Technische Universität Ilmenau
Fakultät für Elektrotechnik und Informationstechnik
Fakultät für Informatik und Automatisierung

Medienprojekt

Universelle Bedienoberfläche
für MIDI-Synthesizer:
Hardware und Microcontroller-Code

vorgelegt von: Holger Prang
Marcel Richter
Christian Stöcklmeier

Verantwortlicher Professor:
Prof. Dr.-Ing. Karlheinz Brandenburg
Dr.-Ing. Heinz-Dietrich Wuttke

Betreuender wiss. Mitarbeiter:
Dipl.-Ing. Ulrich Reiter

Beginn der Arbeit: 1.4.2006

Ende der Arbeit: 30.9.2006

Ilmenau, den 2. Oktober 2006

Danksagung

Wir möchten Herrn Ulrich Reiter dafür danken, dass er uns während des gesamten Medienprojektes mit Rat und Tat zur Seite stand. Er hatte immer ein offenes Ohr für unsere Probleme, und motivierte uns stets durch seine gelassene, aber bestimmte Art.

Inhaltsverzeichnis

1 Einleitung

Als in den späten 80er und frühen 90er Jahren die digitale Klangerzeugung aufkam, führte dies dazu, dass bei sehr vielen Synthesizern die Bedienoberflächen minimiert wurden. Waren früher sehr viele Bedienelemente auf der Oberfläche analoger Geräte angeordnet, hat man diese zu Beginn des Digitalzeitalters wohl aus Kostengründen wegrationalisiert. Dies wurde durch die Digitaltechnik erst möglich gemacht, da man nun komplexe Menüs realisieren konnte, die mit wenigen Knöpfen bedienbar waren. Die Bedienelemente der Geräte wurden auf sehr kleine, teilweise nur numerische Displays und einige wenige Taster reduziert und die bis zu tausend Parameter pro Gerät wurden in vielschichtigen Menüs untergebracht, was die Bedienung erheblich erschwerte. Für einen typisches Beispiel eines solchen Synthesizers siehe Abbildung 1.1. Die Geräte wurden umständlich zu handhaben und der Vorgang der Erstellung

Abbildung 1.1: Oberheim Matrix 1000: Typisches Beispiel minimierter Bedienoberflächen

und Editierung komplexer Klangprogramme gestaltete sich sogar für geübte Benutzer sehr zeitaufwendig. Da viele dieser Geräte jedoch auch für heutige Maßstäbe noch interessante und hochwertige Klänge hervorbringen, wäre eine einfache und intuitive Bedienoberfläche wünschenswert, um diese Synthesizer wieder mit angemessenem Aufwand programmieren und benutzen zu können. Leider eignen sich die zur Zeit kommerziell verfügbaren MIDI Controller nicht für diese Aufgabe, da die standardisierte Einbindung von MIDI zur Entstehungszeit dieser Klangerzeuger noch nicht üblich war. Jeder Hersteller verwendete sein eigenes Format von MIDI Systemexklusiven Daten (kurz: SysEx), um Klangprogramme zu speichern, zu editieren und zu übertragen und die aktuellen MIDI Controller bieten nicht den Grad an Programmierbarkeit, um SysEx Funktionen im erforderlichen Umfang einzubinden.

Dieses Medienprojekt hat zum Ziel, solch einen frei programmierbaren MIDI Controller in Hardware zu realisieren, der optimal an die zu steuernden Synthesizer angepasst werden kann. Zu diesem Zweck wird eng mit einem zweiten Medienprojekt zusammen gearbeitet, das eine Java Software zur Programmierung des Controllers erstellt [3].

2 Konzept

Zur grundsätzlichen Funktionsweise des Gerätes ist zu sagen, dass der MIDI Controller zuerst über ein MIDI Kabel mit einem Rechner verbunden werden muss (siehe Abbildung 2.2), auf dem die Java Software läuft. Mittels dieser Software kann der Benutzer den Controller auf seine Synthesizer programmieren, indem er die Einstellungen, die in der Software vorgenommen wurden, als SysEx Daten per MIDI auf den Controller überträgt. Anschließend wird der PC nicht mehr benötigt; der Controller

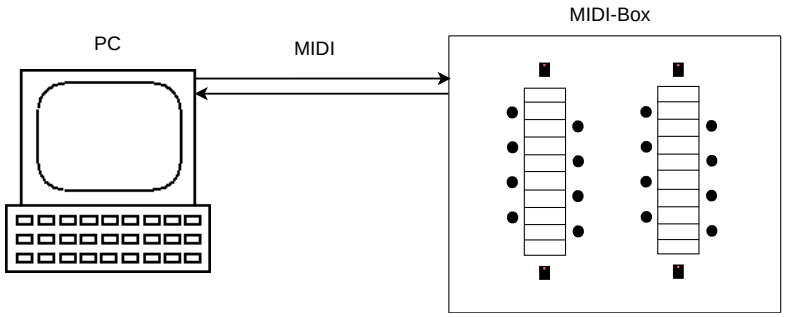

Abbildung 2.2: Verbindung zwischen PC und MIDIbox

wird per MIDI mit dem entsprechenden Synthesizer verbunden (siehe Abbildung 2.3) und kann dann durch die vorangegangene Programmierung die Funktionen des Gerätes steuern. Die Befehle des Keyboards, das auch an den Controller angeschlossen ist, werden vom Controller nur zum Synthesizer getunnelt, um ihn zur Klangerzeugung anzuregen. Ziel des Aufbaus des Controllers ist ein möglichst einfaches

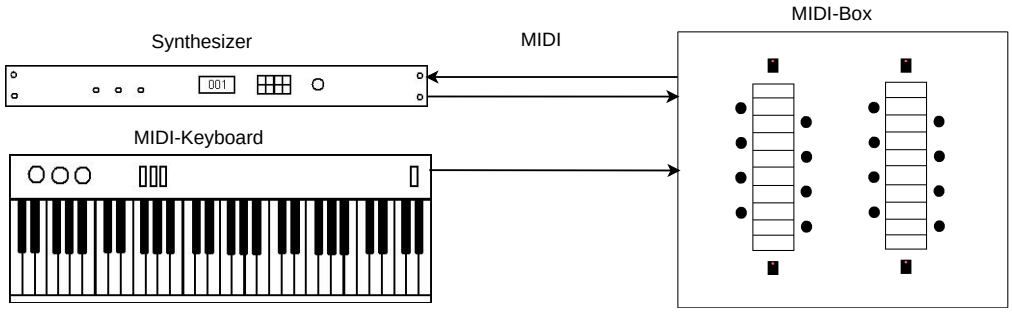

Abbildung 2.3: Verbindung zwischen Synthesizer, Keyboard und MIDIbox

und universelles Design. Der Aufbau der Hardware soll so auf das Bedienkonzept abgestimmt sein, dass eine intuitive Bedienung des Gerätes möglich wird. Es sollen alle Funktionen und Parameter der einzelnen Synthesizer eingebunden werden, ohne jedoch Übersichtlichkeit und schnelle Navigation zu kompromittieren. Es folgt eine Abbildung des fertigen Controllers (Abb. 2.4), die in den folgenden Abschnitten näher erläutert wird.

Abbildung 2.4: Fertiger Controller

2.1 Aufbau der Bedienoberfläche

Das zentrale Element des MIDI Controllers stellen zwei 240x64 Pixel große, grafische Displays (GLCDs) dar, die stets eine funktionale Einheit bilden und mit d0 und d1 bezeichnet sind. Diese sind parallel zueinander vertikal angeordnet, und werden zur Anzeige von Menüs, Parameternamen und Parameterwerten eingesetzt. Die Displays verhalfen dem Controller auch zu seiner Bezeichnung MIDIbox GLCD. Ober- und unterhalb jedes Displays befindet sich zentriert zum Display je ein rechteckiger Knopf, also insgesamt vier, die folgend mit k0 bis k3 bezeichnet werden und der Menünavigation dienen. Alle vier dieser Knöpfe beinhalten LEDs, um ein zusätzliches Feedback für den Ort in der Menüstruktur zu geben, an dem sich der User befindet. Außerdem sind insgesamt 16 Dreh-Encoder (Endlosregler e0 bis e15) mit zusätzlicher Tasterfunktion um die Displays arrangiert, und zwar an jeder vertikalen Kante eines Displays vier. Diese Vierergruppen sind (pro Display) in der Höhe ein Stück gegeneinander verschoben, so dass sie ineinander greifen und so jedem Encoder zwei Zeilen auf einem Display zugeordnet sind (siehe Abbildung 2.5).

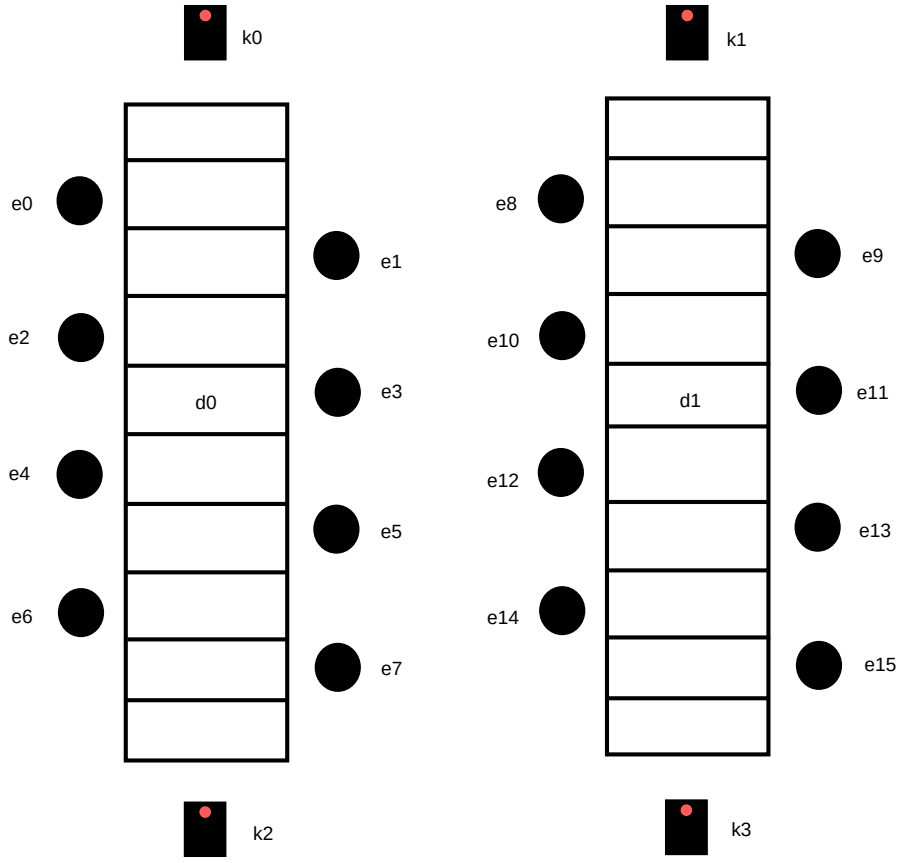

Abbildung 2.5: Arrangement MIDI Controller

2.2 Aufbau der Menüstruktur

Die Bedienung des Geräts ist in drei Menüebenen aufgeteilt (siehe Abbildung 2.6). Die oberste Ebene ist die Device Ebene, in der der zu steuernde Synthesizer ausgewählt werden kann. Nach Auswahl des Synthesizers kommt man automatisch in die Modul Ebene. Hier sieht man einzelne Module, die die verschiedenen Funktionsgruppen des Synthesizers darstellen (z.B. Oszillator, Filter, LFO...), und Parameter zusammenfassen. Pro Device gibt es vier Bänke von Modulen. Wählt man ein Modul aus, gelangt man in die Parameterebene, in der nun die einzelnen Parameter des Synthesizers manipuliert werden können.

