Entwicklung und Fertigung eines MIDI-Interface für DPM-48 Drum Machine

Christian Feyer
916883

PROJEKTBERICHT

eingereicht am
Fachhochschul-Bachelorstudiengang Elektrotechnik

für das Modul BI119

in Kiel

im Januar 2021

Inhaltsverzeichnis

Kurzfassung

Der vorliegende Bericht beschreibt die Planung und Durchführung eines Projektes im Modul BI119 im Wintersemester 2020/2021 an der Fachhochschule Kiel. Für die 1984 vom japanischen Hersteller Sakata entwickelte und in Deutschland von Hammond unter ihrer Eigenmarke vertriebenen Drum-Machine DPM-48 („Digital Percussion Machine“) sollte ein Interface entwickelt werden, welches die Ansteuerung der im Gerät auf EPROM Festspeichern gespeicherten digitalen Schlagzeug-Klänge über jedes MIDI-fähige Gerät ermöglicht. Der Hintergrund für die Entwicklung ist die Limitation des Gerätes, nur mit eingesteckter (optionaler) RAM-Kassette Sequenzen erstellen und abspielen zu können. Eine weitere Möglichkeit zur Ansteuerung der Klänge bietet neben den Tasten auf dem Bedienfeld ein sogenanntes Trigger-Interface. Dieses bot den Anschluss eines optional erhältlichen Interfaces zur Verwendung von Drum-Pads in Ergänzung mit einem richtigen Schlagzeug. Zusätzlich sollte eine Schaltung entwickelt werden, die eine Erweiterung der werksseitig vorhandenen Klänge ermöglicht. Die Auswahl dieser erweiterten Klänge sollte ebenfalls über MIDI erfolgen.

Kapitel 1

Technische Details und Überblick über die DPM-48 Drum-Machine

Eine Drum-Machine oder auch Drum-Computer ist ein elektronisches Musikinstrument, welches die Erstellung und Wiedergabe einer Folge von Schlagzeug- und Percussion-Klängen mit vorgegebenem Tempo und Rhythmus ermöglicht. Ein menschlicher Schlagzeuger und die aufwendige Aufnahme des Schlagzeuges mit Mikrofonen ist dann nicht mehr nötig. Die Erzeugung der Klänge kann durch analoge Synthese oder durch Wiedergabe von Samples erfolgen. Die DPM-48 nutzt in EPROM (erasable programmable read-only memory) gespeicherte Samples. Je nach Gruppe sind mehrere Klänge pro EPROM gespeichert.

Die DPM-48 wurde 1984 vom japanischen Hersteller Sakata auf den Markt gebracht und in Deutschland von Hammond vertrieben. Das Gerät besitzt 22 verschiedene Schlagzeug- und Percussion-Klänge. Die Klänge sind auf acht Gruppen aufgeteilt (siehe Tab. 1.1). Pro Gruppe kann nur ein Klang gleichzeitig angesteuert werden (achtstimmige Polyphonie), da sich alle Klänge einer Gruppe ein EPROM sowie einen zugehörigen Binärzähler teilen.

Die Drum-Machine hat einen eingebauten achtkanaligen Mixer, über die sich eine

Tabelle 1.1: Drum-Gruppen der DPM-48.

Gruppe	Pad 1	Pad 2	Pad 3
1 - Toms	TOM 1	TOM 2	-
2 - Toms	TOM 3	TOM 4	-
3 - Bassdrum	BASS 1	BASS 2	BASS 3
4 - Hi-Hat	CLOSED	ACCENT	OPEN
5 - Snaredrum	SNARE 1	SNARE 2	SNARE 3
6 - Cymbal	RIDE 1	RIDE 2	CRASH
7 - Percussion 1	CABASA 1	CABASA 2	CLAP
8 - Percussion 2	AGOGO 1	AGOGO 2	RIM

Abbildung 1.1: DPM-48 Digital Percussion Machine.

individuelle Mischung der Gruppen für den Mono und den Stereo-Ausgang erstellen lässt. Der Stereo-Ausgang hat ein fest voreingestelltes Panorama für jede Gruppe. Zudem verfügt jede Gruppe über einen Einzelausgang um das Audiosignal separat weiterverarbeiten zu können. Der Sequenzer ist nur nutzbar wenn die optional erhältliche RAM-Cassette an der Gerätefront eingesteckt ist. Diese speichert bis zu 48 Drum-Pattern mit jeweils 32 Schritten. An der Rückseite sind zwei Sync-Buchsen zur Takt-Synchronisierung mit anderem Equipment vorhanden. Diese nutzen den von Roland eingeführten DIN-Sync-Standard von 24ppq (pulses per quarter-note). Zudem findet sich dort eine Buchse für den Anschluss eines Interfaces, welches die Verwendung von elektronischen Drum-Pads zur manuellen Ansteuerung der einzelnen Klänge mittels Drum-Sticks ermöglicht.

Kapitel 2

Anforderungen an das MIDI-Interface

Das zu entwickelnde MIDI-Interface soll die rückseitig angebrachte Trigger-Buchse ersetzen. An deren Stelle soll eine Frontplatte mit dem MIDI-Anschluss und einem User-Interface, bestehend aus einer LED und einem Taster, auf einer UI-Platine befestigt angebracht werden. Diese wird über eine mehradrige Leitung mit der MIDI-Interface Platine verbunden. Die MIDI-Interface Platine, welche den Mikrocontroller und die Peripherie enthält, soll im Gerät auf die Mikroprozessor-Platine der DPM-48 anstelle der mehradrigen Leitung der ausgebauten Trigger-Buchse aufgesteckt werden. Der MIDI-Input ist aus Platzgründen als 3,5 mm TRS-Miniklinke ausgeführt und soll den MIDI-TRS Type A Standard nutzen. Das Interface soll die acht Trigger-Inputs ersetzen und über MIDI-Messages von MIDI-kompatiblen Geräten alle 22 Klänge ansteuern können. Der MIDI-Kanal soll über den Taster erlernbar sein. Zudem soll das Interface die MIDI-gesteuerte Auswahl von erweiterten Sound-Bänken in ausgewählten Gruppen ermöglichen. Das Interface soll eingebaut werden können ohne Änderungen oder Beschädigungen am Gerät zu verursachen.

Kapitel 3

Beschreibung und Entwicklung

Das Blockschaltbild (siehe Abb. 3.1) ist in verschiedene Funktionsblöcke unterteilt. Beginnend von links ist der MIDI-Input abgebildet, dieser ist über einen Optokoppler galvanisch vom Rx-Input (Receive) des UARTs im Mikrocontroller getrennt. Die Firmware im Mikrocontroller wertet die empfangenen MIDI-Messages auf dem eingestellten MIDI-Kanal aus und wählt darauf hin aus der Matrix eine Kombination von drei Pad-Select- und acht Trigger-Leitungen aus um den entsprechenden Klang anzusteuern. Diese Kombination wird durch einen 3-zu-1 Multiplexer, dessen Ausgang auf den Eingang eines 1-zu-8 Demultiplexers gelegt ist, bewerkstelligt. Bei den Pad-Select Leitungen handelt es sich um die Signale, mit der auch die Taster-Matrix des Bedienfelds abgefragt wird. Zusätzlich werden je nach MIDI-Note-Number zusätzliche Adress-Select Leitungen (unten im Bild) geschaltet um ein bis drei zusätzliche Speicherbereiche in ausgewählten EPROMs anzusprechen. Die Pad-Select- und Trigger-Leitungen werden zusammen mit der Spannungsversorgung auf einer zwölf-poligen Stiftleiste (rechts im Bild) von der DPM-48 bereitgestellt. Nicht dargestellt ist ein Taster und eine LED, die als User-Interface dienen und den MIDI-Kanal Lernmodus auswählen und das Vorhandensein eines MIDI-Signals anzeigen.

3.0.1 PCB- und Frontplattendesign

Die Schaltpläne (siehe Seite 25) und PCB Layouts für die Leiterplatten und die Frontplatte wurden mit der freien Software KiCad (https://kicad-pcb.org) erstellt. Es wurden zwei Kupferlagen verwendet. Die Dicke der Platinen beträgt 1,6 mm. Bis auf den Bedientaster, die LED, die Buchsen und Sockel wurden nur SMD Bauteile genutzt um eine hohe Packungsdichte zu erreichen und wenig Berührungspunkte auf der Unterseite der Platine zu haben, da diese dicht auf die vorhandene Mikroprozessor-Platine der DPM-48 aufgesteckt werden sollte. Für einige Bauteile mussten zunächst die Footprints erstellt und zur Ansicht die passenden 3D-Modelle besorgt werden. Die Größe der Interface-Platine beträgt 70 x 80 mm. Alle SMD Bauteile werden auf der Top-Lage (siehe Abb. 3.2 und 3.3) platziert. Die Platine wird über die gewinkelte Buchsenleiste J5 mit der Hauptplatine der DPM-48 verbunden. Der Anschluss der MIDI/UI-Platine erfolgt über die Stiftleiste J2. Zwei weitere Stiftleisten J6 und J7 sind zum Anschluss von EPROM-Adapterplatinen gedacht. Eine zweireihige Stifleiste dient der Verbindung mit einem In-System Programmiergerät zur Programmierung des Mikrocontrollers. Bei-

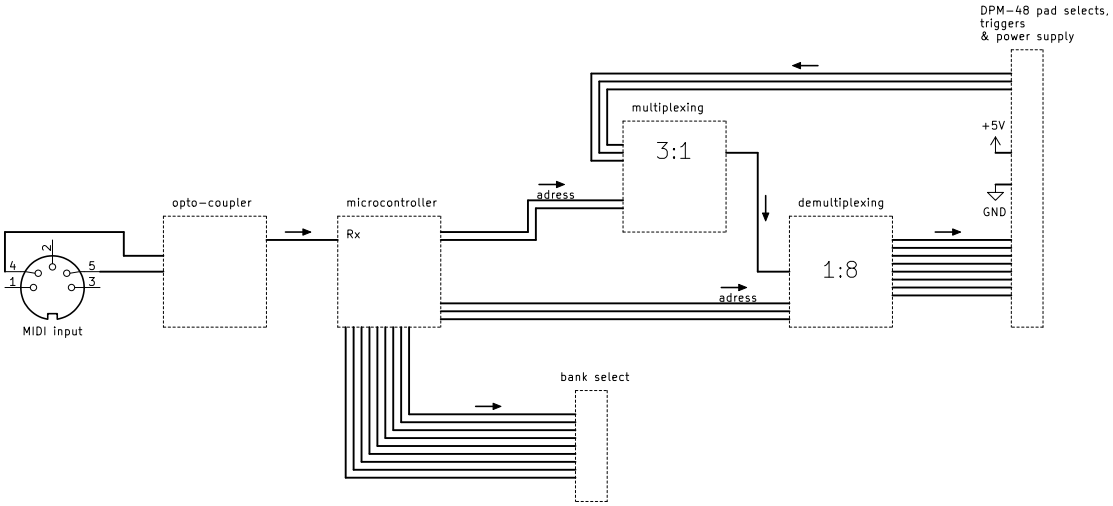


Abbildung 3.1: Blockschaltbild des MIDI-Interfaces.

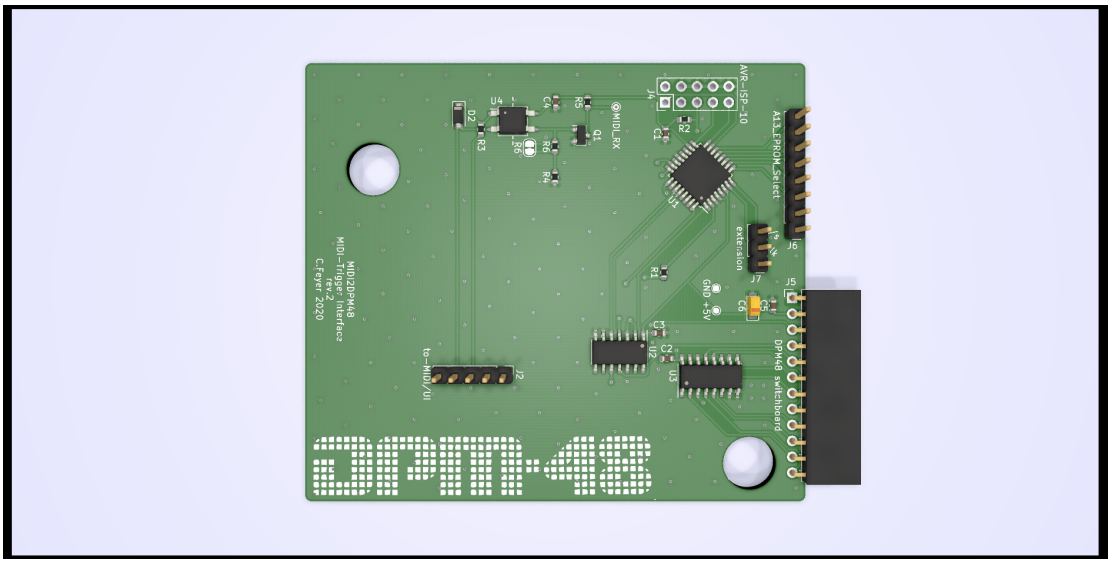


Abbildung 3.2: Draufsicht der Interface-Platine.

de Lagen sind mit einer Massefläche ausgefüllt. Zwei große Aussparungen lassen Platz für die Schrauben der DPM-48 Mikroprozessor-Platine.

