[X] Wettbewerb JUGEND FORSCHT [ ] SCHÜLER EXPERIMENTIEREN

[ ] Biologie
[ ] Chemie
[ ] Geo- und Raumwissenschaften
[ ] Mathematik/Informatik
[ ] Physik
[X] Technik
[ ] Arbeitswelt

THEMA: DIGITALER SYNTHESIZER II

[X] Einzelarbeit [ ] Gruppenarbeit (... Teilnehmer)

Name: Purnhagen Vorname: Heiko
(bei Gruppen hier Sprecher)

Anschrift: Elsasser Str. 64, 2800 Bremen

Geb.Datum: 2. 4. 1969 Telefon: 0421/3499655

Schule: Schulzentrum des Sek. II Horn - Gymnasium -

Schulanschrift: Vorkampsweg 97, 2800 Bremen 33

Schulabschluß angestrebter/~~erzielter~~: x) Abitur

Name des betreuenden Lehrers: ./.

Firma:

Firmenanschrift:

Name des Ausbildungsleiters:

Bei Gruppenarbeiten weitere Namen und Geburtsdaten:

1. Geb.Datum:

2. Geb.Datum:

Frühere Teilnahme an JUGEND FORSCHT / ~~SCHÜLER EXPERIMENTIEREN~~ x):

Jahr 19 85 und weitere Fachgebiet: Technik Preis: 4. Pl. Bundesw.

Ich versichere wahrheitsgemäß, daß ich alle Hilfen und benutzte Literatur angegeben habe.

Heiko Purnhagen
Unterschrift

Bremen, den 19.2.1987
Ort/Datum

x) Nichtzutreffendes bitte streichen

K U R Z F A S S U N G

(Rückseite kann auch beschrieben werden)

Bundesland: Bremen

Name/Anschrift/Telefon

Einzel bzw. Gruppensprecher
Heiko Purnhagen
Elsasser Str. 64
2800 Bremen
Tel. 0421/3499655

2. Gruppenmitglied
./.

3. Gruppenmitglied
./.

Alter: 17 Jahre

Schule/Betrieb/Anschrift

Schulzentrum des Sekundarbereichs II Horn - Gymnasium -
Vorkampsweg 97, 2800 Bremen 33

Fachgebiet und Thema: T E C H N I K

DIGITALER SYNTHESIZER II

In meiner Arbeit beschreibe ich die Entwicklung und Funktionsweise eines digitalen Synthesizers. Er ist die Weiterentwicklung des Synthesizers, den ich im Rahmen des Wettbewerbes "Jugend forscht" 1986 vorgestellt habe. Mein Synthesizer ist nicht nur in der Lage, Töne künstlich zu erzeugen, sondern auch, eingegebene Klänge zu verändern, zu verfremden, oder andere Effekte auf sie anzuwenden.

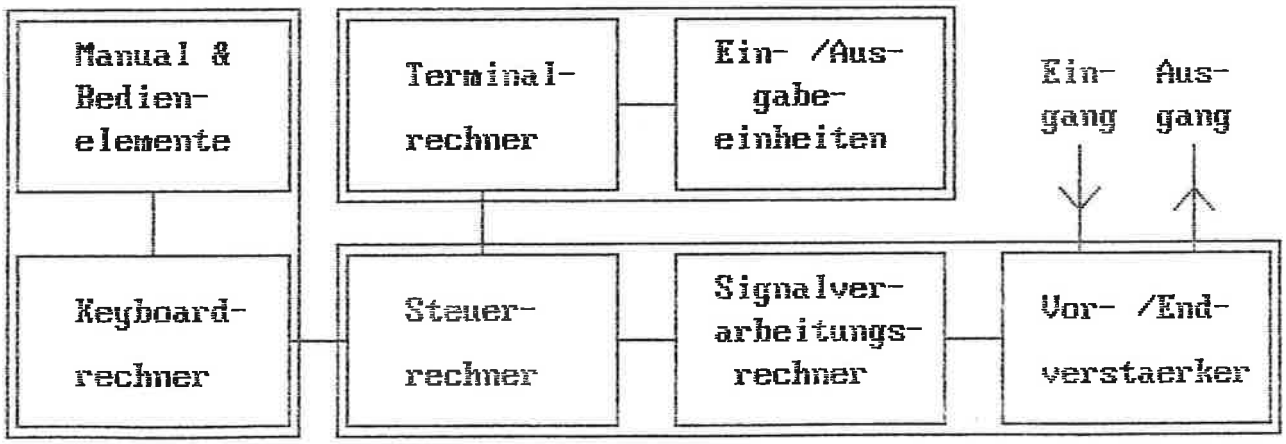
Meine grundlegende Idee besteht darin, das System möglichst vollständig digital aufzubauen. Dabei werden mehrere leistungsfähige Rechner - Mikroprozessoren und digitale Signalverarbeitungsprozessoren - verwendet, die zusammen ein spezialisiertes Multiprozessorsystem bilden. Als Analogschaltungen werden dann neben den A/D- und D/A-Wandlern nur die dazugehörigen Tiefpässe und Verstärkerstufen benötigt. Man kann dabei mit hochauflösenden Wandlern Signale in Studioqualität erzeugen und verändern oder bei einer digitalen Aufnahme die Signale ohne den Umweg über Analogstufen direkt aufnehmen und ggf. direkt auf einem digitalen Tonträger speichern.

In meinem digitalen Synthesizer wird die Ausgangsspannung als Funktion der Zeit 30 000 mal in der Sekunde digital berechnet und mit einem D/A-Wandler in eine Spannung umgewandelt. Sein treppenförmiges Ausgangssignal wird durch einen Tiefpass mit hoher Flankensteilheit geglättet. Das größte Problem ist nun, die Ausgangsspannung mit einer so hohen Rate und einer Genauigkeit von möglichst 16 Bit in Echtzeit zu berechnen. Hierfür benötigte ich einen Digitalrechner, der alle notwendigen Berechnungen für eine Ausgangsspannung in etwa 30 us ausführen kann. Solche Rechenleistungen werden von sogenannten digitalen Signalprozessoren (DSP) erfüllt. Sie werden normalerweise zur Spracherkennung und -synthese, zur digitalen Filterung oder auch Erzeugung von Signalen bzw. für Steuerungszwecke verwendet. Für meine Zwecke schien mir dabei der Signalprozessor TMS 32010 an geeignetsten. Mit unterschiedlichen Programmen für diesen Signalprozessor kann mein Synthesizer verschiedene Syntheseverfahren (z.B.: FM- und PM-Synthese) benutzen und ist dadurch sehr vielseitig.

Zusätzlich zu dem schnellen Signalverarbeitungsrechner sind noch drei weitere Mikrocomputer notwendig. Einer dient zur Berechnung der Eingangsparameter für den Signalprozessor. Er bestimmt so die Klänge und ihre Verläufe. Der Mikrocomputer im Keyboard fragt das 3 1/2 oktavige Manual und weitere Bedienungselemente ab. Das Manual wir dabei 16-stimmig polyphon ausgewertet und ist anschlagsdynamisch. Der letzte Mikrocomputer dient dem Benutzerdialog und wird als Massenspeicher oder als Sequencer benutzt.

Man kann also meinen Synthesizer als spezialisiertes Multiprozessorsystem bezeichnen, bei dem jeder der vier Prozessoren sein eigenes festes Aufgabengebiet hat. Der wichtigste Vorteil meines Systems besteht darin, daß ich nicht von Anfang an auf ein bestimmtes Syntheseverfahren - zum Beispiel FM-Synthese - festgelegt bin, sondern mit nahezu allen möglichen digitalen Syntheseverfahren arbeiten kann und dadurch sehr flexibel bin.

