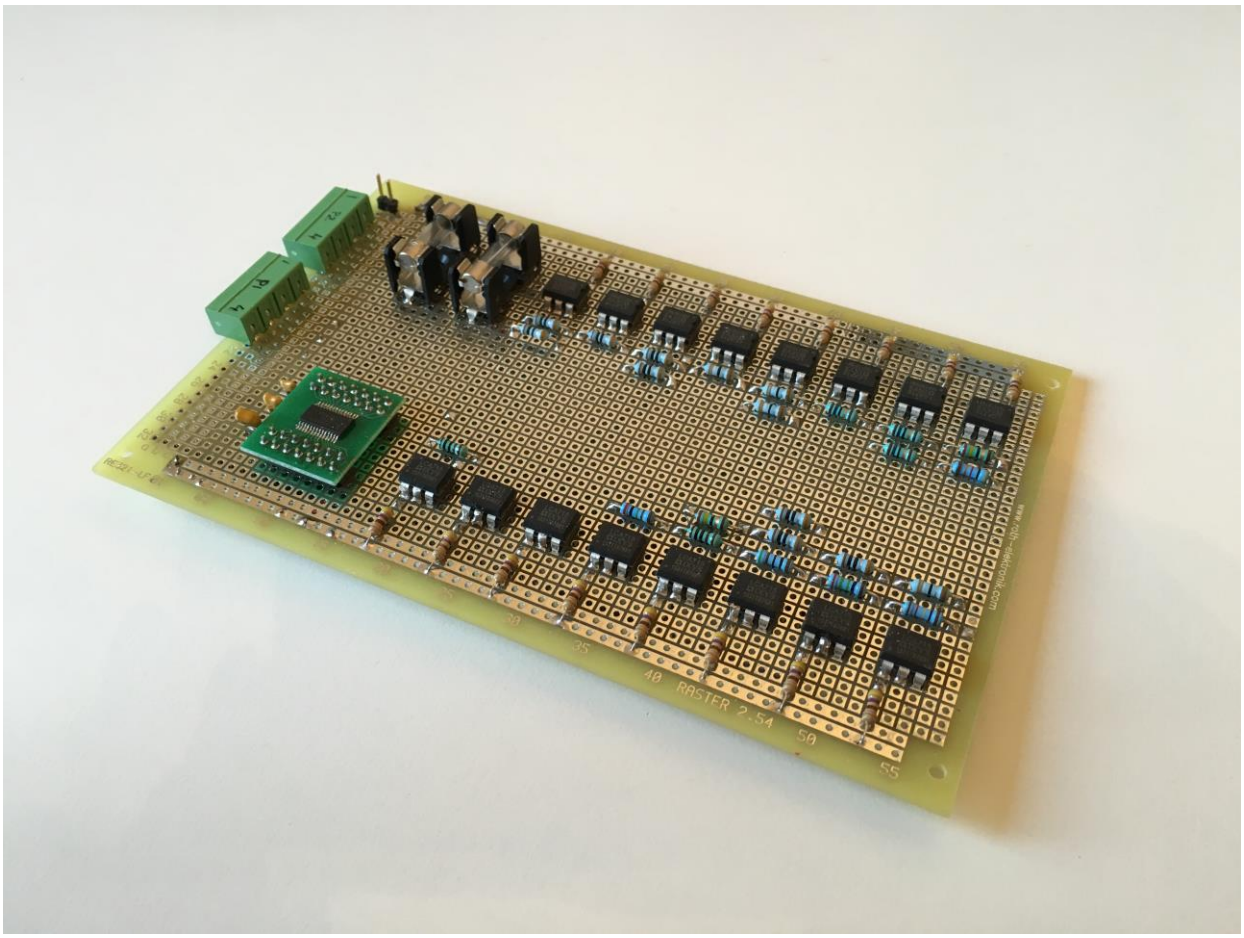
CHALMERS



Simulering av resistansförändring för testutrustning

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

HELENA BERGVALL
JONAS FRUCK

Institutionen för Signaler och system
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016

FÖRORD

Detta examensarbete är vår sista kurs innan examen efter tre härliga och givande år på Chalmers tekniska högskola. Kursen omfattar 15hp och ligger sist av en lång rad kurser där slutmålet är att erövra titeln mekatronikingenjör (180hp). Arbetet har utförts under institutionen för Signaler och system och vår examinator på Chalmers var Manne Stenberg.

Idén till detta examensarbete kom ifrån Aliaro och vi är tacksamma för deras förtroende. Bortsett från lite rapportskrivande i skolans lokaler så har arbetets tio veckor tillbringats på Aliaros kontor i Sörhallen.

Utöver vårt tack till företaget Aliaro vill vi särskilt tacka Roger som varit vår handledare på företaget. Utan Rogers initiativ att erbjuda oss ett examensarbete på Aliaro hade just detta arbete inte blivit av. Vi vill även rikta ett stort tack till Göran som utöver handledningen under vårt examensarbete även bidragit med mycket kunskap och stått ut med våra dumma frågor i flertalet kurser innan denna. Slutligen kommer även ett tack till caféet på hörnet med den underbara apelsin- och chokladglassen. Perfekt för att ladda batterierna!

Helena Bergvall och Jonas Fruck, Göteborg, juni 2016

SAMMANFATTNING

För att testa en nykonstruktion idag så används till stor del specifika testutrustningar. Dessa testutrustningar är ofta både kostnadsineffektiva och tidsineffektiva. På företaget Aliaro där uppgiften har utförts så håller man på att ta fram en generell testutrustning som enkelt ska kunna anpassas till nästa testobjekt. Detta för att spara in på tiden det tar att nykonstruera en testutrustning vilket därmed även minskar kostnaderna. Syftet med detta uppdrag var att utvärdera möjligheten att mjukvarumässigt kontrollera en resistansstege till att kunna efterlikna ett verkligt förlopp. För att kunna göra detta beslöts att dels testa hur snabbt man kunde generera en fyrkantsvåg och fortfarande få bra resultat för att uppskatta hur snabba förändringar som kretsen kan klara av. Vidare skulle det även skapas en sinusvåg för att kunna bedöma kvaliteten på resistansstegen för ett mer varierat förlopp över tid.

Resistansstegen som konstruerats för uppgiften bestod av 13 resistorer i serie som via optokopplare kunde förbikopplas. Optokopplarna gick att styras individuellt eller i grupper om fyra via ett LED drivdon. Mjukvaran som styrde optokopplarna utvecklades i LabVIEW och den implementerades i en så kallad cRIO. Vilken är en hårdvaruenhet som handhar bland annat analoga- och digitala in- och utgångar via olika moduler. Den har en inbyggd FPGA där koden exekverades. Drivdonet till optokopplarna styrdes från cRIO via den synkrona bussen I^2C. Högsta frekvensen för en acceptabel fyrkantsvåg var 519,8 Hz. Den enskilt största faktorn till att det inte gick att åstadkomma högre frekvenser låg hos optokopplarna. Deras omslagningshastighet var för långsam för snabbare resultat. Om man tänker gå vidare med utvecklingen av den framtagna resistansstegen för att implementera den i Aliaros testutrustning så bör man överväga att avsätta tid på att finna en snabbare optokopplare. Detta för att kunna förbättra prestandan på kretsen genom att göra den snabbare. Det gick att generera en sinusvåg, men på grund av olika förseningar så hade vågen alltid korta dippar eller toppar på de ställen där optokopplarna bytte värden i en olycklig kombination. Denna karaktäristik gick inte att komma ifrån oavsett hur låg frekvens som användes vid genereringen av vågen.

ABSTRACT

It is quite common to use a specific test equipment when testing a new design today. These test equipments are often neither cost- nor time-efficient. At the company Aliaro where the assignment has been performed, a general test equipment is being developed that can easily be adapted from one test object to the next. The reason for developing a general test equipment is to save the time it takes to develop new specific test equipments and thereby reduce the costs. The purpose of this assignment was to evaluate the possibility of controlling a resistor ladder with software to make it emulate a real progress. In order to examine this it was decided to generate a square wave, and then increase the frequency to find out what maximum value could be reached before the wave differed too much from its origin. Furthermore, it also included the creation of a sine wave to be able to judge the quality of a more varied pattern over time.

The resistor ladder designed for the task consisted of 13 resistors in a series. Each resistor could be bypassed via an optocoupler. The optocouplers could be controlled individually or in groups of four with an LED-drive. The software that controlled the optocouplers was developed in LabVIEW and implemented in a so-called cRIO, a hardware device that among other things handles analogue and digital inputs and outputs via different modules. It has a built-in FPGA where the code is executed. The actuator for the optocouplers is controlled from the cRIO via the synchronous bus I^2C. The maximum frequency of an acceptable square wave was 519,8 Hz. The single biggest factor that prevented the circuit from achieving higher frequencies was the optocouplers. Their switching speed was too slow for faster results. If the intention is to proceed with the development of the resistance ladder to implement it in Aliaros test equipment one must find a faster optocoupler to improve the performance of the circuit by making it faster. It was possible to generate a sine wave, but because of various delays in optocouplers and in the code, the sine wave always had short dips or peaks at certain intervals where the optocouplers changed values in an unfortunate combination. This characteristic would not go away no matter how slow the frequency used.

INNEHÅLLSFÖRTECKNING

FÖRKORTNINGAR

ACK	Acknowledge
AI	Analog input
BOM	Bill Of Material (Stycklista)
COM	Common (gemensam jord)
DIO	Digital in- and output (digital in- och utgång)
DUT	Device Under Test (testobjekt)
D-sub	D-subminiatyr
ESD	Electrostatic Discharge
FPGA	Field-Programmable Gate Array (på-plats-programmerbar grindmatris)
GND	Ground (jord)
I²C	Inter-Integrated Circuit
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
LED	Light Emitting Diode (lysdiod)
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
NACK	No Acknowledge
REF	Referens
SCL	Serial CLock (synkron klocksignal)
SDA	Serial Data Line (datasignal)

1 INLEDNING

För att få en förståelse för upprinnelsen till detta uppdrag presenteras nedan en kort bakgrund till ämnet testning av industriprodukter. Vidare följer en precisering av uppgiften, att skapa en variabel resistansstege.

1.1 Bakgrund

Att testa en nykonstruktion inom industrin skedde i gamla tider ofta genom att montera ihop något specifikt för ändamålet som sedan testades praktiskt. I takt med att elektroniken blivit smartare och mer komplex har det blivit dyrare att alltid skapa specifik testutrustning för varje nykonstruktion. Nackdelen med specifik testutrustning är att den oftast har en hög utvecklingskostnad i form av material och tid. Till viss del är förfarandet med en för ändamålet specifik utrustning fortfarande aktuell, men behovet av att i ett tidigt stadium kunna utvärdera en konstruktion har ökat markant. Detta för att snabbt kunna göra bedömningen om en produkt har utvecklingspotential. Ur tids- och kostnadsperspektiv är det därför viktigt att använda en generell testutrustning som enkelt kan anpassas till nästa testobjekt. Ett bra sätt att uppnå detta på är genom simulering. Detta för att minska behovet av riktiga komponenter då dessa finns i en stor flora vilket gör att mycket tid behöver läggas på att finna de rätta komponenterna för testutrustningen. I vissa fall är det även så att de befintliga komponenterna inte är tillräckliga för att kunna återskapa förlopp vilket gör att man får vänta på att nya komponenter ska utvecklas. Aliaro är ett företag som specialiserat sig på att skapa kostnadseffektiva, robusta och pålitliga testlösningar åt sina kunder. Detta är möjligt tack vare kombinationen av hög kompetens bland dess medarbetare tillsammans med lång erfarenhet inom området. För att erbjuda kostnadseffektiva lösningar är en viktig del av testsystemen att kunna simulera verkliga förlopp. I takt med att testobjekten blir smartare och smartare behöver den simulerade verkligheten också den bli smartare för att testobjektet inte ska tolka en simulerad enhet som ett fel i systemet. Ett exempel på ett användningsområde där man idag behöver simulera verkligheten är vid test av vissa temperaturgivare. Tidigare räckte det med att lägga på en spänning på styrenhetens ingång för att få denna att tro att en riktig givare är inkopplad. I moderna system är styrenheterna dock mycket smartare och skickar en ström genom givaren vid regelbundna tillfällen för att kontrollera att givaren är hel. Har man då en inkoppling av äldre modell kommer ingen ström att kunna mätas och styrenheten rapporterar givaren som trasig. Konsekvensen av detta blir att testobjektet inte opererar normalt vilket är det motsatta mot vad man försöker uppnå. Aliaro har ett flertal koncept för att lösa problem som det ovan samt andra motsvarande där verkligheten behöver simuleras och de ville med detta uppdrag undersöka vad som var praktiskt möjligt att uppnå till en relativt låg kostnad.

1.2 Syfte

Syftet med uppdraget är att utvärdera möjligheten att efterlikna ett verkligt förlopp i en simulerad miljö. Mer specifikt består uppgiften i att utröna huruvida det är möjligt att med en mjukvarumässigt kontrollerad resistansstege efterlikna ett verkligt förlopp. Hänsyn ska även tas med avseende på kostnaden för denna typ av simulerat beteende då kravet på denna typ av utrustning är att konstruktionen ska vara enkel och relativt billig. Detta för att vara intressant att använda i större antal i framtida tillämpningar.

1.3 Precisering av frågeställningen

Målet med uppdraget var att kostnadseffektivt tillverka en resistansstege som kunde simulera ett verkligt förlopp med tillräcklig noggrannhet. För att bestämma om detta uppfyllts delades uppdraget in i nedanstående delmål:

- Konstruera en styrbar resistansstege.
- Skapa ett användarinterface i LabVIEW för att via I^2C styra resistansstegen.
- Bestämma vad som kan anses vara acceptabel avvikelse för en signal.
- Ändra resistansen i resistansstegen så att spänningen över ett tillkopplat mätmotstånd ändras i form av en fyrkantsvåg.
- Undersöka maximal frekvens på en simulerad fyrkantsvåg som fortfarande anses acceptabel.
- Ändra resistansen i resistansstegen så att spänningen över ett tillkopplat mätmotstånd ändras i form av en sinusvåg.
- Undersöka maximal frekvens på en simulerad sinusvåg som fortfarande anses acceptabel.
- Undersöka och dokumentera eventuella avvikelser för att se om dessa uppstår enligt ett mönster.
- Undersöka om det på ett enkelt sätt är möjligt att rätta till eventuella brister hos signalen.
- Ta fram ett system som uppnår företagets förväntningar och vars tillverkningskostnad anses skälig.

1.4 Begränsningar och krav

För att begränsa arbetets omfattning fanns vid uppstart de förbestämda begränsningar och krav som återfinns nedan.

- Avsatt tid för uppdraget för två personer var 10 veckor per person, där varje vecka bestod av 40 arbetstimmar.
- Programmeringen skulle genomföras med LabVIEW. Användargränssnittet ansågs tillräckligt om det fanns framtaget för LabVIEW RT, med andra ord behövde det inte finnas tillgängligt i Windowsmiljö.
- Styrenheten var bestämd att bestå av National Instruments produkter med ett cRIO chassi-9063 och en NI9401 modul.
- Resistansstegen skulle monteras på ett experimentkort.
- Resistansstegen skulle bestå av 13 seriekopplade resistorer med förbestämda värden.
- Styrande enhet på experimentkortet skulle vara en PCA9635.
- Kommunikationen mellan experimentkortet och cRIO-enheten skulle ske via I^2C, High-speed mode.
- En grundkonstruktion i LabVIEW för kommunikation med I^2C via FPGA fanns att utgå ifrån.
- Elektroniken skulle byggas för att drivas med 5 V, men den skulle även klara av att drivas med 3,3 V.
- Komponenterna som skulle köpas in hade ett kostnadstak på 500 kr.

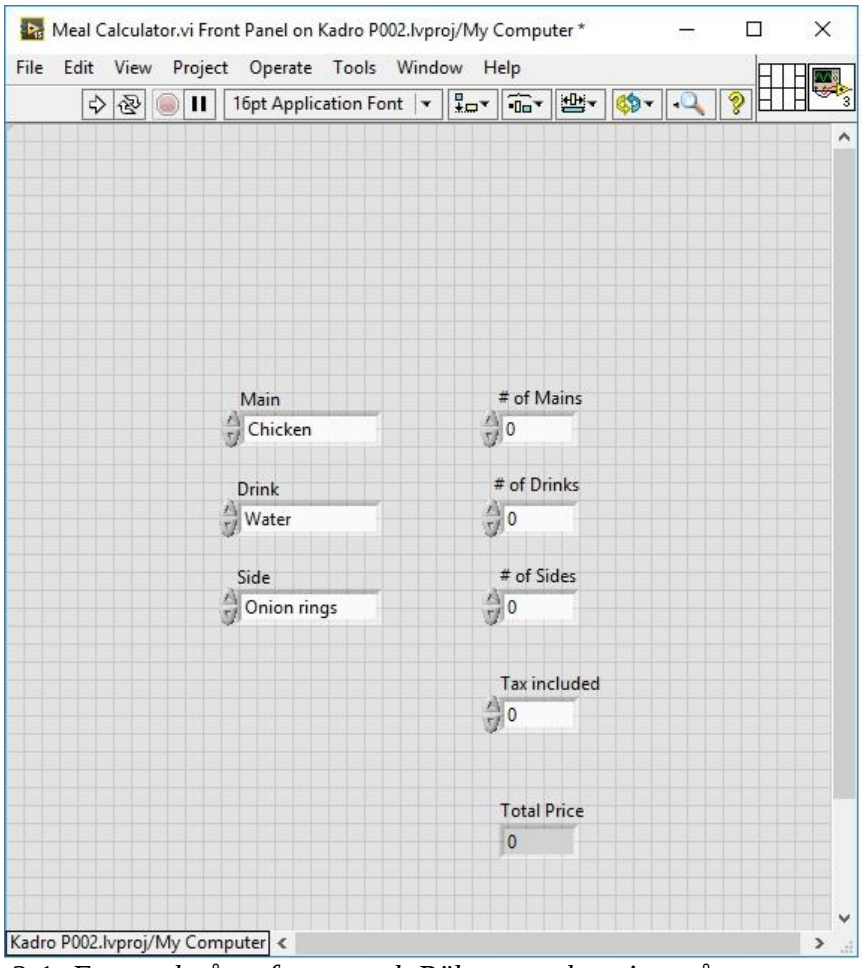
2 TEKNISK BAKGRUND

I detta kapitel ges en summering av olika mjuk- och hårdvara som har relevans för uppdraget.

2.1 LabVIEW

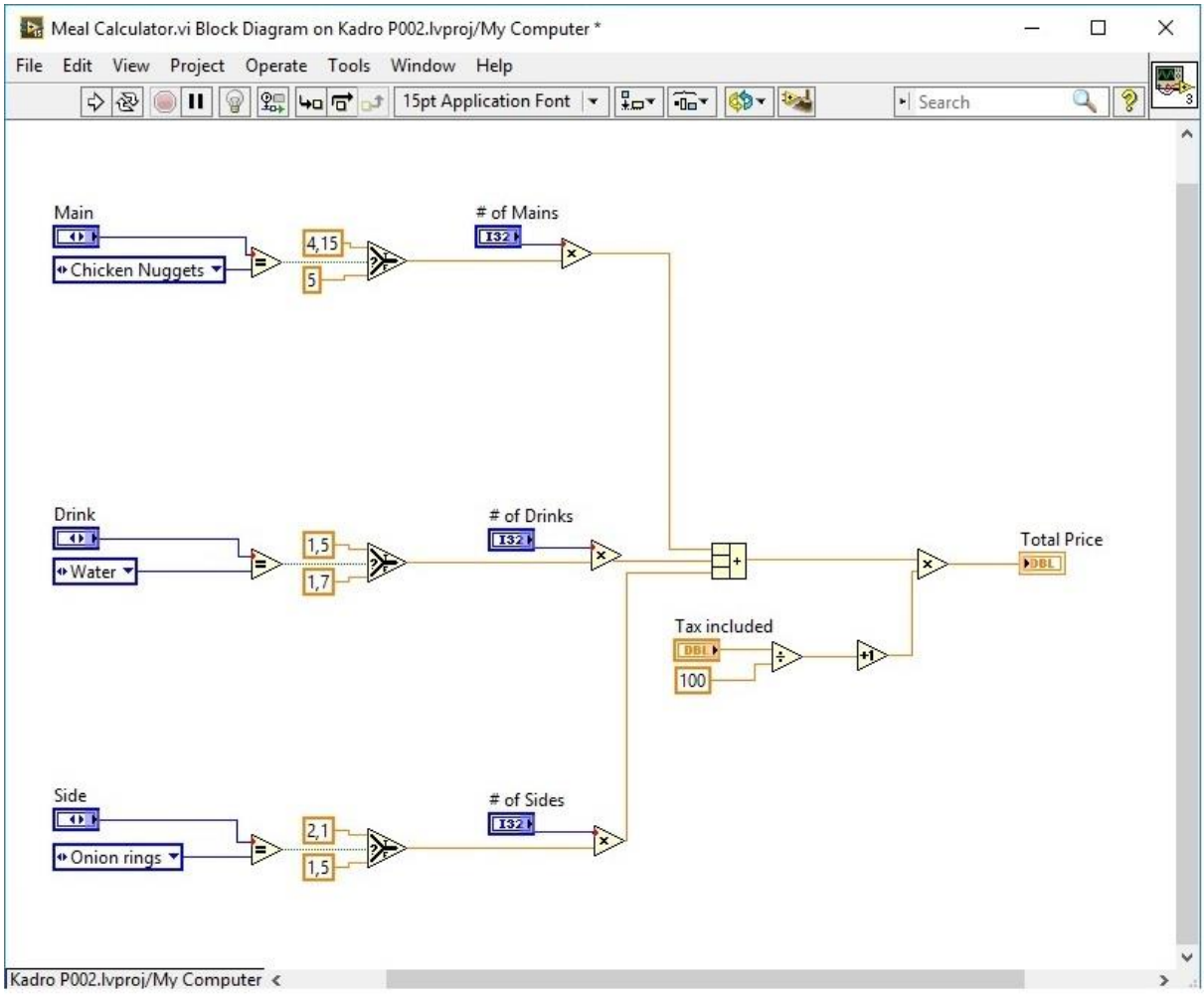
LabVIEW är en plattform och en utvecklingsmiljö för ett grafiskt programmeringsspråk, som ofta benämns G. Programmet är utvecklat av det amerikanska företaget National Instruments. LabVIEW togs först enbart fram för Macintosh och det såg dagens ljus första gången år 1986 även om tanken föddes redan 1983 (Travis och Kring 2007, 8, 22). LabVIEW används ofta för testning, mätning, styrning, analysering av data och för att presentera resultat (National Instruments [NI], 2016). Programmet går att använda på vanliga dataplattformar som Windows, Linux och OS X. Det går även att använda på inbyggda plattformar som FPGA och mikroprocessorer (Travis och Kring 2007, 3).

Ett LabVIEW-program består av ett eller flera VIs, virtual instruments. Ett VI är lite förenklat vad man kan kalla ett LabVIEW-programmeringselement (Mohiuddin, Nawrocki och Bitter 2006, 1). Namnet beror på att deras utförande och funktion ersätter fysiska instrument så som multimeter och oscilloskop (NI 2016). Ett VI har tre huvuddelar: ett blockdiagram, en frontpanel och en ikon (Mohiuddin, Nawrocki och Bitter 2006, 1). Frontpanelen, som motsvarar framsidan på ett instrument, är uppbyggd av ingångar och utgångar. De kallas kontroller (eng. controls) respektive indikatorer (eng. indicators). Via denna panel interagerar programmet med hjälp av knappar, reläer, displayer etc. med omgivningen (Travis och Kring 2007, 5-7).



Figur 2-1: Exempel på en frontpanel. Räknar ut slutpriset på en nota.

