EDI022

Digitala Projekt

Rapport

Björn Åkesson

5/20/2011

A synthesizer is built consisting of a bit-counter and phased-locked-loop to create various tones. The construction can be controlled via an external MIDI-controller that communicates through the standard MIDI protocol. All frequencies will output on a speaker that is built on the machine. A program is to be written in order to handle the MIDI messages and generate the correct output.

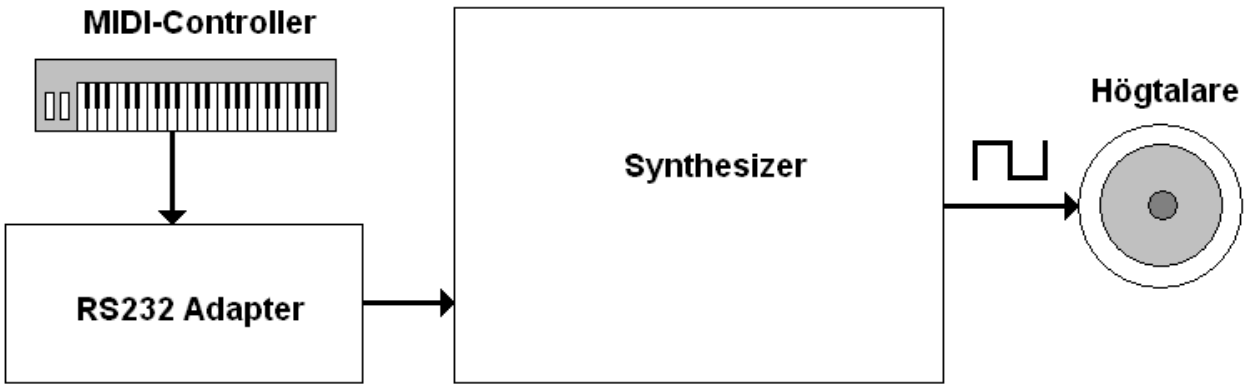
Innehåll

1 Introduktion

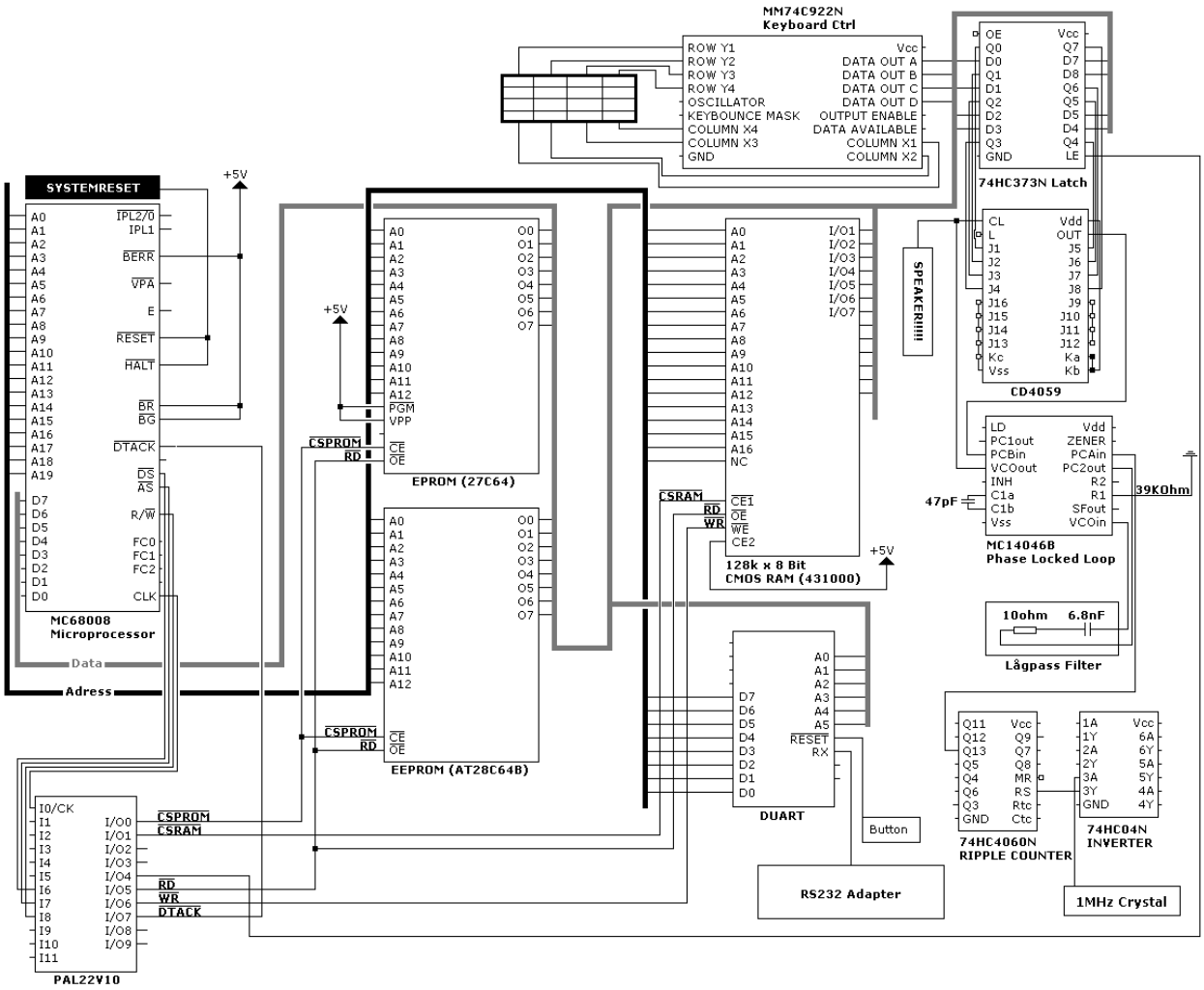
Mitt projekt gick ut på att konstruera en fungerande synthesizer som kan spela upp toner via en högtalare. För att spela upp toner så kan en midiklaviatur kopplas in och styra konstruktionen. För att skapa olika tonlägen så krävs olika frekvenser, dvs. jag har byggt en frekvensgenerator. Det finns dock en del begränsningar, för att kunna skapa musikaliska toner så krävs väldigt specifika och exakta frekvenser. Att få fram dessa frekvenser kommer ta för lång tid och är inte så speciellt relevant för kursen.