Blockschaltbild des digitalen Synthesizers



D I G I T A L E R S Y N T H E S I Z E R II

Ein Beitrag zum Wettbewerb " JUGEND FORSCHT " 1987

Gliederung

von Heiko Purnhagen, Bremen, Elsasser Str. 64
(geb. 2.4.1969) Schüler des SZ Sek II Horn -Gymn.-, Bremen

1. Einleitung

In dieser Arbeit beschreibe ich die Entwicklung und Funktionsweise meines digitalen Synthesizers. Er ist die Weiterentwicklung des Synthesizers, den ich im Rahmen des Wettbewerbes "Jugend forscht" 1986 vorgestellt habe. Mein Synthesizer ist nicht nur in der Lage, Töne künstlich zu erzeugen, sondern auch, eingegebene Klänge zu verändern, zu verfremden, oder andere Effekte auf sie anzuwenden.

Meine grundlegende Idee besteht darin, das System möglichst vollständig digital aufzubauen. Dabei werden mehrere leistungsfähige Rechner - Mikroprozessoren und digitale Signalverarbeitungsprozessoren - benötigt, die zusammen ein spezialisiertes Multiprozessorsystem bilden. Als Analogschaltungen werden dann neben den A/D- und D/A-Wandlern nur die dazugehörigen Tiefpässe und Verstärkerstufen benötigt. Man kann dabei mit hochauflösenden Wandlern Signale in Studioqualität erzeugen und verändern oder bei einer digitalen Aufnahme die Signale ohne den Umweg über Analogstufen direkt aufnehmen und ggf. direkt auf einem digitalen Tonträger speichern.

2. Funktionsprinzipien von Synthesizern

Die ersten Synthesizer arbeiteten nur mit analogen Modulen - sie waren eine Art spezialisierter Analogrechner. Die einzelnen Module werden von (analogen) Spannungen gesteuert und können untereinander mit Kabeln oder Kreuzschienenverteilern verbunden werden bzw. sind fest miteinander verbunden. Die Module lassen sich dabei zwei Gruppen zuordnen.

Module zur Klangerzeugung:
VCO - voltage controlled oscillator - ein spannungsgesteuerter (Frequenz) Oszillator mit verschiedenen Kurvenformen
VCF - voltage controlled filter - ein spannungsgesteuerter (Eckfrequenz) Filter (meist Tiefpass mit 12 - 24 dB/Oktave)
VCA - voltage controlled amplifier - ein spannungsgesteuerter (Verstärkung) Verstärker

Module zur Steuerung der Klangverläufe:
ENV - envelope generator - ein vom Keyboard gesteuerter Hüllkurvengenerator
LFO - low frequency oscillator - ein langsamer Oszillator für Vibrato und ähnliche Effekte

Zusätzlich gibt es natürlich auch Mischer, mit denen man mehrere Signale mixen kann.
Das Keyboard liefert dabei ein Key-Signal, das angibt, ob eine Taste gedrückt ist oder nicht, und eine Spannung, die mit meistens 1 V / Oktave die Tonhöhe angibt. So ist nur monophones Spiel möglich, also nur ein Ton zur Zeit. Will man dagegen polyphon spielen, benötigt man ein Keyboard mit mehreren solcher Ausgänge. An jeden dieser Ausgänge muß man dann wieder einen eigenen monophonen Tonerzeugungsblock anschließen.
Verschiedene Klangfarben lassen sich hier im wesentlichen dadurch realisieren, daß man ein obertonreiches Signal (z.B. einen Sägezahn) erzeugt und dann einen Teil der Oberwellen wieder wegfiltert. Man nennt dieses auch subtraktive Synthese.
Nachteil der analogen Synthesizer ist, daß große Mengen äußerst präziser analoger Bauelemente und ICs verwendet werden müssen, wobei auch das Temperaturverhalten beachtet werden muß. Besonders kritisch sind hier die VCO's.

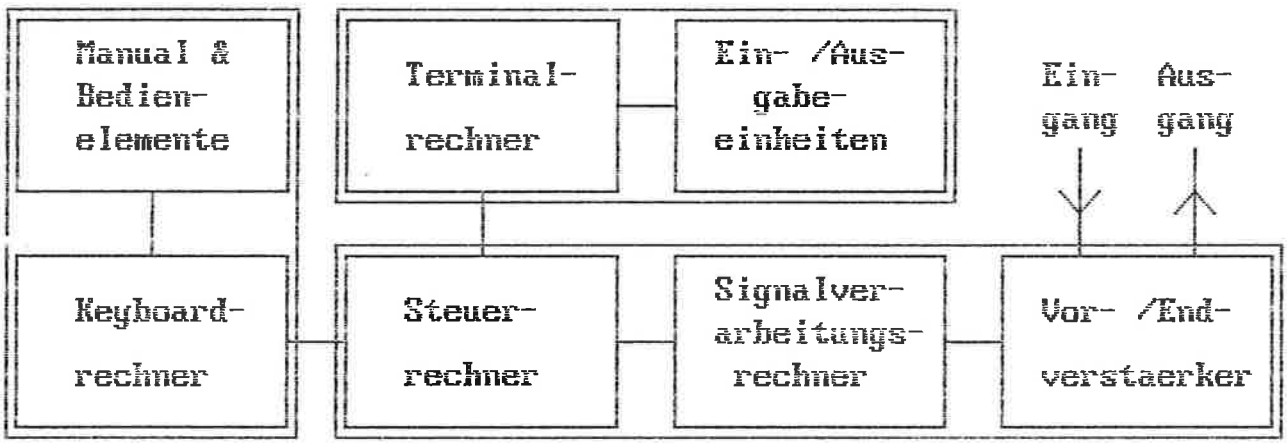
Seit mehreren Jahren gibt es auch Synthesizer auf dem Markt, die teilweise oder vollständig digital arbeiten. So hat man zunächst das polyphone Auswerten des Keyboards und das Ansteuern der analogen Module Computern überlassen und kann so zusätzlich die eingestellten Klangparameter speichern und Klänge schnell reproduzieren. Neuerdings werden statt der VCOs auch DCOs (digital controlled oscillator) verwendet, die digital auch komplexere Ausgangsschwingungen erzeugen können, indem sie ein ROM periodisch auf einen DA-Wandler auslesen.
Vollständig digitale Synthesizer arbeiten nach dem Prinzip der additiven Synthese. Man geht dabei von Sinuschwingungen aus und erzeugt nun Oberwellen durch verschiedene Modulationsarten. So verwenden zum Beispiel die Synthesizer der DX-Serie von Yamaha die Frequenzmodulation (FM). Dabei können sich bis zu sechs digitale Sinusoszillatoren gegenseitig frequenzmodulieren. In der gleichen Weise kann man auch die Phasenmodulation (PM) verwenden. Man kann auch natürliche Klänge digital speichern und auf Tastendruck mit der jeweiligen Frequenz wiedergeben. Dieses Verfahren nennt man "Sound Sampling".

3. Aufbau und Funktionsweise meines Synthesizers

Mit meiner Arbeit habe ich nun versucht, einen vollständig digitalen Sythesizer zu bauen. Dabei bin ich auf einige, teilweise sehr große, Probleme gestoßen, die ich, zusammen mit den jeweiligen Lösungsmöglichkeiten, an der betreffenden Stelle aufzeigen möchte.
Prinzipiell wird die Ausgangsspannung am Synthesizer als Funktion der Zeit digital berechnet. Diese wird dann mit einem D/A-Wandler in eine Spannung umgewandelt. Sein treppenförmiges Ausgangssignal wird durch einen Tiefpass mit hoher Flankensteilheit geglättet. Die berechnete Zahlenfolge wird so in die analoge Ausgangsspannung des Synthesizers ungewandelt.
Das größte Problem ist nun, die Ausgangsspannung mit einer Genauigkeit von möglichst 16 Bit in Echtzeit zu berechnen. Hierfür benötige ich einen Digitalrechner, der die notwendigen Berechnungen für einen Ausgangsspannungswert in etwa 30us ausführen kann. Dieses entspricht einer Wandelrate von 30 KHz. Solche Rechenleistungen werden von sogenannten digitalen Signalprozessoren (DSP) erfüllt. Sie werden normalerweise zur Spracherkennung und -synthese, zur digitalen Filterung oder auch Erzeugung von Signalen bzw. für Steuerungszwecke verwendet. Für meine Zwecke schien mir dabei der Signalprozessor TMS 32010 an geeignetsten.
Deshalb habe ich mit ihm meinen Signalverarbeitungsrechner aufgebaut und als Klanggenerator programmiert. So simuliert er zum Beispiel VCO's, VCA's, Rauschgeneratoren und Mischer. Man kann auf ihm aber auch Programme zur FM- oder PM-Synthese, zum "Sound Sampling" oder zur Erzeugung von Klangeffekten ablaufen lassen.
Die sich langsamer ändernden Steuerparameter für die im Signalverarbeitungsrechner erzeugten Klänge werden dagegen auf einem anderen Rechner berechnet, da der Signalverarbeitungsrechner mit der eigentlichen Tonerzeugung schon voll ausgelastet ist. Dieser Steuerrechner besitzt eine Z80 CPU und arbeitet mit 6 MHz Taktfrequenz. Seine Rechenleistung genügt, um mit etwa 100 Hz Durchlauffrequenz mehrere (niederfrequente) LFO's, ENV's, VCA's und Mischer zu simulieren. Die Rate von nur 100 neuen Werten pro Sekunde, mit der die Parameter für die Oszillatoren berechnet und an den TMS 32010 übergeben werden, ruft, bei Verwendung eines Tricks (siehe 4.3.), keine hörbaren Störgeräusche hervor.