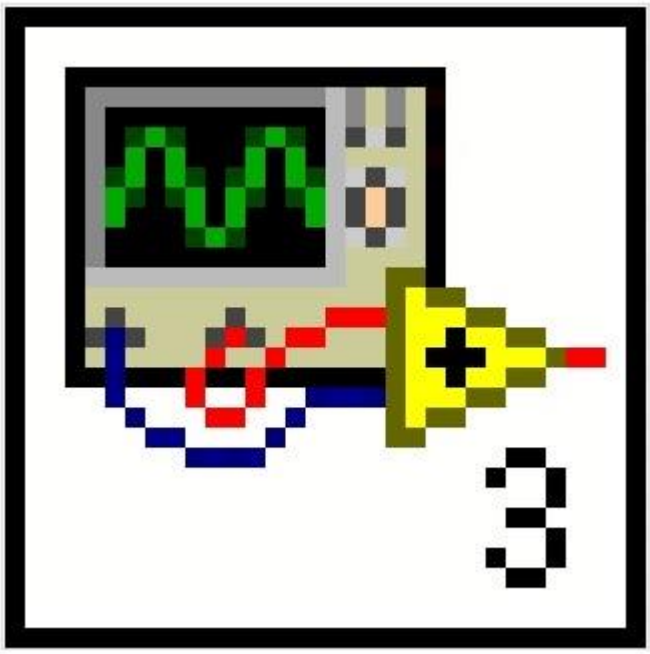
Blockdiagrammet är den grafiska källkoden, det är här man skapar sitt program med hjälp av terminaler, inbyggda funktioner, konstanter, loopar, andra VIs osv. (Travis och Kring 2007, 8). Koden i blockdiagrammet skapas i ett flöde och enligt NIs rekommendationer så bör den skapas så flödet går från vänster till höger (NI 2016). Alla block i LabVIEW är skapade utifrån detta så till vida att alla ingångar finns på vänster sida av blocket och alla utgångar sitter på höger sida. De olika funktionerna binds ihop med varandra med hjälp av trådar (eng. wire) som gör det enkelt att följa processen. Dessa har olika färger beroende på vilken datatyp som den består av, t.ex. grön för booleaner. In- och utgångarna från frontpanelen är direkt länkade till sin motsvarighet i blockdiagrammet (Travis och Kring 2007, 8, 46). En styrka med LabVIEW är att det är enkelt att lägga in parallella händelser och loopar. Dessa läggs helt enkelt in under varandra i blockdiagrammet för att de ska exekveras samtidigt (NI 2016).



Figur 2-2: Exempel på blockdiagram. Tillhörande exemplet på frontpanelen. Räknar ut slutpriset på en nota.

Slutligen har vi VI-ikonen som är en figur som representerar ett VI. Denna ikon har förbestämda mått, men utseendet på figuren kan utformas på egen hand så att ikonen speglar innehållet i dess funktion. När man har skapat ett VI kan man använda det som ett eget program eller som en subrutin i ett annat program. Om man vill lägga in ett VI i ett annat VI som en subrutin (sub-VI) så tar man dess ikon och lägger in den i ett övergripande VI. På detta sätt kan ett VI innehålla flera hundra sub-VIs (Mohiuddin, Nawrocki och Bitter 2006,

1). Varje ikon har en tillhörande Connector Pane, vars syfte är att definiera hur in- och utgångar kopplas till VI-ikonen (Travis och Kring 2007, 15).



Figur 2-3: Exempel på ikon. Visar ett instrument och en komponent.

En fördel med LabVIEW är dess överskådlighet. Det är även enkelt att exekvera flera olika sekvenser parallellt. Programmet innehåller ett stort bibliotek med olika funktioner för att simulera många olika sorters instrument. Det är ett väldigt användarvänligt språk där man inte behöver fundera på saker som pekare, allokering av minnesadresser och dylikt vilket är nödvändigt i t.ex. C (Travis och Kring 2007, 4).

LabVIEW är en proprietär programvara vilket innebär att den inte är en fri programvara. Det är företaget själv som beslutar vad som är öppet och tillgängligt för alla och vad man inte får komma åt. Det finns inga gratisversioner av LabVIEW på marknaden (utom en 30 dagars prova på licens) vilket gör att man måste lägga mycket pengar om man vill använda det, även om man bara vill göra en mindre förändring i befintlig kod. På grund av NI's monopol på LabVIEW så kan de ändra i språket utan att någon utomstående kan kräva kompatibilitet med tidigare versioner. Som programmet ser ut idag så är kod som är skapad i en nyare version inte öppningsbar i tidigare versioner, om man inte gör särskilda inställningar.

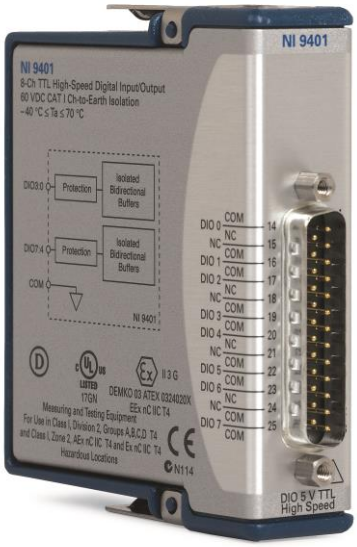
Till skillnad från t.ex. C så finns det inte någon utomstående kommitté som har skapat en standard för hur språket ska vara, t.ex. ANSI, ISO eller IEEE (Wikipedia 2016). Detta skulle kunna spela roll t.ex. vid kompatibiliteten mellan de olika versionerna.

2.2 NI hårdvara

Utöver mjukvaran LabVIEW har även hårdvara från National Instruments använts. En av deras produkter är compactRIO (eng. compact Reconfigurable In- and Output modules) en så kallad cRIO. Den har omkonfigurerbara moduler, realtidsregulator, en FPGA och ett expanderbart ethernetchassi. I detta arbete har en cRIO-9063 använts (se Figur 2-4). Den har en dubbelkärnig 667 MHz processor, 512 MB ickeflyktigt minne och 265 MB arbetsminne. Den har även en gigabit Ethernet port, USB-enhet och en seriell port. Chassit har plats för fyra moduler (NI 2016). Dessa moduler väljs utifrån syfte. NI har ett brett utbud av moduler,

med många olika sorter och kombinationer av digitala och analoga I/O. I denna uppgift har en NI9401 modul använts (se Figur 2-5). Den har använts till I²C-kommunikation mellan cRIO och experimentkort. Det är en DIO modul med åtta kanaler. Den är ställbar i tre olika lägen. Antingen åtta utgångar, åtta ingångar eller fyra in- och fyra utgångar. Modulen har en uppdateringsfrekvens på upp till 10 kHz. Signalnivån är 5 V (NI 2014). Det har även använts en NI 9205 modul. Vilket är en AI-modul med 32 kanaler som mäter spänningar inom området ±10 V med en upplösning på 16-bitar.

Figur 2-4: NI cRIO-9063, bild från NI



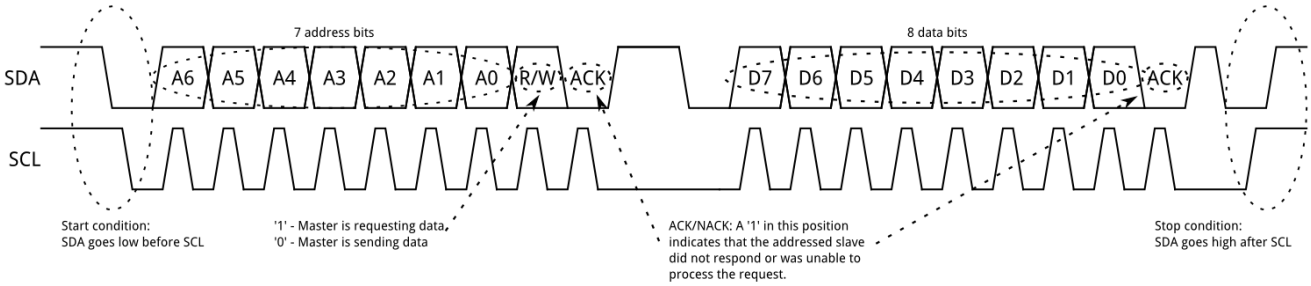
Figur 2-5: NI 9401-modul, bild från NI

2.3 I²C

Kommunikationen mellan cRIO och experimentkortet har skett med I²C, Inter-Integrated Circuit. Det är ett kommunikationsprotokoll som togs fram 1982 av Philips för att användas i deras kort (NXP 2014, 2). Protokollet är en synkron- och seriell buss som i originalutförande hade en kommunikationshastighet på 100 kHz. Kommunikation med bussen fungerar bäst på korta avstånd, upp till ett par meter (Paret, Fenger 1997, 14, 23).

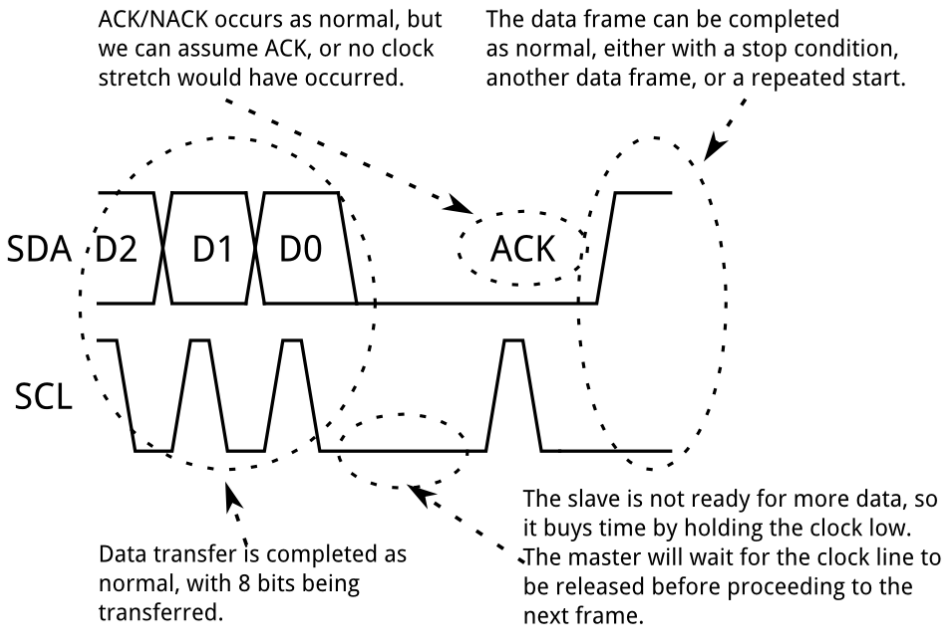
I²C består av två signaler, SCL och SDA (Paret, Fenger 1997, 24). Så länge det inte skickas någon data på bussen, d.v.s. systemet är i stand-by, så är båda signalerna höga. När det finns data så startas en dataöverföring. Den påbörjas alltid med en startsekvens där först SDA sätts låg, därefter sätts SCL låg (se Figur 2-6). Det är enbart vid start- och stoppsekvensen som

SDA ändras när SCL är hög. Startsignalen är ett tecken till att påbörja överföring av data. Denna skickas alltid i grupper bestående av 8 bitar, d.v.s. en byte. Efter given startsignal skickar SCL 8 stycken pulser i den förbestämda kommunikationshastigheten. Vid varje hög klockpuls läses värdet på SDA som antingen är hög eller låg. Den första data-byten som skickas innehåller information om adressen på noden kommunikationen ska ske med samt om det ska skrivas eller läsas till/från denna. De sju första bitarna representerar adressen och den sista biten bestämmer om det ska skrivas eller läsas.



Figur 2-6: I²C-sekvens med signalerna SDA och SCL (Hord 2013)

För att bekräfta att kommunikationen kommit fram och behandlats skickar mottagaren sen en bekräftelsebit, ACK, tillbaka till sändaren. Efter varje byte med data skickas en klocksignal där man kan få en bekräftelse eller brist på sådan huruvida datan har registrerats eller ej av mottagande nod (se Figur 2-7). Om allt gått bra skickar noden tillbaka en nolla på bussen (ACK), men om det gått dåligt så drar noden aldrig ner SDA och den signalen förblir hög (NACK) (Paret, Fenger 1997, 39). Det finns även en möjlighet för slaven att generera tid för att bearbeta informationen den fått innan den mottager nästa. Det gör den genom att hålla klocksignalen låg, så kallad klocksträckning, tills den har tänkt klart. Därefter släpper den klockan så att mastern kan fortsätta skicka information till slaven (NXP 2014, 13). Nackdelen med klocksträckning är att hela bussen kan hänga sig om slaven aldrig släpper klockan (Wikipedia 2016).



Figur 2-7: ACK-sekvens för I²C (Hord 2013)

Efter att adress-byten skickats och bekräftats skickas datan som ska distribueras till noden. Flera byte kan skickas efter varandra. Efter att varje byte med data sänts till noden vänds trafiken för att invänta ACK/NACK från noden. Därefter vänds trafiken tillbaka igen. När all data har skickats kommer en stoppsekvens. Den består av att SCL först sätts hög och därefter sätts SDA hög. Då återgår bussen till viloläge och båda signalerna är återigen höga, redo för nästa datautbyte. (Paret, Fenger 1997, 29-33, 38)

När I²C-protokollet togs fram användes enbart 7 adressbitar vilket begränsade antalet enheter på bussen till 112 stycken. Idag när fler noder används finns även stöd för 10-bitars adressering. Utöver originalhastigheten som senare kom att kallas Standard-mode (100 kHz) finns idag både Fast-mode (400 kHz), Fast-mode plus (1 MHz), High-speed mode (3,4 MHz) och Ultra fast-mode (5 MHz) (NXP 2014, 35).

Protokollet tillåter flera mastrar och slavar på samma buss, där slavar tillfälligt kan ändras till mastrar och vice versa om behovet finns. Om flera mastrar försöker skicka data samtidigt på bussen så är det den som kommer först som vinner kontrollen över bussen. Om de skickar exakt samtidigt till olika adresser så finns det ett prioriteringssystem, som kallas arbitrering. Detta innebär att den master vars mottagare har lägst adress kommer att vinna kontrollen över bussen. Utifall även adressen är samma så kommer den master som först sätter en låg bit då den andra har en hög att vinna kontrollen över bussen (Paret, Fenger 1997, 24, 52-54).

När man använder I²C kan man välja att hoppa över funktionen med ackreditering från mottagaren, då körs systemet i burst-mode. Detta läge är bra att använda om man har ett stabilt system med låg andel fel och där det är viktigare att man inte fastnar än att all data är helt korrekt. En fara med att använda ACK är att systemet kan hänga sig om detta inte genereras (Johansson 2016).

2.4 JIRA

Företaget Altassian har tagit fram en webbaserad mjukvara för projektledning som heter JIRA. Den används för att bland annat få en tydlig översikt över vilka arbetsuppgifter som finns i ett projekt, hur lång tid de olika delarna i ett projekt tar, vem som arbetar på vilken uppgift samt hur långt man har kommit med de olika delarna i projektet. Upplägget på programmet är att man delar in ett projekt i många smådelar (eng. issues) som är små och tydligt avgränsade uppgifter som med fördel kan klaras av att genomföra på några timmar upp till två dagar. Ett antal av dessa issues läggs sedan in i en sprint som är en tidsram man själv bestämmer tiden på, till exempel kan en sprint räcka över två veckor. Issues i en pågående sprint delas in i tre tillstånd: Att göra (eng. to do), pågående (eng. in progress) och klara (eng. done). Vid sprintens början ligger alla issues i att göra-tillståndet, sedan flyttas de av användarna över till pågående allt eftersom någon tar sig an uppgiften. När de är klara flyttas de till det sista tillståndet klara.

Med hjälp av JIRA får man en klar översikt om hur man ligger till i nuvarande sprint och hur många uppgifter som är kvar att göra. På så sätt är det enkelt att bedöma om man kan ta sig an mer uppgifter än det projekterade, om man behöver skjuta upp vissa uppgifter till nästa sprint eller kanske sätta in mer personal på ett projekt där man ligger efter så att det blir klart i tid (Atlassian 2016).

2.5 PCA9635

Den enda komponent på experimentkortet som var förbestämd var PCA9635. Det är en 16-bitars LED drivdon som styrs med en I²C-buss. PCA9635 är en av de första komponenter av

denna typ som är kapabel att köra I^2C-bussen i Fast-mode plus mode (Fm+), vilket tillåter hastigheter upp till 1 MHz. Komponenten har en intern oscillator på 25 MHz som inte behöver någon extern inkoppling för att fungera. Benkonfigurationen är TSSOP28 vilket bland annat talar om hur många ben komponenten har, hur långt avståndet är mellan dessa samt bentjockleken. PCA9635 är adresserbar för att man ska kunna använda flera I^2C-slavar i samma system. Upp till 126 olika adresser kan användas och 7 av komponentens ben är reserverade just för adressering. Beroende på vilka av dessa ben som ansluts till jord och vilka som ansluts till plus fås olika adresser. Komponenten drivs med en spänning mellan 2,3 och 5,5 V. Maximalt kan 25 mA och 5,5 V plockas ut på varje enskild utgång hos komponenten.

3 METOD

Detta kapitel beskriver det planerade tillvägagångssättet för uppgiften. Det första gjordes var att skapa en överblick över storleken på uppgiften och komma fram till vilka delar som ingick. Det gjordes med hjälp av brainstorming. Utifrån de delar som framkom skapades sedan en tidsplan. Denna skickades in till handledaren på Chalmers för godkännande. Därefter bröts de uppkomna uppgifterna ner i ännu mindre delar och med hjälp av JIRA skapades sedan tvåveckors-sprintar för att få en bra översikt över vad som skulle göras under de tio veckor projektet varade. Användandet av JIRA minskade risken för att någon del av uppgiften skulle glömmas bort.

3.1 I^2C

Då styrningen av resistansstegen skedde via I^2C behövdes erforderlig kunskap om hur denna databuss var uppbyggd. Sökning efter facklitteratur i ämnet genomfördes på nätet och i böcker. Detta för att skapa förståelse för hur bussen var uppbyggd, hur många och vilka signaler som ingick, om den behövde initieras eller termineras samt om det fanns några hastighetsbegränsningar att ta hänsyn till. Kunskap om PCA9635 inhämtades från dess datablad för att veta hur denna krets samverkade med I^2C. Den rörde bland annat initiering och adressering för komponenten. Motsvarande information inhämtades också för att förstå hur I^2C fungerar på en cRIO-9074.

3.2 Experimentkort

I uppgiften ingick att konstruera en resistansstege på ett experimentkort där man genom att öppna eller sluta olika brytare skulle kunna justera den totala resistansen över stegen mellan 0-4 kΩ. Det första som gjordes var att besluta konstruktionen för resistansstegen och dess styrfunktion. Därefter ritades ett kretsschema för kretsen upp i programmet KiCad. Vissa av de komponenter som användes till kretskortet var förvalda sedan start. För återstående komponenter undersöktes dess funktion och viktiga parametrar togs fram för att begränsa urvalet av dessa.

Komponenternas placering på experimentkortet bestämdes med hjälp av 3D-modelleringsfunktionen som fanns i KiCad. Därefter löddes komponenterna fast på deras respektive platser och förbindelsekablar drogs mellan de olika benen på komponenterna. När alla komponenter hade monterats så okulärbesiktigades lödningarna. Alla ledningar mättes även upp med multimeter för att kontrollera att kontakten var fullgod. Vidare så jämfördes kabeldragningarna mot kretsschemat så att dessa var korrekt utförda. När allt var kontrollerat kopplades experimentkortet ihop med cRIO-chassit.

3.3 LabVIEW

Det första som gjordes på området LabVIEW var att få en bra genomgång på den kod som redan fanns sedan tidigare. När den kunskapen tillgodogjorts sågs koden över för att se om befintlig kod behövdes kompletteras i samband med att man bytte modul till NI9401. Därefter gjordes de förändringar som krävdes för att koden skulle bli körbar mot experimentkortet. Det vill säga göra så att det via I^2C gick att skicka data till experimentkortet.

När fungerande I^2C kommunikation skapats mellan experimentkort och cRIO-modulen genererades en fyrkantsvåg. Det skedde genom att med I^2C styra resistansstegen på experimentkortet. Efter att fyrkantsvågen genererats undersöktes förutsättningarna för att skapa en sinusvåg. Ny kod för att generera den togs fram.

3.4 Testförfarande

För att på ett bra sätt kunna bedöma kvaliteten på de vågor som genererats så togs det fram ett testförfarande. Detta för att göra det möjligt att genomföra testerna igen vid ett senare tillfälle och då få samma resultat. När det beslutats om hur testerna skulle gå till väga genererades de vågor som skulle testas. Kvaliteten på fyrkantsvågen och sinusvågen bedömdes därefter. Utifrån resultaten som genererades kunde frågan om vad som var högsta möjliga frekvens för respektive vågsort besvaras. Detta med förutsättningen att den genererade vågen fortfarande höll sig inom spannet för vad som ansågs vara en acceptabel avvikelse från en perfekt våg. Avslutningsvis presenterades resultatet över de genererade vågorna och resultatet kunde sedan användas för att dra vissa slutsatser.

4 KONSTRUKTION

Framtagningen av konstruktionen har skett parallellt inom tre olika områden. Här nedan kommer de olika områdena att presenteras var och en för sig. Först presenteras en beskrivning av hårdvaran och de olika delar som ingått i denna. Därefter presenteras en beskrivning av mjukvaran och dess utveckling. Slutligen beskrivs hur de olika testmomenten av resistansstegen bestämts.

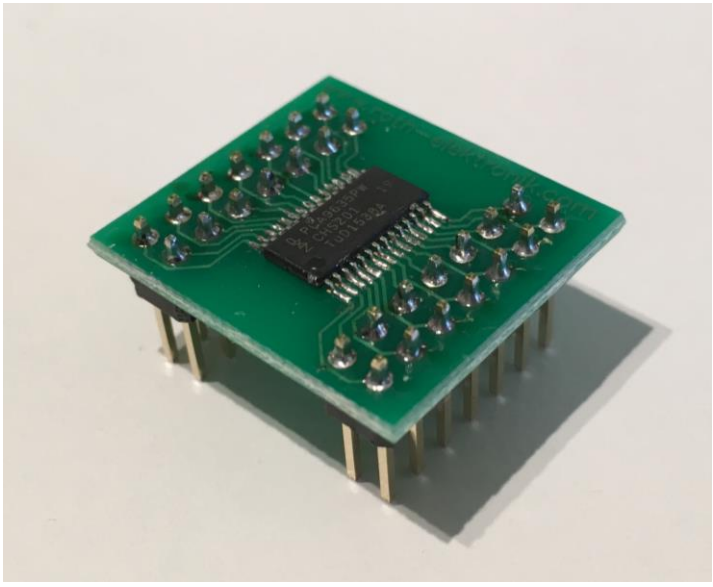
4.1 Hårdvara

För genomförandet av uppgiften behövdes en resistansstege tillverkas som kunde styras med mjukvara. Koden för styrningen var bestämd att exekveras via en NI cRIO-9074 med tillhörande NI 9401 modul. Av de komponenter som använts var det sedan tidigare bestämt att en PCA9635 skulle användas samt att komponenterna skulle monteras på ett experimentkort. Till de ej förvalda komponenterna som användes på experimentkortet skedde en urvalsprocess. Först beslutades det om vilka krav som skulle gälla för komponenten. Därefter valdes den komponent ut som var billigast men ändå uppfyllde kraven. Sökandet efter komponenter begränsades till de tre grossisterna Elfa Distrelec, Farnell element14 och Mouser electronics för att minska ner den uppsjö av komponenter som finns tillgänglig.

