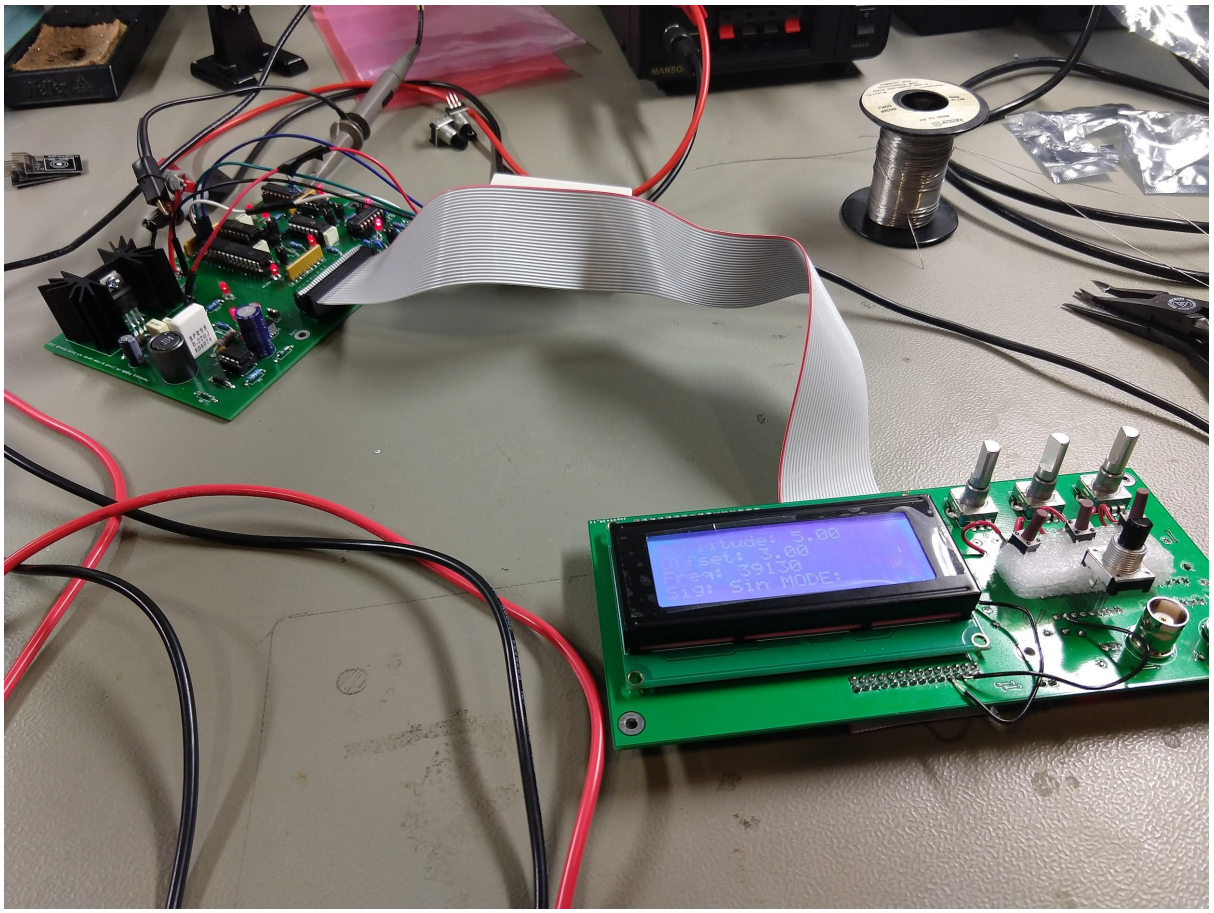


Design och konstruktion av signalgenerator

Signalgenerering och styrning medelst mikrokontroller i kombination med analoga komponenter

Examensarbete inom högskoleingenjörsprogrammet Mekatronik, 2021

Aron Lander
Daniel Strandh

Förord

Idén till projektet “Design och konstruktion av signalgenerator” började egentligen ett par år före examensarbetet började, då på grund av att Aron behövde en signalgenerator, men på grund av snålhet (studentbudget) började fundera över att bygga en istället. Tyvärr blev det aldrig något av idén vid den tidpunkten, men idén visade sig vara väl lämpad som examensarbete.

Efter att ha studerat Mekatronik Högskoleingenjör vid institutionen för elektroteknik i närmare 3 år på Chalmers Tekniska Högskola bestämde vi oss därmed för att konstruera en signalgenerator som examensarbete då flertalet av de kurser vi läst (Elektriska kretsar, Digital konstruktion, Styrprojekt och Maskinorienterad programmering) gett oss nog med bakgrundskunskaper för att kunna åstadkomma detta.

Efter att ha fått möjlighet att genomföra arbetet på Consat i Januari 2021 kunde arbetet påbörjas; dock genomfördes det mesta av arbetet hemifrån på grund av de rådande Coronarestriktioner som rådde vid nämnd tidsperiod. De avslutande delarna (att montera kretskorten) skedde däremot vid Consats kontor på Lindholmen.

Vi riktar därmed våra tacksägelser till Jonas Williamson på Consat för att ha agerat projektledare under projektets gång, Ole Andreas Utstumo på Consat för hjälp med kretsteori vid kretskonstruktion och Jian Yiang för att ha varit tillgänglig som examinator under projektets gång.

Sammanfattning

Då allt fler aspekter av samhället digitaliseras innebär detta en ökad efterfrågan på elektriska kretsar vilket resulterar i att fler nya kretsar måste utvecklas. Vad detta i sin tur mynnar ut i är därmed ett ökat behov av utformning av nya elektriska kretsar, där tillhörande utrustning för kretskonstruktion behövs. Då signalgeneratorer är en av de mest grundläggande verktyg som används vid kretskonstruktion innebär detta att den fyller en viktig funktion i denna utveckling.

Eftersom signalgeneratorer tenderar att vara tämligen dyra finns ett behov av en billig, lättillgänglig signalgenerator som möjliggör kretskonstruktion för fler. Syftet med detta projekt är därmed att ta fram en billig, lättproducerad signalgenerator som är konstruerad av billiga, lättillgängliga komponenter. Signalgeneratorn skall kunna svara på användarinput (såsom att ändra signalens frekvens och amplitud) och generera nämnd signal. Arbetet utförs hos konsultfirman Consat AB, som bland annat specialiserar sig på just kretskonstruktion.

Arbetet involverar utformning av kretsschema, kodbas för den Mikrokontroller som styr signalgeneratorn och utformning av det kretskort som signalgeneratorn konstrueras på.

Slutresultatet av projektet var en signalgenerator som kunde generera en korrekt sinus-signal och svara på användarinteraktion, dock kunde signalgeneratorn inte uppnå den förväntade prestanda (i form av signalens frekvens). Användarinterkationen nådde heller inte upp till den standard som var förväntad då gränssnittet ofta blev oresponsivt. Vidare på grund av tidsbrist prioriterades design och konstruktion av chassi bort från projektet för att fokusera på grundfunktionerna.

Nyckelord: Signalgenerator, funktionsgenerator, MCU, Mikrokontroller

Abstract

As more and more aspects of today's society are becoming digitized, an increased demand for electrical circuits has emerged over the years, resulting in a need to develop an ever increasing amount of different types of circuits. The need to develop the aforementioned circuits does in turn result in an increased demand for the type of equipment utilized when developing electrical circuits, with signal generators being one of the core types of tools being used for electrical circuit development.

Since a signal generator tends to be a rather costly piece of equipment, there is a need for an inexpensive, accessible signal generator which makes circuit development accessible for the masses. The purpose of this project is therefore to develop such a signal generator from cheap components available off the shelf, with the signal generator itself being easy to assemble.

The signal generator should be able to respond to user input (such as changing the amplitude and frequency of the signal) and generate the corresponding signal according to the parameters set by the user.

This project was realized with the help of Consat AB, a Swedish consultancy firm. Due to Covid-19 pandemic, most of the project was implemented remotely from home, while the assembly of the components on the PCB (printed circuit board) was performed at the Consat AB offices.

The project involves the design of a schematic, the codebase utilized by the Microntroller and the design of the PCB which is for the assembly of the signal generator. There were plans to design and 3d-print a case for the signal generator as well, however, due to time constraints, this was dropped from the project.

The end result of the project was a signal generator that could generate proper sinusoidal signals and respond to user interaction, however, the signal generator was never able to achieve the expected performance when it came to the highest possible frequency that could be generated. Also, the user interaction did never really meet the standards first set out for the project; the interface itself turned out to be rather unresponsive.

Keywords: Signal Generator, Function Generator, MCU, Microcontroller

Innehållsförteckning

Terminologi

PCB - Kretskort (eng. Printed Circuit Board)
MCU - Microprocessor (Microcontroller Unit)
MCC - Microchip Code Configurator

1. Inledning

I det här projektet är tanken att konstruera en signalgenerator från diskreta komponenter och 3d-utskrivna chassikomponenter där genereringen av signalen utförs av en mikrokontroller. Projektet är främst tänkt att fungera likt en produktutvecklingsprocess, och innebär därmed att utvecklingen av signalgeneratorn går från idé till en komplett slutprodukt (chassi, pcb-kort etc.).

1.1 Bakgrund

Consat är ett ingenjörshus som verkar inom Fordonsindustrin, Industri, IT, Energi, Life Science och ITS för Kollektivtrafik vilket innebär att de arbetar med bland annat innovation och produktutveckling inom elektronik och inbyggda system. De vill att uppdraget utförs för att se om en fullständig signalgenerator med chassi, fungerande användargränssnitt och relaterad signalmanipulationsfunktionalitet (såsom att kunna styra offset och amplitud) kan konstrueras från enkla diskreta komponenter (med tillhörande mikrokontroller) som finns tillgängliga på marknaden till en låg kostnad.

1.2 Syfte

Projektets syfte är att genomföra en komplett produktutvecklingsprocess, från idé till färdig produkt, där slutresultatet skall vara en fullt fungerande signalgenerator, som skall kunna användas vid testning av elektriska kretsar.

1.3 Avgränsningar

1. Högfrekventa kretsar är inte inkluderade in funktionalitetsspecifikationen; signalgeneratorns frekvens behöver därmed inte överstiga 500 kHz; ~200-250 kHz bör vara tillräckligt för tilltänkta applikationsområden.
2. Spänningen som ska kunna levereras begränsas till +/-10 V.
3. En utgångskanal.

1.4 Precisering av frågeställningen

Är det möjligt att designa en signalgenerator som dels är lätt att konstruera, och dels är byggd från billiga, lättillgängliga komponenter som ändå uppnår en funktionalitetsnivå som är tillräckligt god för att kunna användas vid felsökning/konstruktion av elektriska kretsar?

- Går det att uppnå en tillräckligt låg distortionsnivå av signalen för att signalgeneratorn skall vara funktionsduglig för felsökning av elektriska kretsar utan att introducera så pass mycket brus att de resultat oscilloskopet återger blir oanvändbara?
- Hur mycket mer (eller mindre) distortion genererar signalgeneratorn i förhållande till en kommersiell enhet?
- Går det att uppnå tillräckligt hög frekvens på utsignalen för att göra signalgeneratorn användbar till något mer än ljudkretsar?

2. Teknisk Bakgrund

I detta kapitel presenteras den tekniska bakgrunden till en del av de begrepp, komponenter och kretsar som används i detta projekt.

2.1 Kapacitansmultiplikator

Enligt P. Horowitz, W. Hill [1, s. 578-579] är kapacitansmultiplikator en standardkrets för att motverka krusningar (eng. ripple) på en signal genom att förstärka effekten av en kondensator med hjälp av en transistor. Kretsen består i princip av ett lågpassfilter och spänningsföljare (eng. emitter follower) som är sammankopplade. Det leder till begränsade krusningar och mindre spänningsfall än om bara ett lågpassfilter används.

2.2 Inverterande konverterare

Komponenten kallas också *Inverting buck-boost converter* och används till invertera och eventuellt modifiera signalens amplitud. I [1, s. 648] kan man se att kretsen består av två polariserade kondensatorer, en schottky-diod, en spole och en brytare (en transistor eller liknande). Spolen laddas först upp med energi när brytaren är sluten och sedan när brytaren öppnas kommer strömmen att dras från den negativa sidan genom lasten och filterkondensatorn på grund av spolens strömtröghet. Brytaren sluts och bryts med en viss frekvens beroende på periodtid vilket gör att medelvärdet på utgången kan vara både större eller mindre i amplitud än ingångsspänningen.

2.3 Linjär spänningsregulator

En spänningsregulator är enligt [1, s. 700] en krets för att transformera spänningen utan att använda en transformator. Den jämför ett stick-prov från likströmsutsignalen med en inre referens i en “error amplifier” som ger negativ feedback till en transistor som styr hur mycket ström som ska passera och därmed styr spänningen.

2.4 Sallen And Key-filter

Filtret är 2-poligt och består enligt [2] av en icke-inverterande operationsförstärkare och två resistorer. Detta skapar en spänningskontrollerad spänningskälla med filterkarakteristik med hög ingångsimpedans, låg utgångsimpedans och god stabilitet. Detta medför att det är möjligt att koppla samman flera av dessa filter för att skapa ett filter av högre ordning än bara ett Sallen-and-Key-filter ger.

2.5 Butterworth-filter

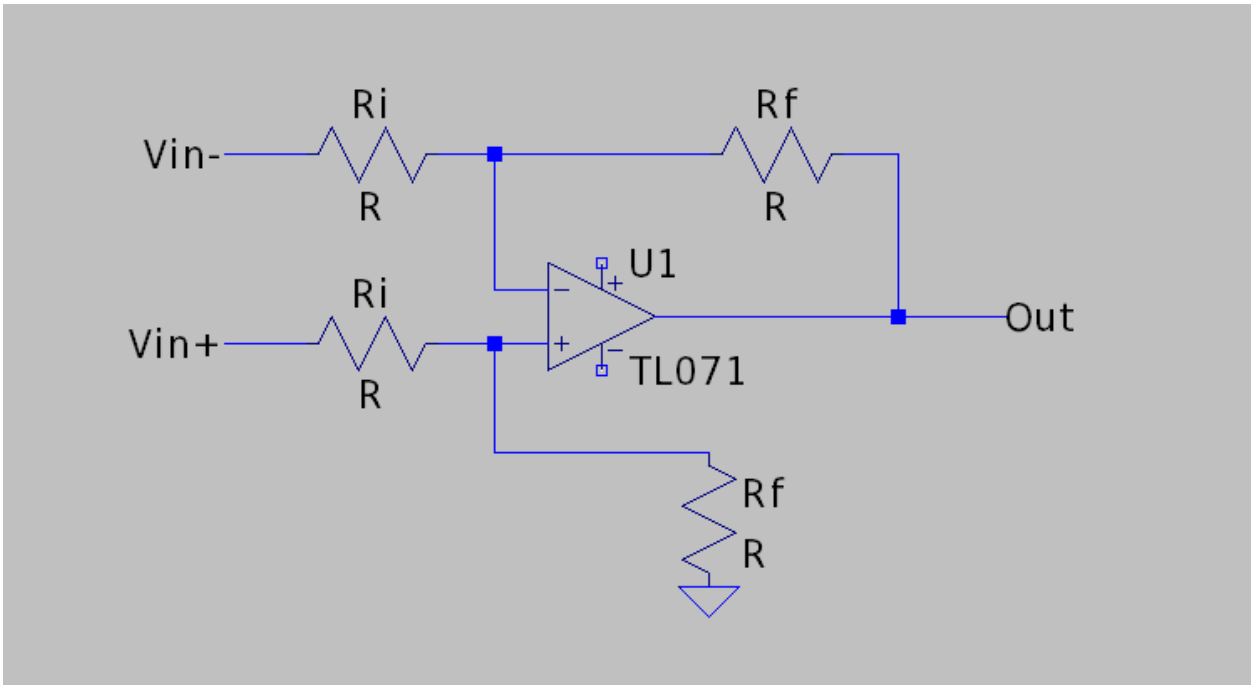
Butterworth-filter är enligt [3] ett lågpassfilter som egentligen är flera kaskadkopplade fåpoliga filter (här: Sallen and Key) för att uppnå ett filter av högre ordning. Det finns färdiga tabeller med andra ordningens polynom multiplicerade med varandra för att underlätta beräkningen av komponenternas värden i de fåpoliga filtren. Med högre ordning på filtret fås en brantare kurva vid brytfrekvensen och det blir väldigt lite krusningar på utsignalen. Dock förlorar man precision vid högre ordningar.

2.6 Differensförstärkare

En differensförstärkare används egentligen främst för att kompensera för störningar i en signal [1, s. 347]. I och med att dessa egenskaper resulterar i att differensförstärkaren kan addera/subtrahera en spänning från en signal innebär detta att den tillhandahåller möjligheten att kunna lägga till en så kallad “offset” till signalen, vilket går att se enligt följande samband [1, s. 348]:

$$G_{diff} = \frac{V_{out}}{V_{in+} - V_{in-}} = R_f / R_i \quad (1)$$

För att ge ett sammanhang till nämnda formel följer en principskiss:



Figur 1 - Kopplingsschema för en differensförstärkare.

Genom att välja R_f=R_i fås en förstärkning (G) på 1 (ty R_f/R_i=G) och formel (1) kan därmed skrivas om till:

$$R_f/R_i = \frac{V_{out}}{V_{in+}-V_{in-}} => V_{out} = (V_{in+} - V_{in-})(R_f/R_i)\,,\ G = 1 = R_f/R_i => V_{out} = V_{in+} - V_{in-} \quad (2)$$

Resultatet av detta är därmed en positiv spänning (likström) på V_{in-} kommer att subtraheras från den variabla signalen på V_{in+}, vilket därmed resulterar i att ett offset erhålles.

3. Metod

Eftersom projektet var av tämligen omfattande karaktär behövdes en metod för dess genomförande. Därmed togs en metod fram för att gradvis genomföra planering, design och implementation.

3.1 Funktionalitetsbeskrivning/Specifikation

Skriva ned en fullständig specifikation över vilken funktionalitet signalgeneratorn skall uppfylla utöver att generera signaler (t.ex. trigger-signal, signaltyper osv.) och att ha ett användargränssnitt. Detta involverar även vilka utgångar respektive ingångar funktionsgeneratorn skall ha.

3.2 Skiss av användargränssnitt

För att kunna avgöra vilka komponenter som skall användas för interaktion måste en skiss över hur användarinteraktionen skall ske tas fram. Detta involverar exempelvis saker som att avgöra huruvida frekvenskontroll sker med en knappsats eller en/flera rotationsgivare.

3.3 Blockschema

För att ha en idé om vilka moment som är involverade i signalgeneratorns arkitektur måste först ett blockschema konstrueras, vilket tillhandahåller en överblicksbild över vilka komponenter som behöver implementeras.

Utifrån blockschemat kan sedan en plan över mikrokontrollers kodarkitektur tas fram.

3.4 Val av komponenter

Innan ett kretsschema kan tas fram måste komponentval göras då olika komponenter har olika pinkonfigurationer, och kretsschemat är beroende av detta. I detta skede sker därmed även val av typ av komponenter att använda för att implementera de olika sektionerna av blockdiagramet.

I detta steg kommer även insamling av datablad att ske för senare referens.

3.5 Design av kretsschema

När komponenterna valts kan ett kretsschema ritas, vilket i slutändan tillhandahåller information om vilka ingångar och utgångar som kommer att användas på mikrokontrollern, men även en fingervisning om vilka krav som kommer att ställas på mikrokontrollern (i form av tillgängliga ben).

3.6 Kodarkitektur

Parallellt med kretsschemat behöver även en kodarkitektur tas fram. Även om specifika saker såsom ingångar och utgångar kommer att behöva fastställas går detta att göra i ett senare skede i och med att C har macron (definerade i header-filer), och dessa utgångar och ingångar därmed kan definieras i dessa.

3.7 Funktonalitetsimplementation

Att steg för steg bygga kretsen på en breadboard och utveckla den kod som behövs för att mikrokontrollern skall kunna hantera dessa komponenter. Implementationen är tänkt att ske steg för steg (block för block), och därmed ge möjligheten att inkrementellt testa funktionaliteten och därmed säkerställa att “blocket” fungerar innan nästa del av mikrokontrollerns respektive kretsens funktionalitet.

3.8 PCB-design

När en fungerande funktionsdesign väl finns på plats, dels som krets, och dels i form av kod för mikrokontrollern behövs en PCB-design (eller flera, beroende på huruvida beslut tas om att använda sk. “daughter boards” eller inte). Att få ut en fullt fungerande PCB åstadkommes genom att designa den i Kicad och sedan beställa den från en leverantör av PCB:er.

3.9 Utformning av chassi

I och med chassits storlek och design är beroende av de inre komponenternas dimensioner (främst pcb-kortet/pcb-korten) kan inte designen göras förens en slutgiltlig PCB-design är framtagen. Designen är tänkt att genomföras medelst CAD-verktyg och skrivas ut på 3d-skrivare.

I och med att det på förhand inte är helt lätt att förutse passning vid 3d-utskrift (då 3d-utskrifter som produktionsmedel kommer med ganska höga toleranser) kommer därmed designen att ske iterativt tills ett chassi med önskvärda egenskaper framkommit.

4. Funktionalitetsbeskrivning

För att få en bild av vilka aspekter som behövde implementeras i projektet behövdes först avgränsningar, vilka utformades genom att ta fram en funktionalitetsspecifikation, vilken delades in i två huvudkategorier: “Vad skall den göra?” och “Interaktionsmöjligheter. “Vad skall den göra?” kan ses som en rent teknisk funktionsspecifikation, medan “Interaktionsmöjligheter” specificerar hur signalgenerator interagerar med sin omvärld.

Vad skall den göra?

Den funktionalitet funktionsgeneratorn skall tillhandahålla kan delas in i två subkategorier: “grundfunktionaltiet” och “extrafunktionalitet”. Grundfunktionaliteten beskriver de aspekter av signalgeneratorn som krävs för att den skall uppfylla de kriterier som krävs för att kunna användas vid elektrisk kretskonstruktion, medan extrafunktionaliteten specificerar aspekter som inte är absolut nödvändiga, men gör signalgeneratorn mer användarvänlig.

Grundfunktionalitet

- Generera signaler
    - Signaltyper
        - Sinus-kurva
        - Fyrkantsvåg
        - Triangelvåg
        - Sågtandsvåg
    - Skicka ut signal på godtycklig frekvens (inom signalgeneratorns begränsningar)
    - Signalamplitud
        - Negativa spänningar
        - Spann +/- 9 V
        - Styrbar offset

Extra funktionalitetet

- Trigger-signal: för att kunna synkronisera oscilloskop, synnerligen användbart för att vara dugligt användbar.
- Sända ut godtycklig DC-spänning inom givet intervall (-10V - +10V)

Interaktionsmöjligheter

Interaktionsmöjligheter inbegriper dels interaktionen mellan användare och maskin, då främst vilka operationer som skall vara möjliga och dels interaktionen mellan funktionsgeneratorn och andra enheter, vilket t.ex. innefattar hur signaler skickas till den testade kretsen.

Människa-maskin

- Input: kan ske medelst knappar, knappsats eller rotationsenkoders
    - Välja frekvens
    - Välja vågform
    - Välja amplitud
    - Välja offset
    - Välja DC-läge eller frekvens
- Feedback: Display, lysdioder, fasta reglage. Informera användaren om vad den gör för tillfället
    - Visa frekvens
    - Visa vågform
    - Visa amplitud
    - Visa offset
    - Visa signalläge (fast DC-signal eller vågformer)
    - Visa läge; huruvida i inställningsläge/vad kommer att ställas om vid interaktion (detta är beroende av hur vi väljer att designa kontrollpanelen)

Maskin-maskin

- Signalutgång: sänder ut vågformen eller DC via en BNC-kontakt.
- Trigger-signal: Sänder ut en fyrkantsvåg på samma frekvens som vågformen via en BNC-kontakt. Skall vara av om DC-läge är på. Denna är 0-5V

5. Skiss av användargränssnitt

Utifrån de specifikationer som tagits fram under funktionalitetsbeskrivningen utformades två olika förslag till användargränssnittet. För att avgöra vilken design som skulle komma att användas behövdes därmed respektive designs för- och nackdelar vägas för att kunna urskilja vilken design som slutligen skulle användas.

De två designförslag som uppkom ur skisseringsprocessingen var inte på något sätt slutgiltiga, utan snarare utkast för att avgöra vilken väg att gå vid den slutgiltiga designen.

5.1 Begränsningar

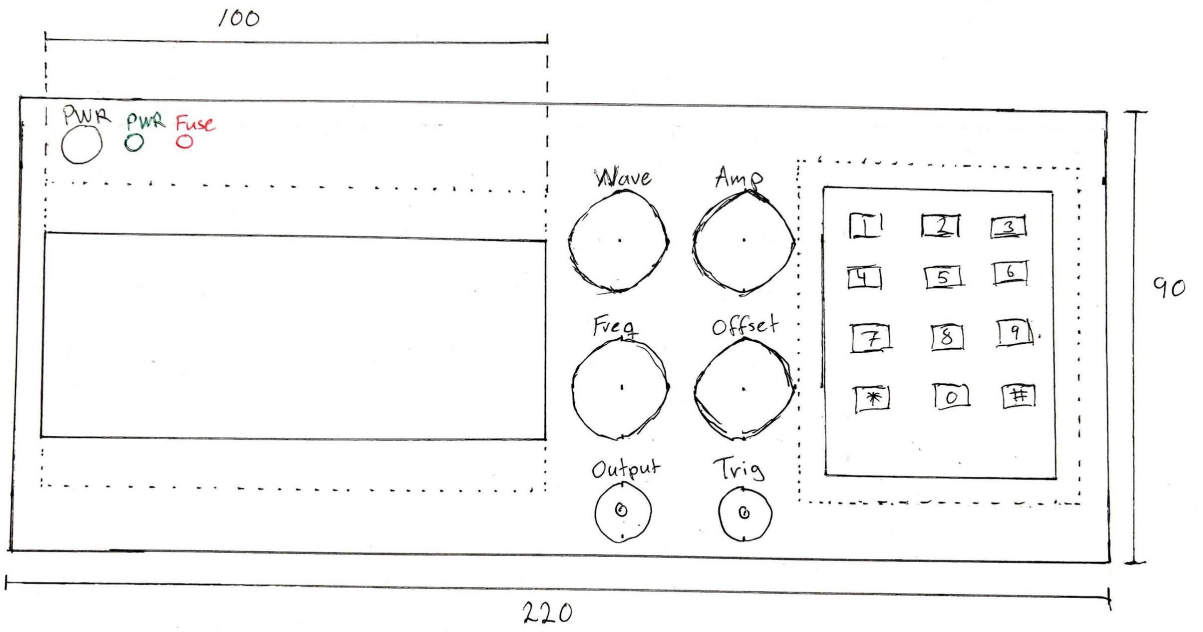
Då 3D-skrivarens maximala utskriftsstorlek är 220mm x 220mm x 250mm (bredd, djup, höjd) måste därmed designen hålla sig inom dessa. I och med att en frontpanel skrivs ut liggandes plant mot skrivarens utskriftsyta är därmed begränsningarna 220 x 220mm.

5.2 Avvägningsprocessen

För att avgöra vilken design som slutligen skulle användas för frontpanelen (med andra ord användargränssnittet) skapades en lista av för- respektive nackdelar hos varje designskiss varpå de viktades utifrån följande aspekter:

- Kostnad av frontpanelens komponenter
- Storleksbegränsningar vid 3D-utskrift (se underrubrik: “Begränsningar” ovan)
- Användarvänlighet
- Komponenternas tillgänglighet

Förslag 1:



Figur 2- Designförslag 1, med knappsats.

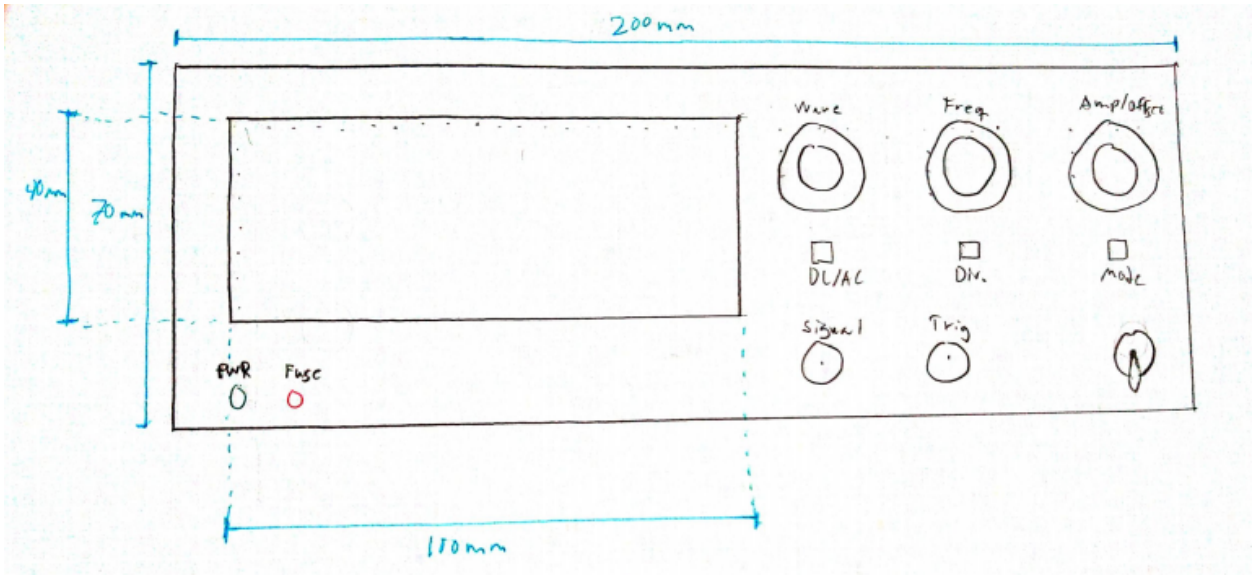
Fördelar

- Knappsats kan användas för att välja exakta värden.
- Mer anpassningsmöjligheter senare i designprocessen. I och med att en knappsats är tämligen flexibel ger den möjlighet till fler funktioner är de som specificerats i funktionsbeskrivningen.

