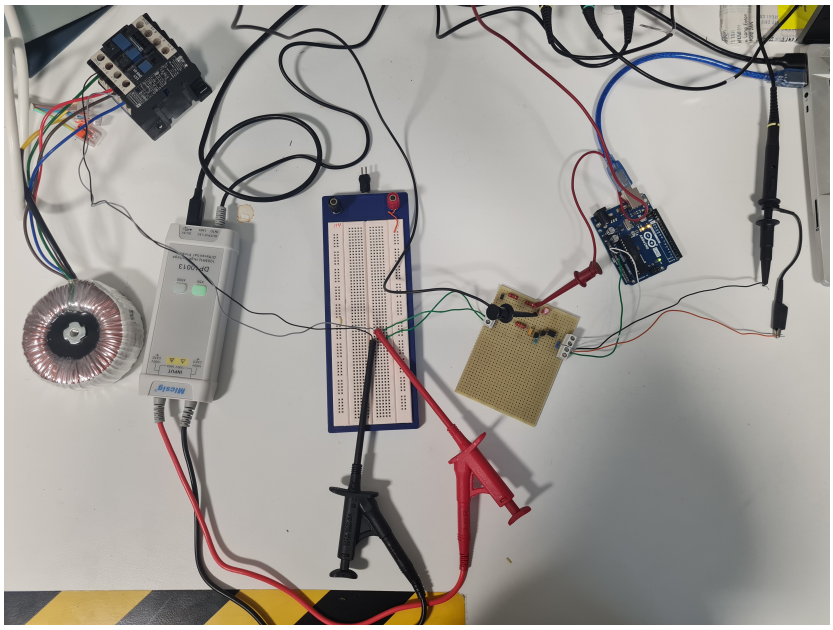
A.1 Breadboard



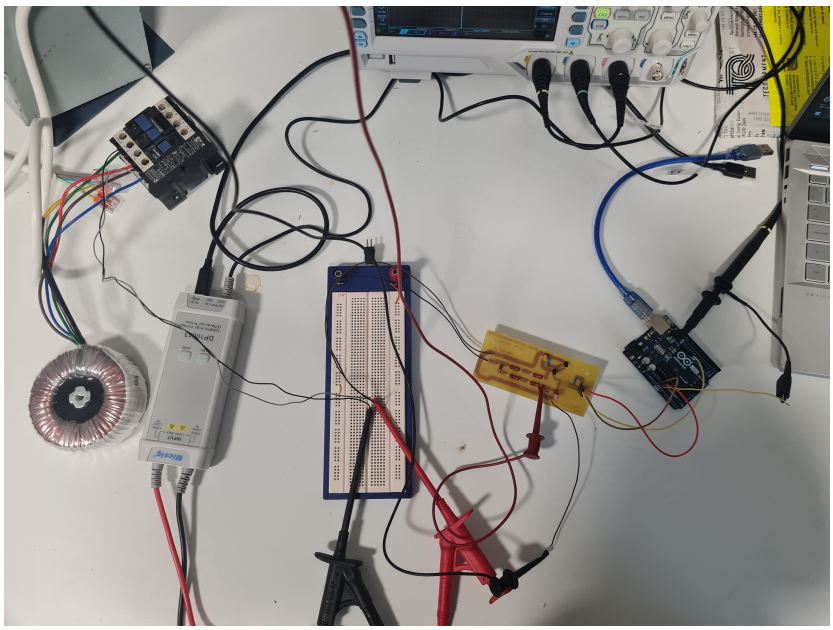
Figur A.1: Oppsett på Breadboard med 50V

A.2 Prototypekort (prefboard)