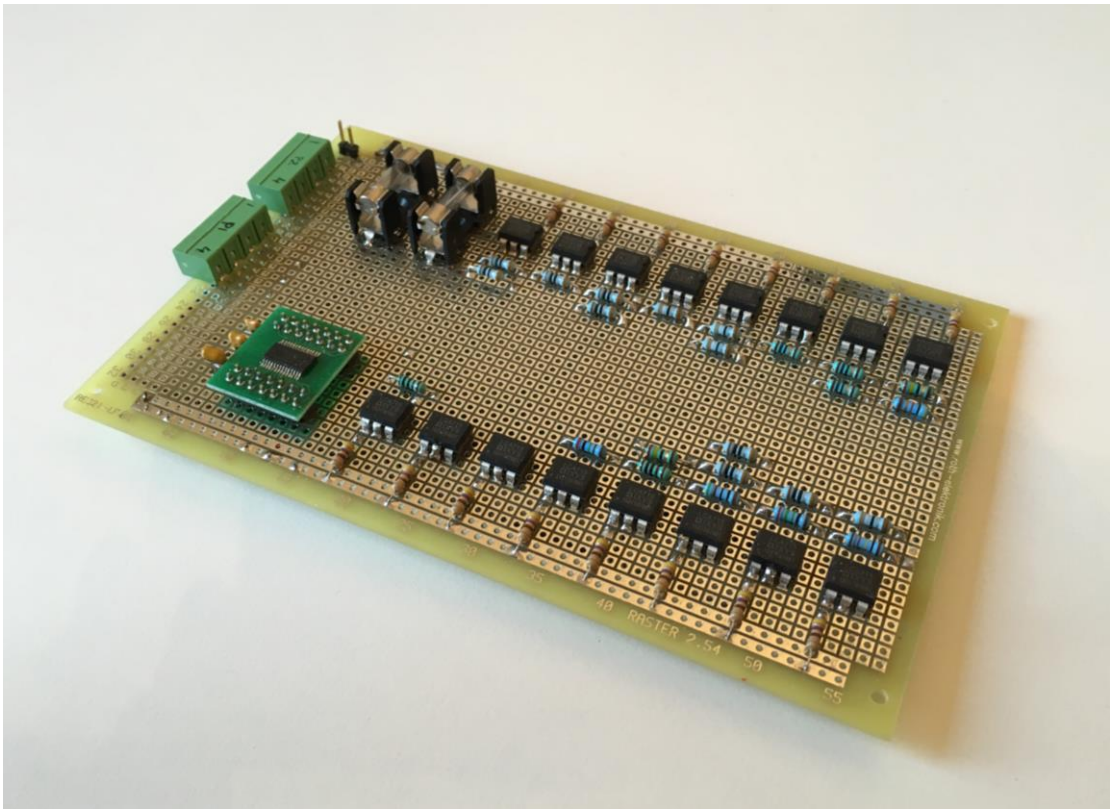
4.1.1 Experimentkort

De elektriska komponenter som använts i uppgiften har monterats på ett så kallat experimentkort. Denna sorts kort är lämpligt vid framtagning av prototypkretskort då man kan placera komponenter fritt på kortet. Komponenterna monterades på experimentkortet med hålmonterad lödning. Kortet som använts har ett rutnät bestående av hål, med tillhörande kopparöar. Dessa är ej sammankopplade med varandra. Längst med långsidorna på experimentkortet finns två rader av hål vars koppar är sammankopplad. En av dessa rader användes för att förse optokopplare med spänning (5 V).

Alla komponenter som använts har funnits att tillgå i hålmonterad version, utom PCA9635. För att underlätta monteringen av denna komponent införskaffades ett adaptermönsterkort där komponenten, via ytmontering, löddes fast. Då benkonfigureringen på komponenten var känd så var det enkelt att finna ett adapterkort. Detta monterades sedan, med stiftlister, på experimentkortet där det kunde lödas fast med hålmontering.



Figur 4-1: PCA9635 med tillhörande adaptermönsterkort



Figur 4-2: Experimentkort med resistansstege, optokopplare, PCA9635, säkringar, kontakter och kondensatorer

4.1.2 Resistorer

Resistorerna som har använts i resistansstegen har valts med maximalt 1 % felmarginal på värdet. De beslöts även ha en maxeffekt som inte skulle understiga 500 mW. För att minska kostnaden för komponenterna användes ibland serie- eller parallellkoppling av resistorer. Detta beslutades då vissa resistorer var betydligt dyrare än andra. I vissa fall har även närliggande värden använts. Tabellen nedan visar vilket värde på resistorerna som önskades i resistanstegen. Vidare står vilka värden på resistorerna som faktiskt användes samt eventuell serie- eller parallellkoppling av dessa.

Önskad resistans (Ω)	Faktiska resistorvärden		Koppling
0,5	1 // 1	0,5	parallell
1	1	1	-
2	1 + 1	2	serie
4	3,9	3,9	-
8	3,9 + 3,9	7,8	serie
16	16	16	-
32	16 + 16	32	serie
64	40,2 + 24	64,2	serie
128	124 + 3,9	127,9	serie
256	255 + 1	256	serie
512	510 + 1 + 1	512	serie
1024	1000 + 24	1024	serie
2048	2050	2050	-

Tabell 4-1: Resistansstegens önskade och faktiska värden samt dess koppling

Totalt var det 13 olika resistorvärden som användes i resistansstegen. Antalet resistorer som användes för att uppnå dessa värden var 23 stycken. Värdet på resistorerna har fördubblats för varje steg med en start på 0,5 Ω. Detta värde fördubblades sedan vilket gjorde att nästa värde blev 1, därefter följde 2, 4, 8 o.s.v. Resistansstegens totala resistans ändrades genom att välja vilka resistorer som skulle ingå i kretsen och vilka som skulle kopplas förbi med hjälp av optokopplare. Utöver de resistorer som användes i resistansstegen användes även resistorer till optokopplarna och PCA9635 för att dessa komponenter inte skulle gå sönder. Följande ekvationer användes för att räkna ut effekten och resistansen för de resistorer som användes till optokopplarna.

$$P = U * I \tag{4.1}$$

Ekvation (4.1) beskriver sambandet mellan storheterna effekt P, spänning U samt ström I och benämns effektlagen.

$$U = I * R \rightarrow \ I = \frac{U}{R} \tag{4.2}$$

Ekvation (4.2) är Ohms lag med storheterna spänning U, ström I samt resistans R (Ohm 1827). Om ekvation (4.2) sätts in i ekvation (4.1) erhålls följande.

$$P = U * \frac{U}{R} = \frac{U^2}{R} = \frac{5^2}{470} = 50\ mW \tag{4.3}$$

Ekvation (4.3) är en kombination av Ohms lag och effektlagen och visar maximala effektutvecklingen som kunde komma och uppstå i resistorerna till optokopplarna. Storheterna är effekt P, spänning U samt resistans R. För att ha en säkerhetsmarginal bestämdes maxeffekten till att inte understiga 250 mW. Maximal ström som får matas ut från PCA9635 är lägre än den ström optokopplaren får matas med. Detta gjorde att PCA9635 värde användes för att räkna ut resistorvärdet för att skydda kretsen.

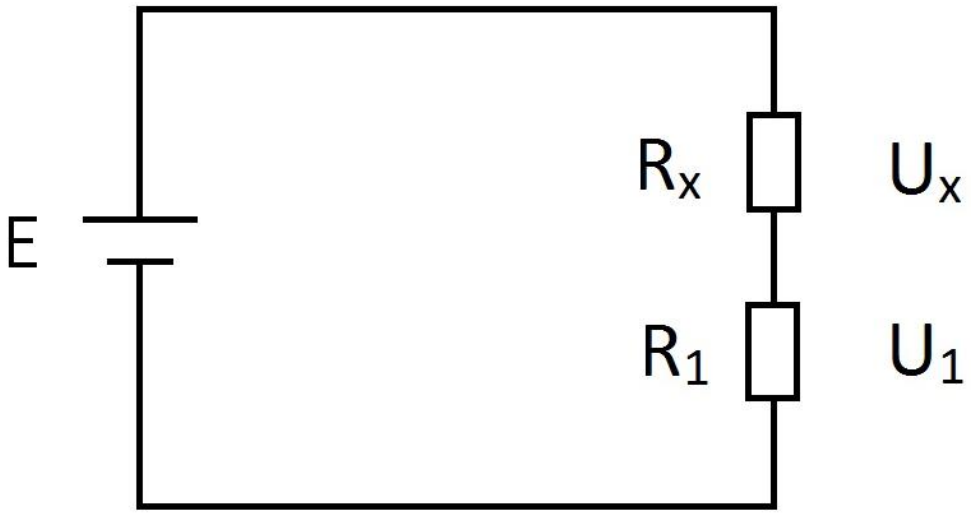
$$U = I * R \rightarrow R > \frac{U}{I} = \frac{5}{25*10^{-3}} = 200\ \Omega \tag{4.4}$$

Ekvation (4.4) är Ohms lag med storheterna spänning U, ström I samt resistans R och används för att räkna ut den resistans som behövdes för att begränsa strömmen ut från PCA9635. För att ha en säkerhetsmarginal valdes en resistor med resistansen 470 Ω.

4.1.3 Mätmotstånd

Om man skulle koppla förbi alla motstånd i hela resistansstegen förelåg stor risk att man skulle bränt upp kretsen. För att förhindra detta infördes ett extra motstånd som låg utanför resistansstegen och därmed fungerade som skydd för kretsen. Detta motstånd valdes till 1 kΩ vilket gjorde att strömmen begränsades till max 10mA om alla motstånd i resistansstegen var förbikopplade.

Det var även över detta motstånd som alla mätningar på kretsen genomförts, därav namnet mätmotstånd. I kretsschemat (Bilaga 1 - Kretsschema) heter detta motstånd R30. Alla mätningar som utförts i uppgiften har, om inte annat nämns, uppmätts över detta motstånd. Istället för att räkna på resistansen i kretsen så har spänningen uppmätts då detta är enklare att genomföra. Om man vill få ut vilken resistans resistansstegen har för en viss spänning så kan ekvation (4.12) användas. Figur 4-3 visar en schematisk bild över resistansstegen med spänningskälla och mätmotstånd. Storheterna som används i Figur 4-3 används även i formlerna nedan.



Figur 4-3: Kretsschema över resistansen över resistansstegen R_x, spänningskällan E och mätmotståndet R_1

$$U_x = R_x * I_x \quad (4.5)$$

Ekvation (4.5) är Ohms lag och beskriver sambandet mellan storheterna U_x (spänningen över resistansstegen), R_x (resistansen hos resistansstegen) och I_x (strömmen genom resistansstegen).

$$U_1 = R_1 * I_1 \quad (4.6)$$

Ekvation (4.5) är Ohms lag och beskriver sambandet mellan storheterna U_1 (spänningen över mätmotståndet), R_1 (resistansen hos mätmotståndet) och I_1 (strömmen genom mätmotståndet).

$$I = I_x = I_1 \quad (4.7)$$

Ekvation (4.7) är Kirchhoffs strömlag och beskriver sambandet mellan strömmarna i en nod. I detta fall sambandet mellan storheterna I (strömmen genom hela kretsen), I_x (strömmen genom resistansstegen) och I_1 (strömmen genom mätmotståndet). Det vill säga alla strömmar är lika stora då de alla går genom samma ledning. Hädanefter kommer enbart strömmen I för hela kretsen användas då det är bevisat att alla strömmar är lika stora.

$$E - U_x - U_1 = 0 \rightarrow U_x = E - U_1 \quad (4.8)$$

Ekvation (4.8) är Kirchhoffs spänningslag och visar att summan av alla spänningar är lika med noll i en sluten krets. Storheterna är E (spänningskällan till resistansstegen), U_x

(spänningen över resistansstegen) och U_1 (spänningen över mätmotståndet). Med ekvation (4-7) in i (4.5) fås följande:

$$U_x = R_x * I \quad (4.9)$$

Ekvation (4.9) är en kombination av Ohms lag och Kirchhoffs strömlag och beskriver U_x (spänningen över resistansstegen) som funktion av R_x (resistansen genom resistansstegen) och I (strömmen genom hela kretsen). Med ekvation (4.7) in i (4.6) fås följande:

$$U_1 = R_1 * I \quad (4.10)$$

Ekvation (4.10) är en kombination av Ohms lag och Kirchhoffs strömlag och beskriver U_1 (spänningen över mätmotståndet) som funktion av R_1 (resistansen hos mätmotståndet) och I (strömmen genom hela kretsen). Genom att bryta ut I ur ekvation (4.9) och ekvation (4.10) och jämföra dessa med varandra fås följande.

$$\frac{U_x}{R_x} = \frac{U_1}{R_1} \quad (4.11)$$

Genom att ersätta U_x med ekvation (4.8) i (4.11) fås följande.

$$\frac{E-U_1}{R_x} = \frac{U_1}{R_1} \quad \rightarrow \quad E - U_1 = \left(\frac{U_1}{R_1}\right) * R_x \quad \rightarrow$$

$$R_x = \frac{E-U_1}{\left(\frac{U_1}{R_1}\right)} = \frac{E-U_1}{(U_1)} * R_1 = \left(\frac{E}{U_1} - \frac{U_1}{U_1}\right) * R_1 = \left(\frac{E}{U_1} - 1\right) * R_1$$

$$R_x = \left(\frac{E}{U_1} - 1\right) * R_1 \quad (4.12)$$

Ekvation (4.12) är slutresultatet av vilken resistans resistansstegen har för en viss spänning över mätmotståndet. Storheterna är R_x (resistansen hos resistansstegen) som funktion av U_1 (spänningen över mätmotståndet). Konstanter tillika storheter E (spänningskällan till resistansstegen) och R_1 (resistansen hos mätmotståndet).

4.1.4 PCA9635

Sedan tidigare var det beslutat att kretskortet skulle styras med I²C via komponenten PCA9635 tillverkad av NXP. På experimentkortet används 13 av PCA9635 utgångar för att kunna styra förbikopplingen av de olika resistorvärdena, enligt den första kolumnen i Tabell 4-1. De tre resterande utgångarna hos komponenten användes för att med hjälp av optokopplare koppla förbi flera resistorer i resistansstegen. Se kretsschemat (Bilaga 1 - Kretsschema) för närmare information om detta.

4.1.5 Optokopplare

För att uppnå så snabba frekvenser som möjligt har valet av vissa komponenter spelat större roll än andra. Optokopplaren är en komponent som finns i många olika utföranden och med mycket varierande omslagningshastighet. Ett önskemål på optokopplaren var att den skulle bestå av en så kallad MOSFET. Det är en förkortning för fälteffekttransistor med metalloxidhalvledare. Detta önskemål gjorde att urvalet av optokopplare på marknaden minskade, men trots det fanns det fortfarande ett väldigt stort urval att välja mellan. Därför ställdes även följande krav på optokopplaren:

- Klara av en ström på minst 100 mA genom utgången.
- Enpolig
- Maximal resistans på 0,15 Ω genom utgången.
- Maximal framström för att sluta optokopplaren på 20 mA. (Måste vara mindre än 25 mA som är maximalt tillåten ström ut från PCA9635)

Därutöver önskades även en omslagstid på mindre än 0,07 ms. Vilket är närmare I²C-bussen i hastighet och skulle ge en snabbare simulering. Men det fanns inte att tillgå med ovanstående krav uppfyllda hos de grossister som använts. Därför uppfylldes inte kravet på 0,07 ms

omslagstid och en optokopplare med en omslagstid på 0,7ms har använts. Komponenten som valdes hade beteckningen LCA715 och tillverkades av IXYS.

4.1.6 Kondensatorer

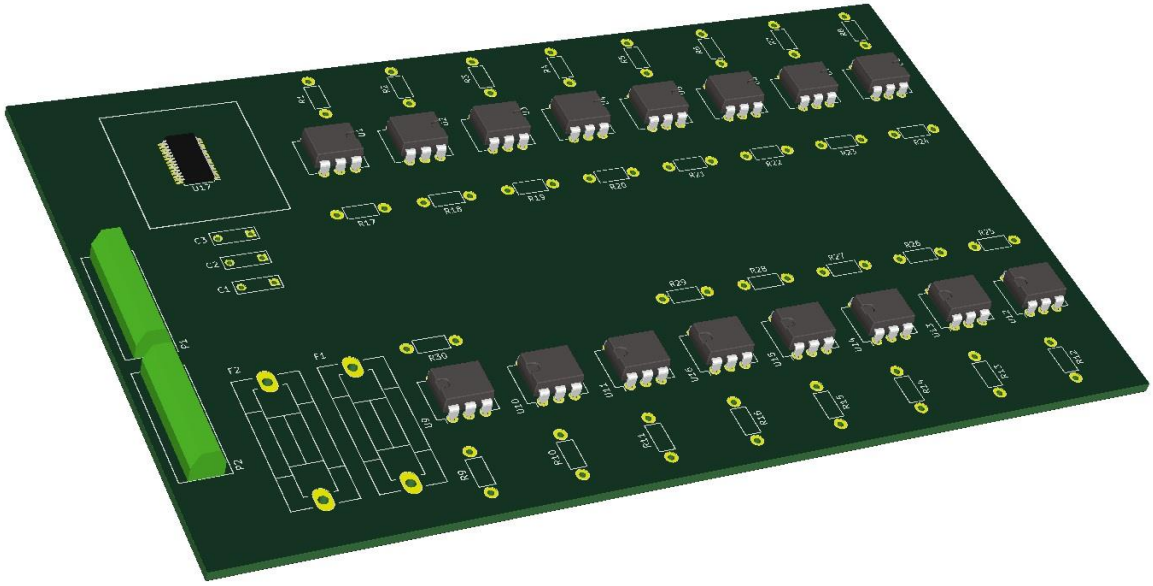
För att spänningen skulle hållas på en jämn nivå till PCA9635 och inte spänningsdippar skulle uppstå användes kondensatorer mellan matningsspänning och jord. Störst risk för spänningsdippar förelåg då optokopplarna slog om. För att skydda kretsen inom olika frekvenser så användes kondensatorer med värdena: 10 nF, 100 nF och 1 µF. Dessa kondensatorer placerades så nära PCA9635 som möjligt för största möjliga nytta.

4.1.7 Säkringar

Säkringar användes för att avsäkra både 5 V- och 10 V-spänningen på kretskortet. Detta för att undvika att bränna komponenter om kortslutning skulle uppstå. Säkringshållare monterades på experimentkortet för att underlätta utbyte av förbrukade säkringar. Sedan tidigare fanns säkringar på 400 mA, dessa var adekvata för uppgiften och användes därmed.

4.1.8 Kretsschema

Det togs fram ett kretsschema (se Bilaga 1 - Kretsschema) över resistansstegen med dess I^2C-mottagare, reläer, resistorer och övriga komponenter. Programmet som användes för framtagandet av kretsschemat var KiCad. När schemat färdigställts användes programmet även för att i grova drag bestämma komponentplaceringen. KiCad har en bra funktion för att generera en 3D modell över kretskortets layout. Den ger en förståelse för det färdiga kretskortets utformning (se Figur 4-4). Utöver dett skapades även en BOM över de komponenter som användes och köptes in (se Bilaga 8 – BOM).



Figur 4-4: 3D bild på kretskortet ritat i KiCAD

4.1.9 Spänningskällor

Experimentkortet spänningssattes från två separata spänningsaggregat. En fördel med separata spänningskällor är att man kan separera kretsarna så att eventuella störningar i kretsen med optokopplarna inte påverkar kretsen med resistansstegen. Detta gör mätningarna på resistansstegekretsen mer exakt. De separata jordarna på de olika aggregaten kopplades samman med en resistor på 1 kΩ. I och med denna ihopkoppling uppnåddes samma

referensjord i de olika kretsarna. Vilket användes för att inte spänningarna i förhållande till varandra inte ska bli för stora. Spänningen till kretsen med PCA9635 och optokopplarna var på 5 V. Kretsen med resistansstegen drevs med 10 V.

4.1.10 Kablage

Hopkopplingen mellan experimentkortet, spänningsaggregaten och cRIO består av kablage tillverkat för ändamålet. Kontaktdonen som använts på experimentkortet var fyrpoliga kontakter från Phoenix. NI 9401 modulen i cRIO chassit är bestyckad med en D-subkontakt och i kablaget har dessa använts i utförandet anpassat till lödning. Spänningsaggregaten är bestyckade med banankontakter.

4.1.11 Inkoppling av NI 9401

NI 9401-modulen kan enbart växla mellan in- och utgångar i grupper om fyra. I^2C-bussens ena signal, SCL, är enkelriktad medan den andra signalen, SDA, växlar riktning. Omslagningshastigheten för modulen för att växla från in- till utgång eller tvärt om ligger på 100 ns. På grund av detta beslöts att dessa signaler skulle kopplas in på varsin grupp i NI 9401-modulen. Kanal noll och fyra valdes då de är de första signalerna i vardera fyrkanalsgrupp. Detta motsvarade ben 14 respektive 20 på modulen. Tack vare detta kunde SCL alltid förbli en utsignal medan SDA växlade mellan att skriva och läsa, till exempel för att kunna ta emot en ACK från slaven när det skrivits till denna. Utöver dessa signaler kopplades även jord (COM) från ben ett på modulen till jorden på kretsen som driver PCA9635 och optokopplarna (5 V kretsen). Alla dessa signaler drogs in till kortet via kontakt P1 (se Bilaga 1 - Kretsschema).

4.1.12 Kostnader

Sedan tidigare fanns ett experimentkort samt kablar för tillverkningen av resistansstegen. De övriga komponenterna till kortet kostade totalt 923 kr. Då är inte hårdvaran från NI inräknad. Den komponent som stod för högst kostnad var optokopplarna, 614 kr för alla 16 st. Därefter kom LED-drivdonet med en kostnad på 124 kr. Kondensatorerna och motstånden kostade från några kronor ner till 50 öre. Problemet med dessa var att de oftast fick köpas i tio-pack. Vid uträkningen av priset har man inte tagit hänsyn till detta utan styckpris har använts.

4.1.13 Problem

Det uppstod några bekymmer under arbetets gång, de presenteras här i punkter.

- Phoenixkontakterna var mycket svåra att montera på kortet. Detta berodde på att benen på kontakterna var något stora i förhållande till hålen på experimentkortet. Innan montering var möjlig behövdes därför benen smalnas av. Problemet löstes genom att slipa benen från deras ursprungliga kvadratiska form till mer runda.

- PCA9635 var en väldigt liten komponent med små ben som satt tätt och därför var svår att löda på adapterkortet. Detta genererade överlödningar där lödfläta fick användas flertalet gånger. På grund av tidigare erfarenheter inom området så blev alla lödningar bra tillslut.

- Kablarna som användes till experimentkortet för att koppla ihop komponenterna var svårlödda. Detta berodde på att de krävde mycket värme för att lödningen skulle bli fullgod. En följd av den höga värmen var att isoleringen till kablagen krympte och klättrade upp vilket frilade mer av kardelen än önskat.

- Tyvärr fanns ingen ESD-duk att tillgå från företaget eller skolan. Så det enda som användes för att undvika elektrostatisk urladdning var försiktighet och ett jordat ESD-armband. Som tur var så skadades inga komponenter trots avsaknaden av en ESD-duk.
- Den optokopplare som användes i uppgiften fanns inte med i komponentbiblioteket i KiCad. En symbol till komponenten skapades enkelt genom att utgå från en befintlig symbol för en liknande komponent. Däremot uppstod svårigheter med att länka bilden till en så kallad footprint. Det är en bild som ritas på kretskortet för att visa var komponenten ska ligga och hur dess benkonfiguration ser ut. Detta problem löstes genom att se på olika instruktionsvideor om KiCad på nätet och använda de lärdomar som införskaffats.

4.2 Mjukvara

För att kunna styra hårdvaran som beskrivits ovan behövdes mjukvara. Nedan beskrivs den kod som fanns färdig vid uppdragets start. Vidare följer en beskrivning av de ändringar som utförts i koden på grund av olika sorters problem som uppdagades. Slutligen följer en beskrivning av den nya kod som skapats för uppgiften.

