Computermusiksysteme

Winfried Ritsch / IOhannes m zmölnig

Unterrichtsbehelfe WS2012/13
10.Juni.1994 - 14. Januar 2013
Institut f. elektronische Musik u. Akustik [1]
Universität f. Musik und Darstellende Kunst Graz[2]

(Diese Unterlagen dienen als Behelfe für einige der Vorlesungstermine und sind keine vollständige Auflistung des Unterrichtsstoffes und beinhalten auch zusätzliche weiterführende Informationen). Sie sind kein vollständiges Skriptum.

[1] `http://iem.at/`
[2] `http://www.kug.ac.at/`

Zusammenfassung

Das Ziel der Lehre ist die Vertiefung der Prinzipien von Computermusik-Systeme und deren Software in für deren Einsatz, vom Prototyping neuer Algorithmen bis hin Realisierung komplexer Klangverarbeitung und -steuerungen in Aufführungen.

Im ersten Teil wird auf Techniken eingegangen, wie Musik mit dem Computer erzeugt und gesteuert wird, im zweiten Teil auf Echtzeit-Anwendungen und diese anhand einer beispielhaften Realisation eines Projektes vorgeführt.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Einsatzgebiete von Computermusik Systemen in der Elektronischen Musik

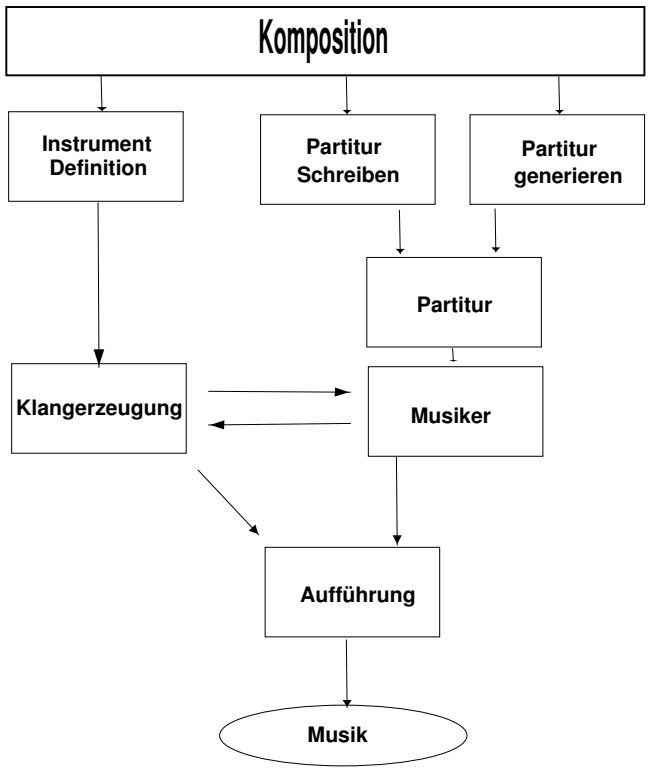


Abbildung 1.1: Arbeitsgebiete in der Computermusik

Bei den Computermusik Sytemen interessiert uns vor allem die Zusammenarbeit von einzelnen Elementen in der Computermusik, die Schnittstellen und Verbindungen zwischen Computer, Programmen und die Integration zu einem anwendbaren System.

Es gibt zur Zeit drei wesentliche Einsatzgebiete von Computermusik Sytemen die sich aus der Abbildung 1.1 ergeben:

- Steuerung: Wie werden Instrumente definiert und deren Daten verarbeitet ?
- Komposition: computergestützte Partiturerstellung, Definitionen und Syntax und zugehörigen Analysen.

- Aufführungen: Schnittstelle zwischen Partiturdaten und der Instrumentendefinitionen zur Erzeugung von Musik.

1.2 Komposition mit Computer

Dabei geht es im klassischen Sinne um die "Partitursynthese" als Generation von Steuerdaten, die als computergestützte Komposition mit `CAC` (Computer Assisted Composition), "Automatische Komposition" und "Algorithmische Komposition" bezeichnet wird.

Voraussetzung für Partiturerstellung mit Computer ist die Findung eines passenden **Algorithmus** zur Bestimmung dieser.

Anmerkung: Algorithmus

> *Der Begriff Algorithmus leitet sich aus den Namen "Al Chorezmi" ab, der als Mathematiker am Hofe des Kalifen zu Bagdad ein Buch mit dem Titel "Regeln der Wiedereinsetzung und Reduktion" schrieb*[1] *und wurde unabhängig voneinander, von verschiedenen Mathematikern unterschiedlich definiert*[2]*.*
>
> *Die Beschreibung eines Algorithmus ist endlich, auch wenn er unendlich viele Ergebnisse liefern kann (z.B.: Berechnung der Quadratwurzel). Es gibt dazu die "Theorie der Berechenbarkeit", welche untersucht ob Problemstellungen algorithmisch lösbar sind oder nicht. (1)*

Den Wunsch nach Kompositions-Maschinen gab es schon, seit die Mathematik als Grundlage der Komposition verwendet wurde (ca. 16.-17.Jh.). Leibniz und Marin Mersenne (1588-1646) führte die Gleichsetzung von Kombinieren und Komponieren an[3].

Der "Kompositions-Computer" wurde erst in der Zeit der Seriellen Musik und der "Musique Concrete" entwickelt. Einer der ersten Einsätze war die Verwendung von Zufallsgeneratoren die von Lochstreifenleser gesteuert wurden.

---

[1] laut Heinz Zemanek, Al Chorezmi oder auch Muhammed Al Chwarizmi (geb. ca . 783 gestorben ca. 850), der Namensgeber des Algorithmus,in:ders., das geistige Umfeld in der Informationstechnik, Berlin Springer 1992, S.51-64

[2] u.a. in den 30er Jahren: Herbrand, Gödel, Church, Kleene und Turing

[3] Eberhardt Knobloch in Maß, Zahl und Gewicht, Herzog August Bibliothek,1989 S.249-264

Abbildung 1.2: Computer Illiac I, 1952

Hierzu gibt es eine Reihe von Beispielen. Es entstand das in den USA als erste Komposition mittels Computer geltende Werk "Illiac Suite" für Streichquartett von Leajaren A. Hiller und Leonard M.Isaacson (1955-56), generiert am Computer Illiac I (siehe Abbildung 1.2) in der University of Illinois (USA).

Anmerkung: illiac suite

> *Lejaren Hiller's 'Illiac Suite' (1957) was the first time a computer was used to make musical decisions. It was the start of algorithmic composing and ancestor of computer interaction. Also other pieces by Hiller, among them 'An Avalanche for Pitchman', 'Prima Donna', 'Player Piano', 'Percussionist and pre-Recorded Playback', and it's wild! With Jan Williams, percussion, Robert Dick, flute, Nora Post, oboe, Royal McDonald, pitchman, Norma Marder, prima donna, and Robert Rosen, percussion.*[4] *(2)*

Es wird bei Algorithmischer Komposition historisch bedingt unter anderem zwischen "Aleatorik" und "Deterministik" unterschieden.

Bei der stochastischen Musik (Aleatorik) steht der Zufallsprozess im Mittelpunkt des Interesses. Dafür wurden viele Arten von Zufallsgeneratoren verwendet und den speziellen Anforderungen angepasst.

[4] Wergo(#WE124)

Ein Beispiel dafür ist das Werk "Idle Chatter" von Paul Lansky, welches Sprachmuster mittels Zufallsprozess variiert und damit Sprache erzeugt "ohne etwas zu sagen".

Anmerkung: Technical Notes - Idle Chatter

> *Idle Chatter was created on an IBM 3081 mainframe in 1985. just_more_idle_chatter and Notjustmoreidlechatter were made on a DEC MicroVaxII in 1987 and 1988, respectively. All three 'chatter' pieces use a technique known as Linear Predictive Coding, granular synthesis and a variety of stochastic mixing techniques. The Lesson was made on a MicroVaxII in 1989 using plucked string synthesis, comb filters and granular synthesis. Word Color and Memory Pages were made using a NeXT computer in 1992 and 1993 employing similar techniques. All pieces were written in a computer-music language called Cmix. The pieces were all digitally transferred for mastering so every copy of this CD contains the digital originals of these pieces: there are no copies. (This may not seem like a big deal, but it is.)*
>
> *Paul Lansky*[5] *(3)*

In der deterministischen Komposition wird der Computer vor allem eingesetzt, wenn eine große Menge von Daten, speziell in Form von Wiederholungen oder Variationen auftritt. Hier setzt auch das Gebiet der "Artifiziellen Intelligenz (AI) an, wobei Mutation- und Fraktaltheorie längst ein fixer Bestandteil davon sind. Dabei werden Systeme basierend auf neuronale Netzwerke, fuzzy logic und sonstige Automaten verwendet.

Beiden Richtungen gemeinsam ist es, Steuerdaten für weitere Verarbeitung zu generieren oder modifizieren und nur die wenigsten Programme generierten dabei Klänge.

Ein ganz spezielles Gebiet ist die "Echtzeitkomposition" mit Hilfe des Computers. Der Komponist oder das Kompositionsprogramm greift in "Echtzeit" in die Komposition ein und dirigiert die Partiturerzeugung (sinnbildlich). Damit verbunden ist auch die Entwicklung passender Eingabegeräte (Interfaces), wie zm Beispiel Dirigierstäbe und verschiedene Arten der Sensorik. Neueste Werke greifen auf Messwerte zurück und können als sogenannte Adaptive Kompositionen gelten.

[5] Auszug aus der CD New Computer Music 1987, WER 2010-50

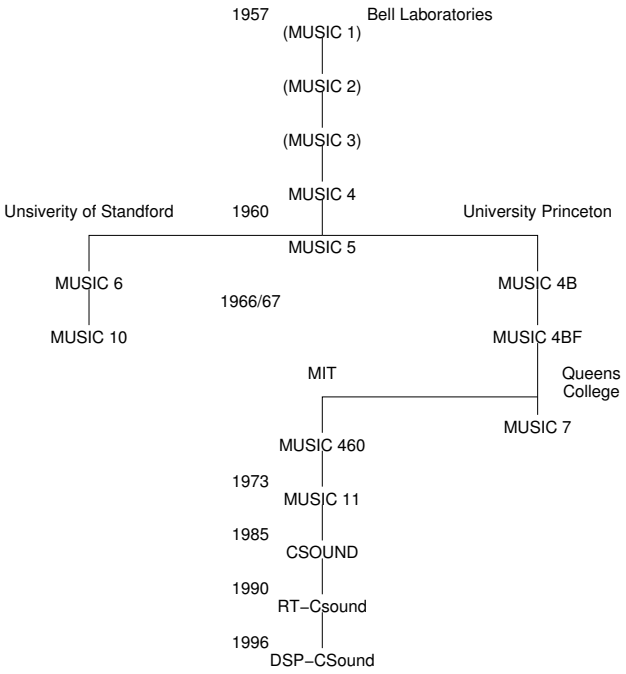


Abbildung 1.3: Stammbaum der frühen Computermusikprogramme

1.3 Sprachen in der Computermusik

Computermusik wurde zuerst mit allgemein üblichen Computern realisiert (Illiac). Später wurden spezielle Musikcomputer gebaut und als eigene Geräte implementiert (z.B.: ISPW). Diese Tendenz ist jedoch wieder rückläufig, da allgemein erhältliche Computer ausreichende Leistung erbringen und mittels Audio-Interfacekarten einfach für Computermusik ausgerüstet werden können. Der Schritt hin zum Echtzeit-Computer für den Live-Einsatz ist dann meist nur mehr eine spezielle Software-Zusammenstellung.

Die ersten Computermusiksprachen wurden in Amerika in den Bell Labratories entwickelt. Der erste allgemein verwendbare Computermusik-Compiler von Max V. Mathews 1953 war MUSIC 3. Dieser wurde von vielen Institutionen weiterentwickelt. Der in Abbildung 1.3 aufgelistete Stammbaum der Computermusik -Sprachen ist nicht vollständig sondern zeigt nur die Musix X Linie auf.

Music 11 wurde für die PDP-11 entwickelt. MUSIC-5 und MUSIC-4BF in FORTRAN IV geschrieben.

Abbildung 1.4: Computer PDP-11 an der Music-11 lief

Der Arbeitsablauf der Komposition am Computer war fast überall identisch und ist in Abbildung 1.5 dargestellt.

In weiterer Folge entstand "CMusic" und "CSound".

Ein entfernter Abkömmling, oder auch als Weiterentwicklung zu bezeichnen, wurde im IRCAM Paris erstellt und ist eine visuelle Programmiersprache namens "MAX" welche mittels des Programmes "fts" in Echtzeit ausgeführt wird und von Miller Puckette entwickelt wurde.

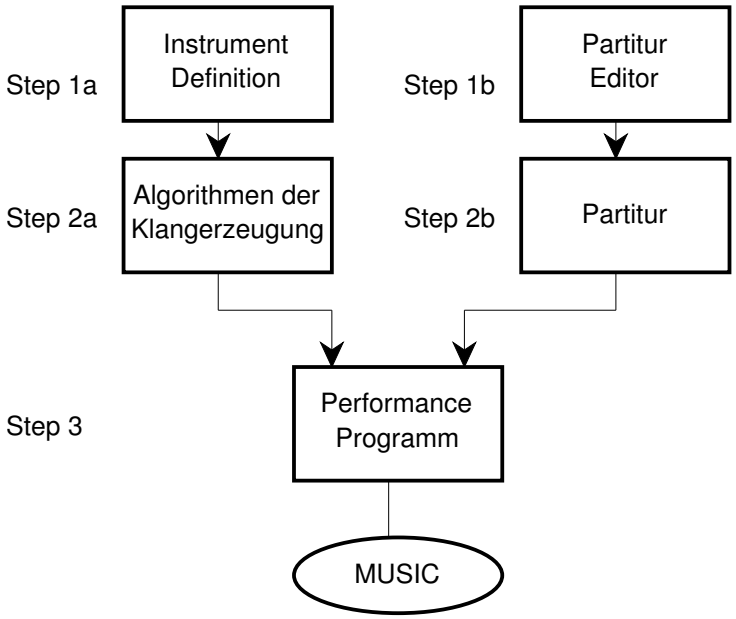


Abbildung 1.5: Prozess der Musikerstellung in MUSIC 4

Kapitel 2

Grundlagen

2.1 Struktur von Musik und Computer

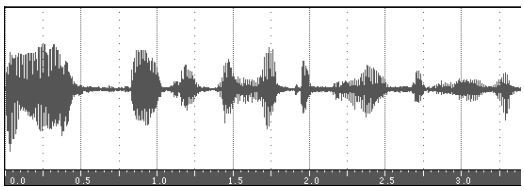


Abbildung 2.1: klassische grafische Audiodatendarstellung

Um Musik mittels Computer erzeugen zu können, muss zuerst ein mathematisches Modell gefunden werden, welches Musik repräsentiert.

2.1.1 Zeit-Amplituden Diagramme

Die einfachste Repräsentation ist eine Liste von diskreten Werten (Samples) als Audiosignal. Dies bedeutet, dass die gesamte Information in der Musik in einer Dimension zusammengefasst und über die Zeit abgebildet wird, den sogenannten Audiodateien. Eine übliche grafische Darstellung davon ist in Abbildung 2.1 dargestellt, welche der Signal-Darstellung an einem Oszilloskop entspricht.

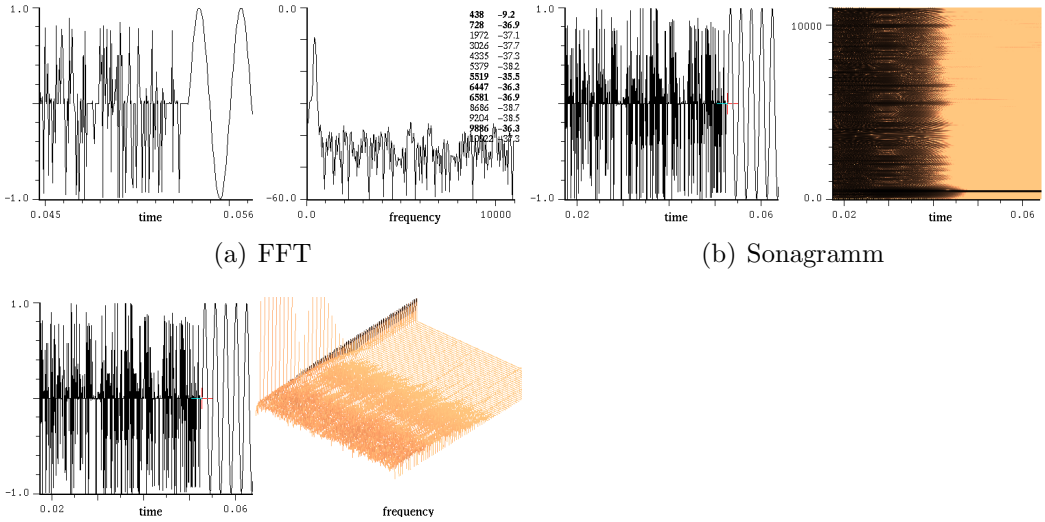
Hier lässt sich kaum etwas über die Frequenz- oder Phaseninformation des Signals sagen, allerdings ist das Zeitverhalten der Signale und damit ihrer Hüllkurven erkennbar, weshalb diese Darstellung eher zur Orientierung innerhalb von Audiodateien als zur Analyse des Inhalts dient. Tatsächlich wird bei den meisten Audio-Editoren die Hüllkurven-Information des Signals vorberechnet dargestellt und manchmal als eigene Datei abgespeichert. Erst bei großen Zoom-Werten wird das eigentliche Signal zur Darstellung verwendet.

2.1.2 Zeit-Frequenz Repräsentation von Audiosignalen

Weitere Repräsentationen sind Diagramme in der Frequenz-Zeit Domäne, wie das Spektrogramm, das Sonagramm oder die Waveletanalyse.

2.1.3 Parametrische Musik-Repräsentation

Da der Mensch jedoch durchaus zwischen verschiedenen Stimmen und Ereignissen in der Musik unterscheiden kann, was traditionell gesehen den Instrumenten und ihren Tönen entspricht, ist es von Vorteil entsprechend dieser Informationen die Daten zu zerteilen. Man spricht in der Literatur



(a) FFT

(b) Sonagramm

(c) Spektrogramm

Abbildung 2.2: klassische grafische Audiodatendarstellung

auch von parametrischer und nicht parametrischer Methode zur Musiksignal-Repräsentation[1]. Die Parametrisierung der Musik ergibt sich meist aus dem Produktionsablauf.

Die Parametrisierung kann in erster Linie durch Organisation in mehrere Spuren (Tracks) erfolgen, in denen das Musiksignal aufgeteilt ist. Dabei spricht man von den sogenannten Mehrspur-Systemen (Mehrkanalsystem). Sie werden auch als Hardiskrekorder (Abb. 2.3) bezeichnet. Hier ist als Verarbeitung der Audiodaten "Schneiden" (Reorganisation der Daten) und die Anwendung von "Filter" und "Effekten" möglich. Das Signal wird dabei in Richtung der Zeitachse, nicht aber in der "Vertikalen" (Schichten /Ebenen) bearbeitet. Spezielle Eingriffe wie zeitliche Veränderungen sind mit speziellen Algorithmen realisierbar. Die meisten Programme können die einzelnen Spuren oder Teile aus diesen parallel in einem getrennten Prozess, dem *Soundeditor*, bearbeiten, generell aber werden die Spuren über einen Mixer (virtuell im Computer oder am Mischpult eventuell computergesteuert) zusammengeführt.

Weiters gibt es verschiedene Analyseverfahren, die ein Musiksignal in pa-

[1]siehe S.3, "Representation of Musical Signals", edited by Giovanni De Poli, Aldo Piccialli, Curtis Roads, MIT Press, 1991

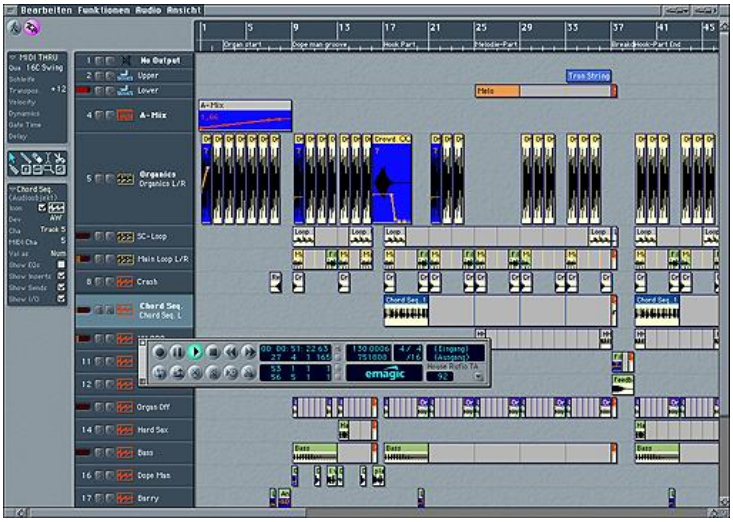


Abbildung 2.3: Harddiskrekorder Audio und Steuerinfo

rametrische Daten verwandelt und bestimmten Zwecken, nicht zuletzt der Kompression von Audiodaten, dient. In der Computermusik wurden vor allem Linear Prediction Coding und Formantenanalyse verwendet. Das Ergebnis sind Listen von Parametern aus welchen wiederum das Musiksignal mehr oder minder genau zusammengesetzt werden kann.

Um noch mehr Kontrolle über die Komposition selbst zu erhalten ist es sinnvoll eine weitere Aufspaltung in Steuerinformationen und Signal Produktion vorzunehmen, wobei eine genau definierte Schnittstelle zwischen diesen Teilen erforderlich ist. Der Schnittpunkt ist von System zu System verschieden. Es ergibt sich dadurch eine Aufteilung in die Definition der Signalgeneration und eine Repräsentation der Musik als Steuerdaten für die Signalgeneration.

Eine der ersten Formen der symbolischen Musikrepräsentation ist die Notenschrift, bei der die Musik als Ereignisse notiert ist. Diese Ereignisse können für den Computer als Parameter in cmusic-Notation oder gleich in der Programmiersprache C notiert werden, wie sie in Abbildung 2.5 und 2.6 sehen können. Bei Steuerdaten wird of eine grafische Notation verwendet, so unter anderem von Stockhausen in seinen ersten Elektronischen Arbeiten.

Zur Darstellung von Steuerdaten kann zwischen zwei verschiedenen Grundmodellen unterschieden werden, dem "Klangkontinuum" und dem "Ereignismodell" (engl. event model).

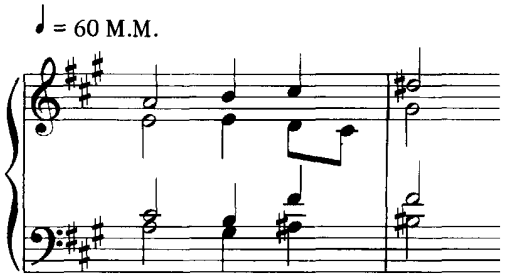


Abbildung 2.4: Notenschrift als Modell der Ereignis-Notation

```
48,2 50,1 52,1  54,2              (soprano notes)
43,2 43,1 41,.5 40,.5 47,2        (alto notes)
40,2 38,1 45,1  45,2              (tenor notes)
36,2 35,1 37,1  39,2              (bass notes)
```

Abbildung 2.5: Noten in C notiert

Diese unterscheiden sich in der Repräsentation Daten:

Ereignismodell: $< Zeit >< Wert >, < Zeit >< Wert >, ..., < Zeit >< Wert >$

Datenflußmodell: $< Samplerate >:< Wert >, < Wert >, ... < Wert >$

Durch die Entwicklung der digitalen Synthesizer seit Mitte der 80er Jahre gab es aus der Definition von MIDI heraus eine Tendenz zur Verwendung des Ereignismodells, wobei diese Tendenz noch durch die Verwendung von sogenannten Sequenzer-Programmen verstärkt wurde. Ein typische Sequenzer ist der "Emagic Logic" (Abb. 2.7) der fast identisch zu "Steinberg Cubase" verschiedene Formen der Notation von Ereignissen bietet, jedoch immer auf der Ereignisebene arbeitet. So wurde Anfang der 90er Jahre angefangen, die Schwächen des Ereignismodells zu kaschieren und durch spezielle Notationsformen auszugleichen, wie etwa Kontrolldaten-Verläufe. Dies war vor allem bei Automatisierung von Mischpulten wichtig. Wirklich gelang das nur durch Verwendung von Audiodateien und Harddiskrecording (HDR).