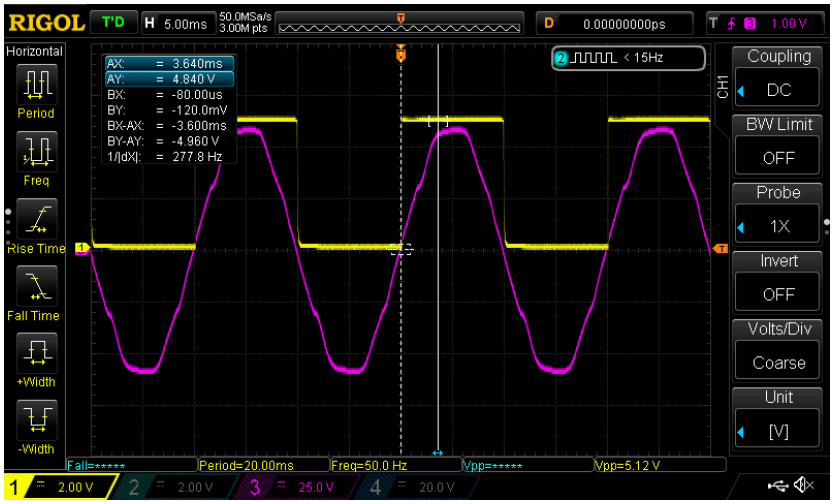
A.3 Etsa krinskort



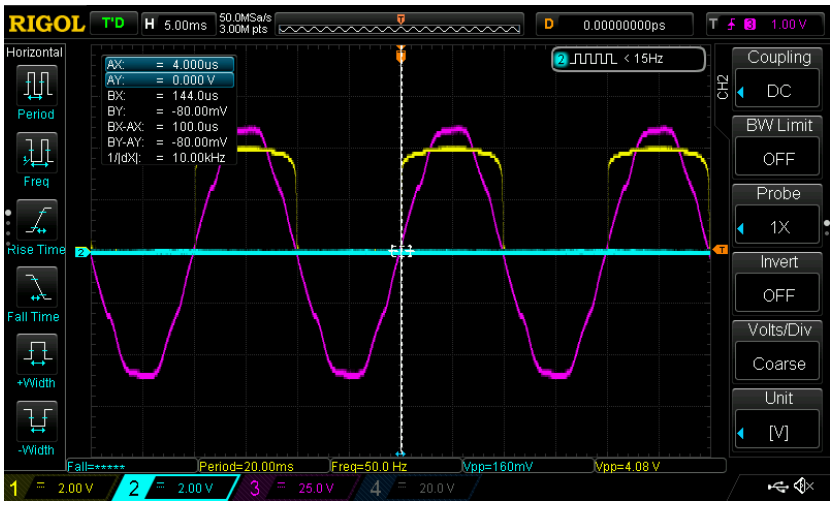
Figur A.2: Oppsett med det lodda kortet



Figur A.3: Oppsett med det etsa krinskortet



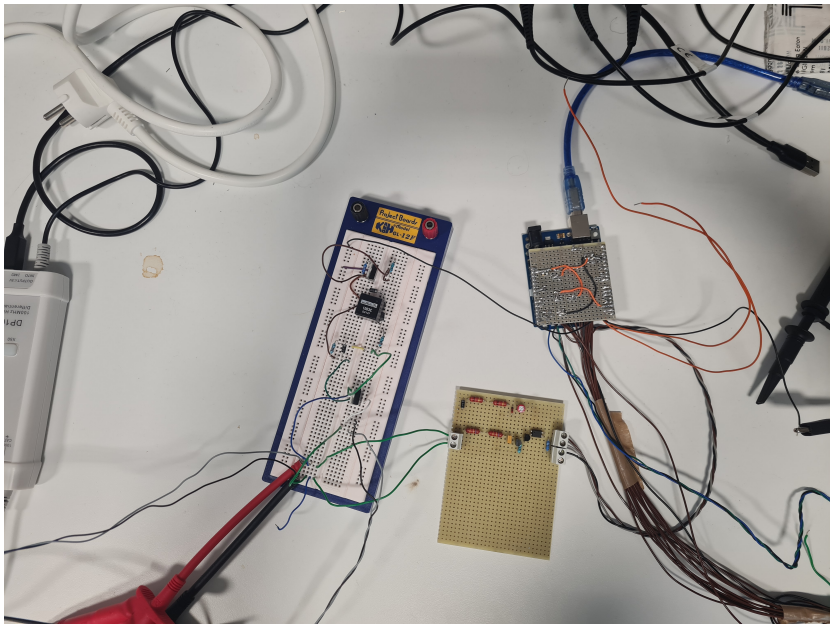
Figur A.4: Firkantpulssignalet, den gule grafen, med Schmitt trigger



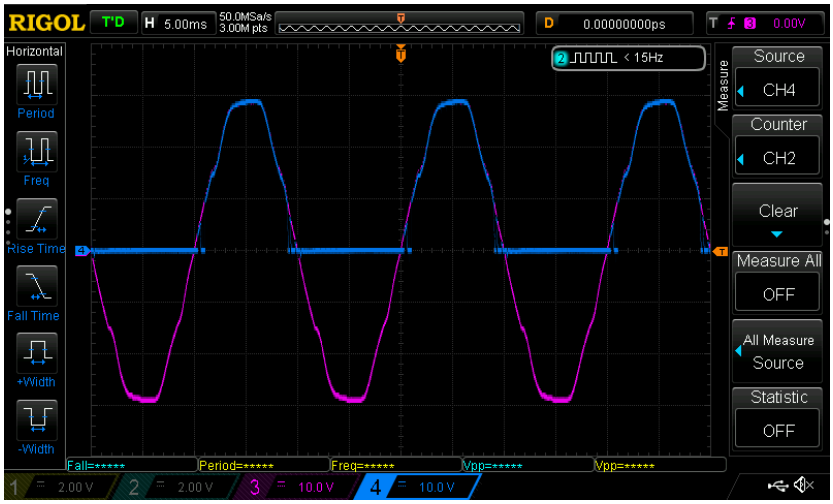
Figur A.5: Firkantpulssignalet, den gule grafen, utan Schmitt trigger

Tillegg B

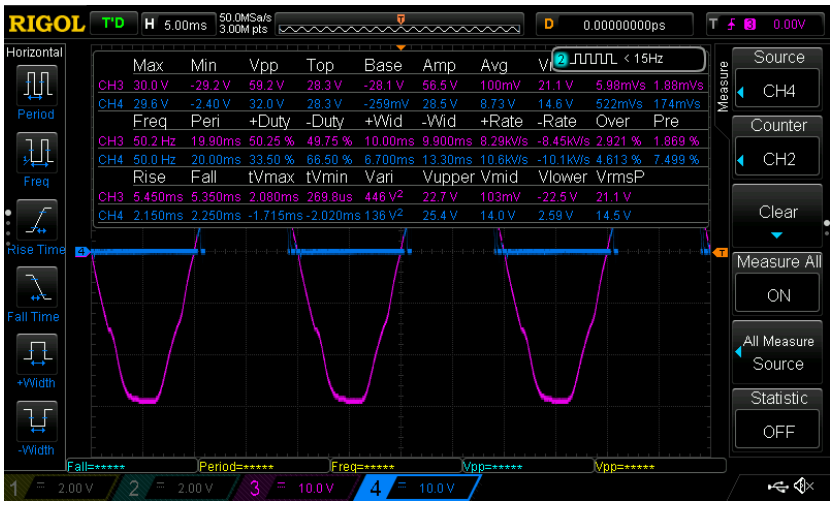
Tyristorkrins



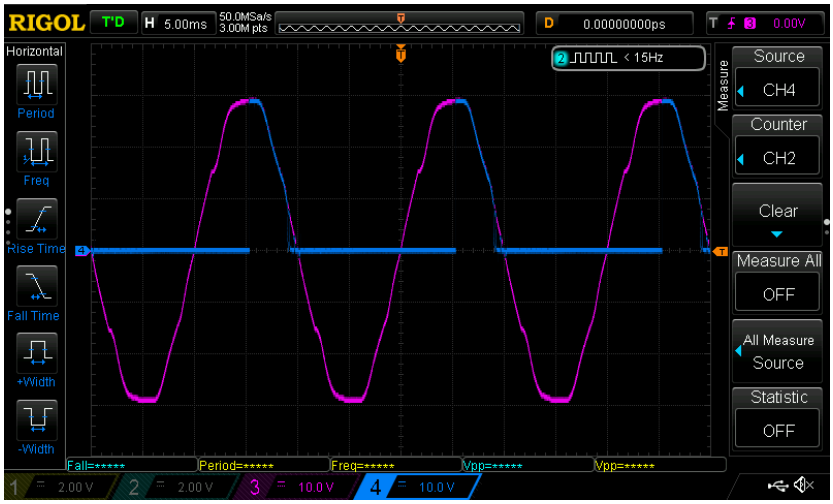
Figur B.1: Oppsett testing av tyristorkrins