4.2.1 Befintlig mjukvara

Vid uppdragets start fanns redan en grundkonstruktion i LabVIEW för I²C-kommunikation. Den hade en frontpanel där man kunde göra olika inställningar för I²C kommunikation (se Figur 4-5). De inställningar som bestämdes här var:

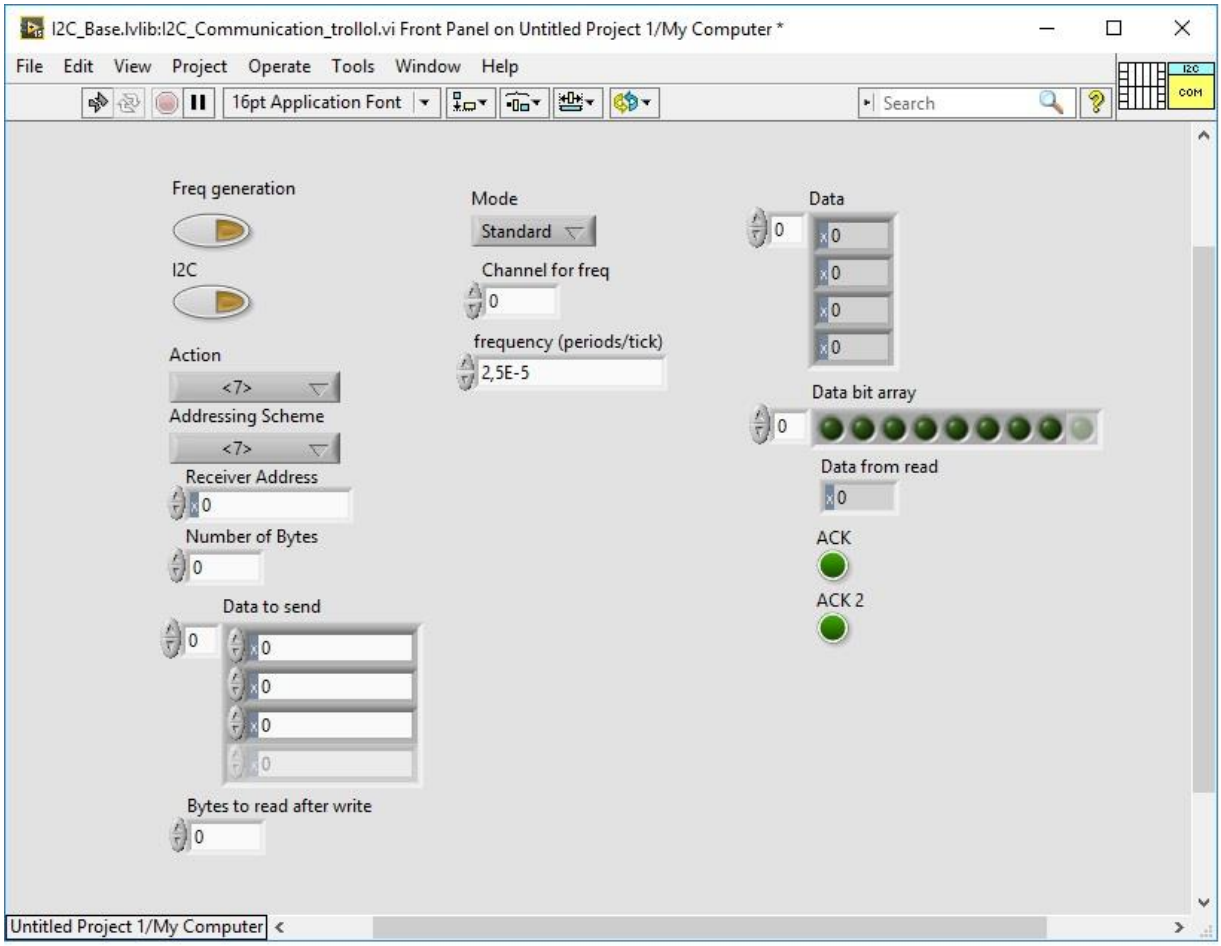
- Hastighet på bussen. Valet stod mellan:
  - Standard (100 kb/s)
  - Fast (400 kb/s)
  - High-speed (3,4 Mb/s)
- Skriva, läsa eller både och till slavenheten
- 7 eller 10 bitars adress
- Adressen på slavenheten
- Antal databyte som ska skicka
- Datan (skrivs in som en array med 8 bitar per element)
- Antal byte som ska läsas efter man har skrivit till slavenheten, om man har valt att både skriva och läsa till slavenheten

När man fyllt i alla värden klickade man sedan antingen på knappen I²C, om det enbart skulle skickas en array med data, eller knappen Freq generation om man ville generera en fyrkantsvåg genom att skicka data upprepade gånger. Vid val av freq generation fick man även bestämma:

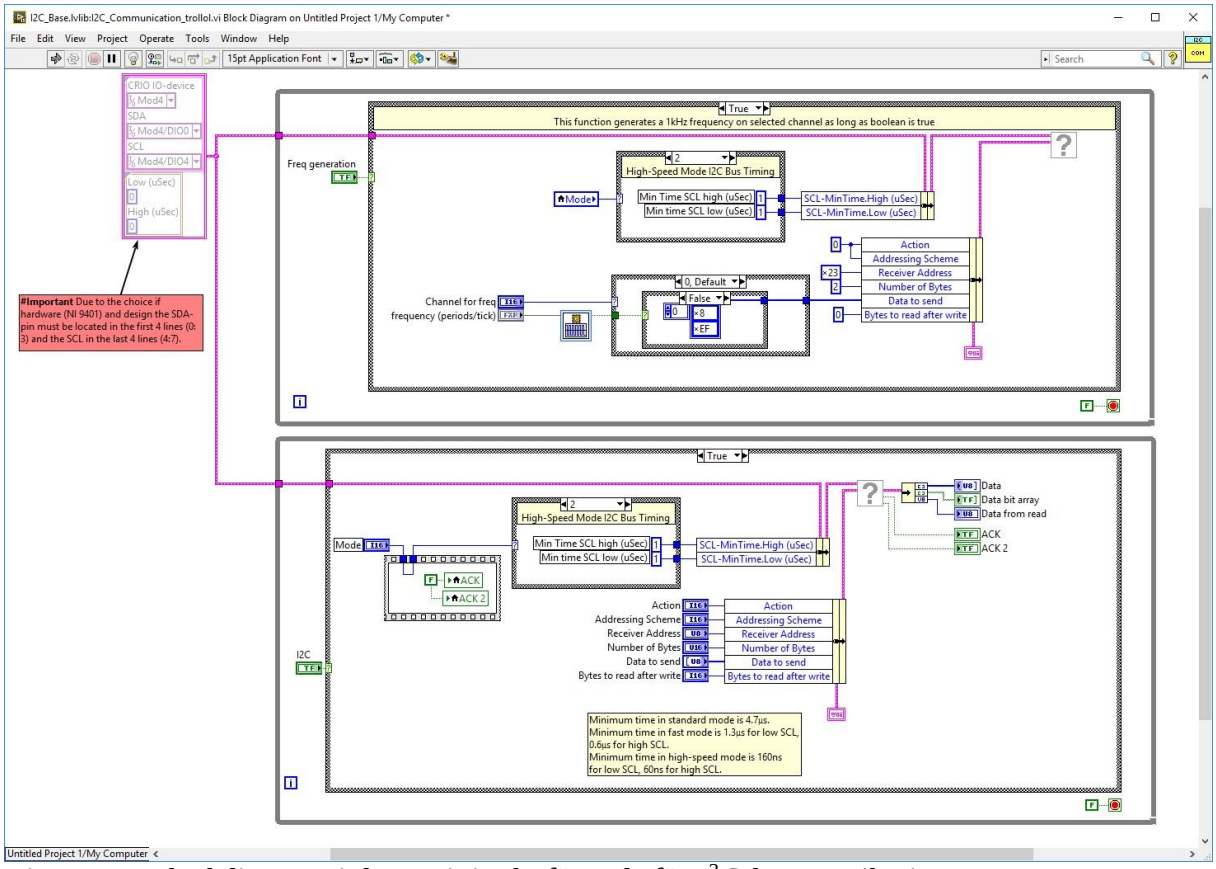
- Frekvens på upprepningen
- Val av varifrån man hämtar värdet på datan

Det som gick att utläsa från frontpanelen var:

- Data mottagen från slavmodulen om man befann sig i läsläge (detta används inte här)
- Bekräftelse från slavmodulen om mottaget meddelande (ACK och ACK 2)



Figur 4-5: Frontpanel i dess originalutförande för I^2C-kommunikation



Figur 4-6: Blockdiagram i dess originalutförande för I²C-kommunikation

4.2.2 Initiering av LED-drivdon

PCA9635 behövde initieras vid uppstart. Detta gjordes genom att ställa in komponentens olika funktioner i dess två första register. De hette MODE1 och har adressen 00 hexadecimalt respektive MODE2 med adressen 01 hexadecimalt. Alla register i komponenten bestod av åtta bitar, b0-b7. De initieringsvärden som först användes vid uppstart var:

MODE1

b0=0	Reagerar ej på så kallade LED All Call anrop
b3-b1=000	Reagerar ej på I²C-buss subadressanrop
b4=0	Normalt läge, utgångarna kan ändra värde
b7-b5=000	Visar hur autoincrement är inställt (read only)

Om värdena ovan användes innebar det att datan som skickades till MODE1 registret blev 0000 0000 binärt.

MODE2

b1-b0=01	Fördefinierad inställning för $\overline{OE}$, används ej
b2=1	Konfigurering av utgångar till totem-pole struktur
b3=0	Utgångarna ändrar värde vid stopp-sekvens
b4=0	Utvärdet är ej inverterat
b5=0	Förinställd på dimning istället för blinkning, ingen av dessa användes
b7-b6=0	Reserverade bitar som ej används

Detta motsvarar 0000 0101 binärt.

För att ändra ett register på PCA9635 skickades två byte med data. Den första byten, adressbyten, talade om vilket register som skulle påverkas och den andra byten, databyten, talade om hur de skulle påverkas.
Så för att initiera kretsen skickades följande värden på bussen:

00 00 (hex) Detta innebär att adress 00, MODE1, sätts till värdet 00.
01 05 (hex) Detta innebär att adress 01, MODE2, sätts till värdet 05.

Efter att initieringen var genomförd gjordes försök att öppna och stänga de 16 optokopplarna. Det gick att styra fyra olika optokopplare med varje adressbyte. Vilket gjorde att det behövdes fyra register för att kunna styra alla 16 optokopplare som använts till resistansstegen. De optokopplare som användes för att styra resistansstegen var fördelade på adresserna 14-17, hexadecimalt. Den lägsta resistansen på 0,5 Ω låg på adress 14 och den högsta resistansen på 2048 Ω låg på adress 17. De tre optokopplare som styrde förbikopplingen av stegen låg även de på adress 17. Varje optokopplare styrdes av två databitar var i databyten. Om dessa bitar var 00 så skulle optokopplaren öppnas, men om de var 01 skulle optokopplaren slutas. Så om databyten innehöll 0101 0101 binärt (55 hexadecimalt) skulle alla fyra optokopplare slutas och resistorerna kopplas förbi. För att sluta alla 16 optokopplare behövde man alltså skicka 8 byte motsvarande de hexadecimala värdena:

14 55
15 55
16 55
17 55

4.2.3 Styrning av optokopplare

Den befintliga koden för överföringen via I^2C var inte helt färdig så vissa problem uppstod under uppdragets gång. En sak som genererade ett följdproblem var att koden från början var konstruerad för en annan NI-modul, nämligen NI 9403. Denna byttes ut mot en NI 9401. Den största skillnaden mellan dessa två moduler i detta sammanhang var att NI 9401 var snabbare. 0,1 µs jämfört med 7 µs. Modulbytet gjorde att vissa ändringar var nödvändiga för att kommunikationen skulle fungera mellan experimentkortet och den nya modulen. Bland annat var konfigurationen av portarna annorlunda. I NI 9401-modulen var portarna grupperade fyra och fyra där man satte varje grupp till in- eller utgång. I föregående modul kunde varje port sättas till in- eller utgång var för sig. Det hade gjorts ändringar för detta i koden innan uppdragets start, men det uppstod även andra bekymmer relaterat till att modulen bytts ut.

Vid inkoppling av de två bussignalerna till ett oscilloskop upptäcktes att SCL-signalen inte genererade förväntat pulståg utan förblev oförändrad. Detta kunde härledas till att NI 9401-modulen till skillnad från tidigare modul behövde ha portarna förinställda som utgångar. Efter att porten till SCL initialt ställdes om till en utgång så genererades därefter önskat pulståg. Experimentkortet kopplades ihop med NI 9401-modulen och slaven tilldelades adressen 60 hex. Därefter initierades kortet enligt ovan och därefter provades att skicka data till experimentkortet genom att växelvis skicka 00 och 55 till de olika adressregistren (14-17). För att sluta en optokopplare krävdes att en spänning på 0 volt skickades in på ben 2 på optokopplaren. För att den skulle öppnas behövdes en spänning på 5 V. Spänningsmätning på optokopplarnas styrsignaler gjorde det möjligt att kontrollera om PCA9635 behandlade datan rätt och därmed skickade rätt styrsignal till rätt optokopplare. Tyvärr gick det inte att styra optokopplarna när datan skickades. För att komma till rätta med problemet testades att ändra fördröjningen i koden från ticks till mikrosekunder för att generera mer tid för att läsa av värden. En tick är den tid som en exekvering i FPGA tar att genomföra, i detta fallet är en tick

25 ns. Detta förändrade ingenting. PCA9635 ändade ibland optokopplarna så att dessa slog om när man skickade över något på bussen. Förändringen som skedde var dock helt slumpartad och inget mönster stod att finna i när eller varför den ändrade på några, eller flera av optokopplarna. En svårighet vid dessa tester var att finna var felet låg. Så länge inget fungerat som förväntat kunde problemet lika gärna finnas i hårdvaran som mjukvaran.

För att försöka få klarhet om det fanns något mönster i hur styrningen av optokopplarna betedde sig gjordes en grundlig genomkörning av alla optokopplare. PCA9635 initierades på nytt för att säkerhetsställa att den var rätt inställd. Efter det skickades först data för att ställa om optokopplarna enskilt en och en. Därefter testades att öppna eller sluta alla optokopplare på samma registeradress. Detta förfarande upprepades flertalet gånger. Oftast gick det att sätta en optokopplare hög om den innan var låg, däremot gick det inte att påverka den åt andra hållet, från hög till låg. Utöver detta gick inget mönster att tyda. Nästa steg i att hitta en lösning var att byta ut NI 9401 mot den tidigare modulen NI 9403 för att testa om denna kunde styra optokopplarna. Det gav samma oregelbundna resultat som innan.

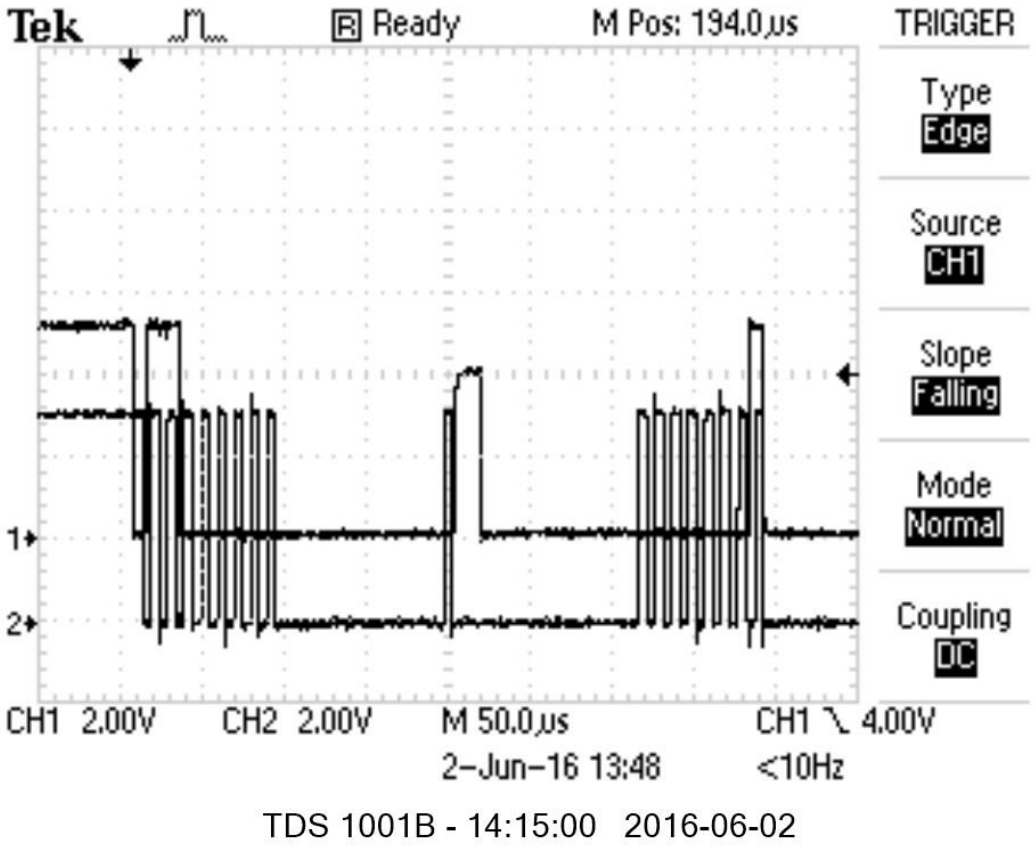
I detta skede av problemlösandet lånades en annan I^2C-slav, en realtidsklocka med beteckningen DS3231 tillverkad av Maxim intergrated. Det gjordes för att testa om koden för kommunikationen fungerade. Här skedde en framgång då denna slav gick att styra. På grund av denna framgång så kunde det konstateras att problemet, med att inte kunna styra optokopplarna enligt förväntning, högst troligt inte fanns i koden. Detta ledde tankarna till det som var specifikt för experimentkortet. Då optokopplarna slagit om tidigare minskades sannolikheten för att det var ett hårdvarufel. Nästa område att undersöka vid problemlösningen var att se över initieringen av PCA9635. Det framkom att man i ursprungskoden inte helt fått till funktionaliteten med stop-sekvensen för I^2C-kommunikationen vilket gjorde att initieringssekvensen justerades utifrån denna information. I MODE2 ändrades bit 3 från 0 till 1 så att utgångarna uppdaterade värdet vid ACK istället för vid stop-sekvens. Vidare ändrades även strukturen på utgångarna från totem-pole till open-drain i MODE2 bit 2. Detta gjorde att initieringsvärdena istället blev:

00 00
01 09

I samband med att initieringsvärdet uppdaterades gjordes även en justering på I^2C-knappen. Tidigare hade knappen haft inställningen "switch when pressed", det vill säga att den bytte värde mellan på och av varje gång man tryckte på knappen. Den byttes till "latch when pressed", det vill säga att den blev återfjädrande. Efter dessa justeringar kördes koden i RT-miljö för att spara in tid på kompilering till FPGA. Nu fungerade det att styra optokopplarna. Det vill säga att den uppmätta spänningen på ben två på optokopplarna bytte värde i enlighet med det som skickades. Det enda problem som sågs var att man oftast fick skicka iväg datan två gånger innan det gavs genomslag och optokopplarna slogs om. Med hjälp av ett oscilloskop kunde konstateras att problemet med dubbeltryckningen inte berodde på experimentkortet då bussen inte heller förändrades förrän vid den andra tryckningen på I^2C-knappen. Efter genombrottet med att kunna ändra optokopplarna byttes modulerna igen och NI 9401 testades med samma resultat. Detsamma gällde då koden efter kompilering provades att köras från FPGA. Optokopplarna gick att styra.

Vid granskning av de två signalerna SDA och SCL via oscilloskop kunde man se att den delen av dataskickandet på bussen som tog längst tid var före och efter ACK, (se Figur 4-7). Fördröjningen vid ACK hänvisas till att NI 9401 modulen behövde växla håll på SDA. En annan sak som uppdagades vid granskning av SDA och SCL var att SCL sattes hög väldigt

nära ändringen av SDA. För att minska risken att slaven läste av fel värde, om datan inte hann ändras helt innan klockpuls genererats, så implementerades en fördröjning i koden. Detta skedde genom att dela upp befintlig fördröjning på två där halva fördröjningen lades innan triggandet av klocksignalen och halva fördröjningen lades efter denna. Effekten blev att klockpulsen senarelades och hamnade i mitten av databiten vilket var det önskade resultatet.



Figur 4-7: I²C-kommunikation som visar att ACK tar lång tid att verifiera

4.2.4 Auto-increment

När det fungerade att ändra värden på de olika registren ett och ett provades därefter att använda funktionen auto-increment. Dess funktion var att sätta flera efter varandra följande register utan att behöva skicka registeradressen mellan varje databyte. Man behövde fortfarande skicka information om vilken adress man skulle starta på men till skillnad från ovan ändrade man de tre första bitarna i adressregistret. Den sjunde biten aktiverade auto-increment då den blev satt till ett och de två påföljande bitarna representerade vilken sorts auto-increment som användes. För att börja med auto-increment på registeradress 14 skrevs därför 94 hex istället. Därpå följde en databyte för vardera påföljande register med start på register 14. Mängden register som det skulle skickas till bestämdes på frontpanelen genom att begränsa antalet byte som skulle skickas. Så ville man med hjälp av auto-increment ändra på datan hos de fyra registren 14-17 så sattes antalet byte till fem. Den första byten talade om vad som var startregistret sedan följde datan för de fyra registren. Ett exempel på data som skickades var: 94 00 01 04 05. Här borde register 14 tilldelats värde 00, register 15 tilldelats värde 01 och så vidare.

Så var inte fallet när det testades på optokopplarna. Det gick att se att auto-increment var igång då två påföljande register ändrade värde. Det första värde som ändrades var på den

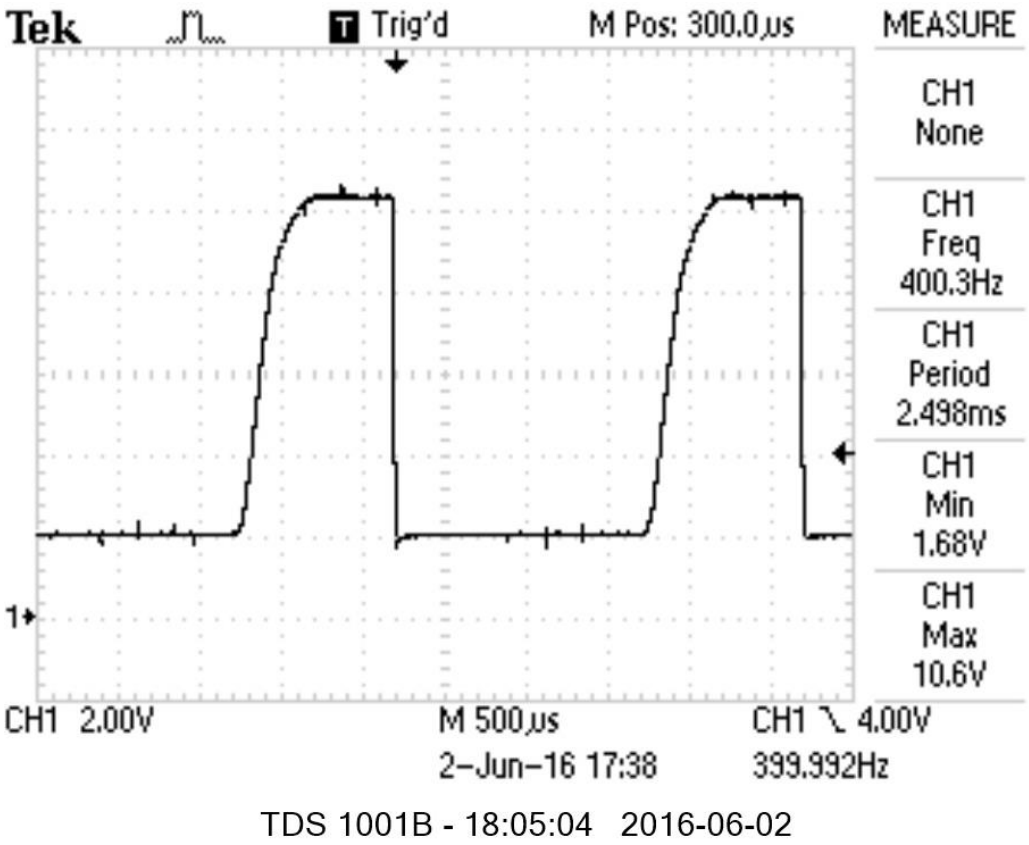
adress där auto-increment skulle starta. Det bytte även värde till det förväntade. Däremot tilldelades det efterföljande registret alltid värdet 00, oavsett vilken data man försökte skicka. Det var 00 som lades ut på bussen. Detta beteende uppstod oberoende av vilket register auto-increment startades på.