HDRs sind Simulationen von Bandmaschinen, Mischpulten, Effektgeräten, die automatisierbar sind, und ein paar zusätzlichen Vorteilen, die man vom Wechsel in die digitale Ebene bekommt, wie dem Springen in der Zeit, das Verschieben von Spuren entlang der Zeitachse, etc.

```
#include <stdio.h>
#include <math.h>
/*
 * note definitions
 */
struct {
    int   pitch ;
    float dur ;
} chorale[4][6] = /* "Es ist genug"--J.S.Bach (first phrase) */
{
    48,2., 50,1., 52,1., 54,2., -1,0., -1,0.  , /* soprano notes */
    43,2., 43,1., 41,.5, 40,.5, 47,2., -1,0.  , /* alto notes */
    40,2., 38,1., 45,1., 45,2., -1,0., -1,0.  , /* tenor notes */
    36,2., 35,1., 37,1., 39,2., -1,0., -1,0.    /* bass notes */
} ;
```

Abbildung 2.6: Noten in C notiert

Eine weitere Abstrahierung ergab sich aus der Sicht der "Musik als Prozess". Dies bedeutet, dass Musik als Definition von Prozessanweisungen geschrieben wird und damit algorithmische Musik in Reinform entwickelt wurde. Dazu beigetragen haben Echtzeit-Programme, deren Notation in grafischer objektorientierter Form stattfand und eine Komposition in Form einer Art Regelungstechnik-Diagramms dargestellt werden kann. Der Einsatz ergab sich vor allem bei automatisierten Klang-Installationen und "unendlich" langen Kompositionen für spezielle Zwecke.

Als ein frühes Beispiel des Prozess-Denkens könnten die Werke von Alvin Lucier gelten, speziell das Stück "I am sitting in a room", wo er mittels eines Raummikrofons über ein Delay rückgekoppelt und den Raum mit einen Satz zum Schwingen bringt (Damals nicht live, da technisch noch nicht möglich).

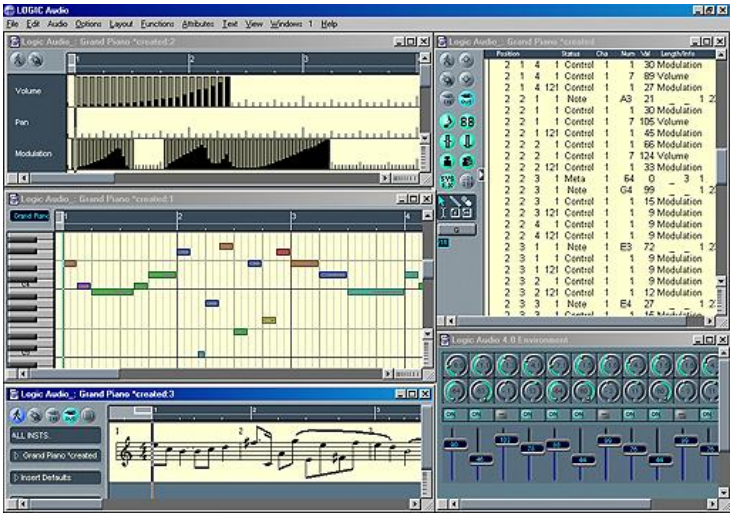


Abbildung 2.7: Sequenzer mit MIDI-Daten: Emagic Logic

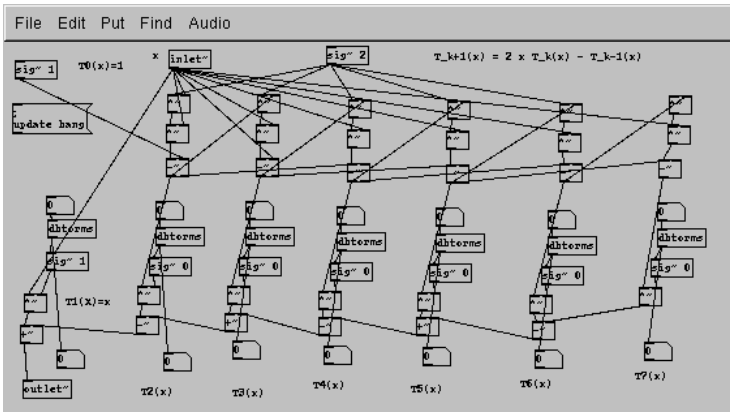


Abbildung 2.8: Prozess-Notation mit Pd

2.2 Komposition und Computer

Der Weg der "Computer Assisted Composition" (CAC) oder "Computer Mediated Composisition" (CMC) wird von Richard Moore mit dem Flussdiagramm in Abbildung 2.9 beschrieben.

nun in Vorlesung Algoritmische Komposition, hier ein paar Schagworte:

- Algorithmen
- Aleatorik
- Interaktives Komponieren
- markov-ketten,...
- neuronale Netzwerke...
- state machines,...,zelluläre Automaten
- fuzzy logic

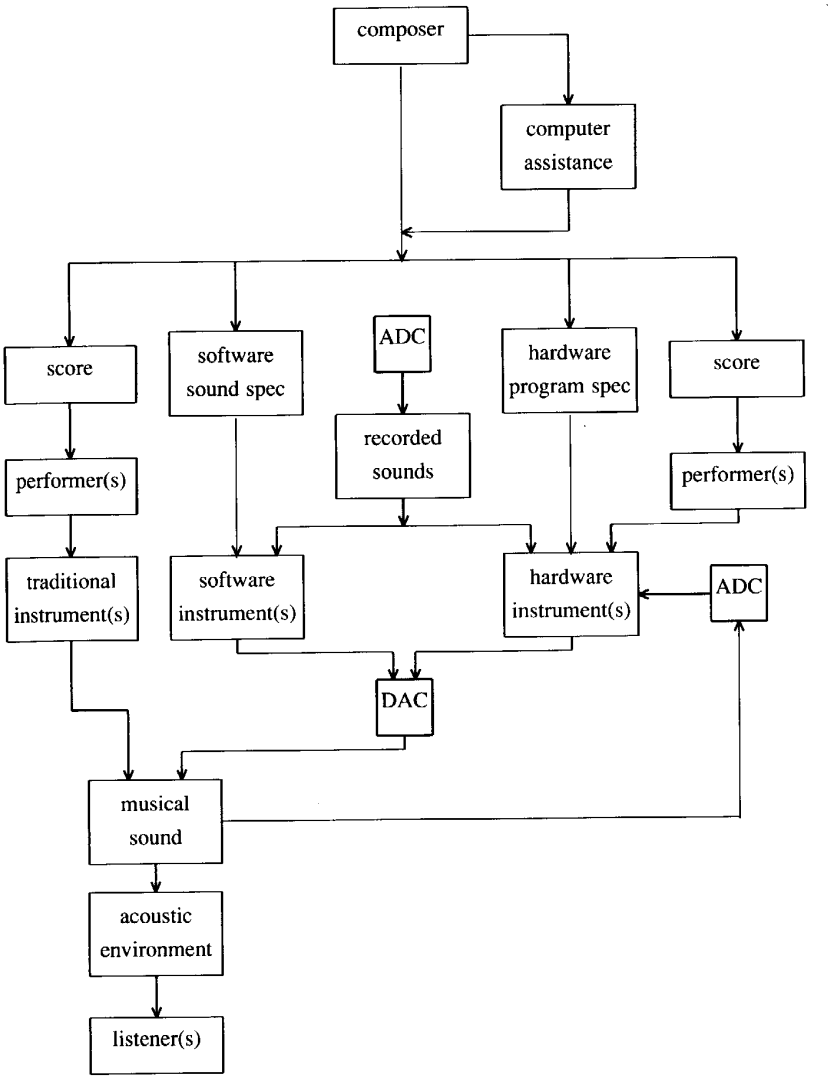


Abbildung 2.9: Computer Mediated Composition - Flussdiagramm laut Richard Moore 1993

Kapitel 3

Techniken der Steuerung

Begriff: **Steuerungstechnik**

> *Steuerungstechnik ist die Technik von Einrichtungen, die nach einem vorgegebenen Plan technologische Prozesse beeinflussen. Wenn deren Zweck durch Steuerung selbsttätig erfüllt wird, handelt es sich um Automatisierung.* *(1)*

Vereinfacht betrachtet, beeinflussen Steuerungstechnische Einrichtungen, auch Steuerungen genannt, Prozessabläufe mit Hilfe von Informationen, die sie im Sinne des Prozess-Ziels verarbeitet haben. Teile der Steuerungstechniken sind Sensoren, Informationswandler, Informationsübertrager, Steuergeräte und Aktoren. Wird das Ergebnis des Aktors wiederum durch Sensoren erfasst und verarbeitet, so wird die Steuerung zur Regelung (Regelungstechnik).

Die Steuerungstechnik in der Musik entwickelte sich traditionell mit dem Instrumentenbau, sodass zum Beispiel beim Klavier und Orgelbaus komplexe mechanische Steuerungen erfunden wurden, wobei ein Hauptziel, die Anpassung der Steuerung an den Musiker im Mittelpunkt des Interesses war. So ist die Klaviatur eine fast schon genormte Einheit zur Steuerung von Klavieren, welche exakten Abmessungen entspricht. Die Randbedingungen. Steuerungstechniken haben damit als Randbedingungen für Anforderungen und Verwendbarkeit meist die Wahrnehmungs- und Aktionsgrenzen des Menschen.

Mit dem Auftauchen von Synthesizer wurden dann erste Protokolle und Spezifikationen für Steuerungen von den Synthesizer-Hersteller Moog entworfen, die sich als de facto Standard ergab. Bei digitalen Steuerungen wurde es schon schwieriger sich auf gemeinsame Spezifikationen zu einigen, siehe MIDI.

In der heutigen Zeit, wo der Computer praktisch alle Bereiche der Elektronischen Musik übernommen hat, werden zunehmend Standardschnittstellen des Computers zu Steuerungsmethoden in der Musik, von USB, Firewire über Ethernet und Wireless-Lan Standard.

In den folgenden Kapiteln werden wir auf die einzelnen Steuerungstechniken eingehen, wie sie sich historisch entwickelten haben.

3.1 Analogsteuerungstechnik

Realisiert wurde die Analog-Steuerungstechnik hauptsächlich mittels Spannungssteuerung (engl. Voltage Control,VC) und bezeichnet das Verfahren Synthesizer Module durch Spannung zu steuern.

Analogssteuerungen waren die ersten Elemente in der Elektronischen Musik die definierte Schnittstellen hatten. Der bekannteste Standard zur Spannungssteuerung war Robert A.Moog eingeführt worden, dessen Festlegungen von praktisch allen anderen Analog-Synthesizer übernommen wurden.

Es gibt zwei wesentliche Größen die bei den Synthesizern definiert gesteuert werden konnten:

- Frequenzen
- Lautstärken

Alle anderen Größen wie Güte bei den Filtern hatten keine normierten Spannungen und wurden meist so implementiert, das 0-10V den gesamten verstellbaren Bereich umfasst.

Da die Steuerung prinzipiell über Spannungen funktionierte, wurde eine Spannungsanpassung vorausgesetzt, dies bedeutet hochohmiger Eingang, niederohmiger Ausgang (keine Leistungsanpassung).

3.1.1 Amplitudensteuerung

Um einen entsprechenden Bereich abzudecken, wurde von Moog eine Steuerspannungsbereich von 0-10 V gewählt, der einem Bereich von 100 dB entsprach, also 10 dB/V.

Die JND[1] gerade noch wahrnehmbare Differenz bei Amplitudenhören ist ca. 1 dB (0.2–0.4 dB bei optimalen Bedingungen), so erhalten wir eine Auflösung von 0.1 V, was sehr genau elektronisch verarbeitet werden kann.

Wird nun diese Steuerspannung von einen Computer geliefert, so braucht dieser einen D/A-Wandler, was bei einen Bereich von 100 dB sozusagen eine Auflösung von 1/100 bedeutet. Es reichen daher relativ unkritische 8-Bit Wandlern mit 256 Werte, wobei 1/2 Bit Rauschen hineingerechnet wird.

[1] just noticeable difference

3.1.2 Frequenzsteuerung

Hier ergibt sich bei 10 V Steuerbereich und der Definition von 1V/Oktave (Moog) ein Bereich von 10 Oktaven. Dabei wurde noch fixiert, dass 0 V einer Frequenz von 20 Hz entsprechen:

U (V)	f (Hz)
0	20
1	40
2	80
...	...
10	20.480

Wir erhalten dies aus der Formel:

$$f = 20\ 2^U Hz$$

U ... Spannung in Volt, f ... die Frequenz in Hertz

Damit wurde der gesamte benötigte Frequenzbereich abgedeckt. Wird nun eine JND von 1 % angenommen, so muss z.B.: bei 100 Hz ein Unterschied von 1 Hz noch unterschieden werden können, das zu einer Auflösung der Gleichung nach U führt:

$$U = \frac{\log(f/20)}{\log(2)} V$$

$\Delta f = 101Hz - 100Hz$:

$$\Delta U = \frac{\log(100/20) - \log(101/20)}{\log(2)} V = 0.0144V$$

Dies bedeutet bei einem Bereich von $10V$ ca. 700 Schritte Auflösung und somit bei einer 1/2 Bit Genauigkeit eines D/A-Wandlers 11 Bit. Verwendet wurden meistens 12 Bit Wandler, vor allem um genaue Mikrotonalitäten zu gewährleisten und bei extremen Frequenzen Schwebungen möglichst zu vermeiden. Für die Steuerung von frequenz-genauen Schwebungen höherfrequenter Töne reicht diese Auflösung jedoch nicht aus.

3.2 MIDI

MIDI , die Abkürzung für 'Musical Instrument Digital Interface', ist eine digitale Schnittstelle zur Steuerung von elektronischen Geräten und wurde speziell zur Steuerung von Musikinstrumenten entwickelt.

3.2.1 MIDI Spezifikationen

Die Spezifikationen wurden von zwei Gruppierungen, der MMA[2] und der JMSC[3], festgelegt. Dabei handelt es sich hauptsächlich um Vertreter von Firmen, die in diesen Gesellschaften sitzen. Davon ist eine für Japan und die zweite für den Rest der Welt zuständig. Es wurde eine gemeinsame Spezifikation, die hier als Grundlage dient, von der IMA[4] in Einverständnis mit beiden Komitees herausgegeben.

Dabei ist die Bezeichnung MIDI 1.0, Version 4.1 der aktuelle Stand. Aufwärtskompatibilität wird nur innerhalb von MIDI 1.0 garantiert. MIDI-Spezifikationen abweichend vom Standard 1.0 sind, von der Hardware und dem Datenformat her nicht unbedingt kompatibel.

Diese Spezifikationen bestehen aus einer Beschreibung der Hardware, des Datenformat und der zugehörigen Syntax. In ihr sind sämtliche gültige Befehlsformate und deren Verwendung angeführt. Von einer genaueren Beschreibung davon möchte ich hier aber Abstand nehmen und dabei auf die Spezifikation verweisen [**?**].

MIDI, ursprünglich für Live-Aufführungen gedacht, hat sich aber inzwischen auch im Studio-, Videobereich und anderen Sparten durchgesetzt. Es macht es möglich Informationen, wie "Note ein/aus", "vorgefertigte Einstellung auswählen", Ausdrucksveränderungen, u.s.w , zwischen verschieden Musikinstrumenten und anderen Geräten, wie Sequenzer, Computer, Lichtsteuerungen, Mischpulten, u.s.w., auszutauschen. Es besteht oft auch noch die Möglichkeit den internen Dateninhalt eines Gerätes zu übertragen.

Die Verbreitung dieses Interfaces übertraf alle Erwartungen. So kam kaum mehr eine neues elektronisches Gerät für elektronische Musik auf dem Markt, welches nicht diesen Anforderungen entsprach und dies schon bei sehr billigen Geräten und die meisten Computer sind damit ausrüstbar.

Eine große Verbreitung ergab sich im sogenannten "Homerecording-Bereich" und dies meist im Zusammenhang mit Klein-Computern. Einer der

[2] MIDI Manufacturers Association
[3] Japan MIDI Standards Committee
[4] International MIDI Association, Los Angeles, CA 90056 USA

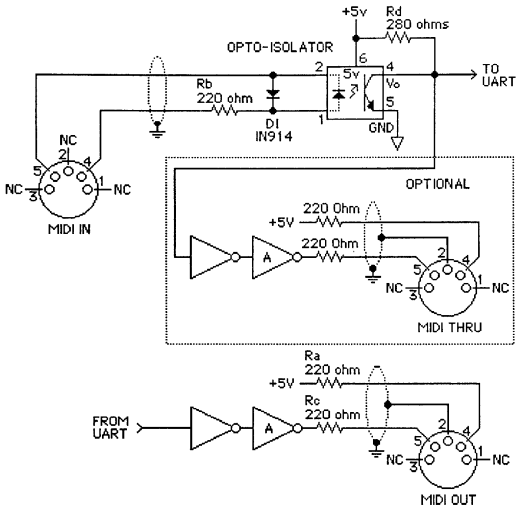


Abbildung 3.1: MIDI-Hardware laut Spezifikationen 1.0

ersten Computer mit fixer MIDI-Schnittstelle war der Atari ST und setzte sich im europäischen Bereich Anfang der 90er Jahre in den Studios durch.

Doch mit dieser Ausbreitung wurden auch die Anforderungen höher. Dies wurde vor allem durch das immer breiter werdende Software-Angebot noch gesteigert. Dazu zählen vor allem Editoren für Synthesizer und Sequenzer.

Dadurch tauchten jedoch immer mehr Probleme bei der Handhabung dieser Netzwerke auf. Dies einerseits bei den Verschaltungsmöglichkeiten, andererseits in der Auslastung der Kanäle.

Ein weiterer Bereich war auch die Entwicklung von verschiedenen Steuergeräten und Sensoren für MIDI-Geräte.

Eine Weiterentwicklung der Spezifikationen ergab nicht MIDI V2.0, wie viele erhofften, jedoch eine genauere Zuordnung von Programm zu Klängen und Controllern, welche General MIDI genannt wurden und mittlerweile (Nov.99) den Stand General MIDI 2.0 erreichten.

Hardware

Die in den MIDI Spezifikationen 1.0 vorkommenden Muster-Schaltung ist in Abbildung 3.1 dargestellt und muss folgende Anforderungen erfüllen:

3 Anschlüsse: MIDI-IN, MIDI-OUT, MIDI-THRU (5-poliger DIN-Stecker)

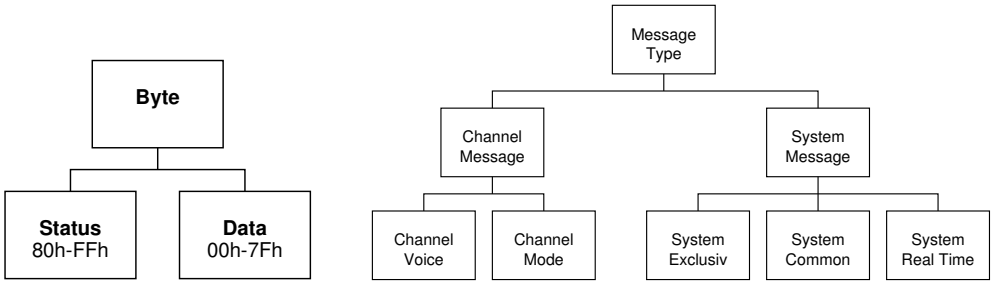


Abbildung 3.2: MIDI-Hardware laut Spezifikationen 1.0

Eingang Galvanisch getrennt (Optokoppler mit Anstiegszeit min. 2 usec.)

Seriele Übertragung: 5 mA Stromschleife (z.B.: TTL-Treiber)

Übertragungsrate : 31.25 kBaud (+/- 1%)

Datenformat : 1 Start-, 8 Daten-, 1 Stopbit, kein Parity

Übertragungstrecke : 2-pol. (gechirmtes) Kabel, 15m max.

Syntax Definition

Bei MIDI werden alle Daten in 8 Bit (Bytes) Zahlen übertragen. Dabei unterscheidet man prinzipiell mit dem höchstwertigen Bit zwischen Status- und Datenbyte (Abb. 3.2). Eine weitere Unterscheidung gibt sich beim Typ der Befehle (messages) in Kanal bezogene Befehle (channel messages) und System bezogene Befehle (system messages). Erstere werden als Modus-Befehle (mode messages) oder Kanal-Befehle (voice messages) gesendet. Zweitere teilen sich in System-Exklusiv Befehle (system exclusiv messages), gemeinsame (system common message) und Echtzeit-Befehle (realtime messages).

Es sind alles Ereignis bezogene Befehle und besitzen 1 Statusbyte und keine oder mehrere Datenbytes. (Abb. 3.3). Bei den Kanalbefehlen gibt es 16 Kanäle (4 Bit) zur Auswahl, welche im Statusbyte inkludiert sind.

Das Statusbyte bestimmt (Ausnahme Realtime-Messages) den Status und muss daher nur einmal für mehrere aufeinander folgende Befehle mit gleichen Status gesendet werden. Dies wird als *Running-Status* bezeichnet.

Realtime-Messages haben keine Datenbytes und verändern den Status nicht. Deshalb können diese zwischen Datenbytes anderer Befehle eingeschoben werden.

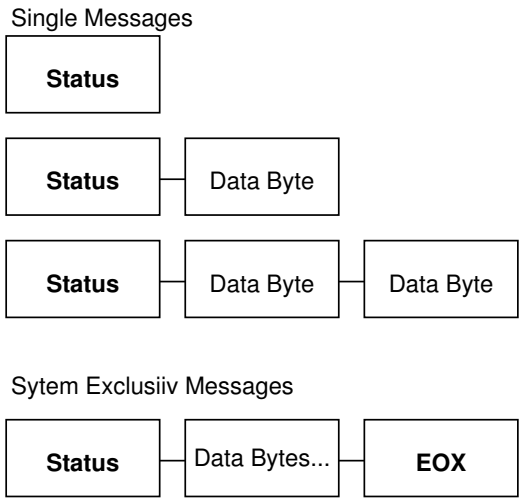


Abbildung 3.3: MIDI-Hardware laut Spezifikationen 1.0

Der Informationgehalt der Daten ergibt aus:

1. Statusbytes bzw. aktuellen Status (Running Status)
2. Datenbytes
3. Zeit des Auftretens !

Die Tabellen mit den einzelnen Befehlen sind im Anhang aufgelistet.

Controllers

Syntax: $Bn < controlnumber >< value >$

Die Controller werden mittels Nummern unterschieden und teilen sich in mehrere Gruppen:

Ctrl.Nummer	Bedeutung
0..31	MSB, kontinuierliche Controller
32..64	LSB, kontinuierliche Controller
65..95	zusätzliche Single-Byte Controller
96..102	Incr.,Decr Kontrolle über Parameter
102..120	nicht definiert

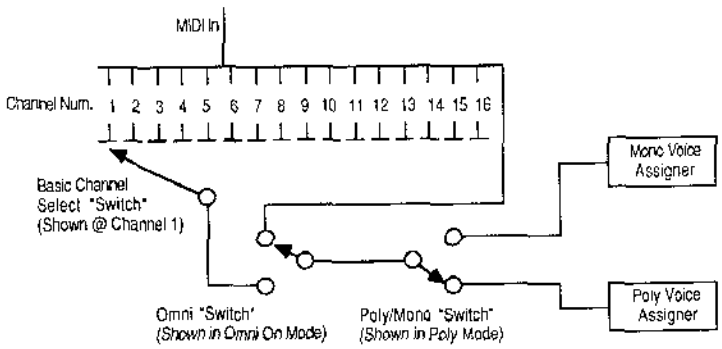


Abbildung 3.4: Moden für MIDI-Geräte

Eine Ausnahme sind die Nummern 121..127, welche System-Modus sind.

Probleme gibt es mit den MSB+LSB Summen beim kontinuierlichen Steuersequenzen beim Übergang zwischen zwei MSB-Werten, da es hier zu Sprüngen kommen kann.

Betriebs-Modi

System Exclusiv Messages

Sie beginnen mit $F0h$ und enden mit $F7h$. Dazwischen dürfen als Daten nicht mehr als 256 Bytes liegen. Weiter gibt es ein Software-Handshake Protokoll um größere Datenmengen mit kleineren Paketen zu übertragen.