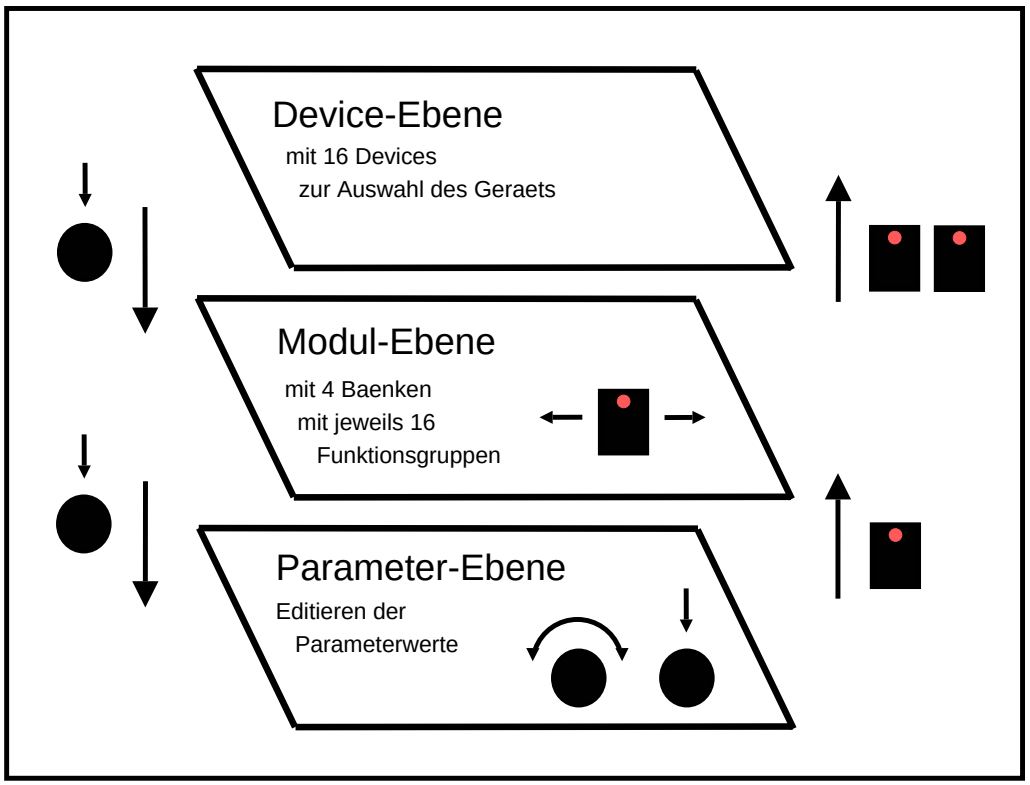

Abbildung 2.6: Die Menü Ebenen mit Navigationsanleitung

2.3 Bedienung

Die Bedienung des Gerätes umfasst die Navigation durch die Menüs und das Einstellen der Parameterwerte (siehe Abbildung 2.7). Nach dem Einschalten startet

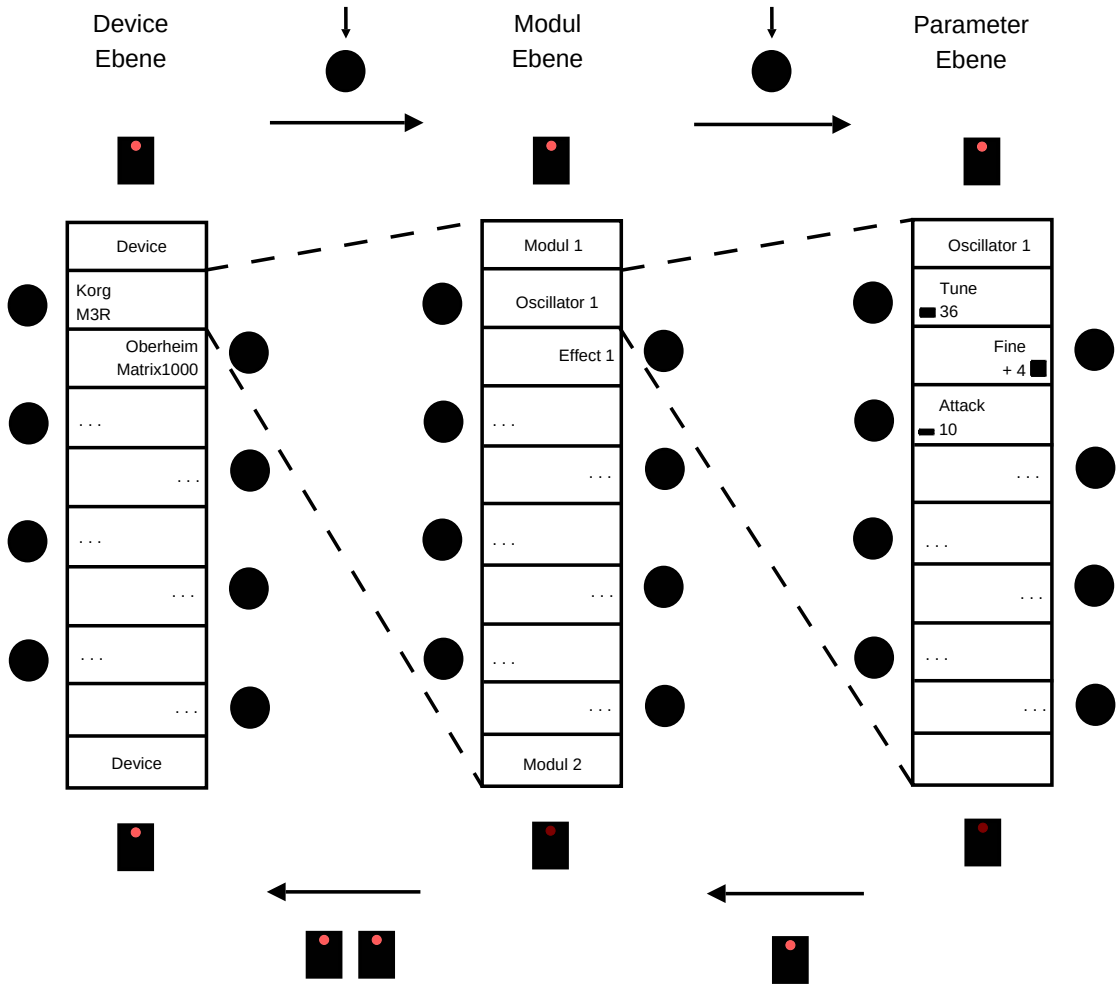

Abbildung 2.7: Menünavigation anhand eines Displays

das Gerät in der Device Ebene, in der die Namen der zuvor schon eingespeicherten Synthesizer (maximal sechzehn) neben den Endlos-Encodern angeordnet sind. Hier leuchten alle vier Menü LEDs. Durch Drücken eines Encoders wird der entsprechende Synthesizer ausgewählt, und die Modul Ebene aufgerufen. Hier sind die Module wie eben auch schon die Synthesizer neben den Dreh Encodern angeordnet und können durch Drücken des jeweiligen Encoders ausgewählt werden. An dieser Stelle muss angeführt werden, dass es pro Device nicht nur eine Seite von Modulen gibt, sondern vier Seiten a sechzehn Module. Diese Seiten (im folgenden Bänke genannt) können wie in Abbildung 2.8 durch die Menütaster k0-k3 umgeschaltet werden. In der Modulebene leuchtet jeweils die LED, die zur aktuellen Bank gehört.

Das Auswählen eines Moduls durch Drücken eines Encoders ruft dann die Parameter Ebene auf, in der nun die Parameter neben den Encodern zu sehen sind. In der oberen Zeile eines jeden Encoders findet sich der Name des Parameters und in der

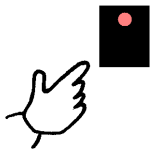

Abbildung 2.8: Drücken eines Menüknopfes

unteren der gerade eingestellte Parameterwert. Die Darstellung des Wertes variiert je nach der Bit Auflösung des Parameters, oder wenn eine textliche Beschreibung der Werte eingestellt ist. Die Parameterwerte kann der User durch Drehen (oder Drücken, je nach Funktion) der Encoder nun manipulieren, und die gewünschten Änderungen am Klangprogramm vornehmen.

Die Navigation rückwärts durch das Menü erfolgt nur über die rechteckigen Menü Taster k0-k3. Um von irgendeiner Ebene zurück zur Device Ebene zu gelangen, drückt man die oberen beiden Taster (k0 und k1) gleichzeitig (siehe Abbildung 2.9).

Ansonsten gelangt man durch Drücken eines Tasters k0-k3 sowohl aus der Parameter Ebene als auch innerhalb der Menü Ebene zu den entsprechenden Bänken der Menü Ebene.

Abbildung 2.9: Gleichzeitiges Drücken der oberen Menüknöpfe

3 Hardware

Zur Realisierung des Hardware Konzepts wurde auf das Open Source Projekt „ucapps“ [1] aufgebaut, da hier schon fertige modulare Hardwarelösungen für MIDI Controller zu finden sind. Außerdem gibt es ein Betriebssystem für den verwendeten PIC Microcontroller, das MIOS, das dem Entwickler eine Sammlung von Funktionen zur Programmierung des PIC an die Hand gibt. Diese ersparen dem Programmierer dann größtenteils die Auseinandersetzung mit den zeitkritischen Aspekten einer MIDI Anwendung und die Low Level Programmierung des PICs.

3.1 ucapps

Das Open Source Projekt „ucapps“ bietet einen modularen Ansatz für sämtliche Hardware MIDI Anwendungen mit dem Namen MBHP, MIDIbox Hardware Plattform. Für jede Funktionsgruppe eines MIDI Geräts steht eine separate Platine (Modul genannt) zur Verfügung, die der User je nach gewünschten Funktionen seines Gerätes frei zusammenstellen kann. Das Herzstück ist hier ein PIC der Firma Microchip, der auf dem Core Modul sitzt. Darüber hinaus gibt es eben das MIOS Betriebssystem für den PIC, das eine Vielzahl an nützlichen Funktionen für die Programmierung mitbringt. Die Programmierung ist seit kurzer Zeit durch einen Wrapper auch in C möglich, was ein Projekt unserer Komplexität erst realistisch gemacht hat. Außerdem werden auf der Seite schon fertige Hardware Controller (MIDIboxes) vorgestellt, die aus diesen Modulen zusammengestellt wurden und mit fertiger Anwendungssoftware den Besuchern der Seite zum Nachbau oder als Anwendungs- und Programmierbeispiele zur Verfügung stehen. Es finden sich auch ein Forum und ein Wiki auf der Seite, die umfassenden Support sicherstellen. Teil des Medienprojektes ist es, unsere Ausarbeitungen abschließend hier zu veröffentlichen.

3.2 Module - ucapps

Von den zur Verfügung stehenden MBHP Modulen verwendeten wir zwei Core Module, zwei Digital In Module für alle Encoder und Taster, zwei Digital Out Module für die 4 LEDs der Menütaster und die MIDI Transfer LEDs und zwei BankStick Platinen (12 BankSticks) mit insgesamt 768kB Kapazität zur Speicherung der Menüs, Parameter und SysEx Strings. Darüber hinaus fanden ein MIDI Merger für einen zusätzlichen MIDI Input und ein IIC Modul als alternativer MIDI Out Verwendung. Die Module werden hier nicht im Detail besprochen, da sie auf der ucapps Internetseite eingehend dokumentiert sind. Wir werden in dieser Arbeit schwerpunktmäßig aufzeigen, wie die Module speziell in unserem Controller Anwendung fanden. Zur Verkabelung der Module siehe Abschnitt 3.4.1.