Die Frontplatte (siehe Abb. 3.4) wurde ebenfalls mit KiCad erstellt, die Maße betragen 16 x 42 mm. Die verschiedenen Bohrungen wurden zunächst als Footprints erstellt und in einer Bauteilebibliothek gespeichert. Auch bei der Frontplatte sind beide Kupferlagen mit einer Massefläche ausgefüllt. Für den „MIDI“ Schriftzug wurde das Logo der DPM-48 verändert. Die zugehörige Platine (siehe Abb. 3.5) misst 25 x 25 mm und ist einseitig kupferkaschiert. Sie trägt den Taster, die MIDI-TRS Buchse und die LED. Als Zugentlastung für die fünfadrige Verbindungsleitung zur Interface-Platine sind fünf Bohrungen vorgesehen durch die die Leitungen vor dem Anlöten gefädelt werden.

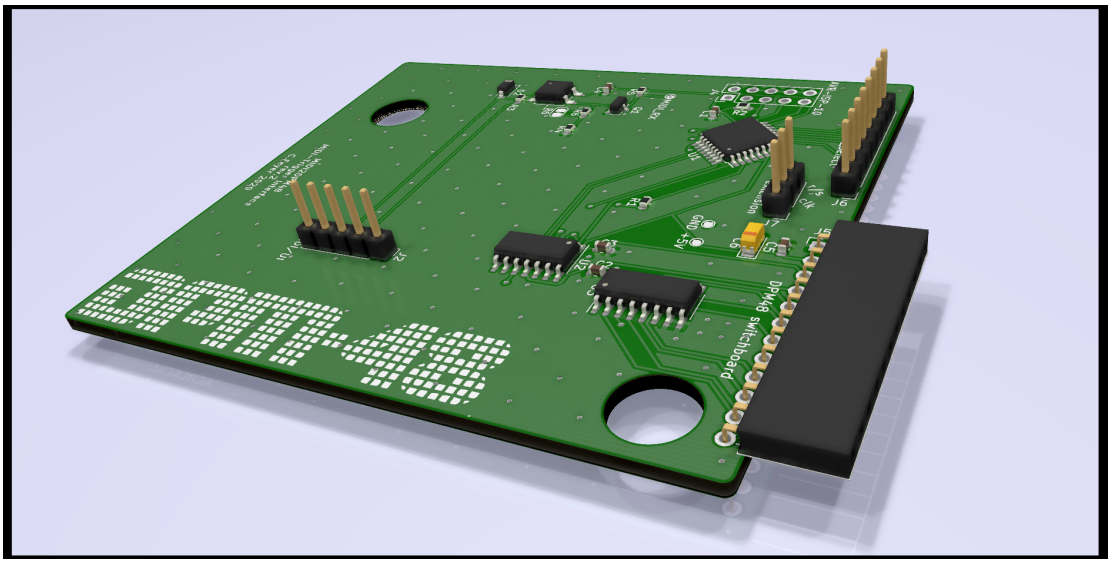


Abbildung 3.3: Schrägansicht der Interface-Platine.

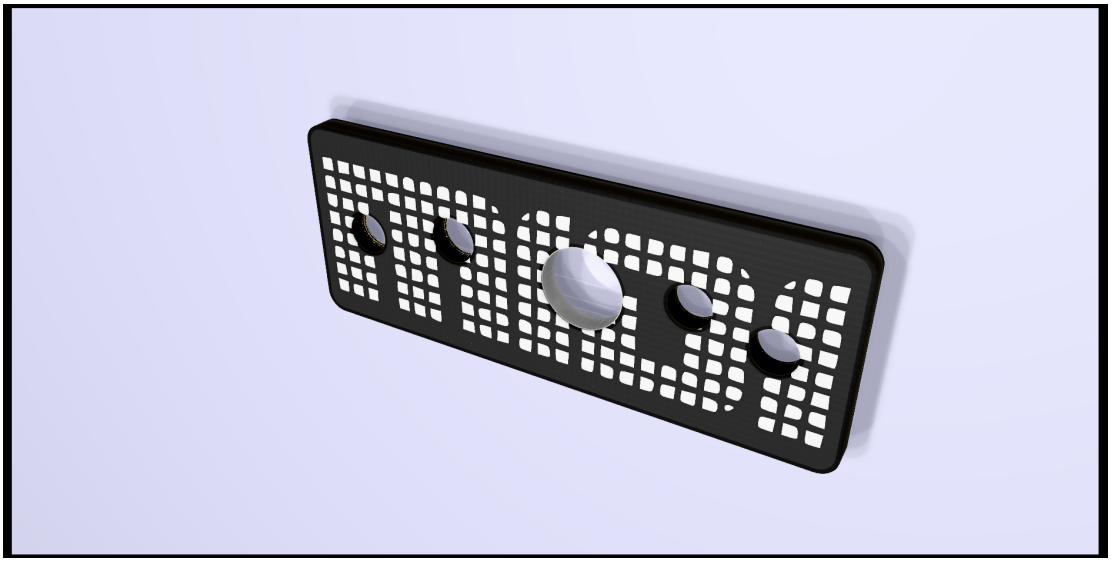


Abbildung 3.4: Ansicht der Frontplatte.

Die EPROM-Adapterplatinen ermöglichen die Verwendung von EPROMs mit größerem Speicherplatz in den vorhandenen Sockeln in der Drum-Machine. Es werden bis zu drei weitere Sound-Bänke pro Gruppe unterstützt. Dies entspricht dem vierfachen Speicherplatz des ursprünglich verwendeten EPROM. Auf der Unterseite der Platine (siehe Abb. 3.7) befinden sich zwei Reihen dünner Stiftleisten die anstelle des EPROM in den vorhandenen Sockel gesteckt werden. Auf der Top-Lage (siehe Abb. 3.6) befindet sich ein Sockel in den ein EPROM mit zwei- oder vierfachem Speicherplatz gesteckt wird. Über die A13 und A14 benannten Stifte werden die Speicherbänke von der Interface-Platine

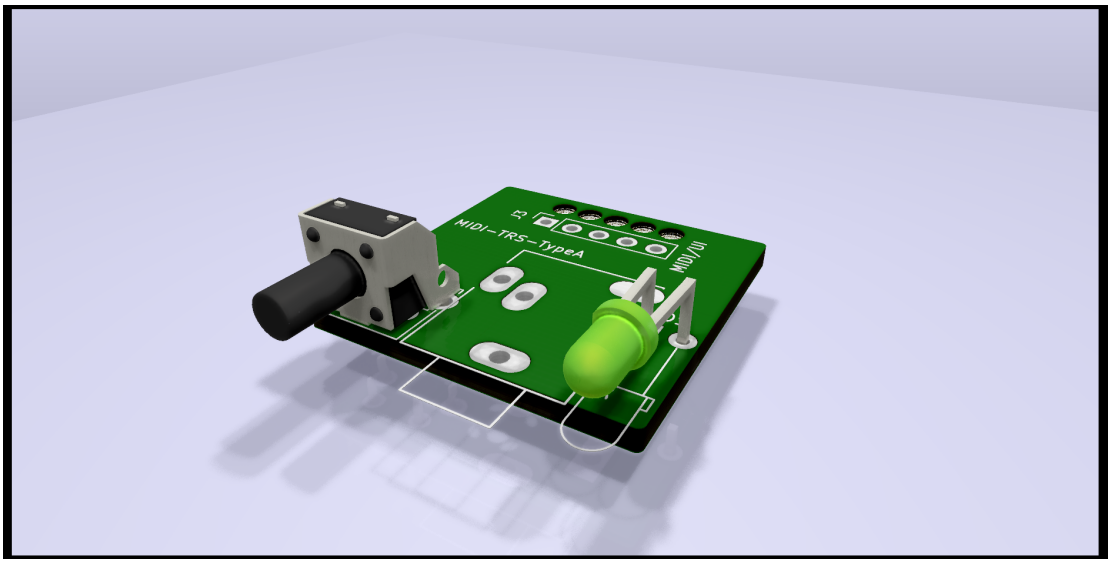


Abbildung 3.5: Ansicht der MIDI/UI-Platine.

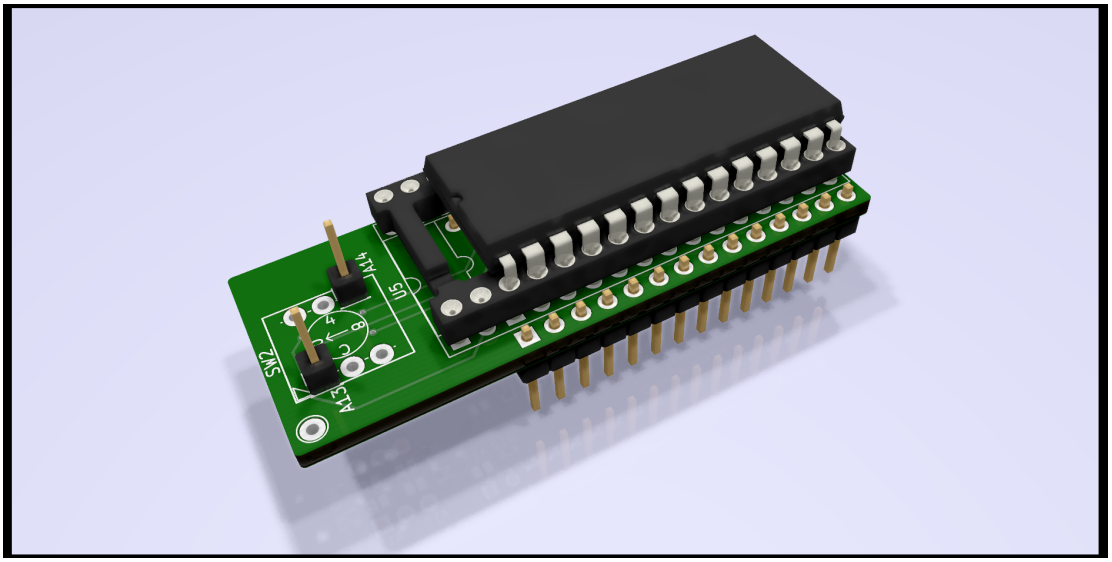


Abbildung 3.6: Ansicht eines EPROM-Adapters.

ausgewählt.

3.0.2 PCB-Herstellung

Die erzeugten Platinendaten wurden zur Fertigung an einen Leiterplatten-Hersteller geschickt. Zusätzlich zu den Platinen wurde dort ein SMD-Stencil (Metallschablone für die Lotpaste) bestellt. Für die Platinen wurde doppelseitig kupferkaschiertes Platinenmaterial FR4 mit einer Stärke von 1,6 mm, farbiger Lötstopplack (grün bzw. schwarz) und weißer Siebdruck ausgewählt. Alle Bohrungen und Fräsungen werden während des

Fertigungsprozesses im Werk gemacht. Die offenliegenden Lötstellen werden vorverzinnt.

3.0.3 Bauteilebestellung

Aus KiCad wurde eine Stückliste (BOM – Bill of materials, siehe Seite 9) ausgehend vom Schaltplan exportiert. Diese wurde bearbeitet und Anhand der verschiedenen Kenngrößen wie beispielsweise Gehäuseform (Spalte „Footprint“) und Wert oder Typ des Bauelementes (Spalte „Comment“) wurde bei einem Lieferanten ein Warenkorb angelegt. Je nach benötigter Anzahl und Staffelung der Preise wurden ganze Gurtabschnitte eingekauft.