Der Steuerrechner selber benötigt nun Informationen darüber, welche Tasten auf dem Keyboard gedrückt sind. Auch muß der Synthi-Spieler Parameter während des Spiels verändern können (z.B. Lautstärke, Modulationsstärke und Klangfarbe). Diese Informationen werden von einem dritten Rechner ermittelt, der das Keyboard und mehrere Regler abfragt und diese Daten dann an den Steuerrechner weitergibt. Dieser Keyboardrechner verwendet ebenfalls eine Z80 CPU mit 6 MHz Taktfrequenz und ist im Gehäuse des Keyboards untergebracht. Er ist über eine schnelle parallele Schnittstelle mit dem Steuerrechner verbunden.
Für den Dialog mit dem Benutzer beim Programmieren der Klänge und zum Speichern der Klangdaten auf Massenspeichermedien (Disketten) benötige ich noch einen vierten Rechner. Diesen Terminalrechner kann man auch als Sequencer programmieren und dann Tonfolgen oder Melodien automatisch spielen. Zur Zeit verwende ich hierfür ein CP/M-System mit einer 8085 CPU und 3 MHz Taktfrequenz. Der Terminalrechner könnte später zusammen mit einem Diskettenlaufwerk auch noch im 19-Zoll-Gehäuse des Signalverarbeitungs- und Steuerrechners untergebracht werden.
Man kann also meinen Synthesizer als spezialisiertes Multiprozessorsystem bezeichnen, in dem jeder der vier Prozessoren sein eigenes festes Aufgabengebiet hat.

Blockschaltbild des digitalen Synthesizers



4. Der Signalverarbeitungsrechner

Im folgenden beschreibe ich den Rechner, den ich mit dem Signalprozessor TMS 32010 aufgebaut habe.

4.1. Der Signalprozessor TMS 32010

Von den mir bekannten Signalprozessoren ist der TMS 32010 von Texas Instruments für meine Anwendung am besten geeignet, da er über die notwendige Rechenleistung verfügt, einfach zu programmieren ist und man an ihn direkt 4K Word RAM und bis zu 8 Ein- und Ausgabekanäle anschließen kann. Es können dabei sowohl Programme als auch Daten im RAM abgelegt werden, auf das er in 200 ns zugreifen kann. Sein internes Daten-RAM hat eine Kapazität von 144 Words. Er kann so, mit maximal 20 MHz getaktet, 5 MIPS (million instructions per second = Millionen Befehle pro Sekunde) ausführen.

Der TMS 32010 besitzt einen Hardware-Multiplizierer, der in 200 ns, also innerhalb eines Befehls, eine vorzeichenbehaftetete 16*16-Bit-Multiplikation ausführen kann und einen 32 Bit breiten Akkumulator und eine ebenso breite ALU. Er hat so beim Multiplizieren, einer bei der Signalverarbeitung sehr wichtigen Operation, mehr als die 1000-fache Rechengeschwindigkeit eines normalen 8-Bit Mikroprozessors (z.B. 8085 / 3 MHz). Einen Multiplikations-Akkumulier-Zyklus, die Hauptoperation eines digitalen Filters, kann er in normalerweise 400 ns ausführen.

4.2. Hardware

Der Signalverarbeitungsrechner ist auf mehreren Euro-Karten (100 * 160 mm) aufgebaut, die mit einer Busplatine untereinander verbunden sind. Die Karten habe ich in Fädeltechnik aufgebaut, da man so auch bei einer hohen Bestückungsdichte (die Prozessor-Platine besitzt 22 ICs) eine Karte in relativ kurzer Zeit verdrahten kann. Es ist dann auch sehr einfach möglich, Fehler zu verbessern und Veränderungen vorzunehmen.
Auf der Busplatine liegen neben den Versorgungsspannungen von 5 V, +15 V und -15 V und Masse der 16 Bit breite Datenbus, 16 Selectleitungen für die je 8 Ein- und Ausgabeports und die Steuersignale Reset, Clockout, INT und BIO. So kann man neben der Prozessor-Platine je nach Bedarf eine oder mehrere Ein- und Ausgabe-Platinen auf den Bus stecken. Im Augenblick sind dieses eine Wandler-Platine, eine Interface-Platine und eine RAM-Platine mit 32 K Word RAM (Erweiterung auf 128 K Word RAM geplant). Für die Stromversorgung verwende ich ein 50W-Schaltnetzteil, das die drei benötigten Spannungen liefert. Es befindet sich im 19-Zoll-Gehäuse und versorgt auch den Steuerrechner.

4.2.1. Die Prozessor-Platine

Auf der Prozessor-Platine habe ich neben dem Signalprozessors TMS 32010 auch das von ihm benötigte RAM und ein Interface untergebracht, mit dem ich den TMS 32010 steuern und auf sein RAM vom Steuerrechner aus zugreifen kann, um so Programme und Daten laden und kontrollieren zu können.
Da ich den TMS 32010 mit seiner maximalen Taktfrequenz von 20 MHz takte, benötige ich schnelle RAMs mit einer Zugriffszeit von weniger als 70 ns. Ich verwende vier statische CMOS-RAMs HM6168HP-55, die zu je 4K mal 4 Bit organisiert sind und eine Zugriffszeit von 55 ns haben. Diese RAMs kann man direkt an den TMS 32010 anschließen und erhält so einen Speicher mit der vom TMS maximal adressierbaren Kapazität vom 4 K Word.
Zur Erzeugung der Chip-Select-Signale für die maximal 8 Ein- und Ausgabeeinheiten auf dem Bus verwende ich zwei 1 aus 8 Dekoder-ICs. Ich führe auch Reset und Clockout und die beiden Eingänge INT und BIO auf den Bus. Das Signal am BIO-Eingang kann vom Signalprozessor über einen speziellen Befehl abgefragt und ausgewertet werden (siehe 4.2.3.).
Das Interface, mit dem der Steuerrechner auf das RAM des TMS zugreifen und das Reset-Signal für den TMS erzeugen kann, habe ich auch auf der Prozessor-Platine untergebracht. Es wird mit dem Steuerrechner über ein 16-poliges Flachbandkabel verbunden, auf dem ein Teil der Signale des Busses des Steuerrechners liegen. Der aktuelle Pegel des Reset-Signales wird mit zwei Leuchtdioden auf der Vorderseite der Platine angezeigt.