3.2.1 Core Modul

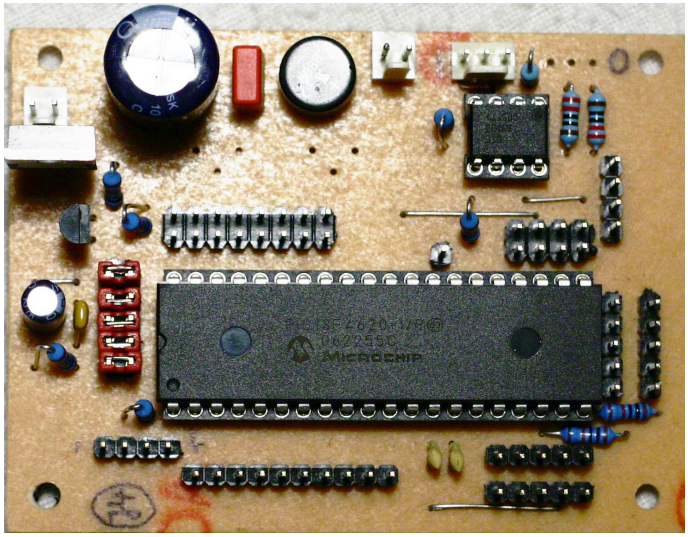

Abbildung 3.10: Core Modul

Das Core Modul (siehe Abbildung 3.10) ist das Herzstück einer jeden MIDIbox. Auf ihm findet sich der Hauptprozessor, der PIC, der den Programmcode beinhaltet und für sämtliche Abläufe zuständig ist. Hier muss die Stromversorgung angeschlossen werden und es finden sich ein MIDI In und ein MIDI Out Port. Das Display und sämtliche MBHP Module werden an das Core Modul angeschlossen. Wir verwenden zwei Stück davon (Core 0 und Core 1), die über MIDI verbunden sind, um unsere zwei Displays ansteuern zu können. Das erste übernimmt alle Ein- und Ausgaben und tunnelt die Information für das zweite Display per SysEx über die MIDI Schnittstelle an das zweite Core Modul. Dieses verwertet nur die für sich bestimmten SysEx Nachrichten und gibt alle anderen an den Out Port der MIDIbox GLCD weiter (siehe Abbildung 3.11).

Da unser Programmcode die Grenzen des von der ucapps regulär verwendeten PIC18F452 gegen Ende des Projekts bei weitem sprengte, entschieden wir uns, auf einen alternativen PIC, den 18F4620 umzusteigen. Dieser PIC hat den Vorteil, dass er doppelt soviel Flash Speicher für Programmcode (64k statt 32k) und auch mehr RAM (ca. 4k statt ca. 1,5k) zur Verfügung hat. Bei der aktuellen Version dieses PICs gibt es jedoch einen Fehler in der Herstellung, so dass er manchmal Probleme beim Versenden von MIDI Nachrichten bereitet (es werden zufällig zusätzliche Nullen in die MIDI Nachricht eingebaut). Für diesen Fehler mussten wir noch einen Workaround bauen, der auch auf der ucapps vorgeschlagen wurde: es wird eine zusätzliche IIC MIDI Platine eingesetzt, die an den PIC angeschlossen wird, und dann den MIDI Ausgang übernimmt (siehe 3.2.6, IIC MIDI Modul). In unserem Fall sitzt jetzt auf dem Core 0 ein PIC18F4620, auf dem Core 1 jedoch immer noch ein PIC18F452. Auf dem Core 1 benötigten wir keinen größeren Speicher, da dort nur die Displayanzeige und ein Tunnel für MIDI Nachrichten laufen.

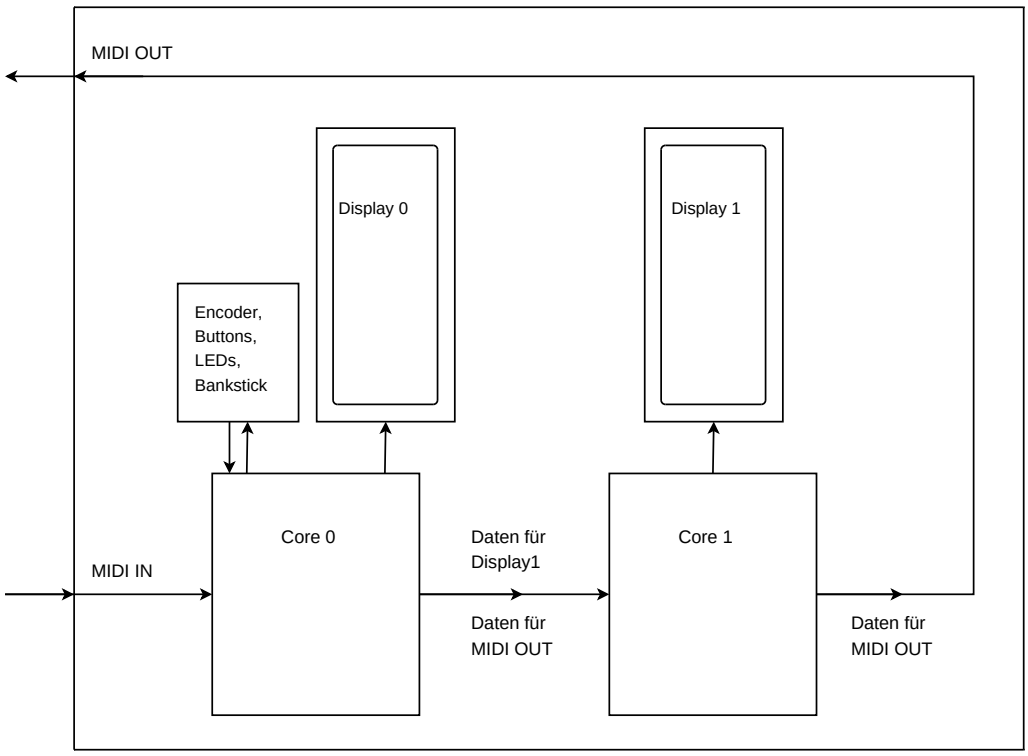

Abbildung 3.11: Beide Cores mit Displays

3.2.2 Digital In Modul

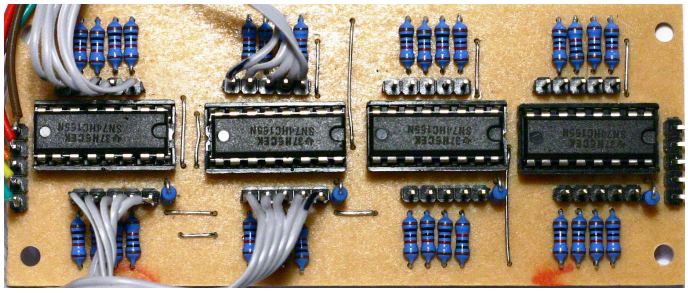

Abbildung 3.12: Digital In Modul

Alle digitalen Eingabeelemente sind an das Digital In (DIN) Modul angeschlossen (siehe Abbildung 3.12). Hierzu zählen bei unserem Gerät die vier rechteckigen Menü Taster, alle 16 Dreh Encoder (deren Drehfunktion wird intern auch durch Schalter registriert) und auch die Tasterfunktion aller Encoder. In unserer MIDIbox benötigten wir wegen der hohen Zahl an Bedienelementen zwei DIN Module, eines für die Drehfunktion aller Encoder und eines für alle Taster. Vier Encoder auf jeder Displayseite sind auf einer Platine zusammengefasst, um eine Befestigung an der Frontplatte zu ermöglichen (siehe Abbildung 3.13). Die Encoder Gruppen sind fortlaufend am DIN Modul angeschlossen; sprich Encoder 0 bis 3 sind die vier ganz links auf der Bedienoberfläche, Nummer 4 bis 7 sind die Encoder rechts vom linken Display etc. Dies steht jedoch der logischen Nummerierung, wie sie auch bei uns im Programm verwendet wird, entgegen, in der die Nummerierung zwar auch von oben

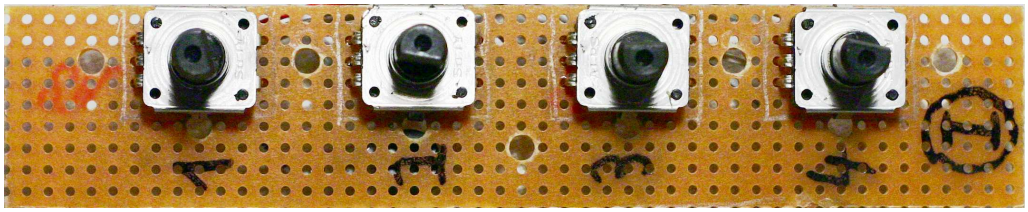

Abbildung 3.13: Encoderplatine

nach unten erfolgt, aber alle 8 Encoder eines Displays einbezieht (siehe Abbildung 3.14). Zu diesem Zweck gibt es in den Funkionen, die auf Eingaben reagieren, eine

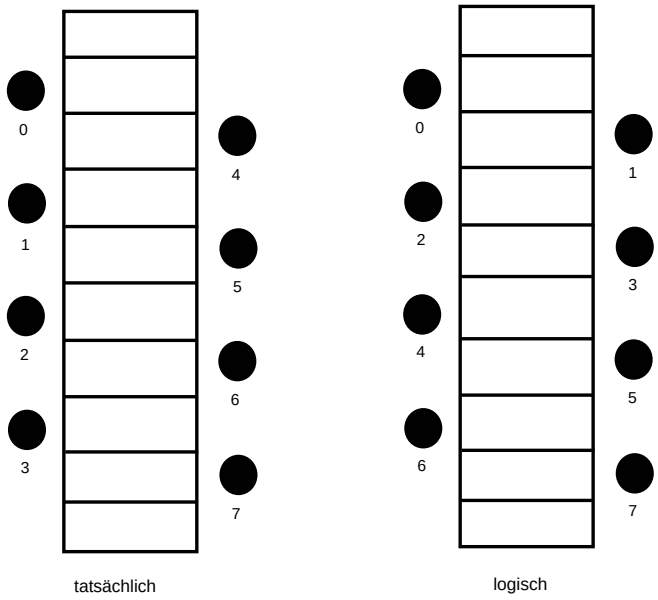

Abbildung 3.14: Tatsächliche und logische Nummerierung der Encoder anhand des linken Displays

Umsetzung von der tatsächlichen zur logischen Nummerierung (siehe hierzu Kapitel 4.6).

3.2.3 Digital Out Modul

An das Digital Out (DOUT) Modul sind alle digitalen Ausgabeelemente (außer den Displays) angeschlossen (siehe Abbildung 3.15). Das DOUT Modul in unserer MIDIbox (siehe Abbildung 3.15) steuert die vier LEDs an, die in den Menütastern sitzen(Pins 0-3). Außerdem ist an Pin 7 der Umschalter für die beiden BankStick Platinen angeschlossen (siehe Abschnitt 3.3.4).

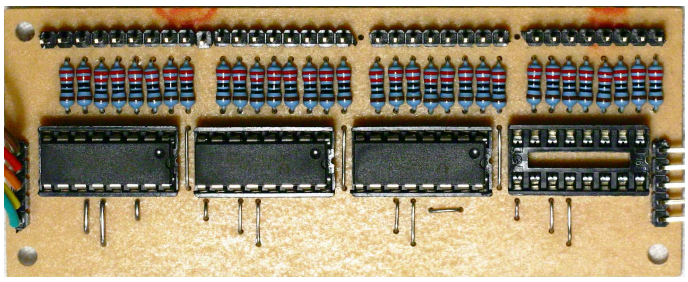

Abbildung 3.15: DOUT Modul Core 0

3.2.4 BankStick

Der BankStick (siehe Abbildung 3.16) stellt den universellen Datenspeicher für die MBHP MIDIboxen dar. In der MIDIbox GLCD dient er zur Speicherung der Device-, Modul- und Parameterdaten, die durch die Java Editor Software (siehe [3]) erstellt und vom PC per MIDI auf die MIDIbox übertragen werden. Bis zu acht BankSticks mit je 64 kB (also 512kB) können an einem Core Modul betrieben werden. Diese Speicherkapazität haben wir mit den vielfältigen Menüeinträgen und insgesamt 16384 Parametern vollständig ausgeschöpft. Um für jeden Menüeintrag und jeden Parameter noch zusätzliche SysEx Strings speichern zu können, mussten wir selbst noch 4 BankSticks (mit insgesamt 256kB) an das Core Modul anschließen. Zwischen beiden BankStick Platinen schaltet ein integrierter elektronischer Schalter, der über den Pin 7 des DOUTs am Core 0 geschaltet wird. Die Vorgehensweise wird im Kapitel 3.3.4 genau erläutert.