Nackdelar

- Dyrare på grund av inköp av knappsats
- Ökad komplexitet vid utformning av frontpanel (fler komponenter måste passas in på den tillgängliga ytan)

Förslag 2:



Figur 3 - Designförslag 2, huvudsakligen rotationsenkodrar.

Fördelar

- Billigare att tillverka - knappsats kostar minst 50 sek, samma mängd rotationsenkoders (ungefär 6-8 sek styck)
- Mindre utskriftsyta - Marginaler finns ifall något behöver ändras (2 cm till godo), vilket i sin tur resulterar i en mer kompakt slutprodukt.
- Minskad intern komplexitet, ty färre användargränssnittskomponenter behöver interagera med mikrokontrollern

Nackdelar:

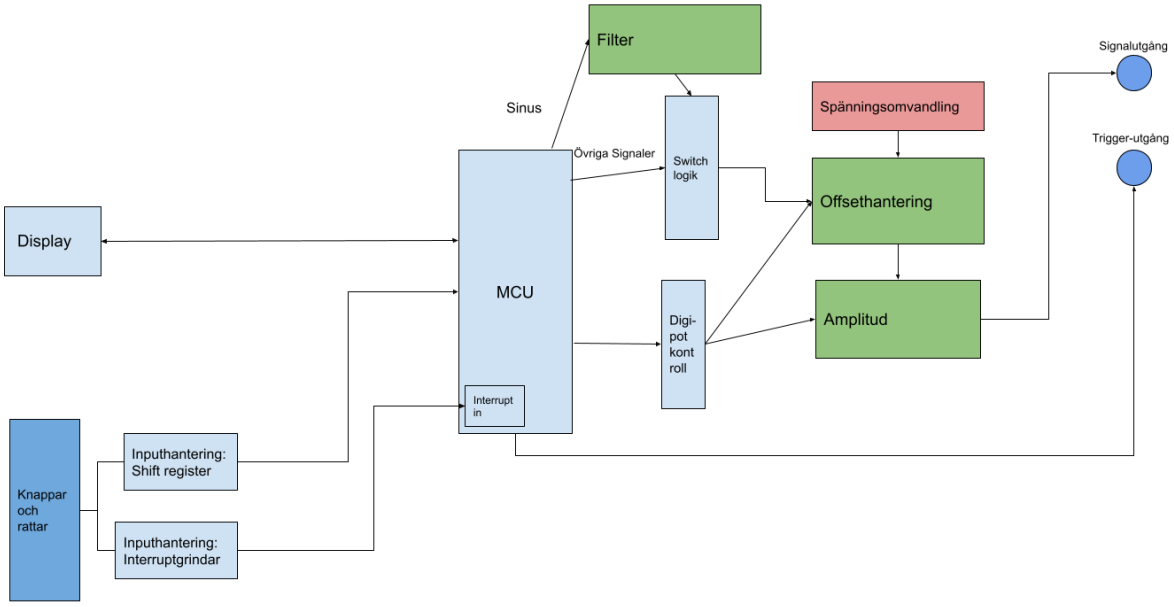
- Mindre användarvänlig i jämförelse med en knappsats, att byta från en låg till hög frekvens skulle i fallet med design 2 kräva flera byten av “divisor” och ganska mycket justering av “freq”-reglaget för en ändring av frekvens från t.ex. 150Hz till 24550Hz.

5.3 Slutgiltig bedömning av alternativen

I och med att kostnad är en av de främsta optimeringsparametrar som är specificerade för projektet, och att 3d-skrivarens utskriftsyta är begränsad till 220mm x 220mm x 250mm valdes “Designskiss 2” som det användargränssnitt signalgeneratorn skulle komma att använda.

6. Blockschema/Arkitektur

För att kunna utforma signalgeneratorns kopplingsschema (vilket i sin tur är en förutsättning för att kunna utforma en PCB) behövde först signalgeneratorns övergripande arkitektur utformas. Detta löstes genom att rita ett blockschema, vilket tillhandahöll en grov, men översiktlig arkitekturell vy av signalgeneratorn.



Figur 4 - Blockschema över systemets arkitektur

Vid läsning av blockschemat (Figur 4) bör följande två saker hållas i åtanke: schemat läses från riktning vänster till höger, med andra ord från användarinput till utsignal, och att blockschemat är indelat efter färg, där:

- Mörkblå designerar någon typ av interaktion; på vänster sida av blockschema handlar det om användarinteraktion medan det på höger sida rör sig om maskininteraktion (signaler).
- Ljusblå designerar digitala komponenter, däribland MCU och grindlogik.
- Grön indikerar att det komponenten är analog, t.ex. Filter.
- Röd indikerar att komponenten är relaterad till spänningshantering av något slag; i det här fallet rör det sig enbart om en switchande DC-DC-regulator.

6.1 Detaljbeskrivning av blockschemats indelning

För att ge en överblick över vad blockschemats olika komponenter gör och för att ge en bild av hur de hänger samman följer en beskrivning av varje individuell komponent i blockschemat.

Display

Signalgeneratorns metod för att ge återkoppling till användaren. I det här fallet en ASCII-display som tillhandahåller information såsom amplitud, frekvens, offset och signaltyp.

Knappar och rattar

För att hantera användarinput krävs någon form av reglerdon, vilket löses genom 3 stycken rotationsenkoderar och 3 stycken knappar (se “5. Skiss av användargränssnitt”).

Inputhantering: Shift register och Interruptgrindar

Genom att ha en en kombination av kombinatorisk logik och två stycken seriekopplade skiftregister kan användargränsnittet kommunicera med MCU:n genom enbart 4 stycken in-/utgångar kombinerat med möjligheten att styra MCU:n via interrupts istället för kontinuerlig inputhantering, vilket sparar klockcykler och därmed möjliggör en högre klockfrekvens för utsignalen.

MCU

Mikrokontrollern är den huvudsakliga komponenten i signalgeneratorn, och används dels för dess huvudfunktionalitet, nämligen att generera signaler, men även hantering av kringfunktion såsom användarinteraktion.

Digital potentiometerkontroll

Då både offsethantering och amplitud-blocken är konstruerade från operationsförstärkare behövs potentiometrar för att kontrollera dessa, och vilket löses genom att använda två digitala potentiometrar (dock i en sammansatt kapsel).

Offsethantering

Då man ofta behöver kunna generera en signal med en offset, t.ex. en sinuskurva från -2.5 till +2.5V behövs någon form av offsethantering, vilket löses medelst en differensförstärkare, konstruerad från operationsförstärkare som beskrivs i [1, s. 354].

Amplitud

Då signalen som MCU:n genererar alltid kommer att ha en amplitud mellan 0-5V används en inverterande operationsförstärkare för att kunna uppnå en amplitud mellan -10V till +10V.

Filter

Då signalgeneratorn genererar en digital sinuskurva innebär detta att signalen när den kommer ut från signalgeneratorn har en “trappstegsform”, och därmed behöver “jämnas ut” för att få formen av en proper sinuskurva. Detta löses genom att filtrera signalen genom ett butterworthfilter enligt [3].

Switchlogik

Då enbart sinuskurvor skall filtreras måste det finnas möjlighet att kontrollera huruvida utsignalen från MCU:n skall gå igenom filtret eller direkt ut genom BNC-kontakten, vilket löses genom att MCU:n via switchlogiken kan välja om en filtrerad eller ofiltrerad version av signalen skall gå in i offsethantering och amplitud-blocken innan den slutligen når signalutgången.

Spänningsomvandling

Eftersom utsignalen från MCU:n enbart har ett spann mellan 0 till 5 V, men senare komponenter i kedjan, såsom offsethanteringen, kommer att behöva avge en signal upp till -10 till +10 V behövs negativ strömförsörjning till operationsförstärkarna i nämnda komponenter. Då strömförsörjningen till signalgeneratorn enbart tillhandahåller +15V DC löses detta därmed genom att ha en DC-DC-omvandlare som konverterar +15V DC till -15V DC.

7. Design av kretsschema

Då en MCU enbart skulle kunna ge en signal mellan 0 och 5V utan några möjligheter till kontroll av den genererade signalen, behövs ytterligare kretsar (subsystem) för att uppnå önskad funktionalitet (se kapitel 4. Funktionalitetsbeskrivning). Därmed följer en beskrivning av hur signalgeneratorns subsystem utformades.

7.1 Ingående spänningsreglering

I och med att de digitala komponenter signalgeneratorn använder sig av behövs en strömförsörjning på +5V, vilket löstes med en linjär spänningsregulator av typen L7805 (se Figur 5). För att användas för sitt tilltänkta syfte kräver den inte mycket mer än ett par bypass-kondensatorer och en diod för att förhindra backspänning [1, s. 602].

I och med att det är tämligen svårt att beräkna total strömkonsumtion i och med de otaliga variabler som uppkommer (inte heller tillverkarna av kretsarna tillhandahåller exakta siffror på detta) blev vi helt sonika tvungna att approximera en förbrukning och därmed överdimensionera kylningen. Slutsatsen blev därmed att dimensionera för 700 mA då det lär ligga en bra bit i överkant i förhållande till vad komponenterna konsumerar. Då spänningsfallet mellan ingång på L7805 (U1) är närmare 10V (+15v till +5V) resulterar detta i en tämligen kraftig värmeutveckling i U1, vilket beräknades enligt formel 9.2 [1, s. 624]:

$$T_J = T_A + (R_{\Theta JC} + R_{\theta CS} + R_{\theta SA})P \quad (3)$$

Där:
T_J: “Junction temperature”, mao. temperatur i kretsen
T_A: Temperatur i chassit, här antaget som rumstemperatur (25℃)
$R_{\Theta JC}$: Termisk resistans mellan krets och TO-220-förpackning
$R_{\theta CS}$: Termisk resistans mellan kylfläns och TO-220-förpackning
$R_{\theta SA}$: Termisk resistans mellan kylfläns och omgivande luft

Värden för $R_{\Theta JC}$och $R_{\theta CS}$ ges av databladet för L7805CV [4], där $R_{\Theta JC}$= 5℃/W och $R_{\theta CS}$antas med hjälp av kylpasta eller termisk kudde vara 0.5℃/W [1, s. 624].

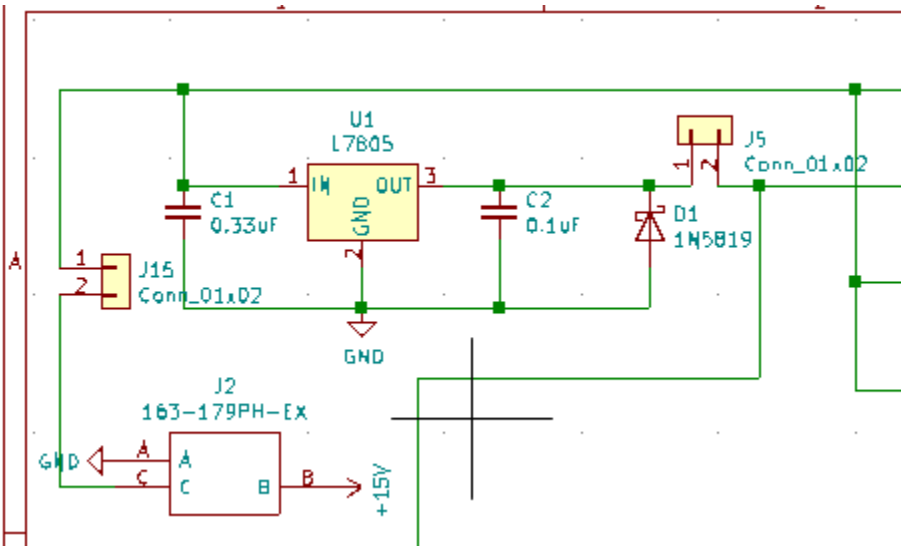
Då databladet för L7805CV [2] anger att maximal T_J = 125℃ sattes en gräns på 100℃ för T_J för att uppnå en viss säkerhetsmarignal, medan T_A antogs till 30℃, då det i övrigt inte finns någon värmegenererande komponent i signalgeneratorns chassi.

I och med att spänningsfallet över U1 uppgår till 10V, och maximal ström genom U1 antas vara 600mA är P = 6W.

För att slutligen beräkna den maximala termiska resistans kylflänsen får inneha användes skrevs Formel (3) om till följande:

$$\frac{T_J - T_A}{P} - R_{\theta JC} - R_{\theta CS} = R_{\theta SA} \quad (4)$$

Detta resulterade slutligen i ett maxvärde för $R_{\theta SA}$<= 5.97℃/W. Den kylfläns som valdes för syftet var TA-220-25E, med en termisk resistans på 4.7℃/W [5].



Figur 5 - Kretsschema för ingående spänningsreglering.

7.2 Kapacitansmultiplikatorer

Då utsignalen från signalgeneratorn använder sig av analog elektronik, och signalgeneratorn är tänkt att användas som en referenssignal bör det finnas så lite störningar som möjligt hos strömförsörjningen till de analoga komponenterna. För att lösa detta sitter det därmed en kapacitansmultiplikator vid +15V- och -15V-ingången till de analoga komponenterna (för mer information bakom teorin kring kapacitansmultiplikatorer, se sektion 2.1).

För att undvika spänningsfall valdes en så liten resistor som möjligt, medan kondensatorvalen skiljde sig något mellan den positiva och negativa strömförsörjningen. För den negativa spänning hölls kondensatorn mindre (se Figur 6) för att minska latensen för att uppnå kontinuerligt spänning, vilket också möjliggjordes av en tämligen hög krusningsfrekvens hos DC-DC-konverteraren U5 (~50kHz).

I fallet hos den positiva 15V-strömförsörjningen antogs krusningsfrekvensen ligga närmare elnätet (50Hz), vilket därmed rättfärdigade en större kondensator (se Figur 7).

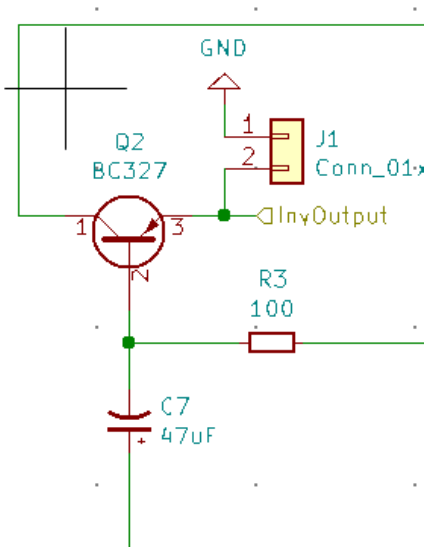
Formel för hel liktriktarbrygga hämtad från [1, s. 33]:

$$\Delta V = \frac{I_{load}}{2fC} \quad (5)$$

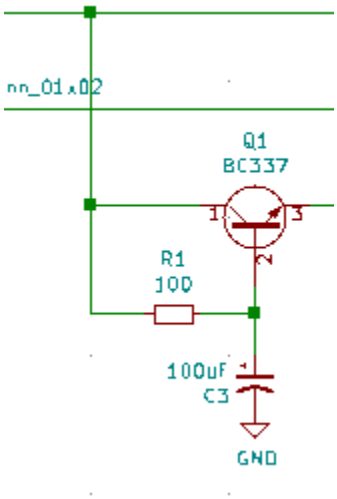
I och med att transistorns främsta syfte är att hålla ned strömmen som passerar genom kondensatorn (se formel 5 som relaterar till krusningshantering i en likriktarbrygga), och därmed kunna hålla nere storleken men fortfarande filtrera så låga frekvenser som möjligt kunde brytfrekvenserna beräknas som ett vanligt lågpassfilter :

$$f_c = \frac{1}{2\pi RC} \quad (6)$$

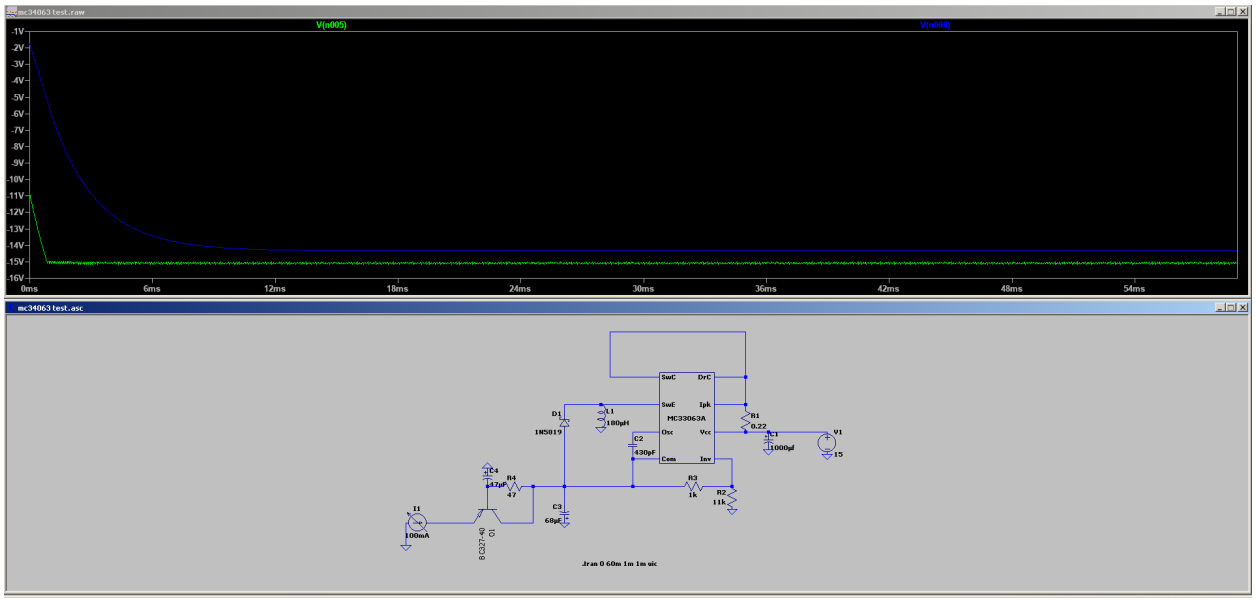
där f_c är brytfrekvens, R är motståndet och C kapacitansen, se [6], vilket illustreras i figur 8.



Figur 6 - Kretsschema för kapacitansmultiplikator med mindre kondensator



Figur 7 - Kretsschema för kapacitansmultiplikator med större kondensator



Figur 8 - Signal före (grön) och efter (blå) kapacitansmultiplikator.

7.3 Hantering av användarinput

För användarinput var tanken att använda sig av avbrott men samtidigt använda så få IO-ben på MCU:n som möjligt. Fördelen med detta tillvägagångssätt är att färre klockcykler kommer ödslas på användarinputhantering; istället för att kontinuerligt titta på det händer något i skiftregistret triggas en interrupt, varpå MCU:n läser in vilken knapp/rotationsenkoder som har triggat från skiftregistret.

Implementation av detta skedde genom att koppla upp tre stycken rotationsenkoders och tre stycken tryckknappar parallellt till två stycken skiftregister av typen CD74HCT165E [7] och två stycken CD4072BE OR-grindar [8]. Medan rollen för skiftregistrena var att överföra användarinput seriellt användes CD4072BE-grindarna för att kunna konsolidera samtliga knapptryck till en specifik avbrottssignal. Anledningen till att OR-grindarna behövdes var för att undvika att trigga samtliga signaler i shift registret (mao. hade det aldrig gått att sammankoppla samtliga inputsignaler till en “kabel” för interrupten).

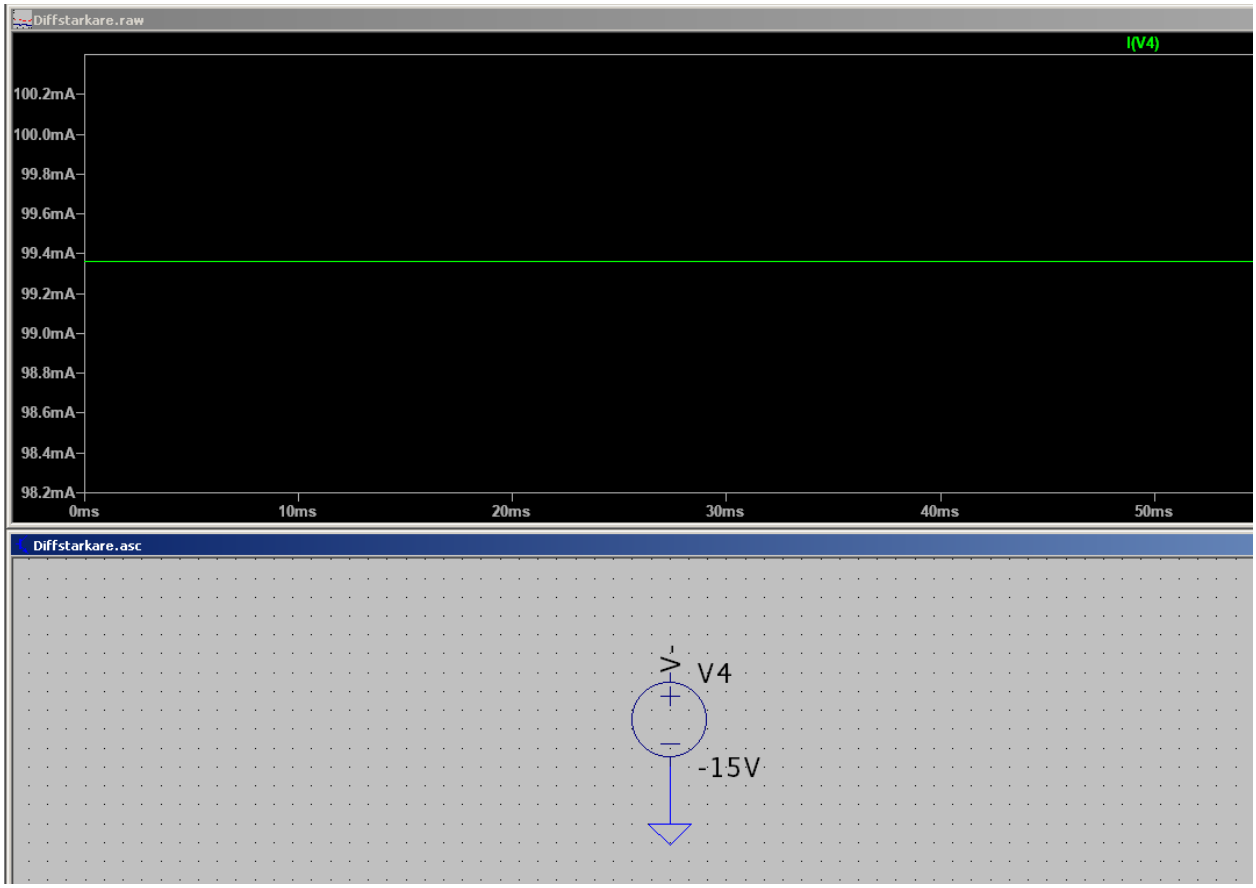
Då samtlig logik till och med serialisering av användarinput befinner sig på dotterkortet, som monteras på frontpanelen av signalgeneratorn sker överföring av den serialiserad datan via en flatkabel (J3, se Bilaga 2) till moderkortet.

7.4 Negativ spänningsförsörjning

Då signalgeneratorn behöver kunna generera negativa spänningar för att kunna avge användbara signaler, men strömadaptern till signalgeneratorn enbart har möjlighet att tillhandahålla +15V behövdes även en hjälpkrets för negativ strömförsörjning, vilket löstes med hjälp av DC-DC-konverteraren MC34063ECN i en spänningsinverterande konfiguration [6, s.11].

För att anpassa nämnd konfiguration till våra krav valdes följande värden (vissa var dock redan givna utifrån hur övriga kretsen är konfigurerad):

- I_{out} = 300mA: antaget värde för att ha vissa marginaler för den analoga delarna av signalgeneratorn. Vid simulation (se figur 9) visade sig förbrukningen vara ungefär hälften av detta.
- V_{in} = 15V: ty signalgeneratorn drivs av en 15V strömadapter
- V_{out} = -15V: för att uppnå en viss marginal för +/-10V peak to peak.
- f_{min} = 50*10³ Hz: Genom att hålla switchingfrekvensen hög blir krusningar lättare att filtrera bort för att få en renare strömförsörjning till de analoga komponenterna.
- V_{ripple} = 0.5V: i och med att en kapacitansmultiplikator är placerad efter inverterarkretsen kommer krusningar i nämnd storleksordning inte vara något större problem att filtrera.
- V_{sat}=0.8V: Mättnadsspänning för darlingtonkopplingen i MC34063; ges av tredje grafen vid [6, s. 6].



Figur 9 - Simulering av signalgenerator.

Efter givna värden var bestämda behövde flertalet parametrar beräknas, vars formler står att finna i [6, s.11]. Komponenternas beteckningar är här angivna utifrån de beteckningar från databladet; längre ned i denna sektion följer en nyckel för att kunna tolka in nämnda komponenter i det faktiska kretsschemat.

- t_{on}/t_{off}: i och med att MC34063 är en hysteresisbaserad DC-DC-konverterare hanteras i huvudsak spänning, ström etc. som passerar igenom den via ration av hur länge den är på respektive av (och därmed laddar/laddar ur spolen i kretsen).
- $t_{on}+t_{off}$: Total periodtid för kontrollern, vilket beräknas utifrån tidigare bestämd frekvens (f_{min}).
- t_{off}: beräknas från tidigare bestämd ratio och periodtid. Tid som kontrollern laddar ur spolen.
- t_{on}: Beräknas från tidigare beräknad t_{off} och periodtid f_{min}; tid som DC-DC-konverteraren laddar spolen.
- C_T: Storlek på timing-kondensator. Laddnings/Urladdningstid av denna kondensator styr periodtiden hos DC-DC-konverteraren.
- R_{sc}: Storlek på sense-resistorn, vilken används av DC-DC-konverteraren för att kunna mäta hur mycket ström som passerar genom spolen.

- L_{min}: Minsta storlek som krävs för spolen som används för att utföra spänningskonverteringen.
- C_0: Minimistorlek på bypasskondensatorn vid utgången.
- R1 resp. R2: Spänningsdelare som används för att styra storleken på utgångsspänningen V_{out}.