4.2.2. Die Wandler-Platine

Die Wandler-Platine dient dazu, den vom Signalprozessor berechneten digitalen Ausgangsspannungswert in eine analoge Spannung zu wandeln. Zusätzlich kann eine analoge Spannung in einen digitalen Wert umgewandelt werden. Neben den dazu notwendigen D/A- und A/D-Wandlern befinden sich auf dieser Platine auch Tiefpässe. Sie werden dazu benötigt, die Ein- und Ausgangssignale auf eine Bandbreite von der halben Abtastfrequenz zu begrenzen. Dieses ist aufgrund des Shannonschen Abtast-Theorems notwendig.
Auf meiner jetzigen Platine benutze ich den 12-Bit D/A-Wandler DAC800P und den 8-Bit A/D-Wandler ZN427 mit einer Wandelzeit von 10 us. Für die beiden Filter verwende ich Butterworth-Tiefpässe 3. Ordnung (Flankensteilheit 18 db / Oktave) mit einer Eckfrequenz von 15 KHz. Neben den Ein- und Ausgangsverstärkern benötige ich noch eine Sample-and-Hold-Schaltung vor dem A/D-Wandler, damit sich dessen Eingangsspannung nicht während einer Wandlung ändert. Mit dieser Platine kann man ein Signal mit einem Stör- zu Nutzsignalverhältnis (S/N) von bis zu 72 db erzeugen. Angesprochen wird die jetzige Wandler-Platine über je ein Ein- und Ausgabetor des TMS.

Momentan entwickele ich eine verbesserte Wandler-Platine. Sie soll mit dem 16-Bit D/A-Wandler PCM53 und dem 12-Bit A/D-Wandler ADC674 (Wandelzeit 15 us) arbeiten. Für die Tiefpässe möchte ich die Switched-Capacitor-Filter LTC1062 verwenden und sie als Butterworth-Tiefpässe 10. Ordnung (60 db / Oktave) mit einer Eckfrequenz von 12 KHz betreiben. Durch Verwendung eines Multiplex-Verfahrens soll die Platine auch stereophone Signale verarbeiten können.

4.2.3. Die Interface-Platine

Die Interface-Platine dient zum Datenaustausch zwischen dem Signalverarbeitungsrechner und dem Steuerrechner. Zusätzlich habe ich auf ihr einen programmierbaren Teiler aufgebaut, der den Taktausgang des TMS von 5 MHz durch einen Wert zwischen 1 und 4096 teilt. Er dient zur Erzeugung einer festen Wandelrate.
Die Interfacekarte wird mit dem Steuerrechner über ein 16-poliges Flachbandkabel verbunden. Der Steuerrechner kann über dieses Kabel vier 16-Bit-Latches beschreiben. Die Tri-State-Ausgänge von drei Latches sind mit dem Datenbus des TMS verbunden. Er kann so die in diesen Latches gespeicherten Daten lesen. Umgekehrt kann der TMS über einen Ausgabebefehl ein fünftes 16-Bit-Latch beschreiben, das vom Steuerrechner gelesen werden kann. Das vierte vom Steuerrechner beschreibbare Latch liefert das Teilungsverhältnis für den Frequenzteiler. Den Ausgang des Teilers habe ich mit dem BIO-Eingang des TMS verbunden. Dieser kann sich softwaremäßig damit synchronisieren und so die Wandler mit einer konstanten Abtastrate von normalerweise 30 KHz bedienen.

4.2.4. Die RAM-Platine

Die RAM-Platine besitzt zur Zeit eine Kapazität vom 32 K Word. Dieses entpricht einer Speicherzeit von etwa 1 s. Ich verwende dabei 8 statische RAMs des Typs uPD4364 (8K Byte). Mit RAMs des Typs uPD43256 (32K Byte) beabsichtige ich die Kapazität auf 128K Word - entsprechend etwa 4 s Speicherzeit - zu erweitern. Für die Ansteuerung der Platine verwende ich 3 Aus- und 1 Eingabetor des TMS. Für einen Speicherzugriff sind zwei Ausgaben (max. 24 Adress-Bits und 1 Steuer-Bit) und eine Ein- oder Ausgabe (16 Daten-Bits) notwendig.

4.3. Software

Die Software für den TMS 32010 dient dazu, Töne oder Geräusche mit einer vom Steuerrechner vorgegeben Frequenz, Amplitude und Klangfarbe zu erzeugen, zu mischen und auf den D/A-Wandler auszugeben. Auch kann die Software Effektgeräte simulieren. Das Programm für den TMS darf aufgrund der Abtastrate von normalerweise 30 KHz nicht mehr als 33,4 us * 5 MHz = 167 Taktzyklen pro Durchlauf benötigen. Dieses entspricht etwa 150 Befehlen, da fast alle Befehle nur 1 Taktzyklus benötigen.
Um die Programme für den TMS nicht von Hand übersetzen zu müssen, habe ich mir einen Assembler geschrieben, der ein in der Assemblersprache des TMS 32010 geschriebenes Programm in eine Maschinencode-Datei im Intel-Hex-Format umwandelt. Dieser Cross-Assembler läuft auf einem IBM-XT-kompatiblen System, einem Olivetti M24. Ich habe ihn in PASCAL geschrieben. Er ist etwa 1200 Zeilen lang und kann auch Labels verarbeiten.

4.3.1. Mögliche Syntheseverfahren und ihre Probleme

In meinen ersten Versuchen zu Tonerzeugung habe ich mich zunächst mit der Erzeugung von Sägezahn-Schwingungen beschäftigt. Dazu wird in jedem Durchlauf des TMS-Programmes ein Zähler um einen vorgegebenen Wert erhöht. Die Frequenz des so erzeugten Tons ist proportional zu dieser Schrittweite. Die im Zähler gespeicherte Zahl wird auf den D/A-Wandler ausgegeben. Wenn man nun eine Schrittweite wählt, die keine Potenz von 2 ist, hört man neben dem eigentlichen Ton auch "Störungen", deren Frequenzen keine Oberwellen des Sägezahns sind. Diese entstehen dadurch, daß man ein nicht bandbegrenztes Signal auf den Wandler gibt und so das Shannonsche Abtastheorem verletzt. Dieser Effekt wird um so stärker, je höher die Frequenz des Sägezahns wird. Eine Sägezahnschwingung enthält nämlich alle nur möglichen Oberwellen.
Es ist also nicht sinnvoll, Töne zu erzeugen, deren Oberwellen über der halben Abtastfrequenz liegen. Da aber die Oberwellen die Klangfarbe eines Tones bestimmen sind sie unbedingt notwendig. Eine Möglichkeit, dieses Problem zu lösen, besteht darin, jeweils neben dem Grundton nur die Oberwellen zu erzeugen, die nicht oberhalb der halben Abtastfrequenz liegen. Für diese Fourier-Synthese mit einer variablen Anzahl von Oberwellen benötigt man aber relativ viel Rechenzeit. Eine andere Möglichkeit besteht darin, die Töne anhand einer Tabelle zu erzeugen, in der die Kurvenform einer Periode mit den jeweils in einem bestimmten Frequenzbereich erlaubten Oberwellen enthalten ist. Man muß dabei für jede Oktave die dazugehörige Kurvenform speichern oder auf hochfrequente Oberwellen ganz verzichten.
Es gibt aber noch weitere Möglichkeiten, verschiedene Oberwellenstrukturen zu erzeugen. Dazu bedient man sich mehrerer Sinusoszillatoren und läßt sie einander auf verschiedene Art modulieren. Drei grundlegende Modulationsarten sind möglich: Amplitudenmodulation (AM), Frequenzmodulation (FM) und Phasenmodulation (PM). Wenn man von zwei Sinusschwingungen ausgeht, ergeben sich bei der Amplitudenmodulation maximal zwei neue Ausgangfrequenzen. Bei der Frequenz- und Phasenmodulation dagegen sind es wesentlich mehr.
Man kann mit diesen Modulationsarten eine große Menge verschiedener Klangfarben mit einem relativ geringen Aufwand erzeugen. Da sich die einzelnen so entstandenen Frequenzen selten harmonisch zueinander verhalten, fällt es außerdem dem Hörer nicht auf, wenn Frequenzen oberhalb der halben Abtastfrequenz entstehen.