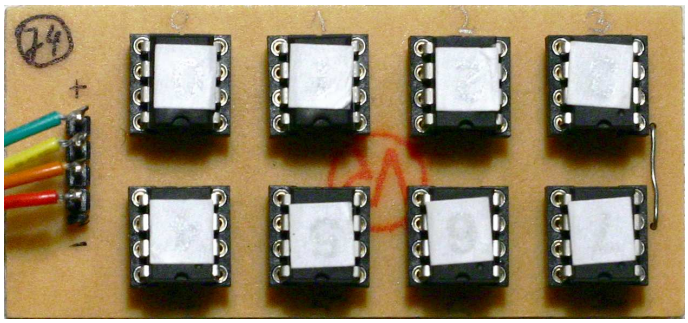

Abbildung 3.16: Ucapps BankStick Platine

3.2.5 MIDI Merger

Der MIDI Merger (siehe Abbildung 3.17) ist ein Modul, das lediglich zwei MIDI Eingänge zusammenführt und an einen Ausgang leitet. In der MIDIbox GLCD wird der Merger benötigt, da sowohl der Synthesizer als auch ein MIDI Keyboard (Sequenzer etc...) daran angeschlossen werden muss (siehe Abbildung 3.18), aber das Core Modul nur einen MIDI Eingang hat. Der Synthesizer muss im laufenden Betrieb häufig seinen Editbuffer (sein Klangprogramm) zum MIDI Controller schicken, um Synchronität zwischen den Geräten zu gewährleisten und das Keyboard wird da-

zu benötigt, den Synthesizer überhaupt zur Klangerzeugung anzuregen. Die MIDI Nachrichten des Keyboards werden im Controller nicht ausgewertet, sondern nur zum Synthesizer durchgetunnelt.

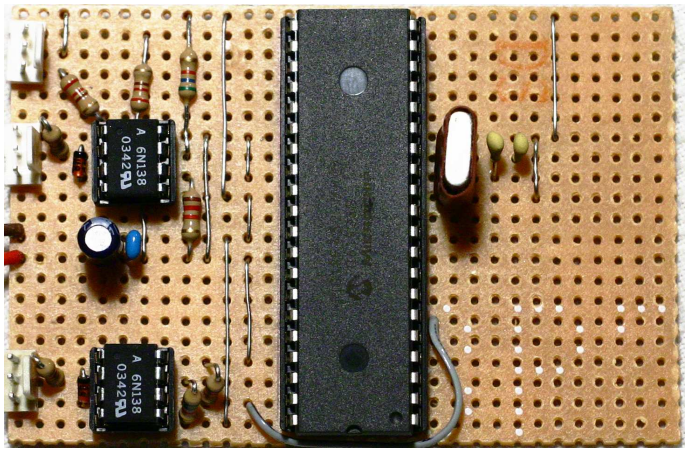

Abbildung 3.17: MIDI Merger

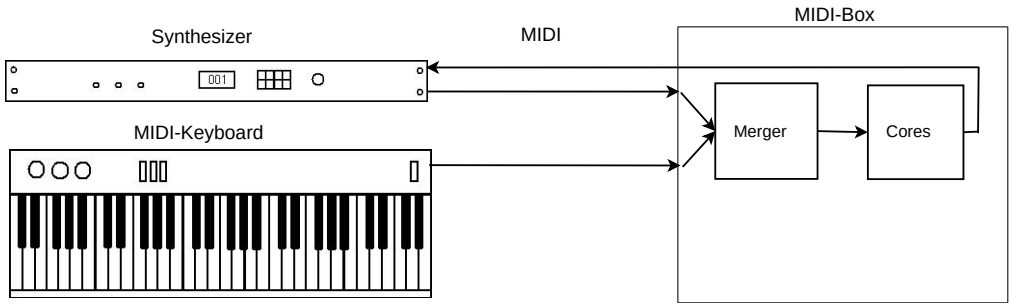

Abbildung 3.18: Verbindung zwischen Synthesizer, Keyboard und MIDIbox mit MIDI Merger

3.2.6 IIC MIDI Modul

Allgemein wird das IIC MIDI Modul (siehe Abbildung 3.19) verwendet, um das Core Modul den IIC Bus als MIDI Interface nutzen zu lassen. Es können problemlos bis zu vier IIC Module an einen Core angeschlossen werden, die dann über eigene ID´s angesprochen werden, und jedes einen MIDI In und einen MIDI Out zur Verfügung stellen. Wir wollen das IIC MIDI Modul jedoch nur als Workaround für den MIDI Out Bug des PIC18F4620 einsetzen (siehe Abschnitt 3.2.1) und benötigen somit auch nur einen MIDI Ausgang, da nur der Ausgang am PIC fehlerbehaftet ist. Für diesen Zweck gibt es eine abgespeckte Version des IIC MIDI Moduls, die „Out only“ Version, die bei uns Verwendung fand. Das Modul wird an den IIC Bus des Core Moduls angeschlossen (J4, parallel zur BankStick Platine). Nun muss man noch die ID 10 (bei der „Out only“ Fassung fest verdrahtet) für das IIC MIDI Modul an einer speziellen Stelle im Header des PICs einstellen (z.B. mit der ucapps Anwendung change_id) und schon werden automatisch alle ausgehenden MIDI Nachrichten nicht

mehr über die interne MIDI Schnittstelle sondern über das IIC Modul versendet (siehe Abbildung 3.20). Hierbei wird auch keinerlei zusätzliche Anpassung des C Codes benötigt.

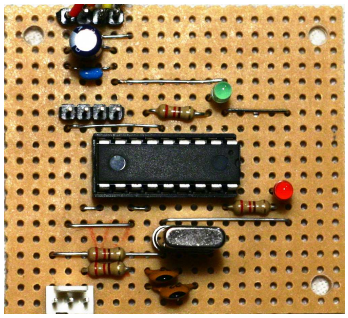

Abbildung 3.19: IIC MIDI Modul

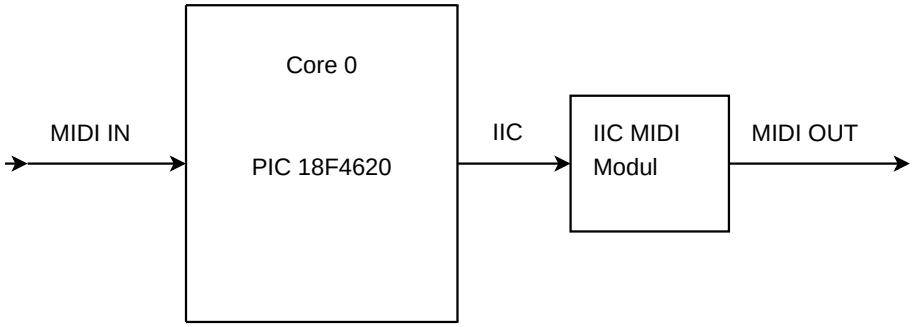

Abbildung 3.20: Core 0 mit IIC MIDI Modul

3.2.7 LTC Modul

Zur Ansteuerung der MIDI In/Out Leds mussten wir noch eine reduzierte Version des von der ucapps verwendeten LTC Moduls nachbauen (siehe Abbildung 3.21). Hierzu haben wir den Teil des Schaltplans verwendet, in dem die MIDI Leds vom Logik Baustein 74HC00 angesteuert werden. Unser LTC Modul ist mit dem Anschluss J11 beider Cores verbunden. Vom Core 0 bezieht es die Leitung für die MIDI In Led und vom Core 1 die Spannungsversorgung sowie die Leitung für die MIDI Out Led. Diese Verkabelung wurde gewählt, da die LEDs genau die MIDI Aktivitäten anzeigen sollen, die an den Anschlüssen nach außen anliegen.

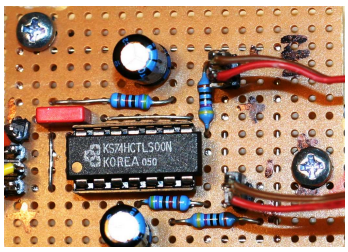

Abbildung 3.21: Reduziertes LTC Modul

3.3 Module - Eigenbau

Zusätzlich zu den verwendeten ucapps Modulen haben wir noch einige kleine Schaltungen selbst aufbauen müssen. Hierzu gehören die Stromversorgung für die Display-Hintergrundbeleuchtungen, die gemeinsame Kontrastregelung der Displays, zwei kleine Adapterplatinen für die Pinumsetzung zum Anschluss der Displays und eine Platine für die Speichererweiterung der BankSticks samt elektronischem Umschalter.

3.3.1 Display Beleuchtung

Die grafischen, großen Displays (GLCDs) die bei unserer MIDI Box zum Einsatz kamen, benötigen zusammen etwa 400mA (typischer Wert, gemessen) für eine angenehme Hintergrundbeleuchtung. Diesen Strom kann das Core Modul nicht so ohne weiteres am Ausgang J2 zur Verfügung stellen, da sich der Festspannungsregler 7805 dann sehr stark erhitzt. Auch auf der ucapps wird davon abgeraten, die Display Hintergrundbeleuchtungen über diesen Port zu versorgen, da der gesamte Strom, den der 7805 liefern muss, 500mA nicht überschreiten soll. Also haben wir eine separate Platine mit einem zusätzlichen 7805 aufgebaut, die nur für die Stromversorgung der Display Hintergrundbeleuchtung zuständig ist (siehe Abbildungen 3.22 und 3.23. Zu diesem Zweck haben wir den 7805 mit ähnlichen Kondensatoren wie auf dem Core Modul Schaltplan beschaltet und an die Hauptspannungsversorgung angeschlossen (9V, siehe hierzu Abschnitt 3.4.2). Somit stellt der Spannungsregler am Ausgang die 5 Volt für die Hintergrundbeleuchtung zur Verfügung. Zur Begrenzung des Stroms ist ein Widerstand von ca. 3,4 Ohm (realisiert durch Parallelschaltung von 3 mal 10 Ohm) eingebaut. Außerdem finden sich Anschlüsse für einen Schalter, der außen an der Rückseite der MIDI Box angebracht ist und der Aktivierung / Deaktivierung der Hintergrundbeleuchtung dient.

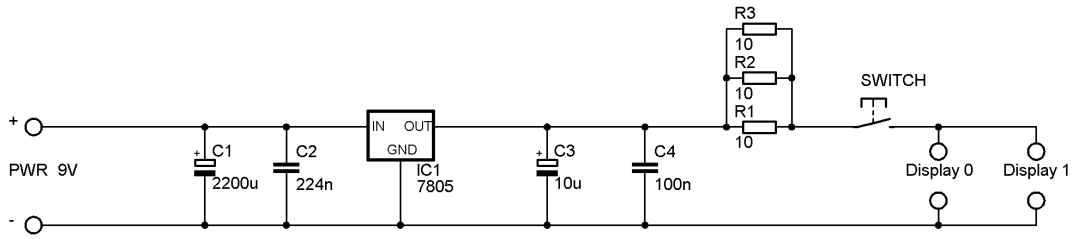

Abbildung 3.22: Schaltplan der Display Beleuchtung

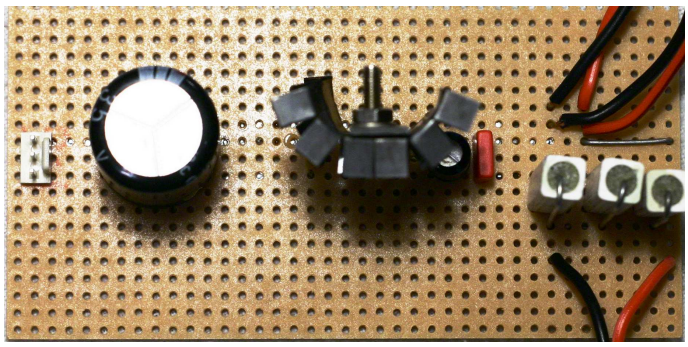

Abbildung 3.23: Platine der Display Beleuchtung

3.3.2 Display Kontrastregelung

Jedes Display wird von dem dazugehörigen Core Modul (siehe Abschnitt 3.2.1) mit Spannung versorgt. Zur Kontrasteinstellung wird eine negative Spannung benötigt die von den Displays selbst erzeugt wird. Die negative Spannung von Display 0 speist die Kontrastreglung für beide Displays. Über eine Leitung werden -9V (siehe Abbildung 3.29) an die Kontrastplatine (siehe Abbildung 3.24 und 3.25) geleitet. Über ein Potentiometer wird die Spannung für beide Displays eingestellt. Diese Spannung geht direkt zurück zu Display 0. Über einen Trimmer kann die Spannung für Display 1 angepasst werden, um Kontrastunterschiede zwischen beiden Displays auszugleichen.