Beräkningsgången som sedan uppstod är följande:

$$t_{on}/t_{off} = \frac{|V_{out}|+V_f}{V_{in}-V_{sat}} = 1.098592 = t_r \quad (7)$$

$$t_{on} + t_{off} = \frac{1}{50*10^3} = 2 * 10^{-5} = tp \text{ [s]} \quad (8)$$

$$t_{off} = \frac{t_p}{t_r+1} = 9.530201 * 10^{-6} \text{[s]} \quad (9)$$

$$t_{on} = t_p - t_{off} = 1.04698 * 10^{-5} \text{[s]} \quad (10)$$

$$C_T = 4 * 10^{-5} * t_{on} = 4.187919 * 10^{-10} F \Rightarrow C_T \approx 419pF \Rightarrow välj\ C_T = 430pF \quad (11)$$

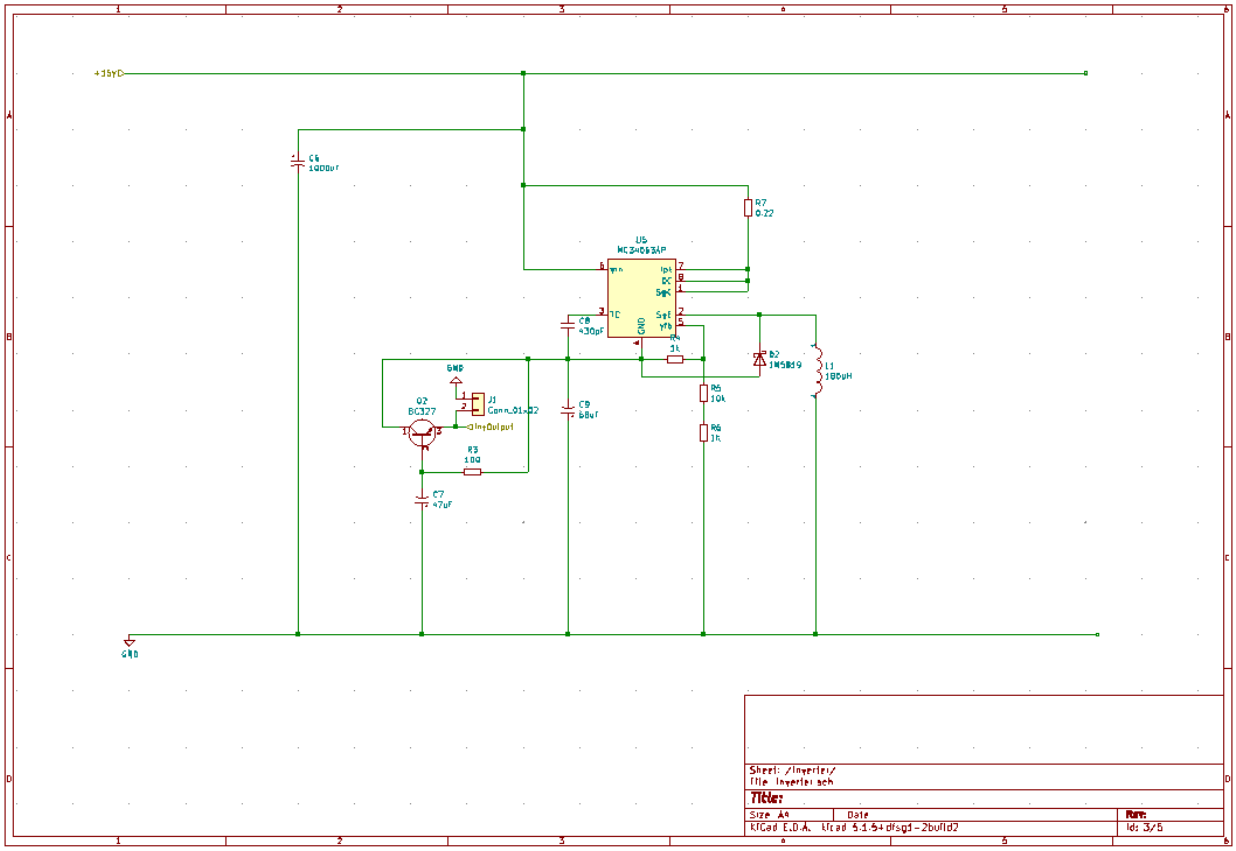
$$I_{pkswitch} = 2 * I_{out}(t_r + 1) = 1.259155 \text{[A]} \quad (12)$$

$$R_{sc} = \frac{0.3}{I_{pkswitch}} = 0.23855\ ohm \Rightarrow välj\ R_{sc} = 220\ mOhm \quad (13)$$

$$L_{min} = t_{on} * \frac{V_{in}-V_{sat}}{I_{pkswitch}} = 1.180722 * 10^{-4} H = 118.07\ \mu H \Rightarrow välj\ minst\ 120\ \mu H \quad (14)$$

$$C_0 = 9 * \frac{I_{out}*t_{on}}{V_{ripple}} = 5.653691 * 10^{-5} = 56.53\ \mu F \Rightarrow välj\ 68\ \mu F \Rightarrow V_{ripple} = \frac{9*I_{out}*t_{on}}{68*10^{-6}} = 416\ mV$$
(15)

$$V_{out} = -1.25 * (1 + \frac{R_2}{R_1}) \Rightarrow \frac{V_{out}}{-1.25} = 1 + \frac{R_2}{R_1} <=> \frac{V_{out}}{-1.25} - 1 = \frac{R_2}{R_1} = 11 \Rightarrow 11k = R_2\ 1k = R_1$$
(16)



Figur 10 - Kretsschema av DC-DC konverterare för negativ spänning.

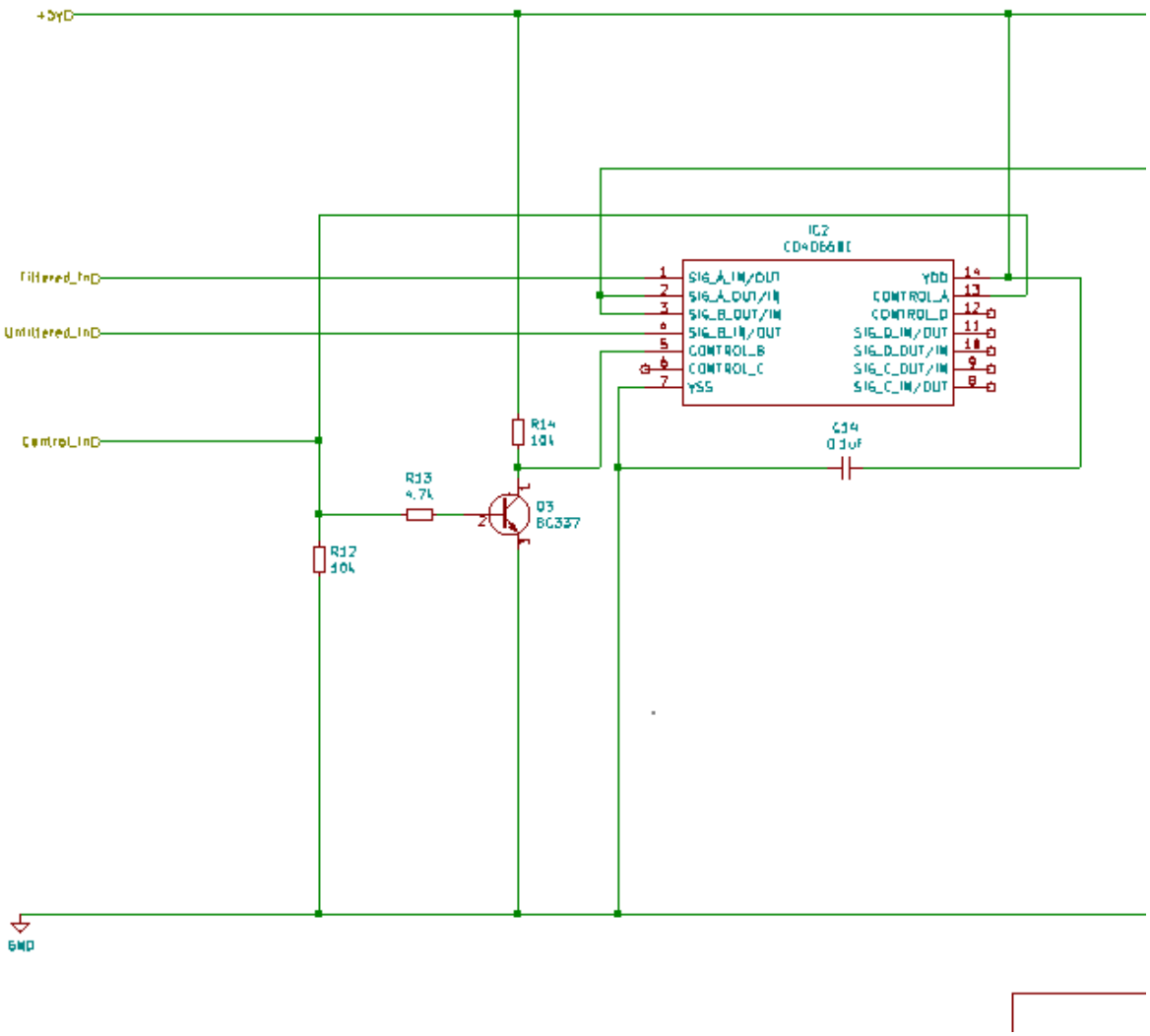
Komponenter i figur 10	Komponenter från datablad och formler
R_{sc}	R7
C8	C_T
C9	C_0
R5	R_1
R6	R_2
L1	L_{min}

7.5 Signalväljare

Då enbart sinus-signalen skall filtreras, medan övriga signaler skall vara ofiltrerade (se sektion 2.5) behövs en signalväljare som styr huruvida den filtrerade eller ofiltrerade signalen fortsätter vidare in i offsethanteringen (se figur 11) innan den slutligen når utgången på funktionsgeneratorn.

För att lösa detta används CD4066 (benämnd som IC2 i kretsschemat), vilken består av en uppsättning CMOS-switchar [10], där 2 av dem används för att välja filtrerad respektive ofiltrerad signal. I och med att CD4066 innehåller spearata switchar som styrs via separata pinnar användes en transistor (Q3) för att minimera antalet IO-pinnar som krävs från MCU:n (1 istället för 2).

Transistorns roll i detta är att när signalen “Control_In” går hög satureras Q3, varpå CONTROL_B på IC2 går låg medan Control_In-signalen går hög, och därmed aktiverar SIG_A_OUT (pinne 2 på IC2), vilket resulterar i att den filtrerade signalen kommer ut på “Output”. När Control_In går låg går CONTROL_A låg, vilket i sin tur stänger Q3 och därmed gör att CONTROL_B går hög, vilket resulterar it att SIG_B_OUT aktiveras (IC2) och den ofiltrerad signalen skickas vidare på “Output”.



Figur 11 - Kretsschema för signalväljare.

7.6 Filter

Eftersom mikrokontrollern skickar ut digitala signaler är dessa inte lämpliga för att generera sinussignaler som då får en trappform istället för den karaktäristiska mjuka form som är önskvärd. Därav används ett lågpassfilter i form av ett butterworth-filter bestående av fyra poler med olika placering. I detta filter används två “Sallen and Key” kretsar i serie för att skapa och placera dessa poler för att få önskat beteende av filtret.

Butterworth med fyra poler valdes för att det ger en bra filtrering av signalen med färre komponenter. Detta då fler poler ger en renare signal enligt [1, s. 401] men för varje polpar som införs medför det två extra operationsförstärkare och medföljande kringkomponenter vilket leder till mer komplexitet och högre kostnad för produkten.

Beräkningar som utfördes för att ta fram värden på komponenterna i filtret

Laplacetransform för filtret (hämtades från [3]):
(1+0.765s+s2)(1+1.848s+s2) (17)

Brytfrekvensen f_b sattes till 300 kHz och den omvandlades till radianer via:
$\omega = f_b \cdot 2\pi = 4,0585\, rad/s$ (18)

För den första delen av filtret valdes C_2 till 100 *pF* som är ett typiskt värde i filter av denna typ och för att underlätta beräkningar och att det inte gör någon större skillnad så sattes också $R_1 = R_2$.
Dessa värden fördes in i dessa formler:
$\frac{0,765}{\omega} = (R_1 + R_2)C_2$ (19)
$\frac{1}{\omega} = R_1^2 C_1 C_2$ (20)
Där 0,765 kommer ifrån den första faktorn i Laplacetransformen.

Resultat: $R_1 = R_2 \approx 2,029 \cdot 10^3\ \Omega$ och $C_1 \approx 683$ pF

För den andra delen av filtret valdes även C_4 till 100 pF och $R_3 = R_4$ och med samma formler (förutom att 0,765 byttes mot 1,848) fås:

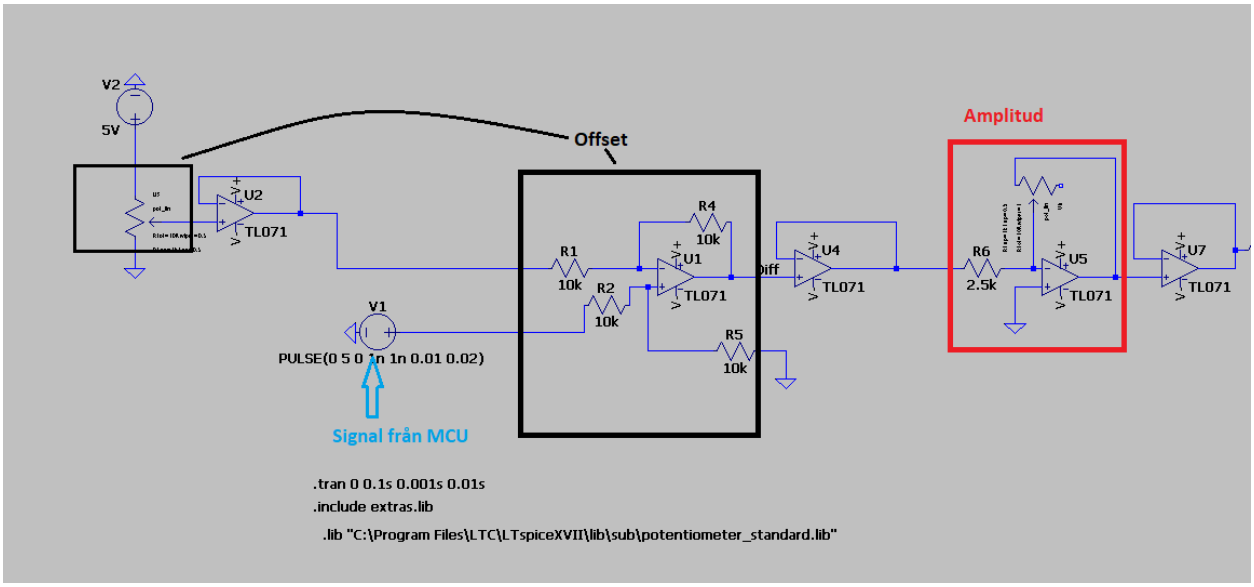
Resultat: $R_3 = R_4 \approx 4,9$ kΩ och $C_3 = 117$ pF.

Komponenterna valdes sedan ur lämpliga standardserier till:
$R_1 = R_2 = 2$ kΩ

$R_3 = R_4 = 4{,}7\ k\Omega$
$C_2 = C_4 = 100\ pF$
$C_1 = 680\ pF$
$C_3 = 120\ pF$

7.7 Offsethantering och amplitudförstärkning

För att hantera offset och amplitud i utsignalen används sex stycken operationsförstärkare, åtta motstånd och två digitala potentiometrar för att uppnå önskvärd funktionalitet. Potentiometrarna styrs genom digitala signaler från MCU och det är potentiometrarna i sin tur som kontrollerar offset och amplitud genom att förändra spänningsförhållandet på ingången till respektive operationsförstärkare. Nämligen är det bara två av dem som påverkar offset och amplitud, de andra är där som impedansbuffrar vid ingång, utgång och mellan offsetkretsen och amplitudkretsen för att förhindra att kretsarna påverkar varandra.



Figur 12 - Offset- och amplitudkrets.

Offsetkretsen består (som ses i figur 12) av en operationsförstärkare och fyra motstånd, kopplad som en differensförstärkare som här egentligen inte används för sitt tilltänkta syfte. Istället för att kompensera för störningar som är dess egentliga syfte skapas en spänningsdifferens vid ingången med hjälp av potentiometern varpå operationsförstärkaren kompenserar sin utgång för spänningsdifferensen och därmed flyttar “nollpunkten” för inkommande signal. För att skifta offset ändras den digitala potentiometern för att ändra spänningsdelningen och resulterar därmed i en annan utgångsspänning på differensförstärkaren. Det är viktigt att alla motståndens värden är nära varandra, allra helst identiska för att förstärkningen ska vara 1 då syftet är att flytta nollpunkten och inte att förstärka. Därför används en motståndsarray med 10kΩ motstånd, då dessa har en större sannolikhet att vara närapå identiska. Motstånden är också till för att öka impedansen i kretsen.

Amplitudkretsen består av en operationsförstärkare som är kopplad i en inverterande konfiguration där det vanliga motståndet är utbytt mot en digital potentiometer som styr spänningen vid ingången. På detta sätt kan amplituden för inkommande signal justeras från MCU. R6 i amplitudkretsen är valt till 2.5 kΩ utifrån att förstärkningen (A) är lika med kvoten mellan potentiometern och R6, med detta värde fås förstärkningen A=4. Detta medför att amplituden kan varieras från 0 till 10 V vilket ger 20 V peak-to-peak vid till exempel en sinuskurva.

För operationsförstärkare används kretsarna TL074 och TL072 [11], motståndsarrayen en 4608X-102-103LF [12] och de digitala potentiometrarna är en MCP4251-103E_P [13].

7.8 Display

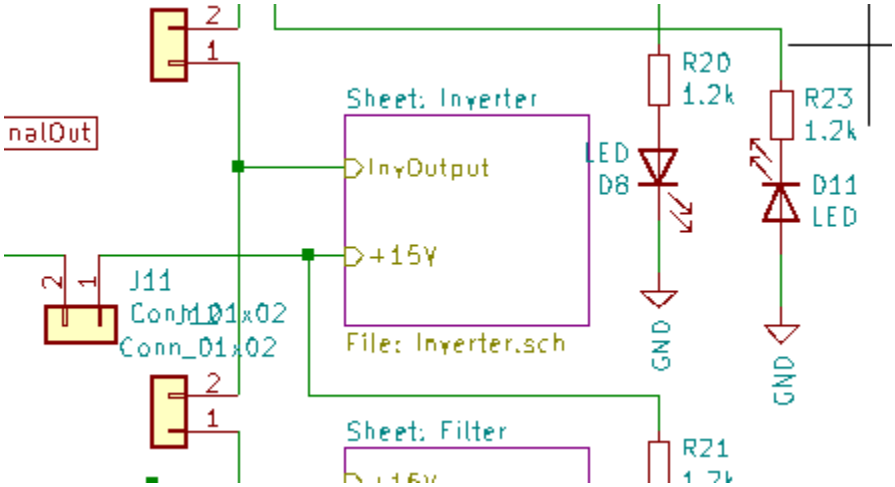
För att underlätta användningen av signalgeneratorn används en 2004A ASCII-display [14] i form av en displaymodul, monterad på dotterkortet. Displayen kan visa upp till fyra rader och 20 tecken per rad. Likt många andra ASCII-displayer är den pin-kompatibel med Hitachi HD44780-kontrollern [15] och har LED-bakgrundsbelysning.

7.9 Konnektivitet mellan moderkort och dotterkort

För att koppla ihop moderkort och dotterkort används en flatkabel med 28 ledare där varannan ledare är jordad för att minska störningar. Flatkabeln kopplas in via fastlödda sockets till respektive kort.

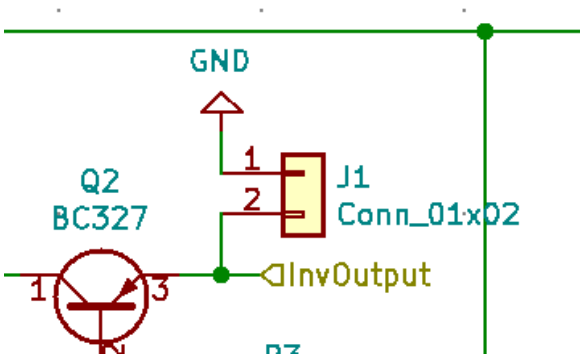
7.10 Funktionalitet för felsökning under prototypstadiet

För att förenkla felsökning har de olika “sektionerna” av kretsen utrustats med LEDar och jumperkopplingar (se figur 13), där LEDarnas uppgift är att indikera huruvida en komponent får ström, medan jumperkopplingarna användas för att kunna testa funktionsgeneratorn inkrementellt, och därmed undvika att t.ex. MCU:n brinner upp för att något var fel med strömförsörjningen.



Figur 13 - Exempel från kretsscheman på LED och jumper placering.

I och med att DC-DC-konverteraren behöver en last för att fungera infogades även en socket (J1 i figur 14) för att kunna addera ett temporärt lastmotstånd vid testning.



Figur 14 - Kretsschema med jumper 1 vid DC-DC-konverteraren.

8. Utformning av PCB

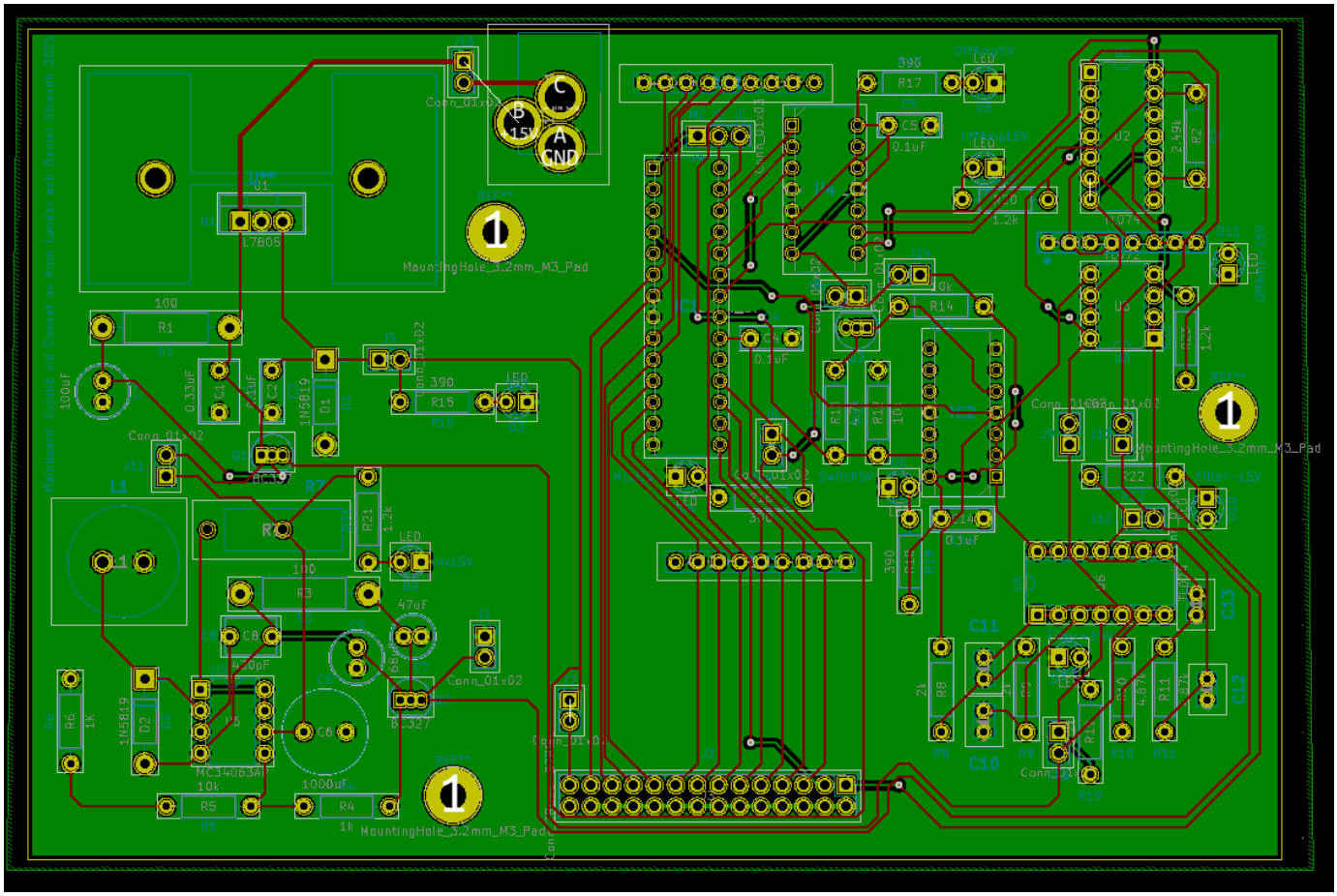
För att designa PCB:erna användes KiCads inbyggda PCB-verktyg då det var smidigt att överföra kretsschemat direkt dit. Där fanns ett så kallat *Rat's nest* (linjer som indikerar vilka in- och utgångar som kopplas samman för att uppfylla kretsschemats design) automatiskt genom konverteringen från kretsschema till PCB-modell och komponenterna hade ofta redan färdiga footprints (PCB-modeller av komponenterna som motsvarar de verkliga komponenterna). Dock fick detta bibliotek kompletteras allteftersom med fler footprints till de komponenter som saknades. I projektet används två stycken PCB:er, ett moderkort och ett dotterkort varav moderkortet huserar mikrokontroller och dess kringkomponenter, strömförsörjning och spänningskonverterting medan dotterkortet huserar display och hantering av input från användaren.

Under processens gång framkom det tydligt att placeringen av komponenter var det viktigaste att tänka på, för om komponenter inte placerades väl var det svårt att dra ledarna utan att de korsade varandra. Därför ägnades mer tid till än planerat att placera komponenterna och rotera dem för att underlätta dragning av ledare än att dra själva ledarna, och i vissa fall ändrades även kretsschemat för att underlätta placeringen och ledningsdragningen.

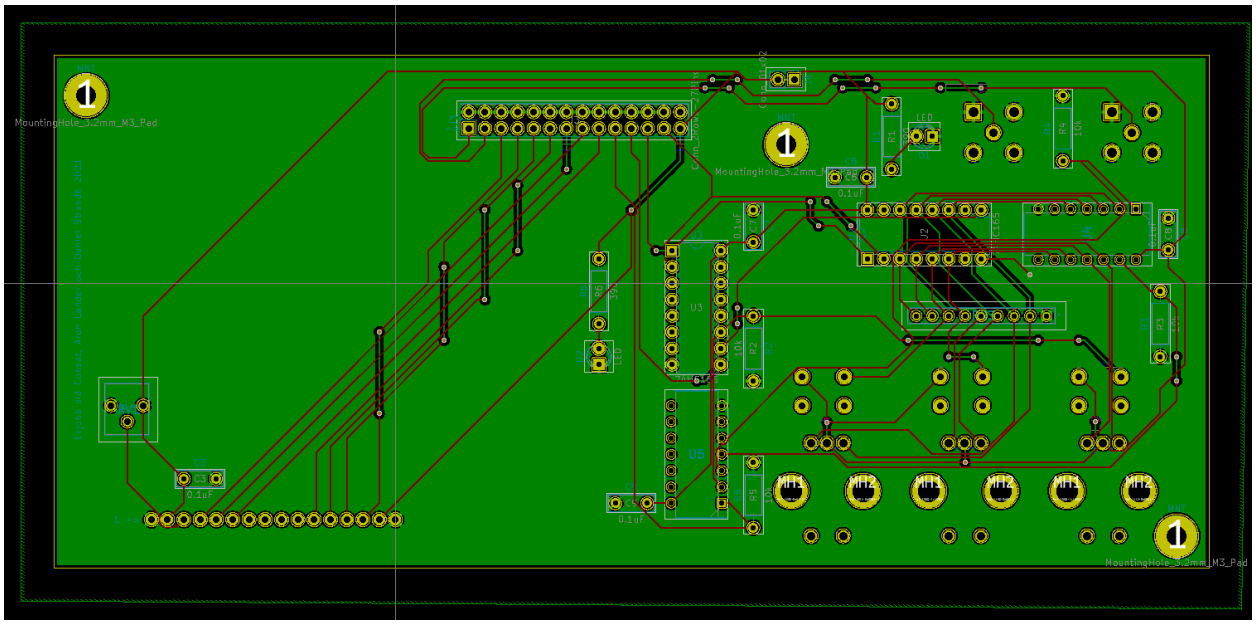
Båda PCB:erna består av två skikt, ett nästintill heltäckande jordskikt och ett skikt där de flesta ledningarna drogs. I vissa fall när ledningarna var tvungna att korsas användes jordskiktet för att gå under den andra ledningen (detta syns i figur 15 och 16 som en svart linje med en grön linje inuti). Det gröna fältet är jordskiktet och man vill undvika att använda det för ledningsdragning så långt det går för att elektronerna ska kunna röra sig fritt mot jord.

Layouten på moderkortet är upplagt med strömförsörjningskomponenterna på vänster sida och de digitala komponenterna på höger sida. Detta för att minska störningar på de digitala kretsarna som kan komma via jordplanet från strömförsörjningssidan.

Dotterkortets layout är baserat på de första designskisserna (figur 3) som togs fram för användargränssnittet. Därmed sitter skärmen på vänster sida på baksidan av PCB:n och knappar och Rotationsenkoders på höger sida på baksidan. Syftet med att de sitter på baksidan av kortet är att de då sitter mot frontpanelen och därmed är åtkomliga för användaren och det sparar plats på framsidan för logiken för användargränssnittet genom att dela upp komponenterna.



Figur 15 - PCB-modell för moderkortet



Figur 16 - PCB-modell för dotterkortet

9. Kodarkitektur

För att binda ihop samtliga komponenter i signalgeneratorn behövdes en mikrokontroller (MCU), vilken självfallet inte har förmågan att göra något om den inte har någon programkod att exekvera. För att uppnå detta användes programmeringsspråket C, då det har goda lågnivåfaciliteter (såsom bitmanipulation och manuell minneshantering i form av råa pekare).

För att göra koden mer överblickbar användes psuedoklasser, ty genom att använda så kallade “structar” med funktionspekare i C går objektorienterade koncept att emulera tämligen väl.

Den önskade funktionaliteten uppnåddes genom att koden (beskriven tidigare i dokumentet) delades upp i två huvudsakliga sektioner: Bibliotek (Libraries) och Drivrutiner (Drivers), där drivrutinernas syfte är att interagera med komponenterna på en lägre nivå (t.ex. via SPI eller direkt interaktion med pinnar på MCU:n) medan biblioteken tillhandahåller högnivåfunktioner, såsom att generera strängar som skall visas på displayen. Slutligen bands dessa komponenter samman genom att förlägga intialiseringsprocessen i “main.c”, som körs vid uppstart av MCU:n.