Efter granskning av koden kunde konstateras att det fanns en hårdkodad begränsning i hur många byte data som kunde skickas. Denna begränsning togs bort genom att öka mängden byte som kunde skickas. Efter att begränsningen togs bort så gick det att skicka fler byte via auto-increment. En adressbyte och därtill två databyte blev då som förväntat. Om det skickades data till fler register, d.v.s. mer än 3 byte totalt, så skickades enbart nollor till de sista registren. Det skedde oavsett vilket värde som skrevs in på frontpanelen. Slutsatsen till ovanstående upptäckt var att det med dåvarande konfiguration inte gick att använda auto-increment för att ändra värden på alla 16 optokopplare vid en sändning av data. Då denna funktion inte behövdes för att genomföra uppgiften så lämnades denna bristfälliga funktion med möjlighet att tas upp senare, om tid gavs.

4.2.5 Generering av fyrkantsvåg

När optokopplarna gick att styra var nästa steg att generera en fyrkantsvåg. Det skedde genom att använda den befintliga frekvensgeneratorn som startades när man tryckte på freq generation (se Bilaga 2 - Ny frontpanel huvudkod). För att spara tid skapades möjligheten att skriva in ny data till frekvensgenereringen samtidigt som programmet var i run-mode. Tidigare kompilerades en konstant array som överfördes till FPGA och var därmed låst till den arrayens värde. Den nya funktionen sparade mycket tid som annars hade lagts på omkompilering av nya värden som skulle köras i frekvensgeneratorn.

På frontpanelen ställdes det in att datan skulle ändras mellan 00 och 55 på registeradress 16. Sedan testades att generera en fyrkantsvåg där man använde den förinställda frekvensen för omslag mellan de två värdena. Vid mätning över mätpunkterna på resistansstegen skedde inga förändringar. Detta trots att det via oscilloskop kunde verifieras att data för omslag skickades till slaven. Felet kunde spåras till att det vid frekvensgenerering användes en hårdkodad slavadress. Den var förinställd på 23 hex. Efter att den ändrades till 60 hex, som är slavens adress, kunde en fyrkantsvåg utläsas över mätmotståndet. Den första fyrkantsvågen som genererades var inte symmetrisk då den höga delen var mycket kortare än den låga. Fördelningen var ca 1/3 mot 2/3. (se Figur 4-8)

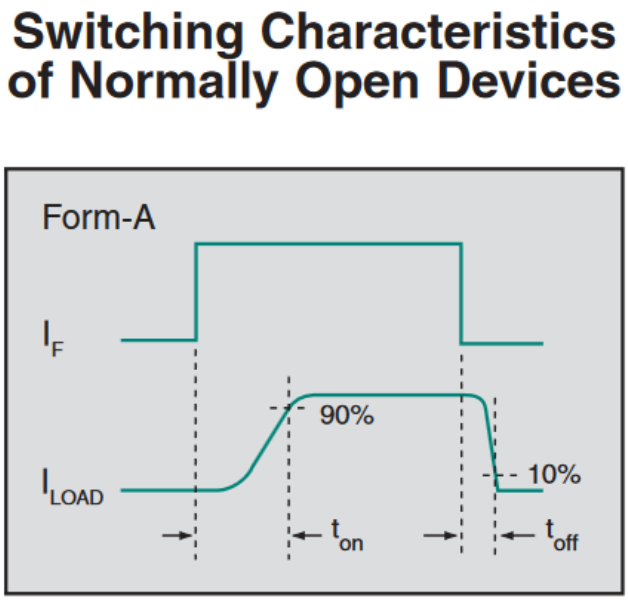


Figur 4-8: Fyrkantsvåg med 1/3 hög och 2/3 låg. Ej optimal

Vid granskning av datatrafiken på SDA konstaterades att datan för låg signal var lika lång som datan för hög signal. Skälet till den osymmetriska fyrkantsvågen fanns därmed hos optokopplarna eller hos PCA9635. Karakteristiken på fyrkantsvågen var lik den hos optokopplarna enligt Figur 4-9 nedan. De hade olika påslags- och frånslagstid. Vid slutande av kretsen förelåg en fördröjning på 0,7 ms och vid öppnande var den 0,115 ms. Detta påverkade signalen genom att dels generera en fördröjning på hela signalen på 0,115 ms samt att skillnaden mellan öppnande och stängande var enligt ekvation (4.13). Med storheterna total fördröjning t_{del}, påslagstid t_{on} samt frånslagstid t_{off}.

$$t_{del} = t_{on} - t_{off} = 0{,}7 - 0{,}115 = 0{,}585\ s \qquad (4.13)$$

Vilket ger den tid som skiljer öppnande mot slutande av optokopplarna. Granskning av den uppmätta signalen påvisade att det var en tidsfördröjning på 0,585 s som skiljde den höga delen från den låga delen på fyrkantsvågen. Därmed kunde det konstaterades att det var optokopplarna som var problemet. Den första fyrkantsvågen som skapades gav ett bra exempel på hur det kan se ut då experimentkortet är för långsamt och inte hinner med.



Figur 4-9: Omslagskarakteristik för optokopplarna

4.2.6 Insamling av spänningsvärden för alla kombinationer av optokopplarna

Resistansstegen och alla dess kombinationer genererade olika spänningar över mätmotståndet. Dessa olika spänningar var enkla att räkna fram för det teoretiska spänningsvärdet. Fast i slutändan så är det de faktiska värdena som är intressanta och inte de teoretiska. Då de faktiska värdena vid några stickprov visade sig skilja från de teoretiska bestämdes att alltid utgå ifrån faktiska värden. Så det beslöts att generera en fil där man kunde se vad alla olika kombinationer av öppna och slutna optokopplare genererade för faktisk spänning. Det innebar att testa de 2^{16} olika kombinationer av optokopplarna som var möjliga, d.v.s. 65 536 stycken och se vilka spänningsvärden de genererade. För att på ett enkelt sätt samla in de olika spänningsvärdena för alla kombinationer så skapades ett sub-VI för denna uppgift (se Bilaga 5, Bilaga 6 och Bilaga 7 för att se Blockdiagram för insamling av alla 65536 värden, samt förstorning på vänstra respektive högra delen av blockdiagrammet).

För att tillämpa denna kod krävdes dock att det på ett enkelt sätt gick att hämta in mätvärdena på spänningen över mätmotståndet. Det löstes genom att montera fast ytterligare en modul i cRIO-9074 chassit, en NI 9205. Porten som användes för insamling av spänningen var AI 31, vilken fanns på ben 37 på modulen. Utöver mätsignalen drogs även en jordsignal från COM till GNDREF. Innan mätvärdet från den nya modulen implementerades i insamlings-sub-VI så skapades först en loop som med jämna intervall läste in spänningen över mätmotståndet och presenterade det på frontpanelen. Vid jämförelse mot en multimeter som mätte spänning över mätmotståndet kunde konstateras att värdet som lästes av i LabVIEW från den nya modulen var adekvat.

Ett andra oscilloskop kopplades in för att mäta både I^2C bussens två signaler och spänningen över mätmotståndet samtidigt. Ett litet problem i form av konstiga mätvärden i detta skede kunde relateras till en dålig oscilloskopprob. De konstiga mätvärdena försvann när denna byttes ut. När signalerna såg normala ut så implementerades inläsningen från den nya modulen in i sub-VI för datainsamlingen. När detta var klart fungerade sub-VI för datainsamling såhär: Vid varje skifte på någon optokopplare mättes spänningen över mätmotståndet och därefter sparades detta värde över till en textfil. Därefter upprepades detta

tills man gått igenom alla olika kombinationer på optokopplare som fanns. Även informationen om hur de olika registren för optokopplarna var inställda vid varje separat tillfälle sparades till textfilen tillsammans med spänningsvärdet för den kombinationen. Från början fungerade inte inskrivningen till textfilen som den skulle då den istället för att lägga till nya värden på en ny rad istället skrev över den befintliga raden. Detta gjorde så det enda som kunde utläsas ur textfilen efter att man exekverat sub-VI var det senaste värdet som skrivits till filen. Detta problem åtgärdades genom att ändra i inställningen på pekaren som pekade ut var nästa data skulle skrivas. Den var först inställd på "current" men ändrades sedan till "end". Efter denna förändring fungerade skrivandet till textfilen och alla värden skrevs därefter in efter varandra på en ny rad. Ett annat fel som uppdagades vid genomläsning av textfilen var att ett av registren inte var inskrivet i rätt sorts talsystem. Textfilen gjorde om de inskrivna värdena på registren från det hexadecimala- till det decimala talsystemet. En av de hårdkodade registeradresserna var inskriven i en variabel som använde det decimala talsystemet och inte det hexadecimala. Detta fick till följd att ingen av de optokopplare som skulle styras av det registret fick någon information om omslagningar. Efter att detta upptäckts och justerats exekverades sedan sub-VI för att samla datainformationen. Det tog ca 10 minuter att skriva ner alla 65 536 olika kombinationer till textfilen.

När väl alla kombinationer var genomkörda och sparade kunde textfilen sorteras med hjälp av Excel. Först låg all data i textfilen den turordning som sub-VI hade testat dem. Därefter sorterades datan stigande utifrån det uppmätta spänningsvärdet i Excel. När värdena sorterats sparades datan åter som en textfil. Denna fil användes sedan för att se hur optokopplarna skulle vara inställda för att kunna generera de spänningsvärden som önskades.

4.2.7 Styrning av optokopplarna från en data-array

För att kunna generera en sinusvåg i den befintliga koden så blev man tvungen att lägga till en ny funktion för detta (se Bilaga 3 – Nytt blockdiagram huvudkod, för hela huvudkoden och Bilaga 4 – Blockdiagram sinusvåg, för just sinusvågen). Funktionen startades genom att man tryckte på en knapp på frontpanelen. Därefter lästes en hårdkodad array in, rad för rad. Varje rad hade fyra byte och varje byte fördelades till vardera register som styrde optokopplarna. När de fyra byten skrivits till de fyra registren ändrades alla 16 optokopplare till det nya värdet de blivit tilldelade. Funktionen fick en styrbar delayfunktion där man kunde påverka tiden mellan inskrivningen till de olika registren så det gick att skicka värden olika snabbt. Hela funktionen lades i en loop så att uppläsningen av arrayen började från början igen när alla rader i arrayen var inlästa.

Innan en sinusvåg försökte genereras gjordes först ett försök att använda ovanstående kod till att rampa upp mätsignalen linjärt. I och med att koden loopades skulle spänningen över mätmotståndet generera en signal som hade formen av en sågtandsvåg. Funktionen var väldigt lik det planerade genererandet av en sinusvåg och ansågs därför vara ett bra första steg. För att få lämpliga värden till arrayen utgick man ifrån textfilen med de olika uppmätta och sorterade spänningarna. Härifrån togs inställningen för optokopplarna, utifrån önskad spänning. Värdena valdes med jämna intervall på 0,25 V. Viss avrundning tillämpades för att finna det värde som var närmast önskad spänning. Första inställningen var minimivärdet på 1,95 V. Därefter följde jämna intervall med start på värdet 2 V, nästa värde var 2,25 V därefter 2,5 V osv hela vägen upp till maxspänningen på ca 9,99 V. Efter att alla inställningar till optokopplarna skrivits in i arrayen testades att köra koden. Genom att koppla oscilloskopet över mätmotståndet kunde därefter konstateras att en sågtand hade genererats.

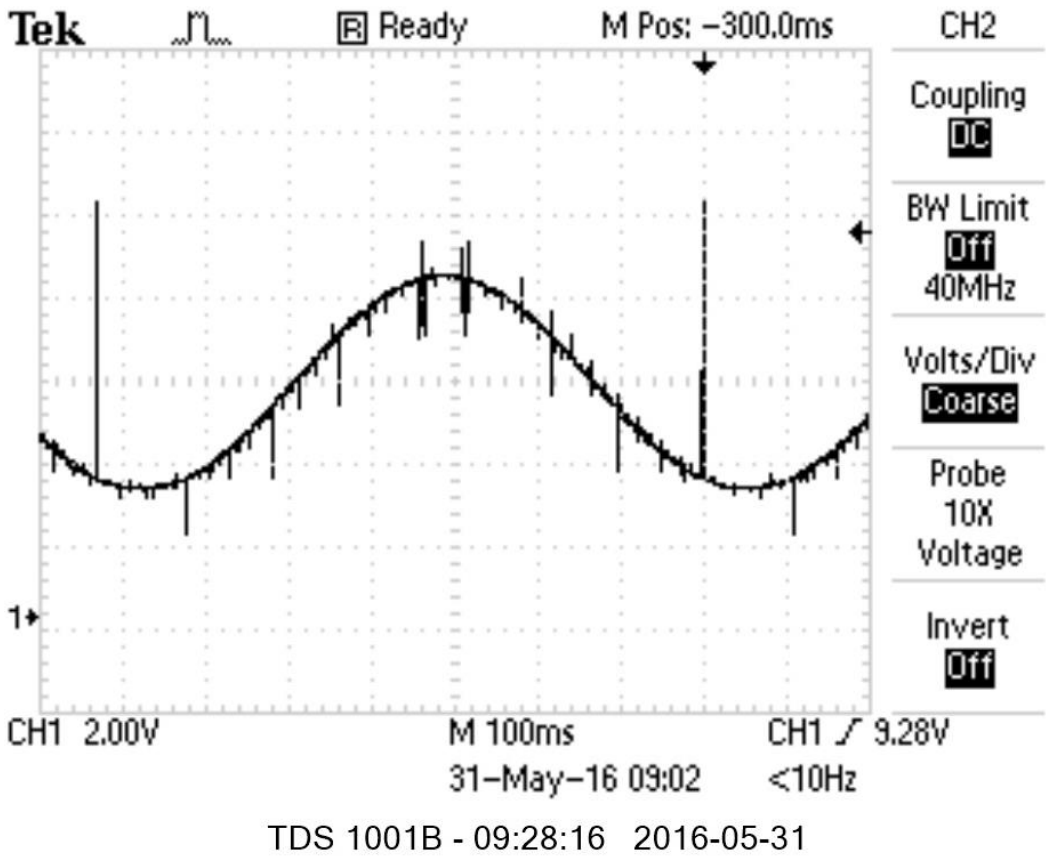
4.2.8 Generering av sinusvåg

Med en konstant tid mellan skrivningarna till optokopplarna krävdes unika värden i följd för att generera en sinuskurva. Derivatan förändras över tid, vilket medför att man inte kan stega igenom en array likt sågtandsvågen. Det första som behövde göras var att simulera en sinussignal för att ta reda på vilka värden i följd som genererar en sinusvåg. Här utnyttjades en av de inbyggda funktionerna i LabVIEW, nämligen deras funktion för sinussimulering. Genom att ställa in amplitud, fasförskjutning, offset och antal sampels i ett färdigt sinus-VI så fick man ut olika värden motsvarande den spänning som önskades få ut på mätmotståndet. Dessa värden jämfördes därefter mot alla spänningsvärden i textfilen. Utifrån vilket värde som var närmast i textfilen så hämtades inställningarna för optokopplarna för denna spänning. Det lades sedan in i en array som fylldes på allt eftersom genereringen av sinusvågen pågick.

När detta var klart gjordes justeringar i huvudkoden så att det gick att läsa in sinusarrayen direkt från det VI som genererade arrayen utan att man behövde skriva in något för hand som tidigare. En stor fördel i samband med denna förändring var att man slapp kompilera koden mellan ändringarna av värdena. En annan var att man eliminerade risken för felskrivningar som den mänskliga faktorn kunnat bidra med vid en eventuell kopiering av värden från ett ställe till ett annat.

Det behövdes även införas ändringar så att den inlästa arrayen först lästes uppifrån och ned och därefter åt andra hållet, nedifrån och upp, varannan gång. Detta så att en hel sinusvåg kunde genereras. Skälet till vändningen berodde på att de inlästa värdena från sinusgenereringen enbart inhämtats från botten av en sinusvåg till nästa topp.

Det hade varit möjligt att läsa in en hel sinusvåg så man sluppit vända på inläsningen av arrayen. Men då cRIO med dess FPGA har begränsat med lagringsutrymme var det bättre att få med många olika värden för större noggrannhet. Vid en hel sinuskurva skulle hälften av värdena bara varit dubbletter, så noggrannheten hade halverats på samma mängd data. När alla förändringar gjorts kördes koden för att testa en sinusvåg. Den mättes över mätmotståndet och visualiserades på oscilloskopet. Här nedan är en figur på hur det såg ut när man använde en delay på 1000 µs mellan de olika registren till optokopplarna.



Figur 4-10: Sinusvåg med stor pik. En delay på 2000 µs används.

Som framgår av bilden ovan så uppstod det vissa oönskade dippar och pikar. Den största av dessa gick upp och tangerade maxvärdet för kretsen. Detta trots att sinusvågen var inställd på att verka mellan 3-8 V. Det fanns misstankar om att det skulle uppstå vissa dippar då optokopplarna öppnades snabbare än de slöts. Vilket medförde att spänningen under kort tid kunder gå ner innan nästa optokopplare hann slutas. Däremot tog det ett tag att lokalisera orsaken till den stora piken. En närmare granskning gjordes på de värden som skickades till optokopplarna där piken genererades. För de övre registren 16 och 17 såg värdena ut såhär:

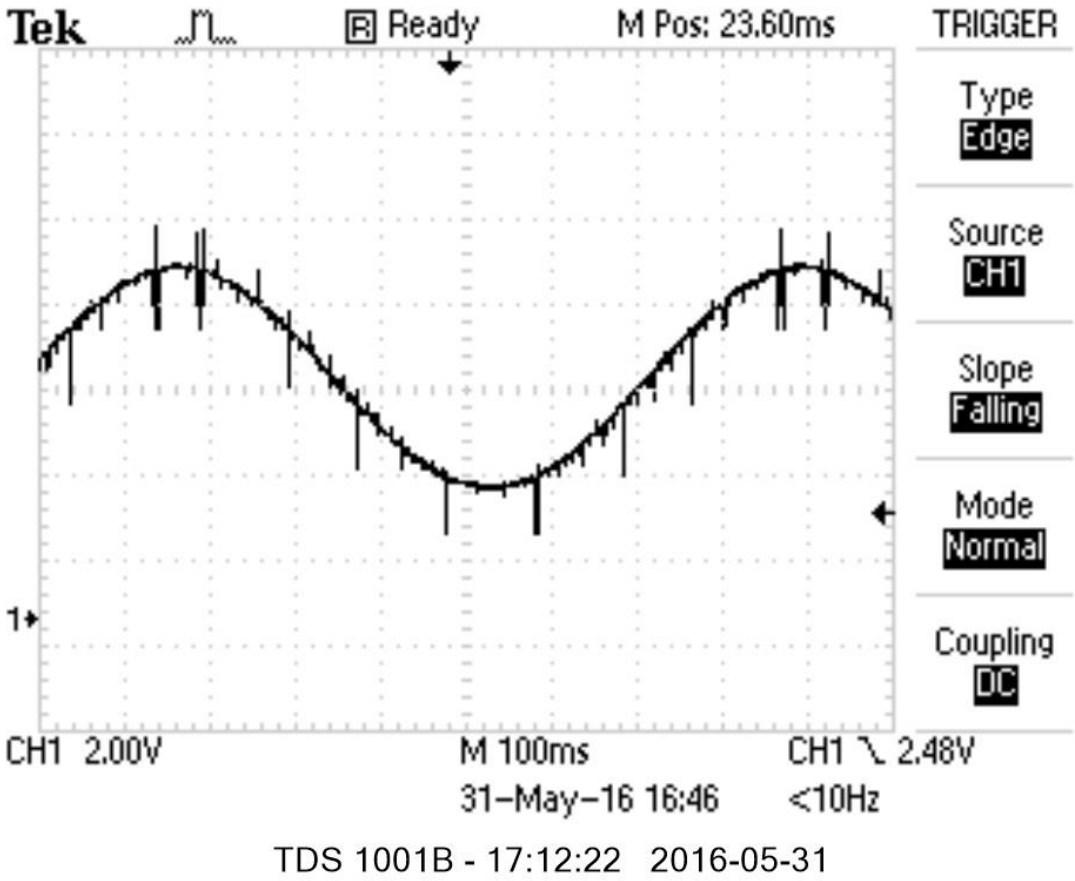
16: 0000 0000 17: 0000 0001, detta ersattes därefter med:
16: 0101 0101 17: 0000 0000

De gråmarkerade nollorna styrde enbart de förbikopplande optokopplare så de kunde man bortse ifrån här. Däremot styrde de övriga siffrorna om man skulle göra förbikoppling eller ej av de resistorer med störst resistans i resistansstegen (se Bilaga 1 - Kretsschema). Ettan längst till höger i register 17, den minst signifikanta biten, kopplade förbi resistorn med värdet 2048 Ω i resistansstegen. Då register 16 ändrades före register 17 samt att det fanns en kort fördröjning mellan skrivningarna till de två registren hade register 16 och 17 en kort stund dessa värden:

16: 0101 0101 17: 0000 0001

Register 16 har hunnit slå om till det nya värdet medan register 17 har kvar sitt gamla värde. Detta innebär att man kopplar förbi alla resistorer med hög resistans samtidigt. Det är detta som ger upphov till (piken) på sinuskurvan. Även de övriga pikarna som föreligger har samma sorts problematik även om deras värden inte genererar riktigt lika stora pikar.

Genom att ändra i arrayen så att ovanstående kombination av register 16 och 17 inte är kvar kunde nedanstående sinusvåg genereras (se Figur 4-11). Som synes uppstod en dipp istället för tidigare pik men den har en betydligt mindre amplitudförändring än piken i förhållande till sinusvågen. Problemet var inte borta, men vågen såg bättre ut.



Figur 4-11: Sinusvåg utan stor pik. En delay på 2000 µs används.

Efter att sinusvågen var genererad och den värsta piken avhjälpt genomfördes inga fler förbättringar eller implementationer i koden.

4.3 Testutförande

Testningen av utrustningen beslöts att ske enligt ett bestämt förlopp. Detta för att kunna återupprepa de tester som genomförts och då kunna få samma resultat. Det som skulle bestämmas vid testerna var maximal frekvens på en fyrkantsvåg och en sinusvåg då de fortfarande ansågs vara acceptabla.

Materialet man skulle utgå ifrån när man bedömde vad som var acceptabelt kom från de värden som uppmättes över mätmotståndet med ett oscilloskop. För att få en tidsbedömning på mätningarna användes oscilloskopets inbyggda funktion för tidsperiodmätning. För att kunna komma upp i högsta möjliga hastighet så skulle all kod köras från FPGA för att slippa den fördröjning som uppstår om man kör i RT, eller i Windows.

4.3.1 Initiering

- Koppla upp resistansstegen till 10 V
- Koppla upp PCA9635 kretsen till 5 V

- Koppla upp ett oscilloskop över mätmotståndet på experimentkortet
- Tillgodose att det är fungerande kommunikation mellan cRIO-chassit och experimentkortet
- Ställ på frontpanelen in:
- *Action* - Write
- *Addressing Scheme* - 7-bitars adress
- *Receiver Address* - 60
- *Number of Bytes* - 2
- Skicka 00 00, 01 09 till PCA9635 för att säkerställa att den är rätt initierad
- Skicka 14 00, 15 00, 16 00, 17 00 till PCA9635 för att nollställa alla optokopplare. Detta göra att strömmen går igenom alla resistorer i resistansstegen.

4.3.2 Fyrkantsvåg

- Initiera enligt ovan
- Ställ in *Number of Bytes freq* till 2
- Skriv in 17 på första och 00 på andra raden i *Data to send freq true*
- Skriv in 17 på första och 55 på andra raden i *Data to send freq false*
- Ställ in *Mode* utifrån vilken hastighet som ska testas, Standard eller High speed mode
- Bestäm ett startvärde i *Frequency (periods/tick)*
- Påbörja generering av fyrkantsvåg genom att trycka på *Freq generation*
- Om den höga delen ej är platt, minska frekvensen så den blir platt
- Bestäm maxvärdet på vågen
- Ställ in oscilloskopet så 2 cykler syns
- Bestäm periodtiden på fyrkantsvågen
- Bestäm längden på delen av fyrkantsvågen som är hög (vågen anses hög när den nått 90 % av maxvärdet)
- Dela längden på fyrkantsvågens höga del med halva periodtiden. Detta ger vågens korrelationsvärde, mot en optimal fyrkantsvåg, i procent.

Ett korrelationsvärde på 100 % är detsamma som optimal fyrkantsvåg. Värdet minskade vid en ökad frekvens.

