Projektrapport
Synthesizer 2000
EITA15 - Digitala System

LUNDS UNIVERSITET

Lunds Tekniska Högskola

Grupp 6: Jacob Persson, Dennis Pålsson, Ramtin Azimzadehtork & Sebastian Holm
Handledare: Bertil Lindvall & Lars-Göran Larsson

Sammanfattning

I det här projektet konstruerades en synthesizer baserad på ett en oktav musikinstrument samt ett antal funktionsknappar. Detta gjordes som en sista uppgift i kursen EITA 15. Instrumentet byggdes upp på en 8-bitar AVR mikrokontroller, ATMega1284, och en DA-omvandlare, TLC7524. Resultatet blev en fungerande synthesizer kapabel att spela 12 noter samt ändra oktav och vågformat. Dessa vågformat är sinusvåg, sågtandsvåg samt fyrkantsvåg.

Arbetet började med att hämta information kring komponenter , främst mikrokontrollern och DA-omvandlaren, om vilka pins resterande komponenter skulle kopplas till. Detta hämtades från olika datablad. När all information var hämtad konstruerades ett kretsschema. Därefter skrevs ett första utkast på kod. Till sist kopplades alla komponenter på ett kopplingsplatta efter kretsschemat.

Abstract

In this project, a synthesizer based on a one-octave musical instrument and a number of function buttons was constructed. This was done as a final assignment in the course EITA 15. The instrument was built on an 8-bit AVR microcontroller, ATMega1284, and a DA converter, TLC7524. The result was a working synthesizer capable of playing 12 notes and changing octaves and waveforms. These waveforms are sine wave, sawtooth wave and square wave.

The work began by retrieving information about components, mainly the microcontroller and the DA converter, about which pins the remaining components should be connected to. This was taken from various data sheets. Once all the information was retrieved, a circuit diagram was constructed. Then a first draft of code was written. Finally, all components were connected on a connection plate according to the circuit diagram.

Innehållsförteckning

1 Inledning 4
1.1 Bakgrund 4
1.2 Syfte 4
1.3 Målformulering
Målet var att konstruera en synt med förmågan att kunna spela en oktav åt gången samt kunna ändra oktav och ändra vågtyp med knappar 4
1.4 Problemformulering 4
2 Teknisk bakgrund 5
2.1 Skapa digitala vågor 5
2.2 Sända med rätt frekvens 6
2.3 Digitalt till analogt ljud 7
3 Metod 8
3.1 Hårdvara 8
3.2 Mjukvara 9
3.3 Källkritik 9
4 Analys 9
4.1 Resultat från mätningar 9
4.2 Kravspecifikationer 9
4.3 Projektval och motivering 10
4.3.1 DDS 10
4.3.2 DAC 10
4.3.3 Kristall 10
4.3.4 Fasackumolatorn 11
4.4 Problem och deras lösningar 11
5 Resultat 11
6 Slutsats 14
6.2 Framtida utvecklingsmöjligheter 14
6.2.1 ADSR 14
6.2.2 Skärm 15
6.2.3 Polyfonisk 15
8 Källförteckning 16
9 Appendix 16
9.1 Kretsschema 16
9.2 C kod 17

1 Inledning

1.1 Bakgrund

Som ett avslutande kursmoment i kursen *Digitala System* gavs uppgiften att skapa en grupp och konstruera en prototyp baserad på mikrokontrollern, ATMega1284. I detta fall programmerades mikrokontrollern i programmeringsspråket C. Detta projekt löser hur ljud kan spelas upp baserat på knapptryck.

1.2 Syfte

Projektets syfte var att konstruera en fungerande prototyp från en idé baserad på ATMega1284 samt att utveckla problemlösningsförmågan, samarbetsförmågan och ingenjörsmässigt skrivande. Det förväntade resultatet är en synthesizer som kan spela upp ljud i olika toner och olika oktav samt vågformer.

1.3 Målformulering

Målet var att konstruera en synt med förmågan att kunna spela en oktav åt gången samt kunna ändra oktav och ändra vågtyp med knappar.

1.4 Problemformulering

Det första problemet som uppstod var vilka komponenter behövdes och hur skulle dessa kopplas samman? Det andra problemet var att ljudkvalitén var mindre tillfredsställande för den kulturellt känsliga. Det tredje problemet var störningar i kretsen.

2 Teknisk bakgrund

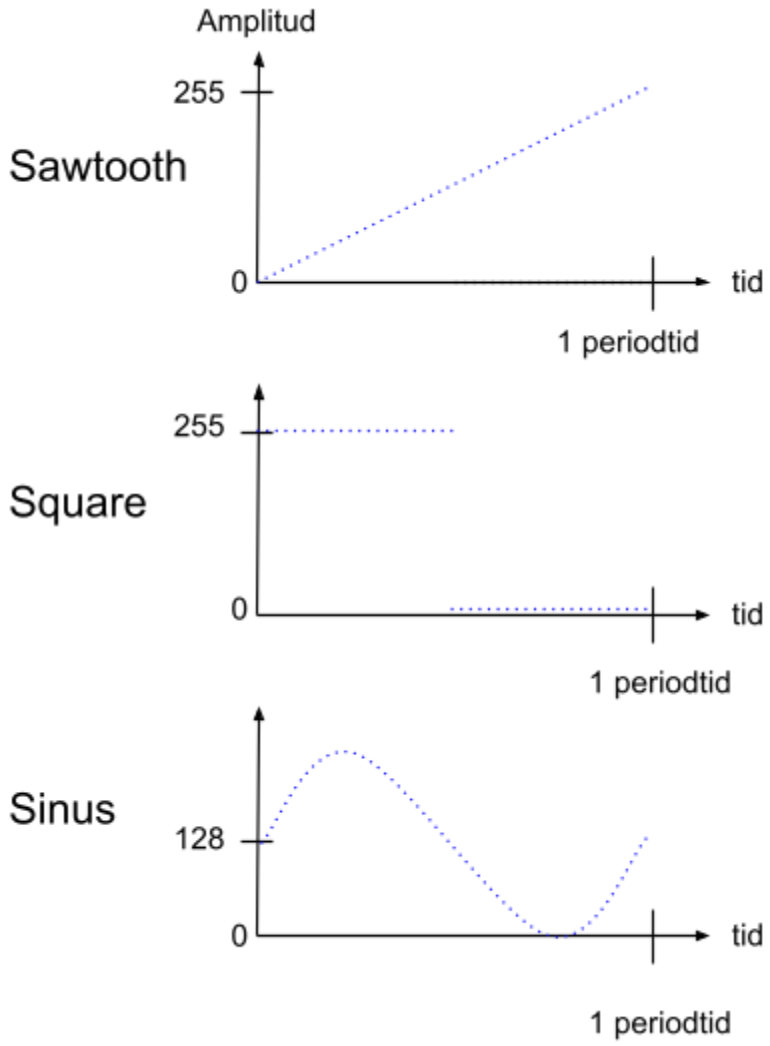
Syntar kan göras på flera olika sätt men projektets valda metod för att skapa ljud var direct digital synthesis även kallat DDS. För detta användes följande komponenter:

- **Processor**: AVR ATmega1284 är en 8-bitars, enkärnig mikroprocessor med 16 MHz klockfrekvens.
- **DA-omvandlare:** TLC7524 gör om en 8-bitars insignal till en analog utsignal.
- **Lågpassfilter:** Ett RC filter för att eliminera brus och jämna ut utsignalen**.**
- **Högtalare**: En analog elektrisk insignal går in i en spole och skapar ett elektriskt fält. Då spolen befinner sig intill en magnet i högtalaren kommer fältet att få spolen att röra sig. Denna rörelse vibrerar ett membran fäst i spolen och ger upphov till en ljudsignal.

Dessa 4 komponenter är grunden i synten där processorn har till uppgift att generera vågor i digital form och sända ut dem med en vald frekvens. DA-omvandlaren gör om den digitala signal till en analog signal och efter att den analoga signalen passerat genom lågpassfiltret spelar högtalaren upp ljudet av denna.

2.1 Skapa digitala vågor

Alla möjliga vågformer kan skapas med en synt men projektets valda vågor för att skapa ljud var sawtooth, square och sinusvågor. Dessa vågors amplitud delades upp i 255 lika stora steg mellan dess botten och topp. Detta då det motsvarar en 8 bitars signal där 0000 0000 motsvarar amplitudnivå 0 och 1111 1111 motsvarar amplitudens topp 255.
Under en periodtid av en våg skapades Sawtooth genom att räkna upp ett steg i taget från 0 till 255, Square genom att sätta hälften av signalerna till 0 och resterande till 255. Slutligen skapades sinusvågen med hjälp av en lookup table, LUT med 256 samplade amplitudvärden av dess svängning under en periodtid.
När aplitudvärdena plottas mot tiden framträder vågornas utseende enligt figur 1.



Figur 1: De tre vågorna som genereras i processorn.

2.2 Sända med rätt frekvens

Om vågorna nu är uppbyggda med amplitudvärden i rätt ordning kan man skicka iväg dessa med en frekvens man väljer att spela, f_{out}. För att göra detta används DDS där en vald frekvens kan fås fram genom att använda klockhastigheten i processorn, f_c enligt ekvationen:

$$f_{out} = \frac{M \, x \, f_c}{2^n} \qquad \text{(ekv. 1)}$$

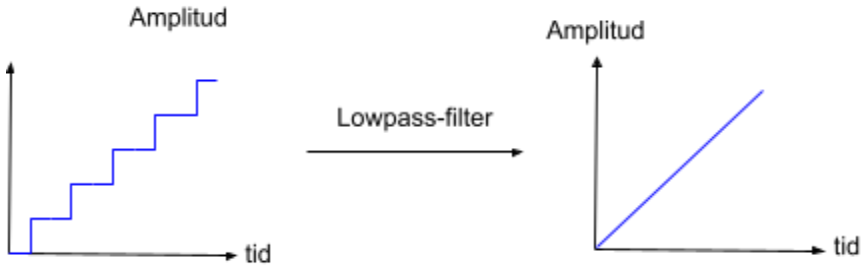
där M är tuning word och 2^n är storleken på fasackumulatorn vilka begrepp diskuteras i detalj i Analog Dialogue. (Murphy & Slattery 2004)

Men i stora drag så kan fasackumulatorn sättas till ett fast värde och då f_c och f_{out} är kända kan tuning word räknas ut. Denna används sedan för att stega igenom amplitudvärdena i ordning och skicka ut dem i bestämda tidsintervall vilket ger rätt frekvens på utsignalen.

2.3 Digitalt till analogt ljud

För att göra om den digitala signalen till analog används en DA-omvandlaren kopplad till en ingående spänning, V_I som anger maxamplituden på utgående signal, V_o. Även de 8 ingångarna för den digitala signalen ansluts. När den digitala signalen kommer in görs den om till det decimala talet D som anger amplituden av utsignalen med ekvationen $V_0 = V_I(D/256)$. Så kommer 0000 0000 in till DA-omvandlaren blir utsignalen 0 V och 1111 1111 blir utsignalen den maximala V_o.

Att den analoga signalen är uppdelad i 256 nivåer ger vågen ett trappstegsformat utseende. Genom att låta signalen gå genom ett lowpass-filter mjukas kanterna till och får en form mer lik vågformen som eftersträvas vilket visas i figur 2.



Figur 2: Den analoga signalen är först uppdelad i 256 nivåer men efter att ha passerat ett lowpass-filter jämnas signalen ut.

Den färdigbehandlade signalen går sedan ut till högtalaren som spelar upp ljudet.

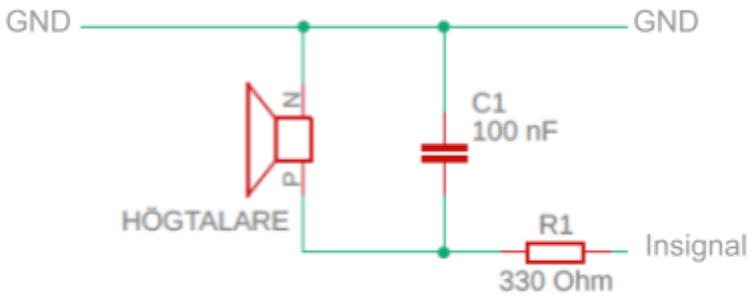
3 Metod

3.1 Hårdvara

Processorn, DA-omvandlaren, högtalaren, resistorn och kondensatorer kopplades på ett perfboard enligt kretsschemat i Appendix 9. Majoriteten av komponenterna förbands med kopplingstråd m.h.a. virverktyg medan strömförsörjningen och JTAG-pinsen mot processorn löddes fast. Även knapparna löddes efter att de virats.