9.1 Utvecklingsmiljö

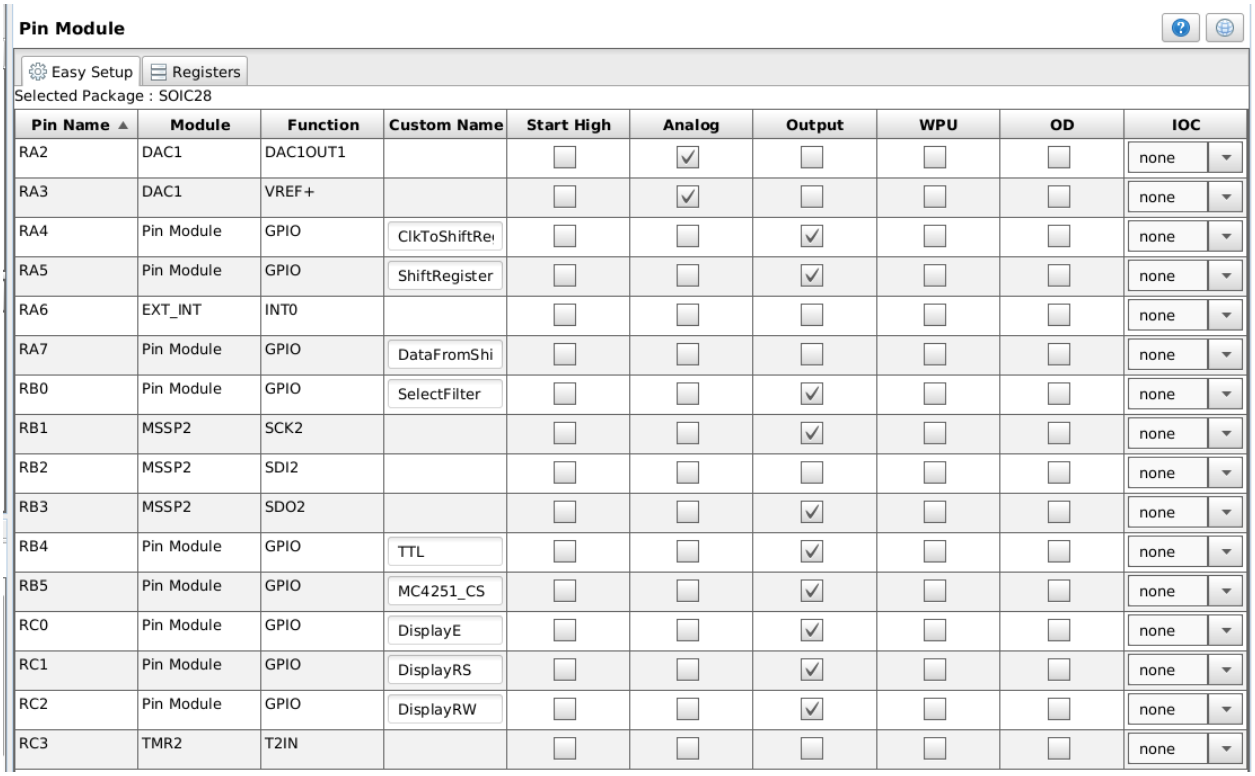
Då den valda MCU:n var av typen “PIC18” från Microchip, användes Microchips utvecklingsmiljö “MPLABX”, vilken även tillhandahöll ett verktyg vid namn “MCC”. MCC tillhandahöll möjligheten att generera kod för konfiguration av ingångar, utgångar (“pinnar”) och flertalet inbyggda moduler i MCU:n, vilket möjliggjorde en kortare utvecklingstid.

Nedan följer en överblick av vad som förkonfigurerades via detta verktyg.

Pin Module

Genererade filer: mcc_generated_files/pin_manager-h och mcc_generated_files/pin_manager.c

För att konfigurera de olika in- och utgångarna på MCU:n användes “Pin Manager”-vyn i MCC-verktyget. För att se hur respektive ben konfigurerades, se figur 17 och följande beskrivning.



Pin Module

Easy Setup | Registers

Selected Package : SOIC28

Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RA2	DAC1	DAC1OUT1		☐	☑	☐	☐	☐	none
RA3	DAC1	VREF+		☐	☑	☐	☐	☐	none
RA4	Pin Module	GPIO	ClkToShiftRe	☐	☐	☑	☐	☐	none
RA5	Pin Module	GPIO	ShiftRegister	☐	☐	☑	☐	☐	none
RA6	EXT_INT	INT0		☐	☐	☐	☐	☐	none
RA7	Pin Module	GPIO	DataFromShi	☐	☐	☐	☐	☐	none
RB0	Pin Module	GPIO	SelectFilter	☐	☐	☑	☐	☐	none
RB1	MSSP2	SCK2		☐	☐	☑	☐	☐	none
RB2	MSSP2	SDI2		☐	☐	☐	☐	☐	none
RB3	MSSP2	SDO2		☐	☐	☑	☐	☐	none
RB4	Pin Module	GPIO	TTL	☐	☐	☑	☐	☐	none
RB5	Pin Module	GPIO	MC4251_CS	☐	☐	☑	☐	☐	none
RC0	Pin Module	GPIO	DisplayE	☐	☐	☑	☐	☐	none
RC1	Pin Module	GPIO	DisplayRS	☐	☐	☑	☐	☐	none
RC2	Pin Module	GPIO	DisplayRW	☐	☐	☑	☐	☐	none
RC3	TMR2	T2IN		☐	☐	☐	☐	☐	none

Figur 17 - Konfiguration av de olika pinnarna på MCU:n.

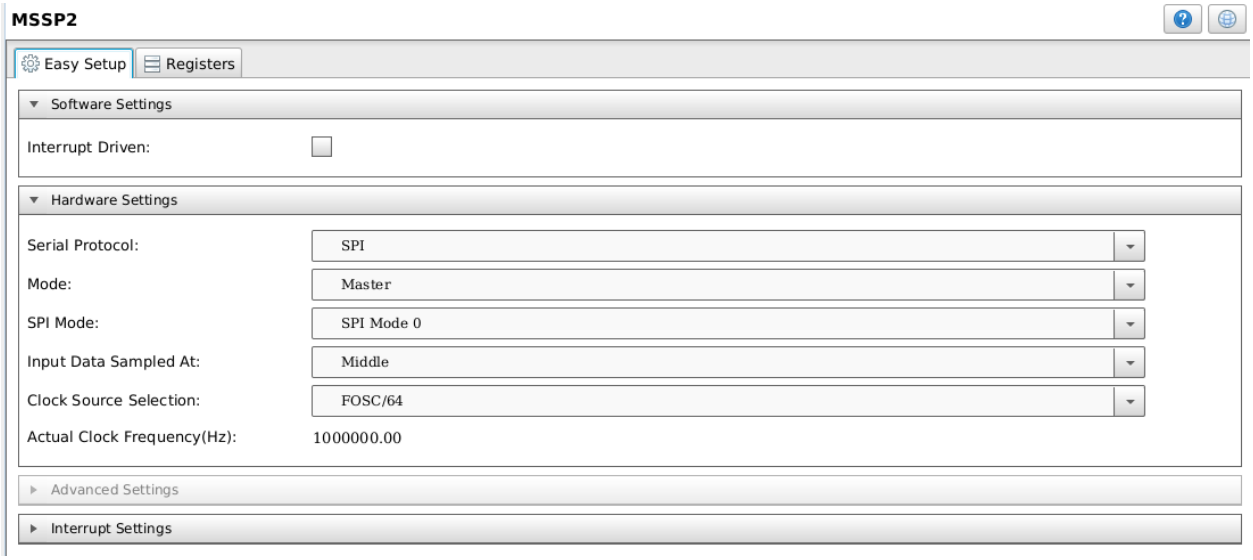
- RA2 - Utgång för MCU:ns inbyggda DAC
- RA4 - Klockutgång till de 2 skiftregister som monterades på dotterkortet (se “7.3 Hantering av användarinput”)
- RA5 - Input Enable-signal till skiftregister för användarinputhantering.
- RA6 - Interruptingång för att fånga upp när en knapp eller rotary encoder används in användargränssnittet på dotterkortet
- RA7 - Används för att läsa in data från skiftregistren i användarinputhanteringen
- RB0 - Styr signalväljarkretsen för att styra huruvida utsignalen skall var filtrerad eller ej (se “7.5 Signalväljare”)
- RB1, RB2, RB3 och RB5 - Används för SPI-kommunikation mellan den digitala potentiometer som styr filter och offset (se “7.6 Filter” respektive “7.7 Offsethantering och amplitud”)
- RB4 - Var tänkt att användas för triggersignal ut, tyvärr hann denna funktionalitet aldrig implementeras.
- RC1, RC2, RC3, RC4, RC5, RC6 och RC7 - Används för kommunikation mellan MCU och ASCII-display.

SPI

Genererade filer: mcc_generated_files/spi2.c och mcc/generated_files/spi2.h

Då den digitala potentiometern kommunicerar med MCU:n via SPI behövde SPI-modulen på MCU:n konfigureras (se figur 18), vilket tillhandahöll den funktionalitet som behövdes för att blanda annat läsa och skriva bytes till och från den digitala potentiometern (MCP4251).

I och med att MCU:n var ämnad att styra MCP4251 konfigurerades SPI-modulen på MCU:n att agera i Master-mode, medan SPI Mode respektive klockfrekvens (clock source) selection valdes utifrån databladets specifikationer [13].

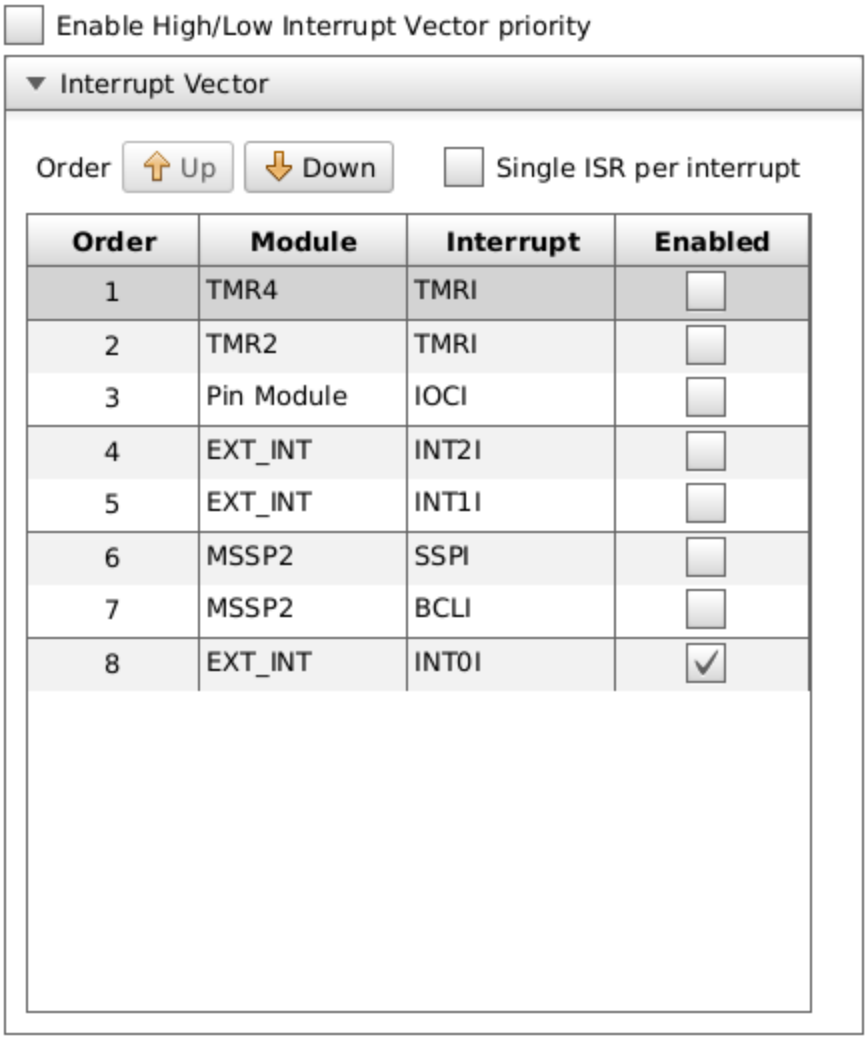


Figur 18 - Konfiguration av SPI-modul i MCC

Interrupts (avbrott)

Generade filer: mcc_generated_files/ext_int.c, mcc_generated_files/ext_int.h, mcc_generated_files/interrupt_manager.c och mcc_generated_files/interrupt_manager.h

Från början var tanken att använda avbrott både för frekvensgenerering och för input från användargränssnittet, dock visade det sig att avbrottbaserad signalgenerering inte gav några märkbara prestandafördelar, men innebar ökad komplexitet, vilket resulterade i att avbrott i slutändan enbart användes för användarinput, som synes i figur 19.



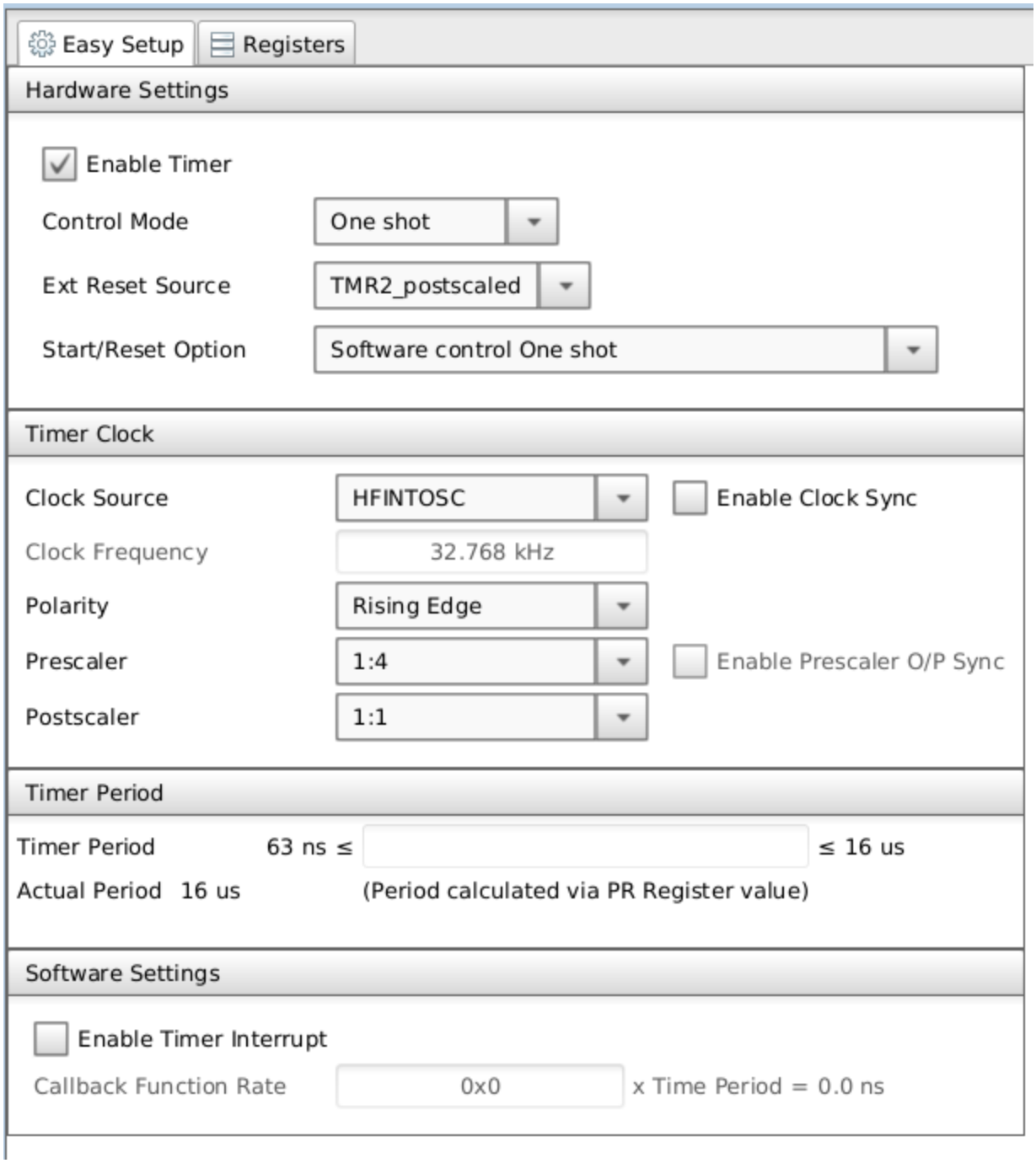
Figur 19 - Konfiguration av abrottsrutiner i MCC

Tillhörande avbrottsrutiner finns beskrivna under sektionen “9.4 Övriga funktioner”.

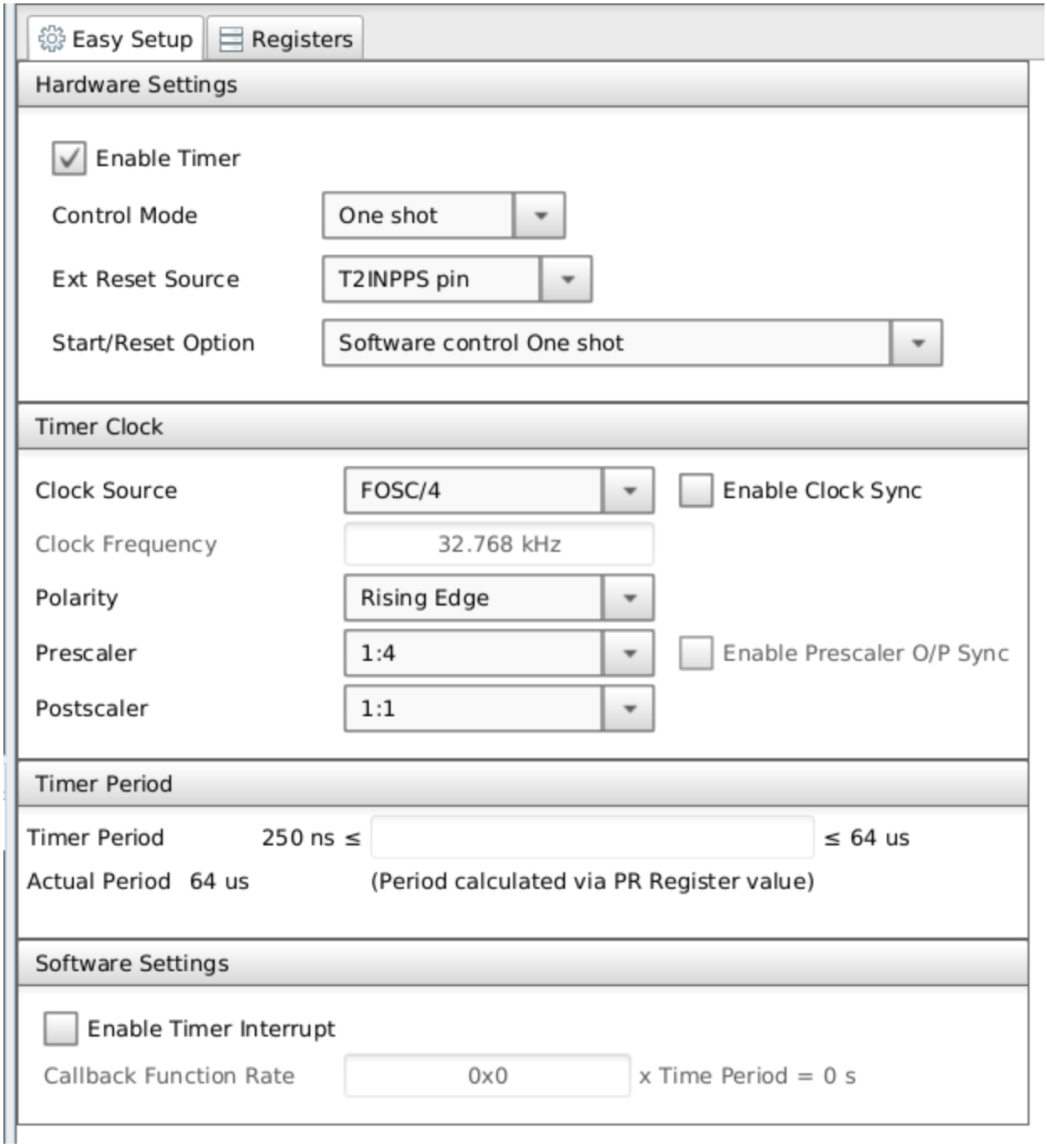
Timers

Genererade filer: mcc_generated_files/tmr4.c, mcc_generated_files/tmr4.h, mcc_generated_files/tmr2.c och mcc_generated_files/tmr2.h

Totalt sett användes två timers för signalgeneratorn, där den första (TMR4, se figur 20) användes för att styra periodtiden (mer om detta under “9.2 Drivrutiner”), medan TMR2 (se figur 21) användes för att hantera de väntetider som krävdes mellan de kommandon som skall skickas till displayen för att uppdatera visad information.



Figur 20 - Konfiguration av TMR4 i MCC, vilket är den timer som användes för att styra periodtiden för signalgenereringen.

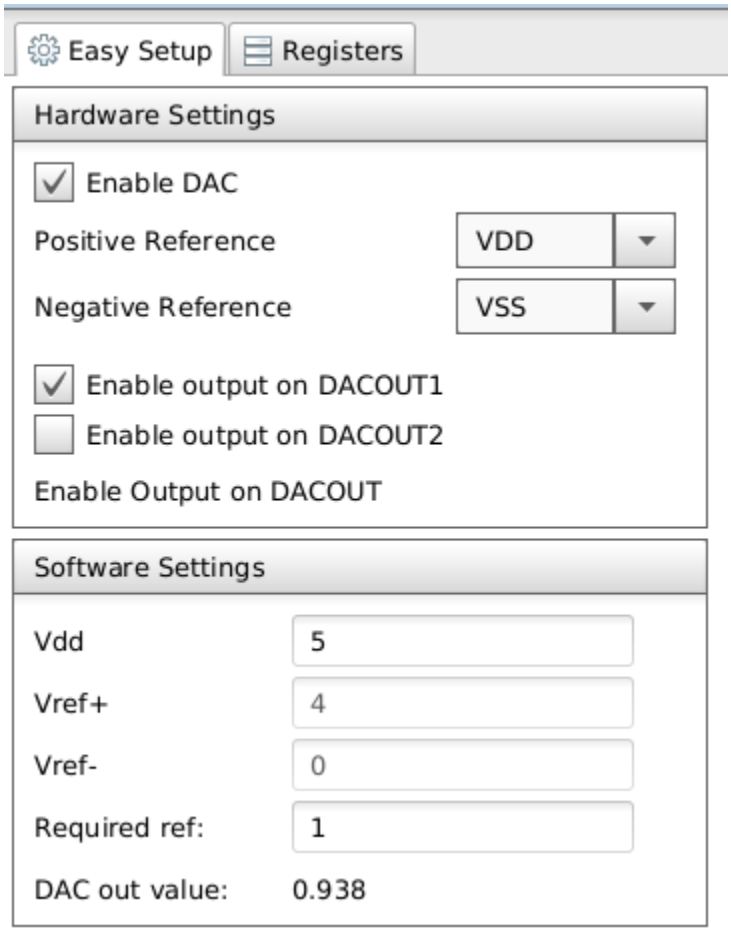


Figur 21 - Konfiguration av TMR2, vilken användes för att generera väntetider för kommunikation till ASCII-displayen. Som synes under “Clock source” och “Timer period” periodtiden hos TMR2 lägre än hos TMR4 då signalgenereringen kräver högre frekvenser för att uppnå önskad funktioanlitet.

DAC

Genererade filer: mcc_generated_files/dac1.c och mcc_generated_files/dac1.h

Då en signalgenerator på något sätt måste kunna skicka ut en analog signal användes den inbyggda 5-bitars-DAC:en. Den enda egentliga konfiguration MCC-verktyget tillhandahöll för detta var att sätta DAC:ens referensspänning, vilken sattes till den strömkälla som användes för själva MCU:n (5V, se figur 22).



Figur 22 - Konfiguration av DAC i MCC

9.2 Drivrutiner

För att binda samman de olika externa komponenterna på PCB:n med signalgeneratorns mjukvara implementerades tre stycken olika drivrutiner.

CDisplayDriver

Filer: Drivers/display.c och Drivers/display.h

Då den displaymodul som använts (tc2004) är en kinesisk kopia av Hitachis HD44780 sker kommunikationen med denna även på samma sätt som för HD44780. Vad som huvudsakligen utmärker denna displaytyp är att den har en buss på 4/8 bitar för data/kommando (beroende på om den körs i 4 eller 8-bitars läge) och 2 flaggbitar för att styra huruvida displayen skall befinna sig i kommando- eller data-läge och huruvida läsning eller skrivning av data från/till displayen skall ske.

Det ansvar som tillfaller denna drivrutin är därmed dels att hantera de tidigare nämnda fördröjningar som krävs för att vänta på att displayen skall utföra sina operationer, att hantera flaggbitarna för de olika lägen som krävs, skicka kommandon till displayen (t.ex. välja var tecken skall skrivas) och läsa/skriva data från/till displayen.

CInputPanel

Filer: Drivers/input_panel.c och Drivers/input_panel.h

På dotterkortet sitter fletalet rotationsenkodare och mikrobrytare som kombinerat med skiftregister och or-grindar som hanterar användarinput, vilken på något sätt måste hanteras. Ansvaret för detta tillfaller Input Panel-drivrutinen (klassnamn: CInputPanel). De två uppgifter den utför är att identifiera vilken mikrobrytare/rotationsenkodare som blivit intryckt/roterade och returnera motsvarande e_input_signal enum till den korresponderande avbrottsrutinen (InputPanelInterrupt) och att styra huruvida inputpanelens skiftregister skall registrera ny data eller tillhandahålla den data de för tillfället håller.

CMCP4251

Filer: Drivers/mc4251.c och Drivers/mc4251.h

För att kommunicera med den digitala potentiometern (MCP4251) genererar drivaren “CMCP4251” de kommandon som behövs för att dels skriva värden till potentiometern (mellan 0 och 256) och dels att styra vilken av de två potentiometrarna som skall styras (en för amplitud och en för offset).

Ett ytterligare ansvar som inte är direkt relaterat till att styra MCP4251-potentiometern som CMCP4251 har är även att verifiera att vald amplitud och offset inte går utanför de tillåtna värdena (med andra ord, de värden som hårdvaran är konfigurerad att hantera), vilket löses genom att offset och amplitud inte kan ändras så att signalen går över +10V respektive -10V.

CMCP4251 använder de underliggande SPI-funktioner som genererades av MCC, och kan därmed ses som en form av lager mellan MCU:ns SPI-modul och övrig kod.

9.3 Bibliotek

CDelayHelper

Filer: Libraries/delay_helper.c och Libraries/delay_helper.h

Då display-drivrutinen behöver vänta in displayen efter att ha skickat data/kommandon till den behövs något sätt att skapa fördröjningar, vilket tillhandahålls av CDelayHelper-klassen. CDelayHelper kan fördrja en rutin i x antal millisekunder, mikrosekunder respektive “block” av 250ns.

CDisplayHelper

Filer: Libraries/delay_helper.c och Libraries/delay_helper.h

CDisplayHelper tillhandahåller dels den funktionalitet som krävs för att skriva ut de olika värden som skall visas på displayen, men har även hand om att skriva ut de textsträngar som indikerar vilket värde som visas (t.ex. vilken typ av signal som är vald).

CWavetableManager

Filer: Libraries/wavetables.c och Libraries/wavetables.h

CWavetableManager innehåller dels de wavetables, vilka är förgenererade tabeller bestående av 32 värden som beskriver t.ex. en sinuskurva, men även den state-information som krävs för att veta vilket värde som skall skickas ut från DAC:en. I och med den tidsbristen som uppstod under projektets slutfas lades aldrig andra typer av signaler till, men CWavetableManager:s kodbas har möjligheten att kunna stödja flera sorters signaltyper än sinus.

CWavetableManager kommunicerar aldrig direkt med DAC:en, utan används av avbrottsrutinen “WaveTableInterrupt” för att ta fram det värde som skall skickas ut till den.

9.4 Övriga funktioner

Utöver de klasser som definerats i form av drivrutiner och bibliotek finns även fristående funktioner i form av interrupt-definitioner; båda i filerna **interrupt_defs.c** respektive **interrupt_defs.h**.

InputPanelInterrupt

När ett avbrott triggas från inputpanelen triggas InputPanelInterrupt, vilken vidare anropar en instans av CInputPanel, som i sin tur identiferar den inkomna signalen, och returnar signaltypen till InputPanelInterrupt som i sin tur agerar på den genom att anropa motsvarande funktion och en instans av CDisplayHelper (anropet till CDisplayHelper sker enbart för att uppdatera den information som visas för användaren).