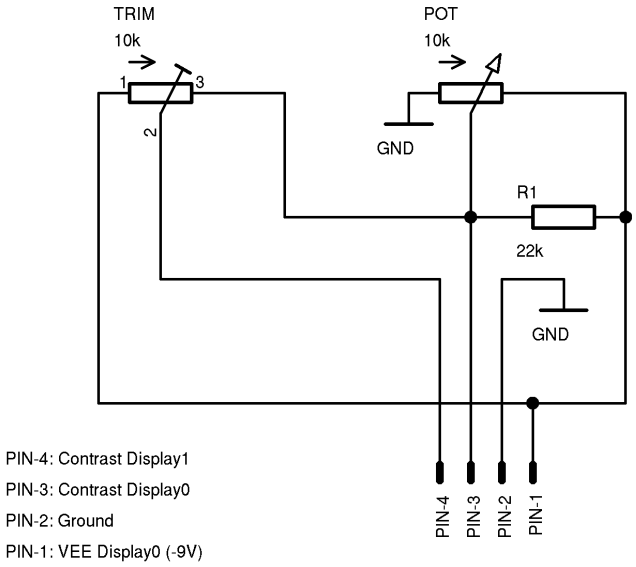

Abbildung 3.24: Schaltplan der Kontrastplatine

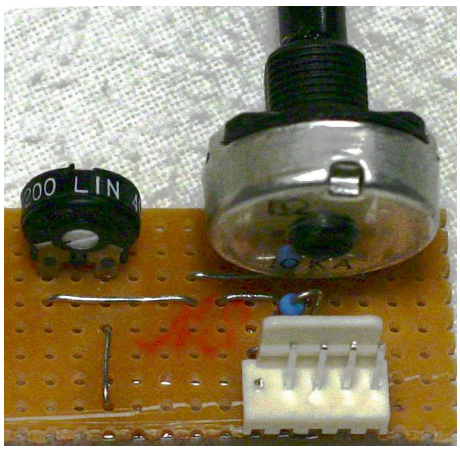

Abbildung 3.25: Kontrastplatine

3.3.3 Display Adapterplatinen

Da die Pinbelegung der Display Stecker nicht mit der Belegung der Display Buchse an den Core Modulen übereinstimmt, haben wir für jedes Display eine Adapterplatine angefertigt, die die Pinbelegung anpasst und zwischen die Flachbandkabel gesteckt wird. In Tabelle 3.1 wird die Umsetzung erläutert. An die Adapterplatine

Pin Display	Pin Core	Symbol	Funktion
1	nc	FG	Frame Ground
2	6	V_{SS}	Ground
3	8	V_{DD}	Power Supply ($V_{dd} > V_{ss}$)
4	nc	V_0	Power Supply for LCD Drive
5	14	$\overline{WR}$	Data write (write Data at „L“)
6	16	$\overline{RD}$	Data read (read Data at „L“)
7	6	$\overline{CE}$	Chip enable (active at „L“)
8	12	$C/\overline{D}$	Command/Data read/write
9	nc	V_{EE}	Negative voltage output
10	8	$\overline{RESET}$	Controller reset (reset at „L“)
11	15	$DB0$	Data Bus LSB
12	13	$DB1$	.
13	11	$DB2$	.
14	9	$DB3$	.
15	7	$DB4$	.
16	5	$DB5$	.
17	3	$DB6$	.
18	1	$DB7$	Data Bus MSB
19	6	FS	Font select
20	nc	nc	not connected

Tabelle 3.1: Pinbelegung der LCD Adapterplatine

ist außerdem die Platine zur Kontrastregelung angeschlossen (siehe Abschnitt 3.3.2). Von einer Adapterplatine wird von der Displayseite an Pin 2 Ground und an Pin 9 die negative Versorgungsspannung V_{EE} (vom Display selbst erzeugt) abgegriffen,

und vom Kontrastmodul verarbeitet. Dieses liefert an Pin 4 beider Adapterplatinen die Spannung V_0 zur Einstellung des Kontrasts zurück.

3.3.4 Speichererweiterung und Umschalter

Das Bankstick Speichermodul von ucapps (siehe Abschnitt 3.2.4) reicht vom Speicherplatz her nicht für unsere Anwendung aus. Deshalb war es erforderlich, ein weiteres Speichermodul aus Banksticks aufzubauen (siehe Abbildungen 3.26 und 3.27). Dieses verfügt über 256kByte Speicherkapazität (4 x 64kByte Chips von Atmel

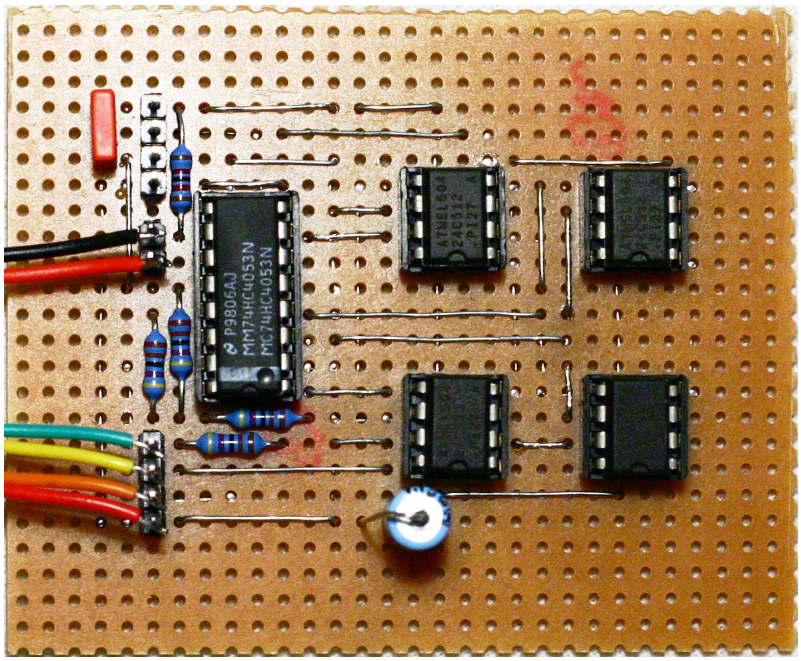

Abbildung 3.26: Bankstick Platine mit Umschalter

- „AT24C512“). Beide Module sind über einen integrierten elektronischen Schalter, einen „74HC/HCT4053 Triple 2-channel analog multiplexer/demultiplexer“, am I2C Bus angeschlossen. Der Schalter ist erforderlich da nicht mehr als acht Eproms gleichzeitig ansprechbar sind, da sie nur über maximal drei Adresspins verfügen. Er wird über den Pin 7 des Digital Out Moduls (siehe Abschnitt 3.2.3 und 4.1) des ersten Cores geschaltet und wechselt zwischen beiden Speichermodulen. Hierbei ist darauf zu achten, dass nicht das IIC MIDI Modul hinter dem Schalter angeschlossen wird, da es sonst beim Umschalten deaktiviert würde. Der Schalter legt die Daten- und die Clockleitung bei einer null an Pin 7 auf das 512kByte Modul, bei einer eins ist das 256kByte Modul aktiv. Die Daten- und Clockleitungen der beiden Module sind über 4.7kOhm Widerstände auf 5V gelegt, die den Ruhezustand auf dem Bus des deaktivierten Speichermoduls sichert. Ist das Verbindungskabel „DOUT für Bankstickschalter“ nicht angeschlossen, liegen über einen 12kOhm Widerstand 5V, also eine eins am Schalter an. In diesem Fall wäre das 256kByte Speichermodul aktiv. Will man das 512kByte Speichermodul nutzen, sollte das Verbindungskabel „DOUT für Bankstickschalter“ mit einem Kurzschlusstecker abschließen. Eine Speichererwei-

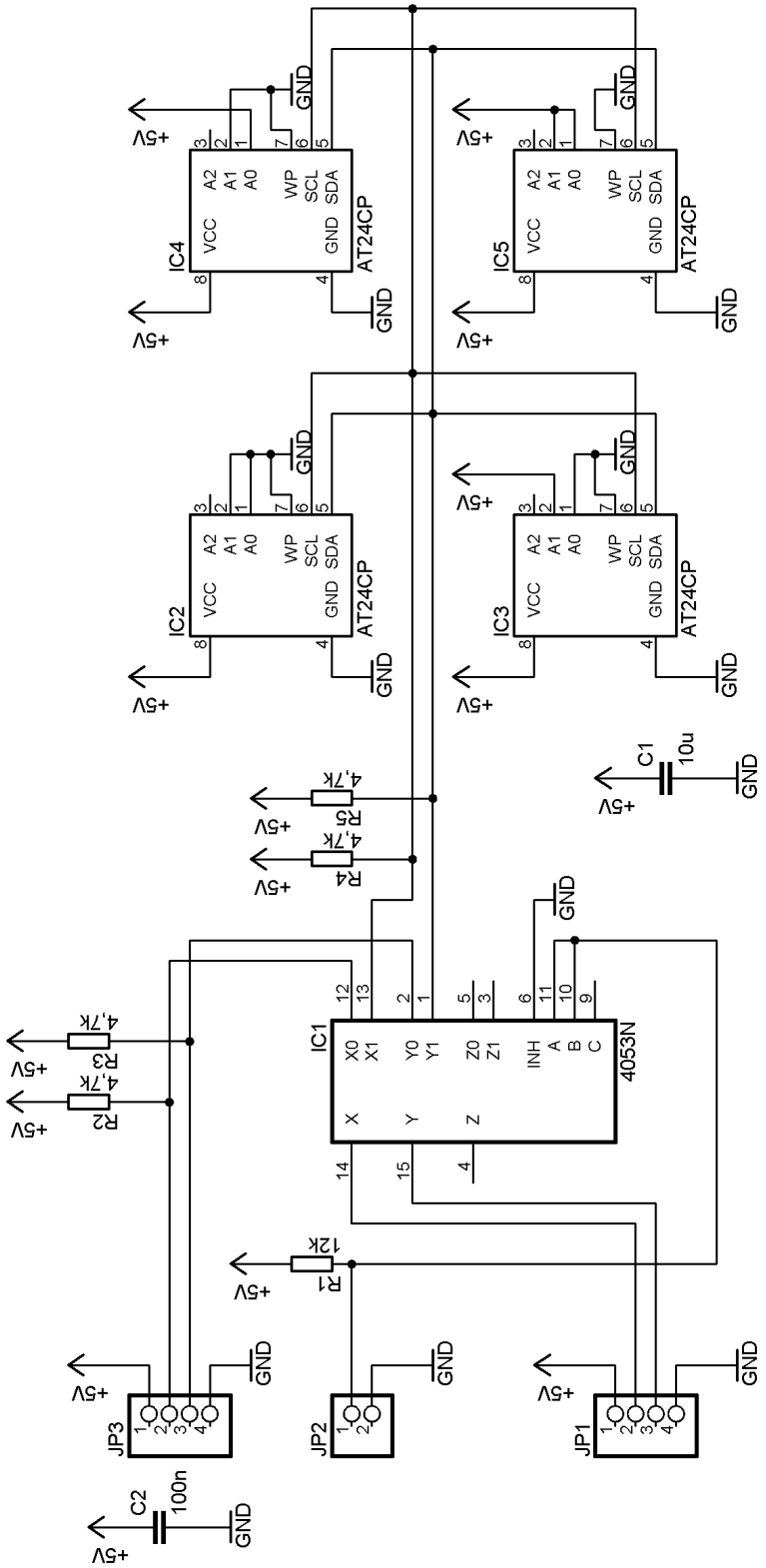

Abbildung 3.27: Schaltplan der zweiten Bankstick Platine mit Umschalter

terung wäre dann möglich, wenn mittels Schalter sichergestellt wäre, dass nur eine BankStick Platine zur Zeit am I2C Bus aktiv ist.

HINWEIS: Die Speicherchips von Atmel und Microchip unterscheiden sich in der Anzahl der Pins zur Adressierung. Die Atmel Chips verfügen nur über zwei Pins und sind deshalb nur als Vierergruppe pro Modul nutzbar. Die Microchip Chips verfügen über drei Pins und sind als Achtergruppe pro Modul nutzbar.