2 Krav Specifikation

- Synthen ska kunna spela olika toner som man kan höra från en högtalare i form av en fyrkantsvåg
- Med hjälp av en MIDI-controller ska man kunna kommunicera med konstruktionen och spela noter
- Det ska inte finnas någon märkbar fördröjning från knapptryck till byte av ton
- Konstruktionen ska ha stöd för ”note-on” och ”note-off” vilket innebär att då ingen tangent är nedtryck från MIDI-controller så ska synthen inte ge ifrån sig något ljud.



Figur 1: Enkel överbild på konstruktionen



Figur 2: Kopplingsschema

3 Hårdvara

3.1 Processor

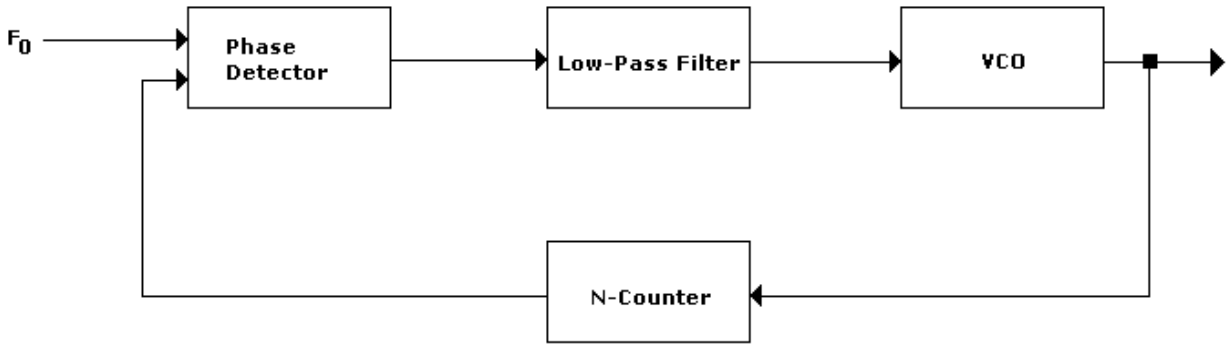
Konstruktionens processor är en M68008 från Motorola med 48 pinnar.