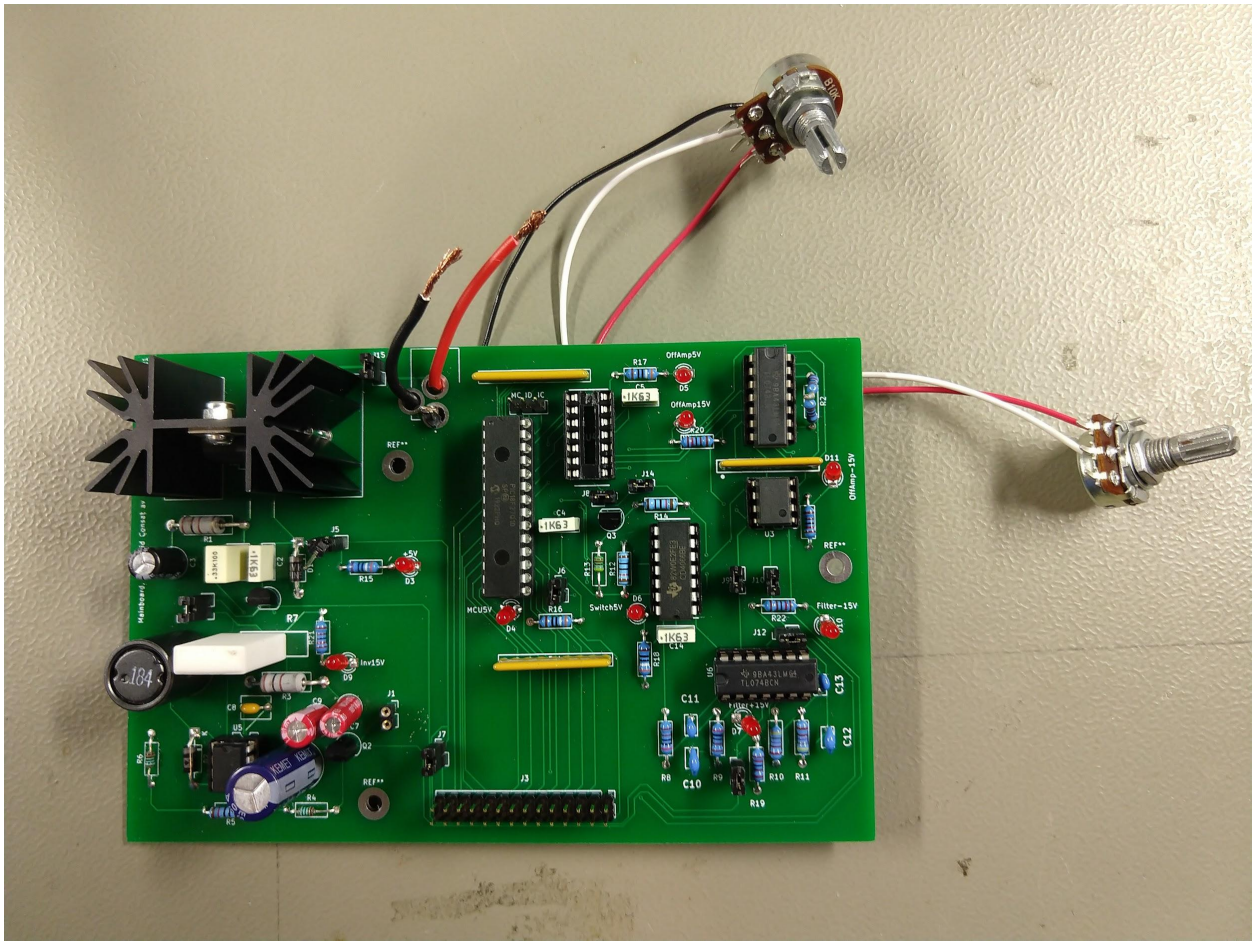
Nämnda signaler kan t.ex. vara byte av signaltyp, ökning/minskning av utsignalens frekvens osv.

10. Slutgiltig montering och testning

För att uppnå en fungerande signalgenerator behövde komponenterna monteras på respektive kretskort. För att slutligen verifiera att förväntad funktionalitet (specificerad i kapitel 1.4 Precisering av frågeställningen).

10.1 Konstruktion

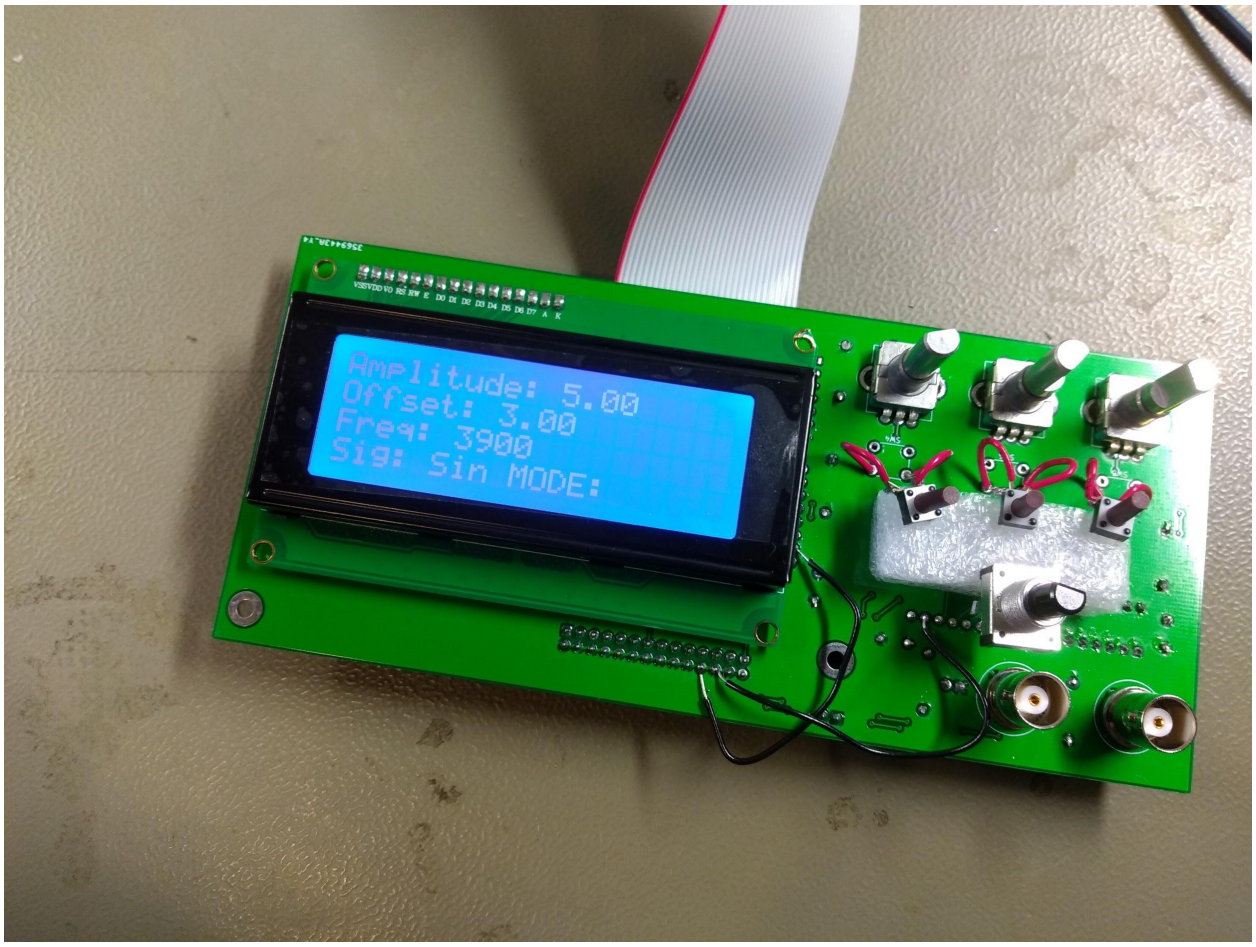
Efter att ha fått hem de två PCB-korten (moderkort och dotterkort) började monteringen av moderkortet, vilken överlag var utan bekymmer, förutom att vi missat 2.49 kOhm-motståndet i beställningen. L7805 var även något för långt från kylflänsen i PCB-designen, men gick att lösa genom att böja dess ben något. Koppling för DAC-vref-inkoppling (referensspänning för den inbyggda digital-till-analog-konverteraren) saknades också i PCB-designen (och kretsschemat) men gick att lösa genom att helt sonika använda använda MCU:ns matningsspänning som referens.



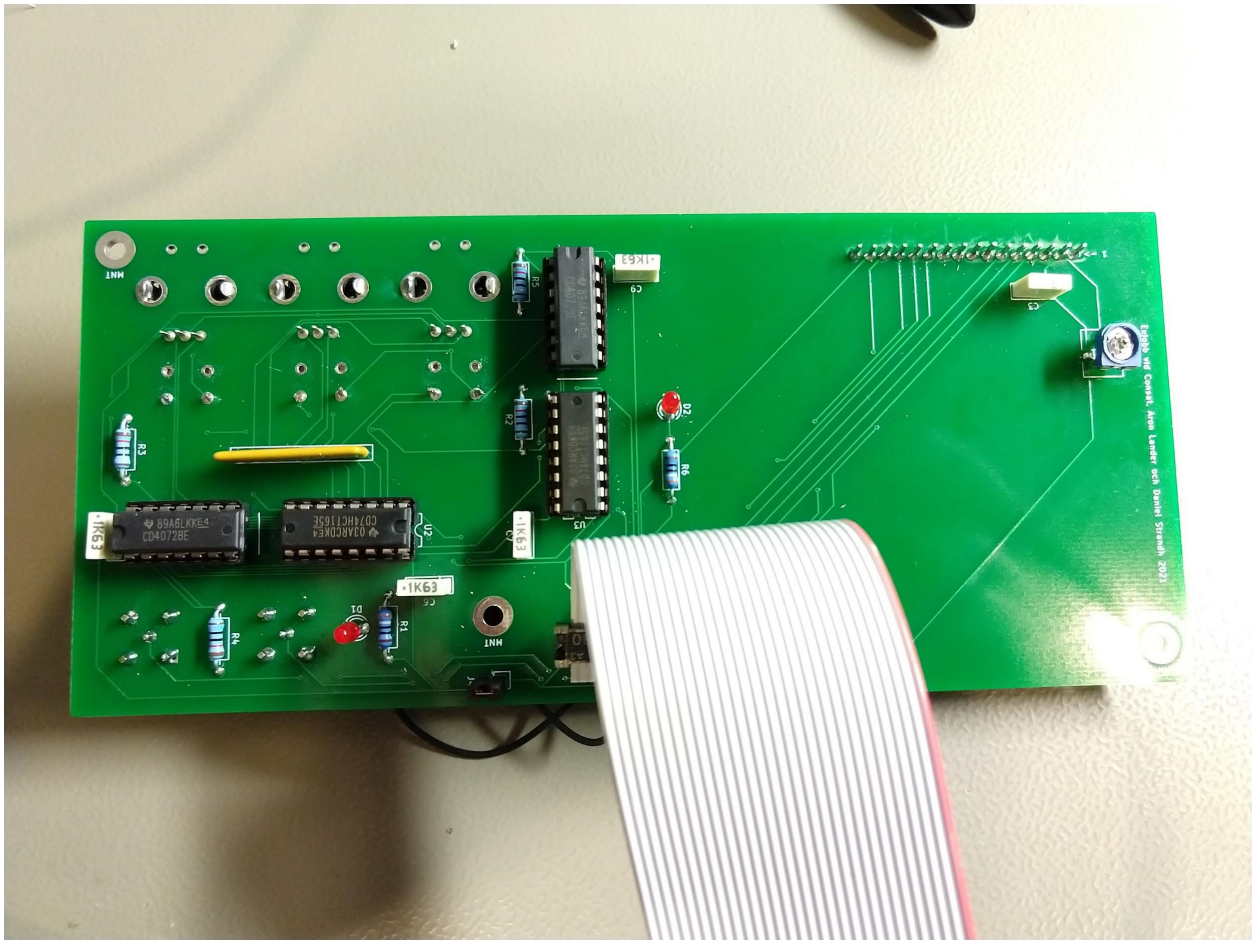
Figur 23 - Moderkortets slutgiligta konstruktion efter modifieringar.

Dotterkortet är det kretskort som är tänkt att sitta som frontpanel, och är därmed det kretskort där skärm och enkodrar är monterade. De designfel som visade sig här var att skiftregistrenas (74hc165) chip enable-pinnar inte var dragna till jord, vilket senare ledde till att de aldrig tog in några signaler. Detta löstes genom att dra kablar mellan jord och dessa ben på kretskortet som syns i figur 24 (de svarta kablarna).

En ändring som gjordes i den sk. footprinten för de taktila brytarna visade sig senare vara felaktig detta då dokumentationen var bristfällig, och ledde därmed till att den faktiska footprinten var felkopplad. Detta ledde till att signalen genom tryckknapparna därmed alltid var +5V, vilket resulterade i att interruptlinjen in till MCU:n alltid var hög, och interrupts kunde därmed aldrig aktiveras. Lösningen var att löda loss dessa knappar, vända på dem och löda kablar mellan dess ben och kretskortet.



Figur 24 - Dotterkort framsida efter modifieringar.



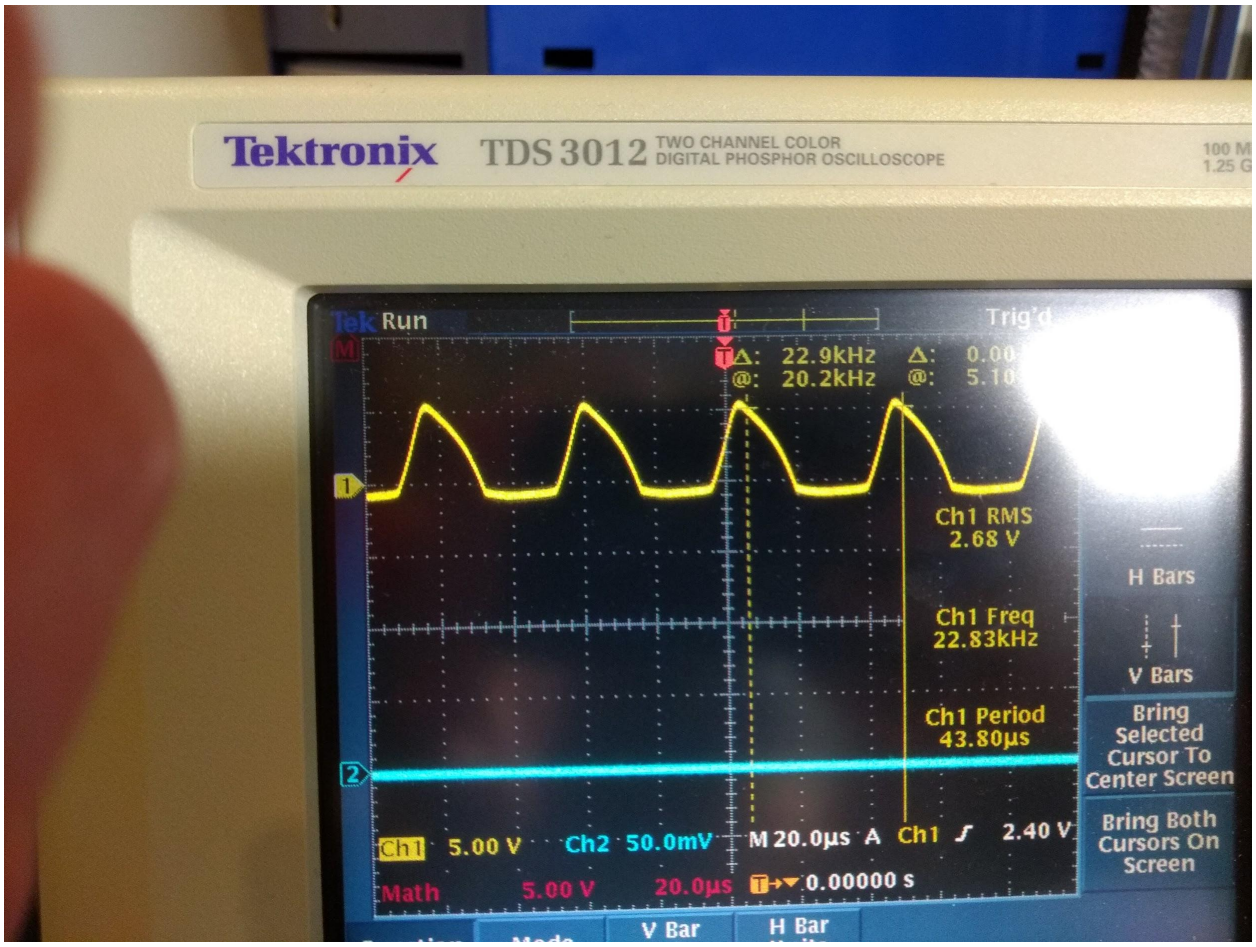
Figur 25 - Dotterkort baksida

10.2 Testning och åtgärder

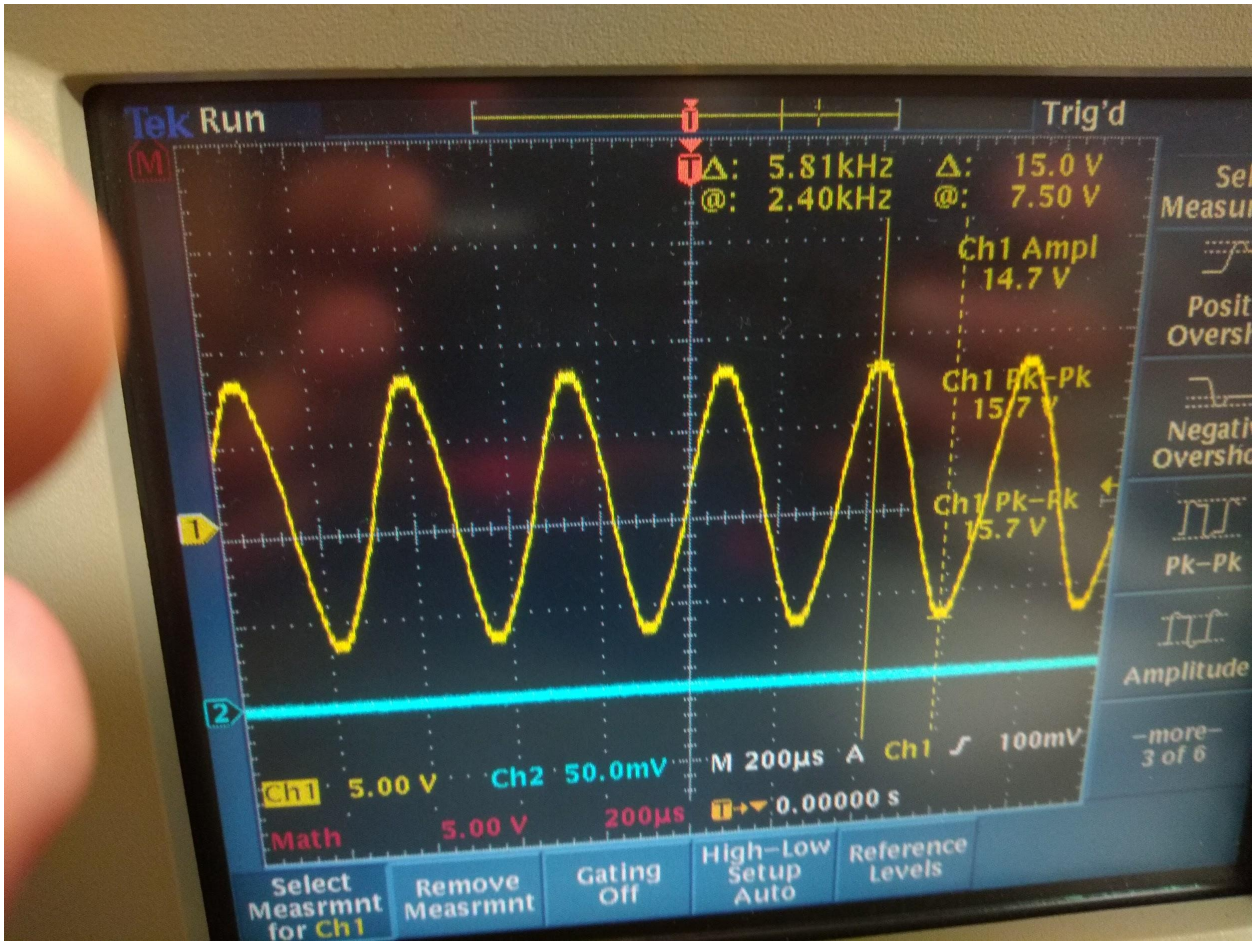
Vid testning uppkom flera problem på vägen, innan en signal ut och den mesta av funktionaliteten kunde erhållas.

Det första problemet som uppkom var att switchmekanismen (benämnd “Switch” i kretsschemat) aldrig aktiverades i koden, vilket resulterade i att signalen från filtret aldrig gick vidare till offset/förstärkning.

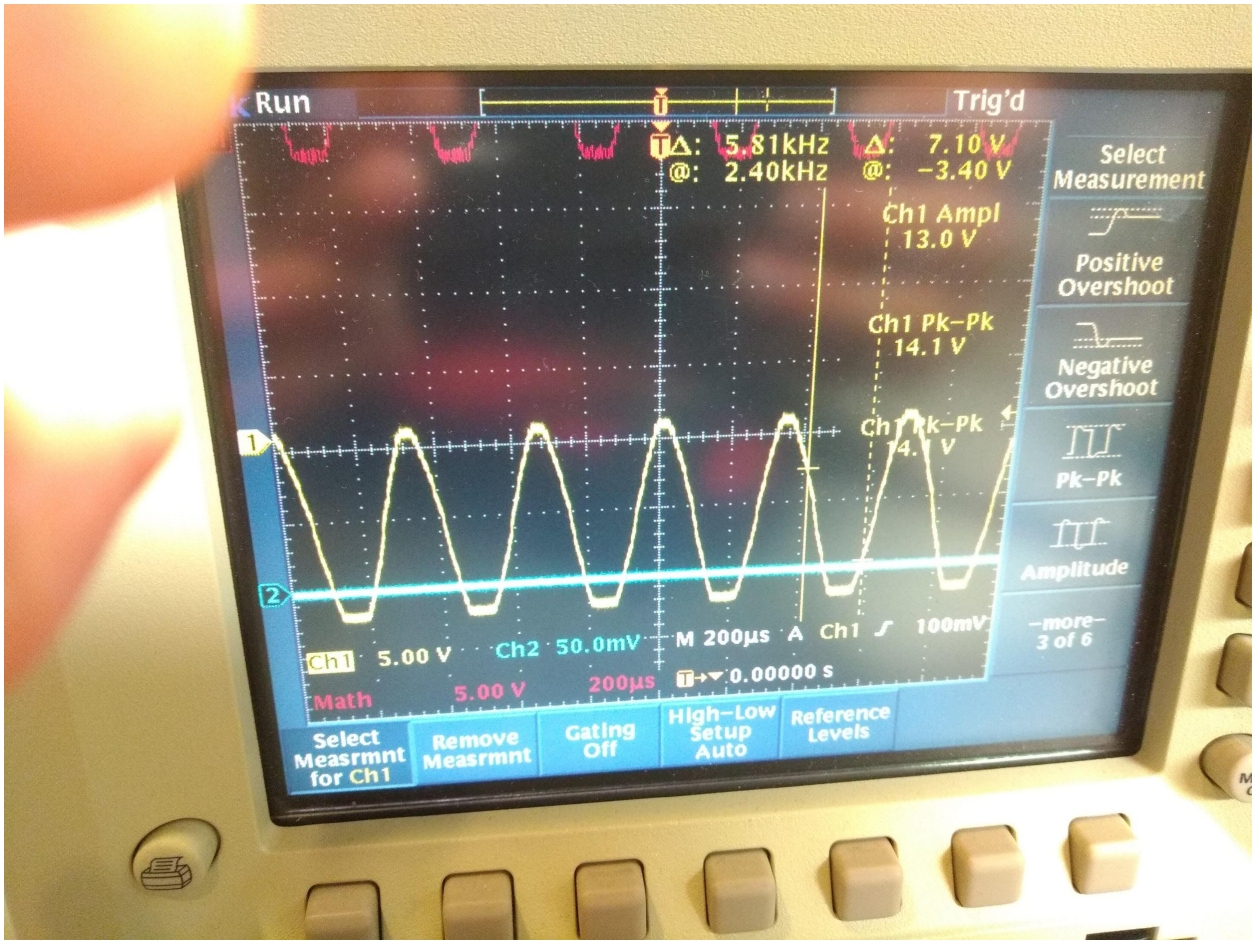
Efter att detta var löst visade sig signalen sakna botten efter offset/förstärkning (se figur 20), vilket berodde på en miss från vår sida: den digitala potentiometern (MCP4251) kunde ej hantera negativa spänningar enligt databladet [13], vilket därmed ledde till att botten på signalen kapades bort, då nämnd potentiometer satt i feedbackloopen hos den operationsförstärkare som har hand om förstärkning av signalen. En potentiell sidoeffekt av detta vara att potentiometern över huvud taget inte agerade på SPI-kommandon, och byttes därmed ut mot två analoga potentiometrar, vilket resulterade i en komplett sinuskurva, där både amplitud och offset kunde regleras (se figur 26, figur 27). I figur synes fungerande offsetjustering där offset justerats negativt till en sådan grad att operationsförstärkaren börjar bottna.



Figur 26 - Kapad botten på sinuskurva



Figur 27 - Fungerande offset och amplitudjustering.



Figur 28 - Offsetjustering för att uppnå bottning i operationsförstärkare.

Skärmen krävde närmare ett par dagar av felsökning då den vid initialisering vägrade visa några tecken, och koden fastnade i evigt väntade. Efter jämförelser av andra implementationer av drivare för skärmar som använder Hitachi HD44780-protokollet gav kommentarerna i källkoden för en Arduinodrivare [16] att vissa modeller som implementerar nämnt protokoll inte stödjer den så kallade “wait ready”-signalen [15], vilket mycket riktigt visade sig vara felet i detta fallet. Genom att helt deaktivera den delen av kodbasen i display.c som verifierade nämnd signal fungerade skärmen som väntat.

På grund av den något naiva implementationen av interrupthantering i hårdvaran; hög signal till skiftregister och OR-grindar skapade rotationsenkodarna stora problem. Då de regel hamnar i ett mellanläge där de fortsätter ge 5V-signal resulterade detta i att de ofta låste fortsatta interrupts (då interrupts är flanktriggade), och därmed gjorde kontrollerna på frontpanelen oresponsiva. För testning gick detta att kringgå genom att kontinuerligt mäta med multimeter och se att övriga rotationsenkoders inte gav signal när en specifik rotationsenkoder eller knapp skulle testas.

Att använda sig av interrupts för att hantera signalhantering som ursprungligen planerat visade sig i slutändan inte fungera med interrupthantering för användarinput (kodare och mikrobrytare

(knappar) då detta resulterade i att signalgenereringens avbrott avbröt de beräkningar som behövde göras för att beräkna ny frekvens, amplitud eller offset. Lösningen på detta var därmed slutligen att istället implementera signalgenerering i en main loop och enbart använda interrupts för inputhantering.

11. Resultat

När väl samtliga tester (se kapitel 10.2 Testning och åtgärder) var utförda sammanställdes resulatet för att få en bild av hur väl signalgeneratorn mötte de specifikationer som fastställdes under projektets planeringsfas (se kapitel 1. Inledning).

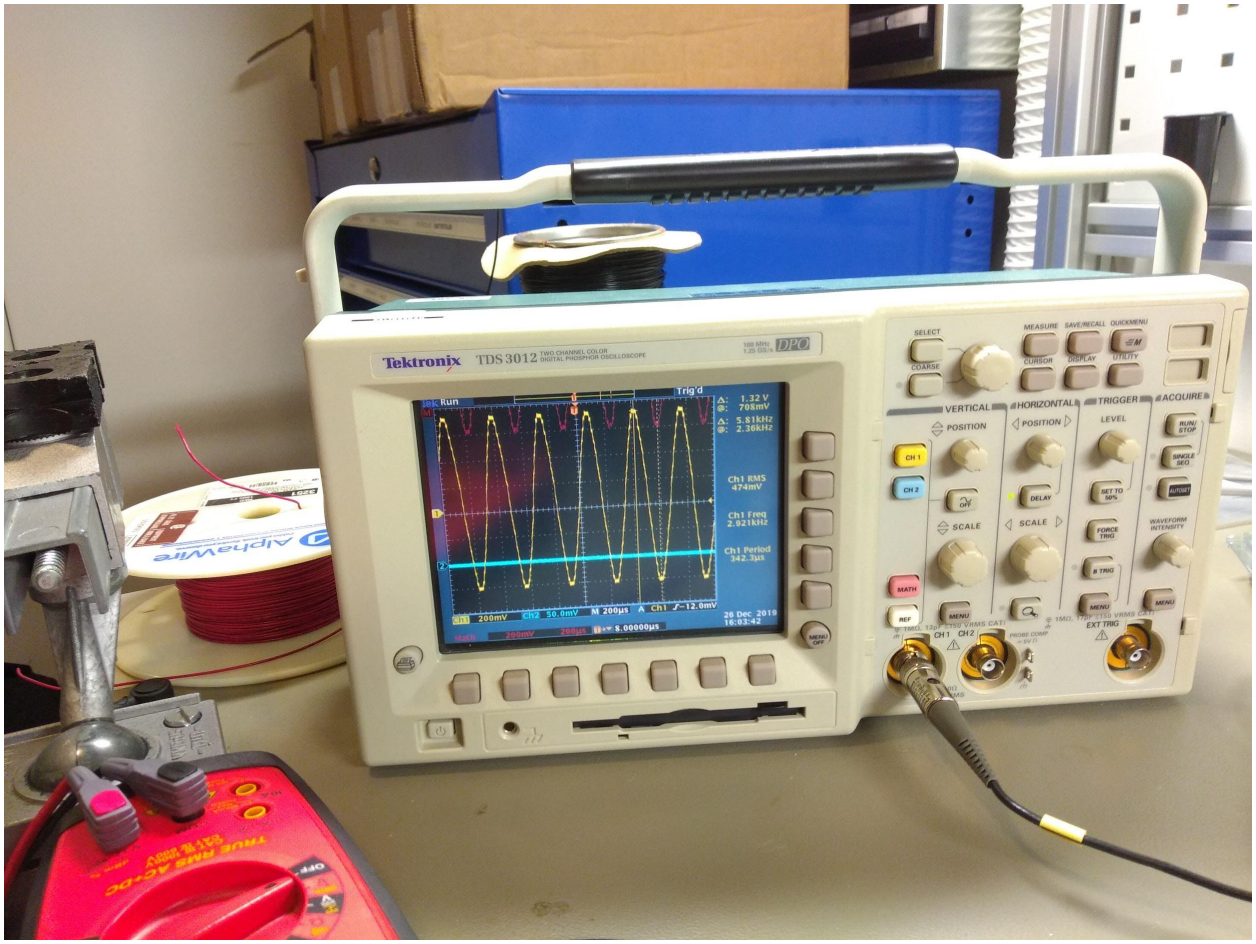
11.1 Frekvens och störningar

Vid testning uppnåddes vid slutgiltig konfiguration (tidigare nämnda mainloop för signalgenerering) som högst en frekvens på 2.9 kHz (figur 29), vilket är långt under den förväntade frekvensen på 250kHz. Vid tester där den mesta funktionaliteten och abstraktionen i koden (mao. färre function calls) deaktiverades kunde en frekvens på 22.65 kHz uppnås (se figur 30), vilket inte heller nådde i närheten av förväntad frekvens.