MIDI2DPM48_rev2

Comment	Designator	Footprint	Reichelt
MIDI2DPM48 mainboard			
100n	C1,C2,C3,C4,C5	Capacitor_SMD:C_0603_1608Metric	X7R-G0603 100N
CP	C6	Capacitor_Tantalum_SMD:CP_EIA-3216-18_Kemet-A	T491A 10U 16
1N4148W	D2	Diode_SMD:D_SOD-123	1N 4148W DIO
MMBT3904	Q1	Package_TO_SOT_SMD:SOT-23	MMBT 3904LT1G
220	R1,R3	Resistor_SMD:R_0603_1608Metric	SMD-0603 220
10k	R2,R5	Resistor_SMD:R_0603_1608Metric	SMD-0603 10k
470	R4,R6	Resistor_SMD:R_0603_1608Metric	SMD-0603 470
ATMEGA88PA-AU	U1	Package_QFP:TQFP-32_7x7mm_P0.8mm	ATMEGA 88PA-AU
4051	U3	Package_SO:SOIC-16_3.9x9.9mm_P1.27mm	SMD 4051
EL357N-G	U4	SOME:EL357N	EL 357N-G
to-MIDI/UI	J2	Connector_PinHeader_2.54mm:PinHeader_1x05_P2.54mm_Vertical	PS 25/5G WS (Stecker)
AVR-ISP-10	J4	Connector_PinHeader_2.54mm:PinHeader_2x05_P2.54mm_Vertical	MPE 087-2-010
DPM48 switchboard	J5	Aschenbach:PinSocket_1x12_P2.54mm_Horizontal_reversed	FIS BL3.36Z
4016	U2	Package_SO:SOIC-14_3.9x8.7mm_P1.27mm	SMD 4016
MountingHole	H1,H2	MountingHole:MountingHole_8.4mm_M8	-
A13_EPROM_Select	J6	Connector_PinHeader_2.54mm:PinHeader_1x08_P2.54mm_Vertical	-
DIN-Sync	J7	Connector_PinHeader_2.54mm:PinHeader_1x03_P2.54mm_Vertical	-
R6	JP1	Jumper:SolderJumper-2_P1.3mm_Bridged_RoundedPad1.0x1.5mm	-
A17	JP2	Jumper:SolderJumper-2_P1.3mm_Open_Pad1.0x1.5mm	-
A18	JP3	Jumper:SolderJumper-2_P1.3mm_Open_Pad1.0x1.5mm	-
A13	JP4	Jumper:SolderJumper-2_P1.3mm_Bridged_Pad1.0x1.5mm	-
A16	JP5	Jumper:SolderJumper-2_P1.3mm_Bridged_Pad1.0x1.5mm	-
A15	JP6	Jumper:SolderJumper-2_P1.3mm_Bridged_Pad1.0x1.5mm	-
A14	JP7	Jumper:SolderJumper-2_P1.3mm_Bridged_Pad1.0x1.5mm	-
MIDI_RX	TP1	TestPoint:TestPoint_Pad_D1.0mm	-
+5V	TP2	TestPoint:TestPoint_Pad_D1.0mm	-
GND	TP3	TestPoint:TestPoint_Pad_D1.0mm	-
EPROM Adapter			
SST39SF010	U5	DIP-32 W15.24	39SF02070-4C-P (256x8k)
DIP32 Socket	U5 socket	Package_DIP:DIP-32_W15.24mm_Socket	IS25T1-332B
SW_Coded_SH-7050	SW2	Aschenbach:RDS-16S-1045-D	KDR 16
100	R13	Resistor_SMD:R_0603_1608Metric	VIS CRCW06031002
10k	R7,R8,R9,R10,R11,R12	Resistor_SMD:R_0603_1608Metric	SMD-0603 10K
4V7	D3	Diode_SMD:D_SOD-123	BZX 384-C4V7 NXP
EPROM-Pin1-14	J8	Connector_PinHeader_2.54mm:PinHeader_1x14_P2.54mm_Vertical	BKL 10120538 (2* 14 of 32 pins)
EPROM-Pin15-28	J9	Connector_PinHeader_2.54mm:PinHeader_1x14_P2.54mm_Vertical	-
A13	J10	TestPoint:TestPoint_THTPad_D2.0mm_Drill1.0mm	-
MIDI/UI board			
SW_Push	SW1	Button_Switch_THT:SW_Tactile_SPST_Angled_PTS645Vx83-2LFS	TASTER 3305B
LED	D1	LED_THT:LED_D3.0mm_Horizontal_O6.35mm_Z6.0mm	KBT L-7104HD
MIDI-TRS-TypeA	J1	Connector_Audio:Jack_3.5mm_Ledino_KB3SPRS_Horizontal	EBS 35
MIDI/UI	J3	Connector_PinSocket_2.54mm:PinSocket_1x05_P2.54mm_Vertical	PS 25/5G WS (Buchse+Kabel)

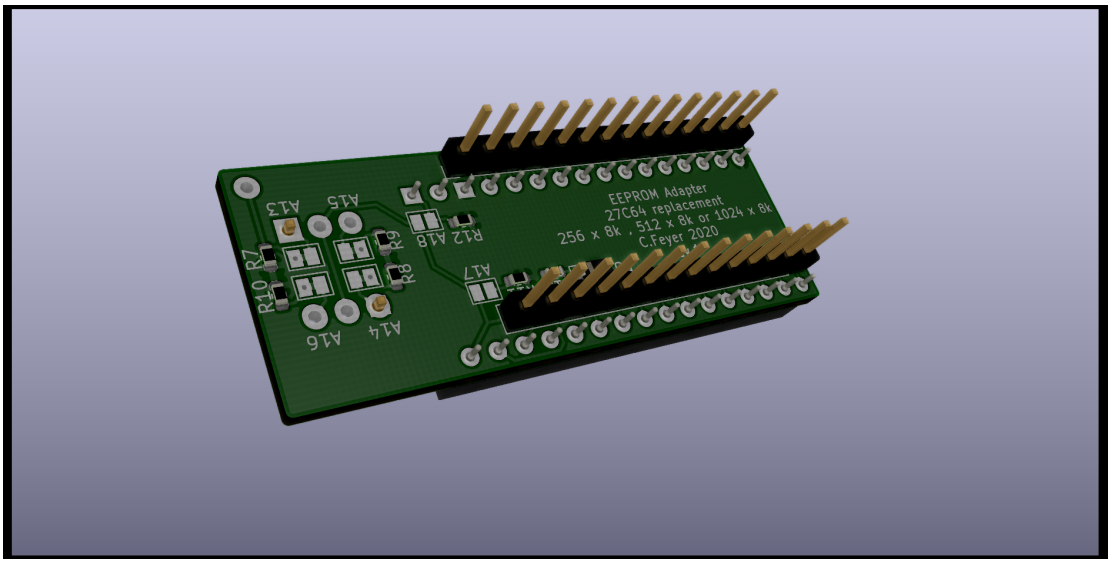


Abbildung 3.7: Unterseite des EPROM-Adapters.

3.0.4 Bestückung der Platinen

Bevor mit der SMD-Bestückung der Platinen begonnen werden konnte, musste zunächst die Lotpaste auf die Top-Lage der Platinen aufgebracht werden. Dazu wurde die Interface-Platine auf der Arbeitsfläche fixiert. Dann wurde die Schablone (Stencil) für die Lotpaste passgenau ausgerichtet und an einer Seite mit Klebeband befestigt. Auf die Schablone wurde Lotpaste aufgetragen und mit einem Zug des Rakels durch die Öffnungen gedrückt. Im Anschluss erfolgte die Bestückung der SMD Bauteile mit einer Pinzette. Die Bauteile wurden vorsichtig auf der Lotpaste fixiert. Die bestückten Platinen wurden anschließend in einem Reflow-Lötofen erhitzt, das eingestellte Temperaturprofil wurde auf die verwendete Paste abgestimmt. Nach dem Abkühlen wurden die Lötstellen unter der Lupe kontrolliert und gegebenenfalls mit einem Lötkolben und Flussmittelstift nachgearbeitet. Im Anschluss wurden die Buchsen und Stiftleisten, sowie die LED, der Taster und die Verbindungsleitung eingelötet.

Kapitel 4

Firmware

Die Auswertung der MIDI-Daten und die entsprechende Auswahl der Klänge in der DPM-48 sollte von einem ATmega88 Mikrocontroller erfolgen. Die zum Betrieb nötige Firmware wurde in der WinAVR-IDE in C geschrieben. Für das MIDI- und EEPROM-handling des Mikrocontrollers wurde quelloffener Code des x0xb0x-Projektes[1] verwendet und mit Hilfe der Application Note AVR094 vom ATmega8 zu ATmega88 portiert[2]. Zur Erstellung des C-Code wurde das AVR Instruction Set verwendet[3]. Der Mikrocontroller wird mit 8 MHz des internen Oszillators an +5 V Betriebsspannung betrieben. Die Firmware erkennt MIDI-Note-On Messages von Note C0 bis A7. Die Klänge werden in allen Oktaven ab C0 wiedergegeben, unabhängig davon, ob sie ein EPROM mit mehreren Bänken verwenden.

4.0.1 Programmablauf

Nach Initialisierung der Hardware und des UART werden die gespeicherten Daten für den MIDI-Kanal und die Zustände der Adress-Leitungen von den Sound-Bank Erweiterungen aus dem internen EEPROM des ATmega88 gelesen. Die Main-Loop enthält eine Abfrage auf Einstellung eines neuen MIDI-Kanals. Diese Abfrage wird wahr wenn die ISR(INT0_vect) eine mindestens viersekündige Betätigung des Tasters erkennt. Dann kann innerhalb von acht Sekunden eine beliebige MIDI Note-On Message auf einem beliebigen Kanal gesendet werden. Dieser Kanal wird dann in der Funktion MIDI_Kanal() als neuer MIDI-Kanal für das Interface im EEPROM gespeichert. Zudem wird auch der aktuelle Status der Adressleitungen der Sound-Bank Erweiterung gespeichert. Wird innerhalb der acht Sekunden, in denen die LED blinkt und der Rimshot-Klang ertönt, keine gültige MIDI-Message erkannt bleibt der zuvor eingestellte Kanal erhalten. Ebenfalls innerhalb der Main-Loop wird der MIDI-handling Code ausgeführt. Die am UART eingehenden seriellen Bytes werden wenn sie Note-On Messages mit Velocity größer Null sind und auf dem eingestellten MIDI-Kanal übertragen wurden an die MUX4051() Funktion übergeben. Diese skaliert die empfangene Note-On Message je nachdem auf welcher Oktave sie gesendet wurde und stellt die Adress-Select Leitungen A13 und A14 für die Sound-Bank Erweiterung ein. Anschließend wird entsprechend der Note-Number eine

[1] https://sourceforge.net/projects/x0xb0x/

[2] http://ww1.microchip.com/downloads/en/AppNotes/doc2553.pdf

[3] http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf

von 22 möglichen Kombinationen der drei Pad-Select und acht Trigger-Select Leitungen an die Funktionen PADset() und Yselect() übergeben. Die Funktion PADset() steuert die Adressleitungen des 3-zu-1 Multiplexers, Yselect() die des 1-zu-8 Demultiplexers. Außerdem setzt letztere auch die Adressleitungen für die Sound-Bank Erweiterung in ausgewählten Gruppen. Mit dem Funktionsaufruf MUXenable(ON) wird dann der entsprechende Klang angetriggert. Der gesamte Quellcode befindet sich in Anhang B (Seite 26).

Kapitel 5

Sound-Daten und erweiterte Sound-Bänke

Die DPM-48 Drum-Machine verwendet zur Wiedergabe der gespeicherten Klänge 8-Bit unsigned PCM (pulse code modulation) als Audioformat. Die Samples der Schlagzeug- und Percussion-Klänge sind auf elf 8 KiByte (8192 Bit) großen EPROMs vom Typ 2764 gespeichert. Zur Analyse wurden die einzelnen EPROMs aus der Drum-Machine genommen und mit einem EPROM-Programmiergerät in den PC eingelesen. Die einzelnen Samples sind zwischen 2 KiByte, 4 KiByte und 16 KiByte groß (siehe Abb. 5.1). Mit der verwendeten Abtastrate von 20 kHz ergeben sich daraus Sample-Längen zwischen ca. 102 ms, 204 ms und 819 ms. Die beiden TOM-Gruppen werden mit 10 kHz abgetastet und ergeben ca. 409 ms lange Klänge. Die Abbildung 5.2 zeigt die importierten Daten jedes EPROMS in einem Audio-Editor. Pro Gruppe ist ein 8-Bit DAC (Digtal-Analog-Converter) bestehend aus Operationsverstärkern in invertierender Summierschaltung mit anschließender erneuter Invertierung aufgebaut. Nachfolgend besitzt jede Gruppe einen VCA, der von einer analogen Hüllkurve geöffnet wird. Die Hüllkurve wird bei einigen identischen Samples mit unterschiedlicher Intensität verwendet (z.B. bei SNARE 1 und SNARE 2 und bei RIDE 1 und RIDE 2).

Zur Erweiterung der Sound-Bänke wurden für ausgewählte EPROMs neue Daten erstellt. Die Anordnung der Daten erfolgt nach den originalen Daten. Es wurden EPROMS mit zwei und vier Sound-Bänken vorbereitet. Der unterste 8 KiByte große Block im Speicher des jeweiligen EPROM beinhaltet die originären Daten des EPROMs der jeweiligen Gruppe. In dem oder den nächsthöheren Blöcken sind neue Samples angeordnet. Die Daten dazu stammen im Fall der digitalen Drum-Machines (Linn LM-1 und 9000, Oberheim DMX und DX und Sequential Circuits Drumtraks) von deren EPROM-Daten und wurden aus dem Companded (mu-law) Datenformat zu 8 Bit unsigned PCM konvertiert. Die Samples der analogen Drum-Machines (Roland TR-606, TR-808, TR-909) wurden als .wav-Files importiert, geschnitten und konvertiert. Die Abbildung 5.3 zeigt die neue Anordnung im Speicher der 16 bzw. 32 KiByte großen EPROMs. Für die Gruppen „HI-HAT“, „PERCUSSION 1“ und „PERCUSSION 2“ wurde eine weitere Bank angelegt, für die Gruppen „BASS“ und „SNARE“ wurden drei weitere Bänke angelegt.

Die Daten wurden mit dem Programmiergerät auf EPROMs vom Typ 27128 bzw. 27256 gebrannt und die Sichtfenster vor versehentlichem Löschen mit lichtdichten Etiketten zugeklebt. Die EPROMs wurden in fünf vorbereitet Adapterplatinen gesteckt (siehe Abb. 5.4).

File	(low)	Block arrangement	(high)	sample rate	Counter
002.BIN	TOM2 (4K)		TOM1 (4K)	10 kHz	IC24
003.BIN	TOM4 (4K)		TOM3 (4K)	10 kHz	IC20
004.BIN	free (2K)	BASS1 (2K)	BASS2 (2K)	BASS3 (2K)	20 kHz	IC15
005.BIN	CLOSED (2K)	ACCENT (2K)	OPEN (4K)	20 kHz	IC17
006.BIN	SNARE3 (4K)		SNARE12 (4K)	20 kHz	IC18
007.BIN			RIDE12 1 of 2 (8K)	20 kHz	IC26
008.BIN			RIDE12 2 of 2 (8K)	20 kHz	IC26
009.BIN			CRASH 1 of 2 (8K)	20 kHz	IC26
010.BIN			CRASH 2 of 2 (8K)	20 kHz	IC26
012.BIN	free (2K)	AGO2 (2K)	AGO1 (2K)	RIM (2K)	20 kHz	IC23
013.BIN	free (2K)	CAB2 (2K)	CAB1 (2K)	CLAP (2K)	20 kHz	IC21

Note:
004.BIN has DC-offset
006.BIN & 007/008.BIN Pad 1+2 uses same sample but different hardware envelope

Christian Feyer 2013/2021

Abbildung 5.1: DPM-48 EPROM Daten.