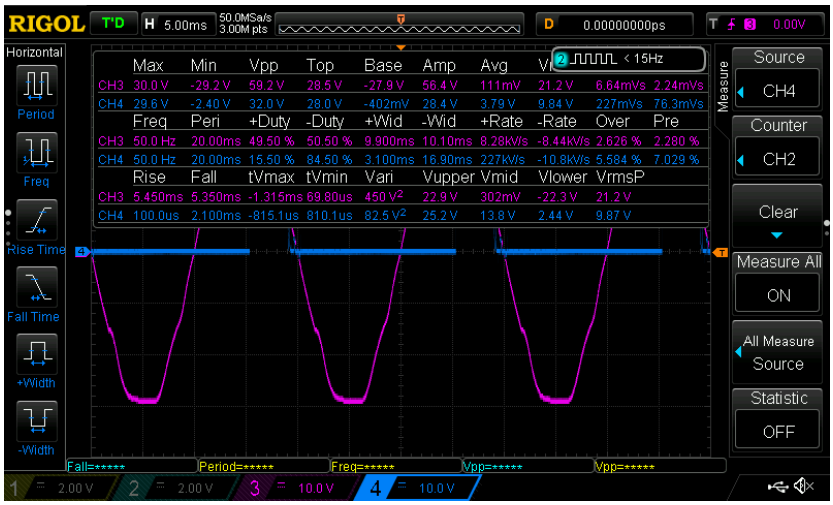
Figur B.2: Testing med $\alpha = 0$



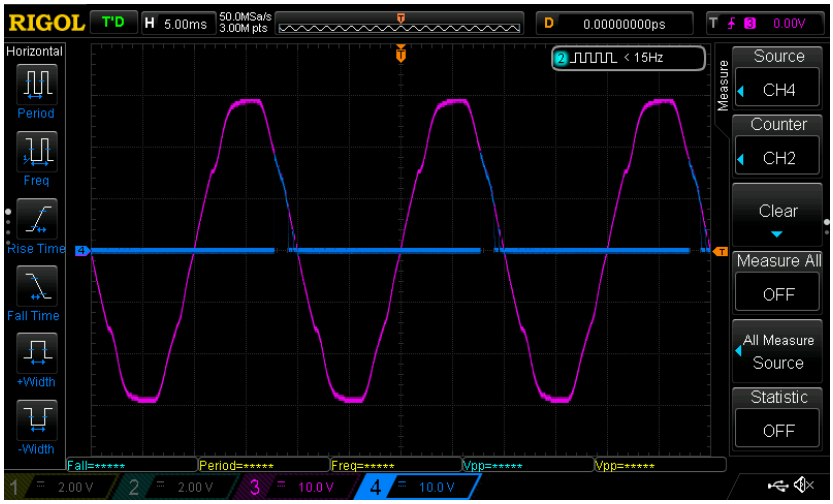
Figur B.3: Verdiar frå oscilloskop med $\alpha = 0$



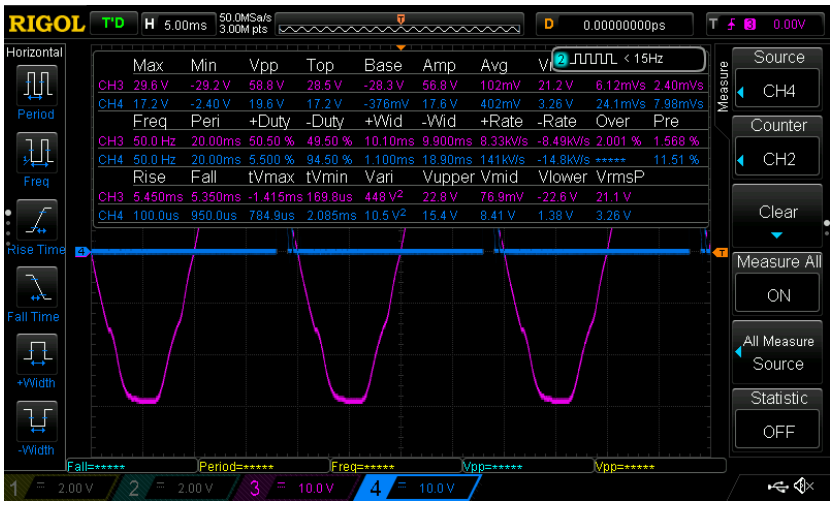
Figur B.4: Testing med $\alpha = 90$



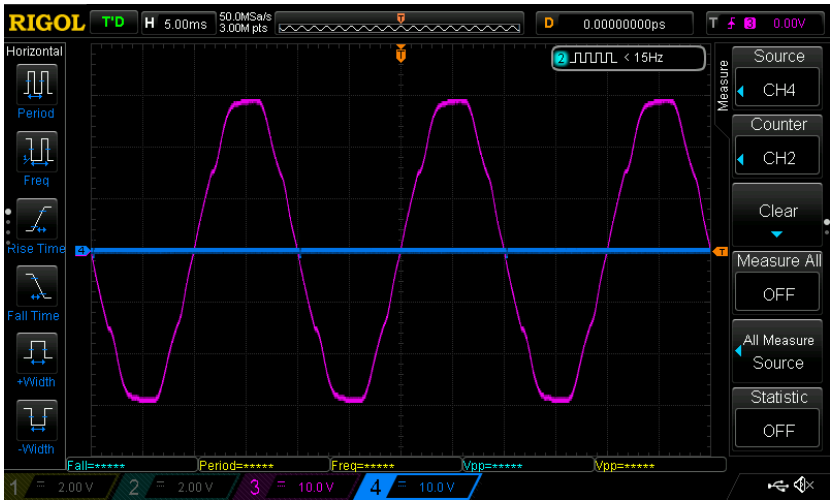
Figur B.5: Verdiar frå oscilloskop med $\alpha = 90$