3.2 Phased-Locked-Loop (PLL)

Fasdetektorn (Phase Detector) har två signaler som input, en referensfrekvens som alltid kommer vara konstant, och output signalen från PLL kretsen som är negativt återkopplad. Funktionen hos fasdetektorn är att jämföra de båda signalerna och producera en felsignal som proportionell till fasskillnaden hos insignalerna. Utsignalen förs sedan vidare in till ett lågpassfilter som filtrerar signalen, tar bort störningar och ser till att inga extremt höga frekvenser försöker låsas. Efter filtret åker signalen genom en VCO (Voltage-Controlled Oscillator) som producerar output frekvensen. Denna frekvens återkopplas som tidigare beskrivits till en räknare och sedan tillbaka till fasdetektorn. Om output frekvensen skulle ändras pga. störning så kommer fasdetektorn att läsa av störningen och korrigera den. För att PLL kretsen ska funka för den tänkta konstruktion så måste den ställas in för att möta kraven.
Resistans och kapacitans ansluts till kretsen för att bestämma frekvensområdet som synthen kommer användas i. I detta fall så är den minst tänkbara frekvensen att använda 61Hz, men för att vara på den säkra sidan så kan den lika väl vara nära 0 då det handlar om ett fåtal frekvenser. Den högsta frekvensen som vi kan tänkas uppnå bör inte överstiga 20000Hz eftersom det mänskliga örat inte kan höra högre. Samtidigt är en frekvens nära den gränsen smärtsam och obehaglig att höra på. Vi väljer frekvensen 10000Hz som blir frekvensmaximum och 0Hz som frekvensminimum.
Detaljerna räknas ut med följande formler:

$$f_{min} = \frac{1}{R_2(C_1 + 32\ pF)} \qquad f_{max} = \frac{1}{R_1(C_1 + 32\ pF)} + f_{min}$$

Med R_1 = 1MOhm, C_1 = 68pF och R_2 oändligt får vi fram ett f_{max} på 10000Hz och ett f_{min} på 0Hz.



Figur 3: Schema på huvuddelarna i en PLL (F0 är referensfrekvensen)

Lågpass filtret måste veta vilket frekvensområde som kommer appliceras för att kunna fungera optimalt. Detta ställs in genom att koppla in önskade värden som resistans och kapacitans. Detaljerna räknas ut med följande formel ($F_C = F_L$):

$$2f_C \approx \frac{1}{\pi}\sqrt{\frac{2\pi f_L}{R3\ C2}}$$

Med $F_C = F_L$ = 10KHz, C2 = 6.8nF och R = 10KOhm får vi ett passande filter för PLL kretsen.

3.3 DUART (Dual Channel UART)

En väldigt viktig och central del i konstruktionen är DUART. Jag har använt mig av kretsen 68681CP och dess uppgift är att sköta kommunikationen mellan midi och processor. Data från MIDI kommer i seriell form och måste konverteras till parallell data som processorn kan läsa av. För att kretsen ska kunna kommunicera med MIDI-controllern så krävs det att båda delar använder samma BAUD-rate. Det sökta värdet är 31250Hz och sätts i DUART kretsen genom att skriva till dess interna register genom adressbitarna, förutsatt att en kristal på exakt 3,6864MHz är inkopplad.

3.4 PAL

PAL kretsens uppgift är att se till så att processorn kan adressera rätt enhet med hjälp av adressbitar. Konstruktionen har en PAL av modellen PAL22V10.

3.5 Bit-Counter

Uppgiften för denna krets är att bestämma nya frekvenser, enheten sköter byten av noter genom att dela en frekvens som skickas in. Räknaren är en så kallad "Divide-by-N" som kan dela frekvenser med ett värde N som kan sättas från 3 till 15999. Kretsen heter CD4059 och programmeras via 16 input pinnar som kallas för "Jam inputs".

3.6 RAM

Ett simpelt RAM minne som vi kan spara data i under körningstiden, värt att tänka på är att RAM minnet töms när strömförsörjningen till konstruktionen upphör. Jag har använt mig av ett 128kB minne som heter 431000.

3.7 EEPROM

Här laddas det slutgiltiga programmet in och bränns fast för att aldrig kunna raderas.

3.8 Binary-Ripple-Counter

Målet är att jobba med relativt små frekvenser, pga. att vi använder en kristall på 1MHz så måste den delas ner på något sätt. Med en 4-bitars binary-ripple-counter så kan vi dela vår höga frekvens ner till cirka 60Hz som i detta fall inte är en helt tillfredställande frekvens, men duger helt klart för en prototyp av den tänkta konstruktionen.