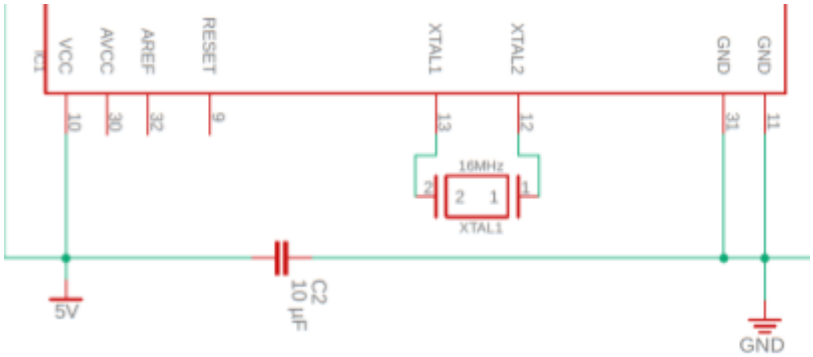
Av de 15 knapparna användes 12 som tangenter, 2 för att skifta upp eller ner en oktav och en för att bläddra mellan sawtooth, square och sinusvåg.

Lowpass-filtret som utgörs av resistorn och kondensatorn ses i kretsschemat vilket ses i figur 3.



Figur 3: I kretsschemat ses R_1 och kondensatorn vilket utgör lowpass-filtret i kretsen innan signalen går in i högtalaren.

En 16MHz kristall kopplades mot processorns XTAL-ingångar för att höja dess frekvens och ytterligare en kondensator sattes in för att stabilisera strömförsörjningen. Detta ses i figur 4.



Figur 4: Inkopplingen av kristallen XTAL på 16 MHz som kopplades mot processorns XTAL-ingångar samt den andra kondensatorn C2 i kretsen.

3.2 Mjukvara

Koden finns i Appendix 9.2 och skrevs i språket C i IDP-programmet Atmel studio 7. kretsschemat ritades med programvaran Fusion 360.

Fasackumulatorn sattes till 2^{32} i ekvation 2 och gjordes om till $M = \frac{f_{out} \cdot 2^{32}}{f_c}$ för att få fram tuning word

Detta användes för att ge rätt frekvens på utsignalen.

3.3 Källkritik

Analog Dialogue är ett nätforum för personer som är intresserade av mjukvara, elektriska kretsar och system som används inom signalbehandling. Utgivaren är det ansedda företaget Analog Devices som bl.a. är leverantör till Nintendo och bör anses som en trovärdig källa. Speciellt då denna artikel länkas till på flera andra ställen och DDS teorin diskuterad här uppenbarligen fungerar.

4 Analys

4.1 Resultat från mätningar

Filtret fungerade bra på sinusvågen och fick denna att komma bra nära den sökta vågformen vilket speciellt ses i figur 10. Denna effekt verkade vara extra viktig vid höga frekvenser om man jämför denna med figur 9.
Hos sawtooth och squarevågen fungerade den däremot sämre och förvrängde vågorna något vilket ses i figur 7 och 8. Däremot gav det en liknande effekt som ADSR och gav tonerna ett mindre skarpt ljud. Överlag gav filtret även mindre högfrekventa störningar hos alla vågtyper vilket var väntat.

4.2 Kravspecifikationer

- Med 12 tangenter kunna spela en kromatisk skala.
- Med knappar kunna ändra oktav som spelas
- Med knappar kunna ändra typ av våg som spelas
- Vågformer som ska genereras är sinus, square och sawtooth

4.3 Projektval och motivering

4.3.1 DDS

Det finns flera metoder för att generera en ljudvåg: additiv syntes, subtraktiv syntes, FM-syntes och så vidare. Valet av DDS som metod för ljudsyntes var främst på grund av dess enkla implementation och anpassningsförmåga i digitala system. DDS är även en välkänd metod inom industrin för tillverkning av funktionsgenerator och digital synthesizer.
En annan fördel är att det är möjligt att snabbt växla mellan olika frekvenser med stor noggrannhet och eftersom det i första hand är en digital teknik, kan en hel del variationer på ljudsignalen skapas med hjälp av den digitala signalbehandlingens (DSP) tekniker. En annan fördel med DDS är att den stöder ett brett spektrum av frekvenser.

4.3.2 DAC

För att omvandla den digitala signalen till analog kan antingen PWM eller DAC användas. DAC var den föredragna metoden på grund av dataöverföringshastigheten då ett 8-bitars värde kan skickas samtidigt genom en port (8 stift) i parallell form medan PWM endast använder ett stift där bitarna skickas en efter en. PWM skulle kräva en separat timer för implementering som skulle lägga till mer bearbetningsbelastning på mikrokontrollern medan med DAC tar det endast att skicka 8-bitars värde till en port. Ytterliggare en fördel är att det alltid finns någon rippel förknippad med PWM i utsignalen även efter användning av ett lågpassfilter, medan filter tar hand om rippel i DAC mycket mer effektivt.

4.3.3 Kristall

Mikrokontrollern använde den interna 8 MHz kristallen som klockkälla som påverkade signal upplösningen enligt ekvation 1; genom att löda en 16 Mhz kvartskristall till mikrokontrollern och ställa in fuse bitarna i mikrokontrollern med hjälp av Atmel studio 7, ändrades klockfrekvensen till 16 MHz extern kristall vilket skulle göra det möjligt för DDS systemet att använda maximal bearbetningshastighet och därmed maximal upplösning för ljudsignalen.