Om korrelationsvärdet befinner sig mellan 0 och 50 procent minskas frekvensen och upprepa de fyra sista punkterna tills man kommer upp i 50 procent.

Om korrelationsvärdet är större än 50 procent ökas frekvensen och därefter upprepas de fyra sista punkterna i listan till dess man kommer till 50 procent.

Frekvensen som uppmäts när skillnaden mellan hög och låg frekvens ligger på 50 procent är den snabbaste frekvens som går att använda för denna hastighet (I^2C-mode).

Desto färre antal perioder per tick som används, desto snabbare fyrkantsvåg genereras. Proceduren med att komma fram till högsta godkända frekvens ska genomföras med två olika I^2C-mode, Standard (100 kHz) och High-speed (3,4 MHz).

4.3.3 Sinusvåg

- Initiera enligt ovan
- Kontrollera att sinusgenereringskoden hämtar värdena till sin array från rätt filen
- Ställ in vilken hastighet som ska testas, Standard- eller High-speed mode
- Ställ in önskat värde på delayen i *Count(µSec)*

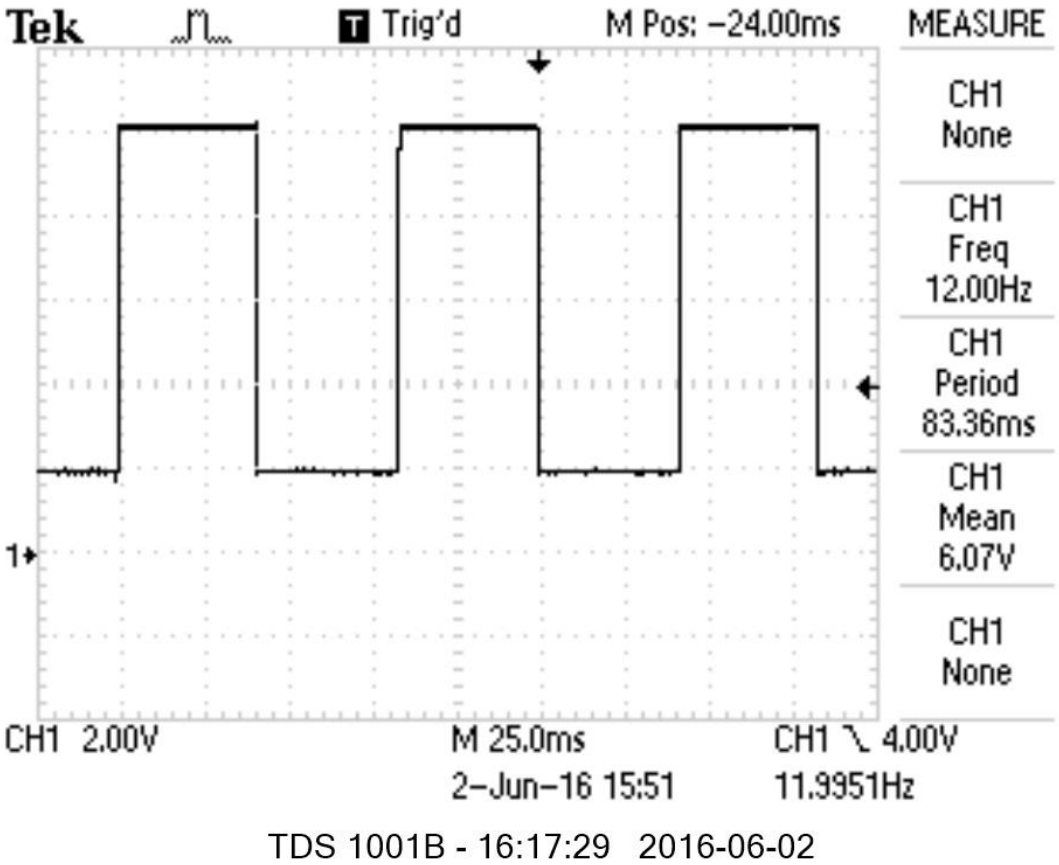
- Ställ in (databyte) *Sine array size* i sinusdelen på frontpanelen till 100 på sine-generation
- Ställ in oscilloskopet så att två perioder syns

Då det aldrig beslöts vad som ansågs vara en godkänd avvikelse ingår inte detta i testförfarandet för sinusvågen. Se diskussionen för mer information om detta.

Värt att nämna är dock att frekvensen på sinuskurvan inte korrelerar med den uppmätta frekvensen för fyrkantsvågen. En periodtid för fyrkantsvågen bestod av två omslag på optokopplarna och förändringarna skedde bara i ett register. Motsvarande siffra hos sinusvågen är 200 omslag med ändringar i datan hos alla fyra register.

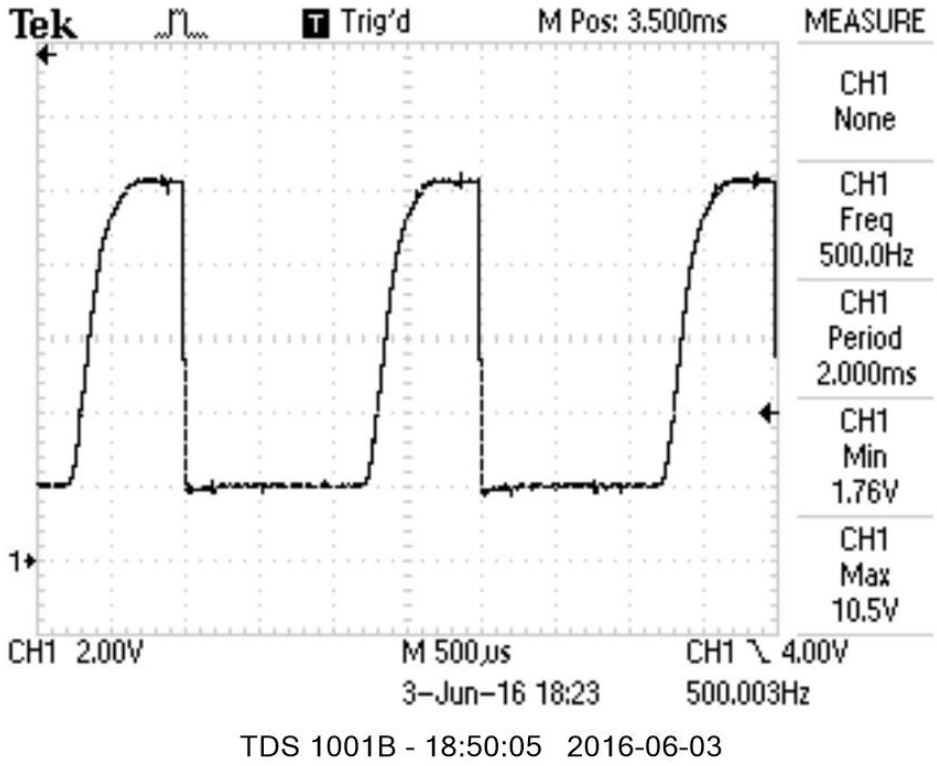
5 RESULTAT

I detta kapitel kommer resultatet av de vågor som genererats redovisas. Föregående kapitel 4.3 Testutförande beskriver just testutförandet som användes för att generera nedanstående vågor. De vågor som genererats är fyrkantsvåg, sågtandsvåg samt sinusvåg.



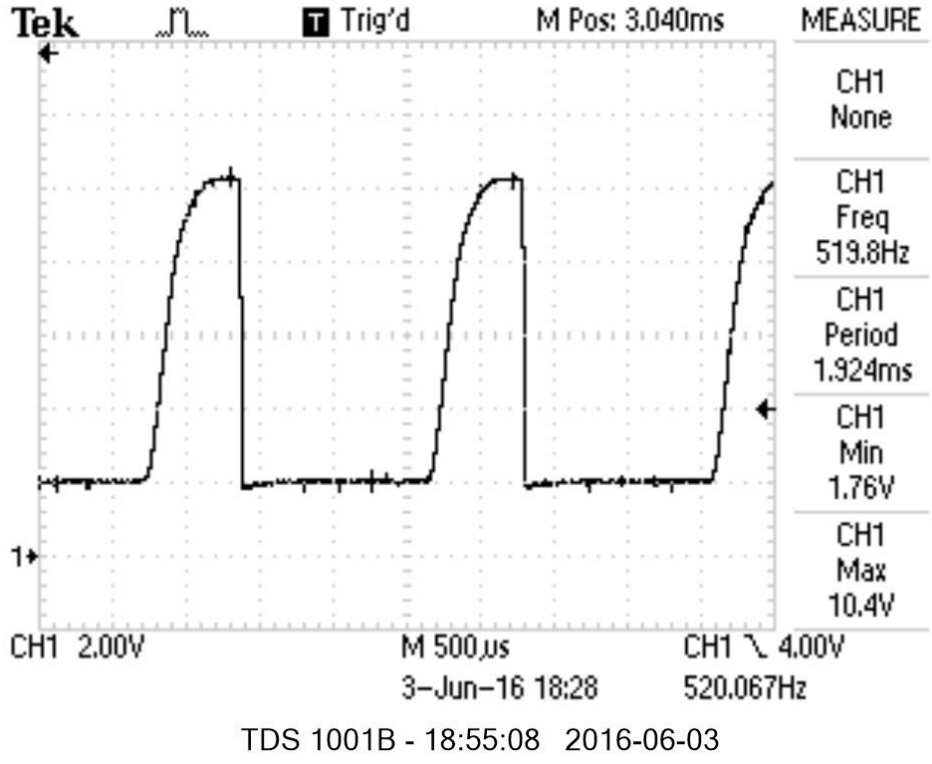
s 5-1: Fyrkantsvåg som nästan är optimal. Frekvens på 12 Hz

s 5-1 är väldigt nära en optimal fyrkantsvåg, korrelationsvärdet är 100 %. Det gjorde tester både med Standard- och High-speed mode, vilket inte gjorde någon skillnad. Bilden är tagen med Standard mode och frekvensen är 12,0 Hz.



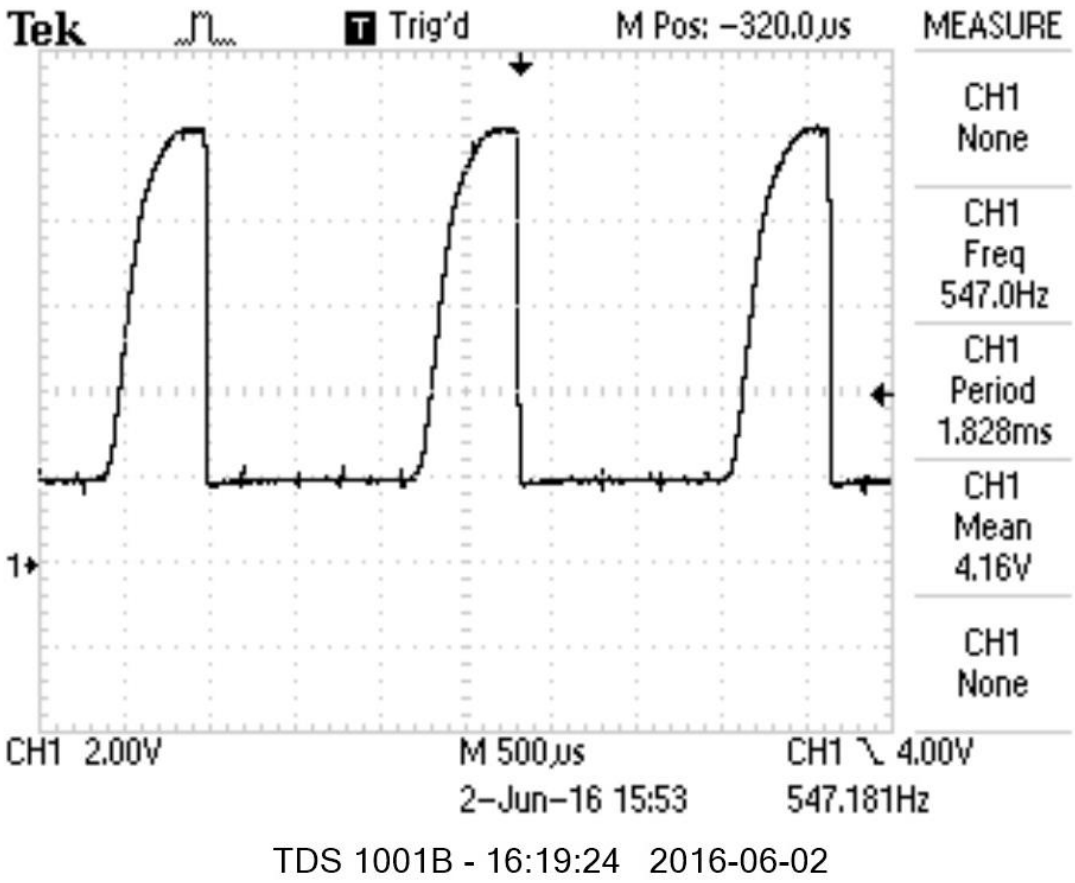
Figur 5-2: Snabbaste fyrkantsvågen som är acceptabel med Standard-mode. Frekvens på 500,0 Hz

Den högsta frekvens som uppnåddes för en acceptabel fyrkantsvåg var 500,0 Hz för Standard-mode. Vågen presenteras i Figur 5-2. Korrelationsvärdet är 50 %.



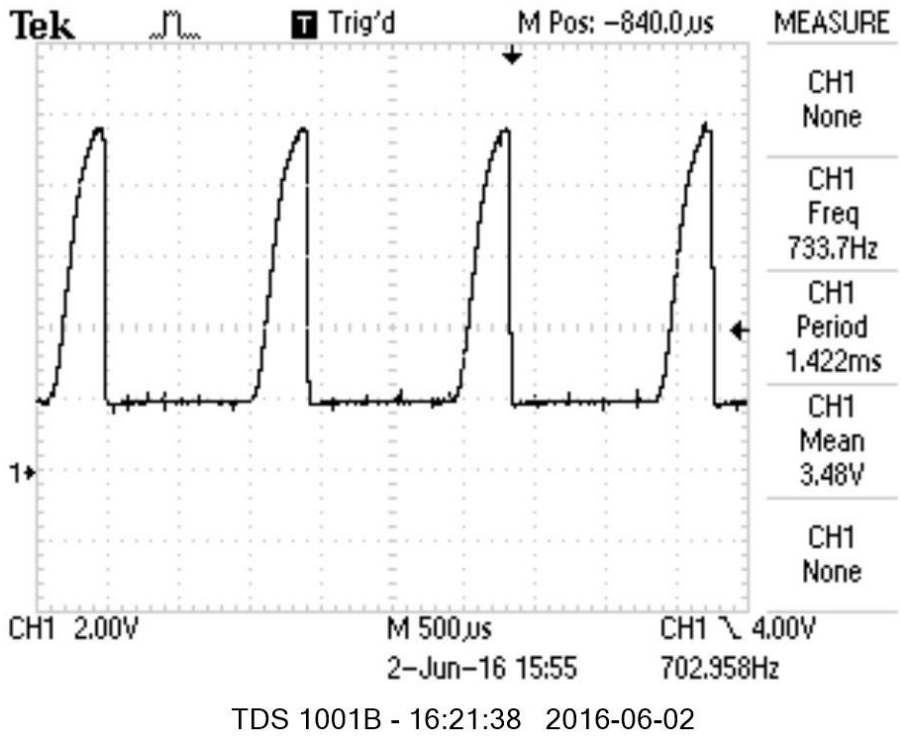
Figur 5-3: Snabbaste fyrkantsvågen som är acceptabel med High-speed mode. Frekvens på 519,8 Hz

Högsta frekvensen som uppnåddes med en acceptabel fyrkantsvåg var 519,8 Hz med High-speed mode. Vågen presenteras i Figur 5-3. Korrelationsvärdet är 50 %.



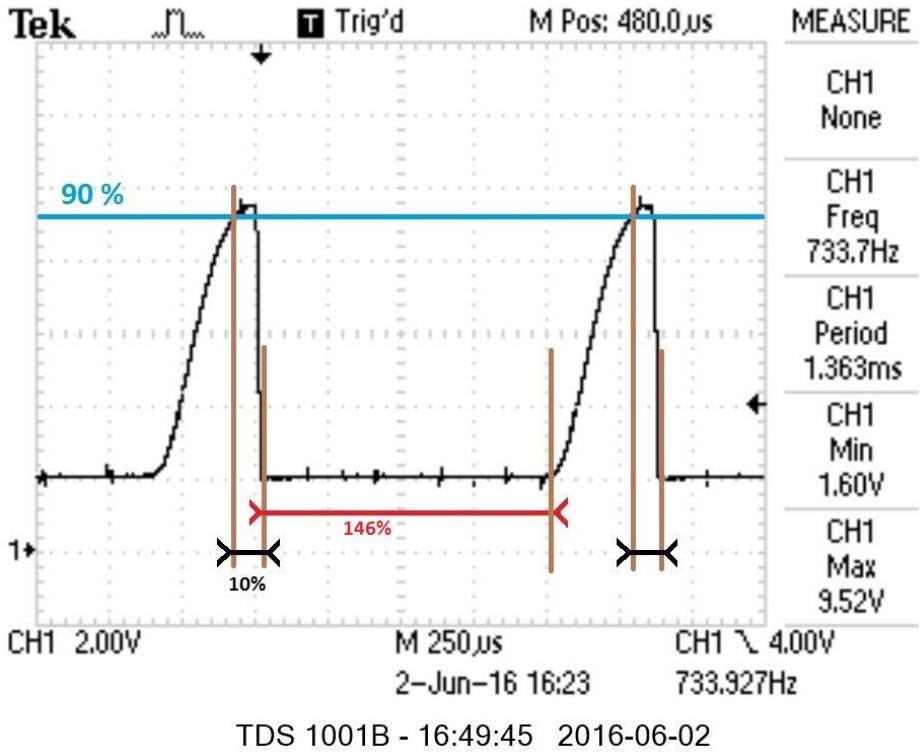
Figur 5-4: Fyrkantsvåg som inte är optimal men stabil. Frekvens på 547,0 Hz

Som man kan se på Figur 5-4 är det inte en optimal fyrkantsvåg. Men pulserna kommer kontinuerligt och vågen är stabil. Minsta värdet är 2,0 V och högsta värdet är 9,9 V. Optokopplarna hinner med andra ord slutas helt innan de ändrar värde igen. Vågen generas med Standard-mode. Frekvensen var 547,0 Hz.



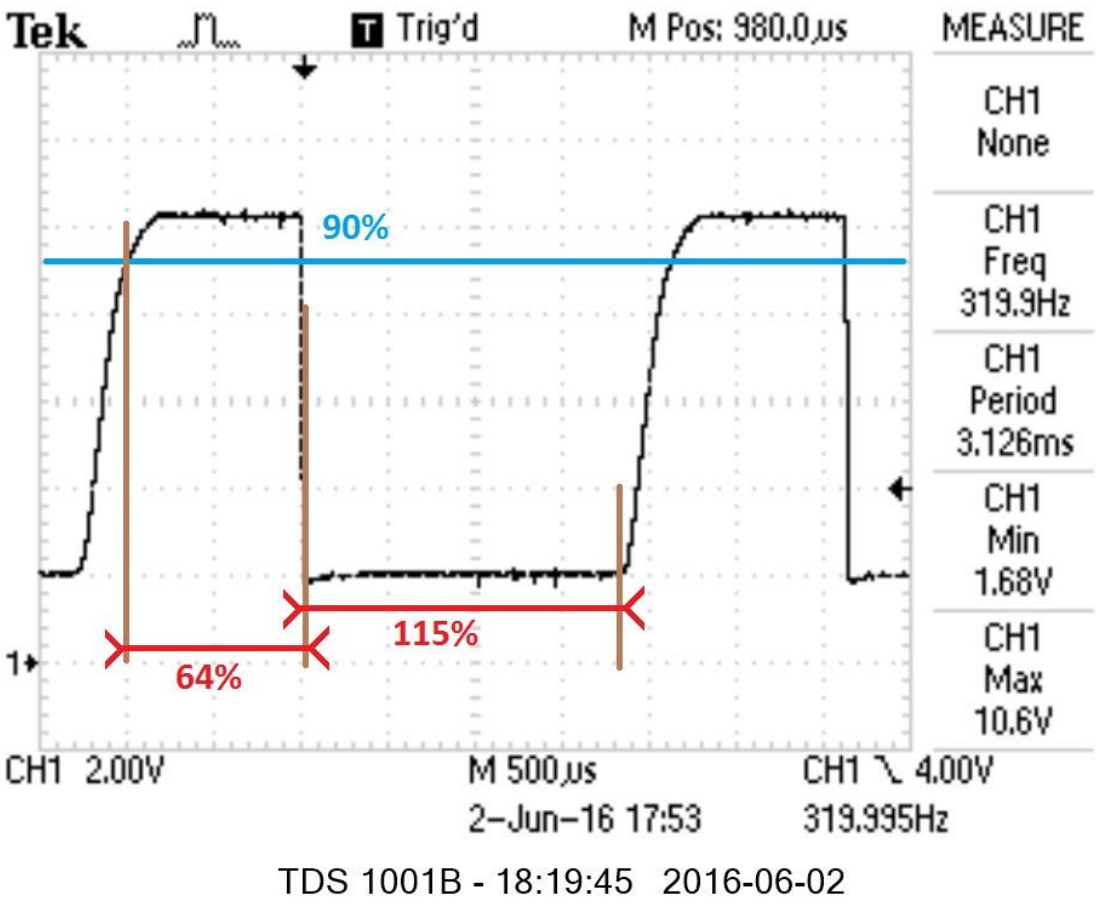
Figur 5-5: Fyrkantsvåg som inte är optimal men stabil. Maximala frekvensen på 733,7 Hz

Figur 5-5 visar även den en inte optimal fyrkantsvåg. Men pulserna är stabila och kommer kontinuerligt. Minsta värdet är 2,0 V men högsta värdet är 9,8 V. Optokopplarna hinner alltså inte slutas helt innan de slår om. Den typiska karakteristiken för optokopplarnas tillslag och frånslag enligt Figur 4-9 är precis vad som visas på tillslaget. Vågen genereras med High-speed mode. Frekvensen som maximalt uppnåddes var 733,7 Hz.



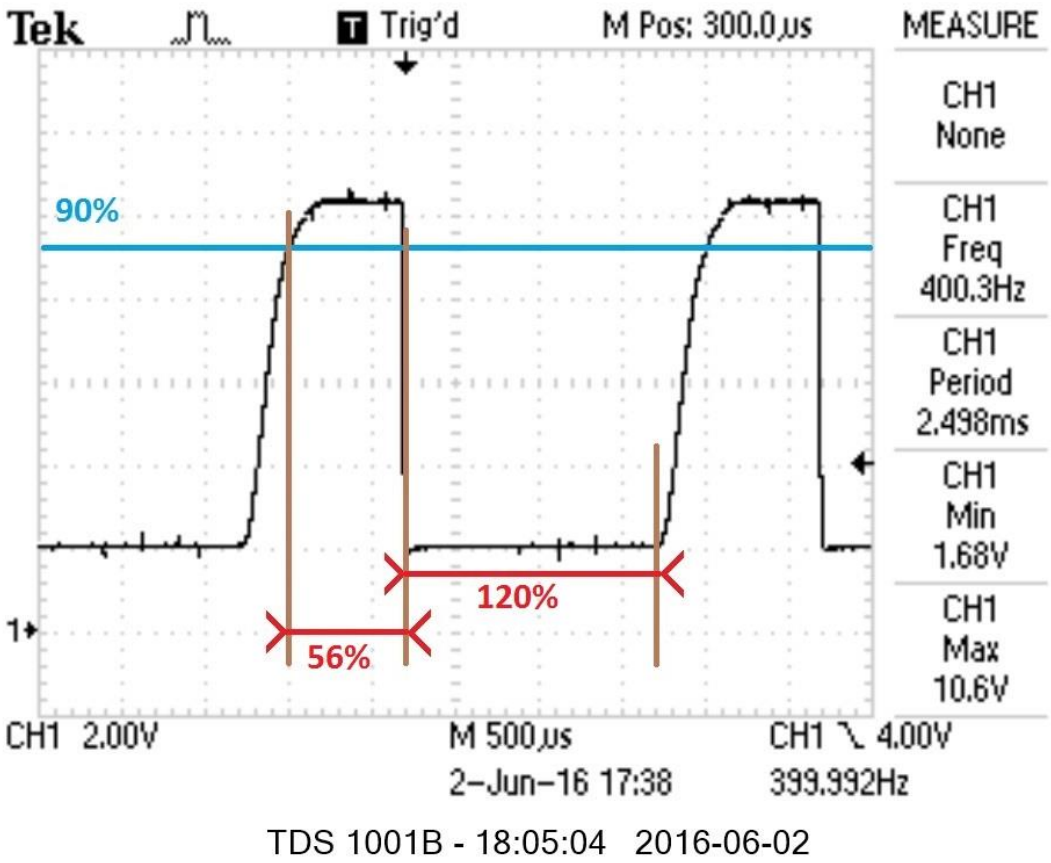
Figur 5-6: Samma våg som Figur 5-3 men med utritat korrelationsvärde på 10 %

Med en tidsaxel med kortare tidsspann kan parametrarna för den snabbaste kurvan bestämmas. Vilket har genomförts i Figur 5-6 ovan som är samma fyrkantsvåg som i Figur 5-5. Där kan man avläsa att signalen har ett korrelationsvärde för hög signal på 10 % i förhållande till en optimal fyrkantsvåg. Den har ett korrelationsvärdevärde för låg signal på 146 % i förhållande till en optimal fyrkantsvåg. Frekvensen är på 733,7 Hz.