Ein Teil des Sysex-Befehle sind fix definiert, wie Sample Dump, Time Code usw. und der Rest wird anhand von Hersteller ID-Nummern unterschieden (siehe Anhang MIDI-Tabellen)

MIDI Clock

Der MIDI Clock wird aus System-Common Messages gebildet und bietet damit Synchronisation zum Beispiel von Drum-Computer, Sequenzer usw. auf Basis des (Noten-) Tempos an. Es werden keine absoluten Zeitwerte übertragen, jedoch kann mittels start, stop und continue auch eine Ablaufsteuerung erfolgen.

Die "Clocks" $F8$ werden mit 24 Schläge per Viertelnote übertragen.

Zur Verwendung dieses "Clocks" gibt es Konverter, welche diese in ein Audiosignal konvertieren und sich damit der Clock zum Beispiel auf eine Ton-

bandspur übertragen lässt. Von dort auch wieder abgespielt, kann er andere MIDI-Geräte synchronisieren.

MIDI Time Code

Der MIDI Time Code basiert auf dem SMPTE Code und lässt sich aus einer Kombination von System Common Messages und System-Exclusiv Messages übertragen. Dabei wird die Frame-Rate (24, 25, 30Drop, 30 - Frame/sec) in vier Teile geteilt, wodurch eine Auflösung von 10ms entsteht. Dieser wird mit den $F1$-Messages codiert, wobei innerhalb von 8 Messages der gesamte Timecode übertragen wird. Alternativ kann unter anderem eine "Full Message" mittels SYSEX gesendet werden.

3.2.2 Probleme und Grenzen von MIDI

Mit dem Ziel MIDI für Steuerungen einzusetzen, ergibt sich das Modell eines einfachen Steuermodell, wie in Abbildung 3.5 gezeigt.

Bei der Vernetzung von Geräten mit MIDI kann ein Sender an mehrere Empfänger geschaltet werden, indem die MIDI-Thru-Buchse des ersten Empfängers auf die MIDI-In-Buchse des nächsten geschaltet wird. Dies ist in Abbildung 3.6 ersichtlich und wird als "Daisychain" bezeichnet.

Wenn mehrere Ausgänge auf einen Input geschaltet werden muss, wie im Beispiel 3.7 gezeigt, muss dieses Gerät zwangsweise mehrere Inputs besitzen oder es wird wie in Abbildung 3.8 gezeigt ein "MIDI-Merger" verwendet.

Das Problem dabei ist, dass ein Merger der mehr als zwei MIDI-Eingänge auf einen Ausgang mischt, nicht garantieren kann dass alle Daten geschickt werden können, da es zu einer Daten-Auslastung über 100% kommen kann. Hier ist eine gute Strategie des Merger notwendig die richtigen Daten wegzulassen und somit "Notenhänger" oder ähnliches zu vermeiden.

3.2.3 MIDI Parser

Der MIDI-Parser ist eine sogenannte "Zustandsmaschine" (State-Machine), da er abhängig vom aktuellen Zustand reagiert. Als Zustand wird der Status und die aktuelle Nummer des Datenbytes verwendet.

Das Flussdiagramm in Abbildung 3.9 zeigt dem im MIDI-Standard vorgeschlagenen Parser.

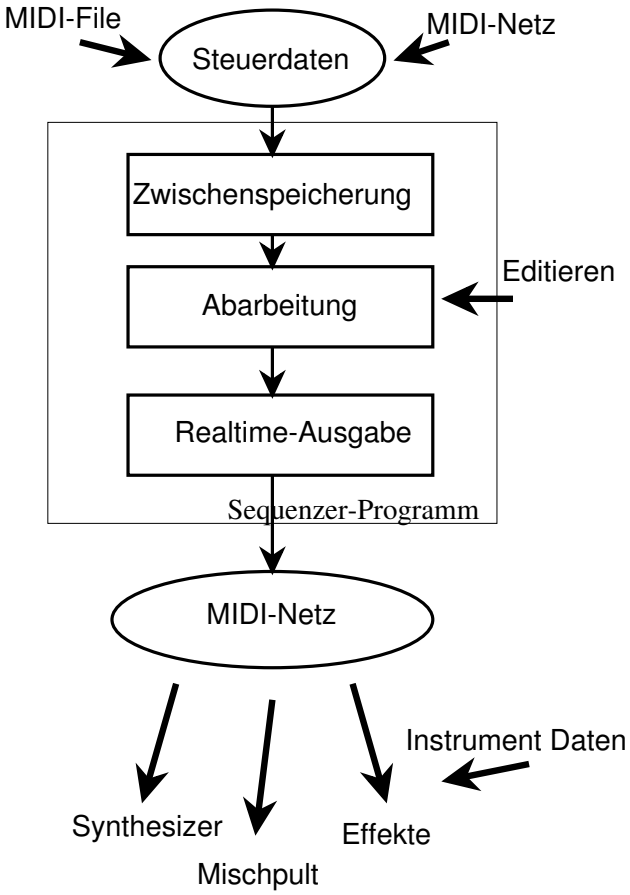


Abbildung 3.5: Einfaches Steuer-System für Elektronische Musik

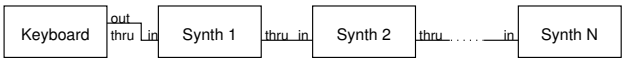


Abbildung 3.6: Einfaches MIDI-Netzwerk

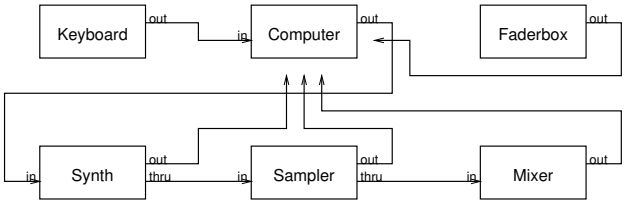


Abbildung 3.7: MIDI im Studio, einfache Verschaltung

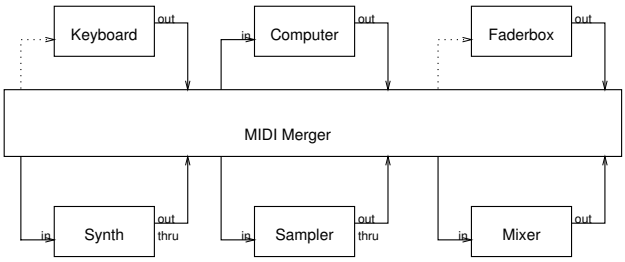


Abbildung 3.8: Einsatz von Merger für bessere Vernetzung

3.2.4 MIDI Files, Implementation

Midi Files ist ein Standard zur Speicherung und Übertragung von MidiInformation welche hauptsächlich auf Sequenzer bezogen ist. Dabei werden die Daten in Form von "Tracks" (Spuren) organisiert. Es werden drei File-Typen unterschieden, welche sich durch die Bedeutung ihrer "Tracks" unterscheiden:

Format 1: Ein Track mit allen Kanälen zugleich.

Format 2: Mehrere parallele Tracks für verschiedene Kanäle.

Format 3: Mehrere sequentielle Tracks mit jeweils allen Kanälen.

Eine skizzenhafte Zusammenfassung ist in Abbildung 3.10 dargestellt.

3.2.5 General MIDI

General Midi ist eine Beschreibung der Bedeutung einzelner Parameter mit einer fixen Vorgabe für Instrument-Zuordnungen zu den Programm Nummern und Kanäle. Es dient dazu das Midi Files richtig klangmässig interpretiert werden können. Eine erste Vereinbarung erfolgte 1991 durch die IMA als GM 1 (General MIDI Level 1).

> *IMA, GM 1]*
>
> *GM1 Device Features*
>
> *To be GM1 compatible, a GM1 sound generating device (keyboard, sound module, sound card, IC, software program or other product)*

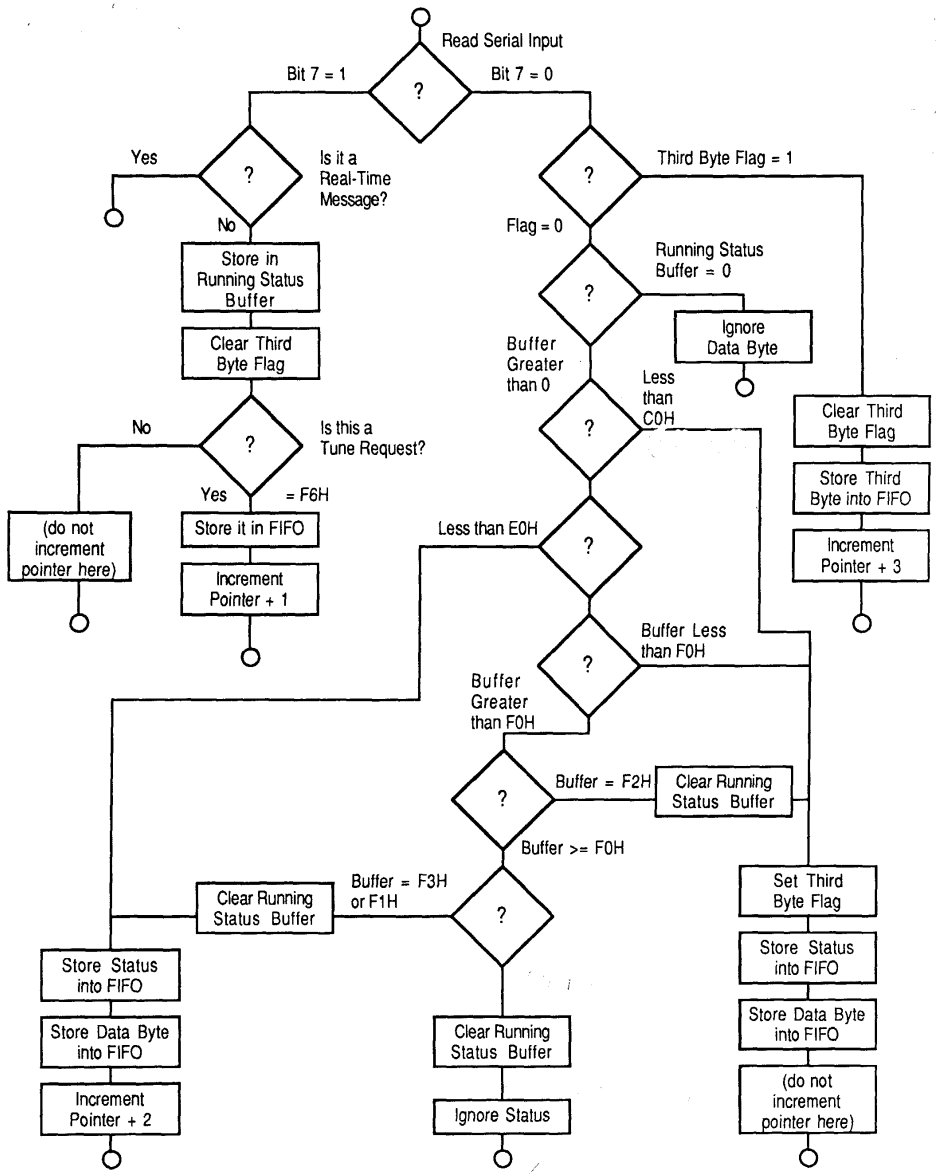


Abbildung 3.9: MIDI-Parser Fluss Diagramm Empfohlen von IMA

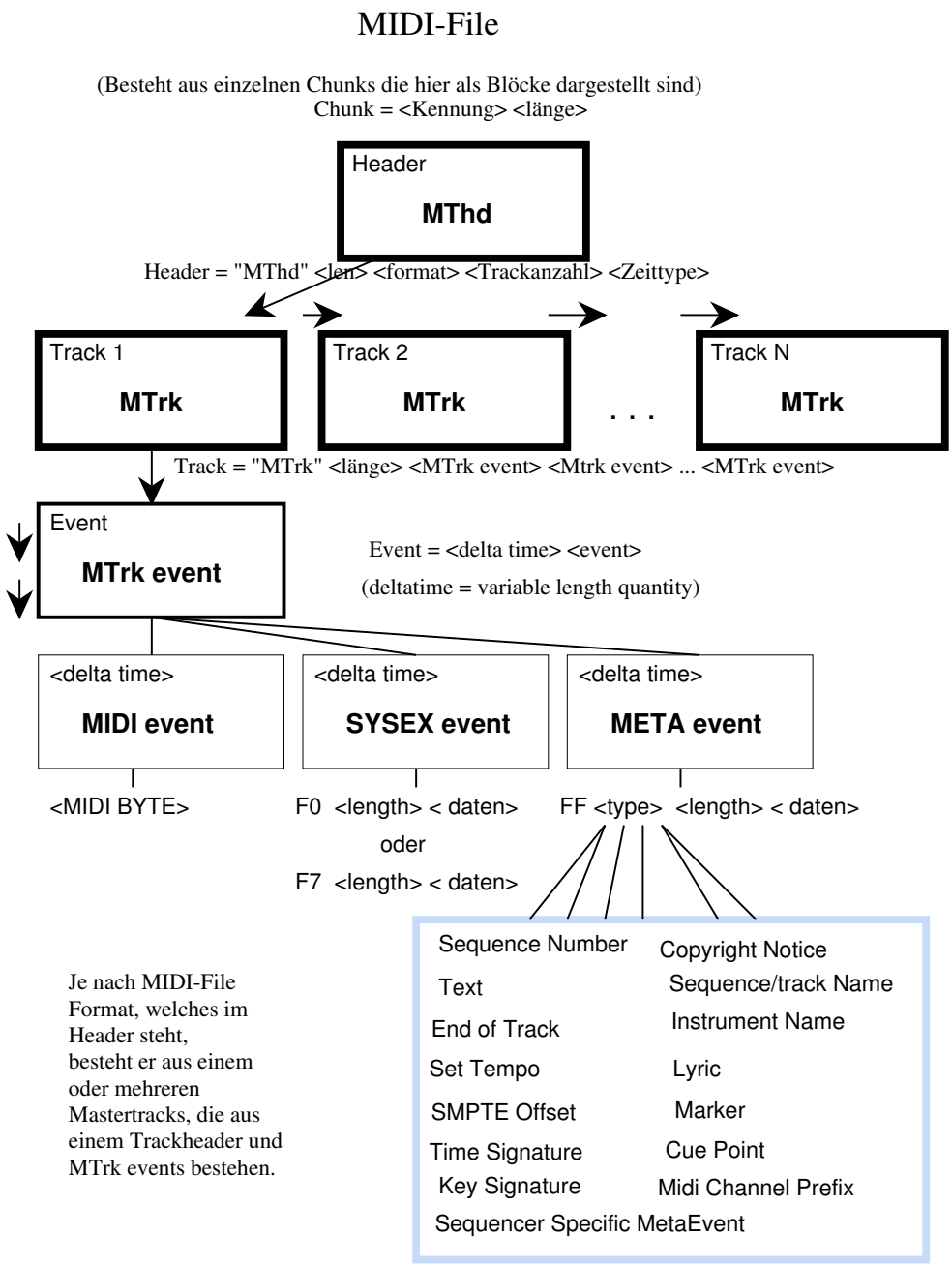


Abbildung 3.10: MIDI File Format

must meet the General MIDI System Level 1 performance requirements outlined below, instantaneously upon demand, and without additional modification or adjustment/configuration by the user.

Voices: *A minimum of either 24 fully dynamically allocated voices are available simultaneously for both melodic and percussive sounds, or 16 dynamically allocated voices are available for melody plus 8 for percussion. All voices respond to velocity.*

Channels: *All 16 MIDI Channels are supported. Each Channel can play a variable number of voices (polyphony). Each Channel can play a different instrument (sound/patch/timbre). Key-based percussion is always on MIDI Channel 10.*

Instruments: *A minimum of 16 simultaneous and different timbres playing various instruments. A minimum of 128 preset instruments (MIDI program numbers) conforming to the GM1 Instrument Patch Map and 47 percussion sounds which conform to the GM1 Percussion Key Map.*

Channel Messages: *Support for continuous controllers 1, 7, 10, 11, 64, 121 and 123; RPN #s 0, 1, 2; Channel Pressure, Pitch Bend.*

Other Messages: *Respond to the data entry controller and the RPNs for fine and course tuning and pitch bend range, as well as all General MIDI Level 1 System Messages.*

[*(1)*

1999 wurde dann GM 2 veröffentlicht, welche GM 1 ersetzt und eine Erweiterung von Controllern und Klangprogrammen darstellt. Zusätzlich wurde 2003 ein Update auf GM 2 V1.1 vorgenommen, womit zusätzlich Panorama-Kurven und verschiedene Stimmungen definiert wurden. GM 1 ist voll aufwärts-kompatibel auf GM 2.

Eine Übersicht in Anhang A.2.2 aufgelistet.

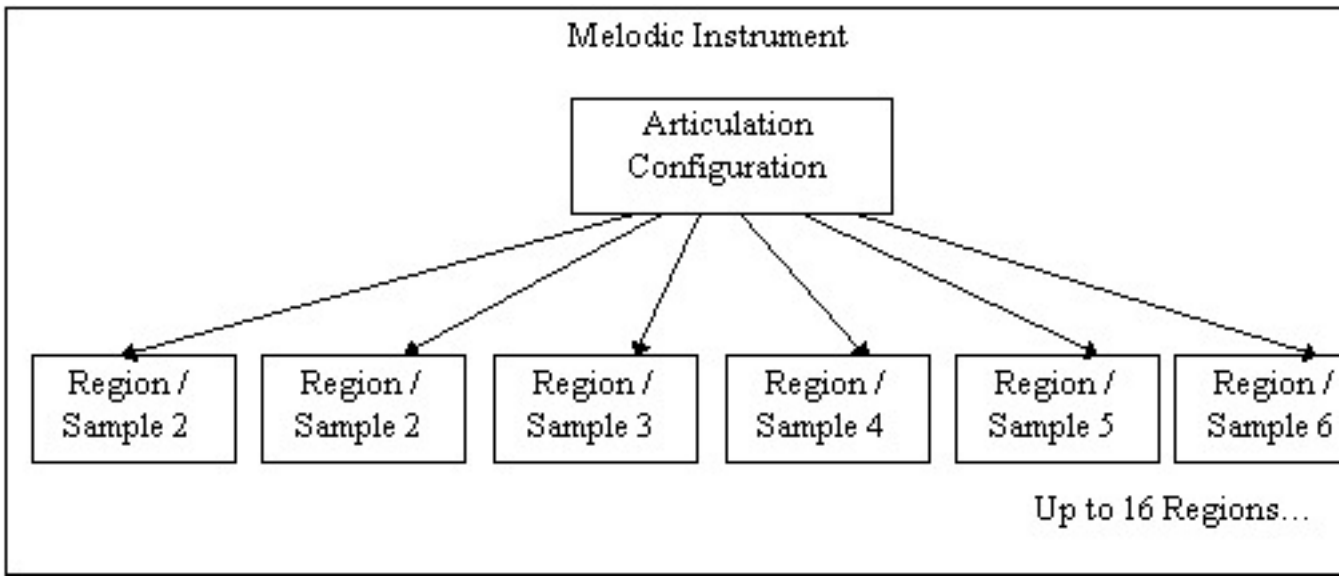


Abbildung 3.11: MIDI Downloadable Sample Instrument

3.2.6 DLS

Unterstützend dazu wurde in den letzten Jahren eine DLS-Specification (Downloadable Sound) durch die MMA ausgegeben, welche Softwaresynthesizer zum Rendern von MIDI-Files, fixieren sollte, bzw. Soundkarten oder Synthesizer speziell mit diesen Möglichkeiten auszurüsten, sodass MIDI-Files als eine Art komprimierbarer Soundfiles benützt werden können.

Dabei gab es nach DLS 1 1997, DLS 2 2000 und ein Update 2006 auf DLS 2.1 .

Weiters wurden dann noch Mobile DLS 2004 für Mobiltelefone eingeführt.

Eine Übersicht in Anhang A.2.3 aufgelistet.

3.2.7 Beispiel: Programm MIDI-File Leser

In Abbildung 3.15 ist der Skizzierte Aufbau eines MIDI-File Players dargestellt.

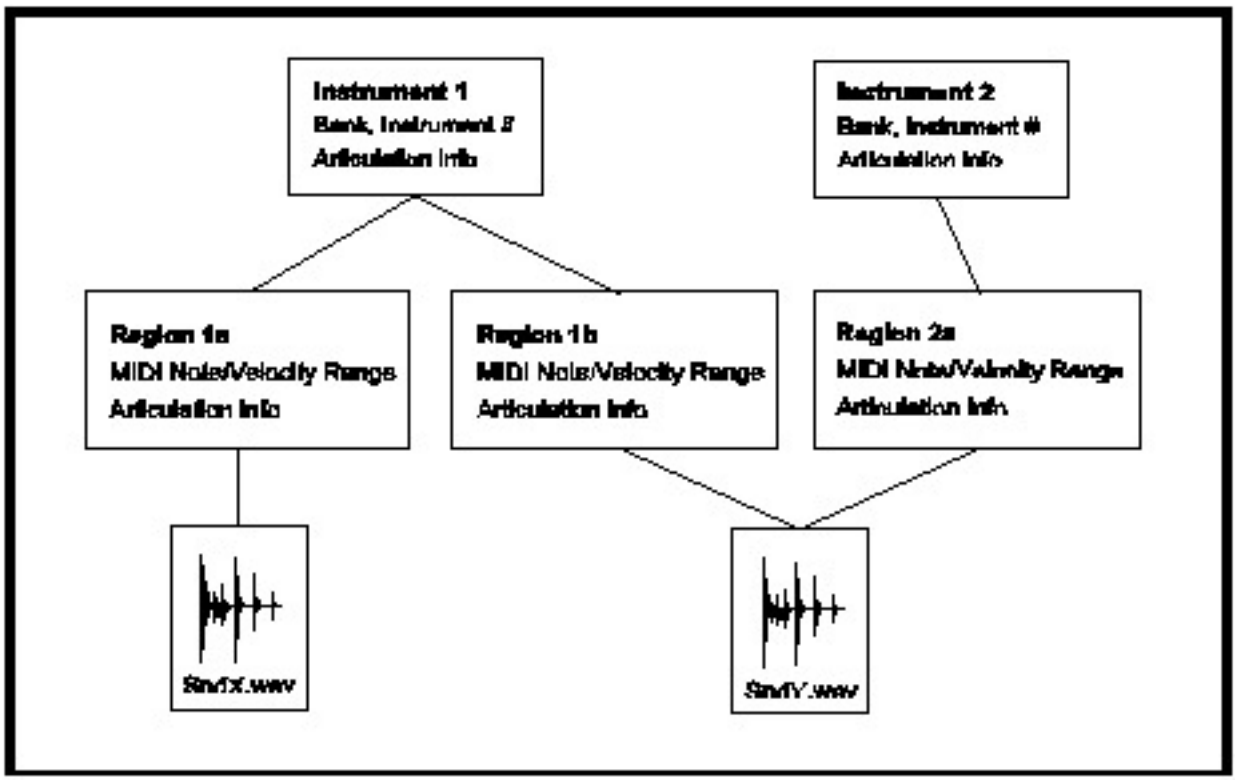


Abbildung 3.12: MIDI Downloadable Sample Instrument Organisation

3.2.8 Beispiel: Programm MIDI-File Player

Eine der Hauptprobleme bei MIDI-Fileplayer ist es, rechtzeitig die Daten abzuschicken. Dies heißt, dass mit einer geforderten Genauigkeit von bis zu 0.3 ms ein Befehl oder ein Byte geschickt werden muss. Dies ist oft ein Problem der Betriebssysteme, ob diese einerseits eine genügend genaue Zeitmessung haben und ob es den Ausgabe-Prozess eine Reaktionszeit kleiner dieser Grenze garantiert. Wird in einen Multitasking-Betriebssystem das Programm durch einen anderen Prozess blockiert, so kann es passieren, dass Bytes zu spät geschickt werden oder verloren gehen. Sogenannte Realtime Betriebssysteme verbessern dieses Verhalten (realtime kernel).