4.3.4 Fasackumolatorn

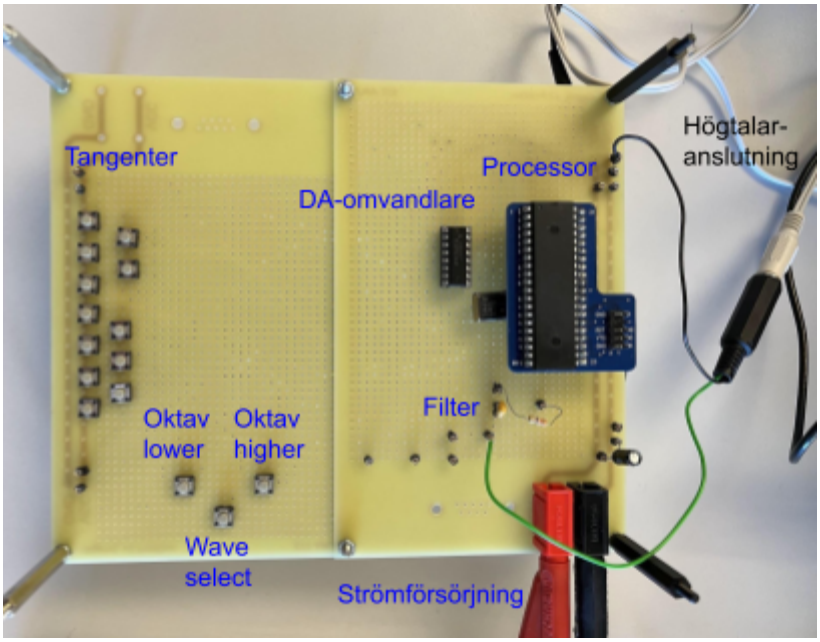
Fasackumulatorn valdes till 32 bitars längd eftersom den skulle ge en mycket fin upplösning för frekvensändringarna, det vill säga för varje steg i tuning word skulle utfrekvensen öka med mängden $f_c / 2^{32}$, vilket är optimalt; med tanke på utfrekvensens intervallet är mellan 60 Hz och 9000 Hz, kommer en 32-bitars fasackumulator att göra det möjligt att få ett mer exakt värde i detta intervall till utfrekvensen.

4.4 Problem och deras lösningar

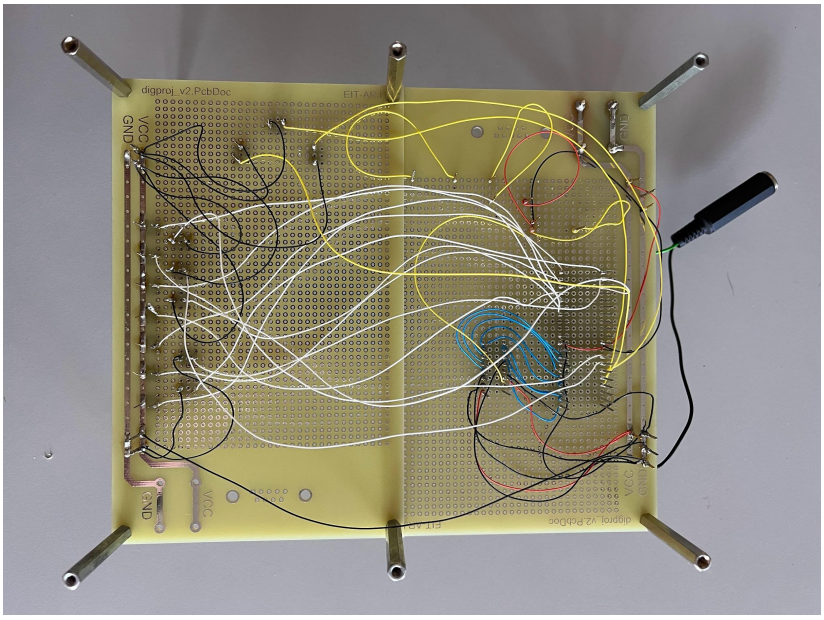
Ett problem som uppstod när vi hade kopplat upp alla knappar var att det blev mycket störning mellan dessa knappar. Störningarna gjorde så att det ibland blev fel toner eller ingen ton alls. Genom att koppla knapparnas jord till en gemensam jordskena kunde vi minska antalet sladdar. Lösningen hjälpte inte enbart mot störningar utan också mot det stora antalet sladdar som behövdes mellan mikrokontrollern och knapparna.

5 Resultat

Den ihopsatta synten ses i figur 5 och 6. Tangenterna kunde spela 60 toner från lägsta 65,4 Hz upp till högsta på 1975,5 Hz. Detta genom att bläddra antingen upp eller ner en oktav med Oktav higher eller Oktav lower. Med Wave select kunde även vågtypen sawtooth, square och sinus bläddras igenom och väljas. En vanlig datorhögtalare AC-691N med extern strömförsörjning och förstärkare kopplades in på högtalaransluntningen och gav ifrån sig tonerna när tangenterna spelades.



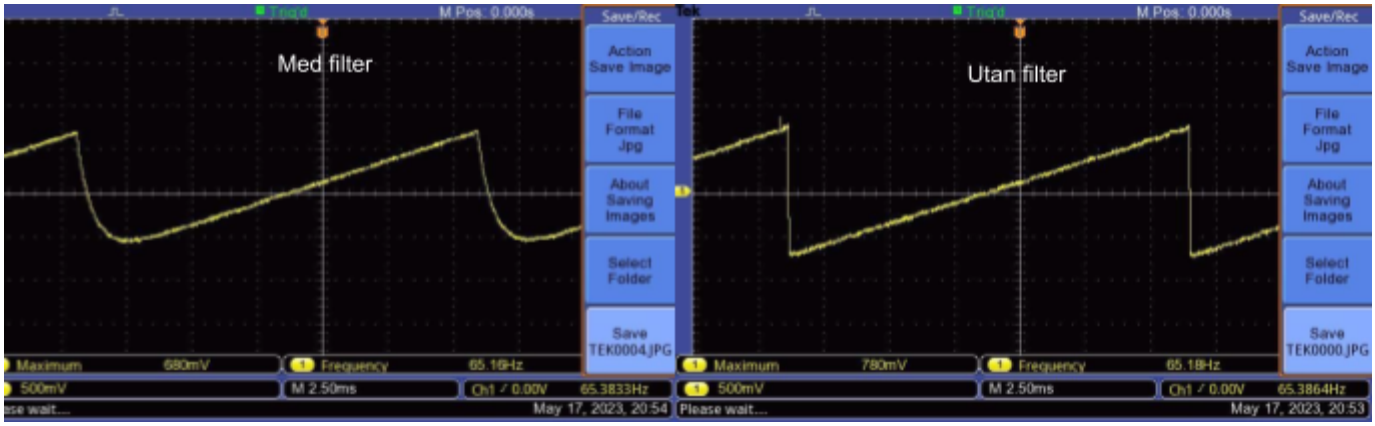
Figur 5: Synten med alla komponenter fästa på perfboard sett från ovan.



