fpga\_synth  
von Dominic Strübi / Claudio Rutishauser / Marco Heinzen

Bild Synthesizer

Juni 2019, Et18b, DTP2, ZHAW

[1. Einleitung 3](#_Toc9948433)

[2. Systemübersicht 3](#_Toc9948434)

[3. Modulbeschreibungen 4](#_Toc9948435)

[Infrastructure 4](#_Toc9948436)

[MIDI 4](#_Toc9948437)

[Midi Input 4](#_Toc9948438)

[MIDI UART 4](#_Toc9948439)

[MIDI Controller 5](#_Toc9948440)

[Tone Generator 6](#_Toc9948441)

[DDS (Polyphonie) 6](#_Toc9948442)

[I2C 7](#_Toc9948443)

[I2S Master 7](#_Toc9948444)

[4. Funktionstests (kurz) 8](#_Toc9948445)

[5. Fazit 8](#_Toc9948446)

[6. Quellenverzeichnis 8](#_Toc9948447)

[7. Appendix 8](#_Toc9948448)

# Einleitung

„fpga\_synth“ ist das DTP2-Projekt von Rutishauser Claudio, Heinzen Marco und Strübi Dominic.   
  
Dies ist die dazu gehörende Technische Dokumentation welche den Aufbau und die Funktionsweise unseres Synthesizers mit Midi Steuerung beschreibt. Der Synthesizer wurde auf dem *FPGA* (Field Programmable Gate Array) des Entwicklungsboards *Altera DE2-115 (Cyclone IV Familie)* mit VHDL-Code realisiert. Dazu wurden modulare Code-Blöcke konfiguriert (Emacs, Notepad++ mit VHDL Maske) welche mithilfe von Modelsim simuliert getestet wurden und mit *Quartus Prime kompiliert* und auf das Entwicklungsboard geladen wurde. Des Weiteren wurde die Software *ModelSim* genutzt, um den Code zu simulieren und zu testen.

Es folg zuerst ein Überblick über das gesamte System, danach werden die einzelnen Module und deren Funktionsweise und Interaktion mit anderen Modulen beschrieben. Abschliessend wird in einem kurzen Fazit auf die Funktionalität des fpga\_synths.

# Systemübersicht

In ABBILDUNG 1 ist eine Übersicht der verwendeten Hardwarekomponenten zu sehen. Über den Anschluss *GPIO 26* des FPGA werden die seriellen Daten des *MIDI-*Keyboards empfangen (Tonhöhe 0-127, Velocity 0-127, Midi Channel und weitere) Die 50MHz-Clock des Entwicklungsboards (*Clock\_50*) , Schalter (*Switches*) und Taster (*Keys*) werden zur Bedienung Die Verarbeitung der Eingangsdaten, die Synthese zu einem polyphonen Klang sowie dessen Ausgabe findet auf dem FPGA Board statt. Der *Audio Codec* auf dem Entwicklungsboard wandelt das digitale Ausgangssignal des *FPGA* in ein analoges Ausgangssignal (DAC) Digitl to Analog Converter), welches schlussendlich über einen Lautsprecher ausgegeben werden kann.

Abbildung 1: Grundsätzliche Funktionsweise, Hardware Komponenten und Blockdiagramm des Codes auf dem FPGA

(Es steht ein LCD-Display zur Verfügung, welches einen wichtigen Teil des *User Interface* bildet. Hier wird das Menü mit Einstellungsmöglichkeiten angezeigt. Schlussendlich dienen die roten LEDs (*LEDR*) zur Zustandsüberwachung. So können zum Beispiel verschiedene Parameter visualisiert werden. )

Abbildung 1 zeigt auch eine Übersicht aller konfigurierten Komponenten/Module und deren Funktionsweise auf (*Input, Tonerzeugung, Output).* Im Infrastructure Block werden die (analogen), externen Signale (wie serieller Midi-Input (GPIO\_26), Switches und Keys) debounced, d.h. es wird aus einem eventuell unsauberen Eingangssignal ein klares HIGH oder LOW für die Taktperiode erzeugt und somit indirekt mit der internen 50 MHz Clock, bzw direkt mit der im Modulo\_Divider Block erzeugten, internen 12.5Mhz synchronisiert. Die 12.5 MHz Clock wird in allen Blöcken verwendet (single clock domain). Desweiteren zählt auch der MIDI Input (UART Receiver), welcher Daten vom Keyboard empfängt und diese zum weiteren Gebrauch aufbereitet.