Im Anhang A.4 ist ein Sourcecode zum testen des Systems aufgelistet. Dieser Code sollte unter Windows 95,98,ME und NT als auch unter Unix(Linux,Irix) kompilierbar sein. Das Programm macht mehrere Zeitabfragen

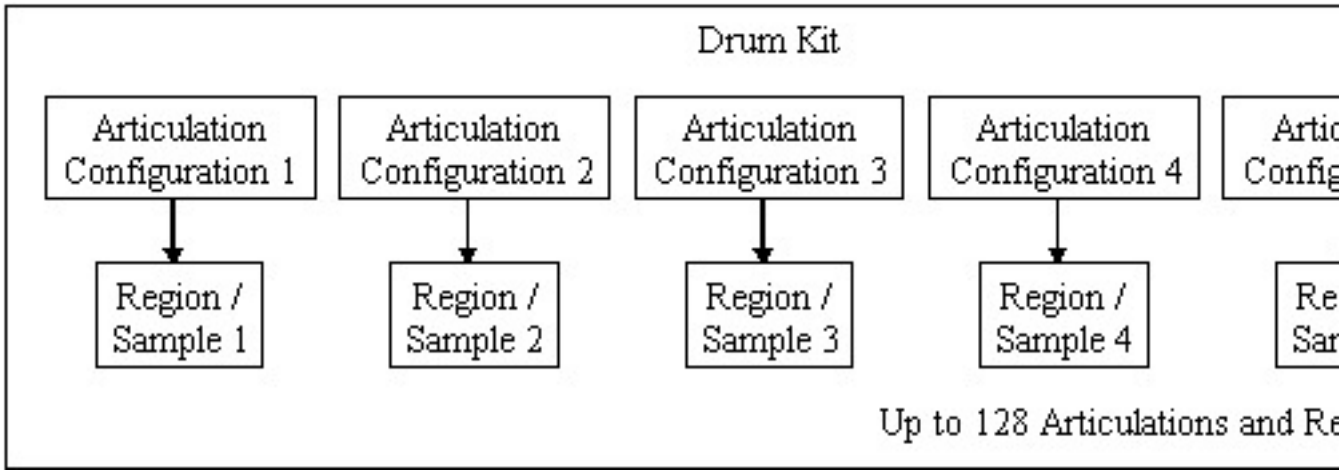


Abbildung 3.13: MIDI Downloadable Sample Instrument

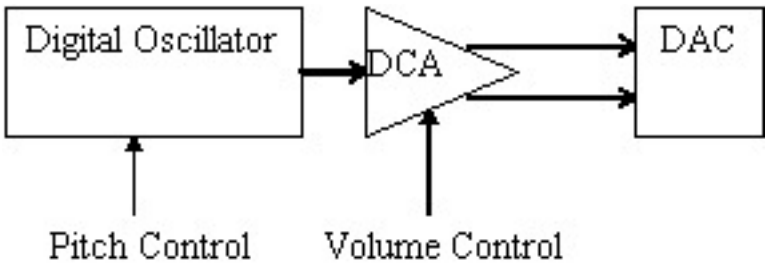


Abbildung 3.14: MIDI Downloadable Sample Oszillator

mittels Systemaufrufe in Folge und misst die vergangene zeit zwischen den Aufrufen und dann bei einer Million aufrufen, die maximale Zeit wie lange es blockiert wurde und wie oft es länger als 1 ms warten musste zu ermitteln.

Ein Beispiel ist in Tabelle 3.1 zu sehen. Hier wurde ein Linux-System mit Kernel 2.2.17 getestet . Dies zeigt, dass diese Rechner schnell genug sind, um sich mit 1000 "flops (Floating Point Operations)" nicht in messbare Zeit-Ungenauigkeiten einlässt. Andererseits ist die Zeitmessung im Mikrosekundenbereich ausreichend aufgelöst. Jedoch bleibt bei vielen Aufrufen die Reaktionszeit über unseren Grenzwert von 1 ms und erreicht sogar 19 ms Maximum. Dies zeigt deutlich dass bei Linux 2.2.17 auf einen Intel-Rechner der Scheduler mit 10 ms "schaltet", und damit bei zwei Zeitscheiben schon

```
------------------------------------------------------------
Linux seneca 2.2.17 #1 Thu Nov 16 14:25:27 CET 2000 i686 unknown
Tue Dec  5 13:33:25 CET 2000
processor : 0
vendor_id : GenuineIntel
model name : Pentium II (Klamath)
cpu MHz : 233.869
cache size : 512 KB
fpu : yes
fpu_exception : yes
bogomips : 466.94
------------------------------------------------------------
#./times
Testing timing in OS-Systems :
-> 1.) 0.000007 sec (difference: 0.00000 sec)(wait =     1 flops)
-> 2.) 0.000010 sec (difference: 0.00000 sec)(wait =     8 flops)
-> 3.) 0.000013 sec (difference: 0.00000 sec)(wait =    27 flops)
-> 4.) 0.000016 sec (difference: 0.00000 sec)(wait =    64 flops)
-> 5.) 0.000021 sec (difference: 0.00000 sec)(wait =   125 flops)
-> 6.) 0.000028 sec (difference: 0.00001 sec)(wait =   216 flops)
-> 7.) 0.000040 sec (difference: 0.00001 sec)(wait =   343 flops)
-> 8.) 0.000055 sec (difference: 0.00001 sec)(wait =   512 flops)
-> 9.) 0.000077 sec (difference: 0.00002 sec)(wait =   729 flops)
-> 10.) 0.000108 sec (difference: 0.00003 sec)(wait =  1000 flops)
Delta time tests between 1000000 time systemcalls:
-> max delta time= 0.019492 sec
-> min delta time= 0.000002 sec
-> 7 times from 1000000 more than 1 ms, that is 0.000700 %
#
```

Tabelle 3.1: Timing Test an einem Linux Computer

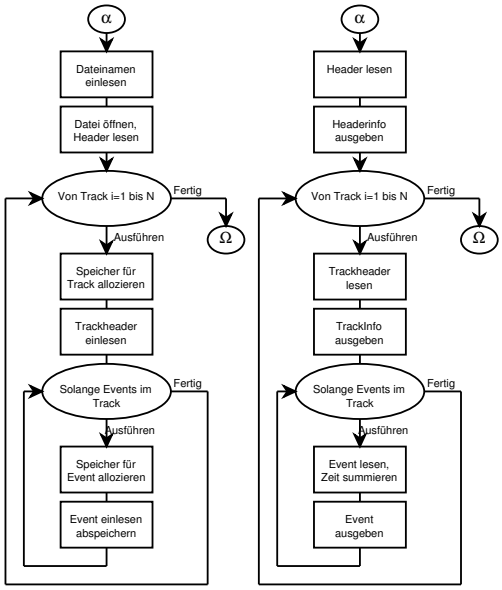


Abbildung 3.15: MIDI File Leser - Einlesen - Ausgabe

20 ms Blockade erreicht werden. Realtime-Kernel können dies verhindern.

Zum Schluss noch einige Flussdiagramme für MIDI-File leser und -spieler.

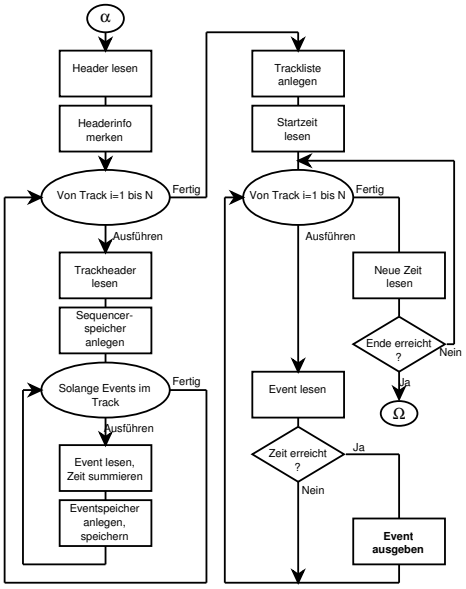


Abbildung 3.16: MIDI File Leser - Player

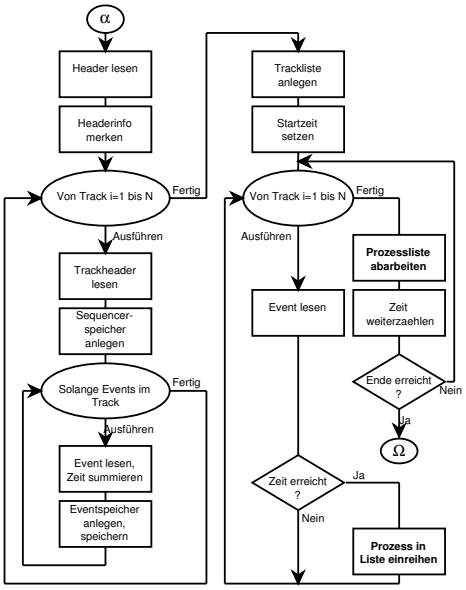


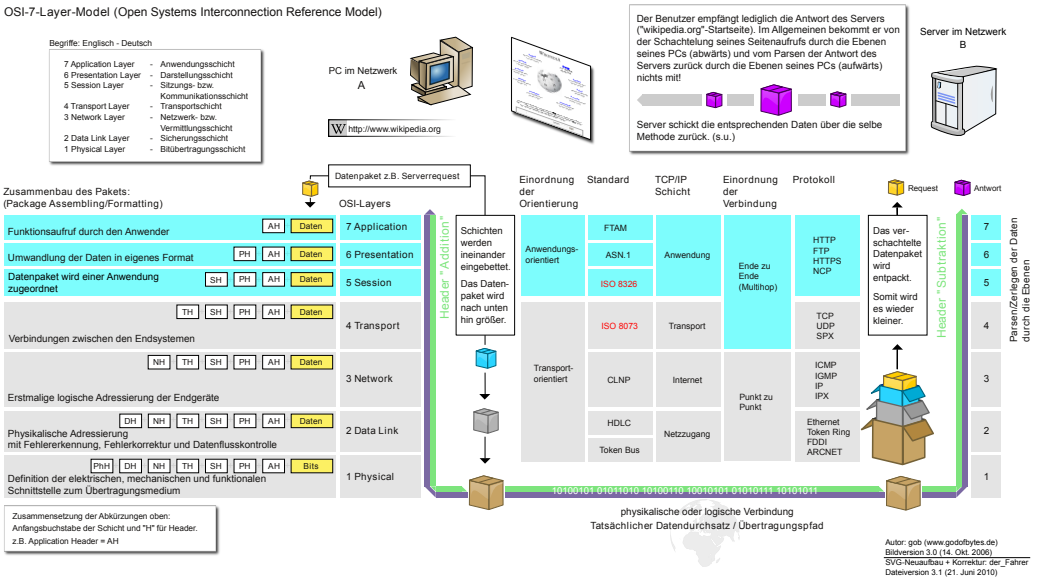
Abbildung 3.17: MIDI File Leser - Ausgabe

3.3 Datennetzwerke

3.3.1 Ethernet

Ethernet-Netzwerke sind bei Xerox PARC zwischen 1973 and 1975 zum Datentransport entwickelt worden, jedoch nicht von vornherein für Steueraufgaben in Echtzeit und vor allem nicht für den Audiobereich. Durch die hohe Daten-Durchsatz und Übertragungsgeschwindigkeiten, kann es jedoch für solche Zwecke verwendet werden.

Zur Datenübertragung auf Basis des Ethernet wird im groben in meheren Layers unterteilt, wobei das Ethernet Layer den untersten darstellt und verschiedene physikalische Layer unter sich hat, wie auch optische oder Coax oder ausgekreuzte Kupferleitungen.



Ethernetdaten werden in Packetform, sogenannten frames verschickt. Als Adressierung dient im wesenlichen die "Media Access Control address" (MAC Addresse).

Das meistverwendeten Protokolle am Ethernet für diese Zwecke sind aufgrund der freien Programmierbarkeit TCP/IP und UDP.

Garantiert bei TCP/IP eine verlustfreie Übertragung durch Handshake, so benötigt es auch eine immer eine Rückbestätigung für den Empfang, was zu wesentlich längeren Übertragungszeiten führt als bei UDP, wo Datenpackete zu einen Empfänger geschickt werden ohne Kontrolle auf Verlust.

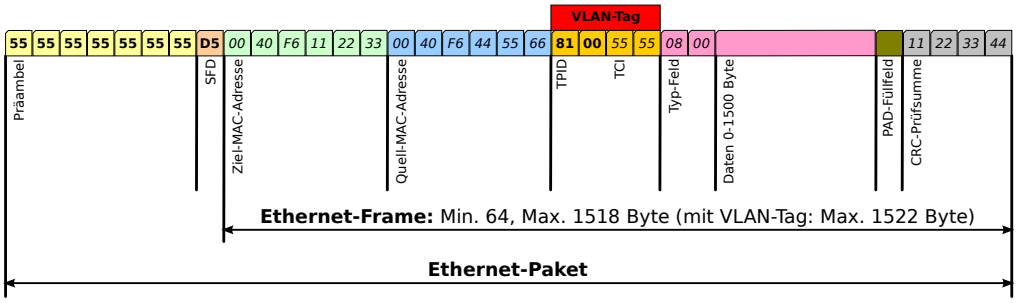


Abbildung 3.18: Ethernet Packet

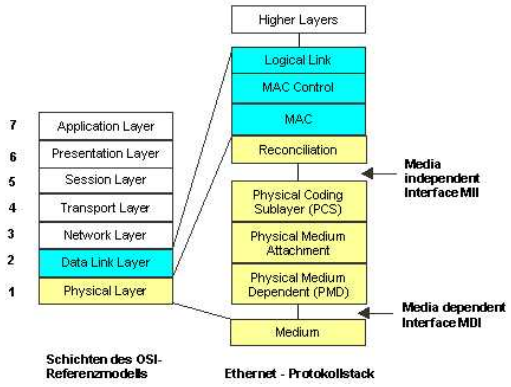


Abbildung 3.19: Protokollstapel des Ethernet

Daher wird häufig UDP für lokale Steuerung verwendet, wo eine Funktion der Vernetzung und garantierte Bandbreite sichergestellt werden kann, oder wo Daten nicht systemkritisch sind.

Der Aufbau des Ethernet Übertragungsprotokoll ist aus dem Protokollstapel ersichtlich.

Ethernet Hardware Layer

Einige frühe Varianten von Ethernet

- *Xerox Ethernet (Alto Aloha System)* - obsolet
- *10Broad36* (IEEE 802.3 Clause 11) – Obsolet.
- *1BASE5* (IEEE 802.3 Clause 12) - Fehlschlag.
- *StarLAN 1*

10-Mbit/s-Ethernet

- 10-Mbit/s-Ethernet mit Koaxialkabel *10BASE2*, IEEE 802.3 Clause 10 (früher IEEE 802.3a)
- Thick Ethernet Transceiver *10BASE5*, IEEE 802.3 Clause 8, ein 10 mm dickes Koaxialkabel (RG8)
- 10-Mbit/s-Ethernet mit Twisted-Pair-Kabel, RJ45 Stecker und Buchse,
- *StarLAN 10*
- *10BASE-T*, IEEE 802.3 Clause 14 (früher IEEE 802.3i) – läuft über vier Adern (zwei verdrillte Paare) eines CAT-3 oder CAT-5-Kabels
- 10-Mbit/s-Ethernet mit Glasfaser-Kabel
- *FOIRL* – Fiber-optic inter-repeater link. Der ursprüngliche Standard für Ethernet über Glasfaserkabel
- *10BASE-SX* – 10/100-Mbit/s-Ethernet über Glasfaser.

100-Mbit/s-Ethernet *Fast Ethernet*

Durchgesetzt hat sich einzig 100BASE-X (IEEE 802.3 Clause 24) für Twisted-Pair-Kabel und Glasfasern mit 125 MBaud

- 100BASE-TX, IEEE 802.3 Clause 25 (früher IEEE 802.3u) benutzt wie 10BASE-T je ein verdrilltes Adernpaar pro Richtung
- 100BASE-FX, IEEE 802.3 Clause 26, 100 Mbit/s Ethernet über Multimode-Glasfaser

Gigabit-Ethernet [**Bearbeiten**] kurz: GbE oder GigE) kommen

- *1000BASE-T*, IEEE 802.3 Clause 40 (früher IEEE 802.3ab) – 1 Gbit/s über Kupferkabel ab UTP-Kabel oder besser Cat-5e oder Cat-6. Die maximale Länge eines Segments beträgt wie bei 10BASE-T und 100BASE-TX 100 Meter

- *1000BASE-TX*, *1000BASE-T2/4* (nicht in IEEE 802.3 standardisiert) – Erfolglose Versuche verschiedener Interessensgruppen
- *1000BASE-SX*, *1000BASE-LX*, IEEE 802.3 Clause 38 (früher IEEE 802.3z) – 1 Gbit/s über Glasfaser.
- *1000BASE-LX/LH*, manchmal auch *1000BASE-LH* (LH steht für *Long Haul*) – Zum Einsatz kommen hierbei Singlemode-Glasfaserkabel mit einer maximalen Länge von 10 km.
- *1000BASE-ZX* – Zum Einsatz kommen Singlemode-Glasfaserkabel mit einer maximalen Länge von 70 km.
- *1000BASE-CX*, IEEE 802.3 Clause 39 – Als Übertragungsmedium werden zwei Adernpaare mit einer maximalen Kabellänge von 25 m verwendet

10-Gbit/s-Ethernet [Bearbeiten] Der 10-Gbit/s-Ethernet-Standard (kurz: 10GbE) bringt zehn unterschiedliche Übertragungstechniken, acht für Glasfaserkabel und zwei für Kupferkabel mit sich. 10-Gbit/s-Ethernet wird für LAN, MAN und WAN verwendet. Der Standard für die Glasfaserübertragung heißt

Power over Ethernet Ebenfalls zur Familie der Ethernet-Standards gehört IEEE 802.3af (IEEE 802.3 Clause 33). Das Verfahren beschreibt, wie sich Ethernet-fähige Geräte über das Twisted-Pair-Kabel mit Energie versorgen lassen. Dabei werden entweder die ungenutzten Adern der Leitung verwendet, oder es wird zusätzlich zum Datensignal ein Gleichstromanteil über die vier verwendeten Adern übertragen. Entsprechend ausgelegte Geräte werden mit -48 V und bis zu 15,4 Watt versorgt. Bis zu 30 W bei -54 V erreicht der Ende 2009 ratifizierte Standard PoE+. Eine Logik stellt sicher, dass nur PoE-fähige Geräte mit Energie versorgt werden. Einigen Firmen propagieren Hardware welche als PoE++ 92W liefern kann.

Audio over Ethernet

Bei *Audio over Ethernet* (machmal auch *AoE* genannt) wird ein Ethernet-basiertes Netzwerk zur Übetragung von Digital-Audio verwendet. Bestehende Protokolle wie *Voice over IP* (*VoIP*) oder Audiostreams über das Internet reichen für die Anwendungen meist nicht aus. Daher wurde spezielle zusätzliche Protokolle entwickelt, wobei sich allerdings noch kein Standard sich etabliert hat (Stand 2010).

Es gibt mehrere unterschiedliche und inkompatible Protokolle für Audio-over-Ethernet. Je nachdem auf welchen OSI-Layer das Protokoll aufsetzt, wird zwischen diesen und deren Möglichkeiten unterschieden.

Es gibt auch die Möglichkeit, wie bei "industrial ethernet", Echtzeit-Ethernet zu implementieren, Dabei werden einzelnen Geräte Zeitframes für ihre Datenübermittlung zugeteilt, wodurch ein Kollisionsschutz der Packete besteht und eine garantierte Übertragung von Daten erfolgen kann.

Industrial Ethernet ist der Oberbegriff für alle Bestrebungen, den Ethernet-Standard für die Vernetzung von Geräten, die in der industriellen Fertigung eingesetzt werden, nutzbar zu machen. In diesem Zusammenhang spricht man dann auch von einem Echtzeit-Ethernet und wird zunehmend bei der Automatisierung von Produktionen verwendet.

Layer 1 Protokolle

Verwenden nur Physikalischen Layer, nicht einmal den Ether-Frame Standard. Daher können diese nicht über Standard-Switches übertragen werden, und es muss bei aufwendigeren Netzen spezielle Verkabelungs-Infrastruktur (Hardware) verwendet werden.

- A-Net by Aviom
- AES50, SuperMAC, an implementation of AES50
- AudioRail
- M11 by AudioRail
- MaGIC by Gibson
- Rocknet by Media Numerics
- REAC by Roland

Diese Protokolle besitzen mittlerweile kaum mehr Bedeutung, da sie meist schon zu standard Netzwerkkarten inkompatibel sind.

Layer 2 Protokolle

Layer 2-Protokolle kapseln Audiodaten in Standard-Ethernet-Pakete. Die meisten von diesen können Standard Ethernet Hubs und Switches verwenden.

- AES51 , ATM services over Ethernet unter verwendung von AES3 audio , ähnlichen der Methode von AES47
- Audio Video Bridging (AVB) Audio Video Bridging (AVB)

  Konsortium um einen Standard auch im Consumer-Bereich zu etablieren, wird zur Zeit stark propagiert (Stand 2010)
- CobraNet: RAVE von QSC Audio

  Älteres Protokoll im Einsatz in einigen kozertsäälen usw.
- EtherSound by Digigram: zB.: NetCIRA by Fostex

  In einger Hardware implementiert, jedoch verwendet fostex neues protokoll
- Hydra by Calrec

Layer 3 Protokolle

Layer-3-Protokolle kapseln Audiodaten in Standard IP-Paketen (normalerweise UDP / IP oder RTP / UDP / IP). Die Nutzung des IP-Protokoll verbessert die Interoperabilität mit Standard-Computing-Plattformen und in einigen Fällen verbessert es die Skalierbarkeit der Audio-Verteilung. Die Schicht 3 Audio-over-Ethernet-Protokolle sind nicht für die Übertragung über lange Strecken im Internet gedacht, dafür gibt es bessere Streaming Protokolle.

- AoIP by Wheatstone Corp
- Livewire by Axia, a division of Telos Systems
- Dante by Audinate
- Q-LAN by QSC Audio Products
- RAVENNA by ALC NetworX

Ravenna schickt sich an AVB zu ersetzen und hat schon Industriepartner gefunden.

Ähnliche Konzepte ohne Ethernet

Die Audio Engineering Society 's MADI oder AES10 , obwohl eine ähnliche Funktion, verwendet es 75 - Ohm -Koaxialkabel mit BNC-Buchsen. Es ist sehr ähnlich im Design von AES3 , die (Kanäle können nur Stereo übertragen).

Die Audio Engineering Society 's AES47 , bietet Linear Audio Vernetzung indem AES3 Audio-Verkehr über ein ATM-Netzwerk mit strukturierter Netzwerkverkabelung (beide Kupfer und Glasfaser). Dies wird meist von Auftragnehmern wie BBC 's Echtzeit-Audio-Anschluss in ganz Großbritannien.

Streaming

3.3.2 Audiofiles und -streams als Datentransport

Zum Transport von Audiodaten können Audiodateien direkt geschickt werden oder in Form von Streams übertragen werden. Mit dem streaming im Internet gab und gibt es dabei fortwährende Weiterentwicklung, welche entweder technologisch oder strategische Gründe hat.

Gerade bei Videostreams in Kombination mit Audio spielen auch Patente eine wichtige Rolle, welche freie Entwicklungen zunehmends behindern. Trotzdem gibt es durch den Prozess der Standardisierung der IETF (Internet EngineeringTask Force) über RFCs (Request for comments) neue Protokollentwürfe, welche sich dafür eignen.

Streaming-Protokolle

Als Streaming-Protokolle bezeichnet man spezielle Protokolle für die Übertragung von Streaming Media-Daten über ein Netzwerk. Beim Live-Streaming werden bereits digitalisierte Audio- oder Videodaten mittels eines Encoders in ein Streaming-Format umgewandelt (kodiert, coded) um dann wieder dekodiert (decoded) zu werden, die kompatiblen Software Module dafür werden CODECs genannt.

Spezielle Protokolle sind nur für das Live-Streaming erforderlich; für das On-Demand-Streaming reichen die konventionellen Protokolle zum Dateitransfer wie FTP, http usw. meist aus. Aufgrund der besseren Performance nutzen Streaming-Protokolle meist das "User Datagram Protocol" (UDP) zum Transport; ggf. können aber auch andere routingfähige Protokolle wie das Transmission Control Protocol (TCP), CLNP oder IPX genutzt werden.

Hier eine Zusammenfassung der wichtigesten Protokolle, welche zum Streaming verwendet werden. (Stand 2003).

Anmerkung: On-demand-Streaming ist oft nur ein Pseudostreaming.