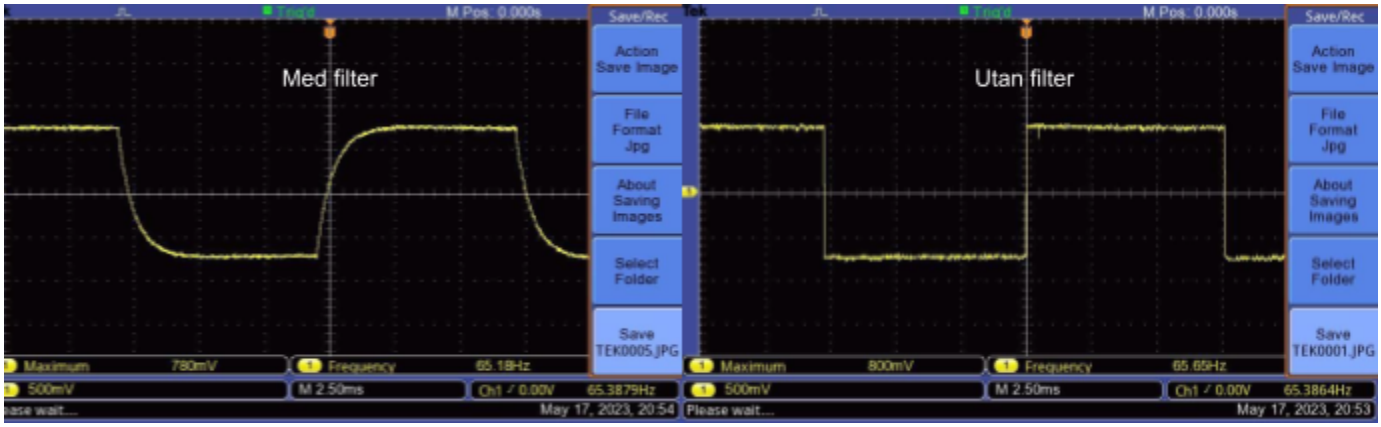
Figur 6: Synten med alla komponenter fästa på perfboard sett underifrån.

Ett oscilloskop kopplades mot högtalares insignal och jord. När sawtooth, square och sinusvågor spelades visades signalerna på dess display enligt figur 7-9 till vänster i bilderna.

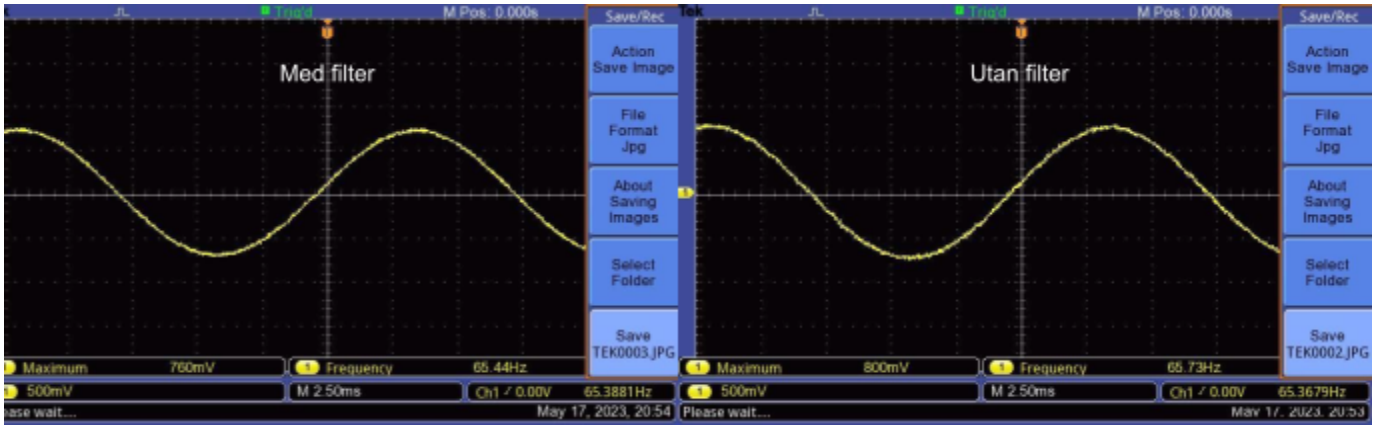
När dessa spelades igen utan kondensatorn, alltså utan filtret, gavs signalerna till höger i figur 7-9.



Figur 7: Oscilloskopets kopplades mot högtalarens insignal och jord vilket gav bilden till vänster när en sawtoothvåg spelades. Till höger spelades samma ton men utan filtret inkopplat.

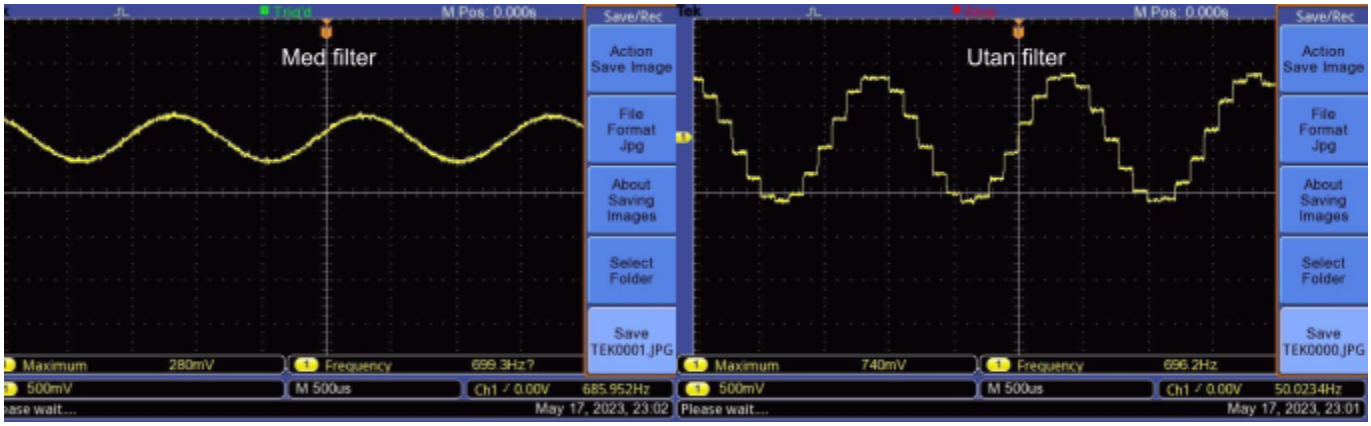


Figur 8: Oscilloskopets kopplades mot högtalarens insignal och jord vilket gav bilden till vänster när en squarevåg spelades. Till höger spelades samma ton men utan filtrert inkopplat.