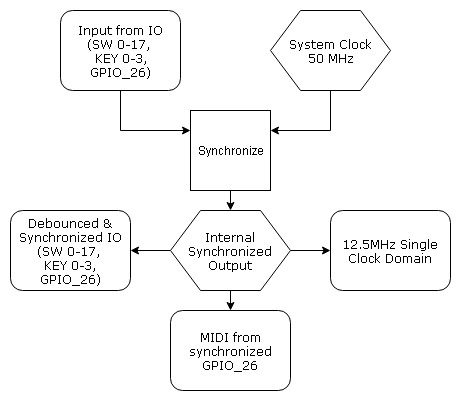
Im Tonerzeugungsblock sind zehn *DDS* enthalten (extra Feature), welche jeweils einen einzelnen Ton erzeugen. Es können also maximal zehn Töne gleichzeitig erzeugt werden. Am Ausgang der Tonerzeugung entsteht dann durch Überlagerung aller zehn Einzeltöne in ein paralleles Signal.

Ausgangsseitig befindet sich der I2C Block welcher durch Preset-Register (Mute, Mute-Left, Mute Right) den Audio Codec ansteuert. Der Block *I2S Master*, welcher die parallelen Daten der Tonerzeugung in serielle Daten wandelt und diese an den *Audio-Codec* sendet.

# Modulbeschreibungen

Die Blöcke folgen dem input – output Signalverlauf (wenn möglich)

## Infrastructure



Der Block *Infrastructure* nimmt alle relevanten Eingangssignale vom Entwicklungsboard auf (Schalter-Signale, Taster-Signale, *MIDI*-Datensignal, 50 MHz Clock) und synchronisiert diese mit dem intern generierten 12.5 MHz-Clock*-*Signal indem diese durch je einmit 12.5MHz getaktetes Delay FlipFlop D-FF geschlauft werden. Als Output hat der Infrastructure Block die 12.5MHz-Clock(50MHz-Clock geteilt durch vier) und alle synchronisierten Signale. Der 12.5MHz-Clock wird von allen Systemkomponenten verwendet (single clock domain). Die Synchronisierung erfolgt mit zwei in Serie geschalteten Flipflops pro Signal. Es werden zwei Flipflops verwendet, damit das synchronisierte Signal keine unerwünschten Prellungen aufweist (debouncing, Vermeidung von Jitter Effekten).

## MIDI

**Musical Instrument Digital Interface**, (engl. „digitale Schnittstelle für Musikinstrumente“), kurz: **MIDI** ist ein Industriestandard für den Austausch musikalischer Steuerinformationen zwischen elektronischen Instrumenten, wie z. B. Keyboards oder Synthesizern. Dieser Standard umfasst sowohl die genaue Beschaffenheit der erforderlichen Hardware als auch das Kommunikationsprotokoll für die zu übermittelnden Daten. (midi.org)

## Midi Input

Das Modul *MIDI Input* sorgt dafür, die synchronisierten (GPIO\_26), seriellen *MIDI*-Befehle zu empfangen und mit diesen den Ton-Array am Output anzusteuern. Der Block besteht aus zwei Unterblöcken: *MIDI UART* und *MIDI Controller*.

MIDI UART

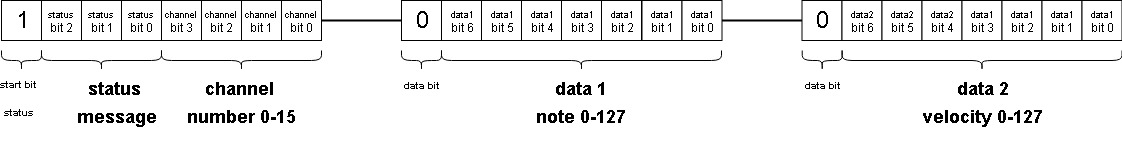
Das asynchrone, serielle *MIDI*-Datenprotokoll basiert auf dem *UART*-Protokoll. Die Länge der Datenpakete ist immer 1 Byte (= word). Es werden pro Message immer 3 Datenpakete versendet bzw empfangen. Das erste Bit der 3 Pakete ist das Start-Bit, gefolgt von 3 Status Bits (Note on, Note Off, Control Change, Program Change etc), danach folgt eine 4-bit lange Adresse für die Channel Nummer (0-15 = 16 Channels), wobei bei diesem Projekt nur der Channel 0 verwendet wurde. Danach folgen 2 Daten-words welche immer mit 0 im ersten Bit beginnen. In Abbildung 3 ist ein solches Paket visualisiert. Es werden teils auch nur die beiden Datenbytes versandt, wenn die Interpretation der Daten nicht ändert und das Statusbyte somit überflüssig ist.