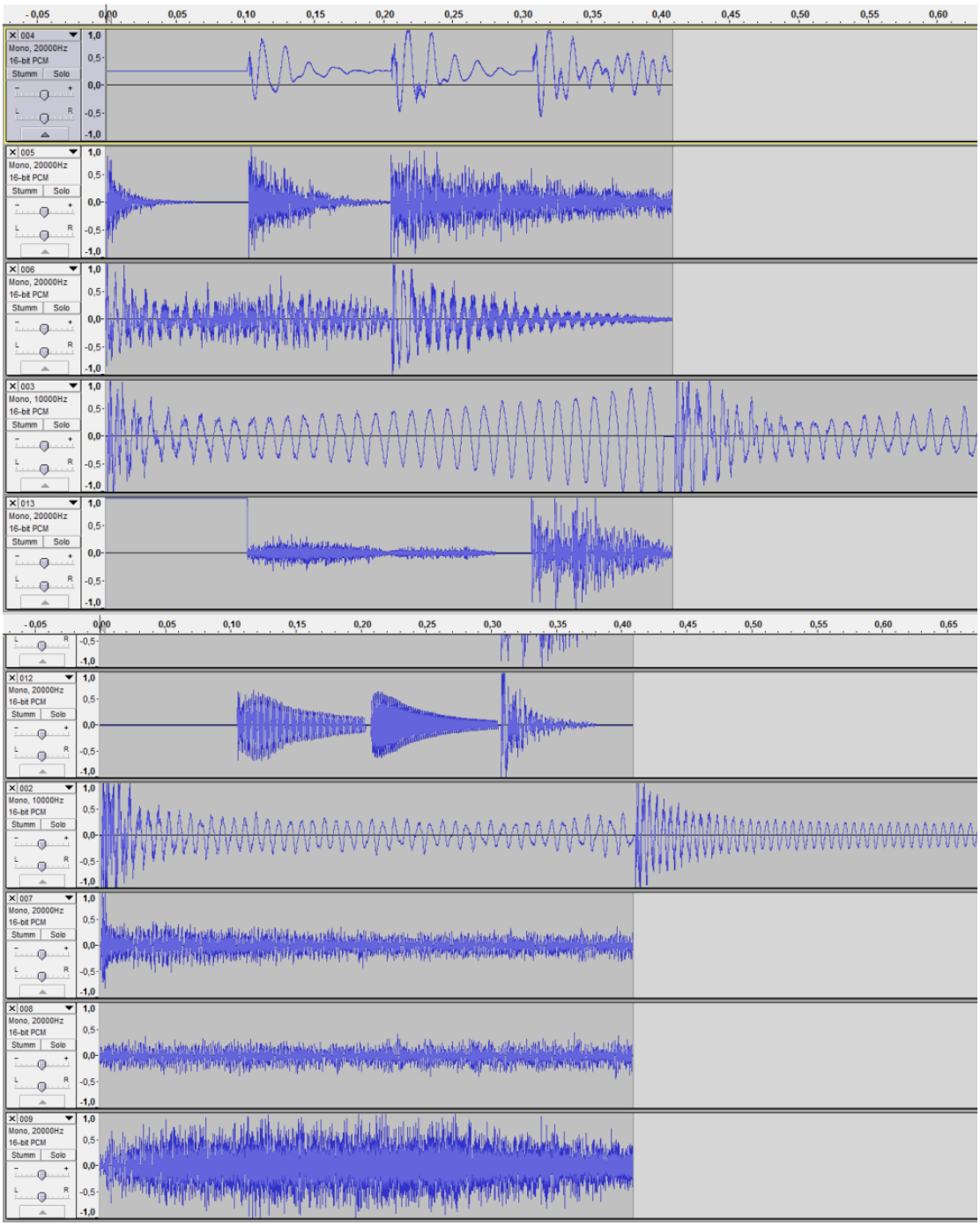


Abbildung 5.2: DPM-48 EPROM Daten im Audio-Editor.

File	(low)	Block arrangement	(high)	sample rate	Counter
004-23012021-C.BIN	free (2K)	BASS1 (2K)	BASS2 (2K)	BASS3 (2K)	20 kHz	IC15
Block 2	free (2K)	bass01 (Linn 9000) (2K)	21KICK (DMX) (2K)	bas1 (LM-1) (2K)	20 kHz	
Block 3	free (2K)	bass1 (Linn Var) (2K)	bass6 (Linn Var) (2K)	bass (Drumtraks) (2K)	20 kHz	
Block 4	free (2K)	bd (TR-606) (2K)	bassdrum6 (TR-808) (2K)	bassdrum1 (TR-909) (2K)	20 kHz	
005-23012021-A.BIN	CLOSED (2K)	ACCENT (2K)	OPEN (4K)	20 kHz	IC17
Block 2	SHAKE6_A (DMX) (2K)	SHAKE6_B (DMX) (2K)	HAT1A (DMX) (4K)	20 kHz	
006-23012021-E.BIN	SNARE3 (4K)	SNARE12 (4K)	20 kHz	IC18
Block 2	SNARE6 (DMX) (4K)	ELECSNAR (DX) (4K)	20 kHz	
Block 3	FATSNARE (DX) (4K)	Snar23 (LM-1) (4K)	20 kHz	
Block 4	snaredrum7 (TR-808) (4K)	snaredrum1 (TR-909) (4K)	20 kHz	
012-23012021-D.BIN	free (2K)	AGO2 (2K)	AGO1 (2K)	RIM (2K)	20 kHz	IC23
Block 2	free (2K)	STIK_A (DMX) (2K)	STIK_B (DMX) (2K)	SNARE6 (DMX) (2K)	20 kHz	
013-23012021-B.BIN	free (2K)	CAB2 (2K)	CAB1 (2K)	CLAP (2K)	20 kHz	IC21
Block 2	free (2K)	? (2K)	CLIKclap (OB Var) (2K)	clikCLAP (OB Var)(2K)	20 kHz	

Abbildung 5.3: Erweiterte EPROM Daten.

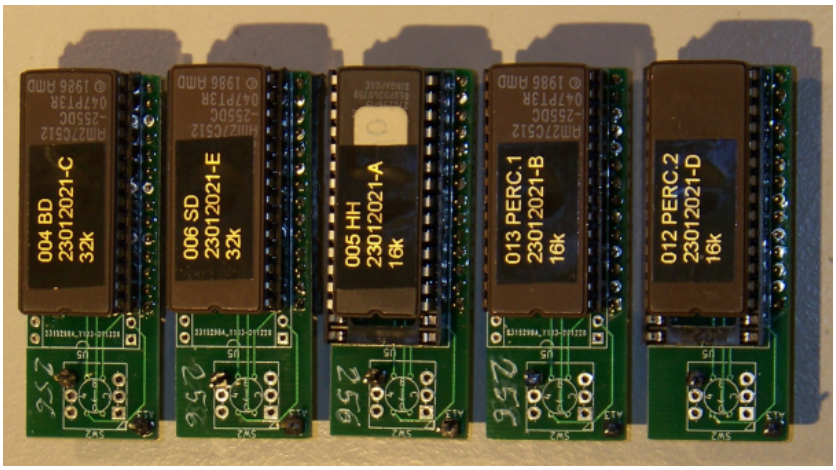


Abbildung 5.4: Beschriebene Sound-Bank Erweiterungen.

Kapitel 6

Einbau des Interfaces und der Sound-Erweiterungen

Die aufgebauten und getesteten Platinen (siehe Abb. 6.1) wurden anschließend in die DPM-48 eingebaut. Nachdem das Gerät geöffnet und die Flachbandleitungen abgezogen waren, wurden Deckel, Rückseite und Boden separiert (siehe Abb. 6.2). Die Trigger-Buchse an der Rückseite wurde ausgebaut und an deren Stelle die MIDI/UI-Platine mit der Frontplatte eingebaut (siehe Abb. 6.3). Die EPROMS der Gruppen, die eine Erweiterung der Sound-Bank erhalten sollten, wurden herausgenommen und durch die Adapter-Platinen ersetzt (siehe Abb. 6.4). Die Interface-Platine wurde auf die weiße 12-polige Stiftleiste der DPM-48-Prozessorplatine gesteckt und auf der Unterseite mit doppelseitigem Schaumklebeband gesichert. Die fünfadrige Leitung zur MIDI/UI-Platine wurde an J2 aufgesteckt und die A13- und A14 Anschlüsse der EPROM-Adapter Platinen entsprechend der Konfiguration der Firmware mit J6 und J7 verbunden (siehe Abb. 6.5). Im Anschluss wurde das Gerät wieder zusammen gebaut. Zum Debuggen der Firmware wurde die ISP-Leitung temporär auf der Interface-Platine belassen und durchs Gehäuse herausgeführt (siehe Abb. 6.6).

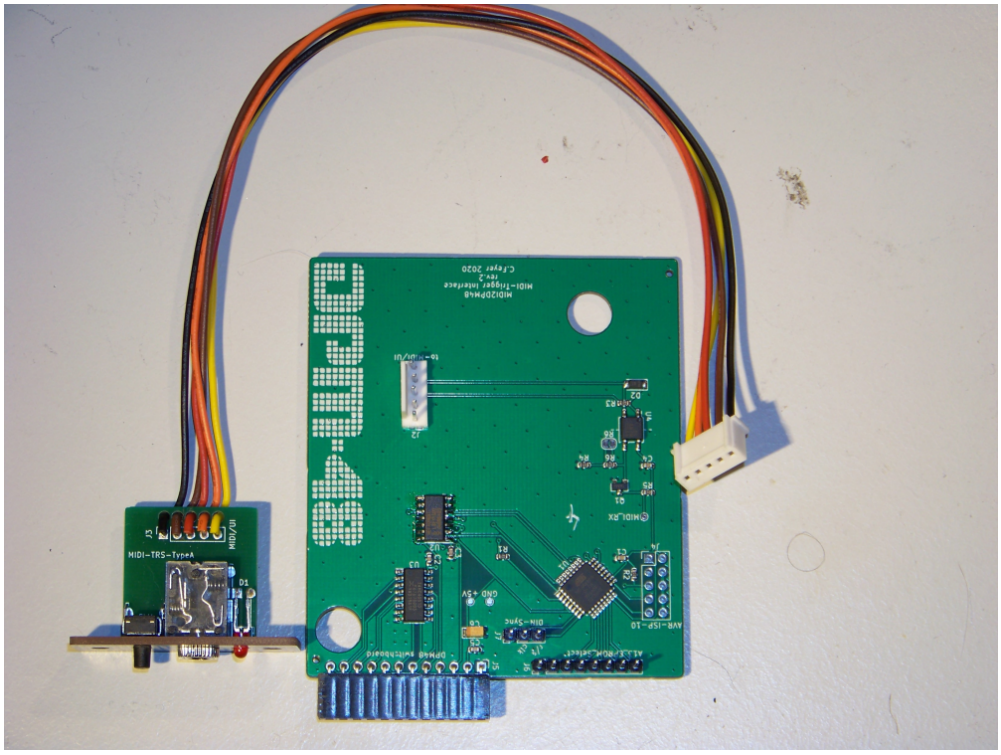


Abbildung 6.1: Fertig aufgebautes Interface mit MIDI/UI.

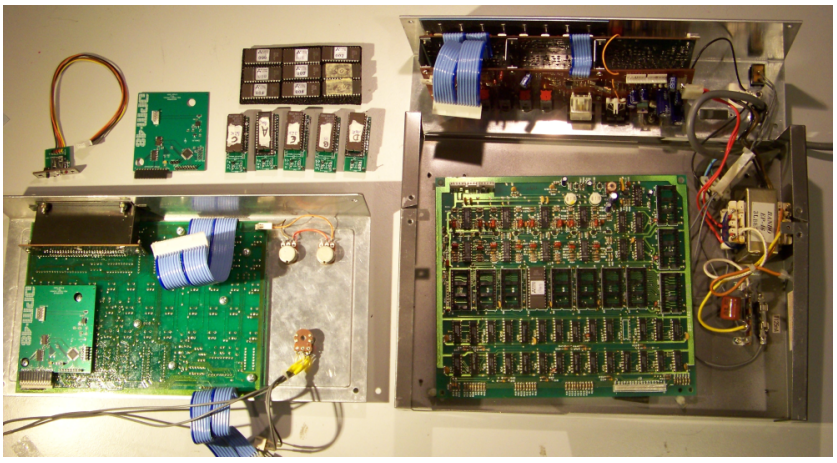


Abbildung 6.2: Zerlegte DPM-48 Drum-Machine mit Interface und EPROMs.

Abbildung 6.3: Eingebautes MIDI/UI-Panel mit TRS-DIN Adapterkabel.