Figur 9: Oscilloskopets kopplades mot högtalarens insignal och jord vilket gav bilden till vänster när en sinusvåg spelades. Till höger spelades samma ton men utan filtret inkopplat.

En cirka 10 gånger högre frekvens spelades och samma mätning utfördes vilket ses i figur 10.



Figur 10: Oscilloskopets kopplades mot högtalarens insignal och jord vilket gav bilden till vänster när en sinusvåg spelades. Till höger spelades samma ton men utan filtret inkopplat.

6 Slutsats

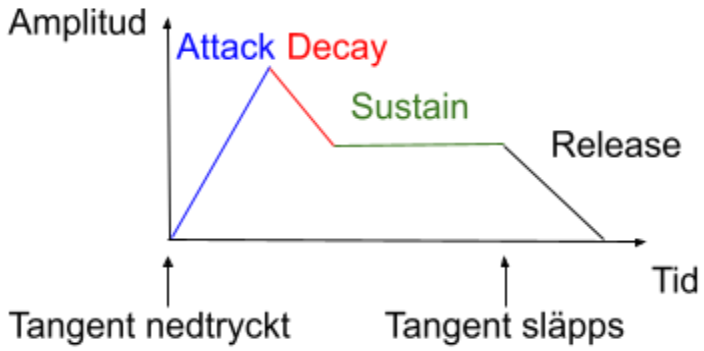
Med ett fåtal komponenter och en ATmega kunde en hyfsat bra synt byggas med DDS. Ljudet blev över förväntan och målet med projektet ansåg uppnått. Detta så pass att även den känsligaste av kulturutövare kan använda den med tillfredsställande resultat.

6.2 Framtida utvecklingsmöjligheter

En synt kan ha oändligt många funktioner för att manipulera och styra ljudet så som musikern önskar. Men i detta arbete gjorde tidsbegränsningen att det var tillräckligt med att kunna spela 12 toner för en kromatisk skala, byta mellan olika oktaver och även variera mellan 3 vågformer för ljudet.
Nedan följer exemplen på ytterligare några funktioner som synten hade kunnat få vilket kan vara ett framtida utvecklingsarbete för den som har lust.

6.2.1 ADSR

När en ton spelas behöver dess amplitud inte gå från en nollnivå direkt upp till maxamplituden vilket kan låta som ett skarpt avbrott. Istället kan man använda sig av ADSR vilket står för Attack Decay Sustain Release som visas i figur 11.



Figur 11: Graf över en ADSR formad ton från att dess tangent har tryckts ner till att den släpps och dör ut.

När en tangent trycks ner stegras tonens amplitud i Attack för att sedan minska i Decay och landa i Sustain vilket ger en mjukare start av tonen som sedan hålls. När tangent sedan släpps går tonen in i release och landar på nollnivån istället för att abrupt avbrytas. Genom att ändra hur lång tid dessa faser är och deras amplitudnivåer kan tonen formas. Sen är ADSR bara ett exempel på hur en ton kan formas och det finns många andra sätt som detta kan utföras.

6.2.2 Skärm

När en ton spelas måste värdet för frekvensen och vågformen avläsas i koden eller ett oscilloskop kopplas till utsignalen för att se vad som spelas. Det hade underlättat att med en lämplig skärm direkt visa detta genom att skriva värdet av fout, dacVal och kanske lägga till lite grafiska effekter.

6.2.3 Polyfonisk

Synten är begränsad att spela en ton åt gången vilket gör den monofonisk. Genom att förändra koden så att den kunde spela flera toner när flera tangenter är nedtryckta samtidigt hade den också kunnat spela ackord och varit polyfonisk.
Detta kan till exempel göras genom att ha två look-up table och två fasackumulatorer som bearbetar parallellt, och de resulterande värdena från varje look-up table skulle adderas och skickas till DAC för att generera ljudet av två toner som spelas tillsammans. För att spela fler toner tillsammans krävs naturligtvis fler loop-up-tabeller och mer bearbetningsarbete för mikrokontrollern.

8 Källförteckning

Murphy, Eva och Slattery, Colm. 2004. Ask The Application Engineer—33: All About Direct Digital Synthesis, *Analog dialogue*.
https://www.analog.com/en/analog-dialogue/articles/all-about-direct-digital-synthesis.html
(Hämtad 2023-04-29).