Abbildung 3: Visualisierung eines MIDI-Datenpaketes

Dieser *MIDI*-Synthesizer unterstützt nur die folgenden Befehle:

|  |  |  |  |
| --- | --- | --- | --- |
| **Befehl** | **Status-Bits** | **Datenbyte 1** | **Datenbyte 2** |
| *Note On* | 000 | Tonhöhe | Anschlagsgeschwindigkeit |
| *Note Off* | 001 | Tonhöhe | Entlastungsgeschwindigkeit |

Der erste Unterblock von *MIDI Input* ist *MIDI UART*, welcher als Empfänger dient, der die seriellen Input Daten in Parallele Daten umwandelt. Er empfängt die seriellen *MIDI*-Bytes und wandelt sie zu acht parallelen Bits am Ausgang um. Dazu wird vom Block noch ein *valid*-Bit ausgegeben, alle acht Datenbits vollständig empfangen wurden und somit gültig sind.

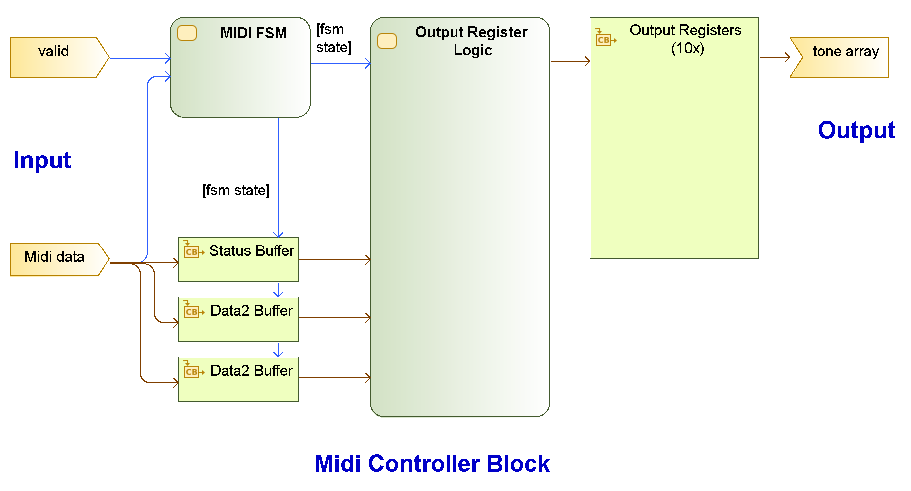
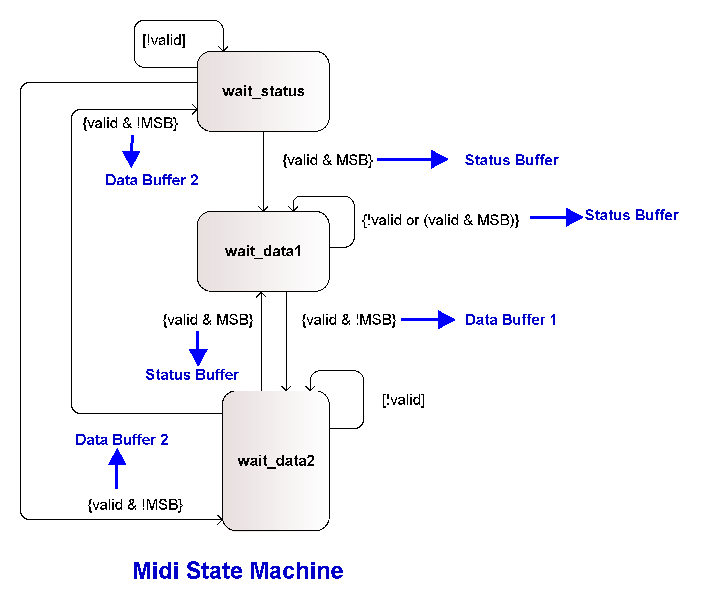
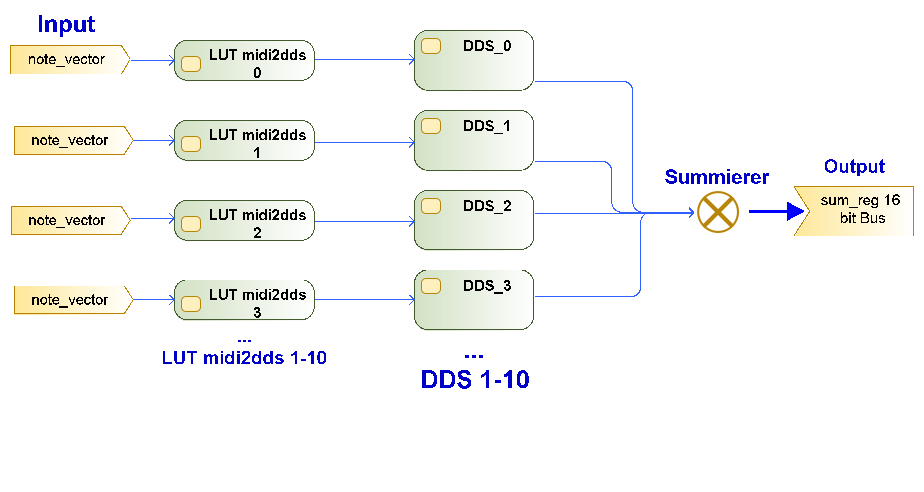
MIDI Controller   