Ein weiteres Syntheseprinzip ist die Wiedergabe gespeicherter Originalklänge ("Sound Sampling"). Hier werden die störenden Oberwellen schon bei der Aufnahme über den A/D-Wandler weggefiltert. Wenn man aber bei der Wiedergabe eine andere Tonhöhe als bei der Aufnahme erzielen möchte, es aber - wie in meinem Fall - nicht sinnvoll ist, die Wandelrate zu verändern, ergibt sich das Problem, daß Zwischenwerte ausgegeben werden müssen, die gar nicht aufgenommen wurden. Man kann dazu zwischen den aufgenommenen Werten linear interpolieren. Bei der Wiedergabe mit einer höheren Tonhöhe ergeben sich dann keine Probleme. Gibt man jedoch den Klang mehr als eine Oktave tiefer wieder, machen sich die bei der Interpolation entstehenden Oberwellen doch störend bemerkbar.

4.3.2. Realisierte Konzepte zur digitalen Klangsynthese

Ich habe nun mehrere Programme geschrieben, die jeweils eins der oben beschriebenen Syntheseverfahren realisieren. Da sie alle auf Sinusoszillatoren beruhen, möchte ich zunächst das Programm für einen solchen Oszillator beschreiben:
Die Sinusschwingungen erzeuge ich anhand einer Tabelle mit 128 Sinuswerten, zwischen denen linear interpoliert wird. Dieses Programm habe ich mit kleinen Veränderungen nach einem Beispielprogramm von Texas Instruments geschrieben, da es in Bezug auf die Ausführungszeit optimiert ist. Es erhöht einen Winkel-Zähler je einmal pro Durchlauf um einen frequenzbestimmenden Wert und ermittelt dann mit eben dieser Tabelle den Sinus des Winkels. Das Programm benötigt pro Oszillator 19 Befehle und dauert 23 Taktzyklen, also 4,6 us.
Die bei der Synthese benötigten aktuellen Parameter, also Amplituden- und Frequenzwerte, werden über die Interface-Platine übergeben. Dazu wird in ein Word des Interfaces die Nummer des zu ändernden Parameters und in ein anderes der neue Parameterwert geschrieben. Dann wird für die Dauer eines Programmdurchlaufes eine Bit des dritten Words gesetzt. Das TMS-Programm fragt nun einmal pro Durchlauf dieses Word ab und übernimmt gegebenenfalls den neuen Wert in seinen Datenspeicher. Da auch der Steuerrechner häufig eine 16*16-Bit-Multiplikation durchführen muß, ein Programm dazu aber etwa 500 us benötigen würde, habe ich das TMS-Programm so geschrieben, daß, falls gerade kein Parameter geändert wird, das Produkt der in zwei Words des Interfaces gespeicherten Zahlen in das vom Steuerrechner lesbare Word geschrieben wird. So benötigt der Steuerrechner für eine Multiplikation insgesammt weniger als 100us.

Ich habe zunächst ein Programm geschrieben, das gleichzeitig 4 Sinusschwingungen mit verschiedenen Frequenzen und Amplituden erzeugt. Als ich diese Programm nun zum ersten Mal vom Steuerrechner steuern ließ, stellte sich heraus, daß sich die erzeugten Sinustöne sehr rauh anhörten, wenn man ihre Amplitudenwerte relativ schnell veränderte (Alle Parameter werden etwa 100 mal pro Sekunde verändert; siehe 5.2:2.). Dieses kommt daher, daß man einen Sprung im Ausgangssignal erhält, den man als Knacken wahrnimmt, wenn man bei einem Signal, das gerade nicht den Wert Null hat, den Amplituden-Faktor ändert. Treten diese Knack-Geräusche häufig auf, so hört sich das Signal rauh an.
Zur Vermeidung dieser Sprünge im Amplituden-Faktor habe ich die Amplituden-Parameter für die Sinustöne über je einen digitalen Tiefpaß erster Ordnung geleitet, der sich auf dem TMS recht einfach programmieren läßt. Die Eckfrequenz dieses Tiefpasses liegt bei etwa 20 bis 50 Hz, so daß sich der Amplituden-Faktor nicht mehr sprunghaft sondern kontinuierlich ändert und sich das Ausgangssignal wieder "sauber" anhört.

Ein ähnliches Problem ergibt sich auch bei den Frequenzwerten. So hört man bei relativ hohen Tönen und schnellen Frequenzveränderungen, daß sich die Frequenz nicht kontinuierlich sondern in Stufen ändert. Man kann auch diese "Fehler" mit digitalen Tiefpässen vor den Frequenzwerten lösen. Ich müßte dann aber aus Zeitgründen auf einen der Sinusoszillatoren verzichten. Da solche schnellen Frequenzänderungen beim praktischen Gebrauch des Synthesizers nicht auftreten, habe ich diese Tiefpässe jedoch nicht programmiert. Mit diesem Programm kann man nun entweder mit reinen Sinustönen 4-stimmig polyphon oder mit einem nach dem Prinzip der Fourier-Synthese erzeugten Klang aus Grundton und 3 Oberwellen monophon spielen.
Ich habe auch Programme für die AM-, FM-, und PM-Synthese geschrieben. Die AM-Synthese ist dabei für den praktischen Gebrauch schlecht geeignet, da man mit ihr nur sehr einfache Klänge erzeugen kann (siehe 4.3.1). Die FM- und PM-Synthese sind durchaus verwendbar. Sie benutzen auch jeweils 2 bis 4 Sinusozillatoren, die in einer Art Kette hintereinandergeschaltet sind. Dabei beeinflußt jeweils ein Oszillator die Frequenz beziehungsweise die Phase des nächsten Oszillators. Man kan hierbei über das Interface die Frequenzen und die Stärke der Modulation für jeden Oszillator getrennt einstellen.
Mit der RAM-Platine bin ich nun auch in der Lage, das in 4.3.1 beschriebene "Sound Sampling" durchzuführen. Ich kann dabei auf Tastendruck einen natürlichen Klang von etwa 1 s Dauer (bei 32 K Word RAM) aufnehmen und mit ihm dann wie mit einem synthetisierten Klang spielen.
Die RAM-Platine ermöglicht es auch, Klangeffekte zu erzeugen, bei denen Signale zwischengespeichert werden müssen. So habe ich ein Programm geschrieben, mit dem man die Effekte eines Echogerätes - auch "Digital Delay" genannt - erzielen kann.

5. Der Steuerrechner

Der Steuerrechner dient dazu, die Steuerparameter für die Tongenerator-Programme im Signalverarbeitungsrechner abhängig von den Singalen des Keyboardrechners und dem eingestellten Klang zu berechnen. Es werden die für die Tongeneratoren notwendigen Steuerwerte, die den Steuerspannungen eines analogen Synthesizers entsprechen, etwa 100 mal pro Sekunde berechnet. Die Verknüpfung der einzelnen im Steuerrechner simulierten Module geschieht anhand einer Tabelle und kann so leicht geändert werden. Die so berechneten Steuerwerte werden dann über die Interface-Platine an den Signalprozessor übergeben.

5.1. Hardware

Die Hardware des Steuerrechners besteht aus zwei Platinen, der Rechner- und der Interface-Platine, die im 19-Zoll-Gehäuse untergebracht sind und auch aus dem Schaltnetzteil versorgt werden.