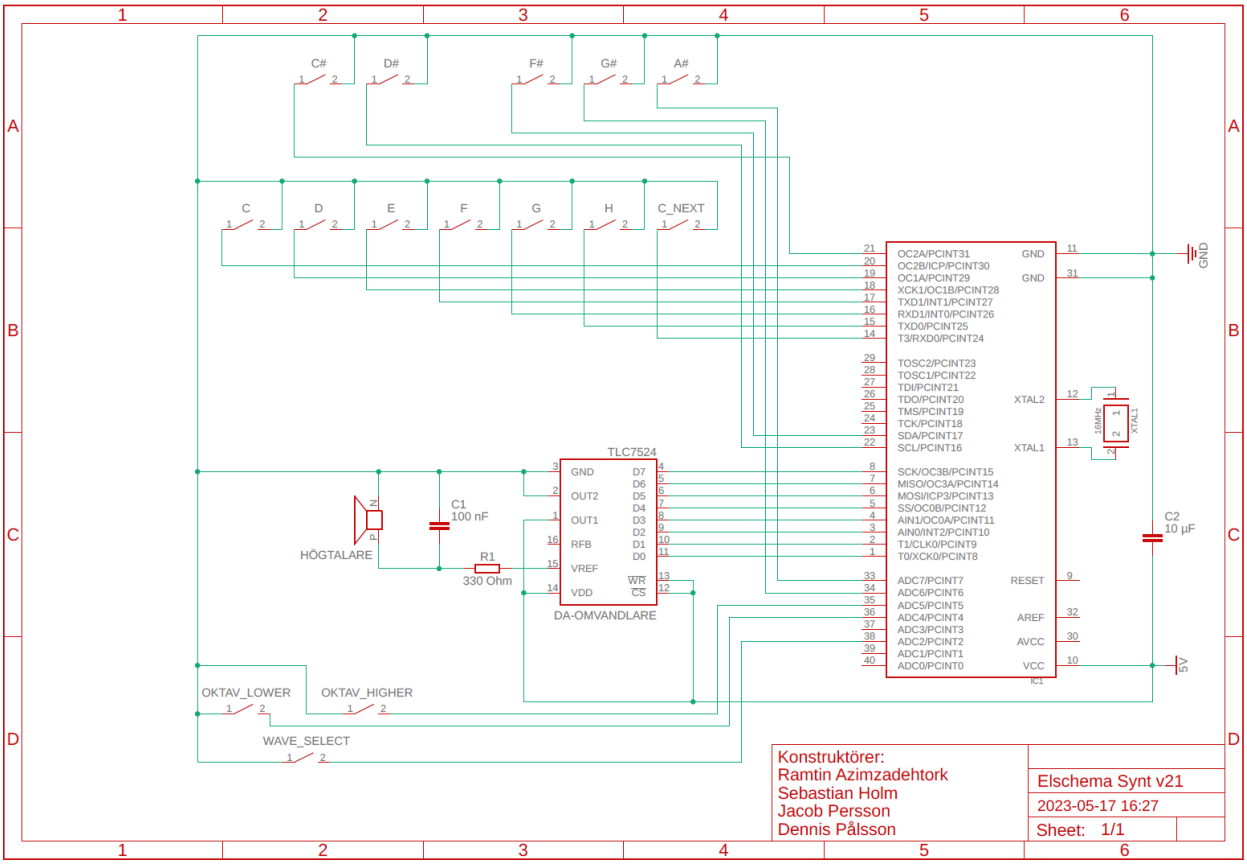
Datablad - Microcontroller ATmega1284
https://www.eit.lth.se/fileadmin/eit/courses/datablad/Processors/ATmega1284.pdf
(Hämtad 2023-04-29).

Datablad - 8-bitars D/A-omvandlare TLC7524
https://www.eit.lth.se/fileadmin/eit/courses/datablad/Analog/adda/tlc7524.pdf
(Hämtad 2023-04-29).

Video som används för att studera om DDS och koden som inspirerade projektet
https://www.youtube.com/watch?v=WwHouxujrNc
https://github.com/GadgetReboot/DDS_Generator_For_Uno
(Hämtad 2023-04-29).

9 Appendix

9.1 Kretsschema



9.2 C kod

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <math.h>
#include <avr/delay.h>


uint32_t phAcc = 0;             // initializing the 32 bit phase accumulator
uint8_t dacVal = 0;             // initializing the 8 bit data to send to DAC
```

```
float fOut = 1000;                  //initializing the target frequency to generate in Hz
volatile uint32_t tuningWord = 0;       // DDS tuning word for target frequency



//base frequency of 12 notes in the octave
float not_C = 65.40639;
float not_Css = 69.29566;
float not_D = 73.41619;
float not_Dss = 77.78175;
float not_E = 82.40689;
float not_F = 87.30706;
float not_Fss = 92.49861;
float not_G = 97.99886;
float not_Gss = 103.8262;
float not_A = 110.0000;
float not_Ass = 116.5409;
float not_B = 123.4708;

uint8_t octav_Changer = 0;                      //this is the value that goas between 0 and 5
whenever the octave hgih or low
                                                                    //buttons are pressed

uint8_t oct = 0;                                        //oct = 2 ^ octav_Changer, which later
multiplies with the base frequency to generate
                                                                    //the appropriate note in the
desired octave

uint8_t octav_Higher_WasPressed = 0;        //variable to check if the octave increasing button
has been pressed
uint8_t octav_Lower_WasPressed = 0;                 //variable to check if the octave decreasing
button has been pressed
uint8_t wave_select_WasPressed = 0;                 //variable to check if the wave changing
button has been pressed

#define DEBOUNCE_TIME 3000 // debounce time

int wave_Select = 0;   //variable to determine the wave from:sine-triangular-square
```

```
//lookup table: consists of 256 sample values of 8 bit sine wave (each value is 8 bit, or 0 to 255 decimal)
uint8_t LUT[] = {128, 131, 134, 137, 140, 143, 146, 149, 152, 155, 158, 162, 165, 167, 170, 173, 176, 179, 182, 185, 188, 190, 193, 196, 198, 201, 203, 206, 208, 211, 213, 215,
   218, 220, 222, 224, 226, 228, 230, 232, 234, 235, 237, 238, 240, 241, 243, 244, 245, 246, 248, 249, 250, 250, 251, 252, 253, 253, 254, 254, 254, 255, 255, 255,
   255, 255, 255, 255, 254, 254, 254, 253, 253, 252, 251, 250, 250, 249, 248, 246, 245, 244, 243, 241, 240, 238, 237, 235, 234, 232, 230, 228, 226, 224, 222, 220,
   218, 215, 213, 211, 208, 206, 203, 201, 198, 196, 193, 190, 188, 185, 182, 179, 176, 173, 170, 167, 165, 162, 158, 155, 152, 149, 146, 143, 140, 137, 134, 131,
   128, 124, 121, 118, 115, 112, 109, 106, 103, 100, 97, 93, 90, 88, 85, 82, 79, 76, 73, 70, 67, 65, 62, 59, 57, 54, 52, 49, 47, 44, 42, 40,
   37, 35, 33, 31, 29, 27, 25, 23, 21, 20, 18, 17, 15, 14, 12, 11, 10, 9, 7, 6, 5, 5, 4, 3, 2, 2, 1, 1, 1, 0, 0, 0,
   0, 0, 0, 0, 1, 1, 1, 2, 2, 3, 4, 5, 5, 6, 7, 9, 10, 11, 12, 14, 15, 17, 18, 20, 21, 23, 25, 27, 29, 31, 33, 35,
   37, 40, 42, 44, 47, 49, 52, 54, 57, 59, 62, 65, 67, 70, 73, 76, 79, 82, 85, 88, 90, 93, 97, 100, 103, 106, 109, 112, 115, 118, 121, 124
};