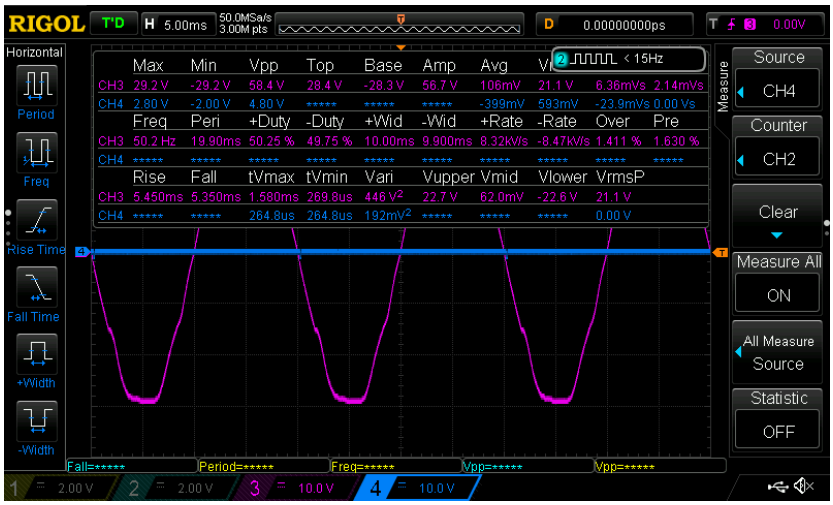
Figur B.6: Testing med $\alpha = 135$



Figur B.7: Verdiar frå oscilloskop med $\alpha = 135$



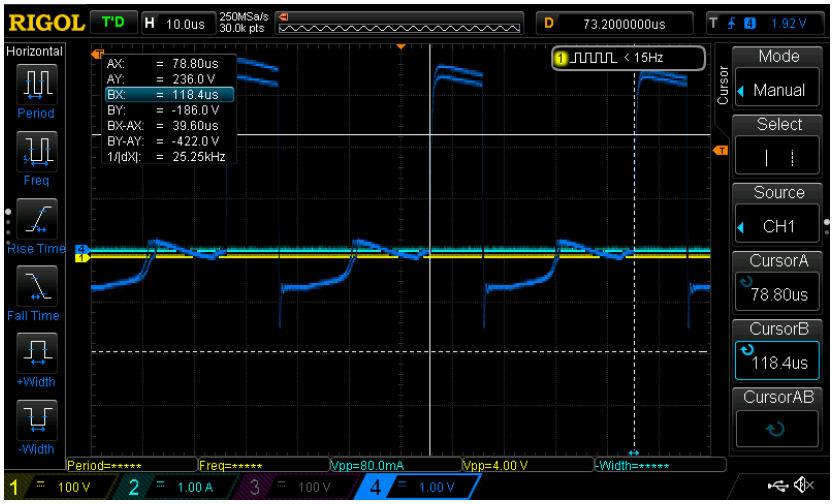
Figur B.8: Testing med $\alpha = 180$



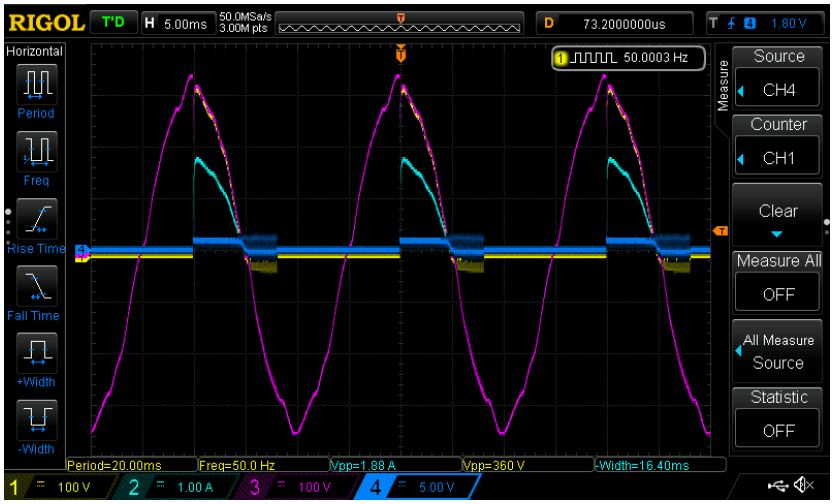
Figur B.9: Verdiar frå oscilloskop med $\alpha = 180$

Tillegg C

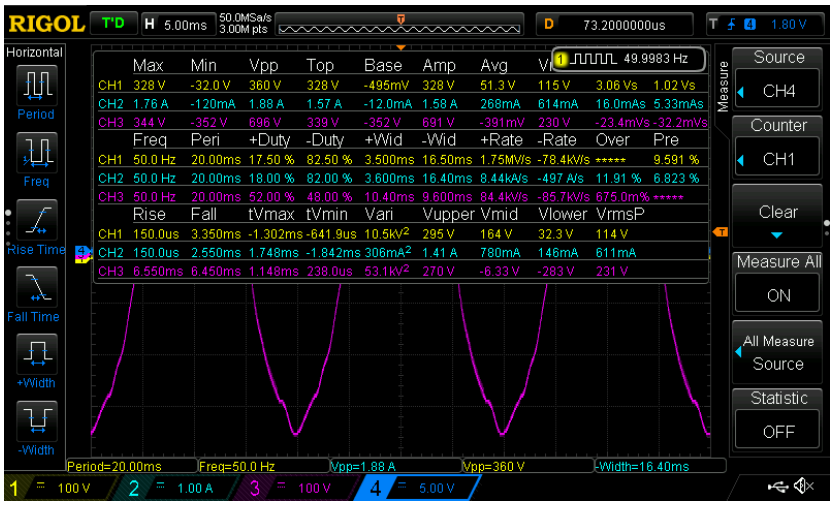
Målingar på gammal omformar



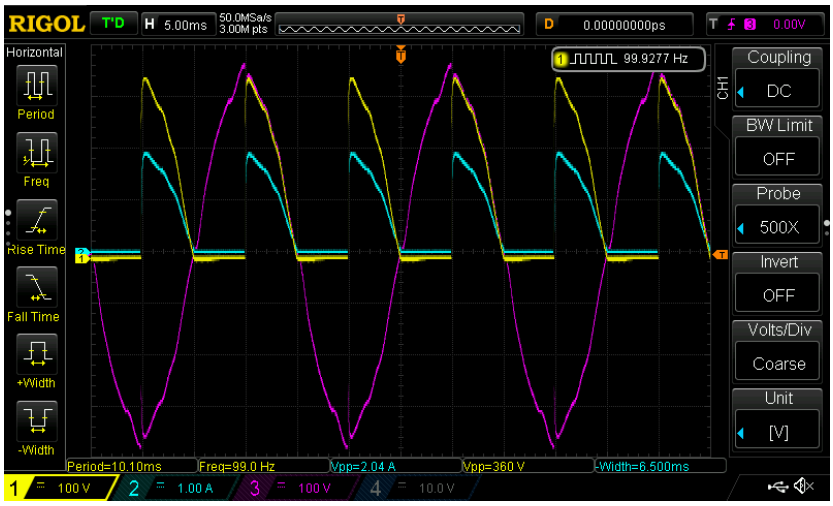
Figur C.1: Frekvens til pulstog på den gamle omformaren