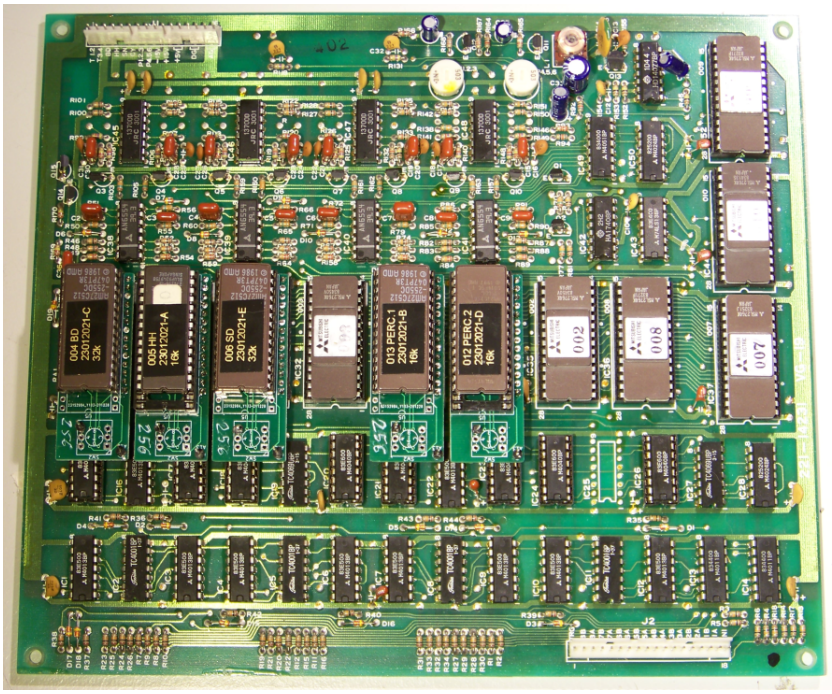


Abbildung 6.4: DPM-48 Soundboard mit eingesetzten EPROM-Adapter-Platinen.

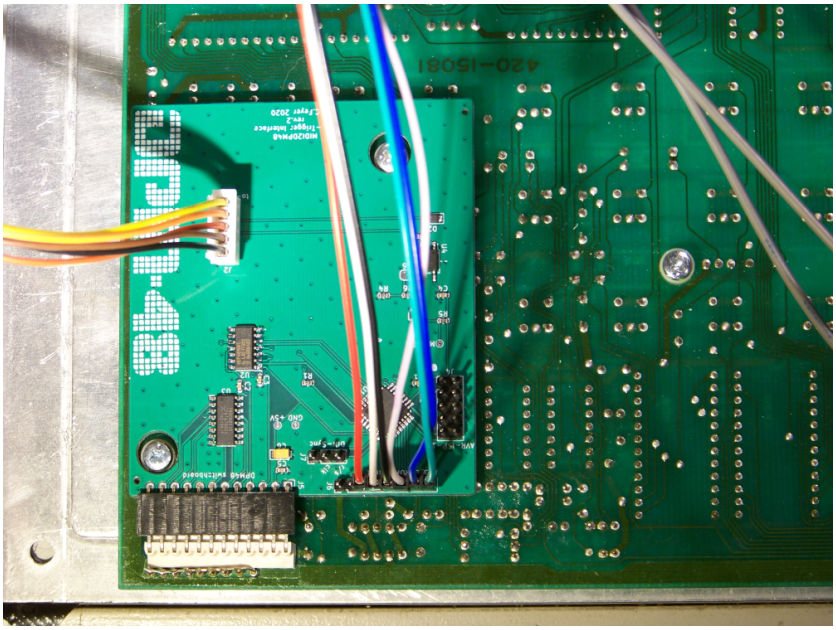


Abbildung 6.5: Auf DPM-48 Prozessorboard aufgesetztes MIDI-Interface.

Abbildung 6.6: DPM-48 mit eingebautem MIDI-Interface und ISP-Leitung.

Kapitel 7

Fazit

Nach erfolgtem Einbau des MIDI-Interfaces in die DPM-48 Drum-Machine können die einzelnen Klänge über ein angeschlossenes MIDI-Keyboard beginnend bei der Note C0 getriggert werden. In den jeweils höheren Oktaven an Note C2, C4 oder C6 können die erweiterten Sound-Bänke angespielt werden. Der Lernmodus für den MIDI-Kanal funktioniert wie gewünscht. In einem weiteren Test wurde ein MIDI-Sequenzer verwendet um auf mehreren Spuren die Klänge in der DPM-48 zu steuern. Auch bei höherem Tempo und vielen gleichzeitigen Noten funktioniert das Interface wie gewünscht.

Das Interface konnte eingebaut werden ohne Änderungen oder Beschädigungen am Gerät zu verursachen. Alle Eingriffe sind reversibel.

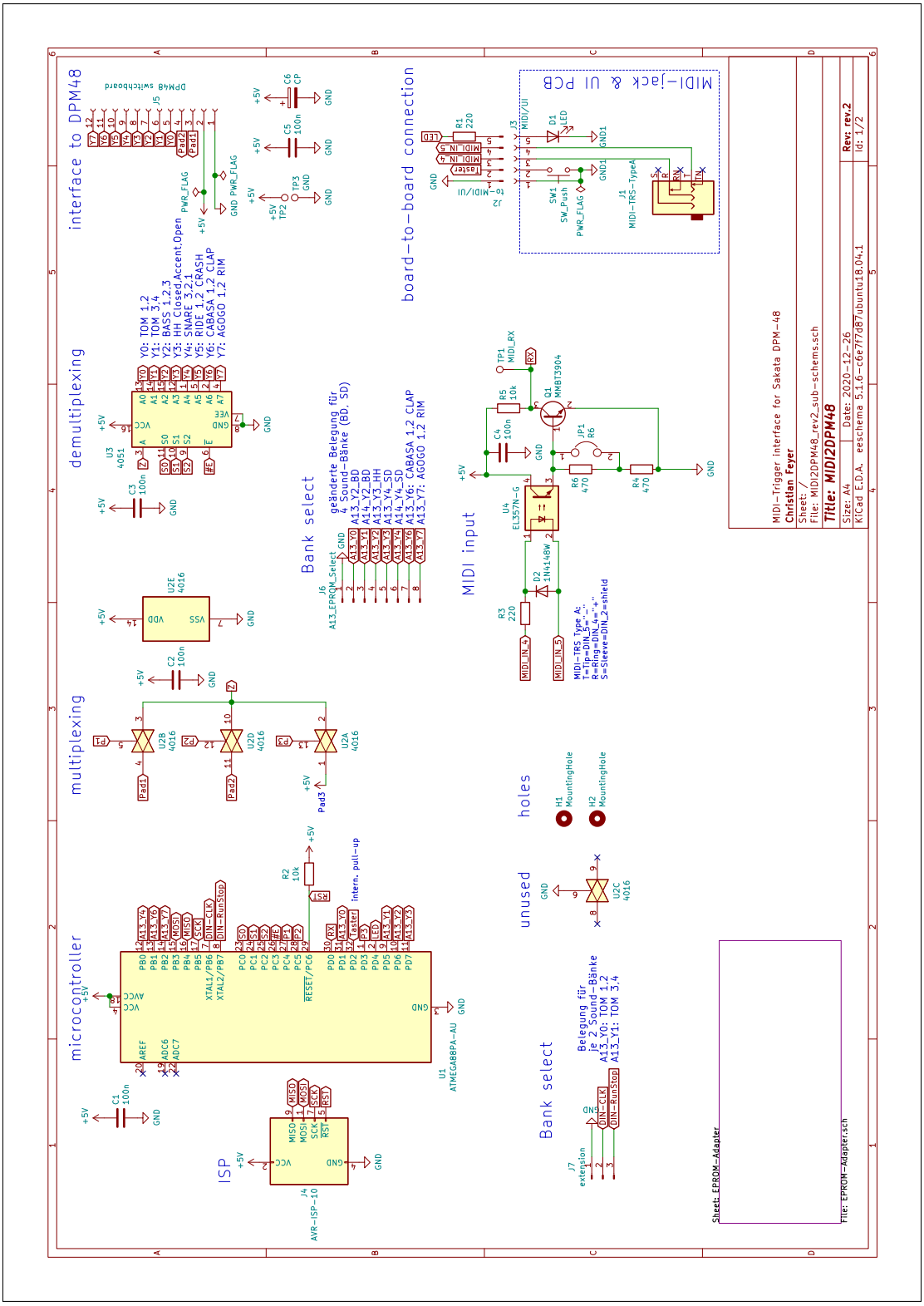
Durch den Einbau des entwickelten MIDI-Interfaces konnten die Limitationen der DPM-48 umgangen werden. Zudem wurde die Klang-Palette durch die Erweiterung der Sound-Bänke vergrößert.

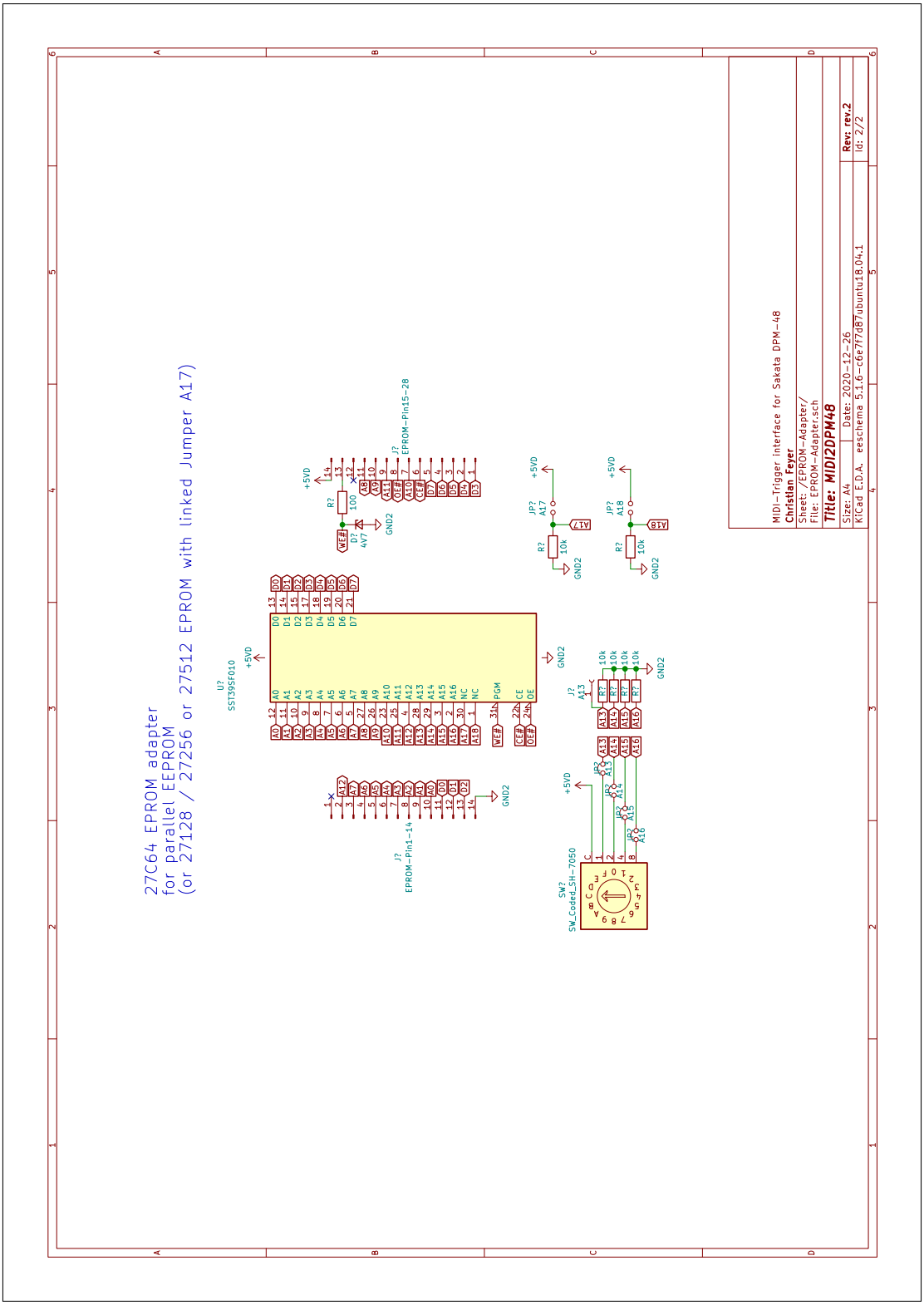
Eine Weiterentwicklung des Interfaces könnte acht unabhängig voneinander über MIDI-CC-Messages einstellbare Clock-Signale bereitstellen. Diese würden dann anstelle der festen Sample-Raten mit den Binärzählern der jeweiligen Gruppe verbunden und könnten so das Stimmen der Klänge ermöglichen.