//debounce funtion for the wave selector button
uint8_t waveDebounce (void){
   if(!(PINA & (1<<PINA3))){
         _delay_us(DEBOUNCE_TIME);
         if (!(PINA & (1<<PINA3)))
         {
               return(1);
         }
         return 0;
   }
}



//debounce funtion for the octave increasing button
uint8_t debounceHigher (void){
   if(!(PINA & (1<<PINA5))){
         _delay_us(DEBOUNCE_TIME);
         if (!(PINA & (1<<PINA5)))
         {
               return(1);
         }
```

```
            return 0;
    }
}



//debounce funtion for the octave descreasing button
uint8_t debounceLower (void){
    if(!(PINA & (1<<PINA4))){
            _delay_us(DEBOUNCE_TIME);
            if (!(PINA & (1<<PINA4)))
            {
                    return(1);
            }
            return 0;
    }
}



//main function
int main(void)
{
    //initialize the ports and pins as output/input
    DDRD = 0b00000000;
    PORTD = 0b11111111;
    DDRC = 0b00000000;
    PORTC = 0b00001111;
    DDRA = 0b00000000;
    PORTA = 0b11111000;
    DDRB |= 0xff;

    //set timer1 interrupt for 9060 Hz @ 16 MHz clock, no prescale
    cli();
    TCCR1A = 0;         // clear register
    TCCR1B = 0;         // clear register
    TCNT1  = 0;         // initialize counter value to 0
    OCR1A = 1766;  // 16MHz / 1766  = 9060 Hz
    TCCR1B |= (1 << WGM12) | (1 << CS10);  // turn on CTC mode, Set CS10 for 1 prescaler
    TIMSK1 |= (1 << OCIE1A);            // enable timer compare interrupt
    sei();
```

```
        while (1)
        {

         //if the octave increasing button has been pressed and the octave number is not greater
than 5
         if (debounceHigher()) {
                  if ((octav_Changer < 5) && (octav_Higher_WasPressed == 0)) {
                           octav_Changer++;                        //increase the octave number
                           oct = pow(2,octav_Changer);             //calculate the octave value
                           octav_Higher_WasPressed = 1;
                           }
                  }    else {
                  octav_Higher_WasPressed = 0;
         }


         //if the octave decreasing button has been pressed and the octave number is not lower
than 0
         if (debounceLower()) {
                  if ((octav_Changer > 0) && (octav_Lower_WasPressed == 0)) {
                           octav_Changer--;                        //decrease the octave number
                           oct = pow(2,octav_Changer);     //calculate the octave value
                           octav_Lower_WasPressed = 1;
                           }
                  }    else {
                  octav_Lower_WasPressed = 0;
         }


         //if the waveform changer button has been pressed
         if (waveDebounce()) {
                  if (wave_select_WasPressed == 0) {
                           if (wave_Select == 3){          //if the wavefrom value is 3 lower the value
to -1
                                    wave_Select = -1;
                                    }
                           wave_Select++;        //increments the wavefrom value
                           }
                  wave_select_WasPressed = 1;
                  } else {
                  wave_select_WasPressed = 0;
         }
```

```
   //this if-else sequence checks which button (note) in the keyboard is pressed and the
   //base frequency of that note is assigned to output frequency, before the output frequency
   //is calculated again with respect to the current selected octave
   if (!(PIND & (1<<PIND6))) { // C
         fOut = not_C;
   } else if (!(PIND & (1<<PIND5))){ // D
         fOut =  not_D;
   } else if (!(PIND & (1<<PIND4))) { // E
         fOut =  not_E;
   } else if (!(PIND & (1<<PIND3))) { // F
         fOut = not_F;
   } else if (!(PIND & (1<<PIND2))) { // G
         fOut =     not_G;
   } else if (!(PIND & (1<<PIND1))) { // A
         fOut =     not_A;
   } else if (!(PIND & (1<<PIND0))) { // B
         fOut =  not_B;
   } else if (!(PIND & (1<<PIND7))) { // C#
         fOut =  not_Css;
   } else if (!(PINC & (1<<PINC0))) { // D#
         fOut =  not_Dss;
   } else if (!(PINC & (1<<PINC1))) { // F#
         fOut = not_Fss;
   } else if (!(PINA & (1<<PINA6))) { // G#
         fOut = not_Gss    ;
   } else if (!(PINA & (1<<PINA7))) { // A#
         fOut =     not_Ass;
   } else {
         fOut = 0;
   }
   fOut = fOut * oct;     //the current frequency is multiplide by oct to generate get calibrated into
the current octave
   tuningWord = pow(2, 32) * fOut / 9060.0;   // tuning word is calculated after the output
frequency has been set

         }

}

/**
* Timer1 interrupt 9060 Hz
*   whenever the output compare match gets the value of 1766, the timer ISR is processed.
 **/
ISR(TIMER1_COMPA_vect)
```

```
{
   uint8_t count = (phAcc >> 24);       //get the highest 8 bits of the 32 bit phase accumulator
                                                                                    //as an index for the look up
table.

   //this if-else sequence generates the appropriate waveform according to the wave_select
value
   if (wave_Select == 0) {
         dacVal = count;                    //   ramp wave:  send counter value to the DAC to generate
a rising ramp
   } else if (wave_Select == 1) {
         dacVal = (count > 127) ? 255 : 0;      // square wave:  send all 0 or all 1 to the DAC for
each half of a wave cycle
   } else if (wave_Select == 2) {
         dacVal = LUT[count];                       //   sine wave:  send look up table sample value to
the DAC
   }

   PORTB = dacVal;               //sends the dacVal to the pins in portB which
                                                        //is connected to the DAC
   phAcc += tuningWord;        //increments the phase accumulator
}
```