Abbildung 4: Blockschaltbild des MIDI Controller

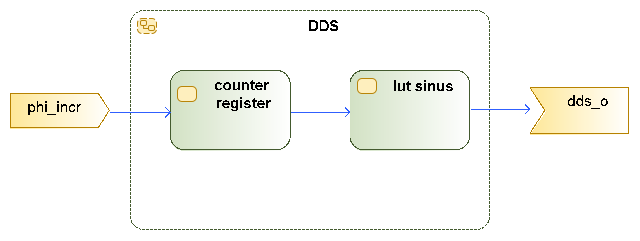
Der zweite Unterblock *MIDI Controller* übernimmt die umgewandelten, parallelisierten Midi Daten sowie das valid-Bit des *Midi UART* Blocks (siehe Blockschaltbild, Abbildung 4).

Die Steuerung der Register und Buffer wird mit einer FSM (Finite State Machine, Abbildung 5) realisiert. Abhängig vom aktuellen Zustand der *State Machine* (*MSB* des empfangenen Bytes (Unterscheidung 1 = Signal- / 0 = Datenbyte) wird der richtige Buffer beschrieben und in den nächsten Zustand gewechselt. Dies geschieht immer zeitgleich, wenn das valid-Bit *HIGH* ist. Ist im Zustand *wait\_data2* das *valid*-Bit auf *HIGH*, dann wurde komplettes Datenpaket empfangen und in den Buffer gelesen. Dadurch wird der Buffer-Inhalt gesteuert von der *Output Register Logic* in ein Output-Register geschrieben. Das Register wird gemäss den Status-Bits des Statusbytes ausgewählt, welche den jeweiligen Midi Befehlen entsprechen. Liegt der Befehl *Note On* vor, so wird eines der zehn Register des Ton-Arrays mit den entsprechenden Werten beschrieben, oder überschrieben, falls alle anderen Register schon besetzt sind. Beim Befehl *Note Off* wird nach einem Registereintrag gesucht, der die entsprechende Tonhöhe enthält, welcher auf null gesetzt wird.

## Tone Generator

Mit dem *Tone Generator* wird ein einzelner Ton erzeugt. Die Tonhöhe wird durch die Note bestimmt, welche in Form eines sieben-Bit-Signals am Eingang steht (Midi: 1. Datenbyte der Midi Message). Es können 128 verschiedene Tönhöhen / Noten gespielt werden (0-127). Im Unterblock *midi2dds* wird der zur gespielten Note entsprechende *phi\_incr* Wert, welcher der Frequenz der Note entspricht im *dds* Block mit Hilfe einer Look Up Tabelle (*LUT*) den passenden 16-Bit Wert aus dem tone\_gen\_pkg herauszulesen und mittels des Signals dds\_o bereitzustellen. dds\_o verändert sich mit jeder Clock Periode und es entsteht eine Sinuswelle mit der entsprechenden Frequenz (z.B. A3 = 440 Hz). Dabei entsteht eine Sinuswelle über der Zeit (f(t) = sin(w\*t)).