3.9 RS-232 Adapter

Signaler från en MIDI-controller överensstämmer inte med signaler som konstruktionen använder, därför måste det ske en konvertering till logiska nivåer som en UART kan läsa. Adaptern består av en optocoupler krets som tar emot signaler från controllern, sedan förs signalen vidare till en MAX232 krets som fungerar som en receiver/driver. Receivern omvandlar en högre spänning till logiska nivåer som konstruktionen kan använda sig av.

3.10 Knappsats med 16 knappar plus Avkodare

För att testa konstruktionen innan MIDI kopplas in, så används en enkel knappsats med 16 knappar där olika noter (frekvenser) kan testas. En enkel knappsats med 4 pinnar på undersidan kopplas till en avkodare som kan tolka vilka knappar som trycks ner. När denna information är klar så skickas värdet vidare till en latch. Latchens uppgift är helt enkelt att hålla kvar det värdet som skickas in, ett värde som skickas vidare till en bit-counter. I den slutgiltiga produkten så är knappsatsen bortkopplad då den är ersatt med en MIDI-controller.

3.11 Övriga komponenter

För att tillgodose enstaka kretsars krav och att få konstruktionens komponenter att jobba tillsammans finns det övriga komponenter. I detta ingår följande:

- Inverterare (74HC04N)
- OR-Grind (74HC32N)
- NAND-Grind (74HC10N)
- RESET-Knapp
- Högtalare

4 MIDI-Protokoll

MIDI signaler skickas som paket, där varje paket består av en sträng bitar. Varje meddelande består av 10 bitar (1 byte data och en start och en stop bit) och ett paket kan bestå av 1-3 olika meddelanden.

Nedanför finns en tabell med intressanta meddelande som är relevanta för projektet. Det finns väldigt många olika signaler för MIDI, men jag anser att det blir för mycket och känns trivialt för en sådan här konstruktion. För ett fullskaligt synthprojekt av kommersiell anda kan det vara intressant, ett sådant projekt hade dock krävt fler personer är en för att genomföra. De olika kanalerna (channel) representerar vilken MIDI-controller man styr från, då protokollet stödjer upp till 16 olika enheter. "Note Off" representerar en tangent som upphör att vara nertryckt och "Note On" betyder att en tangent blivit nedtryckt. "After Touch" ger värde på hur hårt en tangent tryckts på.

1st Byte (Status)	**Function**	**2nd Byte (Data)**	**3rd Byte (Data)**
10000000 = 0x80	Channel1 Note Off	Note Number (0-127)	Note Volume (0-127)
10000001 = 0x81	Channel2 Note Off	Note Number (0-127)	Note Volume (0-127)
10000010 = 0x82	Channel3 Note Off	Note Number (0-127)	Note Volume (0-127)
10000011 = 0x83	Channel4 Note Off	Note Number (0-127)	Note Volume (0-127)
10010000 = 0x90	Channel1 Note On	Note Number (0-127)	Note Volume (0-127)
10010001 = 0x91	Channel2 Note On	Note Number (0-127)	Note Volume (0-127)
10010010 = 0x92	Channel3 Note On	Note Number (0-127)	Note Volume (0-127)
10010011 = 0x93	Channel4 Note On	Note Number (0-127)	Note Volume (0-127)
11010000 = 0x208	Channel1 After Touch	Pressure (0-127)	-
11010001 = 0x209	Channel2 After Touch	Pressure (0-127)	-
11010010 = 0x210	Channel3 After Touch	Pressure (0-127)	-
11010011 = 0x211	Channel4 After Touch	Pressure (0-127)	-
10110000 = 0x176	Channel1 Control/Mode Change	*(Se Tabell 2)*	*(Se Tabell 2)*
10110001 = 0x177	Channel1 Control/Mode Change	*(Se Tabell 2)*	*(Se Tabell 2)*
10110010 = 0x178	Channel1 Control/Mode Change	*(Se Tabell 2)*	*(Se Tabell 2)*
10110011 = 0x179	Channel1 Control/Mode Change	*(Se Tabell 2)*	*(Se Tabell 2)*