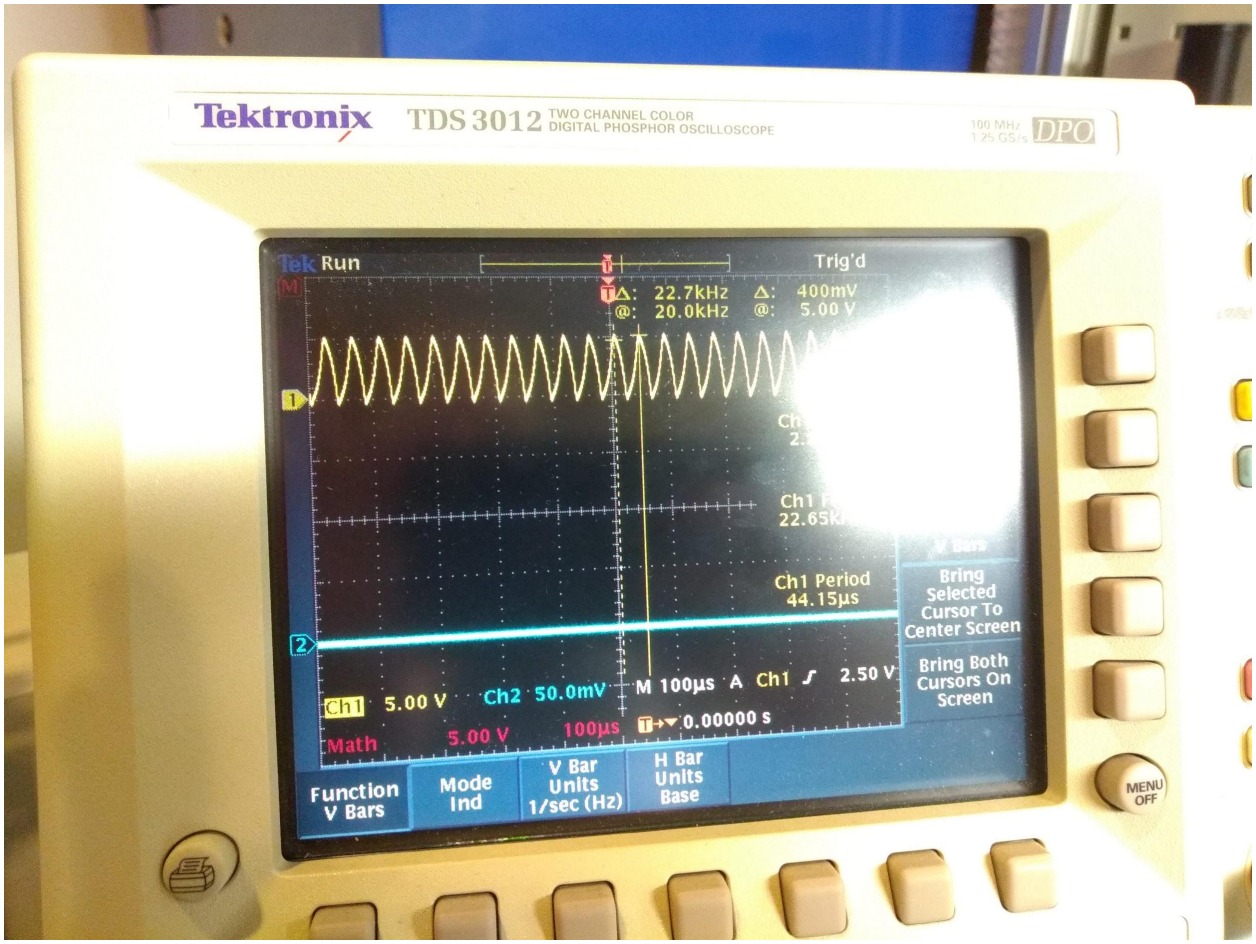
I och med att frekvensen inte kom i närheten av förväntade frekvensen kunde filtret heller aldrig filtrera kvantifieringen av signalerna (för att få renare sinuskurvor), vilket resulterade i att sinuskurvorna även efter filtret fortfarande hade "trappstegsform”. Dock, något som är värt att notera här är att signalen vid testet på 22.65 kHz når dess kvantifierade komponenter över filtrets brytfrekvens (300 kHz), ty på grund av att varje period består av 32 individuella värden uppstår här en frekvens på 724 800 Hz för periodens komponenter, medan den föregående nämnda signalen (2.9 kHz) enbart resulterar i en frekvens på 92 800 Hz.

I och med detta något nedlåtande resultat, och på grund av brist på tid utfördes aldrig några tester av störningar i signalen, då signalgeneratorns huvudfunktionlitet (att kunna generera tillräckligt höga frekvenser med filtrerade sinuskruvor) aldrig kunde uppnås.

De flesta funktioner på prototypen fungerade dock. Strömförsörjningen kunde leverera från +15 till -15 V. Amplitudjustering via potentiometrar fungerade när de byttes mot analoga varianter istället för de digitala som monterades från början, detta då de digitala inte klarade av negativ spänning. Offsethanteringen fungerade (se figur 28).



Figur 29 - Maximal frekvens med mainloop



Figur 30 - Maximal uppnådd frekvens vid deaktivering av funktionalitet.

11.2 Kostnad

Kostnaden för signalgenerator slutade på 44.84€, vilket med dagens penningvärde (2021-08-20) översätter till 460.83 SEK. Detta är något dyrare än beräknat, men vad som går att notera från kostnadskalkylen (se Bilaga 4 - Kostnadskalkyl) är att nästan hälften av kostnaden enbart upptas av hårdvara (kontakter, kablage etc.).

Värt att notera i beräkningen är också att LED:arna aldrig tas med i den totala kostnadsberäkningen då deras enda syfte var felsökning under prototypstadiet, och därmed inte är något som hade funnits med i den slutgiltiga produkten.

Andra produkter på marknaden med likartade funktioner kostar ungefär lika mycket men oftast mer. Till exempel finns [17] hos electrokit och kostar 579 SEK, men de flesta signalgeneratorer avsedda för privatpersoner kostar över 1500 SEK, till exempel denna [18].

12. Slutsats och diskussion

De flesta av kretsarna som designades till prototypen fungerade i huvudsak som tänkt; dock krävdes vissa modifieringar för att uppnå full funktionalitet. Om en andra prototyp skall designas behöver dessa modifieringar finnas med och även vissa delar helt bytas ut.

Den del som var mest experimentell för detta projekt var MCU:n PIC18F27Q10, och då huruvida den skulle klara av att leverera den frekvens som efterfrågades. Som förklarades i sektionerna 10 och 11 uppnådde den inte i närheten av den förväntade prestandan. Därför behöver MCU:n bytas ut mot en annan variant, till exempel PIC18F47Q43 då den har direkt minnesåtkomst (DMA). Detta medför att man inte behöver använda processorns interruptfunktion på samma sätt för att generera signalen och då bör man kunna få en mycket högre frekvens än vad som erhölls vid testerna av denna prototyp. Man kan då spara signalernas grundvärden i minnet och hämta dem direkt utan att lägga all belastning på processorn. I och med att PIC18F47Q43 är en mer avancerad MCU hade den kostat något mer: 2.23€, vilket är högre i jämförelse med PIC18F27Q10, vars pris är 1.52€.

Sedan är det en del andra komponenter som behöver bytas såsom potentiometrar mot analoga alternativt andra digitala som klarar negativ spänning. Det senare bör eftersträvas eftersom man då kan använda MCU:n för att styra amplitud och offset och därmed visa nivåerna på skärmen utan vidare modifieringar.

På grund av tidsbrist utformades aldrig något chassi, vilket är anledningen till att en sektion för denna aspekt av signalgeneratorn aldrig återfinns i rapporten (men är nämnd under kapitel “3.9 Utformning av chassi”).

Ett ytterligare avsteg som gjordes från initial planering var att kretsen byggdes direkt på PCB istället för att först konstruera en prototyp på breadboard.

Sett ur miljösynpunkt bör man vid serieproduktion använda sig av blyfritt lödtenn snarare än blyat lödtenn, som användes under prototypfasen. Anledningen till att blyat lödtenn användes var för att det förenklade konstruktionen av PCB:n då det är lättare att få till bra lödningar med blyat lödtenn.

En miljömässig fördel som dock uppkommer av att konstruktionen använder sig av hålmonterade komponenter är däremot att det blir lättare att reparera enheten, och därmed kan resultera i en reparation av enheten istället för att den byts ut vid bristande funktionsförmåga.

Referenser

[1] P. Horowitz, *The art of electronics*, Third edition. New York, NY: Cambridge University Press, 2015.

[2] ASPENCORE, ”Electronics-Tutorial”, *Sallen and Key Filter*, 2021. https://www.electronics-tutorials.ws/filter/sallen-key-filter.html (åtkomstdatum juli 30, 2021).

[3] ASPENCORE, ”Electronics-tutorials”, *Butterworth Filter Design*, 2021. https://www.electronics-tutorials.ws/filter/filter_8.html (åtkomstdatum maj 04, 2021).

[4] STMicroelectronics, ”L78 - Positive voltage regulator ICs”. DS0422 datasheet.

[5] Ohmite, ”F and R Series Heatsinks”.

[6] ASPENCORE, ”Electronics-Tutorial”, *Passive Low Pass Filter*, 2021. https://www.electronics-tutorials.ws/filter/filter_2.html (åtkomstdatum juli 23, 2021).

[7] Texas Instruments, ”CD54HC165, CD74HC165, CD54HCT165, CD74HCT165 High-Speed CMOS Logic 8-Bit Parallel-In/Serial-Out Shift Register”. SCHS156C.

[8] Texas Instruments, ”CD4071B, CD4072B, CD4075B Types CMOS OR Gates”. SCHS056D.

[9] Texas Instruments, ”MC3x063A 1.5-A Peak Boost/Buck/Inverting Switching Regulators”. SLLS636N.

[10] Texas Instruments, ”CD4066B CMOS Quad Bilateral Switch”. SCHS051H.

[11] Texas Instruments, ”TL07xx Low-Noise FET-Input Operational Amplifiers”. SLOS080P.

[12] Bourns, Inc., ”4600X Series - Thick Film Conformal SIPs”. C1753.

[13] Microchip Technology Inc., ”MCP413X/415X/423X/425X”. DS22060B, 2008.

[14] ORISE Technology Co., Ltd., ”Specification For LCD Module 2004A”. SPLC780D1, 2007.

[15] Hitachi, ”HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver)”. ADE-207-272(Z).

[16] *hd44780 Extensible hd44780 LCD library*. duinoWitchery, 2021. Åtkomstdatum: aug. 13, 2021. [Online]. Tillgänglig vid: https://github.com/duinoWitchery/hd44780/blob/27fd9d3809a92e744eef60b13dfcf4096c91cbe8/hd44780.cpp

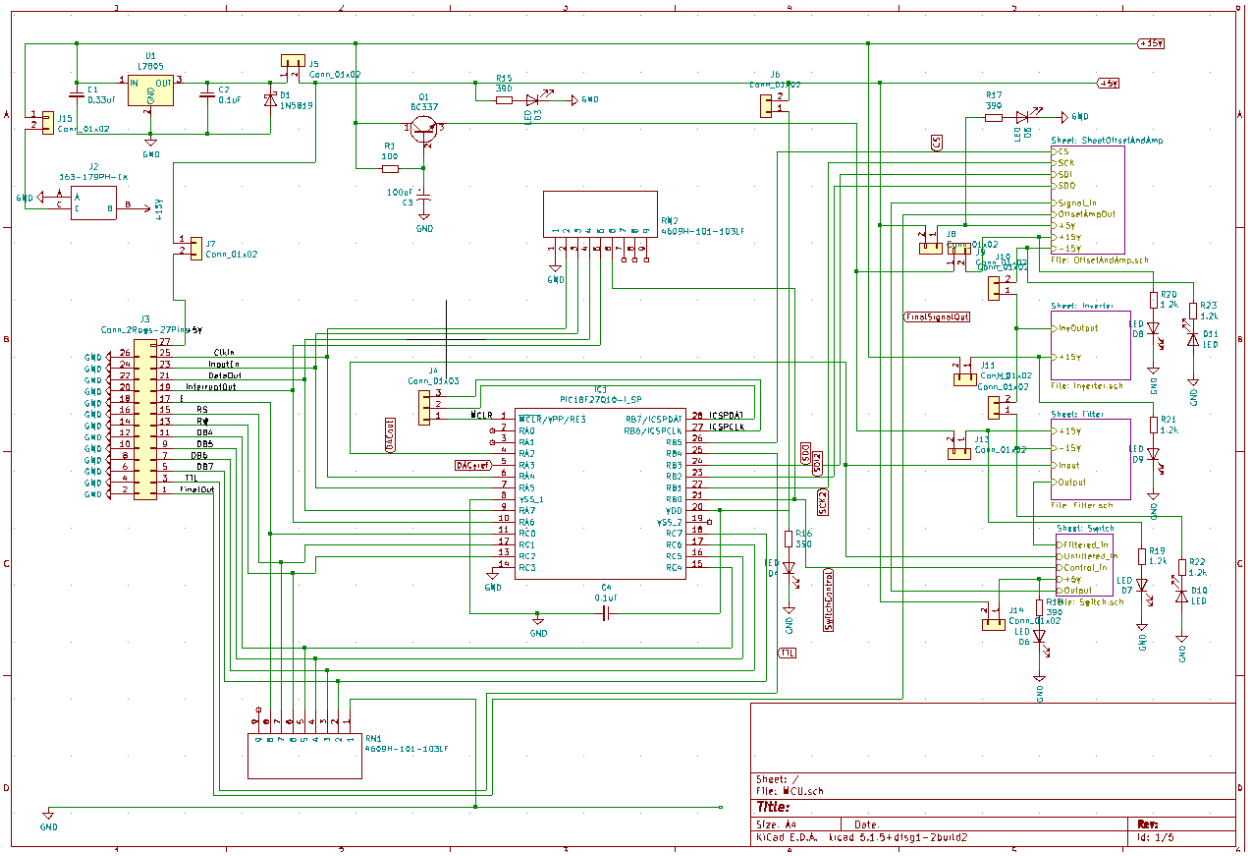
[17] ”Signalgenerator 200kHz FG085”, *Signalgenerator 200kHz FG085*, sep. 26, 2021. electrokit.com/produkt/signalgenerator-200khz-fg085/ (åtkomstdatum sep. 26, 2021).

[18] ”SFG-1003 - Funktionsgenerator, 1x 3MHz”, *SFG-1003 - Funktionsgenerator, 1x 3MHz*, sep. 26, 2021. https://www.elfa.se/sv/funktionsgenerator-1x-3mhz-gw-instek-sfg-1003/p/30088396?q=&pos=1&origPos=18&origPageSize=50&track=true (åtkomstdatum sep. 26, 2021).