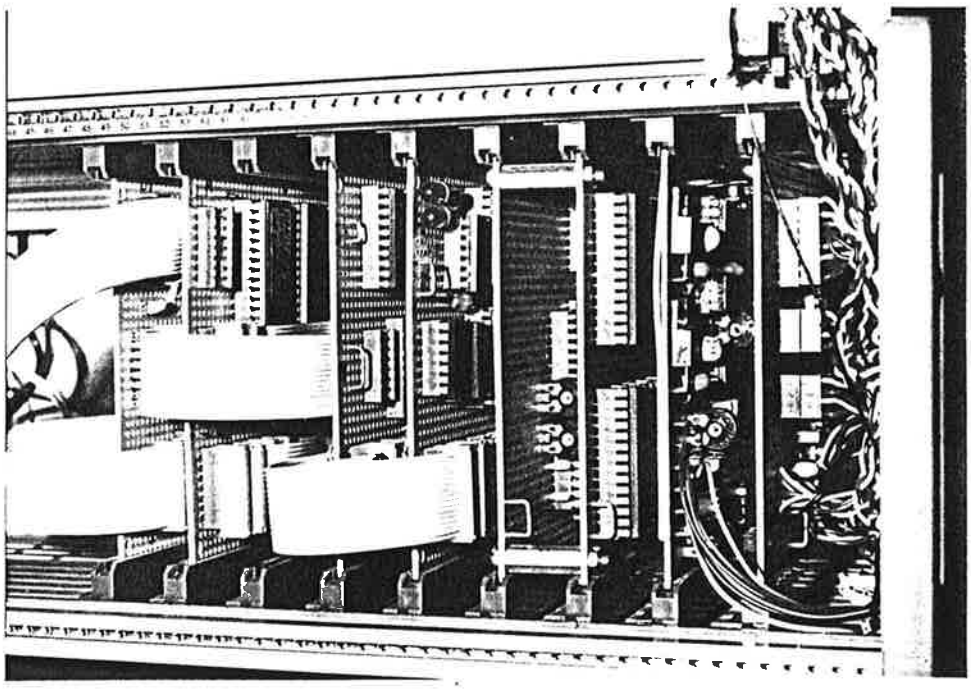
5.1.1. Die Rechner-Platine

Auf der Rechner-Platine sind neben der Z80 CPU, die mit 6 MHz getaktet wird, und einem Timer-IC vom Typ 8253 auch ein 8 K Byte EPROM vom Typ 2764 und drei 8 K Byte RAMs vom Typ 6264 untergebracht. Neben den Adress-Dekodern und den Schaltungen für das Reset- und Taktsignal befinden sich vier Sockel auf der Platine, die mit einem Teil der Bussignale beschaltet sind und über die die Rechnerplatine mit Flachbandkabeln mit der Interface-Platine und dem Signalverarbeitungsrechner verbunden ist.

5.1.2. Die Interface-Platine

Die Interface-Platine beinhaltet zwei parallele Schnittstellen. Sie ist mit dem Steuerrechner und dem Terminalrechner über je ein 16-poliges Flachbandkabel verbunden, auf dem ein Teil der Bussignale des jeweiligen Rechners liegen. Mit dem Keyboard ist sie über ein langes 24-poliges Kabel verbunden. Um Störungen zu verhindern, habe ich jede zweite Ader dieses Kabels auf Masse gelegt. Von den 12 verbleibenden Adern benutze ich 8 für die Daten und 3 für Steuersignale.
Die Schnittstelle zwischen Keyboard und Steuerrechner ist unidirektional. Sie besteht im wesentlichen aus einem 8-Bit-Latch mit Tri-State-Ausgang und einem RS-Flipflop. Die Eingänge des Latches sind über das 24-polige Kabel mit dem Datenbus des Keyboardrechners und die Ausgänge mit dem Datenbus des Steuerrechners verbunden. Das RS-Flipflop dient dem "Handshaking". Zusätzlich besteht die Möglichkeit, vom Steuerrechner aus ein Interrupt-Signal für den Keyboardrechner zu erzeugen. Das Interruptprogramm im Keyboardrechner überträgt daraufhin den aktuellen Datensatz zum Steuerrechner. Ich beabsichtige, diese Schnittstelle auf bidirektionalen Betrieb zu erweitern.
Die Schnittstelle zwischen Steuerrechner und Terminalrechner arbeitet bidirektional. Die beiden Hälften der Schnittstelle sind im Prinzip wie die Schnittstelle zum Keyboard aufgebaut. Die beiden Hälften zusammen bilden so eine einfache bidirektionale Parallelschnittstelle mit "Handshaking".

Steuer- und Signalverarbeitungsrechner



5.2. Software

Die Software für den Steuerrechner habe ich zunächst auf einem Rechner mit einer 8085 CPU entworfen. Ich konnte sie dann aufgrund der Aufwärtskompatibelität zur Z80 CPU im Steuerrechner direkt übertragen.

5.2.1. Systemsoftware

Zunächst mußte ich Systemsoftware für die Steuerrechner schreiben. Hierzu gehört neben einem Monitorprogramm auch ein Hilfsprogramm, mit dem man Programme in den Signalverarbeitungsrechner laden und auch die Interface-Platine dieses Rechners betreiben kann.
Um einen Benutzerdialog von Steuerrechner aus führen zu können, habe ich für den Terminalrechner ein Programm zur Terminalemulation geschrieben, das die Parallelschnittstelle zum Steuerrechner benutzt. Das Monitorprogramm für den Steuerrechner kann Speicherinhalte ausgeben und verändern, Programme starten und Speicherbereiche vom oder zum Terminalrechner kopieren. Der Terminalrechner besitzt auch eine serielle Schnittstelle. Über diese habe ich während der Entwicklungsphase des Synthesizers zum Beispiel die auf einem Olivetti M24 assemblierten Programme für den Signalverarbeitungsrechner übertragen. Das Hilfsprogramm für den Signalverarbeitungsrechner benutzt ebenfalls den Terminalemulator im Terminalrechner für den Benutzerdialog.

5.2.2. Software zur Klangsteuerung und -programmierung

Im normalen Betrieb des Synthesizers läuft auf dem Steuerrechner ein Programm, das dazu dient, die verschiedenen Steuerwerte für den Signalprozessor zu berechnen. Das Programm wird dabei pro Sekunde etwa 100 mal durchlaufen. Als Zeitbasis für diese konstante Rate dient das Timer-IC 8253 auf der Rechner-Platine.
Am Anfang eines Durchlaufes wartet das Programm auf ein Synchronisationssignal des Timer-ICs, um eine konstante Durchlaufrate von 100 Hz zu erreichen. Dann werden die neuen Werte für die Parameter des Tongeneratorprogramms im Signalverarbeitungsrechner berechnet. Ich wollte mich nicht auf eine feste Kombination der einzelnen, die Steuerparameter bestimmenden, Module beschränken. Daher habe ich in einer Tabelle, die pro Durchlauf einmal abgearbeitet wird, zu jedem verwendeten Modul angegeben, wo die benötigten Eingangswerte stehen und wo die Ausgangswerte abgespeichert werden sollen. Für diese Zwischenspeicherung habe ich ein Speicherfeld mit 256 Words vorgesehen. Davon sind je 64 für die ein- und auszugebenden Werte reserviert. Alle Zahlen werden dabei im 16-Bit-Zweierkomplement-Format gespeichert. Die Frequenzinformationen für die Tongeneratoren werden logarithmisch mit einer Auflösung von 4096 Werten je Oktave gespeichert. 0 entspricht dabei dem eingestrichenen C. Dieses hat, wie bei analogen Synthesizern, zum einen den Vorteil, daß man zum Andern der Tonlage nur einen bestimmten Offset zum Frequenzwert dazu addieren muß. Zum anderen hat man in allen Frequenzbereichen die gleiche relative Auflösung.

Mit meinen Steuerrechner-Programm kann ich momentan folgende Module simulieren:

KON lädt eine Konstante in ein Element des Speicherfeldes.
MOV kopiert den Inhalt eines Elementes des Speicherfeldes in ein anderes Element
LFO simuliert einen LFO. Ein Element des Speicherfeldes gibt dabei die Frequenz an. Der aktuelle Wert des Ausgangs wird wieder in ein Element des Speicherfeldes geschrieben. Es sind die Kurvenformen Sinus, Dreieck, Sägezahn aufwärts, Sägezahn abwärts, Rechteck und Sample/Hold (ein sich zufällig ändernder Wert) möglich.

ENV simuliert einen Hüllkurvengenerator. Ein Element des Speicherfeldes gibt dabei an, ob eine Taste gedrückt ist. Der Ausgangswert wird auf ein Element des Speicherfeldes geschrieben. Seine Hüllkurve wird durch 4 Anstiegs- bzw. Abfallraten und den 4 dazugehörigen Amplituden definiert.

ADD addiert die Inhalte zweier Elemente des Speicherfeldes und speichert die Summe wieder in ein Element des Speicherfeldes.

ADDK wie ADD, nur daß eine Konstante addiert wird.