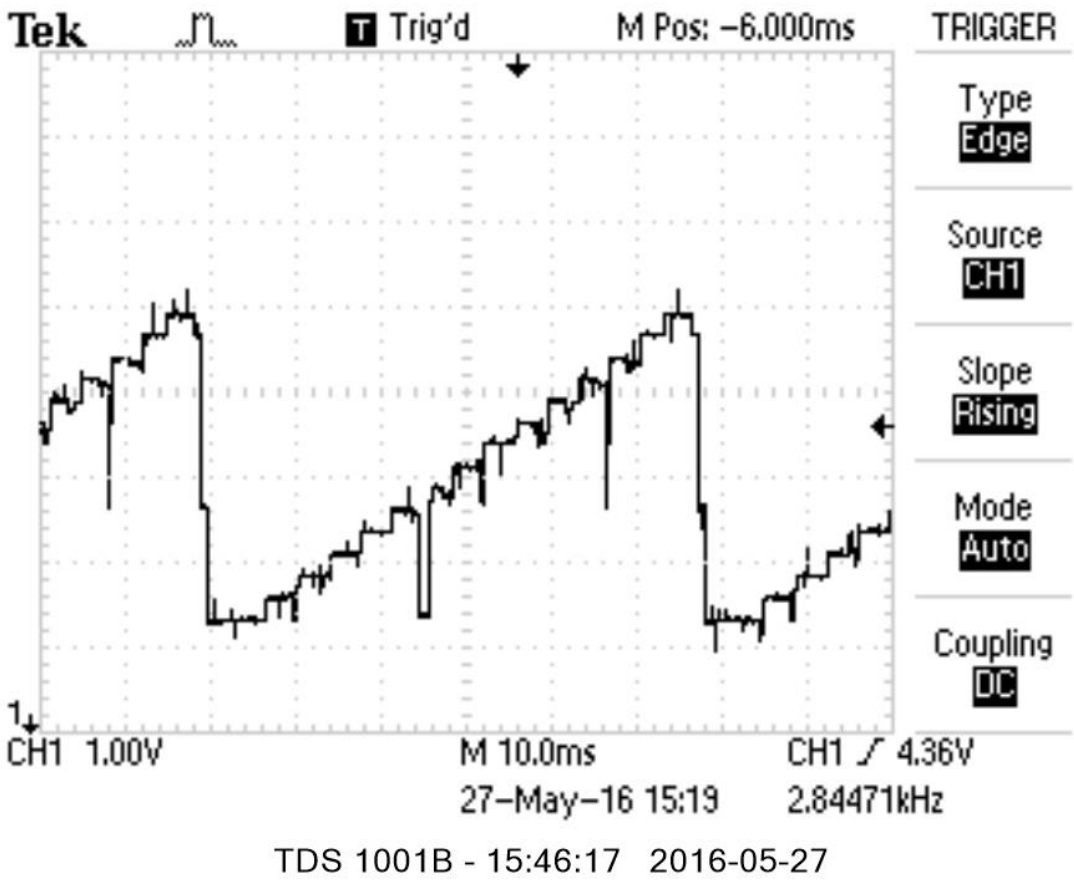
Figur 5-7: Fyrkantsvåg med 64 % korrelationsvärde. Frekvens på 319,9 Hz

Korrelationsvärdet för Figur 5-7 är 64 % och inte acceptabelt. Frekvensen måste sänkas för att generera en acceptabel fyrkantsvåg. Här ses även att den undre delen är 115 % av en optimal fyrkantsvåg, vilket ökar ännu mer om frekvensen höjs.



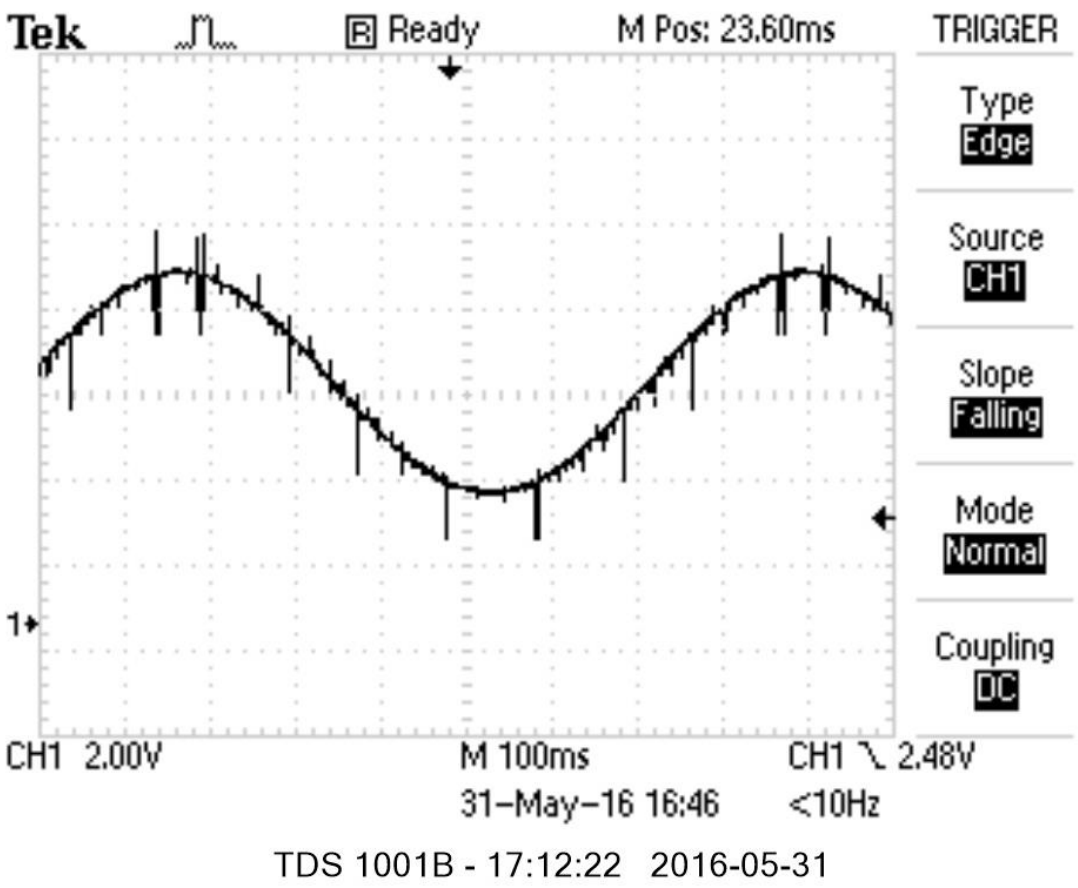
Figur 5-8: Fyrkantsvåg med ett korrelationsvärde på 56 %. Frekvens på 400,3 Hz

Figur 5-8 visar en fyrkantsvåg med ett korrelationsvärde på 56 %. Den undre delen är här 120 % av en optimal fyrkantsvåg. Frekvensen är 400,3 Hz och vågen genereras med High-speed mode.



Figur 5-9: Sågtandsvåg med en frekvens på 172 Hz

Även en sågtandsvåg genererades under uppdragets gång. Figur 5-9 visar hur den såg ut. Den stegades från 2 V upp till 10V med 0,25 V steg samt de två lägsta och högsta värdena, 1,95 V respektive 9,99 V. Frekvensen var 172 Hz.



Figur 5-10: Sinusvåg utan stor pik. Frekvens på 1,42 Hz

Sinusvågen genererades enligt Figur 5-10. Frekvensen på den är lägre än förväntat. 200 olika sampels användes för en period. Pikar och dippar förekom. Frekvensen som uppmättes var 1,42 Hz och genererades med Standard-mode. High-speed gjorde ingen förändring på hastighet eller kvalitet.

Enligt kriterierna för testresultaten presenterade i kapitlet ovan kom vi fram till denna maximala frekvens:

- 519,8 Hz för en acceptabel fyrkantsvåg med High-speed mode.
- 500,0 Hz för en acceptabel fyrkantsvåg med Standard-mode.
- 733,7 Hz för en icke godkänd fyrkantsvåg med High-speed mode.
- 547,0 Hz för en icke godkänd fyrkantsvåg med Standard-mode.
- 1,42 Hz för en sinusvåg, oberoende av mode.

6 DISKUSSION

Under uppdragets gång så har ett experimentkort med en styrbar resistansstege konstruerats. Det fanns ett mål om att kostnaden för experimentkortet med alla dess komponenter skulle kosta mindre än 500 kr men det målet uppnåddes inte. Priset hamnade snarare på det dubbla. Detta berodde främst på de dyra optokopplarna. Att beakta är dock att om man skulle börja serieproducera dessa kort så skulle man få ner priset ordentligt för att komponenterna skulle köpas in i större kvantiteter.

Experimentkortet gick att styra från en frontpanel i LabVIEW för att ställa om optokopplarna så att olika spänningar kunde genereras över ett mätmotstånd. Dessa spänningar gick att variera över tid. Det gjorde att man kunde generera fyrkantsvågor, sågtandsvågor och sinuskurvor. Utifrån resultaten på de fyrkantsvågor och sinusvågor som genererats så har man kunnat uppskatta bestämma kretsens begränsningar. Istället för att skriva in värden för en sinusvåg i arrayen vid generering kan man skriva in värden för valfri vågform som motsvarar ett verkligt förlopp. Utifrån de begränsningar i frekvens som uppmättes på fyrkants- och sinusvågen går det att få en indikation på hur snabba verkliga förlopp som går att efterlikna med resistansstegen.

6.1 Slutsats fyrkantsvåg

Som man kan se i resultatkapitlet så kunde en acceptabel fyrkantsvåg under rådande förhållanden och testmetoder uppnå en maximalfrekvens på 733,7 Hz. Det anses troligt att samma resultat skulle framkomma om studien skulle upprepas, om samma komponenter och kod användes. Däremot så finns det många områden där man skulle kunna påverka resultaten. Här nedan presenteras förslag på saker det kunde lagts mer tid på och hur man kunde ändrat gränser för att få ett bättre resultat.

Ett område där man enkelt kunde påverka resultatet på hastigheten fanns i testkapitlet. Om man istället för nuvarande procentsats för vad som ansågs vara en acceptabel fyrkantsvåg bytte ut detta mot ett annat värde skulle man få ett annat resultat. Som framgått i rapporten så var det optokopplaren som, med god marginal, var den komponent som främst påverkade hastigheten på fyrkantsvågen negativt. Här kunde det ha lagts ner mer kraft på att försöka hitta en bättre komponent för syftet. Till exempel borde man kanske frångått önskemålet om att använda en MOSFET till förmån för att hitta en komponent som var snabbare. Sedan finns det säkert MOSFET-optokopplare som är snabbare än den som användes här. Fast det hade krävt att mer tid lagts ned på att söka efter den. Det skulle även hjälpt om man varit mer bevandrad i hur man söker komponenter på internet, eller vilka andra källor man borde gått via.

Så för att kunna förbättra resultatet på en fyrkantsvåg vid en kommande studie bör man som första åtgärd försöka hitta en snabbare motsvarighet till nuvarande optokopplare. En viktig aspekt att tänka på är dock att ersättningskomponenten fortfarande bör fungera att använda vid likström om man inte vill ändra konstruktionen i grunden och att den ska ha en låg inre resistans. Det sistnämnda för att inte påverka värdena på resistansstegen för mycket. Däremot kanske det kan vara värt att göra vissa avkall på den inre resistansen till förmån för att kunna få ett snabbare resultat.

6.2 Slutsats sinusvåg

Ett stort problem vid genereringen av sinusvågen var att det med nuvarande kod inte gick att köra så fort så att sinusvågen upphörde vara en sinusvåg. Det uppstod inga stora hopp på

kurvan eller andra sorters avbrott. Utöver de få lokala toppar och dippar som förekom var sinuskurvan alltid stabil. Detta trots att fördröjningen innan man skickar nästa omgång värden till optokopplarna kortades ned till noll. Det var denna avsaknad utav ostabila kurvor som gjorde det svårt att bestämma testutförandet.

Då det inte fanns något exempel på hur en dålig sinuskurva såg ut var det mycket svårt att bestämma var gränsen för en godkänd eller inte godkänd kurva skulle gå. I brist på detta så beslöts att inte bestämma några begränsningar på avvikelser för sinuskurvan. Inte heller de befintliga topparna och dipparna fick några bestämda värden för när de skulle vara godkända eller inte. Det berodde främst på att de var likadana oavsett vilken frekvens sinusvågen kördes i. Då det är begränsningar som uppstår på grund av ökad frekvens som är av intresse här så bortsågs från dessa då de inte påverkades av rådande frekvens.

Då det inte gick att få kurvan att anta en dålig form med samma omslagshastigheter som användes när fyrkantsvågen testades tyder det på att det som främst hämmade sinuskurvan från att kunna gå snabbare inte var omslagstiden på optokopplarna.

Om man tittade på SDA vid skickande av data på bussen så syns tydligt att den del av datatrafiken som tar längst tid är verifieringen av ACK (se Figur 4-7). Det beror till stor del på att modul NI-9401 behöver byta håll på SDA från utsignal, till insignal och sedan tillbaka igen. Begränsningen i hastighet här beror på modulen då det tar tid för den att slå om mellan in- och utgång. Det bör dock gå att komma runt detta problem om man ställer om kommunikationen till burst-mode. Det innebär att man inte inväntar någon ACK utan att man bara kör på med förväntningen att allt är bra. Om man använder sig av detta bör man kunna skicka mer data snabbare. Det skulle kunna göra omkopplingen mellan optokopplarna snabbare så att man slipper de toppar som förekommer på sinuskurvan. Det borde även göra det möjligt att skicka tätare data för uppdatering av värdena på sinuskurvan. Vilket bör öka dess frekvens så en snabbare maxfrekvens uppnås. En sak som aldrig fungerade ordentligt var möjligheten att dra nytta av auto-increment-funktionen som fanns hos PCA3695. Det är en funktion som gör att komponenten själv ökar registeradressen med ett, vid varje skickande av data. I nuläget så skickas åtta byte för att ställa om alla optokopplare. Varannan byte innehåller registeradressen och varannan byte innehåller data. Om man istället använder auto-increment så räcker det med att skriva in en registeradress på en byte och därefter fyra byte med data. Detta gör att man bara behöver skicka 5 byte vid värdeändring hos alla optokopplare. Det skulle innebära att mängden data för att slå om alla optokopplare skulle minskas och att det därmed skulle gå snabbare. Så även denna lösning borde kunna öka maxfrekvensen på sinuskurvan.

En förhållandevis enkel implementation som kan skapas för att minska datan på bussen är en liten kodimplementation som jämför nuvarande kombination av optokopplare med nästa kombination av optokopplare. I de lägen som de stämmer överens så skickar man helt enkelt inte om värdet igen och spar tid. En nackdel med denna lösning är att den inte fungerar optimalt med auto-increment.

Ett annat område där man kan lägga ner mycket tid är att optimera vilka optokopplare som slår om. Vid jämförelse av olika kombinationer av satta optokopplare kunde snarlika spänningar fås ut över mätmotståndet. Om man utnyttjar denna information kan man utifrån det göra smarta omslag. Det vill säga att man utgår ifrån den kombination som är satt på optokopplarna innan önskad värdeändring. Därefter anpassar man sedan nästa värde utifrån nuvarande optokopplarkombination. Här väljer man då mellan snarlika värden och den

kombination som är närmast den nuvarande sätts. På så sätt kommer man ändra färre antal register och ändringarna kommer därmed gå snabbare. Även denna funktion skulle bli svår att kombinera med auto-increment, så man skulle få göra en avvägning för vilken av de två metoderna som fungerar bäst.

REFERENSER

Atlassian. 2016. JIRA. Atlassian. https://confluence.atlassian.com/jirasoftwarecloud/getting-started-as-a-jira-software-user-764477920.html (Acc 2016-04-11).

Computer History. *1960: Metal Oxide Semiconductor (MOS) Transistor Demonstrated*. Computer History Museum. http://www.computerhistory.org/siliconengine/metal-oxide-semiconductor-mos-transistor-demonstrated/ (Acc 2016-04-19).

Hord, Mike. 2013, juni 4. Basics [Figur]. https://cdn.sparkfun.com/assets/6/4/7/1/e/51ae0000ce395f645d000000.png (Acc 2016-04-11).

Hord, Mike. 2013, juni 4. Clock stretchning [Figur]. https://cdn.sparkfun.com/assets/8/2/8/8/5/51ae0000ce395f331b000000.png (Acc 2016-04-11).

Karmakar, Supriya. 2014. Novel Three-state Quantum Dot Gate Field Effect Transistor. New Delhi: Springer.

Mohiuddin, Taqi, Nawrocki, Matt och Bitter, Rick. 2006. LabVIEW Advanced Programming Techniques. 2 uppl. Boca Raton: CRC Press. E-bok.

National Instruments. 2016. cRIO-9063. National Instruments. http://sine.ni.com/nips/cds/view/p/lang/sv/nid/213090 (Acc 2016-04-05).

National Instruments. 2016. LabVIEW. National Instruments. http://sweden.ni.com/labview (Acc 2016-04-06).

National Instruments. 2014, nov. 06. NI 9401. National Instruments. http://www.ni.com/datasheet/pdf/en/ds-86 (Acc 2016-04-05).

National Instruments. 2016. Tutorial Core1, lesson 1, Dataflow. [online]. http://sine.ni.com/myni/self-paced-training/app/main.xhtml (Acc 2016-04-08).

NXP. 2014, april 4. UM10204 I²C-bus specification and user manual. NXP. http://www.nxp.com/documents/user_manual/UM10204.pdf (Acc 2016-04-11).

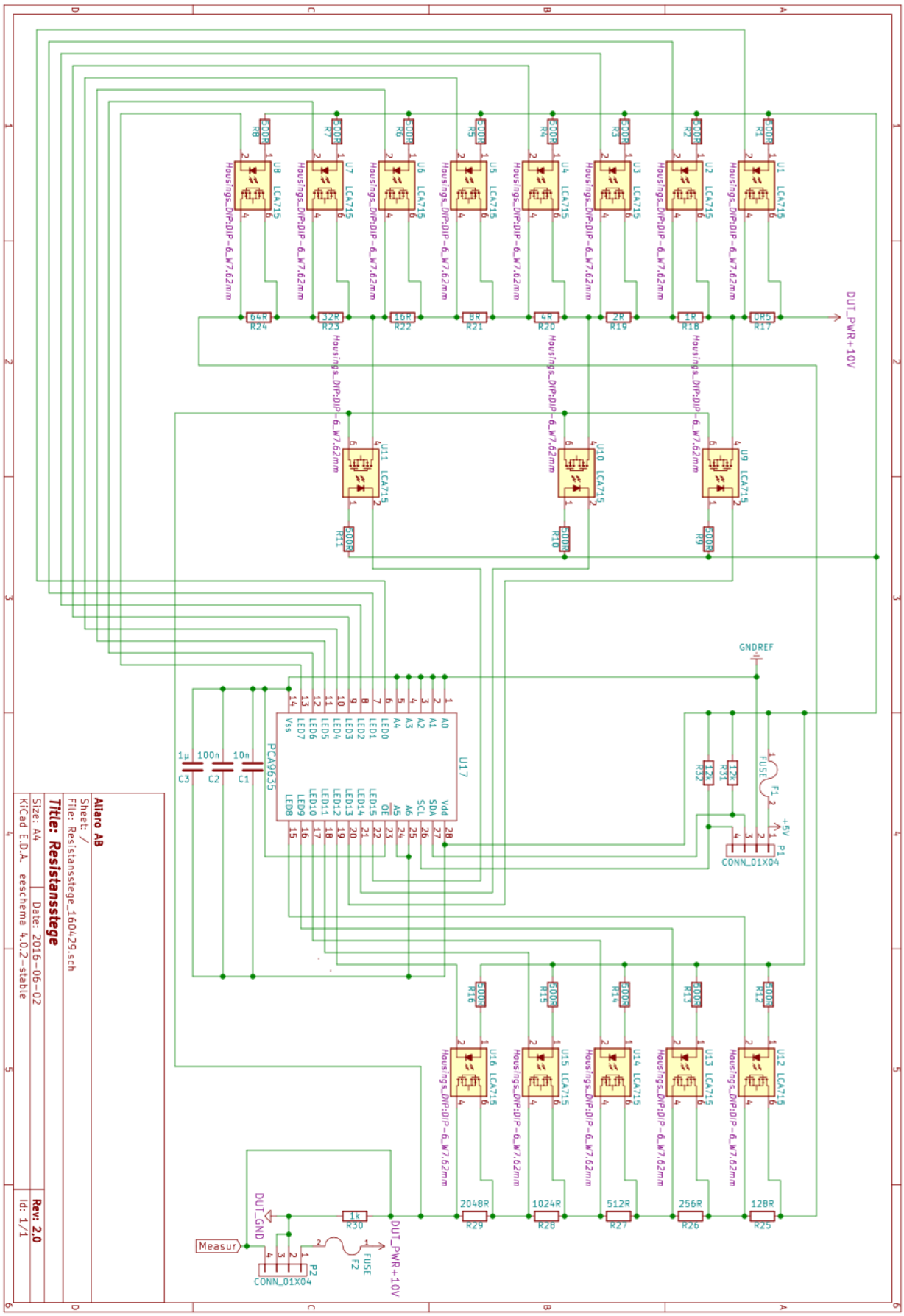
Ohm, Georg S., 1827. *Die galvanische Kette, Mathematisch bearbeitet.* Förlag okänt.
Paret, Dominique and Fenger, Carl. 1997. The I²C Bus, From Theory to Practice. Bath: Bookcraft.

Travis, Jeffrey och Kring, Jim. 2007. LabVIEW for Everyone. 3. uppl. Crawfordsville: Prentice Hall.