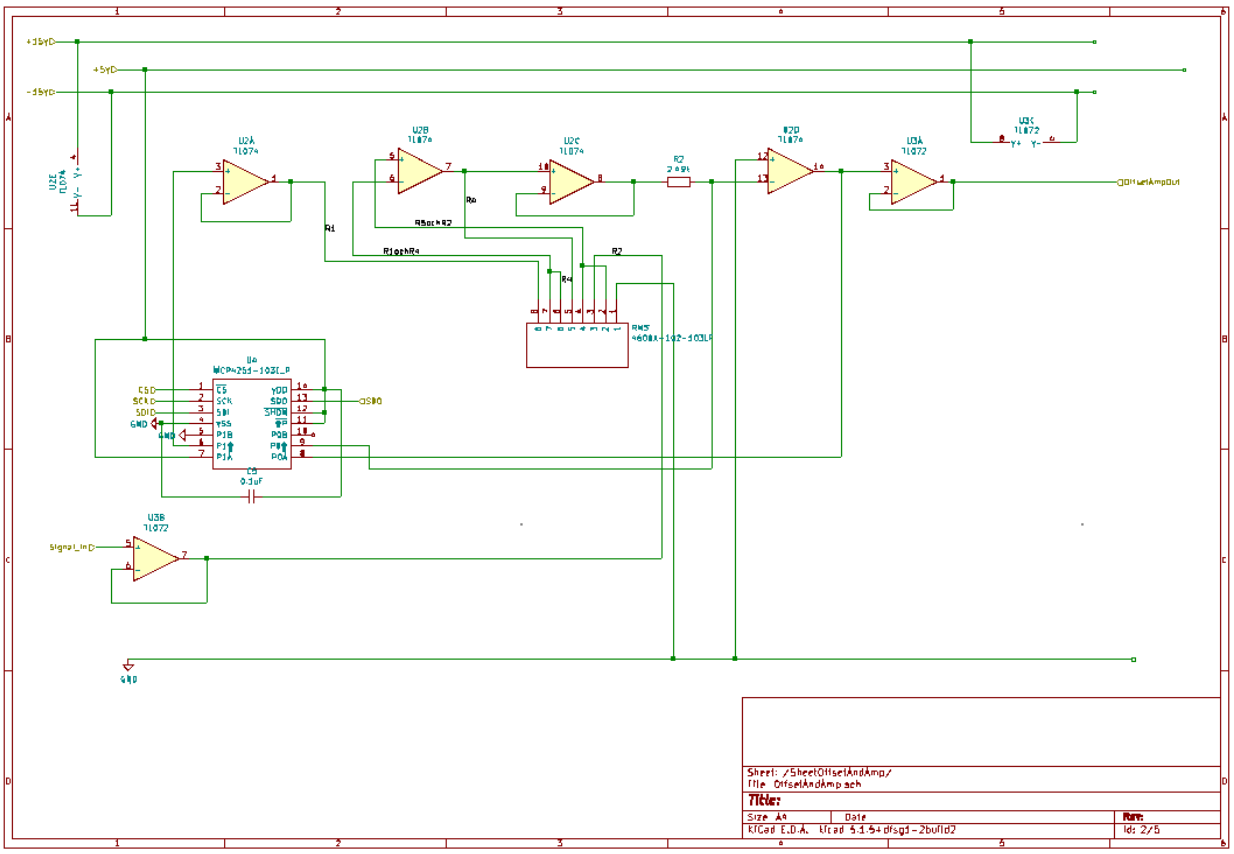
Bilagor

Bilaga 1 - Kretsschema moderkort

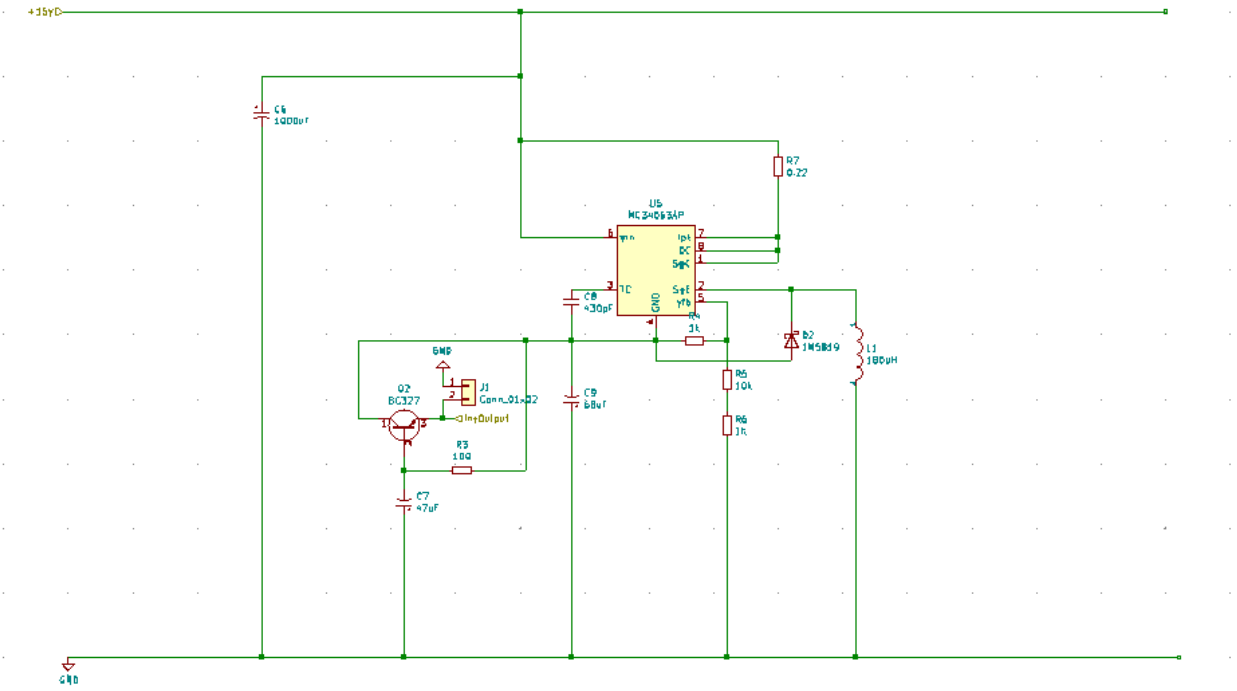
Översikt



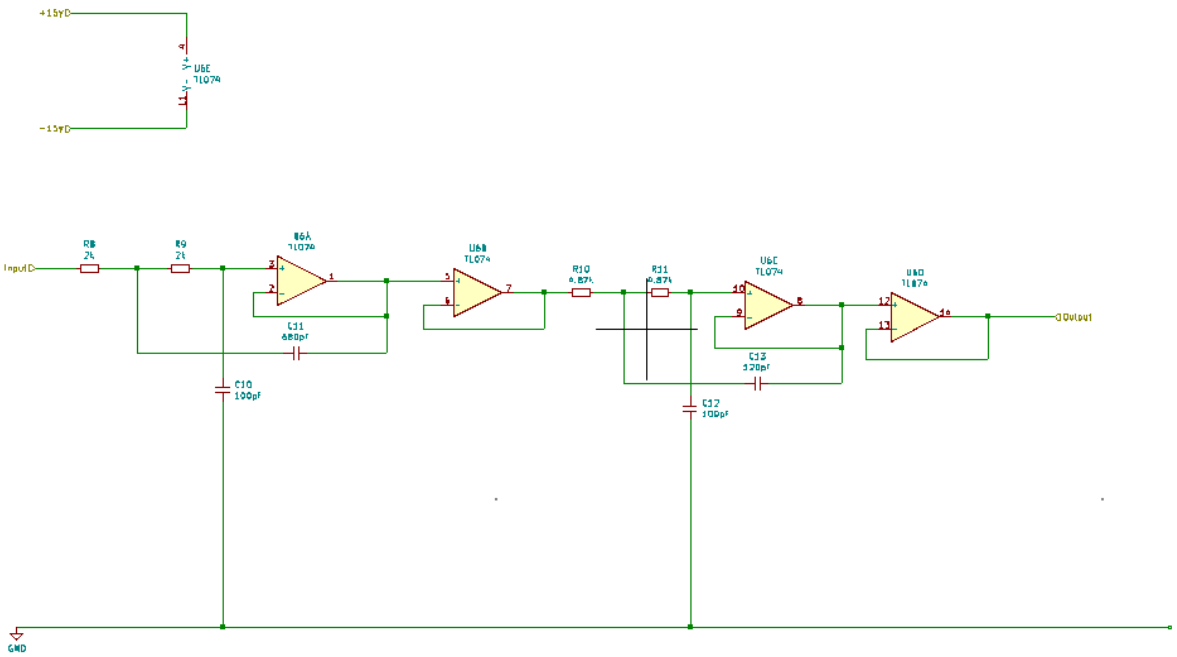
Offset och amplitudhantering



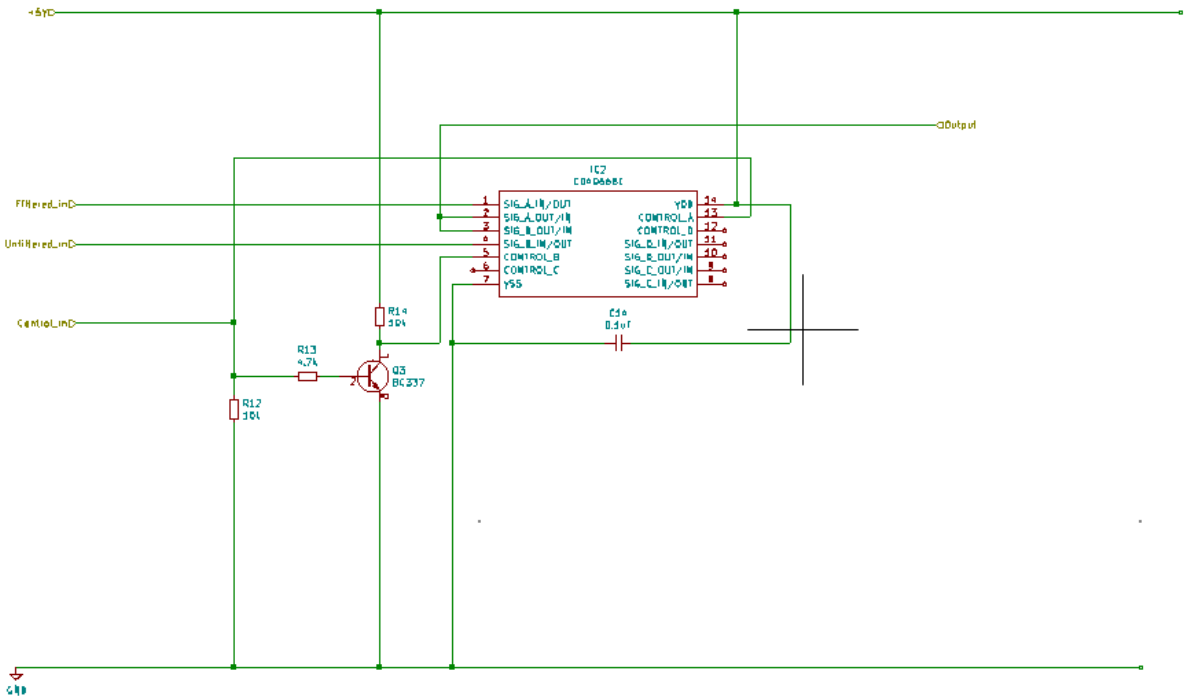
Negativ spänningsförsörjning



Filter

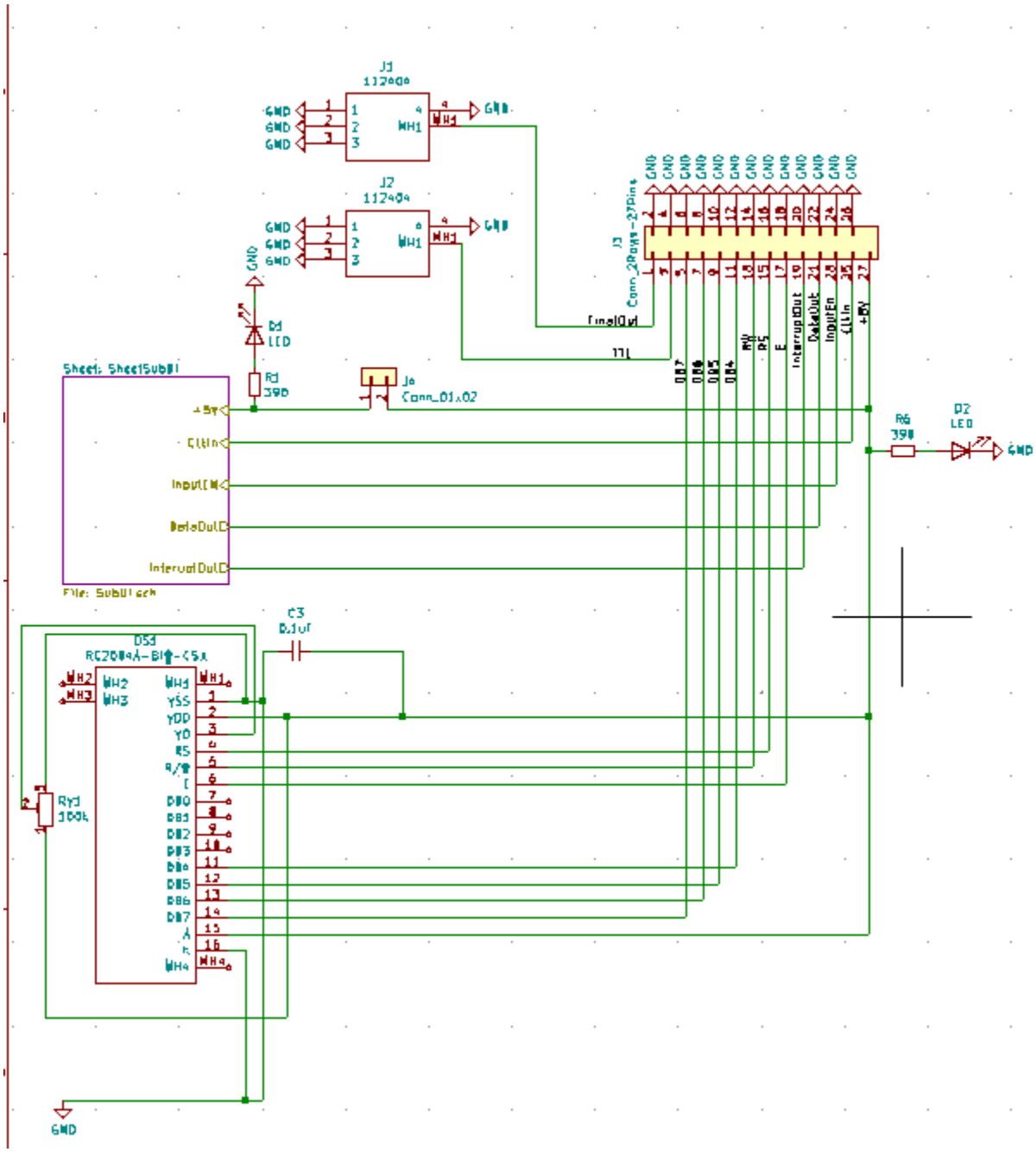


Signalväljare

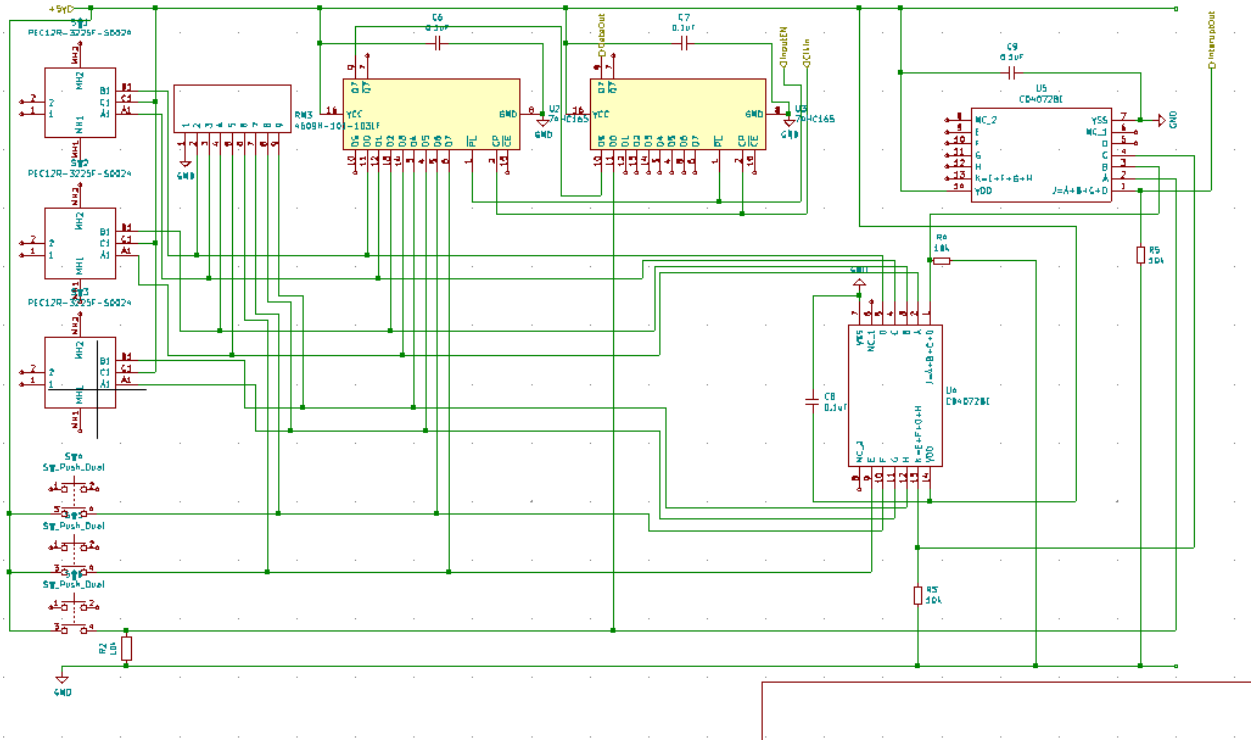


Bilaga 2 - Kretsschema för dotterkort

Översikt



Inputhantering



Bilaga 3 - Källkod

main.c

```
#include "mcc_generated_files/mcc.h"
#include "init.h"
#include "Libraries/delay_helper.h"
#include "Drivers/display.h"
#include "Libraries/display_helper.h"
#include "Drivers/input_panel.h"
#include "interrupt_defs.h"
#include "Drivers/mc4251.h"
#include "Libraries/wavetables.h"

//Driver and library singletons
CInputPanel input_panel;
CDelayHelper delay_helper;
CMCP4251Driver mcp4251_driver;
CWaveTableMananger wavetable_manager;
CDisplayHelper display_helper;

uint32_t current_freq = 3900;
float current_amplitude = 5.0;
float current_offset = 3.0;
s_TimerSettings timer_settings;
uint8_t run_counter = 0;
uint8_t partial_run_done = 0;


/*
                         Main application
 */
void main(void)
{
    // Initialize the device
    SYSTEM_Initialize();

    //Init pseudo classes
    init_delay_helper(&delay_helper);
    init_input_panel(&input_panel, &delay_helper);
    init_mcp4251_driver(&mcp4251_driver);
    init_wave_table_manager(&wavetable_manager);
    init_display_helper(&display_helper);
```

```
    input_panel.toggle_rw(input_panel_write);

    // If using interrupts in PIC18 High/Low Priority Mode you need to enable the Global High and
Low Interrupts
    // If using interrupts in PIC Mid-Range Compatibility Mode you need to enable the Global and
Peripheral Interrupts
    // Use the following macros to:

    //TMR0_SetInterruptHandler(WaveTableInterrupt);
    INT0_SetInterruptHandler(InputPanelInterrupt);

    //Initial settings
    //mcp4251_driver.set_amplitude(mcp4251_driver.calculate_amplitude(current_amplitude));
    //mcp4251_driver.set_offset(mcp4251_driver.calculate_offset(current_offset,
current_amplitude));
    timer_settings = wavetable_manager.calculate_timer(80000);
    display_helper.update_amplitude(&display_helper, current_amplitude);
    display_helper.update_offset(&display_helper, current_offset);
    display_helper.update_frequency(&display_helper, current_freq);
    display_helper.update_signal_type(&display_helper, wave_sinus);
    SelectFilter_SetHigh();

    // Enable the Global Interrupts
    INTERRUPT_GlobalInterruptEnable();

    // Disable the Global Interrupts
    //INTERRUPT_GlobalInterruptDisable();

    // Enable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptEnable();

    // Disable the Peripheral Interrupts
    //INTERRUPT_PeripheralInterruptDisable();

    while (1)
    {
        if (run_counter < timer_settings.runs) {
            run_counter++;
            TMR4_Period8BitSet(255);
            TMR4_StartTimer();
            while (TMR4_Counter8BitGet() != 0) {}
        } else if (!partial_run_done) {
            TMR4_Period8BitSet(timer_settings.partial_runs);
            partial_run_done = 1;
```

```
            TMR4_StartTimer();
            while (TMR4_Counter8BitGet() != 0) {}
        } else {
            DAC1_SetOutput(wavetable_manager.next_position(&wavetable_manager));
            partial_run_done = 0;
            run_counter = 0;
        }
    }

}

/**
 End of File
*/
```

types.h

```
/*
 * File:   types.h
 *
 * Created on May 12, 2021, 11:31 AM
 */

#ifndef TYPES_H
#define           TYPES_H

typedef unsigned char uint8_t;
typedef unsigned short uint16_t;
typedef unsigned long uint32_t;


#endif  /* TYPES_H */
```

Interrupt_defs.h

```
/*
 * File:   interrupt_defs.h
 *
 * Created on June 3, 2021, 2:53 PM
 */

#include "Drivers/input_panel.h"
#include "Libraries/delay_helper.h"
#include "Drivers/mc4251.h"
#include "Libraries/wavetables.h"
#include "mcc_generated_files/mcc.h"
#include "Libraries/display_helper.h"

#ifndef INTERRUPT_DEFS_H
#define          INTERRUPT_DEFS_H

typedef enum {
    single = 1,
    deka = 10,
    hekto = 100,
    kilo = 1000
} e_divider_state;

//External variables defined in main.c
extern CInputPanel input_panel;
extern CDelayHelper delay_helper;
extern CMCP4251Driver mcp4251_driver;
extern CWaveTableMananger wavetable_manager;
extern s_TimerSettings timer_settings;
extern CDisplayHelper display_helper;

extern uint32_t current_freq;
extern float current_amplitude;
extern float current_offset;

void InputPanelInterrupt();
void WaveTableInterrupt();

#endif  /* INTERRUPT_DEFS_H */
```

Interrupt_defs.c

```
#include "interrupt_defs.h"

e_divider_state current_divider_state = single;
e_waveform_type current_waveform = wave_sinus;
e_display_shown_mode current_mode = display_amplitude;

//Handler for input panel interrupts
void InputPanelInterrupt() {

   input_panel.toggle_rw(input_panel_read);
   e_input_signal input_signal = input_panel.identify_signal(&input_panel);

   //Waveform switch
   if (input_signal == signal_wave_incremented || input_signal == signal_wave_decremented) {
      if (signal_wave_incremented) {
         if (current_waveform >= 3)
            current_waveform = 0;
         else
            current_waveform++;
      } else {
         if (current_waveform <= 0)
            current_waveform = 3;
         else
            current_waveform--;
      }

      //Select the correct signal in the switch
      if (current_waveform == wave_sinus)
         SelectFilter_SetHigh();
      else
         SelectFilter_SetLow();

      //Update the wavetable manager accordingly
      wavetable_manager.select_waveform(&wavetable_manager, current_waveform);
      display_helper.update_signal_type(&display_helper, current_waveform);

   //Frequency adjustment
   } else if (input_signal == signal_freq_incremented || input_signal ==
signal_freq_decremented) {
      long new_freq;
      if (input_signal == signal_freq_incremented)
         new_freq = current_freq+1*current_divider_state;
```

```
        else
            new_freq = current_freq-1*current_divider_state;

        current_freq = wavetable_manager.verify_frequency(new_freq);
        //Due to MCC generating it's own reset routine, the external var will need to be used.
        timer_settings = wavetable_manager.calculate_timer(current_freq);
        display_helper.update_frequency(&display_helper, current_freq);

    //Offset and amplitude adjustment
    } else if (input_signal == signal_amp_offset_incremented || input_signal ==
signal_amp_offset_decremented) {
        //Amplitude mode currently selected
        if (current_mode == display_amplitude) {
            //Verify that the amplitude will work with the current offset
            if (input_signal == signal_amp_offset_decremented)
                current_amplitude = mcp4251_driver.verify_amplitude(current_offset,
current_amplitude - 0.1);
            else
                current_amplitude = mcp4251_driver.verify_amplitude(current_offset,
current_amplitude + 0.1);

            //Uint calculated versions of the selected new amplitude with the current offset
            uint16_t new_amplitude = mcp4251_driver.calculate_amplitude(current_amplitude);
            uint16_t new_offset = mcp4251_driver.calculate_offset(current_offset,
current_amplitude);
            //mcp4251_driver.set_offset(new_offset);
            //mcp4251_driver.set_amplitude(new_amplitude);

            display_helper.update_amplitude(&display_helper, current_amplitude);

        } else {
            if (input_signal == signal_amp_offset_decremented)
                current_offset = mcp4251_driver.verify_offset(current_offset - 0.1, current_amplitude);
            else
                current_offset = mcp4251_driver.verify_offset(current_offset + 0.1,
current_amplitude);

            uint16_t new_offset = mcp4251_driver.calculate_offset(current_offset,
current_amplitude);
            //mcp4251_driver.set_offset(new_offset);

            display_helper.update_offset(&display_helper, current_offset);
        }
```

```
    //Switch between dc and normal functions
    } else if (input_signal == signal_dc_ac_switch) {
        if (current_waveform != wave_dc) {
            current_waveform = wave_dc;
            SelectFilter_SetLow();
        } else {
            current_waveform = wave_sinus;
            SelectFilter_SetHigh();
        }

        display_helper.update_signal_type(&display_helper, current_waveform);

    //Rotate between the possible dividers (hertz, dekahertz, 100hertz and khz)
    } else if (input_signal == signal_div_switch) {
        switch (current_divider_state) {
            case single:
                current_divider_state = deka;
                break;
            case deka:
                current_divider_state = hekto;
                break;
            case hekto:
                current_divider_state = kilo;
                break;
            case kilo:
                current_divider_state = single;
                break;
        }

    //Switch between amp and offset
    } else if (input_signal == signal_mode_switch) {
        if (current_mode == display_amplitude)
            current_mode = display_offset;
        else
            current_mode = display_amplitude;

        display_helper.update_shown_mode(&display_helper, current_mode);
    }

    input_panel.toggle_rw(input_panel_write);

}
```

```
/*
 * No longer used due to loop being used instead of interrupts
void WaveTableInterrupt() {
    DAC1_SetOutput(wavetable_manager.next_position(&wavetable_manager));
}
*/
```

Drivers/display.h

```
/*
 * File:   display.h
 *
 * Created on April 15, 2021, 11:17 AM
 */

#ifndef DISPLAY_H
#define          DISPLAY_H

#include "../init.h"
#include "../Libraries/delay_helper.h"

#define DISPLAY_DRIVER_DEBUG

typedef enum {
    display_read,
    display_write
} e_display_rw_mode;

typedef enum {
    display_command,
    display_data
} e_display_op_mode;

typedef enum {
    display_row1 = 0x00,
    display_row2 = 0x40,
    display_row3 = 0x14,
    display_row4 = 0x54
} e_display_row_selection;

/*
 * Pins used in the schematic:
 * RC0: E
 * RC1: RS
 * RC2: RW
 * RC4-RC7: Bits
 *
 */
typedef struct DisplayDriver {
    CDelayHelper delay_helper;
    void (*set_rw_mode) (e_display_rw_mode mode); //Whether to read or write to the display
```

```
    void (*set_op_mode) (e_display_op_mode mode); //Command or data mode
    void (*write) (struct DisplayDriver *self, uint8_t byte);
    void (*write_data) (struct DisplayDriver *self, uint8_t data);
    void (*write_command) (struct DisplayDriver *self, uint8_t command);
    void (*read_status) (struct DisplayDriver *self);
    void (*wait_ready) (struct DisplayDriver *self);
    void (*clear_display) (struct DisplayDriver *self);
    void (*write_character) (struct DisplayDriver *self, uint8_t character);
    void (*goto_position) (struct DisplayDriver *self, uint8_t column, e_display_row_selection
row);
} CDisplayDriver;

void init_display_driver(CDisplayDriver *display_driver);
void display_driver_set_rw_mode(e_display_rw_mode mode);
void display_driver_set_op_mode(e_display_op_mode mode);
void display_driver_write(CDisplayDriver *self, uint8_t byte);
void display_driver_write_data(CDisplayDriver *self, uint8_t data);
void display_driver_write_command(CDisplayDriver *self, uint8_t command);
uint8_t display_driver_read_status(CDisplayDriver *self);
void display_driver_wait_ready(CDisplayDriver *self);
void display_driver_clear_display(CDisplayDriver *self);
void display_driver_write_character(CDisplayDriver *self, uint8_t character);
void display_driver_goto_position(CDisplayDriver *self, uint8_t column, e_display_row_selection
row);


#endif  /* DISPLAY_H */
```

Drivers/display.c

```
#include "display.h"
#include "../mcc_generated_files/mcc.h"

/*
 * Display driver for driving the tc2004 display, look at the implementation
 * made back in "machine oriented programming" for reference, should be about
 * the same API
 */

void init_display_driver(CDisplayDriver *display_driver)
{
    //make sure the RC4-RC7 pins are configured to be digital
    ANSELC &= ~0xF0;
    init_delay_helper(&display_driver->delay_helper);

    display_driver->set_rw_mode = display_driver_set_rw_mode;
    display_driver->set_op_mode = display_driver_set_op_mode;
    display_driver->write = display_driver_write;
    display_driver->write_data = display_driver_write_data;
    display_driver->write_command = display_driver_write_command;
    display_driver->wait_ready = display_driver_wait_ready;
    display_driver->clear_display = display_driver_clear_display;
    display_driver->write_character = display_driver_write_character;
    display_driver->goto_position = display_driver_goto_position;

    /*
     * Launch the display initialization procedure, based on the
     * HD44780U datasheet pg. 46
     */
    display_driver->set_op_mode(display_command);
    display_driver->delay_helper.delay_ms(100);
    display_driver_set_rw_mode(display_write);
    LATC &= ~0xF0;
    LATC |= 0x30;
    DisplayE_SetHigh();
    display_driver->delay_helper.delay_ms(1);
    DisplayE_SetLow();
    display_driver->delay_helper.delay_ms(5);
    DisplayE_SetHigh();
    display_driver->delay_helper.delay_ms(1);
    DisplayE_SetLow();
    display_driver->delay_helper.delay_ms(1);
```

```
    DisplayE_SetHigh();
    display_driver->delay_helper.delay_ms(1);
    DisplayE_SetLow();
    display_driver->delay_helper.delay_ms(1);
    LATC &= ~0xF0;
    LATC |= 0x20;
    DisplayE_SetHigh();
    display_driver->delay_helper.delay_us(1);
    DisplayE_SetLow();
    display_driver->delay_helper.delay_us(39);

    //Set actual parameters
    //2 lines, 5x8 4 bit mode
    display_driver_write_command(display_driver, 0b00101000);
    display_driver->delay_helper.delay_ms(1);
    //Display on
    display_driver->write_command(display_driver, 0b00001100);
    display_driver->delay_helper.delay_ms(1);
    display_driver_clear_display(display_driver);
    //Set entry mode to incrementing
    display_driver_write_command(display_driver, 0x06);
    display_driver->delay_helper.delay_us(40);
}

void display_driver_set_rw_mode(e_display_rw_mode mode)
{
    if (mode == display_write) {
        //Set pins RC4-RC7 to output
        TRISC &= ~0xF0;
        DisplayRW_SetLow();
    } else {
        TRISC |= 0xF0;
        DisplayRW_SetHigh();
    }
}

void display_driver_set_op_mode(e_display_op_mode mode)
{
    if (mode == display_data)
        DisplayRS_SetHigh();
    else
        DisplayRS_SetLow();

}
```

```
void display_driver_write(CDisplayDriver *self, uint8_t byte)
{
   //Write mode + delay to wait for the display controller
   display_driver_set_rw_mode(display_write);
   self->delay_helper.delay_fourth_us(1);

   //Write the upper half of the byte
   LATC &= ~0xF0;
   LATC |= (byte & ~0x0F);
   DisplayE_SetHigh();
   self->delay_helper.delay_fourth_us(2);
   DisplayE_SetLow();
   self->delay_helper.delay_us(1);

   //Write the lower half of the byte
   LATC &= ~0xF0;
   LATC |= (byte << 4);
   DisplayE_SetHigh();
   self->delay_helper.delay_fourth_us(2);

   //Finally, low flank (cycle done)
   DisplayE_SetLow();
   self->delay_helper.delay_us(1);
}

void display_driver_write_data(CDisplayDriver *self, uint8_t data)
{
   self->set_op_mode(display_data);
   display_driver_write(self, data);
}

void display_driver_write_command(CDisplayDriver *self, uint8_t command)
{
   self->set_op_mode(display_command);
   display_driver_write(self, command);
}

uint8_t display_driver_read_status(CDisplayDriver *self)
{
   self->delay_helper.delay_us(1);
   self->set_rw_mode(display_read);
   self->delay_helper.delay_us(1);
   self->set_op_mode(display_command);
```

```
    self->delay_helper.delay_fourth_us(1);

    //Read the lower part
    DisplayE_SetHigh();
    self->delay_helper.delay_us(1);
    uint8_t return_value = (PORTC >> 4);
    self->delay_helper.delay_us(1);
    DisplayE_SetLow();
    self->delay_helper.delay_us(39);

    //And the upper part
    DisplayE_SetHigh();
    self->delay_helper.delay_us(1);
    return_value |= (PORTC & ~0x0F);
    self->delay_helper.delay_us(1);
    DisplayE_SetLow();
    self->delay_helper.delay_us(39);

    return return_value;
}

void display_driver_wait_ready(CDisplayDriver *self)
{
#ifndef DISPLAY_DRIVER_DEBUG
    while ((display_driver_read_status(self) & 0x80) == 0x80) {}
#endif
    self->delay_helper.delay_us(8);
}

void display_driver_clear_display(CDisplayDriver *self)
{
    //display_driver_wait_ready(self);
    display_driver_write_command(self, 0x01);
    self->delay_helper.delay_ms(2);
}

void display_driver_write_character(CDisplayDriver *self, uint8_t character)
{
    display_driver_wait_ready(self);
    display_driver_write_data(self, character);
    self->delay_helper.delay_us(43);
}

//Rows should be specified as 0-3, same goes for column. Start at 0
```

```
void display_driver_goto_position(CDisplayDriver* self, uint8_t column, e_display_row_selection
row)
{
    display_driver_wait_ready(self);
    uint8_t address = column + row;
    display_driver_write_command(self, (0x80 | address));
    self->delay_helper.delay_us(40);
}
```

Drivers/input_panel.h

```
/*
 * File:   input_panel.h
 *
 * Created on April 15, 2021, 11:19 AM
 */

#ifndef INPUT_PANEL_H
#define          INPUT_PANEL_H

#include "../init.h"
#include "../mcc_generated_files/mcc.h"
#include "../Libraries/delay_helper.h"

#define WAVE_INCREMENT 0x8000
#define WAVE_DECREMENT 0x4000
#define FREQ_INCREMENT 0x2000
#define FREQ_DECREMENT 0x1000
#define AMP_OFFSET_INCREMENT 0x0800
#define AMP_OFFSET_DECREMENT 0x0400
#define DC_AC_SWITCH 0x0200
#define DIV_SWITCH 0x0100
#define MODE_SWITCH 0x0080

typedef enum {
    signal_wave_incremented,
    signal_wave_decremented,
    signal_freq_incremented,
    signal_freq_decremented,
    signal_amp_offset_incremented,
    signal_amp_offset_decremented,
    signal_dc_ac_switch,
    signal_div_switch,
    signal_mode_switch,
    signal_not_found
} e_input_signal;

typedef enum {
    input_panel_read,
    input_panel_write
} e_input_panel_mode;

typedef struct InputPanel {
```

```
    CDelayHelper* delay_helper;
    e_input_signal (*identify_signal) (struct InputPanel* self);
    void (*toggle_rw) (e_input_panel_mode mode);
} CInputPanel;

void init_input_panel(CInputPanel *input_panel, CDelayHelper* delay_helper);
e_input_signal input_panel_identify_signal(CInputPanel *self);
void input_panel_toggle_rw(e_input_panel_mode mode);


#endif  /* INPUT_PANEL_H */
```

Drivers/input_panel.c

```
#include "input_panel.h"

void init_input_panel(CInputPanel* input_panel, CDelayHelper* delay_helper) {
   input_panel->delay_helper = delay_helper;
   input_panel->identify_signal = input_panel_identify_signal;
   input_panel->toggle_rw = input_panel_toggle_rw;
}

e_input_signal input_panel_identify_signal(CInputPanel* self)
{
   uint16_t input_short = 0x0000;
   uint8_t found = 0;
   uint8_t bit_count = 0;
   e_input_signal return_value = signal_not_found;

   while (bit_count < 16) {
      //Really ugly workaround caused by not taking into account how the shift register shifts out
bits
      if (bit_count > 6) {
         input_short |= DataFromShiftRegister_GetValue() << bit_count;

         switch (input_short) {
            case WAVE_INCREMENT:
               found = 1;
               return_value = signal_wave_incremented;
               break;
            case WAVE_DECREMENT:
               found = 1;
               return_value = signal_wave_decremented;
               break;
            case FREQ_INCREMENT:
               found = 1;
               return_value = signal_freq_incremented;
               break;
            case FREQ_DECREMENT:
               found = 1;
               return_value = signal_freq_decremented;
               break;
            case AMP_OFFSET_INCREMENT:
               found = 1;
               return_value = signal_amp_offset_incremented;
               break;
```

```
            case AMP_OFFSET_DECREMENT:
               found = 1;
               return_value = signal_amp_offset_decremented;
               break;
            case DC_AC_SWITCH:
               found = 1;
               return_value = signal_dc_ac_switch;
               break;
            case DIV_SWITCH:
               found = 1;
               return_value = signal_div_switch;
               break;
            case MODE_SWITCH:
               found = 1;
               return_value = signal_mode_switch;
               break;
         }
      }

      ClkToShiftRegister_SetHigh();
      self->delay_helper->delay_us(1);
      ClkToShiftRegister_SetLow();
      bit_count++;
   }

   return return_value;
}

void input_panel_toggle_rw(e_input_panel_mode mode)
{
   if (mode == input_panel_read) {
      ShiftRegisterInputEN_SetHigh();
   } else {
      ShiftRegisterInputEN_SetLow();
   }
}
```

Drivers/mc4251.h

```
/*
 * File:
 * Author:
 * Comments:
 * Revision history:
 */

#include "../mcc_generated_files/mcc.h"
#include <assert.h>
#include <math.h>


// This is a guard condition so that contents of this file are not included
// more than once.
#ifndef MCP4251_H
#define          MCP4251_H

#define MCP4251_CONFIG_BITS 0xFF
#define MCP4251_WIPER_0_ADR 0x00
#define MCP4251_WIPER_1_ADR 0x01
#define MCP4251_CONFIG_ADR 0x04
#define MCP4251_READ_CMD 0x03
#define MCP4251_WRITE_CMD 0x00

typedef enum {
   wiper1 = MCP4251_WIPER_0_ADR,
   wiper2 = MCP4251_WIPER_1_ADR
} e_mcp4251_wiper_num;

typedef struct MCP4251Driver {
   void (*write_short) (e_mcp4251_wiper_num wiper, uint16_t data_value);
   void (*set_config_value) (uint8_t config_value);
   void (*set_offset) (uint16_t u_offset);
   void (*set_amplitude) (uint16_t u_amplitude);
   float (*verify_offset) (float offset, float amplitude);
   float (*verify_amplitude) (float offset, float amplitude);
   uint16_t (*calculate_offset) (float offset, float amplitude);
   uint16_t (*calculate_amplitude) (float amplitude);
} CMCP4251Driver;

void init_mcp4251_driver(CMCP4251Driver* mcp4251_driver);
void mcp4251_driver_write_short(e_mcp4251_wiper_num wiper, uint16_t data_value);
```

```
void mcp4251_driver_set_config_value(uint8_t config_value);
void mcp4251_driver_set_offset(uint16_t u_offset);
void mcp4251_driver_set_amplitude(uint16_t amplitude);

//Helper functions
float mcp4251_driver_verify_offset(float offset, float amplitude);
float mcp4251_driver_verify_amplitude(float offset, float amplitude);
uint16_t mcp4251_driver_calculate_offset_uint(float offset, float amplitude);
uint16_t mcp4251_driver_calculate_amplitude_uint(float amplitude);

#endif
```