On-demand-Streaming:

- Hypertext Transfer Protocol (HTTP)

  Das Hypertext Transfer Protocol (HTTP) ist ein Protokoll zur Übertragung von Daten über ein Netzwerk. Es wird hauptsächlich eingesetzt, um Webseiten und andere Daten aus dem World Wide Web (WWW) in einen Webbrowser zu laden.

- File Transfer Protocol (FTP)

  Das File Transfer Protocol (engl. für ?Dateiübertragungsverfahren?, kurz FTP), ist ein im RFC 959

von 1985 spezifiziertes Netzwerkprotokoll zur Dateiübertragung über TCP/IP-Netzwerke.

Live-Streaming:

- Real Time Transport Protocol (RTP)

  Das Real-Time Transport Protocol (RTP) ist ein Protokoll zur kontinuierlichen Übertragung von audiovisuellen Daten (Streams) über IP-basierte Netzwerke.Es dient dazu, Multimedia-Datenströme (Audio, Video, Text, etc.) über Netzwerke zu transportieren, d.h. die Daten zu kodieren, zu paketieren und zu versenden. RTP ist ein Paket-basiertes Protokoll und wird normalerweise über UDP betrieben

- Real Time Control Protocol (RTCP)

  Das RealTime Control Protocol arbeitet mit RTP zusammen und dient der Aushandlung und Einhaltung von Quality of Service (QoS) Parametern.

- Real-Time Streaming Protocol (RTSP)

  Das RealTime Streaming Protocol (RTSP) ist ein Netzwerkprotokoll zur Steuerung der kontinuierlichen Übertragung von audiovisuellen Daten (Streams) oder Software über IP-basierte Netzwerke. Mit ihm wird die Session zwischen Empfänger und Server gesteuert. Es basiert auf HTTP. Das Protokoll wurde von der IETF MMUSIC Group entwickelt und 1998 im RFC 2326 standardisiert.

IP-Telefonie: Unter IP-Telefonie (Internet Protocol-Telefonie; auch Voice over IP, kurz: VoIP gesprochen wie ,,Weup") versteht man das Telefonieren über Computernetzwerke, die nach Internet-Standards aufgebaut sind. Dabei werden für Telefonie typische Informationen, d. h. Sprache und Steuerinformationen für z. B. den Verbindungsaufbau, über ein auch für Datenübertragung nutzbares Netz übertragen. Bei den Gesprächsteilnehmern können sowohl Computer, für IP-Telefonie spezialisierte Telefonendgeräte als auch über spezielle Adapter angeschlossene, klassische Telefone die Verbindung ins Telefonnetz herstellen.

- H.323, H.310, H.320, H.321, H.322, H.324

  H.323 ist eine übergeordnete Empfehlung der ITU-T, in der Protokolle definiert werden, die eine audio-visuelle Kommunikation auf

jedem Netzwerk, das Pakete überträgt, ermöglichen. Es ist zur Zeit in verschiedenen Anwendungen wie zum Beispiel NetMeeting und OpenH323 implementiert.Es ist ein Protokoll der H.32X-Serie, die auch die Kommunikation über öffentliche Telefonnetze und ISDN enthält.

- G.711, G.722, G.728, G.729, G.723

  G.711 ist ein Standard der ITU-T zur Digitalisierung von analogen Audiosignalen. Er wird in der klassischen Festnetz-Telefonie verwendet (PCM-Technik). Auch bei VoIP ist G.711 eine mögliche Sprachcodierung.

- T.120

  Der ITU-T-Standard T.120 besteht aus einer Reihe von Kommunikations- und Anwendungsprotokollen, die der Ermöglichung von Echtzeit-Datenverbindungen für Multimedia-Konferenzen dienen. Mit Hilfe von T.120-basierter Software können mehrere Benutzer an Konferenzschaltungen teilnehmen, auch wenn sie verschiedene Arten von Netzwerken verwenden.

  Verwendet für ,,Program Sharing" (gemeinsames Verwenden eines Programms) wie zum Beispiel Whiteboard-Anwendungen.

Multicasting: Multicast oder Gruppenruf bezeichnet in der Telekommunikation eine Nachrichtenübertragung von einem Punkt zu einer Gruppe (auch Mehrpunktverbindung genannt). Der Vorteil von Multicast besteht darin, dass gleichzeitig Nachrichten an mehrere Teilnehmer oder an eine geschlossene Teilnehmergruppe übertragen werden können, ohne dass sich beim Sender die Bandbreite mit der Zahl der Empfänger multipliziert. Der Sender braucht beim Multicasting nur die gleiche Bandbreite wie ein einzelner Empfänger. Handelt es sich um paketorientierte Datenübertragung, findet die Vervielfältigung der Pakete an jedem Verteiler (Switch, Router) auf der Route statt.

Diese Protokolle sind für allem für Router wichtig welche dies implementiert haben müssen. Dafür muss meistens ein TTL (Time To Live) - Wert in HOPs (Anzahl der Netzwerke zu überbrücken) definiert werden:

- Scalable Reliable File Distribution Protocol (SRFDP)

- Scalable Reliable Real-time Transport Protocol (SRRTP)
- Border Gateway Multicast Protocol (BGMP)

Session Discovery: Wird zur Initierung von Streams, vor allem automatischen benötigt.

- Zeroconf bzw. Rendezvous

  eigentlich zum Streamen aber automatischen Aufbau von Netzen, IP-Address zusweisung.
- Session Announcement Protocol (SAP)

  Das Session Announcement Protocol (SAP) ist ein Netzwerkprotokoll aus dem Bereich der IP-Telefonie, welches Multicast-Multimediasitzungen und andere Multicast-Sitzungen unterstützt und die entsprechende Sitzungskonfiguration an in Frage kommende Teilnehmer weiterleitet.
- SLP

  Das Service Location Protocol, oder kurz SLP, ist ein Protokoll zum Auffinden von Netzdiensten in einem TCP/IP-basierten Netz.

Session Description:
- Digital Media Access Protocol (DMAP) Digital Audio Access Protocol (DAAP) nicht offen modifiziertes HTML f. iTunes, Digital Photo Access Protocol (DPAP) nicht offen, modifiziertesHTML f. iPhoto.
- Session Description Protocol (SDP)

  Das Session Description Protocol (SDP, RFC 2327) dient dazu Kommunikationssitzungen zu verwalten und wird beispielsweise zusammen mit SIP und H.323 bei der IP-Telefonie eingesetzt.

  SDP dient hier dazu, die zwischen den Endpunkten zu verwendenden Codecs, Transportprotokolle, usw. auszuhandeln.

Sonstige beim Streaming genutzte Protokolle:
- Internet Protocol (IP)
- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)
- Internet Group Management Protocol (IGMP)

- Microsoft Media Server Protocol (MMS)
- Remote Desktop Protocol (RDP)

Weiters siehe Streaming: Video und Audio im Internet

3.4 OSC - OpenSound Control

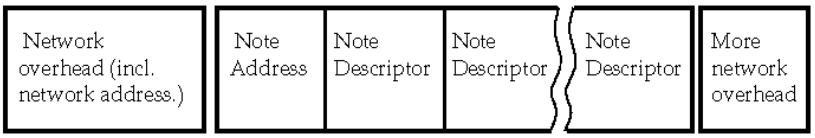


Abbildung 3.20: MPDL Packet Definition

OpenSound Control („OSC") ist ein Protokoll zur Kommuniktion zwischen Computer, Synthesizer und anderen Multimediageräten. Es ist für neue Netzwerktechnologien optimiert und ist unabhängig von der Art der Vernetzung. Es gibt viele Implementationen auf verschiedenen Verbindungstechniken.

Ein Vorläufer von OSC war ZIPI, welches auch in CNMAT an der Universität von Berkley entwicklet wurde. Dieses setzte direkt auf den OSI-Layer (sie Internetprotokoll) auf und hat zu allerlei kontroversieller Diskussion über Vernetzung geführt, jedoch zu keinen Implementationen. Die Entwickler von ZIPI halfen dann auch mit OSC zu entwerfen, welches freier definiert war. Eine Schlüsseldefinition war die "Music Parameter Description Language" von dem das hierachische Addressschema übernommen wurde. (siehe ZIPI Homepage).

Die Haupteigenschaften von OSC sind:

- Offener, dynamischer, URL-basierter symbolischer Namesraum
- Numerische und symbolische Argumente
- Mehrere Ziele einer Nachricht möglich
- sehr genaue Zeitangaben für jede Nachricht
- gebündelte Nachrichten für synchrone Ereignisse
- dynamische Parameterermittlung am Server

Die Definitionen und Unterlagen sind online dokumentiert unter: OpenSoundcontrol at Berkley und Implementationsbeispiele gibt es unter

OpenSoundcontrol Kit at Berkley .

3.4.1 Spezifikationen

Die Spezifikationen sind unter OSC 1.0 veröffentlicht und beschreiben ein Minimum an Definitionen zur Verwendung von OSC.

OSC ist unabhängig vom Transportlayer (im OSI-Layer) definiert, das betdeutet dass es in den Implementationen zusätzliche Vereinbarungen geben muss um eine Kommunikation zu ermöglichen.

Die Spezifikationen im Anhang **??** aufgeführt.

Die Dateneinheit wird als OSC-Packet definiert welches von einem Client zu Severn geschickt.

3.4.2 Anwendung

Eine der Haupteigenschaften von OSC ist der offene Standard, sodass es in einer Vielzahl von Anwendungsgebieten verwendet werden kann und speziell sich eignet verschiedene Bereiche zu vernetzen. So beschränkt sich dieses Protokoll nicht nur auf Steuerung von Musik und Klang sondern auch für visualisierungen, Sensoren und andere Bereiche.

Beispiel ist

Kapitel 4

Musikcomputer

4.1 Hardwarelösungen für Audio u. MIDI-Interfaces

4.1.1 Master/Slave-Buse

4.1.2 DSP-Hardware

4.2 Betriebsystemeinbindung

4.2.1 Generelle Überblick über OS

4.2.2 Echtzeitbetriebsysteme

4.2.3 Kerneldriver und Reaktionszeiten

4.2.4 Übung anhand eines Programierbeispiels für OSC Ein-/ausgabe

Kapitel 5

Aufbau von CM-Software

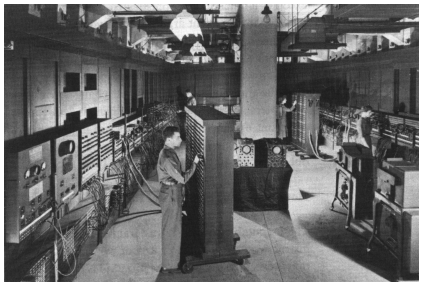


Abbildung 5.1: Programmieren des Eniac Computer 1945 an Schalter-Matrixen

Die Programmierung con Computer hat sich entscheidet verändert, sodaß sich auch die Software selbst anders präsentiert als früher. Wenn jemand in ein Programm wie CSound Daten eingibt, so wird nicht von "notieren" sondern von "programmieren" gesprochen. Noch schwieriger ist diese Entscheidung bei grafisch orientierten Programmen wie "MAX" oder "Pd".

5.1 Stapelverarbeitung von Programmen

5.1.1 Sammlung von Programmen als Funktionen

5.1.2 Abarbeitung mittels Stapeldateien

Idee: "Make a strong bar of sound then a glissando down and a glissando up from a recorded material mand Mixing these down."

Realisation: Mittels CDP.

5.2 Computermusiksprache CSound

Die Realisation von Musik mittels digitalem Computer erfordert das Synthetisieren von digitalen Audiosignalen durch direkte Synthese mittels eines Programms und die Verwendung von Compiler oder Interpreter wobei ein Processing in mehreren Schritten erfolgen kann.

CSound ist ein Interpreter. Der Kompositionsvorgang bedarf jedoch mindestens zweier Quellen und zwar nach dem traditionellen Musikdenken:

1. Definition von Instrumenten in einer Orchesterdatei
2. Definition von Ereignissen in einer Partiturdatei (entspricht einer Notation von Ereignissen)

Ein Orchester ist dabei ein Programm, welches den Algorithmus zur Generation des Audiosignal beinhaltet. Eine Partitur ist ein Liste von Ereignissen, auf das die Instrumente des Orchesterprogramms reagieren. Diese Listen sind nicht ausschließlich Dateien, sondern können auch in Echtzeit als Steuerinformationen in Form von MIDI oder ähnlichen an CSound weitergereicht werden.

Zur Geschichte von CSound ein Exzerpt aus dem Vorwort des Manuals.

> *[. . .]*
>
> *The programs making up the* CSound *system have a long history of development, beginning with the Music 4 program written at Bell Telephone Laboratories in the early 1960's by Max Mathews. That initiated the stored table concept and much of the terminology that has since enabled computer music researchers to communicate. Valuable additions were made at Princeton by the late Godfrey Winham in Music 4B; my own Music 360 (1968) was very indebted to his work. With Music 11 (1973) I took a different tack: the two distinct networks of control and audio signal processing stemmed from my intensive involvement in the preceding years in hardware synthesizer concepts and design. This division has been retained in* CSound*.*
>
> *Because it is written entirely in C,* CSound *is easily installed on any machine running Unix or C. At MIT it runs on VAX/DECstations under Ultrix 4.2, on SUNs under OS 4.1, SGI's under 4.1, and on the Macintosh under ThinkC 4.0. With this single language for audio signal processing, users move easily from machine to machine.*

The 1991 version included many new features. I am indebted to others for the contribution of the phase vouched and FOF synthesis modules. That release also charted a new direction with the addition of a spectral data type, holding much promise for future development. The 1992 release is even more significant for its addition of MIDI converter and control units, enabling CSound *to be run from MIDI score-files and from external MIDI keyboards. Since the newest RISC processors bring to computer music an order of magnitude more speed than did those on which it was born, researchers and composers now have access to workstations on which realtime software synthesis with sensing and control is now a reality. This is perhaps the single most important development for people working in the field. This new* CSound *is designed to take maximum advantage of realtime audio processing, and to encourage interactive experiments in this exciting new domain.*

B.V.

- *v* - Barry Vercoe, Media Lab M.I.T., CSound Manual Einleitung, [?] *(2)*

5.3 Partitur

Die Partitur wird in einen scorefile definiert.

Zu jeder Orchesterdatei, in welcher die Instrumente definiert sind, werden eine oder mehrere Partituren als Scorefiles geschrieben. Als Beispiel eine Partitur zum Ringmodulatorinstrument von Risset.

```
f 2 0 512 7 0 43 1 171 1 84 -1 171 -1 43 0
f 3 0 512 9 1 1 0
  i1 0.000 0.150 424 20000 1000 0.010
  i1 0.150 0.300 727 20000 1000 0.010
  i1 0.450 0.300 1524 20000 2000 0.010
  i1 0.750 0.600 1136 20000 2000 0.010
  i1 1.350 0.600 1342 20000 2000 0.010
  i1 1.950 3.600 424 20000 1000 2.300
  i1 5.550 0.150 727 20000 1000 0.010
  i1 5.700 0.300 1524 20000 2000 0.010
```

```
  i1 6.000 0.300 1136 20000 2000 0.010
  i1 6.300 0.600 1342 20000 2000 0.010
  i1 6.900 0.600 424 20000 1000 0.010
  i1 7.500 3.600 727 20000 1000 2.300
  i1 11.100 0.150 1524 20000 2000 0.010
  i1 11.250 0.300 1136 20000 2000 0.010
  i1 11.550 0.300 1342 20000 2000 0.010
  i1 11.850 0.600 424 20000 1000 0.010
  i1 12.450 0.600 727 20000 1000 0.010
  i1 13.050 3.600 1524 20000 2000 2.300
  i1 16.650 0.150 1136 20000 2000 0.010
  i1 16.800 0.300 1342 20000 2000 0.010
e
```

Hier werden Ereignisse für bestimmte Zeitpunkte definiert. Im wesentlich gibt es:

- i-Anweisungen zum Spielen der Instrumente
- f-Anweisungen um Datenfelder (zb: Wellenformtabellen) anzulegen, auf welche in den Instrumenten referenziert wird.
- s für Abschnitte in der Partitur
- e für das Ende.

Beim i Befehl werden alle Parameter in einer Zeile hintereinander geschrieben, welche dann als $p1, p2, p3, ...$ im Instrument referenziert werden. Die ersten drei Parameter sind dabei mit Instrumentennummer, Startzeit, und Dauer fix definiert, alle weiteren können frei verwendet werden.

Beim f Anweisung ist der erste Parameter der Index des Datenfeldes, der zweite der Zeitpunkt des Erzeugens der Tabelle, der Dritte die Grösse des Feldes und der vierte der verwendete Algorithmus oder auch GEN-Routine genannt. Je nach GEN-Routine unterscheiden sich dann die folgenden Parameter

5.4 Orchester

Das Orchesterfile repräsentiert die Instrumentendefinitionen und bestimmt damit die Signalberechnung.

Als Beispiel ein Ringmodulatorinstrument von Risset:

```
sr          =           44100
kr          =           4410
ksmps       =           10
nchnls      =           1

instr       1

isustain    =    p3-p7
a1          expseg      .001, p7, 1,isustain, .001
a1          oscili      a1, p4, 3

a2          oscili      p5, p6, 2
            out         a1*a2
endin
```

Die ersten Zeilen definieren die Signalproduktionsparameter Samplerate, Controlrate und die Kanalanzahl. Das Verhältnis zwischen Sample- und Controlrate muss ganzzahlig sein und deshalb muss es getrennt angegeben werden, ist aber eigentlich redundant.

Die Syntax für folgenden Zeilen dazwischen lautet:

```
[variable]      <opcode>     [parameter1],[parameter1],[parameter1],....
```

Dabei sind die Zuweisungsvariable und Parameter optional. Ein Instrument wird mit *instr* < *nummer* > eingeleitet und mit *endin* beendet.

Dazwischen befinden sich CSound-befehle mit Zuweisungsvariable links, den Befehlsnamen und Eingabeparameter. Die Funktion *out* erzeugt den Output.

Es gibt mehrere Arten von Variablen, wobei die wichtigsten sind a-Variablen, k-variablen und i-Variablen, welche Audiosignale, Kontrollsignale und Initialisierungsvariablen darstellen. Die Variablen $p1, p2, p3, ...$ sind die Parameter welche von der Partitur als Steuervariable übergeben wird.

Jedes Instrument wird mittels eines Notenbefehls, i-Befehls in der Partitur initialisiert und nach der Notendauer wieder freigegeben, somit ist die Lebensdauer der Variablen gleich mit der Notenlänge bis auf die globalen g-Parameter.

5.5 Interaktive Programme

Abgesehen von der MIDI-Steuerung ergeben sich bei interaktiven Programmen die Forderung einer hörbaren Einflussnahme auf Grund von Benutzervorgängen während der Signalverarbeitung.

5.6 Grafische Oberflächen

Beispiel Cecilia

5.6.1 Echtzeiteinsatz von CSound

5.6.2 Spezielle Hardware fuer CSound

5.7 Echtzeit-System

Echtzeit bedeutet dabei die Zeit, die Abläufe in der ,,realen Welt" verbrauchen. Modellzeit hingegen bedeutet die von einer Software selbstverwaltete Laufzeit. Ist nun diese Modellzeit synchron zu Echtzeit, spricht man davon, dass das System echtzeitfähig ist.

Von Echtzeit-Systemen (englisch real-time system) spricht man, wenn ein System ein Ergebnis innerhalb eines vorher fest definierten Zeitintervalles garantiert berechnet, also bevor eine bestimmte Zeitschranke erreicht ist. Die Größe des Zeitintervalles spielt dabei keine Rolle: Während bei einigen Aufgaben (Motorsteuerung) eine Sekunde bereits zu lang sein kann, reichen für andere Probleme Stunden oder sogar Tage. Ein Echtzeit-System (englisch: real-time system) muss also nicht nur ein Berechnungsergebnis mit dem richtigen Wert, sondern dasselbe auch noch rechtzeitig liefern. Andernfalls hat das System versagt. Umgangssprachlich spricht man auch von in Echtzeit, wenn Programme ohne spürbare Verzögerung arbeiten Diese Definition ist jedoch sehr unsauber. (wikipedia 25.11.2005)

Prinzipiell wird zwischen weicher und harter Echtzeit unterschieden. Hierfür gelten jeweils unterschiedliche Echtzeitanforderungen (aus der Literatur):

- weiche Echtzeitanforderungen: Eingehende Aufgaben haben eine vorgegebene Reaktionszeit (Deadline), deren Verletzung jedoch noch nicht sofort katastrophale Auswirkungen hat und das Ergebnis dennoch von gewissem Wert für den Nutzer ist.

- harte Echtzeitanforderungen: Eine Überschreitung der vorgegebenen Reaktionszeit kann sofort zum maximalen Schaden führen und ein zu spät bereit gestelltes Ergebnis ist für den Nutzer wertlos. Die Einhaltung der Zeitvorgaben ist strikt.

Für Computermusiksysteme hat dies eigentlich nur Bedeutung, wenn der Anwendungszweck bekannt ist. Deswegen wird vor allem bei Fixinstallationen mit grossen Lautsprechersystemen eine harte Echtzeit angestrebt, wobei hier eher der Einsatz des Systems entscheidend ist. Können für Experimente Fehler zwar ärgerlich sein, jedoch unkritisch, so kann bei grossen Beschallungsanlagen gerade digitales Knacksen und Rauschen zu Systemausfällen und physischer Verletzung (der Ohren) führen.

Entscheidend sind für Audioanwendungen eher folgende Kriterien:

ereignisgesteuert und/oder zeitgesteuert

Bei der Ereignissteuerung wird auf ein von aussen kommendes Ereignis sofort reagiert, d.h. eine Verarbeitung gestartet. Gegebenenfalls wird eine gerade laufende Verarbeitung dabei unterbrochen. Die Ereignissteuerung hat den Vorteil, dass sie mit lediglich geringem Zeitverlust auf das Ereignis reagiert. Der Hauptnachteil ist, dass es nicht verhinderbar ist, dass mehrere Ereignisse innerhalb kurzer Zeit auftreten können und es damit zu einer Überlastung des Echtzeit-Systems (mit Verlust von Ereignissen und/oder Überschreitung von Zeitlimits) kommen kann.

Bei der Zeitsteuerung werden Verarbeitungen auf Grund eines vorher festgelegten Zeitplans gestartet. Der Rechenaufwand ist besser planbar. Das Verhalten der Anwendung ist für alle Zeit vorhersagbar, was die Zeitsteuerung für sicherheitskritische Anwendungen eignet.

Synchronizität von Ereignissen und Signalen

Wenn das System sowohl Signale als auch Ereignisse verarbeitet, sollte eine Synchronizität bei der Verarbeitung gewährleistet sein. Das bedeutet, dass die Durchlaufzeiten für Signale und Ergeinisse gleich sein sollten. Dies ist vor allem bei Steuerung mittels MIDI-Daten und gemischten Interfaces wichtig. Weiters sollten diese Daten auch synchron zu anderen Geräten in der Umgebung sein.

Latenz

Latenz (v. lat.: latens = verborgen) bedeutet etwas Verborgenes, unter der Oberfläche, noch nicht in Erscheinung Tretendes. In verschiedenen Zusammenhängen spricht man auch von der Latenzzeit als Zeitraum zwischen einer Aktion und dem Eintreten einer Reaktion. Im Speziellen ist bei Audiosystemen die Verzögerungszeit einer Audiocomputer, um ein Audiosignal vom Eingang zum Ausgang der Karte durchzureichen (analoger Impuls in digitales Signal und zurück). (siehe 5.7.1)

5.7.1 Latenz Audiocomputer

Bei Computersystemen für Echtzeitsoundverarbeitung ergibt sich die Latenz aus der Verbindung zwischen den Latenzen in der Hardware und der in der Software. Die Hardware ist vor allem durch die Soundkarten bestimmt und deren Anbindung an den Computer. Die Software durch Latenzen im Betriebssystem und der Anwendungsoftware.