Figur C.2: Halvbølgetest på gamal omformar

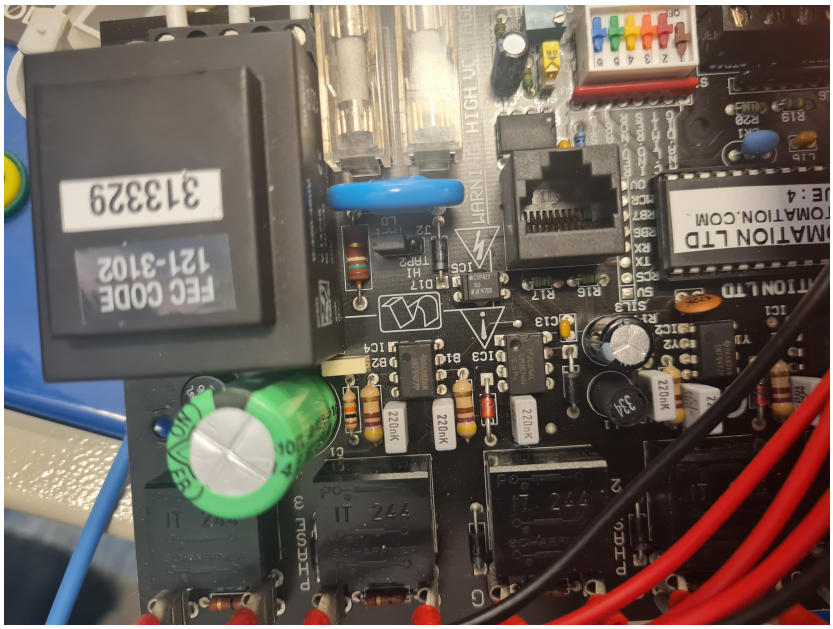


Figur C.3: Verdiar under halvbølgetest av gamal omformar



Figur C.4: Fullbølgetest på gamal omformar

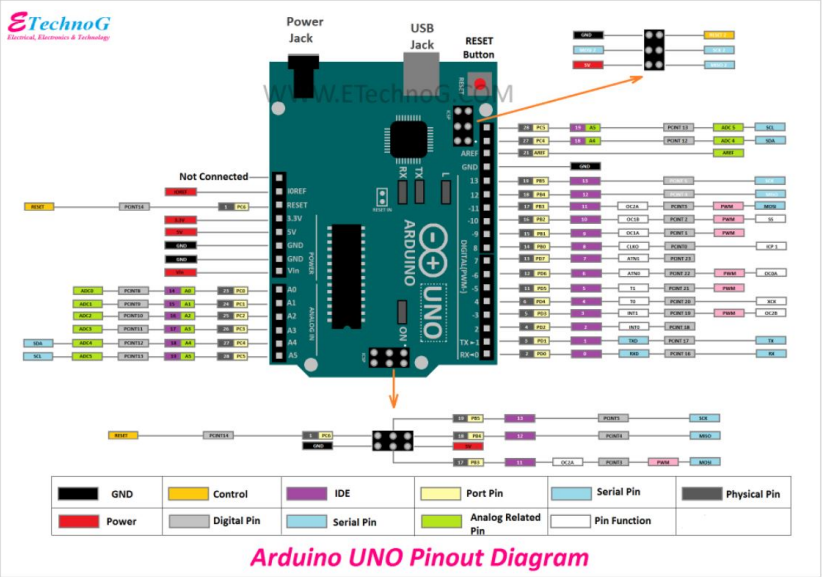
Figur C.5: Display som viser data på den gamle omformaren. Bilete viser korleis ein kalibrerer nullgjennomgangen for synkronisering med pulstog.



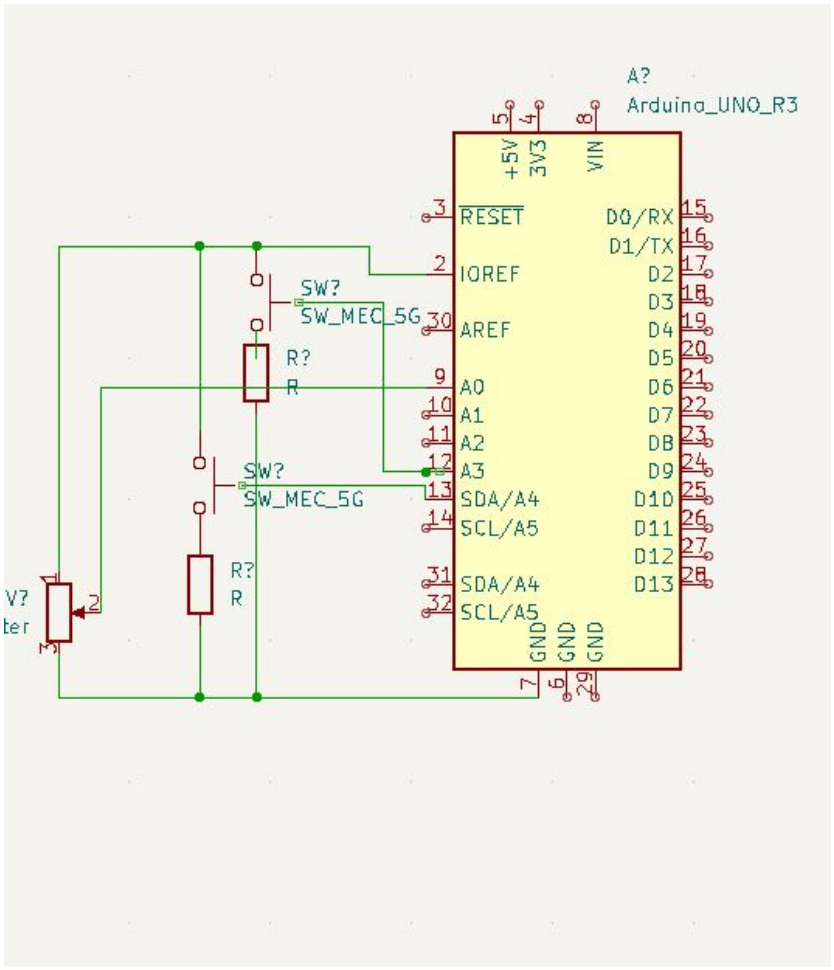
Figur C.6: Bilete av krinsen i den gamle omformaren. Eit forsøk på å forstå korleis denne verk i forhold til den som er bygd i denne oppgåva.