Drivers/mc4251.c

```
/*
 * File:   mc4251.c
 *
 * Created on April 15, 2021, 11:07 AM
 */


#include "../init.h"
#include "mc4251.h"

/*
 * Requirements needed for this aspect: SPI
 */

/**
 * MCP4251 class constructor
 * @param mcp4251_driver
 */
void init_mcp4251_driver(CMCP4251Driver* mcp4251_driver)
{
    mcp4251_driver->write_short = mcp4251_driver_write_short;
    mcp4251_driver->set_config_value = mcp4251_driver_set_config_value;
    mcp4251_driver->set_offset = mcp4251_driver_set_offset;
    mcp4251_driver->set_amplitude = mcp4251_driver_set_amplitude;
    mcp4251_driver->verify_offset = mcp4251_driver_verify_offset;
    mcp4251_driver->verify_amplitude = mcp4251_driver_verify_amplitude;
    mcp4251_driver->calculate_amplitude = mcp4251_driver_calculate_amplitude_uint;
    mcp4251_driver->calculate_offset = mcp4251_driver_calculate_offset_uint;

    //mcp4251_driver_set_config_value(MCP4251_CONFIG_BITS);
}

/**
 * Set a configuration value of the MCP4251 digipot. Really only used for init by the constructor
 * @param config_value
 */
void mcp4251_driver_set_config_value(uint8_t config_value)
{
    uint16_t result = 0x0000;
    SPI2_Open(SPI2_DEFAULT);
    MC4251_CS_SetHigh();
    //Set it to configuration
```

```
    result |= (uint16_t)(SPI2_ExchangeByte(0x41) << 8);
    //Enable all terminals
    result |= SPI2_ExchangeByte(config_value);
    assert(result == 0xFFFF);
    MC4251_CS_SetLow();
    SPI2_Close();
}

/**
 * Write a short to the digipot. Used for setting the value of the potentiometer.
 * @param wiper
 * @param data_value
 */
void mcp4251_driver_write_short(e_mcp4251_wiper_num wiper, uint16_t data_value)
{
    uint16_t formatted_command = 0x0000 | (wiper << 12) | data_value;
    uint16_t result = 0x0000;
    SPI2_Open(SPI2_DEFAULT);
    MC4251_CS_SetHigh();
    //Write the upper byte
    result |= (uint16_t)(SPI2_ExchangeByte((uint8_t)(formatted_command >> 8)) << 8);
    assert(result == 0xFF00);
    //Write the lower byte
    result |= SPI2_ExchangeByte((uint8_t)(formatted_command & ~0xFF00));
    assert(result == 0xFFFF);
    MC4251_CS_SetLow();
    SPI2_Close();
}

/**
 * Convenience method for setting the offset potentiometer
 * @param offset
 */
void mcp4251_driver_set_offset(uint16_t offset)
{
    mcp4251_driver_write_short(wiper1, offset);
}

/**
 * Convenience metho for setting the amplitude potentiometer
 * @param amplitude
 */
void mcp4251_driver_set_amplitude(uint16_t amplitude)
{
```

```
    mcp4251_driver_write_short(wiper2, amplitude);
}

/**
 * Make sure that the offset is within proper ranges
 * @param offset
 * @param amplitude
 * @return
 */
float mcp4251_driver_verify_offset(float offset, float amplitude)
{
    //Ugly workaround to handle that abs is int based. Kind of fixed point representation emulated
    int i_offset = (int)(offset*10);
    int i_amplitude = (int)(amplitude*10);

    //Offset cannot be "larger" than amplitude
    if (abs(i_offset) > i_amplitude) {
        if (offset < 0)
            return amplitude*(-1);
        else
            return amplitude;
    //max offset value is +/-5V
    } else if (abs(i_offset) > 50) {
        return 5;
    //The function generator can only return a maximum of +/- 10V
    } else if (abs(i_offset) + i_amplitude > 100) {
        return 10-amplitude;
    } else {
        return offset;
    }
}

/**
 * Make sure that the amplitude is within proper ranges
 * @param offset
 * @param amplitude
 * @return
 */
float mcp4251_driver_verify_amplitude(float offset, float amplitude)
{
    //Ugly workaround to handle that abs is int based
    int i_offset = (int)(offset*10);
    int i_amplitude = (int)(amplitude*10);
```

```
    if (abs(i_offset) > i_amplitude) {
        if (offset < 0)
            return (-1)*offset;
        else
            return offset;
    } else if (i_amplitude > 100) {
        return 10;
    } else if (abs(i_offset) + i_amplitude > 100) {
        return 10-offset;
    } else {
        return amplitude;
    }
}

/**
 * Convert the float offset to a uint which can be interpreted by the offset potentiometer
 * @param offset
 * @param amplitude
 * @return
 */
uint16_t mcp4251_driver_calculate_offset_uint(float offset, float amplitude)
{
    return (uint16_t)(-((128*offset)/amplitude)+128);
}

/**
 * Convert an ampltiude float to a uint which can be interpreted by the ampltiude potentiometer
 * @param amplitude
 * @return
 */
uint16_t mcp4251_driver_calculate_amplitude_uint(float amplitude)
{
    return (uint16_t)((256*amplitude)/10);
}
```

Libraries/delay_helper.h

```
/*
 * File:   delay_helper.h
 *
 * Created on May 12, 2021, 11:11 AM
 */

#include "../types.h"

#ifndef DELAY_HELPER_H
#define           DELAY_HELPER_H

typedef struct DelayHelper {
    void (*delay_fourth_us) (uint8_t num);
    void (*delay_us) (uint16_t us);
    void (*delay_ms) (uint16_t ms);
} CDelayHelper;

void delay_helper_delay_fourth_us(uint8_t num);
void delay_helper_delay_us(uint16_t us);
void delay_helper_delay_ms(uint16_t ms);
void init_delay_helper(CDelayHelper* delay_helper);

#endif  /* DELAY_HELPER_H */
```

Libraries/delay_helper.c

```
#include "delay_helper.h"
#include "../mcc_generated_files/mcc.h"

void delay_helper_delay_fourth_us(uint8_t num) {
    TMR2_Period8BitSet(num);
    TMR2_Start();
    //Ugly delay
    while (TMR2_Counter8BitGet() != 0) {}
    return;
}

void delay_helper_delay_us(uint16_t us) {
    uint8_t full_runs = (uint8_t)(us/64);
    uint8_t overflow_us = us % 64;
    //Start by calculating max number of periods and overflow
    for (uint8_t full_run_count = 0; full_run_count < full_runs; full_run_count++) {
        TMR2_Period8BitSet(255);
        TMR2_Start();
        while (TMR2_Counter8BitGet() != 0) {}
    }
    TMR2_Period8BitSet(overflow_us*4);
    TMR2_Start();
    while (TMR2_Counter8BitGet() != 0) {}
    return;
}

void delay_helper_delay_ms(uint16_t ms) {
    for (uint16_t ms_count=0; ms_count < ms; ms_count++) {
        delay_helper_delay_us(1000);
    }
    return;
}

void init_delay_helper(CDelayHelper* delay_helper) {
    delay_helper->delay_fourth_us = delay_helper_delay_fourth_us;
    delay_helper->delay_us = delay_helper_delay_us;
    delay_helper->delay_ms = delay_helper_delay_ms;
}
```

Libraries/display_helper.h

```
/*
 * File:   display_helper.h
 *
 * Created on April 15, 2021, 11:30 AM
 */

#ifndef DISPLAY_HELPER_H
#define          DISPLAY_HELPER_H

#include "../Drivers/display.h"
#include "../Libraries/wavetables.h"
#include "delay_helper.h"
#include <math.h>

#define DISPLAY_AMPLTIDUE_OFFSET 11
#define DISPLAY_OFFSET_OFFSET 8
#define DISPLAY_FREQ_OFFSET 6
#define DISPLAY_SIG_OFFSET 5
#define DISPLAY_MODE_OFFSET 15

typedef enum {
    display_amplitude,
    display_offset
} e_display_shown_mode;

typedef struct DisplayHelper {
    CDisplayDriver display_driver;
    void (*update_signal_type) (struct DisplayHelper* self, e_waveform_type signal_type);
    void (*update_frequency) (struct DisplayHelper* self, uint32_t frequency);
    void (*update_offset) (struct DisplayHelper* self, float offset);
    void (*update_amplitude) (struct DisplayHelper* self, float amplitude);
    void (*update_shown_mode) (struct DisplayHelper* self, e_display_shown_mode mode);
    void (*write_string) (struct DisplayHelper* self, char* display_string, uint8_t col_pos,
e_display_row_selection row);
} CDisplayHelper;

void init_display_helper(CDisplayHelper* display_helper);
void display_helper_float_to_char_str(char* char_str, float float_val);
void display_helper_update_signal_type(CDisplayHelper* self, e_waveform_type signal_type);
void display_helper_update_frequency(CDisplayHelper* self, uint32_t frequency);
void display_helper_update_offset(CDisplayHelper* self, float offset);
void display_helper_update_amplitude(CDisplayHelper* self, float amplitude);
```

```
void display_helper_update_shown_mode(CDisplayHelper* self, e_display_shown_mode
mode);
void display_helper_write_string(CDisplayHelper* self, char* display_string, uint8_t col_pos,
e_display_row_selection row);
void strcpy(char* target_string, char* string);

#endif  /* DISPLAY_HELPER_H */
```

Libraries/display_helper.c

```
#include "display_helper.h"

/**
 * Display helper constructor
 * @param display_helper
 */
void init_display_helper(CDisplayHelper* display_helper)
{
    init_display_driver(&display_helper->display_driver);

    display_helper->update_signal_type = display_helper_update_signal_type;
    display_helper->update_frequency = display_helper_update_frequency;
    display_helper->update_offset = display_helper_update_offset;
    display_helper->update_amplitude = display_helper_update_amplitude;
    display_helper->update_shown_mode = display_helper_update_shown_mode;
    display_helper->write_string = display_helper_write_string;

    //Populate the screen with the labels
    display_helper->write_string(display_helper, "Amplitude: ", 0, display_row1);
    display_helper->write_string(display_helper, "Offset: ", 0, display_row2);
    display_helper->write_string(display_helper, "Freq: ", 0, display_row3);
    display_helper->write_string(display_helper, "Sig: ", 0, display_row4);
    display_helper->write_string(display_helper, "MODE: ", DISPLAY_MODE_OFFSET-6,
display_row4);
}

/**
 * Helper function for writing charstrings to the display
 * @param self
 * @param display_string
 * @param col_pos
 * @param row
 */
void display_helper_write_string(CDisplayHelper* self, char* display_string, uint8_t col_pos,
e_display_row_selection row)
{
    char* cur_char = display_string;
    self->display_driver.goto_position(&self->display_driver, col_pos, row);
    while (*cur_char != '\0') {
        self->display_driver.write_character(&self->display_driver, *cur_char);
        cur_char++;
    }
```

```
}

/**
 * Minimal strcpy implementation
 * @param target_string
 * @param string
 */
void strcpy(char* target_string, char* string)
{
    uint8_t i = 0;
    while (string[i]) {
        target_string[i] = string[i];
        i++;
    }
}

/**
 * Update the signal type shown on the screen
 * @param self
 * @param signal_type
 */
void display_helper_update_signal_type(CDisplayHelper* self, e_waveform_type signal_type) {
    char type_string[4];
    switch (signal_type) {
        case wave_sinus:
            strcpy(type_string, "Sin");
            break;
        case wave_triangle:
            strcpy(type_string, "Tri");
            break;
        case wave_saw:
            strcpy(type_string, "Saw");
            break;
        case wave_dc:
            strcpy(type_string, "DC");
            break;
        case wave_square:
            strcpy(type_string, "Sqr");
            break;
    }

    display_helper_write_string(self, type_string, DISPLAY_SIG_OFFSET, display_row4);
}
```

```
/**
 * Update the displayed frequency of the display
 * @param self
 * @param frequency
 */
void display_helper_update_frequency(CDisplayHelper* self, uint32_t frequency)
{
    char freq_string[11];
    uint8_t num_chars = 0;
    uint32_t freq_rest = frequency;
    //Calculate number of digits
    while (freq_rest > 0) {
        freq_rest = freq_rest/10;
        num_chars++;
    }

    //uint8_t num_chars = (uint8_t)log10(frequency);
    freq_string[num_chars+1] = '\0';
    for (uint8_t char_count = num_chars; char_count > 0; char_count--) {
        freq_string[char_count-1] = (frequency % 10) + '0';
        frequency = (uint32_t)(frequency/10);
    }

    display_helper_write_string(self, freq_string, DISPLAY_FREQ_OFFSET, display_row3);
}

/**
 * Generic helper function for converting a float to a char string
 * @param display_helper
 * @param char_str
 * @param float_val
 */
void display_helper_float_to_char_str(char* char_str, float float_val)
{
    uint8_t count_offset = 0;
    if (float_val < 0) {
        char_str[0] = '-';
        count_offset++;
        //Convert to positive to make it easier to divide
        float_val = float_val*(-1);
    }

    //convert to an int, multiply by ten to make it possible to use log10 and modulo
    uint8_t uint_offset = (uint8_t)(float_val*10);
```

```
    /*
     * In case there is a "-"-sign in front, offset the start index and end index by one; 2 equals a
total of 3 indexes, which is the
     * minimum for displaying one digit and one decimal
     */

    uint8_t digits = 0;
    if (uint_offset >= 100) {
        digits = 2;
    } else {
        digits = 1;
    }

    uint8_t num_chars = digits + 2 + count_offset; //up to 2 digits + 1 decimal place + 1 dot
    char_str[num_chars+1] = '\0';

    for (uint8_t char_count=num_chars; char_count > 0; char_count--) {
        if (char_count == num_chars-1) {
            char_str[char_count-1] = '.';
        } else {
            char_str[char_count-1] = (uint_offset % 10) + '0';
            uint_offset = (uint8_t)(uint_offset/10);
        }
    }
}

/**
 * Display the offset value. Can be between -5 and +5 Converts the float to a char string
 * @param self
 * @param offset
 */
void display_helper_update_offset(CDisplayHelper* self, float offset)
{
    char offset_string[6];
    display_helper_float_to_char_str(offset_string, offset);
    display_helper_write_string(self, offset_string, DISPLAY_OFFSET_OFFSET, display_row2);
}

/**
 * Update the amplitude value shown on the display, -10 to +10
 * @param self
 * @param amplitude
 */
void display_helper_update_amplitude(CDisplayHelper* self, float amplitude)
```

```
{
    char amplitude_string[6];
    display_helper_float_to_char_str(amplitude_string, amplitude);
    display_helper_write_string(self, amplitude_string, DISPLAY_AMPLTIDUE_OFFSET,
display_row1);
}

/**
 * Display whether rotating the configuration dial will modify the offset or amplitude
 * @param self
 * @param mode
 */
void display_helper_update_shown_mode(CDisplayHelper* self, e_display_shown_mode
mode) {
    char mode_string[4];
    switch (mode) {
        case display_amplitude:
            strcpy(mode_string, "Amp");
            break;
        case display_offset:
            strcpy(mode_string, "Off");
            break;
    }

    display_helper_write_string(self, mode_string, DISPLAY_MODE_OFFSET, display_row4);
}
```

Libraries/wavetables.h

```
/*
 * File:   wavetables.h
 *
 * Created on April 15, 2021, 11:36 AM
 */

#ifndef WAVETABLES_H
#define           WAVETABLES_H

#include "../types.h"

typedef enum {
    wave_sinus = 0,
    wave_triangle = 1,
    wave_saw = 2,
    wave_square = 3,
    wave_dc = 4
} e_waveform_type;

uint8_t sinus_wavetable[] = {
    16, 19, 21, 24, 26, 28, 30, 31, 31, 31, 30, 28, 26, 24, 21, 19,
    16, 12, 10, 7, 5, 3, 1, 0, 0, 0, 1, 3, 5, 7, 10, 12
};

typedef struct TimerSettings {
    uint8_t runs;
    uint8_t partial_runs;
} s_TimerSettings;

typedef struct WaveTableManager {
    uint8_t current_position;
    e_waveform_type current_waveform;
    void (*select_waveform) (struct WaveTableManager* self, e_waveform_type waveform_type);
    uint8_t (*next_position) (struct WaveTableManager* self);
    s_TimerSettings (*calculate_timer) (uint32_t frequency);
    uint32_t (*verify_frequency) (long frequency);
} CWaveTableMananger;

void init_wave_table_manager(CWaveTableMananger* self);
void wave_table_manager_select_waveform (CWaveTableMananger* self, e_waveform_type
waveform_type);
uint32_t wave_table_manager_verify_frequency(long frequency);
```

```
s_TimerSettings wave_table_manager_calculate_timer(uint32_t frequency);
uint8_t wave_table_manager_next_position (CWaveTableMananger* self);

#endif  /* WAVETABLES_H */
```

Libraries/wavetables.c

```
#include "wavetables.h"

void init_wave_table_manager(CWaveTableMananger* self)
{
   self->select_waveform = wave_table_manager_select_waveform;
   self->next_position = wave_table_manager_next_position;
   self->calculate_timer = wave_table_manager_calculate_timer;
   self->verify_frequency = wave_table_manager_verify_frequency;

   self->current_position = 0;
   self->current_waveform = wave_sinus;
}

void wave_table_manager_select_waveform(CWaveTableMananger* self, e_waveform_type
waveform_type)
{
   self->current_position = 0;
   self->current_waveform = waveform_type;
}

uint8_t wave_table_manager_next_position (CWaveTableMananger* self)
{
   uint8_t out_value;

   switch (self->current_waveform) {
      case wave_sinus:
         out_value = sinus_wavetable[self->current_position];
         break;
   }

   if (self->current_position >= 31) {
      self->current_position = 0;
   } else {
      self->current_position++;
   }

   return out_value;
}

uint32_t wave_table_manager_verify_frequency(long frequency)
{
   if (frequency < 4) {
```

```
        return 4;
    } else if (frequency > 250000) {
        return 250000;
    } else {
        return (uint32_t)frequency;
    }
}

s_TimerSettings wave_table_manager_calculate_timer(uint32_t frequency)
{
    //Frequency of Timer0 is 8MHz => 8*10^6. Timer should be a 32th of the selected frequency
    //uint16_t timer = (uint16_t)(65535-(8000000/(32*frequency)));

    s_TimerSettings return_value;

    //One full run is 250980.4 Hz, however times 32
    uint16_t total_counter = (16000000/(32*frequency));
    return_value.runs = (uint8_t)(total_counter/255);
    return_value.partial_runs = total_counter % 255;

    //uint16_t timer = (uint16_t)(8000000/(frequency*32));
    return return_value;
}
```

Bilaga 4 - Kostnadskalkyl

Komponentnamn	Antal	Antal Köper	Länk	Pris	Beskrivning	Totalpris
1n5819	2	10	https://www.mouser.se/ProductDetail/Rectron/1N5819-T?qs=w4rXzN23dC3jxJaDspMQbA%3D%3D	0.15€	Shottkydiod	0.30€
4608X-102-103LF	1	10	https://www.mouser.se/ProductDetail/Bourns/4608X-102-103LF?qs=%2Fha2pyFadujGrwpxGSma59NTST%252Byzog%252BiVVFuJklW55nyGBD9QY5SQ%3D%3D	0.16€	Resistorkapsel	0.16€
4609H-101-103LF	3	10	https://eu.mouser.com/ProductDetail/Bourns/4609H-101-103LF?qs=%2Fha2pyFaduicZHRNP2rCRyHlXZ%252Bq0N62JcEnlb17Qa8GYwQOaGTg2Q%3D%3D	0.41€	Resistor-array, 10k	1.24€
BC327-40	1	10	https://www.mouser.se/ProductDetail/ON-Semiconductor-Fairchild/BC32740BU?qs=u3sZpSG73%252B4%252BlpJrwvvqyQ%3D%3D	0.20€	PNP-transistor	0.20€
BC337-40	2	10	https://www.mouser.se/ProductDetail/Taiwan-Semiconductor/BC337-40-A1?qs=dD%252BUdDqNEJbF8KTuB1XALg%3D%3D	0.20€	Transistor	0.41€
CD4066BEE4	1	5	https://www.mouser.se/ProductDetail/Texas-Instruments/CD4066BEE4?qs=LSR%252BNeaw%2FwCatOlSuyR4WA%3D%3D	0.42€	CMOS switcharray	0.42€
CD4072BE	2	10	https://eu.mouser.com/ProductDetail/Texas-Instruments/CD4072BE?qs=%2Fha2pyFaduht9sw4eFYABRUJxzjBGNWoHkoTQHsxd9s%3D	0.43€	OR-gates	0.85€
CD74HCT165E	2	10	https://eu.mouser.com/ProductDetail/Texas-Instruments/CD74HCT165E?qs=D5pVkbrsqgJS4tjF0EbWXA%3D%3D	0.48€	Shift register	0.96€
Induktor 180uH	1	4	https://www.mouser.se/ProductDetail/Coilcraft/RFS1317-184KL?qs=W38ilblRkRmDEglK0ApGpQ%3D%3D	1.54€	Induktor	1.54€
Kondensator 47uF	1	10	https://eu.mouser.com/ProductDetail/Wurth-Elektronik/860020472006?qs=sGAEpiMZZMvwFf0viD3Y3aZipiehufnXP0KQMYM5VCQGR0fvVZssXw%3D%3D	0.09€	Elektrolyt Kondensator	0.09€
Kondensator: 0.1uF	8	20	https://eu.mouser.com/ProductDetail/KEMET/R82DC3100AA50J?qs=06HLYMEcjuU8dbQf6h7xiA%3D%3D	0.12€	Kondensator	0.99€

Kondensator: 0.33uF	1	10	https://www.mouser.se/ProductDetail/KEMET/R82DC3330AA60J?qs=sGAEpiMZZMsh%252B1woXyUXjwYbDi8EMxhfAtvY3zZGqwq%3D	0.20€	Kondensator	0.20€
Kondensator: 1000uF	1	10	https://eu.mouser.com/ProductDetail/KEMET/ESK108M025AH4EA?qs=w9hhVRqtvfm1CxCOstj5FA%3D%3D	0.21€	Kondensator	0.21€
Kondensator: 100pF	2	10	https://www.mouser.se/ProductDetail/TDK/FG18C0G1H101JNT06?qs=sGAEpiMZZMuMW9TJLBQkXsjX9hJVC7r9vbqXPKWVClq%3D	0.09€	Keramisk kondensator	0.18€
Kondensator: 100uF	1	10	https://www.mouser.se/ProductDetail/Lelon/RGA101M1EBK-0611G?qs=sGAEpiMZZMsh%252B1woXyUXj%252BLhpznb1dGkVkput2lKmCs%3D	0.10€	Kondensator	0.10€
Kondensator: 120pF	1	10	https://www.mouser.se/ProductDetail/TDK/FG28C0G1H121JNT06?qs=sGAEpiMZZMuMW9TJLBQkXoYaKhvJxpqMbJNWb%252B4Zp%252Bw%3D	0.09€	Keramisk kondensator	0.09€
Kondensator: 430pF	1	10	https://eu.mouser.com/ProductDetail/KEMET/C317C431J5G5TA?qs=sGAEpiMZZMsh%252B1woXyUXj8Q7fntrp%252BEy20qP6t%2FI8UY%3D	0.38€	Keramisk Kondensator	0.38€
Kondensator: 680pF	1	10	https://www.mouser.se/ProductDetail/TDK/FG28C0G1H681JNT06?qs=sGAEpiMZZMuMW9TJLBQkXsjX9hJVC7r98PAx2ny296I%3D	0.10€	Keramisk Kondensator	0.10€
Kondensator: 68uF	1	10	https://eu.mouser.com/ProductDetail/Wurth-Elektronik/860010473006?qs=sGAEpiMZZMvwFf0viD3Y3aZipiehufnXx57sZ%2FDo%2FanewuzOzMYv7w%3D%3D	0.10€	Elektrolyt Kondensator	0.10€
L7805	1	4	https://www.mouser.se/ProductDetail/STMicroelectronics/L7805CV?qs=9NrABl3fj%2FqplZAHiYUxWg%3D%3D	0.44€	Lineär spänningsregulator	0.44€
MC34063EBN	1	5	https://www.mouser.se/ProductDetail/STMicroelectronics/MC34063EBN?qs=OMlU9E68qqTFvsLXUr%2F91g%3D%3D	0.50€	DC-DC-konverterare	0.50€
MCP4251-103	1	4	https://www.mouser.se/ProductDetail/Microchip-Technology/MCP4251-103E-P?qs=tEIKvJhp9wVOWJvgbV8hBA%3D%3D	0.86€	Digital potentiometer	0.86€
PEC12R-3020F-N0024	3	10	https://www.mouser.se/ProductDetail/652-PEC12R3020FN0024/	0.64€	Rotary encoder	1.93€

PIC18F27Q10	1	4	https://www.mouser.se/ProductDetail/Microchip-Technology/PIC18F27Q10-I-SP?qs=y6ZabqHbY%252BvqS1wJzQuUiq%3D%3D	1.39€	MCU	1.39€
RC2004	1	1	https://www.amazon.de/-/en/Display-Serial-Adapter-Yellow-Green/dp/B07XT7LN61/ref=sr_1_17?dchild=1&keywords=2004+LCD&qid=1629464299&sr=8-17	7.42€	ASCI LCD-display	7.42€
Resistor 4.7k	1	10	https://www.mouser.se/ProductDetail/Welwyn-Components-TT-Electronics/MFR3-4K7FC?qs=rXkOWCGOzpJwOznENFDEfw%3D%3D	0.09€	Resistor	0.09€
Resistor: 0.22	1	5	https://eu.mouser.com/ProductDetail/KOA-Speer/BPR58CR22J?qs=1He0yLMpldAHlXM%2Fx8XqjQ%3D%3D	0.67€	Sense resistor	0.67€
Resistor: 100ohm 2W	2	10	https://www.mouser.se/ProductDetail/Welwyn-Components-TT-Electronics/MFP2-100RJI?qs=%252B5fvtLYfc7IXgGrVfZDQ6w%3D%3D	0.10€	Resistor	0.20€
Resistor: 10k	7	50	https://www.mouser.se/ProductDetail/Vishay-Beyschlag/MBB02070C1002FRP00?qs=Jr4%2Ft4s12JUmAfKOFUNhig%3D%3D	0.08€	Resistor	0.53€
Resistor: 1k	2	10	https://eu.mouser.com/ProductDetail/TE-Connectivity-Neohm/LR0204F1K0?qs=sGAEpiMZZMsPqMdJzcrNwtfZC8wqDsjlXf3B5gPsb%2Fw%3D	0.05€	Resistor	0.10€
Resistor: 1.2k	5	20	https://www.mouser.se/ProductDetail/yageo/mfr-25fte52-1k2/?qs=oAGoVhmvjhwqc405HqsL8g==&countrycode=DE&currencycode=EUR	0.05€	Resistor	0.26€
Resistor: 390	5	20	https://www.mouser.se/ProductDetail/yageo/mfr-25fte52-390r/?qs=oAGoVhmvjhw%252bSk8JpY7mLQ==&countrycode=DE&currencycode=EUR	0.05€	Resistor	0.26€
Resistor: 2k	2	10	https://www.mouser.se/ProductDetail/Yageo/MFR-25FBF52-2K?qs=oAGoVhmvjhym1upYmL0LFQ%3D%3D	0.04€	Resistor	0.07€
Resistor: 4.87k	2	10	https://www.mouser.se/ProductDetail/Yageo/MFR-25FBF52-4K87?qs=oAGoVhmvjhxHWNv0%252BUzo5Q%3D%3D	0.04€	Resistor	0.07€
TL072ACP	1	4	https://www.mouser.se/ProductDetail/Texas-Instruments/TL072ACP?qs=0OT4q4QBUTLFvIZ%252B%2FvrwoQ%3D%3D	0.76€	OP-Amp	0.76€

TL074BCN	2	5	https://www.mouser.se/ProductDetail/Texas-Instruments/TL074BCN?qs=vxEfx8VrU7BHurOY5iQdiA%3D%3D	0.95€	OP-Amp	1.90€
TL1105DF100Q	3	10	https://www.mouser.se/ProductDetail/612-TL1105DF100Q	0.12€	Microbrytare	0.35€
FA-T220-25E	1	2	https://www.mouser.se/ProductDetail/Ohmite/FA-T220-25E?qs=wmFqszA%252Be7PHdQv0%252BKXavA%3D%3D	1.58€	Kylfläns	1.58€
WP710A10ID5V	5	10	https://www.mouser.se/ProductDetail/kingbright/wp710a10id5v/?qs=jBF9H7RTBaQpWK8VEJXj4w==&countrycode=DE&currencycode=EUR	0.25€	Lysdiod 5V	
WP710A10SRD14V	5	10	https://www.mouser.se/ProductDetail/kingbright/wp710a10srd14v/?qs=jBF9H7RTBaQWYsbU5GU6zQ==&countrycode=DE&currencycode=EUR	0.27€	Lysdiod 15V	
112404	2	8	https://www.mouser.se/ProductDetail/amphenol/112404/?qs=1K%2FD%252bSl2CTNlg0LiPKjnoQ==&countrycode=DE&currencycode=EUR	2.64€	BNC-connector	5.28€
163-179PH-EX	1	6	https://www.mouser.se/ProductDetail/kobiconn/mj-179ph/?qs=Xb8IjHhkxj5l2UOaIqcGCw==&countrycode=DE&currencycode=EUR	1.12€	DC-kontakt för nätaggregat	1.12€
XG4M-3030	2	8	https://www.mouser.se/ProductDetail/omron/xg4m-3030/?qs=zPELyVczQCN6AwT0SsZloQ==&countrycode=DE&currencycode=EUR	1.91€	Kontakt på flatkabel	3.82€
892-70-028-10-001101	2	10	https://www.mouser.se/ProductDetail/preci-dip/892-70-028-10-001101/?qs=KBM%2FMdeLKeh4q26F0WaCsg==&countrycode=DE&currencycode=EUR	0.86€	PCB-monterad kontakt för flatkabel	1.73€
3365/34-CUT-LENGTH	1	4	https://www.mouser.se/ProductDetail/3m/3365-34-cut-length/?qs=QV10cN0MjFvd%252bE2sukslXQ==&countrycode=DE&currencycode=EUR	1.59€	Flatkabel (antal=meter)	1.59€
PCB Moderkort	1	5		1.71€		1.71€
PCB Dotterkort	1	5		1.69€		1.69€
Summa komponenter						44.84€