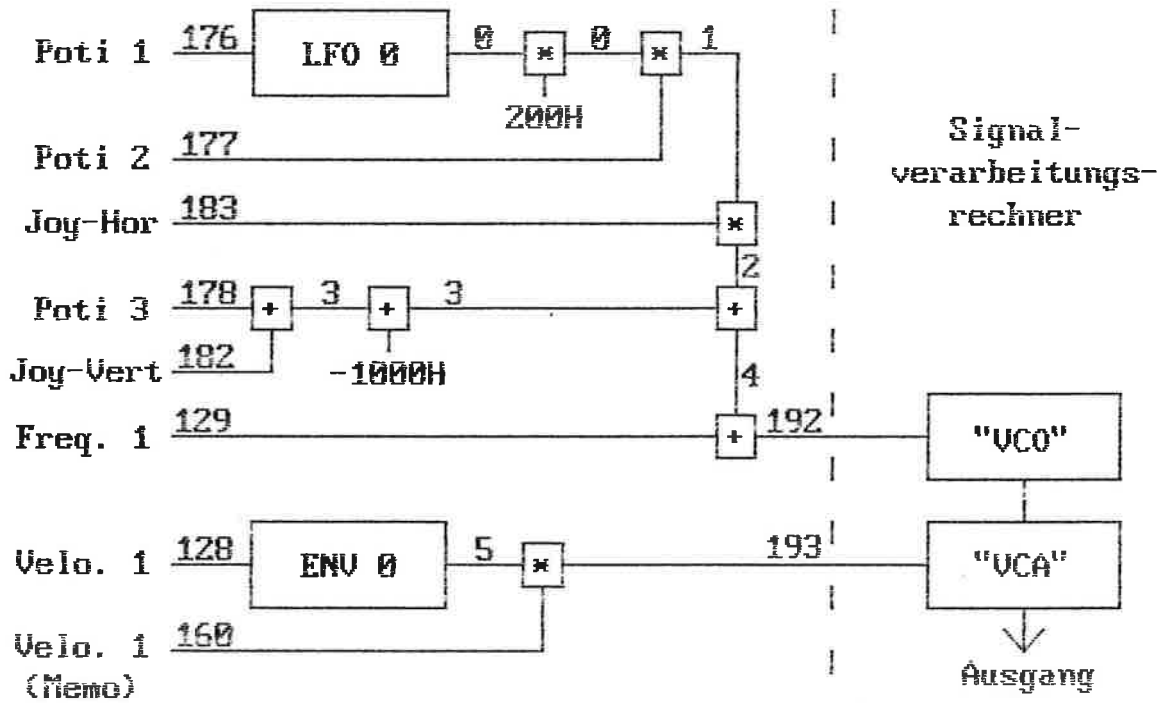
MPY multipliziert die Inhalte zweier Elemente des Speicherfeldes und speichert die Summe wieder in ein Element des Speicherfeldes.

MPYK wie MPY, nur daß mit einer Konstante multipliziert wird.

Es ist im Augenblick möglich, bis zu etwa 50 Module zu verwenden.

Am Ende jedes Durchlaufes werden die neu berechneten Parameter im Ausgabefeld über die Interface-Platine dem Signalprozessor übergeben. Dabei werden die Frequenzinformationen, die im Steuerrechner logarithmisch gespeichert werden, anhand einer Tabelle in den entsprechenden Frequenzwert für das Tongeneratorprogramm umgerechnet. Zusätzlich wird ein neuer Datensatz vom Keyboard geladen. Da das Keyboard maximal 16-stimmig polyphon arbeiten kann, gibt der Datensatz pro Stimme an, welche Taste gedrückt wurde, ob sie noch gedrückt ist und wenn ja, mit welcher Geschwindigkeit sie angeschlagen wurde.
Es wird auch die Stellung der 8 Potis auf dem Keyboard übergeben. Diese Informationen werden in das benötigte Format umgewandelt und dann im Speicherfeld abgelegt. Die Informationen über die Anschlagsgeschwindigkeit bleiben auch noch nach dem Loslassen der jeweiligen Taste gespeichert. Zusätzlich wird vom Keyboard die Information übergeben, welches Klang-Programm mit den 8 Tastern auf dem Keyboard ausgewählt wurde. Ein Klang besteht dabei aus der Tabelle für die Verknüpfung der einzelnen Module und den zur Definition der Hüllkurvenverläufe notwendigen Informationen sowie den bei den einzelnen LFOs gewählten Kurvenformen. Der 8. "Klang" führt zum Ausschalten der Tongeneratoren und dem Verlassen des Programms für den Steuerrechner.

Blockschaltbild eines Klanges



Modultabelle

```
01 00 00 00    LOOP
05 00 B0 00    LFO 0,176,0
0A 00 02 00    MPYK 200H,0
09 00 B1 01    MPY 0,177,1
09 01 B7 02    MPY 1,183,2
07 B2 B6 03    ADD 178,182,3
08 00 F0 03    ADDK -1000H,3
07 02 03 04    ADD 2,3,4
07 04 81 C0    ADD 4,129,192
06 00 80 05    ENV 0,128,5
09 05 A0 C1    MPY 5,160,193
00 00 00 00    END
```

6. Das Keyboard

Die Hauptaufgabe des Keyboardrechners ist es, die Tastatur und die anderen Bedienelemente abzufragen und diese Daten dem Steuerrechner zu übermitteln. Es kann maximal 16-stimmig polyphon arbeiten und ist anschlagsdynamisch.

6.1. Hardware

Der Keyboardrechner besteht aus 4 Platinen (Rechner-Platine, Tastaturinterface, Bedienelement-Platine und Netzteil). Diese Platinen sind zusammen mit dem 3 1/2 oktavigen Manual und den verschiedenen Bedienelementen in einem selbstgebauten Holzgehäuse untergebracht.
Die Rechner-Platine ist mit einer Z80 CPU mit 6 MHz Taktfrequenz, 8 K Byte EPROM und 8 K Byte RAM sowie einem Timer-IC vom Typ 8253 ausgestattet. Es befinden sich auch die Takt- und Resetschaltungen sowie einige Dekoder-ICs auf dieser Platine. Zusätzlich sind zwei Busanschlüsse für 16-polige Flachbandkabel zum Anschluß der beiden anderen Platinen und die Schaltung des Interfaces zum Steuerrechner auf der Rechner-Platine untergebracht.
Mit dem Tastaturinterface kann ich über Multiplexer den Zustand jeder der 44 Tasten des Manuals abfragen. Da ich jede Taste mit einem Umschaltkontakt ausgerüstet habe, gibt es nun die drei verschiedene Zustände: Oben, Mitte und Unten. Aus der Zeit, für die der Umschaltkontakt einer Taste beim Anschlagen in der Mitte bleibt, bestimme ich die Anschlagsgeschwindigkeit. Zusätzlich befindet sich auf dieser Platine ein 8-Bit-A/D-Wandler mit einem Eingangsmultiplexer für 8 Spannungen, mit dem ich die Stellungen der 6 Schieberegler und der beiden Achsen des Joysticks, also der analogen Bedienelemente, abfragen kann.
Auf der Bedienelement-Platine habe ich 8 kleine Schalter und 8 Taster untergebracht, deren Zustand vom Keyboardrechner gelesen werden kann. Jedem Taster ist eine vom Keyboardrechner steuerbare Leuchtdiode zugeordnet. Ich beabsichtige zusätzliche Bedienelemente für die Klangprogrammierung auf dem Keyboard unterzubringen. Das Netzteil erzeugt mit Linearreglern bei einer Ausgangsleistung von 5 W die Spannungen von 5 V und - 5 V.

6.2. Software

Die Software des Keyboardrechners wertet die Informationen vom Manual und den Bedienelementen aus und gibt diese Daten dann in einer geeigneten Form an den Steuerrechner weiter. Zuätzlich enthält sie einige Testprogramme, um die Funktion einzelner Teile des Keyboardrechners zu testen.