Tillegg D

Mikrokontrollarar og programkode

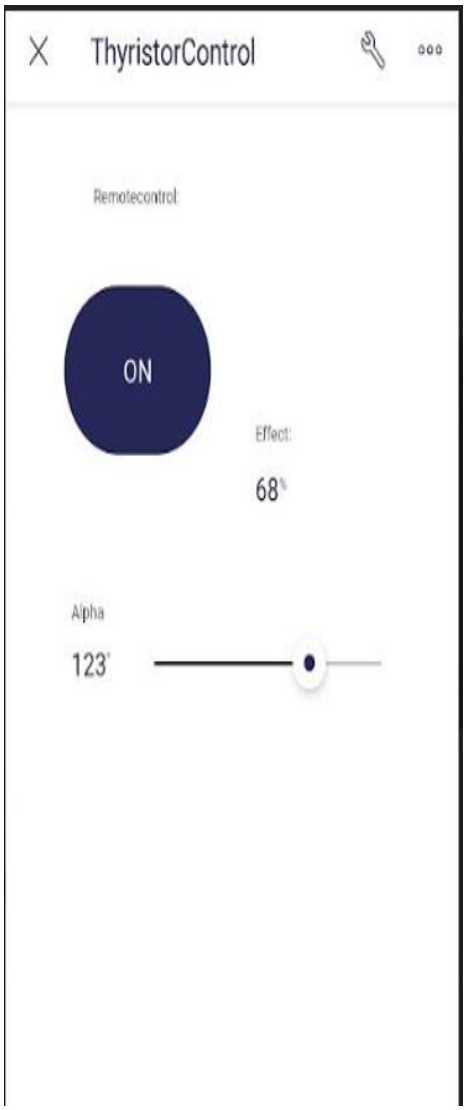


Figur D.1: Arduino Uno pinout oversikt



Figur D.2: Kopling av knappar og potmeter til Arduino Uno

Figur D.3: Skjermdump frå mobiltelefon i Blynk applikasjonen



Figur D.4: Skjermdump frå Blynk-app som viser widgets.

Tillegg E

Programdelar:

Listing E.1: kode for korrekt alfaverdi

```
    if ((currentTimeStamp_ms - prevTimeStamplc_ms) >=
        intervallc_ms){
        potmeter_adc_raw = analogRead(pot_pin);
        alpha_from_pot = ((potmeter_adc_raw*180)/1023);
        //alpha_from_ESP = x;
        if (c == 'R'){
            alpha_degrees = alpha_from_ESP;
            mode = 1;
        }
        else if (c == 'L'){
            alpha_degrees = alpha_from_pot;
            mode = 0;
        }

        //alpha_from_pot = ((potmeter_adc_raw*180)/1023);
        //alpha_degrees = ((potmeter_adc_raw*180)/1023);
        alpha_percentage = 100 - ((alpha_degrees*100)/180);
        timer1setalpha(alpha_degrees, grid_freq_setting_hz);
        lc(alpha_degrees, alpha_percentage);
        prevTimeStamplc_ms = currentTimeStamp_ms;
    }
```

Listing E.2: lc-funksjon

```
void lc(uint16_t alpha_degrees,uint16_t alpha_percentage){
lcd.clear();
lcd.createChar(1, alpha);
lcd.setCursor(0,0);
lcd.write(byte(1));
lcd.print(" = ");
lcd.print(alpha_degrees);
lcd.print("\xDF");
lcd.setCursor(0,1);
lcd.print(alpha_percentage);
lcd.print(" %");
```

```
12  lcd.setCursor(12,0);
13  if (start){
14      lcd.print("ON");}
15  else{
16      lcd.print("OFF");
17  }
18  lcd.setCursor(10, 1);
19  if (mode){
20      lcd.print("Remote");
21  }
22  else{
23      lcd.print("Local");
24  }
25  }
```

Listing E.3: Eige oppsett av alfa

```
1       uint8_t alpha[] = {0x00, 0x09, 0x15, 0x12, 0x12, 0x0D, 0
            x00, 0x00};
```

Listing E.4: pulse-check

```
1   bool is_pulse12_on(){
2   if (((TCCR2A >> COM2A0) & 0x01) && ((TCCR2A >> COM2A1) & 0
        x01)){
3   return true;
4   }
5   else{
6   return false;
7   }
8   }
9
10  bool is_pulse34_on(){
11  if (((TCCR2A >> COM2B1) & 0x01) && ((TCCR2A >> COM2B0) & 0
        x01)){
12  return true;
13  }
14  else{
15  return false;
16  }
17  }
```

Tillegg F

Kode:

F.1 Thyristorcontrol main:

Listing F.1: main program

```
1
2  #include <Arduino.h>
3  #include "isr.h"
4  #include "tyristorlib.h"
5  //For å velgja om ein skal ha pwm-signal eller firkantpuls p
       å utgangane.
6  //Mest for testing og feilsøking, så er det lettare å setja
       utgangar høg istadenfor pulstog
7  #define SQUARE_WAVE 0
8
9  int start;
10 //int stopp;
11 //int flagg12;
12 //int flagg34;
13 static uint16_t period_count = 0;
14 uint8_t zc_watchdog_counter = 0;
15 uint16_t compA_counter = 0;
16
17
18 /*Set opp avbrotsfunksjon. Denne skal registrera eksternt
       avbrot for negativ flanke
19 på pinne 2. */
20 void interruptSetup(){
21     EICRA = (1 << ISC01) | (0 << ISC00);
22     EIMSK = (1 << INT0);
23 }
24 /*Denne funksjonen køyrer for kvart avbrot. ZC_watchdog er
       lagt innf or å registrera
25 dersom det ikkje kjem inn firkantpulssignal*/
26 ISR(INT0_vect){
27     zc_watchdog_counter = 0;
28     compA_counter = 0;
```

```
29      if (start){                 //Dersom start modus, viss
            ikkje skjer ingenting
30          period_count = TCNT1; // Store counter value in
                order to calculate frequency
31          TCNT1 = 0;              //Nullstiller timer, startar
                telja frå null
32
33          OCR1A = get_comp_alpha();   //Set OCR1A til verdien
                som er rekna ut frå alfa frå potmeter/ESP
34          OCR1B = get_comp_alpha() + get_comp_180(); //OCR1B
                er det same, ein halvperiode etter
35
36          #if defined(SQUARE_WAVE) && (SQUARE_WAVE == 0) //Set
                 enten pinne låg eller pulstog av.
37          T34pulseoff();
38          #else
39          PORTD &= ~(1 << 3); // Pin 3 low
40          #endif
41      }
42      else {
43      }
44  }
45
46  /*Denne funksjonen vil køyra kvar gong OCR1B = TCNT1. Det
        vil seia kvar gong ein får ...
47  */
48
49  ISR(TIMER1_COMPB_vect){
50
51      #if defined(SQUARE_WAVE) && (SQUARE_WAVE == 0)
52      T34pulseon();//Set pinne3 høge eller pulstog på.
53
54      #else
55      PORTD |= (1 << 3); // Pin 3 high
56      #endif
57  }
58
59  ISR(TIMER1_COMPA_vect){
60      #if defined(SQUARE_WAVE) && (SQUARE_WAVE == 0)
61      if (compA_counter > 0){   //Dersom compA_counter =< , så
             har ein negativ flanke og pulstoget skal vera av
62          T12pulseoff();
63      }
64      else{                   //Her er positiv flanke
65          compA_counter++;      //For å kunna skru av ved
                negativ flanke
66          T12pulseon();         //pulstoget skal på her ved
                positiv flanke
67          // Oppdaterer OCR1A registeret
68          // så timer kan skru av pulstoget
69          // etter ein full syklus
70          OCR1A = get_comp_180(); //20010;// - temp + TCNT1;
71      }
72
```

```
73      //Her skjer det same berre set utgangar høg og låg
74      #else
75      if (compA_counter > 0){
76          PORTB &= ~(1 << 3); // Pin 11 low
77      }
78      else{
79          compA_counter++;
80          PORTB |= (1 << 3); // Pin 11 high
81          OCR1A = get_comp_180(); //20010;// - temp + TCNT1;
82      }
83      #endif
84  }
```

F.2 Tyristorlib:

Listing F.2: Bibliotek for timerfunksjonar

```
1   #include <Arduino.h>
2   #include "tyristorlib.h"
3
4   uint16_t comp_180 = 0;      //maksimum verdi av alfa
5   uint16_t comp_alpha = 0;    //Faktisk verdi av alfa
6
7   /*Funksjonar som skrur av og på timer1. */
8   void timer1on(){
9       TCNT1 = 0;
10      TIMSK1 |= (1<<OCIE1A) | (1<<OCIE1B);
11  }
12
13  void timer1off(){
14      TCNT1 = 0;
15      TIMSK1 &= ~((1<<OCIE1A)| (1<<OCIE1B));
16  }
17
18  /*Funksjonar som skrur av og på pulstog på ulike pinnar. T12
        er pinne 11 og T34 er pinne 3. I tilegg
19  funksjonar for å sjekka om pulstoga er på eller ikkje. Dei
        gjev verdi true for på og
20  false for av. Desse verdiane blir brukt i compA vektor*/
21  void T12pulseon() {
22      TCNT2 = 0;
23      TCCR2A |=  (1<<COM2A0) | (1<<COM2A1);
24      TCCR2B |= ((0<<CS22) | (0<<CS21)|(1<<CS20));
25  }
26
27  void T12pulseoff(){
28      TCCR2A &= ~( (1<<COM2A0) | (1<<COM2A1));
29      TCCR2B &= ~((1<<CS22) | (1<<CS21)|(1<<CS20));
30  }
31
32  bool is_pulse12_on(){
33      if (((TCCR2A >> COM2A0) & 0x01) && ((TCCR2A >> COM2A1) &
            0x01)){
```

```
34          return true;
35      }
36      else{
37          return false;
38      }
39  }
40
41  bool is_pulse34_on(){
42      if (((TCCR2A >> COM2B1) & 0x01) && ((TCCR2A >> COM2B0) &
            0x01)){
43          return true;
44      }
45      else{
46          return false;
47      }
48  }
49
50  void T34pulseon(){
51      TCNT2= 0;
52      TCCR2A |= (1<<COM2B1 ) | (1<<COM2B0);
53      TCCR2B |= ((0<<CS22) | (0<<CS21)|(1<<CS20));
54  }
55
56  void T34pulseoff(){
57      TCCR2A &= ~((1<<COM2B1 ) | (1<<COM2B0));
58      TCCR2B &= ~((1<<CS22) | (1<<CS21)|(1<<CS20));
59  }
60
61  /*Innstilling/Opsett for timerane. Timer2 skal gje ut
        pulstog, pwm signal.
62  OCR1A og B bestemmer arbeidssyklus. Timer1 skal berre telja
        vanleg oppover.*/
63  void timer2setup(){
64      TCCR2A = _BV(COM2A0)|/*_BV(COM2A1)|*/_BV(WGM22) |/* _BV(
            WGM21) |*/ _BV(WGM20);
65      TCCR2B = _BV(CS22)|  _BV(CS21) | _BV(CS20);
66      OCR2A = 130; //pin 11
67      OCR2B = 130; //pin 3
68  }
69
70  void timer1setup(){
71      TCCR1A =0;
72      TCCR1B = (0 << WGM12) | (0 << CS12) | (1 << CS11) | (0
            << CS10);
73  }
74
75
76  // Alpha som vert henta her, er frå 0 til 180 grader.
77
78  void timer1setalpha(uint16_t alpha, uint8_t grid_freq){
79
80      comp_180 = (16000000/(8*2*grid_freq));  //Reknar ut alfa
            for 180 grader etter kva frekvens som er vald
81      uint32_t x = ((180*1000000)/comp_180);  //
```

```
82
83      if (alpha > 178){ //For å unngå glitching på pulstog,
            vert ustabilt på 180,blir aldri < 178
84          alpha = 178;
85      }
86
87      comp_alpha = ((alpha*1000000)/x); //Faktisk alfaverdi fr
            å potmeter/ESP rekna om til OCRxy-verdi til timer
88
89      if(0 == comp_alpha){  //Alfa kan heller ikkje verta 0,då
             startar aldri timer og det vert ustabilt
90          comp_alpha = 1;
91      }
92
93      if (comp_alpha > (comp_180 - 20)){ //
94          comp_alpha -= 20;
95      }
96
97  }
98
99  //Funksjonar for å henta oppdaterte alfaverdiar til
        mainprogrammet. OCRxy verdiar.
100
101 uint16_t get_comp_alpha(){
102     return comp_alpha;
103 }
104
105 uint16_t get_comp_180(){
106     return comp_180;
107 }
```