3.4 Aufbau der MIDIbox

3.4.1 Interner Aufbau - Blockschaltbild und Verkabelung

Um den Aufbau und die Verkabelung unseres Controllers darzulegen, folgen vier Übersichtstabellen (inkl. Farbkodierung) der verlegten Kabel (siehe Tabellen 3.2, 3.3, 3.4 und 3.5). Außerdem beinhaltet dieser Abschnitt in Abbildung 3.28 und 3.29 ein Blockschaltbild und einen Verkabelungsplan der MIDIbox GLCD.

	MIDI Kabel
Abkürzung	Beschreibung
M0	MIDI In Buchse 0 − > MIDI Merger J1
M1	MIDI In Buchse 0 − > MIDI Merger J3
M2	MIDI Merger J4 − > Core Modul 0 J13
M3	I2C MIDI Modul J5 − > Core Modul 1 J13
M4	Core Modul 1 J12 − > MIDI Out Buchse
Farb-kodierung	Beschreibung
Rot	Data (M-)
Braun	Masse
Schwarz	+5V

Tabelle 3.2: MIDI Kabel - Farbkodierung

	Digital IN und Digital OUT Kabel
Abkürzung	Beschreibung
DI0	Digital In Modul 0 J1 − > Core Modul 0 J9
DI1	Digital In Modul 1 J1 − > Digital In Modul 0 J2
DO0	Core Modul 0 J8 − > Digital Out Modul 0 J1
Farb-kodierung	Beschreibung
Grün	Latch Clock
Gelb	Serial Clock
Orange	Serial In oder Serial Out
Rot	+5V
Braun	Masse

Tabelle 3.3: Digital In - Digital Out - Farbkodierung

	I2C Kabel
Abkürzung	Beschreibung
I2C0	Core Modul 0 J4 − > I2C MIDI Modul J2
I2C1	I2C MIDI Modul J7 − > Speicher Modul 1 J1
I2C2	Speicher Modul 1 J3 − > Speicher Modul 0 J1
Farb-kodierung	Beschreibung
Grün	+5V
Gelb	Serial Data
Orange	Serial Clock
Rot	Masse

Tabelle 3.4: I2C Kabel - Farbkodierung

	Sonstige Kabel
Abkürzung	Beschreibung
	MIDI IN zu LTC
	MIDI OUT zu LTC
	LTC zu MIDI IN LED
	LTC zu MIDI OUT LED
	DIN für Enc(Encoder)
	DIN für Enc(Taster)
	DIN für Taster
	DOUT für Bankstick
	DOUT für Taster LED
DC0	+9V von Netzteil zu Core 0 J1
DC1	+9V von Netzteil zu Core 1 J1
DC2	+9V von Netzteil zu Beleuchtungsmodul
DC3	+5V von Core 0 J2 zu MIDI Merger J2
DC4	+5V von Beleuchtungsmodul zu Display 0
DC5	+5V von Beleuchtungsmodul zu Display 1
Farb-kodierung	Beschreibung
Grau	DIN Signal von Tasten und Encodern
Rot	+5V oder +9V
Braun	Masse
Schwarz	Masse

Tabelle 3.5: Sonstige Kabel - Farbkodierung

3.4.2 Externer Aufbau - Anschlüsse nach außen

In diesem Abschnitt wird die Rückseite der MIDIbox GLCD mit ihren Anschlüssen und Bedienelementen beschrieben, wie in Abbildung 3.30 zu sehen. Die Aufzählung erfolgt von links nach rechts. Die Hauptspannungsversorgung für die MIDIbox GLCD beträgt 9V, und das Netzteil muss im Betrieb 600 mA aufbringen. Ganz links an der

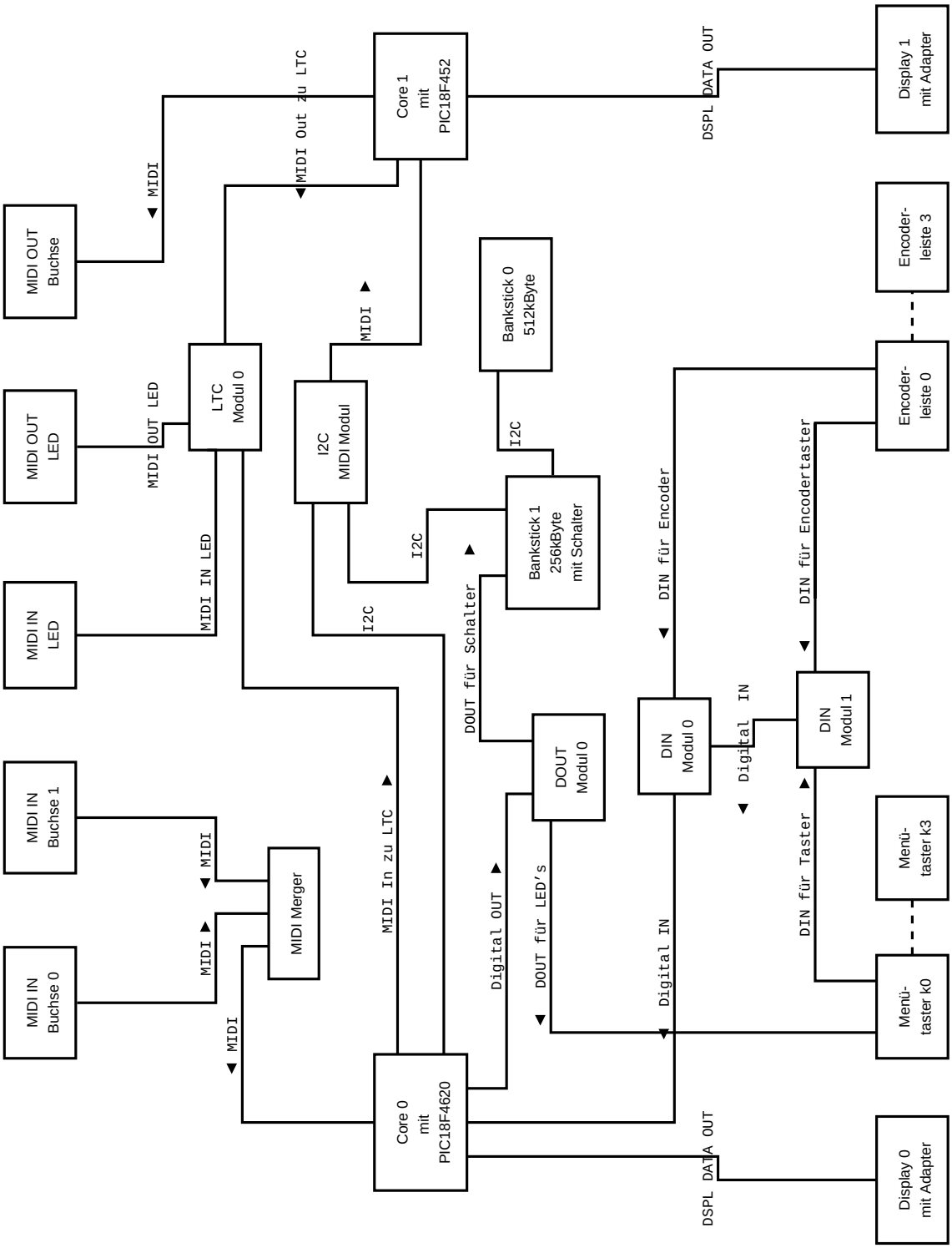

Abbildung 3.28: Blockschaltbild der MIDIbox GLCD

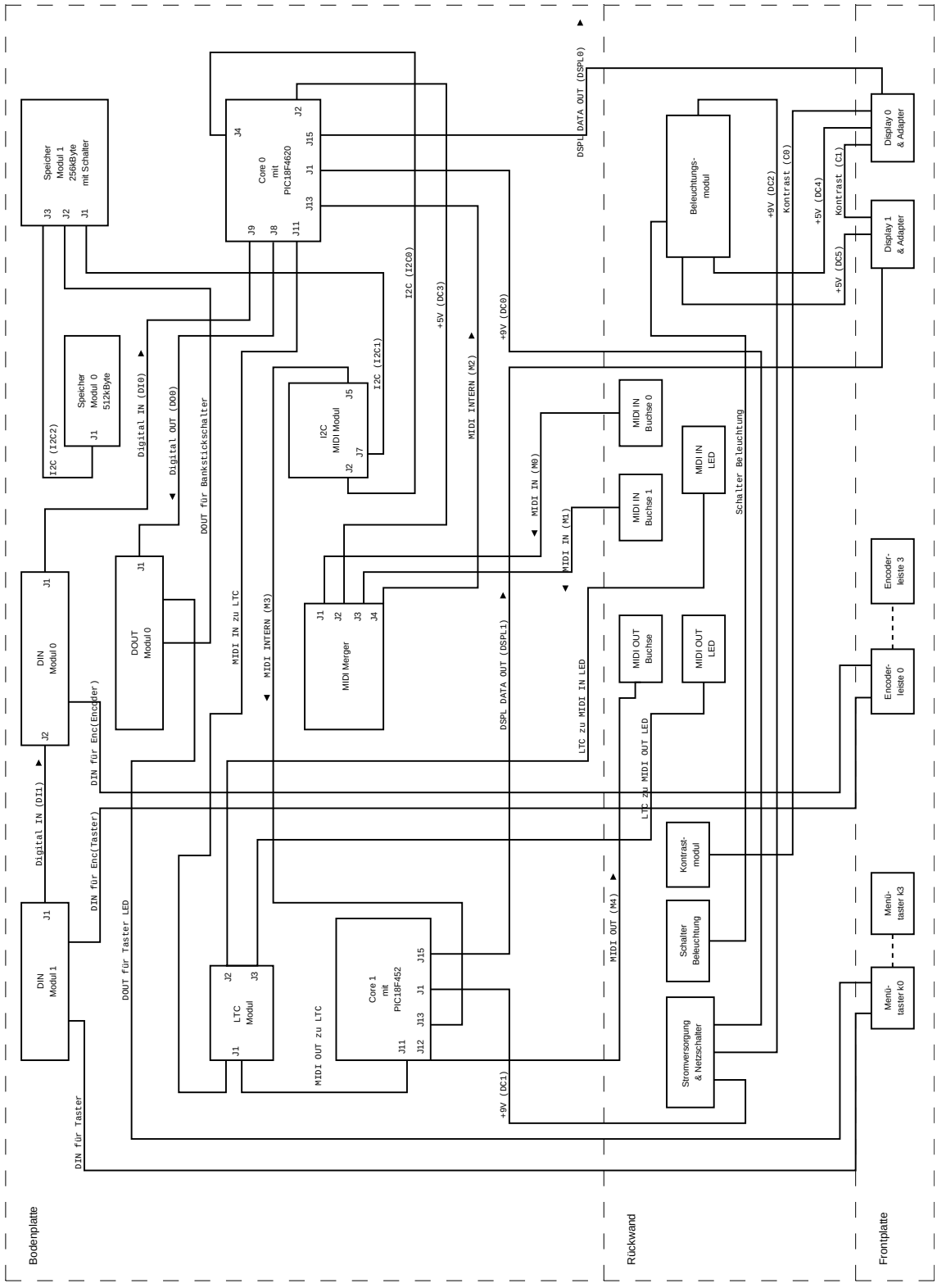

Abbildung 3.29: Verkabelung der MIDIbox GLCD

Abbildung 3.30: Rückseite der MIDIbox GLCD

MIDIbox findet sich ein Hauptschalter, über den diese Spannungsversorgung an die beiden Core Module und die Display Beleuchtungs Platine angeschlossen ist. Rechts daneben ist der Anschluss für das Netzteil. Hier liegt Pin 1 auf Masse und Pin 2 auf +9V. Pin 3 und 4 sind nicht belegt. Die Pinbelegung ist zwar für die ucapps Module nicht relevant, da sie einen Gleichrichter besitzen, aber für die Platine der Spannungsversorgung für die Displays sehr wohl. Danach kommt ein Schalter für die Display Beleuchtung, der zur entsprechenden Platine führt (siehe Kapitel 3.3.1). Als nächstes findet sich ein Potentiometer zur Einstellung des Kontrastes der beiden Displays. Es sitzt direkt auf der Display Kontrast Platine (siehe Kapitel 3.3.2), die intern an der Rückseite des Gehäuses angebracht ist. Als letztes folgen drei MIDI Anschlüsse: zuerst eine MIDI Out Buchse, die vom Core 1 kommt und zwei gleichwertige MIDI Ins, die vom MIDI Merger zusammen gefasst und zum Core 0 geführt werden (siehe Kapitel 3.2.5). Über den MIDI Buchsen finden sich zwei MIDI Transfer Leds, eine grüne, die die MIDI Out Aktivität anzeigt und eine rote für beide MIDI Ins.