Das eigentliche Programm des Keyboardrechners wird in einer Schleife rund 1000 mal pro Sekunde durchlaufen. Dabei werden zunächst die Bedienelemente bearbeitet. Die Stellungen der 8 Potis und Schalter werden dabei direkt im aktuelle Datensatz abgelegt. Bei den 8 Tastern wird gespeichert, welcher von ihnen zuletzt gedrückt wurde. Dieses wird durch seine Leuchtdiode angezeigt. Die Nummer des Tasters wird dann auch im aktuellen Datensatz gespeichert. Bei den Tasten des Manuals wird ermittelt, welche ihren Zustand geändert hat. Hat sie den oberen Zustand (Ruhezustand) verlassen, wird die aktuelle Zeit gespeichert. Erreicht sie bei einem späteren Programmdurchlauf dann den unteren Zustand, wird wieder die aktuelle Zeit gelesen und aus der Zeitdifferenz die Anschlagsstärke berechnet. Als Zeitbasis verwende ich hierbei einen frei durchlaufenden 16-Bit-Zähler im Timer-IC. Wenn eine Taste so angeschlagen wurde, suche ich für sie einen freien der maximal 16 "Ausgangskanäle" des Keyboards und speichere dort die Nummer der Taste und ihre Anschlagsstärke. Ich sorge dabei dafür, daß jeder Kanal maximal lange ausklingen kann. Ist kein Kanal frei, benutze ich den der jeweils "ältesten" gedrückten Taste. Die Anzahl der zugelassenen Kanäle kann ich mit 4 der Bedienelement-Schalter vorgeben und so zwischen monophonem und 2 bis 16-stimmig polyphonem Spiel wählen. Mit der geplanten bidirektionalen Schnittstelle zum Steuerrechner könnte die Anzahl der zugelassenen Kanäle auch vom Steuerrechner bestimmt werden. Den Zustand der 16 Kanäle speichere ich auch im aktuellen Datenfeld. Im Steuer- und Signalverarbeitungsrechner wird dann für jeden Kanal eine eigene "Stimme" erzeugt. Wird vom Steuerrechner ein Interrupt ausgelöst, beginnt das Interruptprogramm im Keyboardrechner den jeweils aktuellen Datensatz über die Parallelschnittstelle zum Steuerrechner zu übertragen.

7. Der Terminalrechner

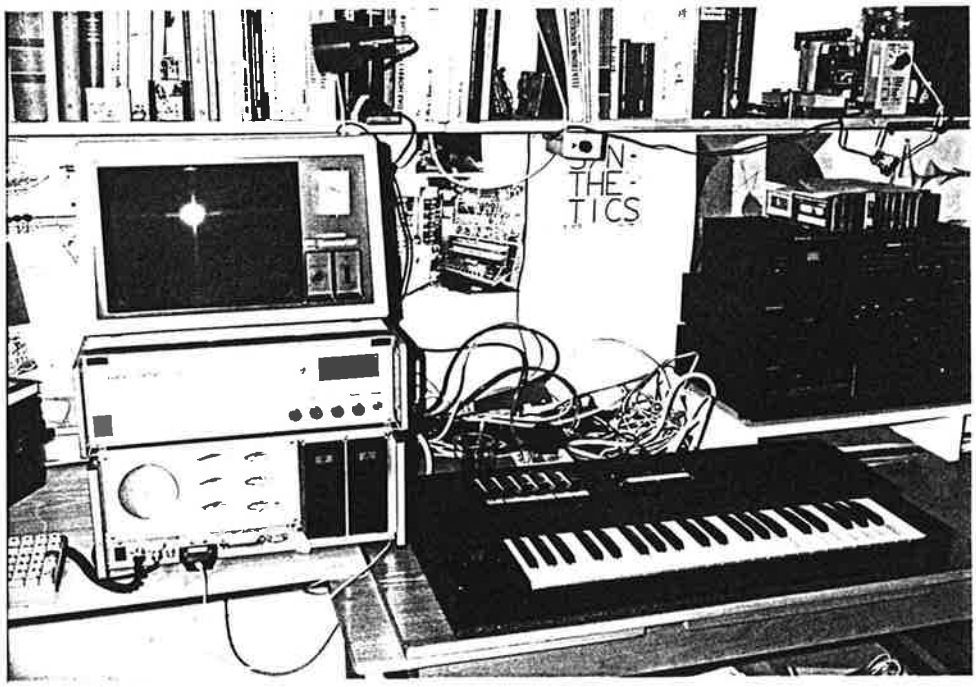
Der Terminalrechner - ein CP/M System - wird zur Zeit als Terminal und Massenspeicher für den Steuerrechner verwendet, mit dem er über ein 16-poliges Flachbandkabel verbunden ist. Auf ihm läuft normalerweise ein Terminalemulationsprogramm ab, mit dem man zum Laden und Speichern der Klangdaten auch auf die Diskettenlaufwerke zugreifen kann.
Ich beabsichtige für ihn auch ein Sequencer-Programm zu schreiben. Diesem müßte man dann vorher ein Melodie oder Tonfolge in geeigneter Form vorgeben oder vorspielen, und er würde diese dann in der gleichen Weise, als wenn sie auf dem Keyboard gespielt würden, an den Steuerrechner weitergeben.

8. Bisherige Ergebnisse und weitere Planungen

Zur Zeit entwickele oder plane ich einige Erweiterungen für den Signalverarbeitungsrechner. Dieses ist zunächt die Erweiterung der RAM-Platine von 32 K Word RAM auf 128 K Word RAM (siehe 4.2.4).
Auch möchte ich eine verbesserte Wandler-Platine mit Wandlern höherer Auflösung aufbauen (siehe 4.2.2.), mit der ich eine bessere Klangqualität erreichen würde.
Weiterhin beabsichtige ich die Schnittstelle zwischen Keyboard und Steuerrechner auf bidirektionalen Betrieb zu erweitern und Bedienelemente für die Klangprogrammierung auf dem Keyboard unterzubringen. Man könnte dann für den Terminalrechner Programme zur Programmierung und Veränderung von Klängen schreiben, die den Grafikbildschirm des Terminalrechners benutzen würden und somit einen sehr übersichtlichen und einfachen Benutzerdialog ermöglichen würden.

Ich kann also abschließend sagen, daß es mir gelungen ist, einen vollständig digital arbeitenden Synthesizer aufzubauen, der durchaus in der Lage ist, Klänge in einer akzeptabelen Qualität zu erzeugen. Ich erreiche dabei nicht die Qualität und Leistungsfähigkeit kommerzieller Systeme. Dieses ist aber auch nicht verwunderlich, wenn man zum Beispiel bedenkt, daß diese zum Teil eigens für das jeweilige Gerät entwickelte ICs verwenden. So weisen natürlich auf Synthesizer-Anwendungen spezialisierte digitale Signalprozessoren eine noch wesentlich höhere Leistungsfähigkeit als der vom mir verwendete Prozessor TMS 32010 auf.
Der wichtigste Vorteil meines Systems besteht aber darin, daß ich nicht von Anfang an auf ein bestimmtes Syntheseverfahren - zum Beispiel FM-Synthese - festgelegt bin, sondern mit nahezu allen möglichen digitalen Syntheseverfahren arbeiten kann. Zudem kann ich meinen Synthesizer sogar als Effektgerät oder als digitales Schlagzeug verwenden. Zwar arbeiten kommerzielle Systeme dabei zum Teil auch mit digitaler Signalverarbeitung, sind aber selten so vielseitig, wie mein digitaler Synthesizer es ist. Er wäre auch für die Entwicklung neuer digitaler Syntheseverfahren aufgrund seiner Vielseitigkeit durchaus brauchbar.

Der digitale Synthesizer in seinem momentanen Ausbaustadium



9. Literatur und Hilfen

Benutzte Literatur:

1) Digitaler Synthesizer (Beitrag für Jugend forscht 1986), Heiko Purnhagen, 1986
2) A Brief Introduction to Electronic Music Synthesizers, Robert A. Moog, BYTE, December '82
3) TMS 32010 User's Guide, Texas Instruments, 1983
4) Precision Digital Sine-Wave Generation with the TMS 32010, Application Report, Texas Instruments, 1984
5) Datenblätter bzw. -bücher der übrigen verwendeten IC's
6) Beschreibungen der benutzten Microcomputersysteme

Benutzte Hilfen:
Es wurden keine weiteren Hilfen benutzt.