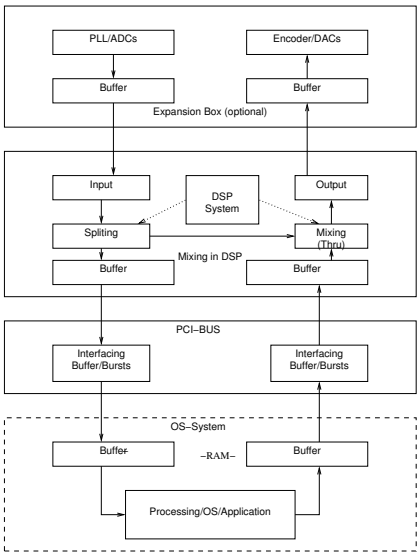


Abbildung 5.2: Audiosignal Computer mit Soundkarte und Expansion Board

Hardware Latenz

In Abbildung 5.2 sind die Komponenten, die ein Audisignal durchlaufen muss, dargestellt. Beim optionalen Expansion board wird oft ein eigenes Bussystem eingeführt oder auch auf im Computer vorhandene Schnittstellen wie Firewire oder USB zurückgegriffen. Bei Standardschnittstellen, welche sich mehrere Geräte of teilen kann es zu beträchtlichen Latenzzeiten kommen. Ansonsten sollten in dieser Kette Verzögerungen von mehreren Samples ein Maximum darstellen, da es dabei meist kein Multitasking gibt und damit konstante Datenweitergaben garantiert werden.

Ein AD- oder DA-Wandler besitzt meistens 1 Sample Zwischenspeicher, jedoch ein Systembus im Computer grössere Blöcke. Zum Beispiel werden beim PCI-Bus die Daten in Bulks von mehreren kByte übertragen, sodass hier entsprechende Verzögerungen einkalkuliert werden müssen.

Software Latenz

Die Softwarelatenz setzt sich aus den Buffer im Betriebsystem und der Anwendung zusammen. Effizient geschriebene Software kann diese Buffer vereinen und sogar mit den Buffern der Audiohardware arbeiten, sodass diese je nach Anwendung verschieden wirken. Als Beispiel wird hier das

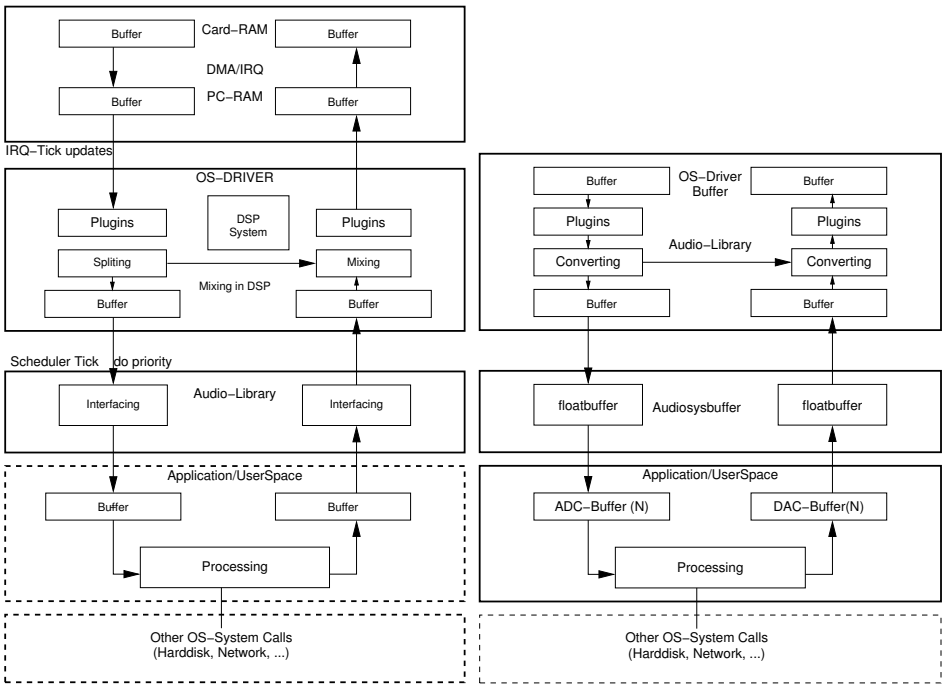


Abbildung 5.3: Signalkette durch das Linux Betriebsystems und der Pure Data Anwendung

Linux-betriebsystem und die Pure Data Anwendung beschrieben, wobei als Übergabe noch eine zusätzlicher Soundsever geschaltet sein kann. Auch unter dem Betriebsystem Windows[1] ist diese Aufteilung ähnlich.

Die Applikation bestimmt meist die Art, wie die Soundkarte angesprochen wird. Dabei kann der Betriebsystemtreiber noch zusätzliche Konvertierungen (der Samplerate, des Formats) vornehmen, was einen zusätzlichen Buffer bedingt. Eine zeitkritische Anwendung sollte hier möglichst des nativen Formats der Soundkarte bedienen, welche jedoch unterschiedlich sein kann, oder eben das Konvertieren, falls es die Daten in einen speziellen Format benötigt, dem Betriebsystem überlassen, welches dafür optimierte Routinen besitzen sollte.

Applikationen bedienen sich meist mehrere Buffersegmente eines Ringbuffers um die Entkopplung zur Hardware zu ermöglichen (siehe Abbildung 5.9)

Soundkarten besitzen oft die Möglichkeit, Signale direkt durchzuschleifen, dabei wird von "Zero Latency Monitioring" (ZLM) gesprochen. Dabei ist es

[1] NT

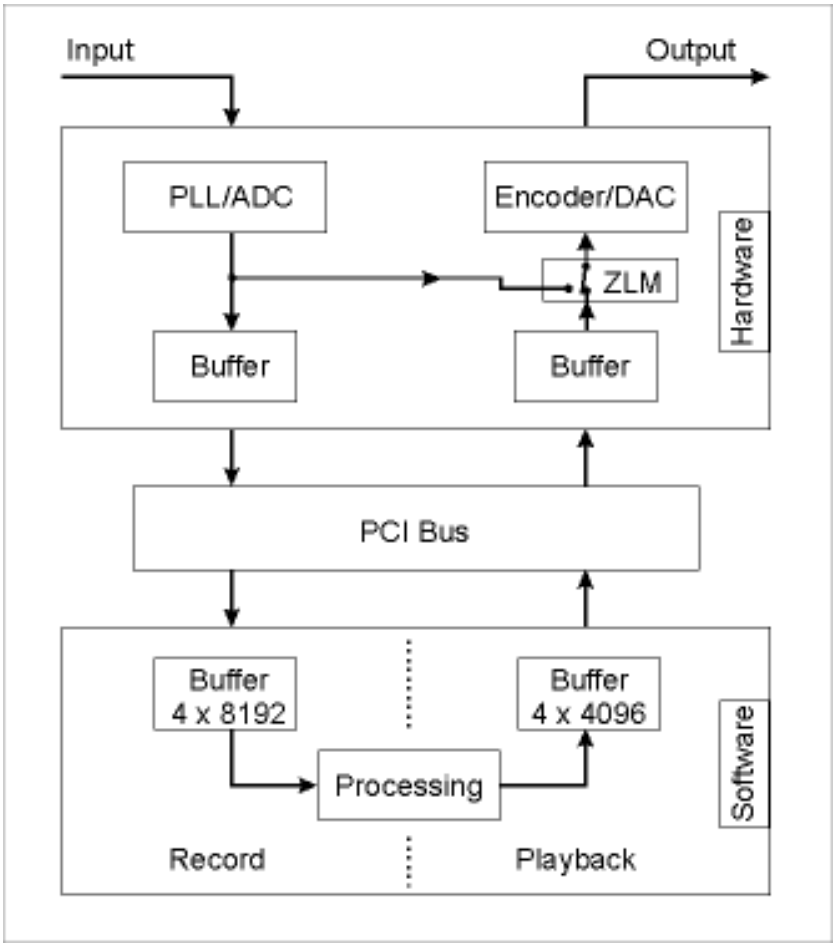


Abbildung 5.4: Zero Latency Monitoring

oft möglich, die Durchschleifungen noch lautstärkemäßig zu gewichten.

5.8 Signalverarbeitende Echtzeitsysteme

Die Entwicklung signalverarbeitender Echtzeitsysteme in der Musik geht eigentlich aus der Entwicklung von Echtzeitereignissystemen hervor, welche mittels Messages und MIDI Steuerungen vorgenommen haben. Die Verwendung von Signalen war eigentlich nur eine Erweiterung, wie man an der Entwicklung vom "MIDI-Patcher" zu "Max/fts" oder an der MSP-Library für Opcode-MAX sehen kann. Jedoch ist diese Entwicklung nicht unabhängig von einen neuen Denken in der Musik oder allgemeiner Medienkunst zu sehen[2].

Dieser weitere Schritte der Abstrahierung von Musik war die Sicht der "Musik als Prozess". Dies bedeutet, dass Musik als Definition von Prozessanweisungen geschrieben wird und damit algorithmische Musik quasi in Reinform entwickelt wurde. Dabei ist es möglich, eine Notation in grafischer objektorientierter Form zu verwenden und eine Komposition in der Form eines Datenflussdiagramms darzustellen. Der Einsatz ergab sich vor allem bei automatisierten Klang-Installationen und "unendlich" langen Kompositionen[3].

Die Sicht der Maschine als Modellierungsprinzip bedingt den Einsatz von Datenflussmodellen, anstatt von Programmflussmodellen wie sie in anderen Programmiersprachen verwendet werden. Datenflussmodelle haben die Graphentheorie als Grundlage[4].

In der Graphentheorie versteht man unter einem Graph, eine Menge von Punkten, zwischen denen Linien verlaufen. Die Punkte nennt man Knoten oder Ecken, die Linien nennt man meist Kanten. Beim gerichteten Graphen werden diese mit Pfeilen gezeichnet. Die Anfänge der Graphentheorie gehen bis in das Jahr 1736 auf Leonhard Euler zurück (Königsberger Brückenproblem)[5].

---

[2]Christa Lichtenstern, Vom Mythos zum Prozessdenken. Ovid-Rezeption - Surrealistische Ästhetik - Verwandlungsthematik der Nachkriegskunst, Weinheim (Acta humaniora) 1992

[3]Als ein frühes Beispiel des reinen Prozessdenkens als Komposition könnten die Werke von Alvin Lucier angesehen werden. Speziell das Stück "I am sitting in a room", wobei mittels eines Raummikrofons und wiederholtes Aufnehmen der Wiedergabe den Raum mit einen gesprochenen Satz als Ausgangsmaterial zum Schwingen bringt.

[4]Eine Grundlage dazu schufen die Informatiker, siehe "Software Synthesis from Dataflow Graphs", Shuvra S. Bhattacharyya, Praveen K. Murthy, Edward A. Lee, Kluwer Academic Publishers, Norwell Massachusetts, 1996

[5]Am konkreten Beispiel bezieht es sich auf die Stadt Königsberg und die Frage, ob es einen Rundweg gibt, bei dem man alle sieben Brücken der Stadt über den Pregel genau einmal überquert und wieder zum Ausgangspunkt gelangt. Euler bewies, dass es keinen solchen Rundweg geben kann.

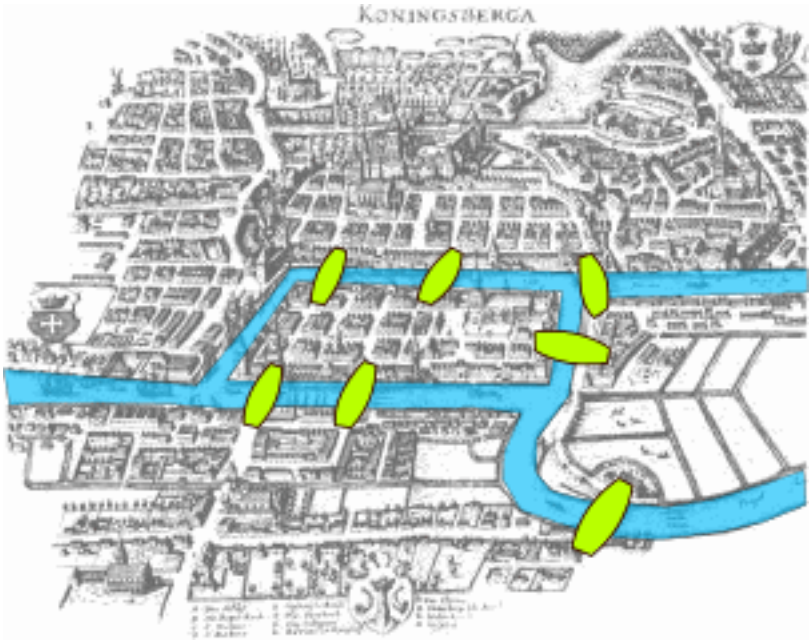


Abbildung 5.5: Das Königsberger Brückenproblem, von Euler gelöst

Beim Signaldatenfluss ist ein gerichteter Graph eine Ansammlung von Operationen, welche visuell durch Boxen oder andere grafische Elemente dargestellt sind, verbunden mit Linien (die eigentlich Pfeile sein sollten). Es gibt Operationen ohne Eingänge, welche als Quellen von Daten dienen und welche ohne Ausgang, die als Senken oder Schnittstellen zu anderen Systemen betrachtet werden können. Alleinstehende Operationen dienen meist zur Definition. In diesem Graph werden jedoch zwei unabhängige Datenflussebenen repräsentiert, eine Signalebene (Signal Dataflow) und eine Ereignisebene (Message System).

Der Scheduler - Dispatcher Mechanismus ist für die Verteilung der Daten und damit den Datenfluss zuständig. In welcher Reihenfolge Daten verteilt werden geht nicht immer direkt aus der grafischen Repräsentation hervor. Daher ordnet der Scheduler die Reihenfolge der Operationen und der Dispatcher arbeitet sie dann ab. Bei Pd wird ein spezieller Scheduler verwendet der angepasst auf das System während der Laufzeit in den Graph eingreift. Dies ist wesentlich für Echtzeit-Programmieren von Pd und damit Programmieren bzw. Komponieren zum Teil einer Performance werden kann.

Der Computer selbst besitzt von vornherein nicht die Eigenschaft eines

Datenflusssystems, sondern er muss erst dazu eingeschränkt werden sich als solches anzubieten. Dieses Datenflussparadigma macht aus einem Computer eine Maschine, welche ununterbrochen Daten fließen lässt und damit ein Werkzeug zum Bau von Maschinen. Dies ist ein Umgang mit Daten, der sich von anderen Programmen unterscheidet.

Ein solches Programm ermöglicht damit ein reaktives System in Echtzeit. Es beobachtet ununterbrochen das umgebende System mit seinen Interfaces und reagiert auf deren Information (Aktionen) in Echtzeit, im Gegensatz zu einem transformationellen System, welches nur an vorgesehenen Punkten des Programmablaufes mit dem umgebenden System reagiert. Damit ist Pd eine typische Definition von robotischen Systemen.

Da Datenflussdiagramme grafische Notationen sind, lag es nahe, die Programmiersprachen dazu mit grafischer Repräsentationen zu realisieren, die sogenannte "grafische Programmierung" oder "Visual Programming" und ahmt auch das Patchen von frühen analogen Synthesizern nach, wobei Signale mittels Kabel durch Geräte geleitet werden. Von dort stammt auch die Terminologie "patchen,...".

> The visual-patching paradigm actually has its routes in signal processing textbooks and early analog hardware (i.e., analog synthesis, analog computers, etc.) and is thus a natural means for expressing constructs that are ultimately signal processors and controllers. Even systems that originally started out as text based, such as CSound, now boast visual front-ends ("The on-line graphical specification of computer procedures." written 1966 William Robert Sutherland siehe

Wahrscheinlich wurde die erste grafische Programmiersprache 1966 von William Robert Sutherland geschrieben. Die wichtigsten Programme dieser Gattung wurden jedoch in den 1980er Jahren entwickelt. [6]

Hier ein Definition von Meyers:

> Visual Programming (VP) refers to any system that allows the user to specify a program in two-(or more)-dimensionsional fashion. [...] conventional textual languages are not considered

[6] Guy L. Steele, Jr., The Definition and Implementation of a Computer Programming Language Based on Constraints, Massachusetts Institute of Technology, Cambridge, MA, 1980

two dimensional since the compilers or interpreters process them as long, one-dimensional streams. [7]

[7]"Taxonomies of Visual Programming and Program Visualization", B. A. Myers, 1990, journal jvlc 1, pages= "97-123", volume 1

5.8.1 The Synchronous Data-flow Concept

There are a lot of possibilities doing real-time digital signal processing, but in computer music there is a preference to use a kind of data-flow model.[8]. Since native *data-flow computers* were not commercial implemented until now, there is a need to code this principles in conventional *van Neumann computers*.

Here we take the term *data-flow* as the idea, that data is ,,flowing" through operations and not like stacks of operations computed in sequence (statements flowing through a Processor). The program is represented as a data-flow-graph (directed graph) in which vertices, called *actors*, represent computations and the edges represent *FIFO channels*. These channels queue data values, encapsulated in objects called *tokens*, which are passed from the output of one computations to the input of another. These tokens can be anything meaningful, e.g. in the time domain *sample arrays* or in frequency domain *spectral arrays*.

With *synchronous* data-flow I mean, that all data flows are synchronized, so that in one graph all data is processed at the same rate (that is not the sample rate, but mostly a division of it). These makes all scheduling much easier than doing asynchronous computations. To handle different sample rates or similar concepts a mechanism is implemented known as down- and up-sampling.

In this system there is no direkt concept of controlling the processed data with events, but this is done in a separate task, which could be synchronized to the signalprocess. Parameter control could also be done through IPC-concepts like shared memory asynchronous to the signal-flow.

So there are three main issues which must be supported, handling data for data-flow in the form of *buffers*, handling the computation in form of *operations* and handling the data-flow-graphs doing the scheduling/dispatching and providing a *data-flow interface*.

The power of a system lies mainly in simplicity of calling the operation stack, efficient use of buffers, also called signals, and probably the most important, an code optimized library of operations. In modern computers with complicated and different architectures of prefetches, cache and so on, coding should be done not to complicate so it will give the compiler more opportunities for code optimation, especially if we have different target

[8]developed for early Computer Music languages like Music V, which still is a basic idea of most of the computer systems generating music

CPU-architectures, where this is handled different.

Memory usage is not anymore an issue on computers, but cache sizes had become as big as the whole RAM memory in previous days, so fitting all signals, storage and code in the secondary (or third) or better in the primary cache will significant speed up the process. So efficient buffer handling can improve the computation speed.

Buffer as Connections

In dsp context we talked about signals as audio-data, but this word is very much occupied by the "Unix signals" or "signals" in libraries like GTK[9] or QT[10], so here the word "connection" to connect operations inputs to outputs is used. In scientific books, they sometimes known as edges to actors (= operation) or arrows, are a kind of FIFO-channels also called tokens, since it represent the data in the data-flow graph.

The implementation of connections are buffers, which the operations use as data for their computations. Since we want to use as few as possible buffers, one buffer can be used by several operation at the same schedule tick for different purposes, if their length and organization fits their tasks.

Buffers are assigned at schedule time of audio processing, not at construction time. This is important because the buffer length is defined as global that can be changed at run time. In an some versions this can be done during process time, which assumes that operations always get the actual number of tokens per schedule.

Constant buffers are buffers owned by the operation and which cannot be changed by the scheduler, so static or operational data can be stored in it. This should be used only for output buffers and must be declared at initialize or construction phase of an operation. Be careful to respect the buffer-Len for the operation.

Default buffers are constant buffers which are used as input buffer if no connection is made to it.

5.8.2 The Graph

A graph is a collection of operations connected by buffers. For most systems a directed graph is used, which means an output of one operation points to one

[9]The Gnu Tool Kit used eg. for gnome

[10]Troll Techs Toolkit QT used eg. by KDE

or more inputs of other operations, one input can be feed only by one output. There are operations without inputs (or not connected inputs) which are called sources and operations with no output or unconnected outputs which are called drains. (Actually the input points to a output in most software implementations, but for constructing the DSP-Graph it is more easier to think the other way round.)

Here no recursion is allowed. There is a lot of discussion how to implement recursion in data-flow graphs, because it always implies a delay of at least one buffer length somewhere in the recursion loop, and it is often hard to decide where it happens and therefore unpredictable behavior can occur. So this has been forbidden. If some ones needs recursion he can do it with a special pair of operations mostly called "throw" and "catch" to realize this and define the delay at this point.

navigate in operation list

There are several methods to navigate in the *Sychronous Data Flow Graph* (`SDFGraph`). A first approach is to go through the list of operations or connections or search a connection or operation in the list with its pointer. A second method is search the drains or sources and follow the connections.

5.8.3 Scheduler - Dispatcher

There are a lot of scheduling algorithms available, but in use of SDFGraphs and a single process dispatcher it is quit straight forward to solve. There are several problems which the scheduler has to do:

1. to test the SDF graph for recursion- or connection failures
2. to get the execution graph out of the data paths
3. to test the buffer consistency and optimize buffer usage
4. assign the data buffers.

The next tasks are to do for the dispatcher, that is:

1. split the dispatch process as child from main process
2. assign the shared memory if needed

3. assign the additional needed memory for the operation
4. start/stop execution loop
5. run the execution graph
6. free resources and terminate the child process

scheduling algorithm

There should be different scheduler algorithms to choose from, for specific solutions. One of them is a straight forward one which just makes an execution stack and the dispatcher runs this.

Another one is a conditional scheduler called "boolean SDF scheduler", where the output of the operation is tested for change and operation are executed only when needed with the overhead of testing.

Here is a description of such an algorithm, showing the problems and pitfalls that can be encountered:

We have a data-flow graph as shown in figure5.6. The job of the scheduler is to get the processing order of the operations and the assignment of buffers, with the goal to use as few buffers as possible.

A naive approach to tackle this problem, is to trace the graph in a top-down approach, where the signal is generated at the top and flows down through the operations.

We start with an arbitrary source (an object with no inputs), that is to be computed first and assign it a number *1*), and assign two buffers *a* and *b* to its outputs. These output buffers are automatically propagated as inputs to all connected objects. Since both outputs of *1* are connected to objects that have additional inputs which are not yet assigned, we have to skip these for now.

We continue by searching for another source object (*2*), and assign another buffer *c* to its sole output. Again, this buffer gets assigned as inputs for all the connected objects.

Next we search for an object, that has all inputs assigned and find *3*. We can reuse the input buffers as outputs, and propagate the buffers to the connected objects.

Repeating these steps, we assign buffers for *4* (re-using buffers *a* and *b*) and *5*, which is a drain (that is, it has no more outputs), so we continue by finding the next unassigned object (backtracking from *5* to *4* and then to *3*,

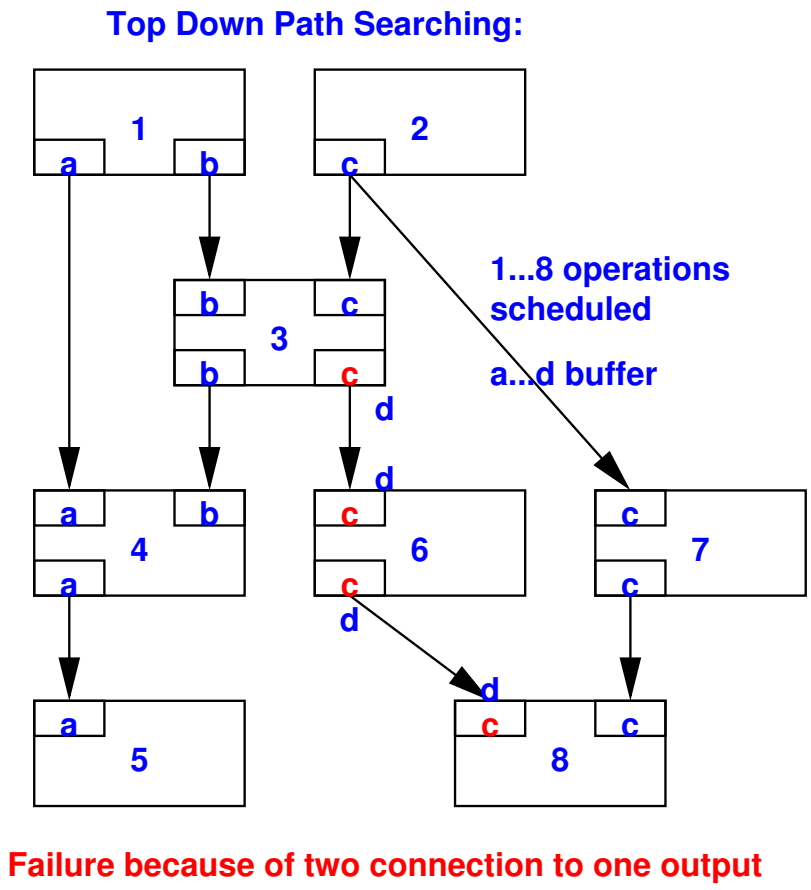


Abbildung 5.6: Top-Down Schedule of an example graph

always looking for a connected object that has not been done yet), and find *6*. Repeating these steps, we come up with a graph where all the objects have input and output buffers assigned.