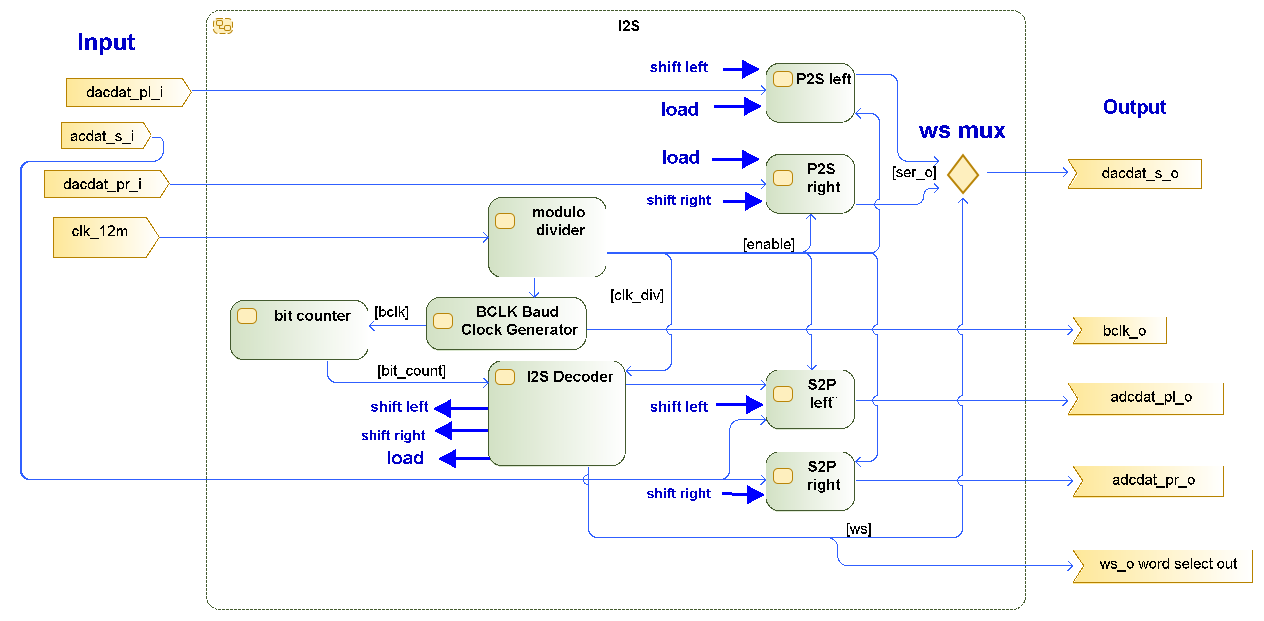
### DDS (Polyphonie)

Im *DDS*-Block wird aus der Look Up Table die entsprechende Sinuswelle in Form von diskreten 13-Bit-breiten Werten im Bereich -4096 bis +4095 in einem Array abgespeichert. Mit dem Block *counter register* wird der korrespondierende Index erzeugt, mit welchem in der LUT der Wert mit Hilfe von phi\_incr herausgelesen wird. Im *counter\_register* Block wird jeweils mit einem 48kHz-Clock-Zyklus um *phi\_incr* hochgezählt. Je grösser also *phi\_incr*, desto schneller wird das Array durchlaufen und desto höher ist die Frequenz der erzeugten Sinuswelle. Der ausgelesene Wert wird zum Schluss am Ausgang *dds\_o* ausgegeben.

## I2C

Der *I2C*-Block steuert den Audio Codec mittels I2C Protokoll und besteht aus zwei Unterblöcken, *Codec Controller* und *I2C-Master*. *Codec Controller* wählt aus einer Auswahl von vordefinierten Presets durch einen Array ein vordefiniertes Datenpaket aus und stellt dieses dem *I2C-Master* zur Verfügung. Dabei kann mit den Switches 0-3 am Board ausgewählt werden, ob der linke, der rechte oder beide analogen Audio Ausgangskanäle auf stumm geschaltet werden. Der *I2C-Master* wiederum schickt das Paket an den *Audio-Codec*.