Anhang A

Schaltplan

Schaltpläne für MIDI-Interface, MIDI/UI-Platine und EPROM-Adapter


interface to DPM48
demultiplexing
multiplexing
microcontroller
board-to-board connection
MIDI-jack & UI PCB
MIDI input
Bank select
unused holes
ISP
MIDI-Trigger interface for Sakata DPM-48
Christian Feyer
Title: MIDI2DPM48
Rev: rev.2
Date: 2020-12-26
Size: A4
KiCad E.D.A. eeschema 5.1.6-c6e7f7d87ubuntu18.04.1
Id: 1/2
Sheet: EPROM-Adapter
File: EPROM-Adapter.sch


27C64 EPROM adapter
for parallel EEPROM
(or 27128 / 27256 or 27512 EPROM with linked Jumper A17)
MIDI-Trigger interface for Sakata DPM-48
Christian Feyer
Sheet: /EPROM-Adapter/
File: EPROM-Adapter.sch
Title: MIDI2DPM48
Size: A4
Date: 2020-12-26
Rev: rev.2
KiCad E.D.A. eeschema 5.1.6-c6e7f7d87ubuntu18.04.1
Id: 2/2
U2
SST39SF010
J2
EPROM-Pin15-28
J1
EPROM-Pin1-14
SW?
SW_Coded_SH-7050
JP?
A17
JP?
A18
R?
10k
R?
100
D?
4V7
+5VD
GND2
WE#
CE#
OE#
PGM
NC