4 Software

Um eine komplexe Funktionalität des MIDI Controllers, wie in der Aufgabenstellung gefordert, zu realisieren, wurde die Hochsprache C verwendet. Zusammen mit den vielen Systemfunktionen des MIOS (MIDI Operating System) ist der Programmcode stark vom Microcontroller abstrahiert. Grundlegend besteht das Programm aus einer Reihe von Eventhandlern, die vom MIOS beim Eintreten der entsprechenden Ereignisse aufgerufen werden. Es existieren Handler für das Eintreffen von MIDI

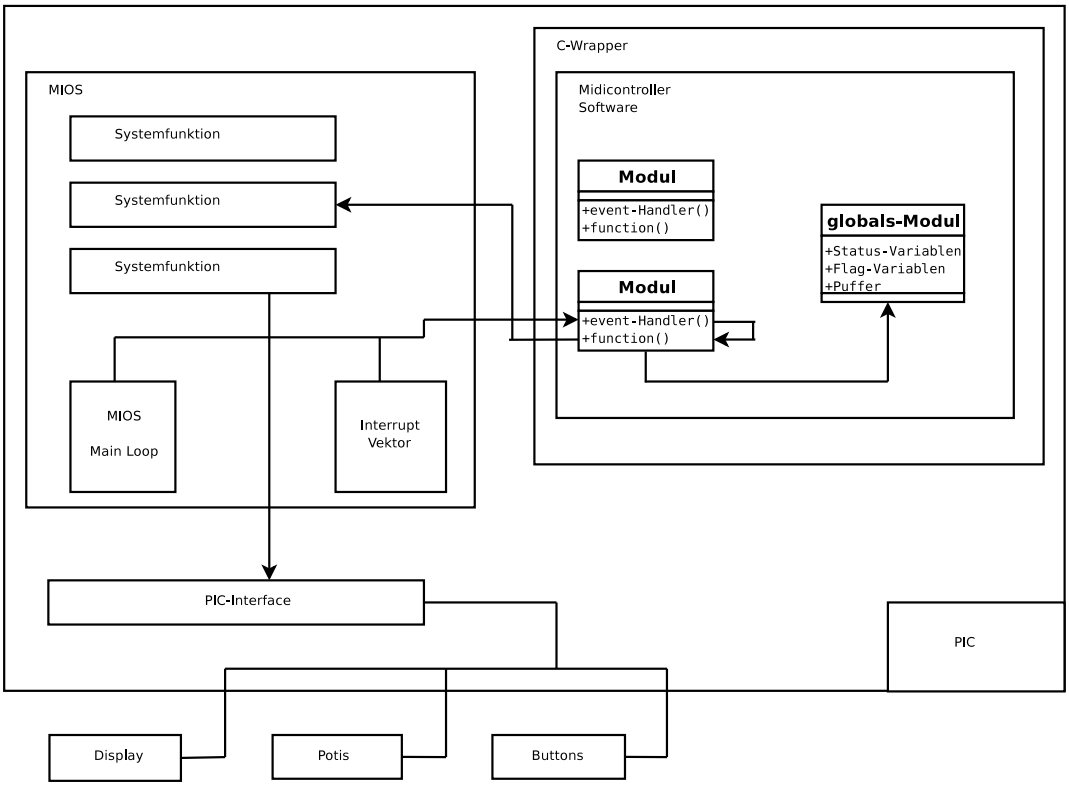

Abbildung 4.31: Struktogramm des generellen Aufbaus und Kommunikation der Software des Betriebssystems und der Hardware

Nachrichten, das Ablaufen eines Timers, das Update der Displays und verschiedene Benutzereingaben. Der modulare Aufbau des Programms (siehe Abbildung 4.31) orientiert sich an diesen Eventhandlern, die in verschiedene Quelldateien aufgeteilt sind. Ausgehend von der main.c werden diese Module mithilfe von *include* Anweisungen zu einer großen Quelldatei zusammengefügt (Abbildung 4.32). Der üblicherweise zum Verbinden von Modulen genutzte Linker konnte leider nicht verwendet werden da dieser zu korrumpiertem Binärcode führte.

Die Kommunikation zwischen den Modulen ließ sich nur mittels globaler Variablen realisieren, da das MIOS jedes Modul nur mittels nicht vom Programmierer parametrisierbarer Ereignisse aufruft. Die globalen Variablen zur Kommunikation bzw. zum Speichern des aktuellen Zustandes werden zentral in der Datei *globals.c* verwaltet (auch als *globals* Modul bezeichnet). So ist beispielsweise das Ändern der Anzeige bei Empfang von MIDI Nachrichten oder Benutzereingaben aus verschiede-

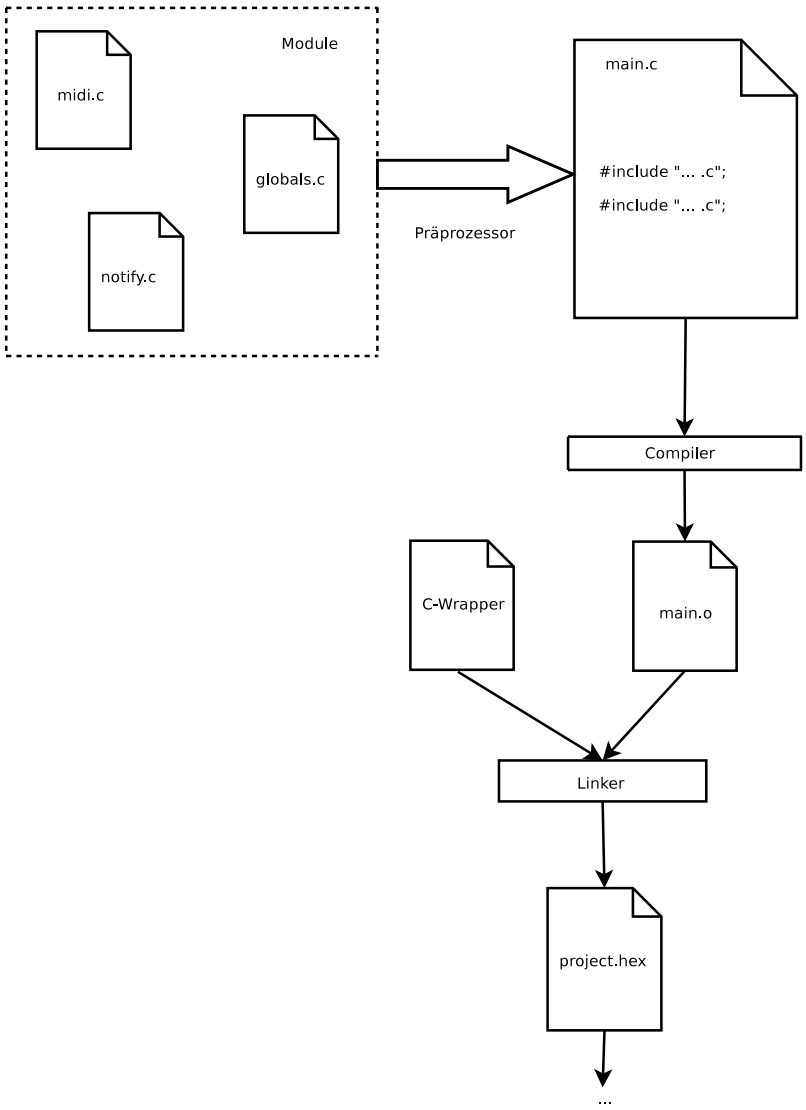

Abbildung 4.32: Einbinden der Module und Erstellung des Binaries

nen Modulen heraus notwendig. Da die Displays aber nur innerhalb eines bestimmten Eventhandlers (und somit auch eines Moduls) geändert werden dürfen, muss dieser über eventuell geänderte Displaybereiche mittels globaler Flag- und Status-Variablen informiert werden. In Abbildung 4.33 ist exemplarisch die Kommunikation

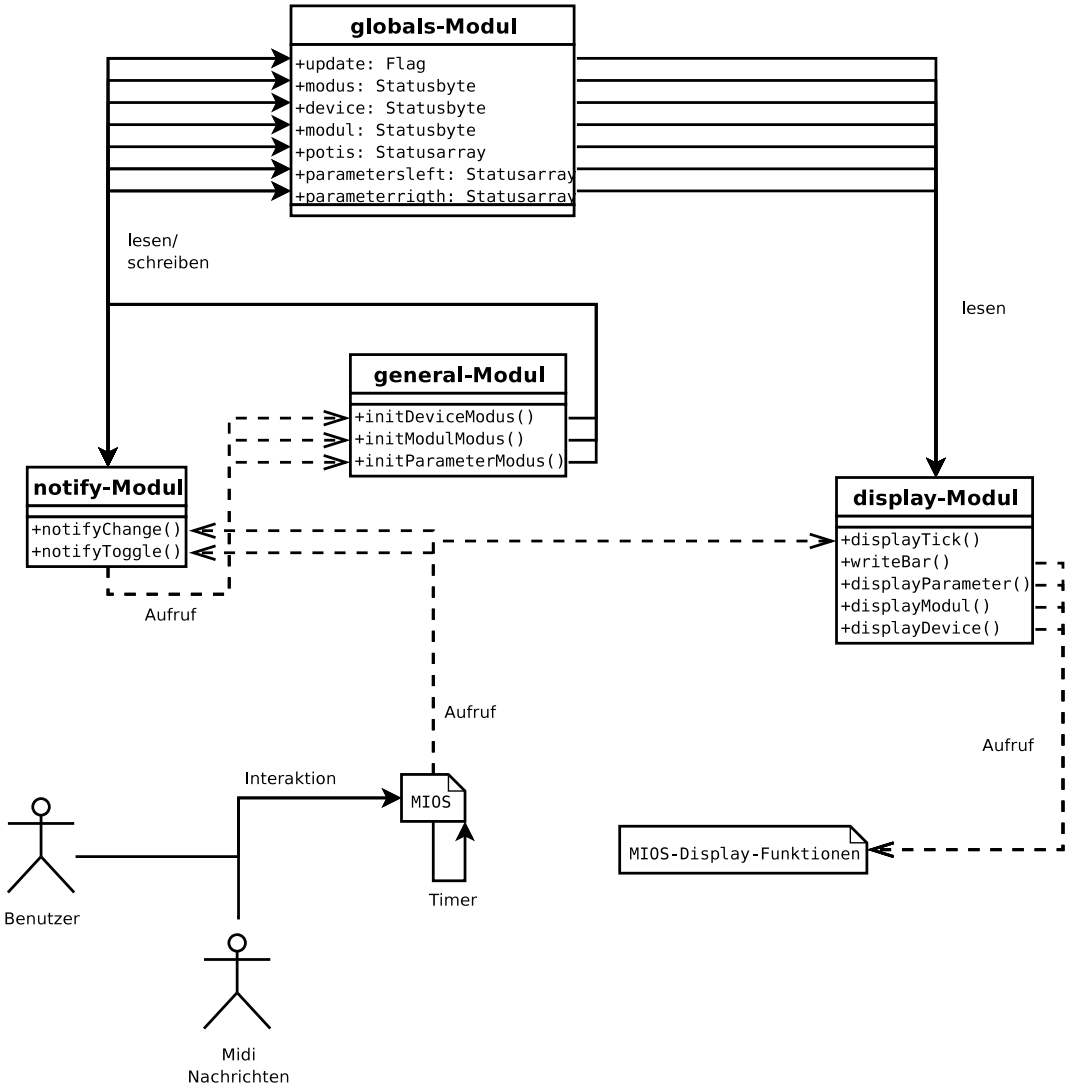

Abbildung 4.33: Kommunikation der Module untereinander mittels des *globals*-Moduls

zwischen den verschiedenen Modulen zu sehen. Module wie *notify*, *midi* oder *timer* errechnen aus den globalen Variablen und dem auftretenden Ereignis neue Zustände des MIDI Controllers. Andere Module können mithilfe dieser Zustände wieder Aktionen ausführen. In Abbildung 4.33 wird z.B. das *notify* Modul durch ein externes Ereignis aufgerufen, woraufhin die globalen Variablen gelesen und geschrieben werden. Bei dem nächsten Aufruf der *Display_Tick* Funktion wird auf die veränderten Variablen Rücksicht genommen und eine angepasste Ausgabe auf die Displays geschrieben.