## I2S Master



Der *I2S*-Block besteht sowohl aus einer parallel-seriell- als auch einer seriell-parallel-Schnittstelle und wird zur Audioübertragung verwendet. Der aktuelle Wert von *dacdat\_pl\_i, dacdat\_pr\_i* welcher parallel am Eingang anliegt, wird seriell an den *Audio-Codec* geschickt wobei der Eingang *dacdat\_pl\_i* der Amplitude des linken Audiokanals entspricht und *dacdat\_pr\_i* dem rechten. Die parallelen Daten werden abwechselnd gesendet (zuerst links, dann rechts). Durch den *bit\_counter*-Block, welcher einen 7-Bit-*counter* (0-127) auf den *i2s decoder* führt, steuert damit indirekt den links-rechts Kanalwechsel. Der *i2s decoder* erzeugt aus dem bit\_counter die Signale *load*, *shift\_l*, *shift\_r*, und ws (*word select*). Das Signal *ws* signalisiert, ob das linke oder rechte Schieberegister aktiviert wird. Das Signal *load*, wird jeweils auf *HIGH* gesetzt, wenn der *counter* auf 0 zurückgesetzt wird. Im Zeitraum 0 und 63 des /-bit-counters wird das Signal *shift\_l* auf HIGH gesetzt und somit mit dem linken Schieberegister geschoben, analog wird im Zeitraum 64-127 das rechte Schieberegister mit shift\_r aktiviert. Die Übertragungsrate für die Ansteuerung des Audio Codec wird im Block mit einem *modulo divder* erzeugt und entspricht der halben 12.5MHz-Frequenz. Die erzeugte Baudrate (bclk) wird als *enable* Signal auf die verschiedenen Schieberegister geführt. Es wird lediglich dann geschoben, wenn *enable* *HIGH* ist.

Die seriellen Daten am Ausgang *dacdat\_s\_o* werden schliesslich auf den *Audio-Codec* geführt.

Die seriell-parallel-Schnittstelle wurde nur zu Testzwecken implementiert um einen Feedback-Loop zu erzeugen und den Block zu testen. Mit der SP-Schnittstelle können z.B. analoge Audio Eingangsdaten seriell empfangen und intern parallel gewandelt werden.

# Funktionstests (kurz)

Bilder wenn nötig

Bevor neu entwickelte Elemente in das Hauptdesign übernommen wurden, wurden diese mittels *ModelSim* und auf der Hardware getestet. In Abhängigkeit zu den erstellten Elementen wurden hierbei sowohl *Testbenches* für die gesamte Schaltung als auch für einzelne Komponenten verwendet.

Nach erfolgreicher Absolvierung der Simulation wurde das Design auf die Hardware übertragen und erneut getestet. In diesem zweiten Testschritt konnten Fehler wie z.B. ein nicht korrekt angeschlossene 12.5MHz erkannt und im weiteren Verlauf behoben werden.

In den Simulationen wurden Datenübertragungen (*I2S, I2C, Midi*) mittels automatisierten Tests überprüft. Die geprüften Schaltungen wurden auf die Hardware übertragen und einem Hörtest unterzogen.

# Fazit

Erweitern , future possibilities, schlusswort/fazit

Der fpga\_synth Midi Synthesizer funktioniert einwandfrei. Ein kleines Detail ist das Knacksen beim spielen der eher tieferen Noten, da das Signal nicht genau bei den Nulldurchgängen der Sinuskurve geschnitten wird (note off)

Es gibt nun zahlreiche Möglichkeiten, den Synthesizer zu erweitern. Ob FM-Synthese oder andere Syntheseformen als verschiedene Modi bis zu externen Controllern (z.B. via Midi Through des Keyboards). Neben einer GUI mittels LCD Display wären sicher Envelopes und Filter (LFO) sowie Delay und Reverb von Interesse.

Der fpga\_synth *MIDI*-Synthesizer kann mit jedem Midi-fähigen Gerät verwendet werden (z.B. Sequenzer), das folgende grundlegende *MIDI*-Befehle versenden kann: *Note Off*, *Note On*.Die Verarbeitung weiterer Befehle ist nicht implementiert, wäre aber wünschenswert.

# Quellenverzeichnis

Alle Illustrationen und Layout: Marco Heinzen © 2019  
Allgemeine Beschreibung & Spezifikation MIDI von midi.org

# Appendix

Projektevaluation, ¾ seiten -> olat

Zeitplan, Projektmanagement nur im file ordner  
Grosse pläne und bilder