Anhang B

C-Quellcode

B.1 Hauptprogramm

MIDI2DPM48_V2.c

```
/*
 * MIDI2DPM48_V2.c
 *
 * Created: 27.01.2013 22:18:25
 * Edited: 17.01.2021 (migration von ATmega8 zu ATmega88)
 *  Author: Christian Feyer
 *
 *  MIDI to Trigger Interface for Hammond/Sakata DPM48 8-Bit linear PCM 2764 EPROM based Drum-machine
 *
 *    Ersetzt die Trigger-Buchse (fuer Drum-Pad Interface) am Geraet. Das MIDI-Interface simuliert das Verhalten des Drum-Pad Interfaces.
 *
 *    8 Triggergruppen * 3 Trigger-Signale -> 22 verschiedene Sounds
 *    zusaetzliche A13 und A14 Select-Leitungen fuer zweite Speicherbank in 27*128 EPROMs bzw. zweite, dritte, vierte Speicherbank in 27*256 EPROMs
 *
 *    MIDI Kanal Lernmodus, speichert auch aktuellen Zustand der A13 select Leitungen:
 *       ca. 4s Taster betaetigt halten, blinkende LED (oder RIM) signalisiert dann ca. 8s lang
 *       den Lernmodus, eingehender NOTE_ON MIDI Befehl auf ausgewaehltem Kanal gibt den neuen
 *       MIDI Kanal vor
 *
 *    Die 22 Sounds sind auf zwei Oktaven verteilt (die jeweils letzten beiden Noten der zweiten Oktave werden nicht benutzt)
 *
 *    Niedrigste Note, Original-Bank: 0x0C = 12 = C0 = TOM1 (A13 = 0; A14 = 0)
 *              zweite Bank: 0x24 = 36 = C2 = TOM1 (A13 = 1; A14 = 0)
 *              dritte Bank: 0x3C = 60 = C4 = TOM1 (A13 = 0; A14 = 1)
 *              vierte Bank: 0x54 = 84 = C6 = TOM1 (A13 = 1; A14 = 1)
 *
 *    Sounds werden in allen Oktaven wiedergegeben. Unabhaengig davon, ob sie ein EPROM mit mehreren Baenken verwenden
 *
```

```
29  *
30  *  Mikroprozessor: ATmega88P
31  *  intern RC Osc 8MHz
32  *
33  *  FUSEs:
34  *  [ATmega8:   Low FUSE: 0xE4    High FUSE: 0xD9   Lockbits: 0xFF]
35  *  ATmega88P:    Low Fuse: 0xE2    High Fuse: 0xDF   Extended Fuses: 0xF9  Lock: 0
        xFF
36  *
37  * -Int. RC Osc. 8 MHz; Start-up time: 6 CK + 64 ms
38  * -Serial program downloading (SPI) enabled
39  * -Boot Flash section size=1024 words
40  *
41  */
42
43 #define F_CPU 8000000UL   // intern 8MHz
44
45 #include <avr/io.h>
46 #include <util/delay.h>
47 #include <stdlib.h>     // fuer: itoa(int value, char* buffer, int radix)
48 #include <avr/interrupt.h>  // Headerdatei fuer Interrupts
49 #include "MIDI_IN.h"
50 #include "int_eeprom.h"
51
52 // Eingaenge
53 #define TASTER PD2  // Taster an INT0
54
55 // Ausgaenge
56 #define S0 PC0  // Select Leitung 0 4051 MUX
57 #define S1 PC1  // Select Leitung 1 4051
58 #define S2 PC2  // Select Leitung 2 4051
59 #define nE PC3  // Enable Leitung   4051
60 #define P1 PC4  // Schalter PAD1 Signal DPM48
61 #define P2 PC5  // Schalter PAD2 Signal DPM48
62 #define P3 PD3  // Schalter PAD3 Signal DPM48
63 #define LED PD4 // LED
64
65 // EPROM select Leitungen A13
66 // Original setup, bzw. Reihenfolge von Stiftleiste J6
67 // #define A13_Y0 PD1 // A13 EPROM select fuer Kanal Y0 TOM12
68 // #define A13_Y1 PD5 // A13 EPROM select fuer Kanal Y1 TOM34
69 // #define A13_Y2 PD6 // A13 EPROM select fuer Kanal Y2 BASS
70 // #define A13_Y3 PD7 // A13 EPROM select fuer Kanal Y3 HI-HAT
71 // #define A13_Y4 PB0 // A13 EPROM select fuer Kanal Y4 SNARE
72 // nicht fuer Y5 (RIDE/CRASH), zuviele EPROMS
73 // #define A13_Y6 PB1 // A13 EPROM select fuer Kanal Y6 PERC1
74 // #define A13_Y7 PB2 // A13 EPROM select fuer Kanal Y7 PERC2
75
76 // EPROM select Leitungen A13 & A14
77 // nur A13: 2 Sound-Baenke; A13&A14: 4 Sound-Baenke
78 #define A13_Y0 PB6  // A13 EPROM select fuer Kanal Y0 TOM12
79 #define A13_Y1 PB7  // A13 EPROM select fuer Kanal Y1 TOM34
80 #define A13_Y2_BD PD1 // A13 EPROM select fuer Kanal Y2 BASS
81 #define A14_Y2_BD PD5 // A14 EPROM select fuer Kanal Y2 BASS
82 #define A13_Y3_HH PD6 // A13 EPROM select fuer Kanal Y3 HI-HAT
83 #define A13_Y4_SD PD7 // A13 EPROM select fuer Kanal Y4 SNARE
84 #define A14_Y4_SD PB0 // A14 EPROM select fuer Kanal Y4 SNARE
```

```
85 // nicht fuer Y5 (RIDE/CRASH), zuviele EPROMS
86 #define A13_Y6 PB1  // A13 EPROM select fuer Kanal Y6 PERC1
87 #define A13_Y7 PB2  // A13 EPROM select fuer Kanal Y7 PERC2
88
89
90
91
92
93 // MUX 4051 Switch Select Z->Y0..7
94 #define Y0 0  // TOM 1+2
95 #define Y1 1  // TOM 3+4
96 #define Y2 2  // BASS 1+2+3
97 #define Y3 3  // Cl./ Acc./ Open HH
98 #define Y4 4  // SNARE 1+2+3
99 #define Y5 5  // RIDE 1+2 / CRASH
100 #define Y6 6  // CABASA 1+2 / CLAP  (PERC.1)
101 #define Y7 7  // AGOGO 1+2 / RIM    (PERC.2)
102 #define OFF 0
103 #define ON 1
104
105 #define MIDIIN_ADDR_EEADDR 0x1  // int. eeprom speicherplatz fuer MIDI Kanal
106 #define A13_ADDR_EEADDR_D 0x2 // int. eeprom speicherplatz fuer A13 selects PORTD
107 #define A13_ADDR_EEADDR_B 0x3 // int. eeprom speicherplatz fuer A13 selects PORTB
108
109 // globale variablen
110 volatile uint8_t key, vel;    // MIDI key & vel
111 volatile uint8_t sel_eben=8;    // merkt letzten Y select
112
113 // in MIDI_IN.c
114 extern volatile uint8_t note, velocity;
115 extern volatile uint8_t midi_in_addr;
116
117 volatile uint8_t learnflag=0; // wird 1 wenn TASTER 3s betaetigt
118 volatile uint8_t a13_select=00; // A13 select flag
119
120 // Funktionen
121 void HARDWARE_init(void);
122 void USART_init();
123 void PADset (uint8_t pad);
124 void Yselect (uint8_t sel);
125 void MUXenable (uint8_t status);
126 void MUX4051 (uint8_t key, uint8_t vel);
127 void MIDI_Kanal(void);
128 void TIMER_init(void);
129 uint8_t get_midi_addr(uint8_t eeaddr);
130 void save_a13_selects(void);
131 void get_a13_selects(void);
132
133
134 int main(void)
135 {
136   HARDWARE_init();  //Initialisierung der Hardware
137   USART_init();   //Initialisierung der USART-Schnittstelle
138
139   MUXenable(OFF);   //MUX ausschalten
140
141   midi_in_addr = get_midi_addr(MIDIIN_ADDR_EEADDR); // MIDI Kanal aus eeprom holen
```

```
142   get_a13_selects();  // gespeicherte A13 holen
143
144     while(1)
145     {
146     if(learnflag == 1) // MIDI Kanal Wechsel
147     {
148       _delay_ms(2);
149       MIDI_Kanal();
150       learnflag=0;
151     }
152
153     do_midi_mode(); // Hauptprogramm
154     }
155 }
156
157 ISR(INT0_vect)  // ISR fuer TASTER betaetigt
158 { // warte 200ms
159   // wenn taster noch gedrueckt(null): timer starten & LED an, sonst return
160   // wenn bei timer compare taster noch gedrueckt: learnflag setzen
161
162   _delay_ms(200);
163   if ( !(PIND & (1<<TASTER)) )  // wenn TASTER 0 ist (gedrueckt)
164   {
165     PORTD |= (1<<LED);  // LED an
166     TIMER_init();
167     while (TCNT1 < 23437) // laeuft fuer 3 sekunden, TASTER solange gedrueckt halten
168     {
169       if ( PIND & (1<<TASTER) ) // wenn TASTER 1 ist (losgelassen)
170       {
171         PORTD &= ~(1<<LED); // LED aus
172         return; // ABBRUCH
173       }
174     }
175     //setze global flag
176     learnflag=1;
177     PORTD &= ~(1<<LED); // LED aus
178     return;
179   }
180   else return;  // ABBRUCH
181 }
182
183 void get_a13_selects(void)    // holt gespeicherte A13 aus int. eeprom
184 {
185   uint8_t a13=0;
186
187   // default wert
188   if (internal_eeprom_read8(A13_ADDR_EEADDR_D) == 0xFF)
189   {
190     a13 = 0;
191   }
192   else
193   {
194     a13 = internal_eeprom_read8(A13_ADDR_EEADDR_D);
195   }
196   PORTD |= a13; //nur vormals maskierte bits setzen
197
198   // default wert
```

```
199   if (internal_eeprom_read8(A13_ADDR_EEADDR_B) == 0xFF)
200   {
201     a13 = 0;
202   }
203   else
204   {
205     a13 = internal_eeprom_read8(A13_ADDR_EEADDR_B);
206   }
207   PORTB |= a13; //nur vormals maskierte bits setzen
208 }
209
210 void save_a13_selects(void)   // speichert aktuelle A13 selects im int. eeprom
211 {
212   uint8_t a13=0;
213
214   a13 = PIND & ( (1<<A13_Y4_SD)|(1<<A13_Y3_HH)|(1<<A14_Y2_BD)|(1<<A13_Y2_BD) ); //
        nur A13s ausmaskieren
215   internal_eeprom_write8(A13_ADDR_EEADDR_D, a13); // burn to EEPROM
216
217   a13 = PINB & ( (1<<A13_Y7)|(1<<A13_Y6)|(1<<A14_Y4_SD)|(1<<A13_Y0)|(1<<A13_Y1) );
        //nur A13s ausmaskieren
218   internal_eeprom_write8(A13_ADDR_EEADDR_B, a13); // burn to EEPROM
219 }
220
221 void MIDI_Kanal(void) // holt neuen MIDI Kanal und speichert im int. eeprom
222 {
223   // ausserhalb der INT0 ISR
224   // waehrend zaehler laeuft max. 8 sekunden blinken, dabei MIDIgetch/MIDIgetchar:
225   //   Kanal ausmaskieren, in variable midi_in_addr schreiben (& eeprom), LED aus
226   uint8_t d;
227
228   TIMER_init();
229   while (TCNT1 < 65535) // fuer 8 sekunden warten auf MIDI-Kanal-Empfang
230   {
231     // blinken verbessern
232     //if (TCNT1H & (1<<5)) PORTD ^= (1<<LED); // LED Blinken waehrend Timer1 laeuft
233
234     if (TCNT1 == 10000 || TCNT1 == 20000 || TCNT1 == 30000|| TCNT1 == 40000 || TCNT1
         == 50000 || TCNT1 == 60000)
235     {
236       PORTD ^= (1<<LED);  // LED Blinken
237
238       //RIM abspielen 6x
239       //PAD3
240       PADset(3);
241       //Y7
242       Yselect(Y7);
243       MUXenable(ON);
244       _delay_ms(2);
245       MUXenable(OFF);
246     }
247
248
249
250
251     if (midi_getch()) // if theres a char waiting in midi queue...
252     {
```

```
253       //PORTD &= ~(1<<LED); // LED au
254       d = midi_getchar();
255       if (d >> 4 == 0x9) // nur NOTEON
256       {
257         midi_in_addr = d & 0xF; // empfangenen MIDI Kanal in Variable schreiben
258         internal_eeprom_write8(MIDIIN_ADDR_EEADDR, midi_in_addr); // burn to EEPROM
259         save_a13_selects();
260         PORTD &= ~(1<<LED); // LED aus
261         return; // FERTIG
262       }
263     }
264   }
265   PORTD &= ~(1<<LED); // LED aus
266   return;
267 }
268
269 uint8_t get_midi_addr(uint8_t eeaddr) // holt MIDI Kanal aus int. eeprom
270 {
271   uint8_t midi_addr;
272
273   midi_addr = internal_eeprom_read8(eeaddr);
274   if (midi_addr > 15)
275     midi_addr = 15; // MIDI Kanal 16
276   return midi_addr;
277 }
278
279 void TIMER_init(void) // Timer init fuer Tastendrueckzeit
280 {
281   // compare-wert_3s = fClk/(1/t) * 1/prescaler = 8MHz/(1/3s) * 1/1024 = 23437
282   TCCR1B |= (1<<CS10)|(1<<CS12);      //Timer 1 ON, Prescaler = 1024
283   TCNT1=0; //Zaehlregiser reseten
284 }
285
286 void HARDWARE_init(void)  // Init aller Ein/ Ausgaenge/ Pullups/ Interrupt enable
287 {
288   DDRD &= ~((1<<TASTER));                      //Eingang PORTD
289   DDRC |= (1<<S0)|(1<<S1)|(1<<S2)|(1<<nE)|(1<<P1)|(1<<P2);    //Ausgang PORTC
290   DDRD |= (1<<P3)|(1<<LED)|(1<<A13_Y2_BD)|(1<<A14_Y2_BD)|(1<<A13_Y3_HH)|(1<<
        A13_Y4_SD); //Ausgang PORTD
291   DDRB |= (1<<A13_Y0)|(1<<A13_Y1)|(1<<A14_Y4_SD)|(1<<A13_Y6)|(1<<A13_Y7); // Ausgang
         PORTB
292   PORTD |= (1<<TASTER);                 //interner PULL UP Widerstand
293
294   EIMSK |= (1<<INT0);   // enable INT0
295   EICRA |= (1<<ISC01);  // IRQ bei fallender Flanke
296
297   sei();    // Set Interrupts (global)
298 }
299
300 void USART_init() //Funktion zur Initialisierung der USART
301 {
302   UBRR0H = 0;   // Einstellen der Baudrate von 9600
303   UBRR0L = 15;  // Formel:
304           // fOSC/(16*baud)-1
305           // hier: 8MHz/(16*9600)-1= 51,083->51
306           // altern: 8MHz/(16*19200)-1= 25,041->25
307           // ***************************************
```

```
308           // MIDI: 8MHz/(16*31250)-1= 15
309           // Test-Verbindung docklight: Baud 38400 -> 12
310   UCSR0B = (1<<RXCIE0) | (1<<RXEN0);    // Empfangen aktivieren
311                     // Receive Interrupt Flagfreischalten
312   UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
313                     // Asynchron  / 8-data-bits / 1 Start / 1 Stop
314 }
315
316 void PADset (uint8_t pad) // Schaltet die PAD1..3 Signale vom DPM48 auf den MUX-
       Eingang
317 {
318   switch (pad)
319   {
320   case 1: PORTC &= ~((1<<P2));  //PAD2 FET ruecksetzen
321       PORTD &= ~((1<<P3));  //PAD3 FET ruecksetzen
322       PORTC |= (1<<P1);   //PAD1 FET setzen
323     break;
324
325   case 2: PORTC &= ~((1<<P1));  //PAD1 FET ruecksetzen
326       PORTD &= ~((1<<P3));  //PAD3 FET ruecksetzen
327       PORTC |= (1<<P2);   //PAD2 FET setzen
328     break;
329
330   case 3: PORTC &= ~((1<<P1)|(1<<P2));  //PAD1 PAD2 FET ruecksetzen
331       PORTD |= (1<<P3);   //PAD3 FET setzen
332     break;
333   }
334   //_delay_us(10);
335 }
336
337 void Yselect (uint8_t sel) // Waehlt einen der 8 Schalter im MUX aus
338 {
339
340
341   // wenn sel == sel_eben dann nE auf HIGH (switch OFF) -> retrigger
342   if (sel == sel_eben) MUXenable(OFF);
343   sel_eben = sel;
344   switch(sel)
345   {
346     case Y0: // TOM 1+2
347     {
348       PORTC &= ~((1<<S0)|(1<<S1)|(1<<S2));  //ruecksetzen
349       if (a13_select == 01)
350       {
351         PORTB |= (1<<A13_Y0); // set A13_Y0
352       }
353       else PORTB &= ~(1<<A13_Y0); //reset A13_Y0
354       break;
355     }
356
357     case Y1: // TOM 3+4
358     {
359       PORTC &= ~((1<<S1)|(1<<S2));  //ruecksetzen
360       PORTC |= (1<<S0);   //setzen
361       if (a13_select == 01)
362       {
363         PORTB |= (1<<A13_Y1); // set A13_Y1
```

```
364       }
365       else PORTB &= ~(1<<A13_Y1); //reset A13_Y1
366       break;
367     }
368
369     case Y2: // BASS
370     {
371       // MUX Schalter
372       PORTC &= ~((1<<S0)|(1<<S2));  //ruecksetzen
373       PORTC |= (1<<S1);   //setzen
374
375       // A13&A13 select Leitungen
376       if (a13_select == 01)
377       {
378         PORTD &= ~(1<<A14_Y2_BD); //reset A14_Y2
379         PORTD |= (1<<A13_Y2_BD);  // set A13_Y2
380       }
381       else if (a13_select == 10)
382       {
383         PORTD |= (1<<A14_Y2_BD);  // set A14_Y2
384         PORTD &= ~(1<<A13_Y2_BD); //reset A13_Y2
385       }
386       else if (a13_select == 11)
387       {
388         PORTD |= (1<<A14_Y2_BD);  // set A14_Y2
389         PORTD |= (1<<A13_Y2_BD);  // set A13_Y2
390       }
391       else // 00
392       {
393         PORTD &= ~(1<<A14_Y2_BD); //reset A14_Y2
394         PORTD &= ~(1<<A13_Y2_BD); //reset A13_Y2
395       }
396
397       break;
398     }
399
400     case Y3: // HI-HAT
401     {
402       PORTC &= ~((1<<S2));  //ruecksetzen
403       PORTC |= (1<<S0)|(1<<S1);   //setzen
404       if (a13_select == 01)
405       {
406         PORTD |= (1<<A13_Y3_HH);  // set A13_Y3
407       }
408       else PORTD &= ~(1<<A13_Y3_HH); //reset A13_Y3
409       break;
410     }
411
412     case Y4: // SNARE
413     {
414       PORTC &= ~((1<<S0)|(1<<S1));  //ruecksetzen
415       PORTC |= (1<<S2);   //setzen
416     // A13&A13 select Leitungen
417       if (a13_select == 01)
418       {
419         PORTB &= ~(1<<A14_Y4_SD); //reset A14_Y4
420         PORTD |= (1<<A13_Y4_SD);  // set A13_Y4
```

```
421       }
422       else if (a13_select == 10)
423       {
424         PORTB |= (1<<A14_Y4_SD);  // set A14_Y4
425         PORTD &= ~(1<<A13_Y4_SD); //reset A13_Y4
426       }
427       else if (a13_select == 11)
428       {
429         PORTB |= (1<<A14_Y4_SD);  // set A14_Y4
430         PORTD |= (1<<A13_Y4_SD);  // set A13_Y4
431       }
432       else // 00
433       {
434         PORTB &= ~(1<<A14_Y4_SD); //reset A14_Y4
435         PORTD &= ~(1<<A13_Y4_SD); //reset A13_Y4
436       }
437 
438       break;
439     }
440 
441     case Y5: // RIDE+CRASH
442         PORTC &= ~((1<<S1));  //ruecksetzen
443         PORTC |= (1<<S0)|(1<<S2);   //setzen
444       break;
445 
446     case Y6: // PERC.1
447     {
448       PORTC &= ~((1<<S0));  //ruecksetzen
449       PORTC |= (1<<S1)|(1<<S2);   //setzen
450       if (a13_select == 01)
451       {
452         PORTB |= (1<<A13_Y6); // set A13_Y6
453       }
454       else PORTB &= ~(1<<A13_Y6); //reset A13_Y6
455       break;
456     }
457 
458     case Y7: // PERC.2
459     {
460       PORTC |= (1<<S0)|(1<<S1)|(1<<S2);   //setzen
461       if (a13_select == 01)
462       {
463         PORTB |= (1<<A13_Y7); // set A13_Y7
464       }
465       else PORTB &= ~(1<<A13_Y7); //reset A13_Y7
466       break;
467     }
468   }
469   _delay_ms(3); // Delay fuer Yselect: fPAD-Signale=1kHz -> 1ms
470   a13_select = 00;  // reset flag
471 }
472 
473 void MUXenable (uint8_t status) // Schaltet die Schalter im MUX hochohmig
474 {
475   switch(status)
476   {
477     case OFF: PORTC |= (1<<nE);   //nE setzen,  Switches Off
```

```
478       break;
479
480     case ON: PORTC &= ~((1<<nE)); //nE ruecksetzen,  Switches On
481   }
482 }
483
484 void MUX4051 (uint8_t key, uint8_t vel) // waehlt PAD und Y fuer best. MIDI-Note
485 {
486   //PORTD ^= (1<<LED);
487   //MUXenable(OFF);
488   //Sende_Char(0xBB); //funktioniert
489
490   // wenn key > 0x21 schalte A13 flag ein und key= key - 24
491   if (key > 0x21 && key <= 0x39)
492   {
493     //PORTD |= (1<<LED);  //set A13
494     key = key - 24;   // minus 2 Oktaven um die Noten-Nummern C0 (0x0C) bis A1 (0x21
        ) in switch(key) auszuwaehlen
495     a13_select = 01;  //  zweite Sound-Bank im EPROM
496
497   }
498   else if (key > 0x39 && key <= 0x51)
499   {
500     //PORTD |= (1<<LED);  //set A13
501     key = key - 48;   // minus 4 Oktaven um die Noten-Nummern C0 (0x0C) bis A1 (0x21
        ) in switch(key) auszuwaehlen
502     a13_select = 10;  // dritte Sound-Bank im EPROM
503
504   }
505   else if (key > 0x51)
506   {
507     //PORTD |= (1<<LED);  //set A13
508     key = key - 72; // minus 6 Oktaven um die Noten-Nummern C0 (0x0C) bis A1 (0x21)
        in switch(key) auszuwaehlen
509     a13_select = 11;  // vierte Sound-Bank im EPROM
510
511   }
512   //else PORTD &= ~(1<<LED); //reset A13
513   // a13_select = 00 entspricht der Ersten Sound-Bank im EPROM (Original DPM-48
        Sounds)
514
515   //Sende_Char(0xB0); //funktioniert
516   switch(key)
517   {
518       case 0x0C: //C0   TOM1
519           //PAD3
520           PADset(3);
521           //Y0
522           Yselect(Y0);
523           MUXenable(ON);
524           break;
525
526       case 0x0D: //C#0  TOM2
527           //PAD2
528           PADset(2);
529           //Y0
530           Yselect(Y0);
```

```
531             MUXenable(ON);
532             break;
533 
534         case 0x0E: //D0   TOM3
535             //PAD3
536             PADset(3);
537             //Y1
538             Yselect(Y1);
539             MUXenable(ON);
540             break;
541 
542         case 0x0F: //D#0  TOM4
543             //PAD2
544             PADset(2);
545             //Y1
546             Yselect(Y1);
547             MUXenable(ON);
548             break;
549 
550         case 0x10: //E0   BASS1
551             //PAD1
552             PADset(1);
553             //Y2
554             Yselect(Y2);
555             MUXenable(ON);
556             break;
557 
558         case 0x11: //F0   BASS2
559             //PAD2
560             PADset(2);
561             //Y2
562             Yselect(Y2);
563             MUXenable(ON);
564             break;
565 
566         case 0x12: //F#0    BASS3
567             //PAD3
568             PADset(3);
569             //Y2
570             Yselect(Y2);
571             MUXenable(ON);
572             break;
573 
574         case 0x13: //G0   Closed Hi-Hat
575             //PAD1
576             PADset(1);
577             //Y3
578             Yselect(Y3);
579             MUXenable(ON);
580             break;
581 
582         case 0x14: //G#0    Accent Hi-Hat
583             //PAD2
584             PADset(2);
585             //Y3
586             Yselect(Y3);
587             MUXenable(ON);
```

```
588             break;
589
590         case 0x15: //A0   Open Hi-Hat
591             //PAD3
592             PADset(3);
593             //Y3
594             Yselect(Y3);
595             MUXenable(ON);
596             break;
597
598         case 0x17: //B0   SNARE2
599             //PAD3
600             PADset(3);
601             //Y4
602             Yselect(Y4);
603             MUXenable(ON);
604             break;
605
606         case 0x16: //A#0    SNARE1
607             //PAD2
608             PADset(2);
609             //Y4
610             Yselect(Y4);
611             MUXenable(ON);
612             break;
613
614         case 0x18: //C1   SNARE3
615             //PAD1
616             PADset(1);
617             //Y4
618             Yselect(Y4);
619             MUXenable(ON);
620             break;
621
622         case 0x19: //C#1    RIDE1
623             //PAD1
624             PADset(1);
625             //Y5
626             Yselect(Y5);
627             MUXenable(ON);
628             break;
629
630         case 0x1A: //D1   RIDE2
631             //PAD2
632             PADset(2);
633             //Y5
634             Yselect(Y5);
635             MUXenable(ON);
636             break;
637
638         case 0x1B: //D#1    CRASH
639             //PAD3
640             PADset(3);
641             //Y5
642             Yselect(Y5);
643             MUXenable(ON);
644             break;
```

```
645
646       case 0x1C: //E1   CABASA1
647           //PAD1
648           PADset(1);
649           //Y6
650           Yselect(Y6);
651           MUXenable(ON);
652           break;
653
654       case 0x1D: //F1   CABASA2
655           //PAD2
656           PADset(2);
657           //Y6
658           Yselect(Y6);
659           MUXenable(ON);
660           break;
661
662       case 0x1E: //F#1    CLAP
663           //PAD3
664           PADset(3);
665           //Y6
666           Yselect(Y6);
667           MUXenable(ON);
668           break;
669
670       case 0x1F: //G1   AGOGO1
671           //PAD1
672           PADset(1);
673           //Y7
674           Yselect(Y7);
675           MUXenable(ON);
676           break;
677
678       case 0x20: //G#1    AGOGO2
679           //PAD2
680           PADset(2);
681           //Y7
682           Yselect(Y7);
683           MUXenable(ON);
684           break;
685
686       case 0x21: //A1   RIM
687           //PAD3
688           PADset(3);
689           //Y7
690           Yselect(Y7);
691           MUXenable(ON);
692           break;
693   }
694   _delay_ms(2);
695   MUXenable(OFF);
696 }
```