F.3 ISRlib:

Listing F.3: Bibliotek for ISRfunksjonar

```
1       #include <Arduino.h>
2       #include "isr.h"
3       #include "tyristorlib.h"
4       //For å velgja om ein skal ha pwm-signal eller
            firkantpuls på utgangane.
5       //Mest for testing og feilsøking, så er det lettare å
            setja utgangar høg istadenfor pulstog
6       #define SQUARE_WAVE 0
7
8       int start;
9       //int stopp;
10      //int flagg12;
11      //int flagg34;
12      static uint16_t period_count = 0;
13      uint8_t zc_watchdog_counter = 0;
14      uint16_t compA_counter = 0;
15
16
```

```
17      /*Set opp avbrotsfunksjon. Denne skal registrera
            eksternt avbrot for negativ flanke
18      på pinne 2. */
19      void interruptSetup(){
20          EICRA = (1 << ISC01) | (0 << ISC00);
21          EIMSK = (1 << INT0);
22      }
23      /*Denne funksjonen køyrer for kvart avbrot. ZC_watchdog
            er lagt innf or å registrera
24      dersom det ikkje kjem inn firkantpulssignal*/
25      ISR(INT0_vect){
26          zc_watchdog_counter = 0;
27          compA_counter = 0;
28          if (start){                 //Dersom start modus, viss
                ikkje skjer ingenting
29              period_count = TCNT1; // Store counter value in
                    order to calculate frequency
30              TCNT1 = 0;              //Nullstiller timer,
                    startar telja frå null
31
32              OCR1A = get_comp_alpha();   //Set OCR1A til
                    verdien som er rekna ut frå alfa frå potmeter
                    /ESP
33              OCR1B = get_comp_alpha() + get_comp_180(); //
                    OCR1B er det same, ein halvperiode etter
34
35              #if defined(SQUARE_WAVE) && (SQUARE_WAVE == 0)
                    //Set enten pinne låg eller pulstog av.
36              T34pulseoff();
37              #else
38              PORTD &= ~(1 << 3); // Pin 3 low
39              #endif
40          }
41          else {
42          }
43      }
44
45      /*Denne funksjonen vil køyra kvar gong OCR1B = TCNT1.
            Det vil seia kvar gong ein får ...
46      */
47
48      ISR(TIMER1_COMPB_vect){
49
50          #if defined(SQUARE_WAVE) && (SQUARE_WAVE == 0)
51          T34pulseon();//Set pinne3 høge eller pulstog på.
52
53          #else
54          PORTD |= (1 << 3); // Pin 3 high
55          #endif
56      }
57
58      ISR(TIMER1_COMPA_vect){
59          #if defined(SQUARE_WAVE) && (SQUARE_WAVE == 0)
```

```
60          if (compA_counter > 0){   //Dersom compA_counter =<
                , så har ein negativ flanke og pulstoget skal
                vera av
61              T12pulseoff();
62          }
63          else{                   //Her er positiv flanke
64              compA_counter++;      //For å kunna skru av ved
                    negativ flanke
65              T12pulseon();         //pulstoget skal på her
                    ved positiv flanke
66              // Oppdaterer OCR1A registeret
67              // så timer kan skru av pulstoget
68              // etter ein full syklus
69              OCR1A = get_comp_180(); //20010;// - temp +
                    TCNT1;
70          }
71
72          //Her skjer det same berre set utgangar høg og låg
73          #else
74          if (compA_counter > 0){
75              PORTB &= ~(1 << 3); // Pin 11 low
76          }
77          else{
78              compA_counter++;
79              PORTB |= (1 << 3); // Pin 11 high
80              OCR1A = get_comp_180(); //20010;// - temp +
                    TCNT1;
81          }
82          #endif
83      }
```

F.4 Blynk og ESP32 kode:

Listing F.4: Kode for ESP32

```
1       #define BLYNK_TEMPLATE_ID "TMPLkmvDl3Rk"
2       #define BLYNK_DEVICE_NAME "ThyristorControl"
3       #define BLYNK_AUTH_TOKEN "EzGU43EoA-pAcudir3sa8-AyjagU-8
            S0"
4
5       #include <WiFi.h>
6       #include <WiFiClient.h>
7       #include <BlynkSimpleEsp32.h>
8       #include <Arduino.h>
9       #include <Wire.h>
10
11
12      char auth[] = BLYNK_AUTH_TOKEN; //Dette hentast frå
            Blynk app eller nettside for oppretta template
13
14      char ssid[] = "Galaxy S20 FEA6EC";//Nettverks namn
15      char pass[] = "ooxq1367";//Netttverks passord
16
```

```
17      BlynkTimer timer;
18      uint32_t percentageBlynk;
19      uint32_t alphaBlynk;
20      int BlynkMode;
21      u_char mode;
22
23      //Hentar alfaverdi frå app, reknar ut i prosent og
            sender tilbake til app
24      BLYNK_WRITE(V1)
25      {
26          alphaBlynk = param.asInt();
27          percentageBlynk = (100 - ((alphaBlynk*1000)/1800));
28          Blynk.virtualWrite(V3, percentageBlynk);
29      }
30      //Les av knapp i applikasjon og vel modus deretter.
31      BLYNK_WRITE(V0){
32          BlynkMode = param.asInt();
33
34          if (BlynkMode == 1){
35              Serial.println("ON");
36              mode = 82;
37          }
38          else {
39              Serial.println("OFF");
40              mode = 76;
41          }
42      }
43      //Funksjon fir I2C, som kommuniserer med Arduino Uno
            slaven.
44      void mastersendtoslave(){
45          Wire.beginTransmission(127);
46          Wire.write(mode);
47          Wire.write(alphaBlynk);
48          Wire.endTransmission();
49          delay(100);
50      }
51
52
53      void setup()
54      {
55          Serial.begin(9600);
56          Blynk.begin(auth, ssid, pass);
57          Wire.begin();
58          pinMode(4, OUTPUT);
59      }
60
61      void loop()
62      {
63          Blynk.run(); //Opprettar kontakt med Blynk
64          mastersendtoslave(); //Køyrer funksjon
65      }
```