Tabell1: Grundläggande funktioner som nedtryckta noter

1st Byte (Status)	**Function**	**2nd Byte (Data)**	**3rd Byte (Data)**
(Se Tabell 1)	Portamento Time	00000101 = 0x05	Value (0-127)
(Se Tabell 1)	Channel Volume	00000111 = 0x07	Value (0-127)
(Se Tabell 1)	Balance	00001000 = 0x08	Value (0-127)
(Se Tabell 1)	Portamento On/Off	01000001 = 0x41	$\leq$ 63 Off, $\geq$ 64 On
(Se Tabell 1)	Portamento Control	01010100 = 0x84	Value (0-127)

Tabell2: Speciella funktioner som Portamento (Glidande toner) och balans (vänster och höger)

5 Slutsats

Hela projektet har varit intressant men tagit allt för lång tid. Trots lång spenderad tid så fungerar inte konstruktionen som den ska. Jag har inte lyckats få PLL kretsen att låsa rätt frekvenser, det som har producerats har varit en låg ton, en hög ton och snabba frekvensskiftningar som resulterat i konstiga läten. MIDI kommunikationen har heller inte blivit slutförd, MIDI signaler från en klaviatur skickas genom en adapter och fram till receive pinnen på DUART som visar att logiska nivåer skickas. Att kunna läsa och spara värden i DUART har inte fungerat. Problemet ligger antingen i initieringen eller läsningen av receive bufferten.

6 Appendix A (PAL Kod)

```
'Koden saknar viktiga enheter som RAM, EPROM. Anledning är att av ett misstag så
raderades den befintliga
'PAL uppsättning som var väl fungerande. Den nuvarande kodningen är tillräcklig
för att kunna registrera input
'från MIDI och ha tillgång till räknaren som byter frekvens.

Title             Synth
Pattern           Minne
Author            Bjorn Akesson
device 22V10

CLK         1
A13         2
A14         3
A15         4
A16         5
A17         6
A18         7
A19         8
DS          9
AS          10
RW          11
GND         12
DTACK       15
DUART       17 '0xA0000
WR          18
RD          19
NCOUNTER    20 '0x80000
VCC         24
start
DUART /= /AS * A19 * /A18 * A17 * /A16 * /A15 * /A14 * /A13;
NCOUNTER /= /AS * A19 * /A18 * /A17 * /A16 * /A15 * /A14 * /A13;
RD /= /DS * RW;
WR /= /DS * /RW;
DTACK /= /DUART + /NCOUNTER;
end
```

7 Appendix B (C-Kod)

```
/*********************************************************
Koden fungerar för användning av knappsats med 16 knappar,
där varje knapp registreras och kan användas
*********************************************************/

unsigned short int butt;
unsigned short int counter;
unsigned short int *output;

void main(void)
{
  init();

  greta:
  /* Wait for interrupt */
  goto greta;
}


init()
{
  output = (unsigned short int *)0x80000;
  mem = (unsigned short int *)0x20000;

  _avben();
  return;
}
```

```
exp5()
{

      butt = *output;
      butt = butt & 0x0F;

      switch (butt){
         case 0x00 : counter = 0x00; break;
         case 0x01 : counter = 0x01; break;
         case 0x02 : counter = 0x02; break;
         case 0x03 : counter = 0x03; break;
         case 0x04 : counter = 0x04; break;
         case 0x05 : counter = 0x05; break;
         case 0x06 : counter = 0x06; break;
         case 0x07 : counter = 0x07; break;
         case 0x08 : counter = 0x08; break;
         case 0x09 : counter = 0x09; break;
         case 0x0A : counter = 0x0A; break;
         case 0x0B : counter = 0x0B; break;
         case 0x0C : counter = 0x0C; break;
         case 0x0D : counter = 0x0D; break;
         case 0x0E : counter = 0x0E; break;
         case 0x0F : counter = 0x0F; break;
         default: counter = 0xBA;  break;
      }
      *output = counter;

      return;
}
```

8 Appendix C (Initialisering av DUART)

Initiering sker via hyperterminalen för att få en kommunikation mellan DUART och COM3 porten.

```
O A0002 1A  //Reset MR1 Pointer and disable Receiver and Transmitter for
            //configuration
O A0002 30  //Reset Transmitter
O A0002 20  //Reset Receiver
O A0000 13  //Use 8 bits with no parity
O A0001 BB  //Set Baud Rate to 9600 for testing of communication with COM3 port
O A0002 05  //Enable Transmitter and Receiver after configuration is done
```