During execution of the graph, the objects are called in the order of their assigned numbers (*1* to *8*).

Unfortunately the algorithm gave us a faulty buffer assignment, as we can see that *8* has the buffer *c* assigned for both its inputs (thus effectively having the same token on both inputs, which is unlikely to be true, if *6* and *7* are modifying the inputs in different ways).

To solve this correctly, we would need to use a 4th buffer *d* for one of the inputs of *8*.

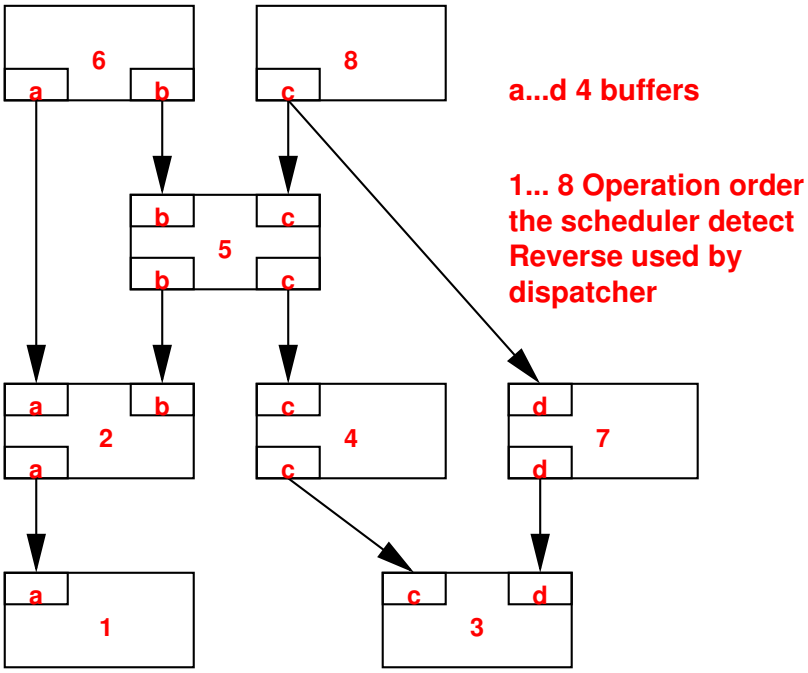


Abbildung 5.7: Bottom-Up Schedule of an example graph

The problem arises because one output is allowed to have multiple connections (in our example, the output of object *2* is connected to both *3* and *7*).

A better solution is shown in figure 5.7, which uses a bottom-up approach, by starting with an arbistrary drain and tracking connections to its input(s).

The resulting object indices are reversed regarding the execution order (thus, objects have to be called from *8* to *1*).

This algorithm also works on independent parallel graphs.

Another problem is recursion.

As shown in figure 5.8, the scheduling algorithm will stop if it encounters a recursive connection, as it is unable to find enough drains (or sources, in the top-down case) in order to add all the objects to the operation stack.

straight forward scheduler

The scheduler is run every time the DSP-engine is started or a connection is made/changed or a operation is inserted/removed/changed.

Following steps are performed:

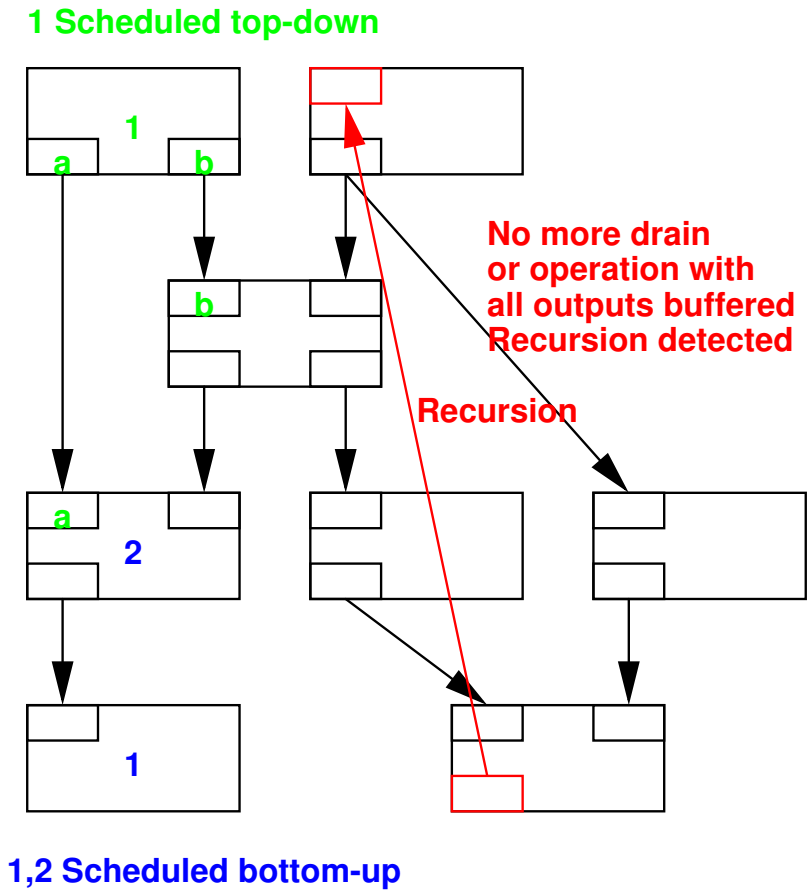


Abbildung 5.8: Recursion Problem on scheduling algorithm

1. The scheduling algorithm is performed to assign buffers, detect recursion and get the order of execution of the operation, putting them on the operation stack.

2. The scheduler scans the operations for not connected inputs to prepare the Constant Buffers for that.

3. An algorithm is performed to start with sources and their buffer-length, and looks for needed buffer sizes in the following operations.

4. If the above succeeded, the dispatcher might be forked into a different thread. After that, buffer space gets allocated and assigned to the

buffer. [11]

The Time Source and AD/DA Interface

For many applications it is necessary to have a synchronous time with the input and output of samples on the computer. Since oscillators on AD/DA can differ from the system-clock, it is better to use the AD/DA as time source. Also the time can float a little bit because of Audio I/O Buffers, so the computation for the sound is correct.

Since more than one AD/DA-channel can be used, there is the possibility to use multiple IO-Streams (such as provided by multiple sound-cards) in parallel. In this case, the user has to guarantee that the AD/DAs are synchronized (by hardware). The Operation `DAC` or `ADC` can be used to get the data from this to/from the SDFGraph.

As parameter the sample-rate, -format and buffersizes (which are the "advance") can be adjusted, but it has to be known which of these parameters the hardware supports.

5.8.4 Signale und Buffer

in einen synchronen Datenflusssystem gibt sind eigentlich Samplerate und Buffergrössen fixiert. Nun gibt es doch Möglichkeiten mit verschiedenen Vektoren und Buffern umzugehen.

Beschreibung ausgehend vom Bedarf, FFT-Signale zu verwenden, wird das Block System erklärt.

[11] This ensures that that buffer space is only allocated in the forked process and not doubled. This does not improve much on current Unix systems, because page faults are only produced if memory is accessed by both processes, which should never happen to buffer space. However, it is possible that this gives some improvement on other systems or multiprocessor architectures.

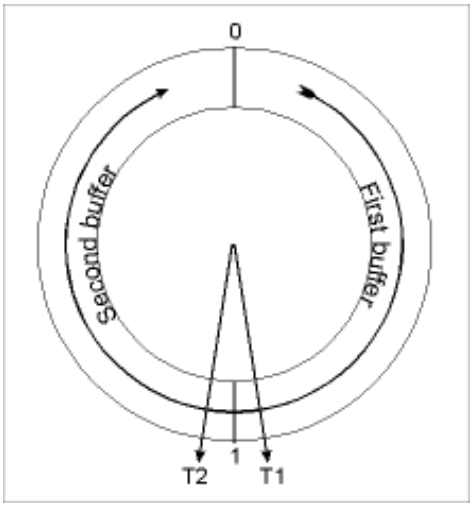


Abbildung 5.9: Double Buffer

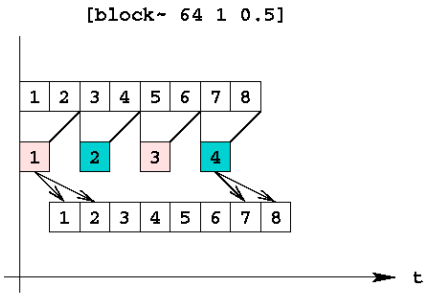


Abbildung 5.10: Block Umschichtung um 0.5

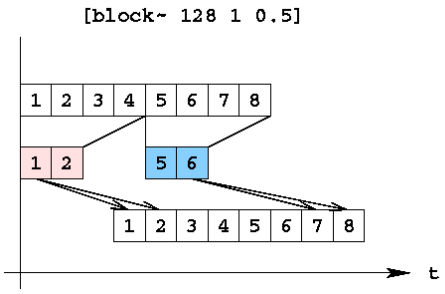


Abbildung 5.11: Block 128 1 0.5

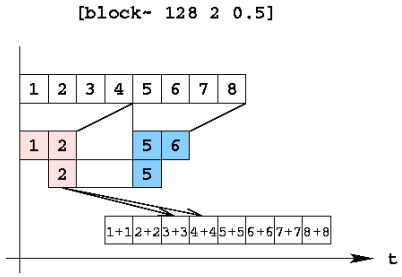


Abbildung 5.12: Block 128 2 0.5

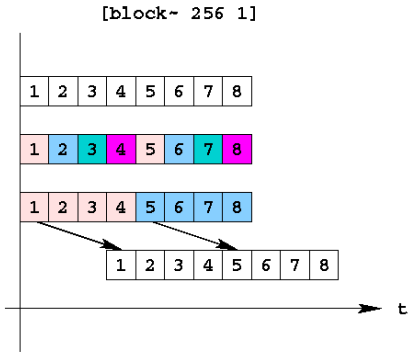


Abbildung 5.13: Blockumschichtung auf 256 Samples

5.9 Event Orientierte Systeme

Das Prinzip der Eventsteuerung lernten wir schon bei MIDI kennen. Dieses kann jedoch nochmals generalisiert werden. Der ursprüngliche Ausgangspunk von Messages ist die Beschreibung von Aktionen beim Musizieren, wie ,,das Spielen einer Note" oder ,,das Bedienen eines Regler" und damit eigentlich Steuern von Geräten.

Ein Ereignis besteht aus dem Zeitpunkt seines Auftretens und einer Datenrepräsentation des Ereignis. Diese Ereignisse werden meist "Messages" genannt und stellen Nachrichteneinheiten dar.

5.9.1 Message System

Die meisten grafischen Programmiersprachen orientierten sich an MAX/FTS von Ircam um ihr Message-System aufzubauen. Da sich zwischen den verschiedenen Programmen meist nur wenig in der Architektur unterscheidet können wir hier als Beispiel das Ereignissystem von `Pd` betrachten. Da dieses auch als Sourcecode vorliegt, können Internas besser analysiert werden.

Anhand der Implementation eines Ereignissystems in `Pd` werden im Folgendem die Eigenschaften und Anforderungen an ein solches System beschrieben.

5.9.2 Anatomie einer Message

Eine Nachricht, engl. Message, besteht aus einer **Typdefinition**, dem *Selector* und einer beliebigen Anzahl Argumenten.

Wie bei modernen Interpretersprachen üblich, wird der Typ einer Message zur Laufzeit festgelegt, zum Beispiel bei *python*.

$Message = Selector[Arguments]$;

$Arguments = [Arg1][Arg2]...[ArgN]$

Ein *Selector* ist eine `Symbol`, das den Typ der Argumente festlegt, wobei es mehrere vordefinierte Typen gibt, ein Automatismus ist jedoch eingebaut: ,,als Selector ist alles was nicht als Zahl interpretiert werden kann ein Symbol, wobei dieses keine Leerzeichen, Beistriche und Strichpunkte enthalten darf".

Bei den Dataflow-Programmen ist der Selector frei wählbar, beziehungsweise wird er zur Laufzeit definiert.

bang: Null Message ohne Argumente

float: Gleitkommanzahl (wird auch für Integer verwendet)

symbol: Ein Zeichenkette als Symbol, wird für Namen verwendet und wird in einer Tabelle indiziert angelegt.

list: eine Liste von Messages außer der Liste selbst (und bang nur symbolisch)

signal: kann nicht im Message System verwendet werden und ist für Signale reserviert.

Dazu gibt es zur besseren Handhabbarkeit Erweiterungen und Abkürzungen.

Anything: ist ein Platzhalter für jede beliebige Message.

Zahl: Weiters werden Zahlen auch ohne Selector als Zahlen erkannt.

Es gibt hier kein Konzept des "Strings", was bei der Verarbeitung von sehr vielen Worten ein Performanceproblem darstellt.

Alle Symbole werden in einer "Hashtable" gespeichert und über Zeiger referenziert. ies hat den Vorteil das Speicherplatz gespart wird und wie in Pd mit einen 12Bit Hash gespeichert.

5.9.3 Senden von Messages

Messages werden in Pd über eine Verbindungstabelle mit einer Zielfunktion assoziert, welche mit dem Argumenten der Message aufgerufen wird. Diese Zielfunktion wird durch den Eingang eines Objektes und den Selector der Message bestimmt. Sendet diese Funktion wieder eine Message, so kann diese wiederum eine Funktion aufrufen. Erst wenn die Funktion keine Messages mehr versendet, oder keine Funktion mehr assoziert ist, wird deren Verarbeitung beendet.

Wenn eine Funktion aufgerufen wird, so werden deren Argumente und Rücksprung-Addresse auf einen Funktionsstack abgelegt.

Damit wird mittels Verbindungen (Connections) eine bedingte Funktionsaufrufskette definiert, welche durch Senden einer Message in diese Kette eine entsprechenden Vergrösserung des Funktionsstacks bewirkt. Überschreitet dieser Stack eine bestimmte Grösse, gibt es (wahrscheinlich) einem rekursiven Aufruf, was auch als Rückkoppelung bezeichnet wird. In diesem Fall wird ein "Stack Overflow" Fehler generiert und diese Funktionskette unterbrochen.

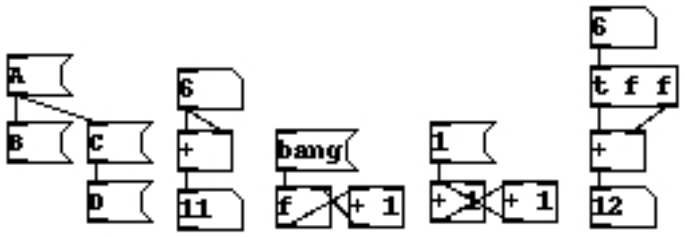


Abbildung 5.14: Beispiele von Versand von Messages in Pd

Eine weitere Logik zur Weiterleitung von Messages, gibt es durch einen Message Stack, welcher über das Send-Objekt oder intern in Funktionen verwendet wird. Hier wird die Message auf einen Message Stack gelegt und dieser wird dann abgearbeitet bis er leer ist.

Eine dritte Logik ist Messages mit einer Zeit zu versehen, zu der sie aufgerufen werden sollten. Damit lassen sich Messages auf spätere Zeitpunkte vertagen oder gezielt Zeitstrukturen generieren.

Beispiele von Versenden von Messages und deren Probleme in Abbildung 5.14

5.9.4 Reaktionsanforderung

5.9.5 Zeitlinie

5.10 Multirate Systeme

Dabei können paralell mehrere Sampleraten auftreten.

5.10.1 Up- and Downsampling

Samplerate-Konversion in der Anwendung.

5.11 Variable Rate Systems

Idee ist das sich die Samplerate der Leistungsmöglichkeit des Computers anpasst, wie es in der Grafik bei OpenGL der Fall ist.

Anhang A

Anhang

A.1 Liste der universitaeren Softwareentwicklung bis 1994

Es gibt viele Computermusik -Programme, von denen die an Universitäten entwickelten und/oder nicht kommerziellen Softwarepakete auflistet sind. Diese Liste ist nicht vollständig und wurde von der ICMA[1] herausgegeben als Beispiel für weltweite Entwicklungen in diesem Bereich. Danach explortierten diese Programme, wovon sich aber nur wenige über die Zeit durchsetzten.

```
##########################################################################
##########################################################################

ICMA Computer-Music Software Library --> Index

Last Updated:  10.11.94

Program Numbers Updated since last version:  NONE

Program Numbers Added since last version:  33

Copyright 1992 - 1994, International Computer Music Association

##########################################################################
##########################################################################

1   Ravel (composition):  James Binkley
2   PIP - Program In the shape of a Pear (composition):  James Binkley
3   Bol Processor (composition):  Bernard Bel
4   Lemur/LemurEdit package (audio analysis):  Kelly Fitz and Bryan Holloway
5   Cellular Automata Music 1.0 (composition):  Dale Millen
6   CMU MIDI Toolkit (MIDI):  Roger Dannenberg
7   CCRMA Music Kit and DSP Tools Distribution (sound synthesis):
                                          David A. Jaffe and Julius O. Smith
8   HMSL (algorithmic composition):  Phil Burk and Larry Polansky
9   DMIX (composition):  Daniel V. Oppenheim
10   CSOUND (sound synthesis):  Barry Vercoe
11   CMIX (composition):  Paul Lansky
12   Symbolic Composer (composition):  Nigel Morgan and Peter Stone
```

[1] International Computer Music Association

```
13   MODE (composition):  Stephen T. Pope
14   Soundhack (sound synthesis):  Tom Erbe
15   Nyquist (composition):  Roger Dannenberg
16   Csnd.app (sound synthesis):  Stephen David Beck
17   CONVERT (audio analysis):  Jesus Villena
18   CSOUND (Native Mode) for the Power Macintosh (sound synthesis):
19   Common Music for Windows 3.1 (composition):  Heinrich Taube and Joe Fosco
20   SoftSamp for Windows/CSOUND (sound synthesis):  Dustin Barlow
21   Sound Utility Programs for NeXTStep Computers (Utility):  Jean Laroche
22   abc2mtex (representation):  Chris Walshaw
23   MiXViews (sound synthesis):  Douglas A. Scott
24   inSanity (algorithmic composition):  Garth T. Zenie
25    (sound synthesis):  Amir Guindehi
27   IMPROVISE (composition):  David Pannett
28   FORMULA (Forth Music Language) (algorithmic composition):  David Anderson
29   STOCHGRAN (granular synthesis):  Mara M. Helmuth
30   Aleatoric Composer (algorithmic composition):  Carl Christensen
31    (sound synthesis):  Roberto H. Bamberger
32   LucyTuning Codes (composition):  Charles Lucy
33   Musical Set Complete (composition):  Craig Shoemaker
```

Hex	STATUS Binary D7-D0		NUMBER OF DATA BYTES	DESCRIPTION
Channel Voice Messages				
8nH	1000nnnn		2	Note Off
9nH	1001nnnn		2	Note On (a velocity of 0 = Note Off)
AnH	1010nnnn		2	Polyphonic key pressure/Aftertouch
BnH	1011nnnn		2	Control change
CnH	1100nnnn		1	Program change
DnH	1101nnnn		1	Channel pressure/After touch
EnH	1110nnnn		2	Pitch bend change
Channel Mode Messages				
BnH	1011nnnn	(01111xxx)	2	Selects Channel Mode
System Messages				
F0H	11110000		*****	System Exclusive
	11110sss		0 to 2	System Common
	11111ttt		0	System Real Time

NOTES:

nnnn:	N-1, where N = Channel #, i.e. 0000 is Channel 1, 0001 is Channel 2, and 1111 is Channel 16.
*****:	0iiiiiii, data, ..., EOX
iiiiiii:	Identification
sss:	1 to 7
ttt:	0 to 7
xxx:	Channel Mode messages are sent under the same Status Byte as the Control Change messages (BnH). They are differentiated by the first data byte which will have a value from 121 to 127 for Channel Mode messages.

Abbildung A.1: Tabelle 1: Zusammenfassung der Status-Bytes

A.2 Auszug aus MIDI-Spezifikation

A.2.1 MIDI V1.0 Syntax Tabellen

laut MIDI-Standard V1.0 Spezifications.

TABLE II

CHANNEL VOICE MESSAGES

STATUS Hex	STATUS Binary	DATA BYTES	DESCRIPTION
8nH	1000nnnn	0kkkkkkk 0vvvvvvv	Note Off vvvvvvv: note off velocity
9nH	1001nnnn	0kkkkkkk 0vvvvvvv	Note On vvvvvvv ≠ 0: velocity vvvvvvv = 0: note off
AnH	1010nnnn	0kkkkkkk 0vvvvvvv	Polyphonic Key Pressure (Aftertouch) vvvvvvv: pressure value
BnH	1011nnnn	0ccccccc 0vvvvvvv	Control Change (See Table III) ccccccc: control # (0-120) vvvvvvv: control value ccccccc = 121 thru 127: Reserved. (See Table IV)
CnH	1100nnnn	0ppppppp	Program Change ppppppp: program number (0-127)
DnH	1101nnnn	0vvvvvvv	Channel Pressure (Aftertouch) vvvvvvv: pressure value
EnH	1110nnnn	0vvvvvvv 0vvvvvvv	Pitch Bend Change LSB Pitch Bend Change MSB

NOTES:

1. nnnn: Voice Channel number (1-16, coded as defined in Table I notes)

2. kkkkkkk: note number (0 - 127)

3. vvvvvvv: key velocity
   A logarithmic scale is recommended.

4. Continuous controllers are divided into Most Significant and Least Significant Bytes. If only seven bits of resolution are needed for any particular controllers, only the MSB is sent. It is not necessary to send the LSB. If more resolution is needed, then both are sent, first the MSB, then the LSB. If only the LSB has changed in value, the LSB may be sent without re-sending the MSB.

Abbildung A.2: Tabelle 2: Kanal Stimmen Befehle

CONTROL NUMBER (2nd Byte value)		CONTROL FUNCTION
Decimal	Hex	
0	00H	Undefined
1	01H	Modulation wheel or lever
2	02H	Breath Controller
3	03H	Undefined
4	04H	Foot controller
5	05H	Portamento time
6	06H	Data entry MSB
7	07H	Main volume
8	08H	Balance
9	09H	Undefined
10	0AH	Pan
11	0BH	Expression Controller
12-15	0C-0FH	Undefined
16-19	10-13H	General Purpose Controllers (#'s 1-4)
20-31	14-1FH	Undefined
32-63	20-3FH	LSB for values 0-31
64	40H	Damper pedal (sustain)
65	41H	Portamento
66	42H	Sostenuto
67	43H	Soft pedal
68	44H	Undefined
69	45H	Hold 2
70-79	46-4FH	Undefined
80-83	50-53H	General Purpose Controllers (#'s 5-8)
84-90	54-5AH	Undefined
91	5BH	External Effects Depth
92	5CH	Tremolo Depth
93	5DH	Chorus Depth
94	5EH	Celeste (Detune) Depth
95	5FH	Phaser Depth
96	60H	Data increment
97	61H	Data decrement
98	62H	Non-Registered Parameter Number LSB
99	63H	Non-Registered Parameter Number MSB
100	64H	Registered Parameter Number LSB
101	65H	Registered Parameter Number MSB
102-120	66-7BH	Undefined
121-127	79-7FH	Reserved for Channel Mode Messages

Abbildung A.3: Tabelle 3: Kontroller Nummern

Parameter Number		Function
LSB	MSB	
00H	00H	Pitch Bend Sensitivity
01H	00H	Fine Tuning
02H	00H	Coarse Tuning

Abbildung A.4: Tabelle 3a: Registrierte Parameter

STATUS Hex	STATUS Binary	DATA BYTES	DESCRIPTION
Bn	1011nnnn	0ccccccc 0vvvvvvv	Mode Messages ccccccc = 121: Reset All Controllers vvvvvvv = 0 ccccccc = 122:Local Control vvvvvvv = 0, Local Control Off vvvvvvv = 127, Local Control On ccccccc = 123: All Notes Off vvvvvvv = 0 ccccccc = 124: Omni Mode Off (All Notes Off) vvvvvvv = 0 ccccccc = 125:Omni Mode On (All Notes Off) vvvvvvv = 0 ccccccc = 126: Mono Mode On (Poly Mode Off) (All Notes Off) vvvvvvv = M, where M is the number of channels. vvvvvvv = 0, the number of channels equals the number of voices in the receiver. ccccccc = 127: Poly Mode On (Mono Mode Off) (All Notes Off) vvvvvvv = 0

NOTES:

1. nnnn: Basic Channel number (1-16)

2. ccccccc: Controller number (121 - 127)

3. vvvvvvv: Controller value

Abbildung A.5: Tabelle 4: Kanal Modus Befehle

STATUS Hex	STATUS Binary	DATA BYTES	DESCRIPTION
F1H	11110001	0nnnnddd	MIDI Time Code Quarter Frame nnn: Message Type dddd: Values
F2H	11110010	0lllllll 0hhhhhhh	Song Position Pointer lllllll: (Least significant) hhhhhhh: (Most significant)
F3H	11110011	0ssssss	Song Select ssssss: Song #
F4H	11110100		Undefined
F5H	11110101		Undefined
F6H	11110110	none	Tune Request
F7H	11110111	none	EOX: "End of System Exclusive" flag

Abbildung A.6: Tabelle 5: Systemweite Befehle

STATUS Hex	STATUS Binary	DATA BYTES	DESCRIPTION
F8H	11111000		Timing Clock
F9H	11111001		Undefined
FAH	11111010		Start
FBH	11111011		Continue
FCH	11111100		Stop
FDH	11111101		Undefined
FEH	11111110		Active Sensing
FFH	11111111		System Reset

Abbildung A.7: Tabelle 6: Systemweite Echtzeit Befehle

STATUS Hex	STATUS Binary	DATA BYTES	DESCRIPTION
F0H	11110000	0iiiiiii . (0ddddddd) . . . (0ddddddd)	Bulk dump, etc. iiiiiii: identification (See note 1) Any number of data bytes may be sent here, for any purpose, as long as they all have a zero in the most *significant bit*.
F7H	11110111		EOX: "End of System Exclusive"

NOTES:

1. iiiiiii: identification ID (0-127). If the ID is 0 the following two bytes are used as extensions to the manufacturer *ID*.