B.2 MIDI-handling

MIDI_IN.h

```
/*
 * The software for the x0xb0x is available for use in accordance with the
 * following open source license (MIT License). For more information about
 * OS licensing, please visit -> http://www.opensource.org/
 *
 * For more information about the x0xb0x project, please visit
 * -> http://www.ladyada.net/make/x0xb0x
 *
 *                                      *****
 * Copyright (c) 2005 Limor Fried
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
 * IN THE SOFTWARE.
 *                                      *****
 *
 */


#ifndef MIDI_IN_H_
#define MIDI_IN_H_

// midi channel messages
#define MIDI_IGNORE 0x0  // ignore running status
#define MIDI_NOTE_ON 0x9
#define MIDI_NOTE_OFF 0x8
#define MIDI_PITCHBEND 0xE
#define MIDI_CONTROLLER 0xB

#define MIDI_ALL_NOTES_OFF 123

// system messages
#define MIDI_START 0xFA
#define MIDI_CONTINUE 0xFB
#define MIDI_STOP 0xFC
#define MIDI_CLOCK 0xF8
#define MIDI_SONG_POS_PTR 0xF2  // selects where in the track/pattern to play in
    relevant modes
#define MIDI_SONG_SELECT 0xF3   // selects which track/pattern to play
```

```
53
54 #define MIDISYNC_PPQ 24
55
56 int midi_getch(void);
57 int midi_getchar(void);
58 void do_midi_mode(void);
59
60
61 #endif /* MIDI_IN_H_ */
```

MIDI_IN.c

```
 1 /*
 2  * The software for the x0xb0x is available for use in accordance with the
 3  * following open source license (MIT License). For more information about
 4  * OS licensing, please visit -> http://www.opensource.org/
 5  *
 6  * For more information about the x0xb0x project, please visit
 7  * -> http://www.ladyada.net/make/x0xb0x
 8  *
 9  *                                      *****
10  * Copyright (c) 2005 Limor Fried
11  *
12  * Permission is hereby granted, free of charge, to any person obtaining a
13  * copy of this software and associated documentation files (the "Software"),
14  * to deal in the Software without restriction, including without limitation
15  * the rights to use, copy, modify, merge, publish, distribute, sublicense,
16  * and/or sell copies of the Software, and to permit persons to whom the
17  * Software is furnished to do so, subject to the following conditions:
18  *
19  * The above copyright notice and this permission notice shall be included in
20  * all copies or substantial portions of the Software.
21  *
22  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
23  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
24  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
25  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
26  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
27  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
28  * IN THE SOFTWARE.
29  *                                      *****
30  *
31  */
32
33
34 #include <avr/io.h>
35 #include <avr/interrupt.h>
36 #include <stdio.h>
37 #include "MIDI_IN.h"
38
39 #include <util/delay.h>
40 #define LED PD4 // LED
41
42 #define MIDI_Q_SIZE 32
43 volatile uint8_t midi_q[MIDI_Q_SIZE];      // cyclic queue for midi msgs
44 volatile static uint8_t head_idx = 0;
45 volatile static uint8_t tail_idx = 0;
46
```

```
47 volatile uint8_t c;
48 volatile uint8_t note, velocity;
49
50
51 volatile uint8_t midi_in_addr;   // store this in EEPROM
52 uint8_t midi_running_status = 0;  // suck!
53
54 // interrupt on receive char
55 ISR(USART_RX_vect)    // wird abgearbeitet sobald Interrupt USART_RXC_vect
       stattgefunden hat
56 {
57   uint8_t c = UDR0;
58
59   /*
60   if (c == MIDI_CLOCK)
61   {
62   // raise dinsync clk immediately, and also sched. to drop clock
63     // (MIDISYNC -> DINSYNC conversion);
64     if (sync != DIN_SYNC)
65   {
66       sbi(DINSYNC_PORT, DINSYNC_CLK); // rising edge on note start
67       dinsync_clock_timeout = 5;      // in 5ms drop the edge, is this enough?
68     }
69
70     if (! playing) return;
71     midisync_clocked++;
72     return;
73   }
74 */
75
76   //putstring("0x"); putnum_uh(c); putstring("   ");
77   midi_q[tail_idx++] = c;    // place at end of q
78   tail_idx %= MIDI_Q_SIZE;
79
80
81   if (tail_idx == head_idx) {
82     // i.e. there are too many msgs in the q
83     // drop the oldest msg?
84     head_idx++;
85     head_idx %= MIDI_Q_SIZE;
86   }
87 }
88
89
90 void do_midi_mode(void)
91 {
92   //char c;
93   //uint8_t last_bank;
94   //uint8_t note, velocity;
95
96 /*
97   // turn tempo off!
98   turn_off_tempo();
99
100  // show midi addr on bank leds
101  clear_bank_leds();
102  set_bank_led(midi_in_addr);
```

```
103
104   read_switches();
105   delay_ms(100);
106   read_switches();
107   delay_ms(100);
108   read_switches();
109   last_bank = bank;
110
111   prev_note = 255;        // no notes played yet
112   */
113
114   //while (1) //----> immer nur einmal aufrufen
115   //{
116     /*
117   read_switches();
118     if (function_changed) {
119       midi_notesoff(); // clear any stuck notes
120       return;
121     }
122
123
124     if (last_bank != bank) {
125       // bank knob was changed, change the midi address
126       midi_in_addr = bank;
127
128       // set the new midi address (burn to EEPROM)
129       internal_eeprom_write8(MIDIIN_ADDR_EEADDR, midi_in_addr);
130
131       clear_bank_leds();
132       set_bank_led(midi_in_addr);
133
134       last_bank = bank;
135     }
136   */
137
138
139     // if theres a char waiting in midi queue...
140     if (midi_getch())
141   {
142     PORTD ^= (1<<LED);
143       // if its a command & either for our address or 0xF,
144       // set the midi_running_status
145       c = midi_getchar();
146
147       if (c >> 7)
148     {       // if the top bit is high, this is a command
149     if ((c >> 4 == 0xF) ||    // universal cmd, no addressing
150       ((c & 0xF) == midi_in_addr)) // matches our addr
151     {
152       midi_running_status = c >> 4;
153     }
154     else
155     {
156       // not for us, continue!
157       midi_running_status = MIDI_IGNORE;
158       //continue;
159     }
```

```
160       }
161
162       switch (midi_running_status)
163     {
164     case MIDI_IGNORE:
165     {
166       // somebody else's data, ignore
167       break;
168     }
169
170     case MIDI_NOTE_ON:
171     {
172       if (c >> 7)  // if the last byte was a command then we have to get the note
173       note = midi_getchar();
174       else
175       note = c;  // otherwise, this was a running status, and c is the note
176
177       velocity = midi_getchar();
178     /*
179       putstring("MIDI note on (note "); putnum_ud(note);
180       putstring(") (velocity "); putnum_ud(velocity);
181       putstring(")\n\r");
182     */
183
184       //if(velocity != 0x00) PORTD ^= (1<<LED); // Toggle LED
185       if(velocity != 0x00)
186       {
187         MUX4051(note, velocity);  // Uebergebe nur echten NOTE_ON
188       }
189
190       break;
191     }
192
193     case MIDI_NOTE_OFF:
194     {
195       if (c >> 7)
196       note = midi_getchar();
197       else
198       note = c;
199
200       velocity = midi_getchar();
201     /*
202       putstring("MIDI note off (note "); putnum_ud(note);
203       putstring(") (velocity "); putnum_ud(velocity);
204       putstring(")\n\r");
205     */
206
207       //midi_note_off(note, velocity);
208
209       break;
210     }
211
212     case MIDI_PITCHBEND:
213     {
214       //putstring("MIDI Slide\n\r");
215
216       break;
```

```
217     }
218
219     default:
220     /*putstring("Received Unknown MIDI: 0x"); putnum_uh(c);
221     putstring("\n\r"); */
222     break;
223       } // end of switch
224     }
225   }
226 //}
227   // midi handling code!
228
229
230 int midi_getch(void) // checks if there is a character waiting!
231 {
232   if (head_idx != tail_idx)
233     return 1;
234   return 0;
235 }
236
237
238 int midi_getchar(void)
239 {
240   uint8_t c;
241
242   while (head_idx == tail_idx);
243
244   cli();
245   c = midi_q[head_idx++];
246   head_idx %= MIDI_Q_SIZE;
247   sei();
248
249   return c;
250 }
```

B.3 internes EEPROM

int_eeprom.h

```
 1 /*
 2  * The software for the x0xb0x is available for use in accordance with the
 3  * following open source license (MIT License). For more information about
 4  * OS licensing, please visit -> http://www.opensource.org/
 5  *
 6  * For more information about the x0xb0x project, please visit
 7  * -> http://www.ladyada.net/make/x0xb0x
 8  *
 9  *                                     *****
10  * Copyright (c) 2005 Limor Fried
11  *
12  * Permission is hereby granted, free of charge, to any person obtaining a
13  * copy of this software and associated documentation files (the "Software"),
14  * to deal in the Software without restriction, including without limitation
15  * the rights to use, copy, modify, merge, publish, distribute, sublicense,
16  * and/or sell copies of the Software, and to permit persons to whom the
17  * Software is furnished to do so, subject to the following conditions:
```

```
18  *
19  * The above copyright notice and this permission notice shall be included in
20  * all copies or substantial portions of the Software.
21  *
22  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
23  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
24  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
25  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
26  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
27  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
28  * IN THE SOFTWARE.
29  *                                      *****
30  *
31  */
32
33
34 #ifndef INT_EEPROM_H_
35 #define INT_EEPROM_H_
36
37
38
39 #ifndef sbi
40 #define sbi(p,b) (p) |= (1<<(b))
41 #endif
42
43 #ifndef cbi
44 #define cbi(p,b) (p) &= ~(1<<(b))
45 #endif
46
47
48 uint8_t internal_eeprom_read8(uint16_t addr);
49 void internal_eeprom_write8(uint16_t addr, uint8_t data);
50
51
52
53 #endif /* INT_EEPROM_H_ */
```

int_eeprom.c

```
1  /*
2   * The software for the x0xb0x is available for use in accordance with the
3   * following open source license (MIT License). For more information about
4   * OS licensing, please visit -> http://www.opensource.org/
5   *
6   * For more information about the x0xb0x project, please visit
7   * -> http://www.ladyada.net/make/x0xb0x
8   *
9   *                                      *****
10  * Copyright (c) 2005 Limor Fried
11  *
12  * Permission is hereby granted, free of charge, to any person obtaining a
13  * copy of this software and associated documentation files (the "Software"),
14  * to deal in the Software without restriction, including without limitation
15  * the rights to use, copy, modify, merge, publish, distribute, sublicense,
16  * and/or sell copies of the Software, and to permit persons to whom the
17  * Software is furnished to do so, subject to the following conditions:
18  *
19  * The above copyright notice and this permission notice shall be included in
```

```
20  * all copies or substantial portions of the Software.
21  *
22  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
23  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
24  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
25  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
26  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
27  * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
28  * IN THE SOFTWARE.
29  *                                      *****
30  *
31  */
32
33
34 #include <avr/io.h>
35 #include <avr/interrupt.h>
36 #include <stdio.h>
37 #include "int_eeprom.h"
38 #include "MIDI_IN.h"
39
40
41
42 //*************************************************
43 //         Internal EEPROM
44 //*************************************************
45
46 uint8_t internal_eeprom_read8(uint16_t addr) {
47   // EEWE -> EEPE
48   loop_until_bit_is_clear(EECR, EEPE); // wait for last write to finish
49   EEAR = addr;
50   sbi(EECR, EERE);        // start EEPROM read
51   return EEDR;            // takes only 1 cycle
52 }
53
54 void internal_eeprom_write8(uint16_t addr, uint8_t data) {
55   //printf("writing %d to addr 0x%x...", data, addr);
56   // EEWE -> EEPE
57   loop_until_bit_is_clear(EECR, EEPE); // wait for last write to finish
58   EEAR = addr;
59   EEDR = data;
60   cli();                // turn off interrupts
61   // EEMWE -> EEMPE
62   sbi(EECR, EEMPE);     // these instructions must happen within 4 cycles
63   // EEWE -> EEPE
64   sbi(EECR, EEPE);
65   sei();                // turn on interrupts again
66   //putstring("done\n\r");
67 }
```