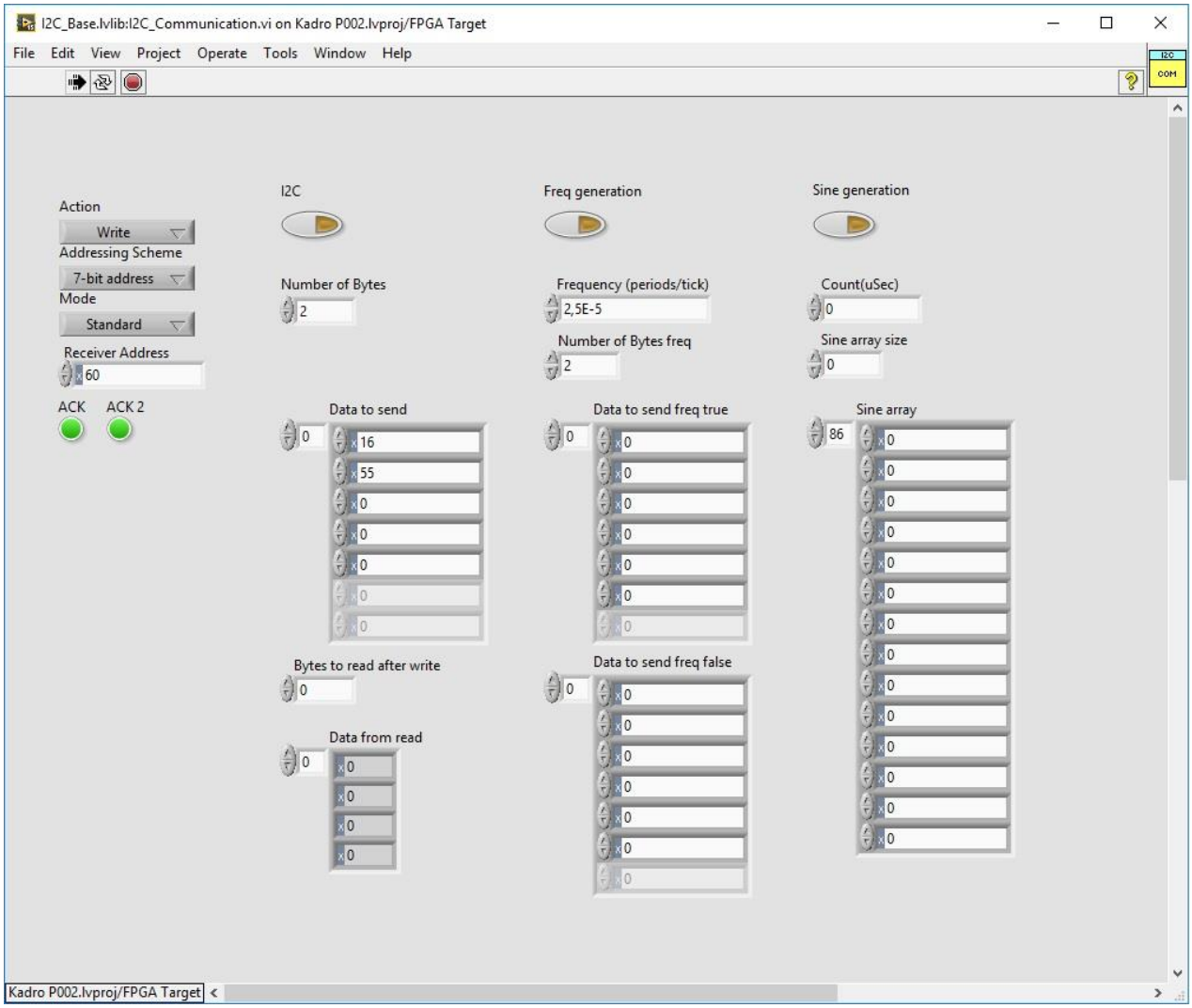
Wikipedia. 2016, mars 16. I²C. Wikipedia. https://en.wikipedia.org/wiki/I%C2%B2C (Acc 2016-04-06).

Wikipedia. 2016, mars 31. LabVIEW. Wikipedia. https://en.wikipedia.org/wiki/LabVIEW (Acc 2016-04-07).

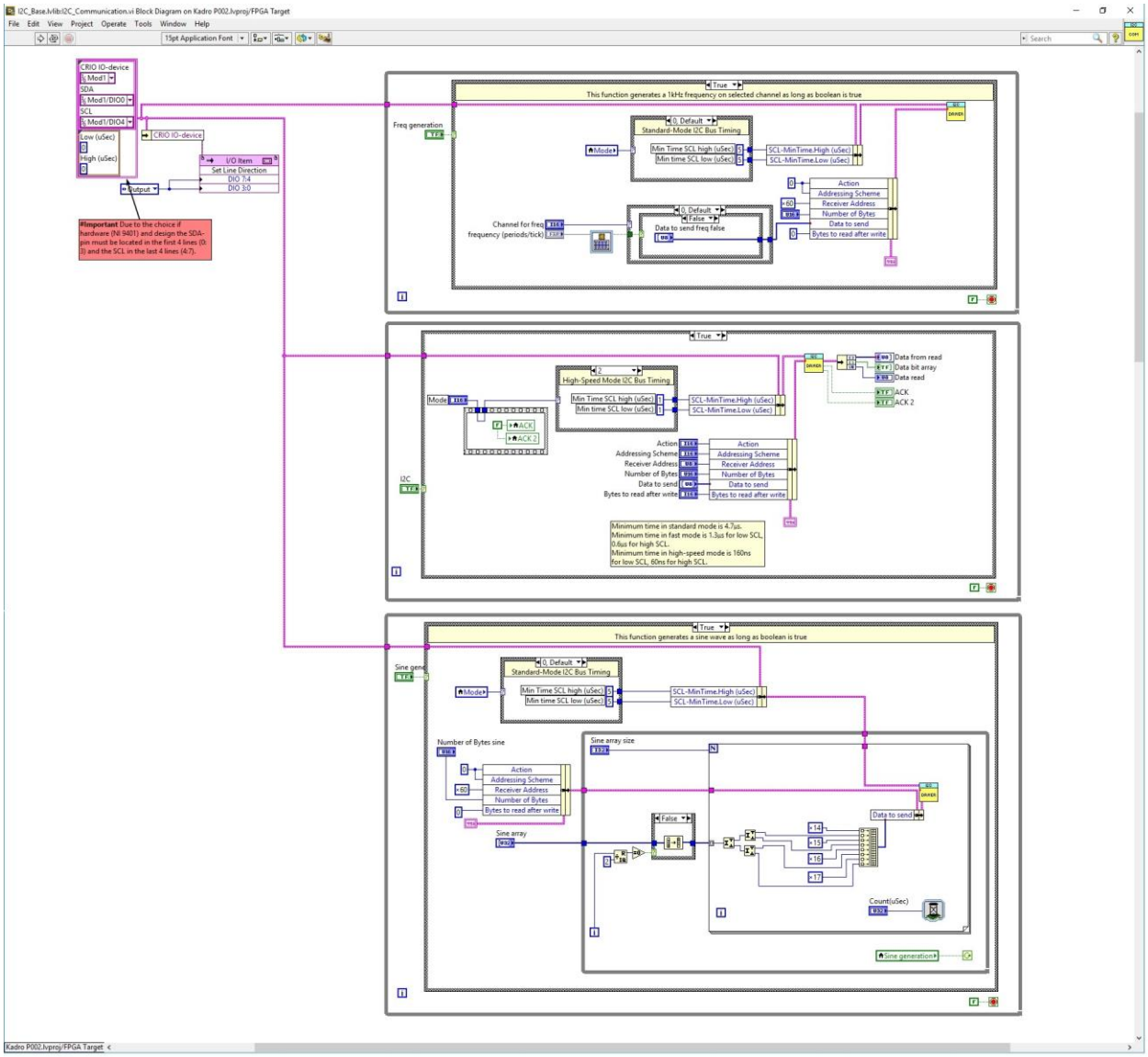
BILAGA 1 - Kretsschema



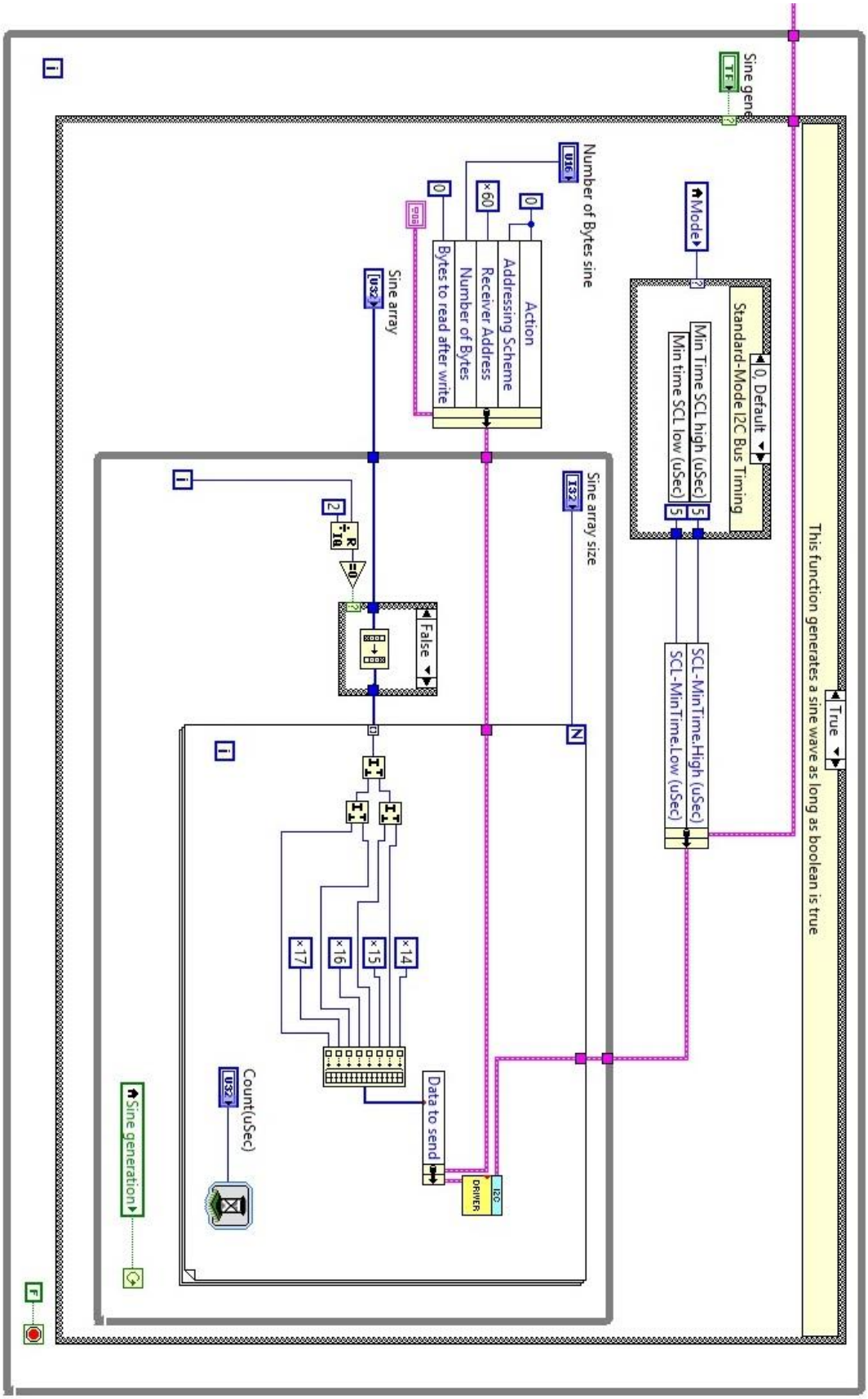
BILAGA 2 - NY FRONTPANEL FÖR HUVUDKOD



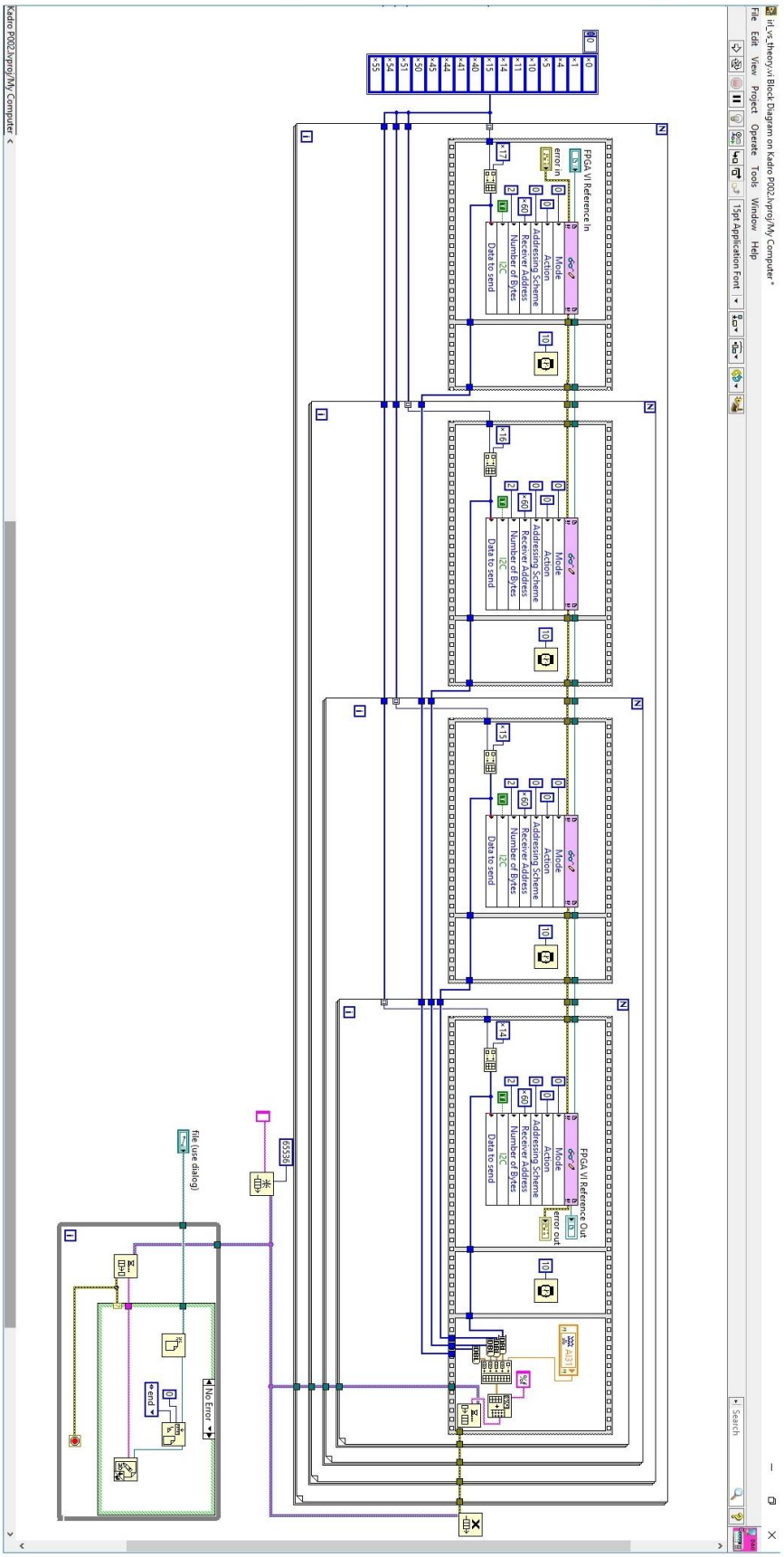
BILAGA 3 - NYTT BLOCKDIAGRAM HUVUDKOD



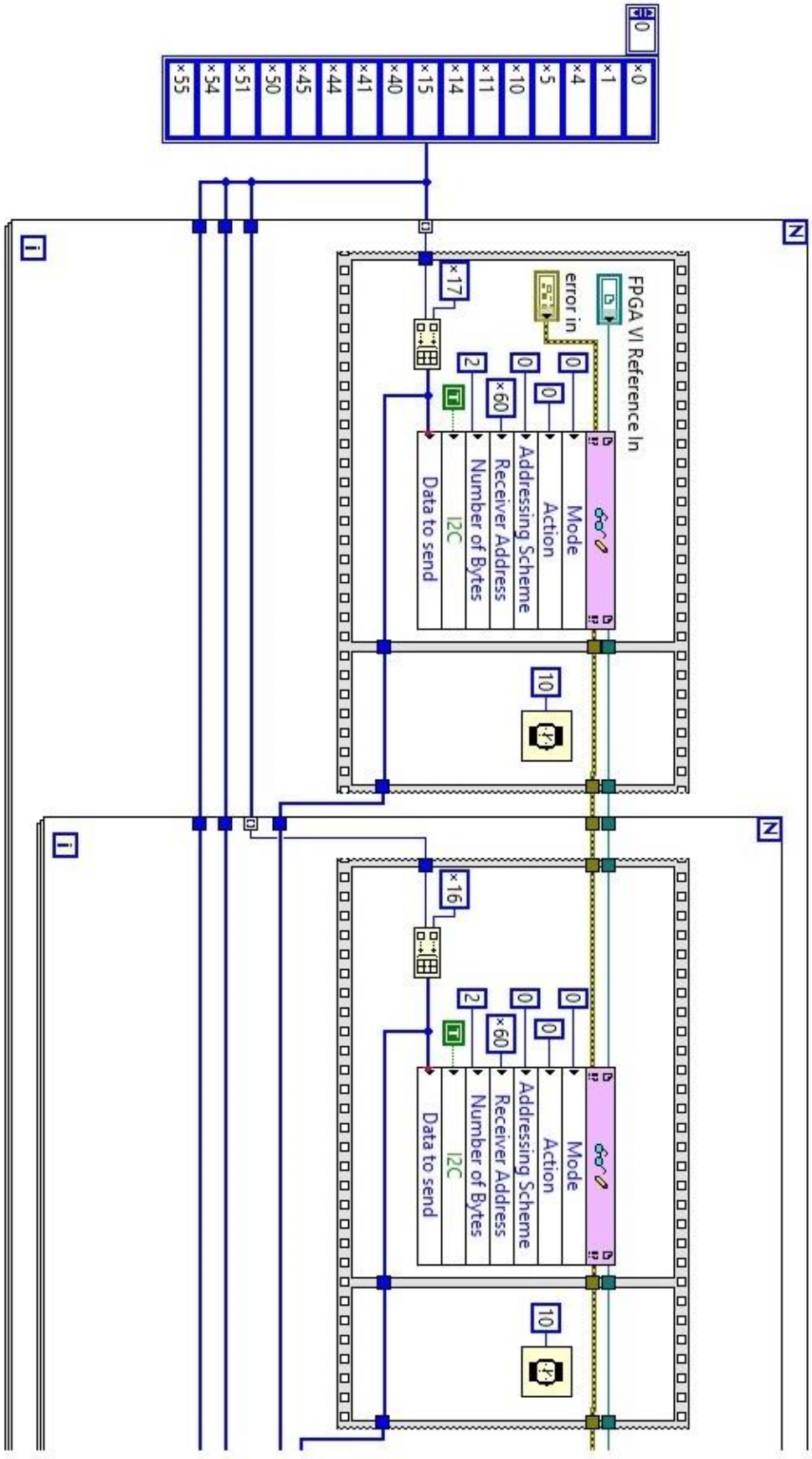
BILAGA 4 – BLOCKDIAGRAM SINUSVÅG



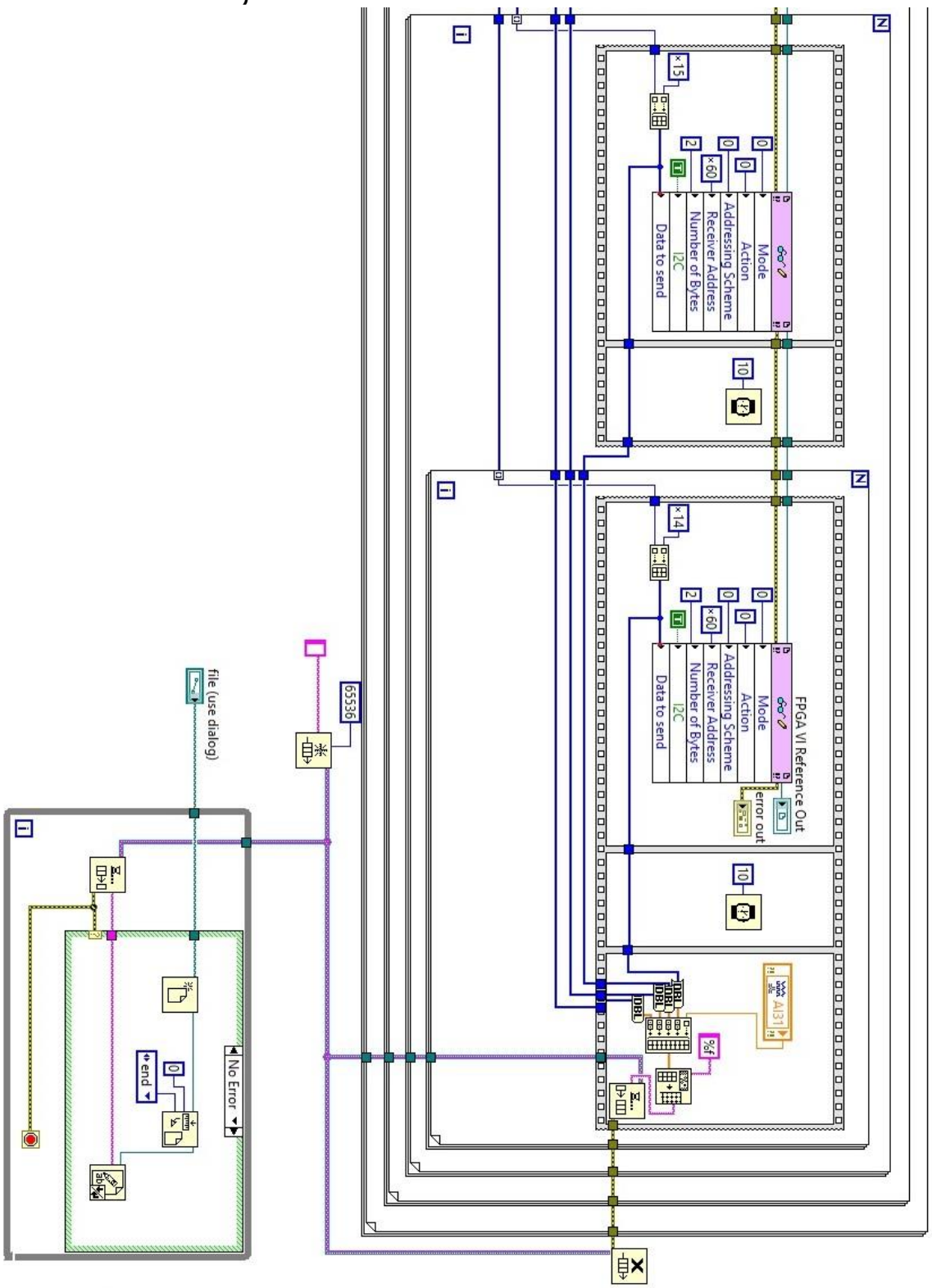
BILAGA 5 – BLOCKDIAGRAM FÖR INSAMLING AV ALLA 65536 VÄRDEN, HELA



BILAGA 6 – BLOCKDIAGRAM FÖR INSAMLING AV ALLA 65536 VÄRDEN, VÄNSTRA DELEN



BILAGA 7 – BLOCKDIAGRAM FÖR INSAMLING AV ALLA 65536 VÄRDEN, HÖGRA DELEN



BILAGA 8 - BOM

Reference	Value	Footprint	Datasheet
R17	0R5	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R19	2R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R18	1R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R20	4R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R21	8R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R23	32R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R22	16R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
U17	PCA9635	Housings_SSOP:TSSOP-28_4.4x9.7mm_Pitch0.65mm	1854073
R24	64R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R25	128R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R26	256R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R27	512R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R29	2048R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R28	1024R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R30	1k	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
P1	CONN_01X04	Connect:bornier4	2472977
U1	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U2	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U3	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U4	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U5	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U6	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U7	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U8	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U9	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U10	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U11	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U16	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U15	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U14	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U13	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
U12	LCA715	Housings_DIP:DIP-6_W7.62mm	Mouser 849-LCA715
P2	CONN_01X04	Connect:bornier4	2472977
C1	10n	Capacitors_ThroughHole:C_Rect_L7_W2.5_P5	
C2	100n	Capacitors_ThroughHole:C_Rect_L7_W2.5_P5	
C3	1Âµ	Capacitors_ThroughHole:C_Rect_L7_W2.5_P5	
R12	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R13	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R14	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R15	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R16	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R1	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	

R2	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R3	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R4	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R5	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R6	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R7	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R8	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R11	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R10	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R9	500R	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
F2	FUSE	Fuse_Holders_and_Fuses:Fuseholder5x20_horiz	
F1	FUSE	Fuse_Holders_and_Fuses:Fuseholder5x20_horiz	
R32	12k	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	
R31	12k	Resistors_ThroughHole:Resistor_Horizontal_RM7mm	