2. All bytes between the System Exclusive Status byte and EOX must have zeroes in the MSB.

3. A manufacturer's ID number can be obtained from the MMA or JMSC.

4. Status or Data bytes (except Real-Time) should not be interleaved with System Exclusive

5. No Status Bytes (other than Real-time) should be sent until after an EOX has terminated the System Exclusive message. If however, a Status Byte other than EOX is received during a System Exclusive message, the message is *terminated*.

6. Three System Exclusive ID numbers have been set aside for special purposes: 7DH is reserved for *non-commercial* use (e.g. schools, research, etc.) and is not to be used on any product released to the public; 7EH (Non-Real Time) and 7FH (Real Time) are used for extensions to the MIDI specification.

Abbildung A.8: Tabelle 7: System Exklusiv Befehle

SUB-ID #1	SUB-ID #2	DESCRIPTION
Non-Real Time (7EH)		
00	--	**Unused**
01	(not used)	**Sample Dump Header**
02	(not used)	**Sample Data Packet**
03	(not used)	**Sample Dump Request**
04	nn	**MIDI Time Code**
	00	Special
	01	Punch In Points
	02	Punch Out Points
	03	Delete Punch In Point
	04	Delete Punch Out Point
	05	Event Start Point
	06	Event Stop Point
	07	Event Start Points with additional info.
	08	Event Stop Points with additional info.
	09	Delete Event Start Point
	0A	Delete Event Stop Point
	0B	Cue Points
	0C	Cue Points with additional info.
	0D	Delete Cue Point
	0E	Event Name in additional info.
05	nn	**Sample Dump Extensions**
	01	Multiple Loop Points
	02	Loop Points Request
06	nn	**General Information**
	01	Identity Request
	02	Identity Reply
7C	(not used)	**Wait**
7D	(not used)	**Cancel**
7E	(not used)	**NAK**
7F	(not used)	**ACK**
Real Time (7FH)		
00	--	**Unused**
01	nn	**MIDI Time Code**
	01	Full Message
	02	User Bits

NOTES:

1. The standardized format for both Real Time and Non-Real Time messages is as follows: F0H <id number> <channel number> <sub-ID#1> <sub-ID#2>..... F7H

Abbildung A.9: Tabelle 8: Systemn Exklusiv ID Messages

NUMBER	MANUFACTURER	NUMBER	MANUFACTURER
	American Group		**European Group**
01H	Sequential	20H	Passac
02H	IDP	21H	SIEL
03H	Octave-Plateau	22H	Synthaxe
04H	Moog	24H	Hohner
05H	Passport Designs	25H	Twister
06H	Lexicon	26H	Solton
07H	Kurzweil	27H	Jellinghaus MS
08H	Fender	28H	Southworth
0AH	AKG Acoustics	29H	PPG
0BH	Voyce Music	2AH	JEN
0CH	Waveframe Corp	2BH	SSL Limited
0DH	ADA	2CH	Audio Veritrieb
0EH	Garfield Elec.	2FH	Elka
0FH	Ensoniq	30H	Dynacord
10H	Oberheim		**Japanese Group**
11H	Apple Computer		
12H	Grey Matter Response	40H	Kawai
14H	Palm Tree Inst.	41H	Roland
15H	JL Cooper	42H	Korg
16H	Lowrey	43H	Yamaha
17H	Adams-Smith	44H	Casio
18H	Emu Systems	46H	Kamiya Studio
19H	Harmony Systems	47H	Akai
1AH	ART	48H	Japan Victor
1BH	Baldwin	49H	Meisosha
1CH	Eventide	4AH	Hoshino Gakki
1DH	Inventronics	4BH	Fujitsu Elect
1FH	Clarity	4CH	Sony
		4DH	Nisshin Onpa
00H 00H 07H	Digital Music Corp.	4EH	TEAC Corp.
00H 00H 0BH	IVL Technologies	4FH	System Product
00H 00H 0CH	Southern Music Systems	50H	Matsushita Electric
00H 00H 0DH	Lake Butler Sound	52H	Fostex
00H 00H 10H	DOD Electronics		
00H 00H 14H	Perfect Fretworks		
00H 00H 16H	Opcode		
00H 00H 18H	Spatial Sound		
00H 00H 19H	KMX		
00H 00H 20H	Axxes		

* Current as of June, 1988

Abbildung A.10: Tabelle 9: System Hersteller IDs, Stand Juni 1988

A.2.2 General MIDI V 2.0

```
Summary of GM2 Requirements

GENERAL REQUIREMENTS
 Number of Notes: 32 simultaneous notes
 MIDI Channels: 16
  - Simultaneous Melodic Instruments = up to 16 (all Channels)
  - Simultaneous Percussion Kits = up to 2 (Channel 10/11)
SUPPORTED CONTROL CHANGE MESSAGES (Some Optional)
 - Bank Select (cc#0/32)
 - Modulation Depth (cc#1)
 - Portamento Time (cc#5)
 - Channel Volume (cc#7)
 - Pan (cc#10)
 - Expression (cc#11)
 - Hold1 (Damper) (cc#64)
 - Portamento ON/OFF (cc#65)
 - Sostenuto (cc#66)
 - Soft (cc#67)
 - Filter Resonance (Timbre/Harmonic Intensity) (cc#71)
 - Release Time (cc#72)
 - Attack time (cc#73)
 - Brightness (cc#74)
 - Decay Time (cc#75)
 - Vibrato Rate (cc#76)
 - Vibrato Depth (cc#77)
 - Vibrato Delay (cc#78)
 - Reverb Send Level (cc#91)
 - Chorus Send Level (cc#93)
 - Data Entry (cc#6/38)
 - RPN LSB/MSB (cc#100/101)
SUPPORTED RPNs (Registered Parameter Numbers)
 - Pitch Bend Sensitivity
 - Channel Fine Tune
 - Channel Coarse Tune
 - Modulation Depth Range (Vibrato Depth Range)
 - RPN NULL
SUPPORTED UNIVERSAL SYSTEM EXCLUSIVE MESSAGES
 - Master Volume
 - Master Fine Tuning
 - Master Coarse Tuning
 - Reverb Type
 - Reverb Time
 - Chorus Type
```

```
 - Chorus Mod Rate
 - Chorus Mod Depth
 - Chorus Feedback
 - Chorus Send to Reverb
 - Controller Destination Setting
 - Scale/Octave Tuning Adjust
 - Key-Based Instrument Controllers
 - GM2 System On
GM 2 INSTRUMENT SOUND SET
GM 2 PERCUSSION SOUND SET
```

A.2.3 Downloadable Sounds Overview

Overview of Key DLS-2 Elements

The DLS Level 2 synthesizer consists of the following basic elements for each voice:

- A sampled sound source with loop and release
- Two 6-segment envelope generators characterized as DAHDSR (Delay-Attack-Hold-Decay-Sustain-Release)
- Two Low Frequency Oscillator (LFO) generators
- A low pass filter with resonance and dynamic filter cutoff frequency
- Standardized response to MIDI controllers

Minimum Device Requirements:

The device must meet the following minimum requirements to be considered compliant with the DLS Level 2 specifications:

- Minimum of 32 digital oscillators each with individually controlled DCA, DCF, LFO generators (two per oscillator), and envelope generators (two per oscillator).
- Minimum sample playback rate of 22.05 KHz
- Minimum sample memory of 1,048,576 x 16-bit words

- Minimum of 512 waves stored simultaneously (subject to sample memory constraints)
- Minimum of 256 instruments stored simultaneously (subject to articulation data constraints)
- Minimum of 1,024 regions stored simultaneously (subject to articulation data constraints)
- Minimum of 8,192 explicit connections stored simultaneously (does not include default connections)
- If the device claims support for both DLS and GM, it must be able to support both of them simultaneously without encroachment on the above minimum requirements

DLS Conditional File Chunks:

DLS Level 2 introduces new functionality into the control logic, namely a block called a ,,Conditional Chunk." Conditional Chunks can be used to create libraries that are both forward and backward compatible. As an example, an instrument in a library may contain DLS Level 1 chunks, DLS Level 2 chunks, and chunks for a proprietary device. The file parser will then select the appropriate chunks for the specific device in use.

A.3 OpenSound Control Specification

version 1.0, March 26 2002, Matt Wright

Introduction

OpenSound Control (OSC) is an open, transport-independent, message-based protocol developed for communication among computers, sound synthesizers, and other multimedia devices.

OSC Syntax

This section defines the syntax of OSC data.

Atomic Data Types

All OSC data is composed of the following fundamental data types:

int32 32-bit big-endian two's complement integer

OSC-timetag 64-bit big-endian fixed-point time tag, semantics defined below

float32 32-bit big-endian IEEE 754 floating point number

OSC-string A sequence of non-null ASCII characters followed by a null, followed by 0-3 additional null characters to make the total number of bits a multiple of 32. (OSC-string examples) In this document, example OSC-strings will be written without the null characters, surrounded by double quotes.

OSC-blob An int32 size count, followed by that many 8-bit bytes of arbitrary binary data, followed by 0-3 additional zero bytes to make the total number of bits a multiple of 32.

The size of every atomic data type in OSC is a multiple of 32 bits. This guarantees that if the beginning of a block of OSC data is 32-bit aligned, every number in the OSC data will be 32-bit aligned.

OSC Packets

The unit of transmission of OSC is an *OSC Packet*. Any application that sends OSC Packets is an *OSC Client*; any application that receives OSC Packets is an *OSC Server*.

An OSC packet consists of its *contents*, a contiguous block of binary data, and its *size*, the number of 8-bit bytes that comprise the contents. The size of an OSC packet is always a multiple of 4.

The underlying network that delivers an OSC packet is responsible for delivering both the contents and the size to the OSC application. An OSC packet can be naturally represented by a datagram by a network protocol such as UDP. In a stream-based protocol such as TCP, the stream should begin with an int32 giving the size of the first packet, followed by the contents of the first packet, followed by the size of the second packet, etc.

The contents of an OSC packet must be either an *OSC Message* or an *OSC Bundle*. The first byte of the packet's contents unambiguously distinguishes between these two alternatives.

OSC Messages

An OSC message consists of an *OSC Address Pattern* followed by an *OSC Type Tag String* followed by zero or more *OSC Arguments*.

Note: some older implementations of OSC may omit the OSC Type Tag string. Until all such implementations are updated, OSC implementations should be robust in the case of a missing OSC Type Tag String.

OSC Address Patterns

An OSC Address Pattern is an OSC-string beginning with the character '/' (forward slash).

OSC Type Tag String

An OSC Type Tag String is an OSC-string beginning with the character ',' (comma) followed by a sequence of characters corresponding exactly to the sequence of OSC Arguments in the given message. Each character after the comma is called an *OSC Type Tag* and represents the type of the corresponding OSC Argument. (The requirement for OSC Type Tag Strings to

start with a comma makes it easier for the recipient of an OSC Message to determine whether that OSC Message is lacking an OSC Type Tag String.)

This table lists the correspondance between each OSC Type Tag and the type of its corresponding OSC Argument:

The meaning of each OSC Type Tag

OSC Type Tag Type of corresponding argument

i int32

f float32

s OSC-string

b OSC-blob

Some OSC applications communicate among instances of themselves with additional, nonstandard argument types beyond those specified above. OSC applications are not required to recognize these types; an OSC application should discard any message whose OSC Type Tag String contains any unrecognized OSC Type Tags. An application that does use any additional argument types must encode them with the OSC Type Tags in this table:

OSC Type Tags that must be used for certain nonstandard argument types

OSC Type Tag Type of corresponding argument

h 64 bit big-endian two's complement integer

t OSC-timetag

d 64 bit ("double") IEEE 754 floating point number

S Alternate type represented as an OSC-string (for example, for systems that differentiate "symbols" from "strings")

c an ascii character, sent as 32 bits

r 32 bit RGBA color

m 4 byte MIDI message. Bytes from MSB to LSB are: port id, status byte, data1, data2

T True. No bytes are allocated in the argument data.

F False. No bytes are allocated in the argument data.

N Nil. No bytes are allocated in the argument data.

I Infinitum. No bytes are allocated in the argument data.

[Indicates the beginning of an array. The tags following are for data in the Array until a close brace tag is reached.

] Indicates the end of an array.

OSC Type Tag String examples.

OSC Arguments

A sequence of OSC Arguments is represented by a contiguous sequence of the binary representations of each argument.

OSC Bundles

An OSC Bundle consists of the OSC-string "#bundle" followed by an *OSC Time Tag*, followed by zero or more *OSC Bundle Elements*. The OSC-timetag is a 64-bit fixed point time tag whose semantics are described below.

An OSC Bundle Element consists of its *size* and its *contents*. The size is an int32 representing the number of 8-bit bytes in the contents, and will always be a multiple of 4. The contents are either an OSC Message or an OSC Bundle.

Note this recursive definition: bundle may contain bundles.

This table shows the parts of a two-or-more-element OSC Bundle and the size (in 8-bit bytes) of each part.

Parts of an OSC Bundle

Data Size Purpose

OSC-string "#bundle" 8 bytes How to know that this data is a bundle

OSC-timetag 8 bytes Time tag that applies to the entire bundle

Size of first bundle element int32 = 4 bytes First bundle element

First bundle element's contents As many bytes as given by "size of first bundle element"

Size of second bundle element int32 = 4 bytes Second bundle element

Second bundle element's contents As many bytes as given by "size of second bundle element"

etc. Addtional bundle elements

OSC Semantics

This section defines the semantics of OSC data.

OSC Address Spaces and OSC Addresses

Every OSC server has a set of *OSC Methods*. OSC Methods are the potential destinations of OSC messages received by the OSC server and correspond to each of the points of control that the application makes available. "Invoking"

an OSC method is analogous to a procedure call; it means supplying the method with arguments and causing the method's effect to take place.

An OSC Server's OSC Methods are arranged in a tree strcuture called an *OSC Address Space*. The leaves of this tree are the OSC Methods and the branch nodes are called *OSC Containers*. An OSC Server's OSC Address Space can be dynamic; that is, its contents and shape can change over time.

Each OSC Method and each OSC Container other than the root of the tree has a symbolic name, an ASCII string consiting of printable characters other than the following:

Printable ASCII characters not allowed in names of OSC Methods or OSC Containers

character name ASCII code (decimal)

' ' space 32

number sign 35

* asterisk 42

, comma 44

/ forward slash 47

? question mark 63

[open bracket 91

] close bracket 93

{ open curly brace 123

} close curly brace 125

The *OSC Address* of an OSC Method is a symbolic name giving the full path to the OSC Method in the OSC Address Space, starting from the root of the tree. An OSC Method's OSC Address begins with the character '/' (forward slash), followed by the names of all the containers, in order, along the path from the root of the tree to the OSC Method, separated by forward slash characters, followed by the name of the OSC Method. The syntax of OSC Addresses was chosen to match the syntax of URLs. (OSC Address Examples)

OSC Message Dispatching and Pattern Matching

When an OSC server receives an OSC Message, it must invoke the appropriate OSC Methods in its OSC Address Space based on the OSC Message's OSC Address Pattern. This process is called *dispatching* the OSC Message to the OSC Methods that *match* its OSC Address Pattern. All the matching

OSC Methods are invoked with the same argument data, namely, the OSC Arguments in the OSC Message.

The *parts* of an OSC Address or an OSC Address Pattern are the substrings between adjacent pairs of forward slash characters and the substring after the last forward slash character. (examples)

A received OSC Message must be disptched to every OSC method in the current OSC Address Space whose OSC Address matches the OSC Message's OSC Address Pattern. An OSC Address Pattern matches an OSC Address if

1. The OSC Address and the OSC Address Pattern contain the same number of parts; and

2. Each part of the OSC Address Pattern matches the corresponding part of the OSC Address.

A part of an OSC Address Pattern matches a part of an OSC Address if every consecutive character in the OSC Address Pattern matches the next consecutive substring of the OSC Address and every character in the OSC Address is matched by something in the OSC Address Pattern. These are the matching rules for characters in the OSC Address Pattern:

1. '?' in the OSC Address Pattern matches any single character

2. '*' in the OSC Address Pattern matches any sequence of zero or more characters

3. A string of characters in square brackets (e.g., "[string]") in the OSC Address Pattern matches any character in the string. Inside square brackets, the minus sign (-) and exclamation point (!) have special meanings:

    - two characters separated by a minus sign indicate the range of characters between the given two in ASCII collating sequence. (A minus sign at the end of the string has no special meaning.)
    - An exclamation point at the beginning of a bracketed string negates the sense of the list, meaning that the list matches any character not in the list. (An exclamation point anywhere besides the first character after the open bracket has no special meaning.)

4. A comma-separated list of strings enclosed in curly braces (e.g., "{foo,bar}") in the OSC Address Pattern matches any of the strings in the list.

5. Any other character in an OSC Address Pattern can match only the same character.

Temporal Semantics and OSC Time Tags

An OSC server must have access to a representation of the correct current absolute time. OSC does not provide any mechanism for clock synchronization.

When a received OSC Packet contains only a single OSC Message, the OSC Server should invoke the correponding OSC Methods immediately, i.e., as soon as possible after receipt of the packet. Otherwise a received OSC Packet contains an OSC Bundle, in which case the OSC Bundle's OSC Time Tag determines when the OSC Bundle's OSC Messages' corresponding OSC Methods should be invoked. If the time represented by the OSC Time Tag is before or equal to the current time, the OSC Server should invoke the methods immediately (unless the user has configured the OSC Server to discard messages that arrive too late). Otherwise the OSC Time Tag represents a time in the future, and the OSC server must store the OSC Bundle until the specified time and then invoke the appropriate OSC Methods.

Time tags are represented by a 64 bit fixed point number. The first 32 bits specify the number of seconds since midnight on January 1, 1900, and the last 32 bits specify fractional parts of a second to a precision of about 200 picoseconds. This is the representation used by Internet NTP timestamps.The time tag value consisting of 63 zero bits followed by a one in the least signifigant bit is a special case meaning "immediately."

OSC Messages in the same OSC Bundle are *atomic*; their corresponding OSC Methods should be invoked in immediate succession as if no other processing took place between the OSC Method invocations.

When an OSC Address Pattern is dispatched to multiple OSC Methods, the order in which the matching OSC Methods are invoked is unspecified. When an OSC Bundle contains multiple OSC Messages, the sets of OSC Methods corresponding to the OSC Messages must be invoked in the same order as the OSC Messages appear in the packet. (example)

When bundles contain other bundles, the OSC Time Tag of the enclosed

bundle must be greater than or equal to the OSC Time Tag of the enclosing bundle. The atomicity requirement for OSC Messages in the same OSC Bundle does not apply to OSC Bundles within an OSC Bundle.

A.4 Testen der Zeitgenauigkeit eines Computers

A.4.1 times.h

```
/*************************************************************/
/* times.h: time functions                                   */
/*            V 1.00 (juli 97)            by (iem - w.ritsch)*/
/*************************************************************/
#ifndef TIMES_H
#define TIMES_H

typedef struct systime /* universal time structure */
{
    int st_sec;
    int st_microsec;
} t_systime;

/* Function prototypes */
void sys_gettime(t_systime *tv);
int sys_microsecsince(t_systime *tv);
double sys_secsince(t_systime *tv);

#endif
```

A.4.2 times.c

```
/*************************************************************/
/* times.c:     time funtions and test (main)                */
/*            V 1.00 (juli 97)            by (iem - w.ritsch)*/
/*************************************************************/
/* OS-depend includes */
#ifdef UNIX
#include <unistd.h>
#include <sys/time.h>
#ifndef __linux__
#include <bstring.h>
#endif
#endif
#ifdef NT
#include <winsock.h>
#include <sys/types.h>
#include <sys/timeb.h>
#endif
```

```
/* common includes */
#include <string.h>
#include <stdio.h>
#include "times.h"


/* timebasis for unixs and nts */

void sys_gettime(t_systime *st)
{
#ifdef UNIX
    struct timeval now;

    gettimeofday(&now, 0);
    st->st_sec = now.tv_sec;
    st->st_microsec = now.tv_usec;
#endif
#ifdef NT
    struct _timeb now;
    _ftime(&now);
    st->st_sec = now.time;
    st->st_microsec = now.millitm * 1000;
#endif
}
int sys_microsecsince(t_systime *st)
{
    t_systime stnow;
    sys_gettime(&stnow);
    if (stnow.st_sec > st->st_sec + 1000) return (0x7fffffff);
    else return (1000000 * (stnow.st_sec - st->st_sec) +
     stnow.st_microsec - st->st_microsec);
}

double sys_secsince(t_systime *st)
{
    t_systime stnow;
    sys_gettime(&stnow);
    if (stnow.st_sec > st->st_sec + 1000000) return (0x7fffffff);
    else return (1. * (stnow.st_sec - st->st_sec) +
     (0.000001 * (stnow.st_microsec - st->st_microsec)));
}

#ifdef MAIN
/* --- Variables ------ */
```

```
int main()
{
  long i,j;
  long count;

  t_systime starttime;

  double diff,mindiff,maxdiff;
  double newtime,oldtime;

  double times[11];
  volatile int dummy = 1;

  puts("Testing timing in OS-Systems :");

  sys_gettime(&starttime);

  for(i=0l ; i<11l ;i++){

 times[i] = sys_secsince(&starttime);

 for(j=0l;j < i*i*i;j++)
dummy = dummy * dummy;
  }

  for(i=1l;i<11l;i++){
 printf("-> %ld.) %f sec (difference: %7.5f sec)(wait = %5ld flops)\n",
  i,times[i],times[i]-times[i-1l],i*i*i);
  }


  mindiff = 100.0;
  maxdiff = 0.0;
  count = 0l;

  sys_gettime(&starttime);
  oldtime = sys_secsince(&starttime);

  for(i=0; i<1000000l; i++){

 newtime = sys_secsince(&starttime);
 diff = newtime-oldtime;

 if(diff > maxdiff)
```

```
maxdiff=diff;


 if(diff < mindiff)
mindiff = diff;

 if(diff > 0.001)
count++;

 oldtime=newtime;
  }

  printf("Delta time tests between %ld time systemcalls:\n",i);
  printf("-> max delta time= %f sec\n",maxdiff);
  printf("-> min delta time= %f sec\n",mindiff);
  printf("-> %ld times from %ld more than 1 ms, that is %f %% \n",
count,i,(double) (100l*count)/(double) i);

  return 0;
}

#endif /* MAIN */
```