4.1 Speicherfunktionen

Im Sourcefile memory.c finden sich drei Funktionsgruppen zum Lesen/Beschreiben der Banksticks und des Edit Buffers im RAM. Die erste Gruppe besteht aus den Funktionen *writeBank* und *readBank*, die eine kontinuierliche Adressierung der 12 Banksticks realisieren. Das eigentliche Problem ist, dass die MIOS Speicherfunktionen nur einen Bankstick von 64kB adressieren können. Will man einen anderen Bankstick adressieren, muss zuerst mit einer zusätzlichen Funktion der Chip umgeschaltet werden. Diese Herangehensweise ist für ein Projekt unserer Größe, bei dem ständig auf den Speicher zugegriffen werden muss, sehr unvorteilhaft. Außerdem unterstützt das MIOS höchstens 8 Banksticks, was für die MIDIbox GLCD zu wenig war. Also haben wir neue Funktionen zur Speicheradressierung verfasst, die eine drei Byte *long* Adresse übergeben bekommen und auf alle 12 Banksticks einheitlich zugreifen können. Die Adresse beinhaltet nun im höchstwertigen Byte den Bankstick Chip (0 bis 11) und in den zwei niederwertigen Bytes die Adresse auf diesem Chip. Zwei Fälle mussten in der Funkion berücksichtigt werden, nämlich die Umschaltung zwischen den einzelnen Chips auf einer Bankstick Platine und zusätzlich die Umschaltung zwischen der ucapps Bankstick Platine und der von uns gebauten. Diese wurde zu diesem Zweck mit einem elektronischen Schalter versehen, der über Pin 7 des DOUT Moduls angesteuert wird. Da Lese- und Schreibzugriff fast identisch sind, werden sie hier auch zusammen behandelt. Wie in Abbildung 4.34 zu sehen, wird nach einem Aufruf einer Bankstick Funktion zuerst geprüft, ob der übergebene Adressbereich gültig ist. Ist dies der Fall, wird die Bankstick Platine, auf die gerade zugegriffen wird, mit der zuletzt aktiven verglichen. Hierbei liegt die ucapps Platine im Adressbereich 0x00000 bis 0x7FFFF und die von uns gebaute im Bereich 0x80000 bis 0xBFFFF. Um Zeit zu sparen, wird nur umgeschaltet, wenn sich die aktuelle Platine von der zuletzt verwendeten unterscheidet. Das gleiche Prinzip findet sich anschließend beim einzelnen Bankstick Chip wieder, der auch nur neu eingestellt wird, wenn der zuletzt verwendete nicht gleich dem aktuell adressierten ist. Ist dann die der Adresse entsprechende Platine und der Bankstick Chip eingestellt, erfolgt der Speicherzugriff über die reguläre MIOS Funktion. An dieser Stelle wäre auch eine Implementierung einer Speichererweiterung in Form weiterer Bankstick Platinen in die Software einzufügen (siehe Abschnitte 4.5 und 3.3.4). Es wäre eine neue if Abfrage in Form der zweiten und dritten mit entsprechend anderem Adressbereich hinzuzufügen, in der dann der entsprechende Pin der neuen Platine am DOUT gesetzt werden müsste.

HINWEIS: Hier muss angemerkt werden, dass alle Variablen mit denen die Adresse verrechnet wird auch vom Typ *long* sein oder zu *long* gecastet werden müssen, da sonst das höchstwertigste Byte nicht auf die Rechnungen reagiert.

Die zweite Funktionsgruppe besteht aus den Funktionen *writeStream* und *readStream*,

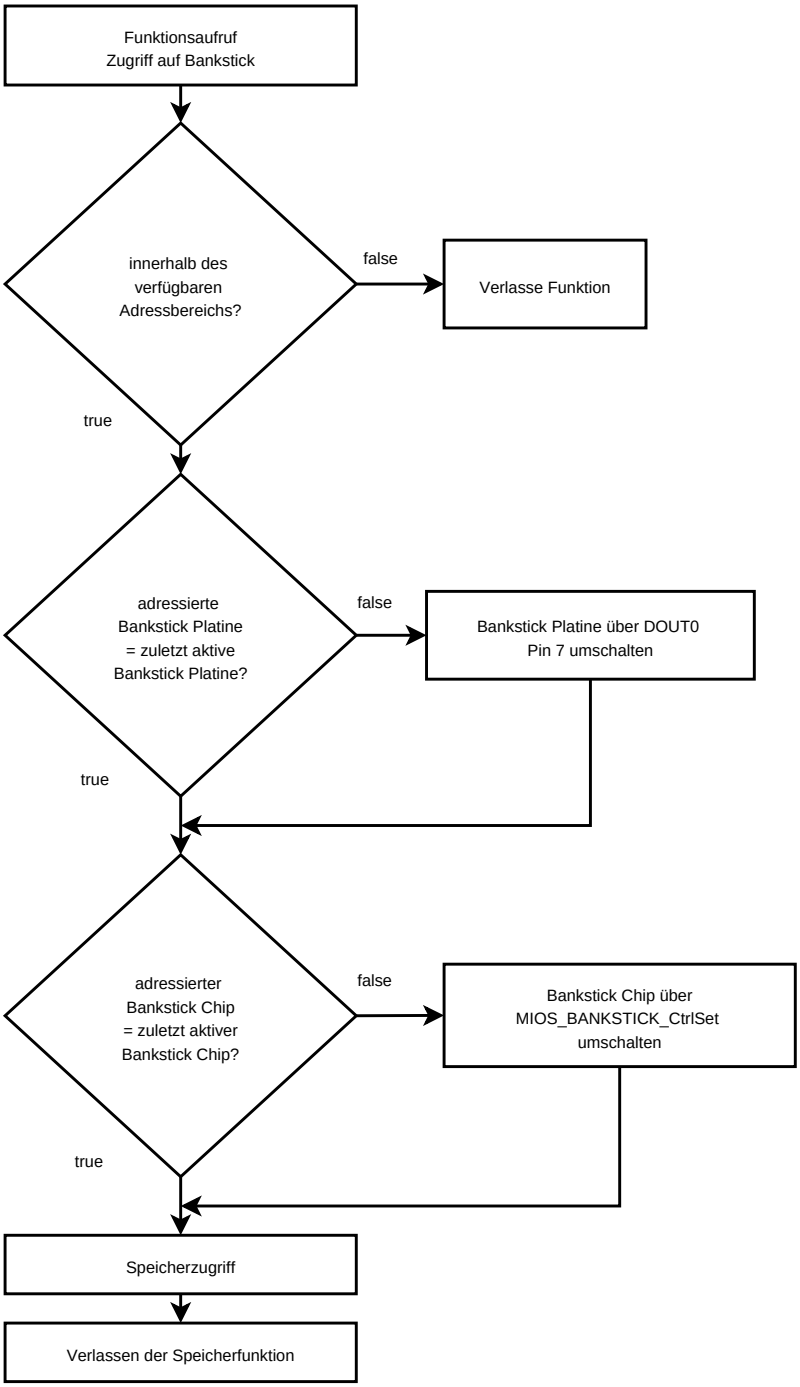

Abbildung 4.34: Speicherzugriff auf Banksticks

die jeweils ein Byte an der in der globalen Variable mem_adress gespeicherten Adresse aus dem Speicher lesen oder hineinschreiben und dann die Variable inkrementieren. Diese Funktionen können im Zusammenhang mit über MIDI empfangenen oder versendeten Daten eingesetzt werden, um einen Datenstrom zu realisieren.

Die dritte Funktionsgruppe besteht aus den Funktionen $writeEditBuffer$ und $readEditBuffer$. Da es Probleme gab, Arrays zu verwenden, die größer als 256 Byte sind, haben wir acht solcher kleinen Arrays ($edit\_buffer0$ bis $edit\_buffer7$)zu einem Array der Größe 2kB zusammengefasst, das jetzt über diese Funktionen adressiert werden kann, und Platz genug für den Edit Buffer eines Synthesizers bietet.

4.2 Editbuffer

Der Editbuffer ist der Speicher im Synthesizer in dem alle Klangparameter enthalten sind. Manche Synthesizer besitzen mehrere Editbuffer für verschiedene Parametergruppen, wie z.B. Klangerzeugung oder Effektsektion, deshalb bietet die MIDIbox mehrere Stellen in den Navigationsebenen um einen Dumprequest, eine Editbufferanfrage, zu senden (siehe Abschnitt 4.6). Die Device- und Modulstruktur besitzt ein Feld zur Adressierung der Dumpstruktur (siehe Abschnitt 4.3) in der alle Informationen enthalten sind, die zum Anfordern und Empfangen eines Editbuffers nötig sind (siehe Abschnitt 4.5). Der empfangene Editbuffer wird im Ram der MIDIbox abgespeichert. Die zu ladenden Parameter eines Moduls lesen ihren Initialwert aus der Stelle im Editbuffer die der Benutzer in der Java Software definiert hat und im *edit* Feld der Parameterstruktur gespeichert ist (siehe Abschnitt 4.4). Um Synthesizer die nur über den Transfer eines Editbuffers programmierbar sind, ebenfalls mit der MIDIbox bedienen zu können gibt es die Möglichkeit, den eingelesenen Editbuffer in der Form an den Synthesizer zurückzusenden, dass dieser sein aktuelles Klangprogramm mit den Daten des neuen Editbuffers überschreibt (siehe Anschnitt 4.7).

4.3 Speicheraufteilung

Die BankStick Platinen dienen zur Speicherung aller Konfigurationen der MIDIbox. Alle Geräte sowie deren Module und Parameter werden in ihm abgelegt. Da die MIDIbox das Versenden von SysEx Strings unterstützen soll, wurde der Speicher in zwei Bereiche (siehe Tabelle 4.6) unterteilt. Zum einen in einen statischen Bereich, in dem die Daten abgelegt sind, die für jedes Device, Modul oder jeden Parameter gespeichert werden müssen, zum anderen in einen dynamischen Bereich, in dem je nach Bedarf weitere Informationen abgelegt werden können. Der dynamische Bereich ist in sechzehn gleichgroße Abschnitte aufgeteilt, die je einem Device zugeordnet sind. Um ein Device, Modul oder einen Parameter mit optionalen Konfigurationen zu erweitern, enthalten die statischen Datenstrukturen Zeiger auf den dynamischen Speicher (siehe Abbildung 4.36). Diese Zeiger sind relativ zur Anfangsadresse des dynamischen Blocks berechnet, zu dem das Device und auch deren untergeordnete Module und Parameter gehören.

Beginn	Ende	Inhalt
0	512k	statisch
512k	768k	dynamisch

Tabelle 4.6: Speicheraufteilung

4.3.1 statische Speicheraufteilung

Startadresse	Endadresse	Inhalt
0	160	[Devcies]
161	20.640	[Device-Module]
1.441	327.680	[Parameter]

Tabelle 4.7: statische Speicheraufteilung

Aufgeteilt ist der statische Speicher in einen Device, Modul und einen Parameterblock (siehe Tabelle 4.7). Dabei sind im Deviceblock alle sechzehn Devices (siehe Tabelle 4.8) nacheinander abgelegt. Darauf folgen die Module die äquivalent zu den Geräteinformationen in Reihe abgelegt werden. Dabei ist zu beachten, dass zuerst die Module nach ihren zugehörigen Devices gruppiert werden, was beispielsweise bedeutet, dass alle Module, die zum ersten Device gehören auch am Anfang des Modulblocks gespeichert werden. Innerhalb der Devicegruppierung werden die Module zu Banken zusammengefasst und je sechzehn Module je Bank nacheinander im Speicher abgelegt. Da zu jedem Modul sechzehn Parameter gehören werden diese zum Schluss, ähnlich der Module im Eprom, gespeichert. Die Speicherung ist in Abbildung 4.35 detailliert dargestellt.

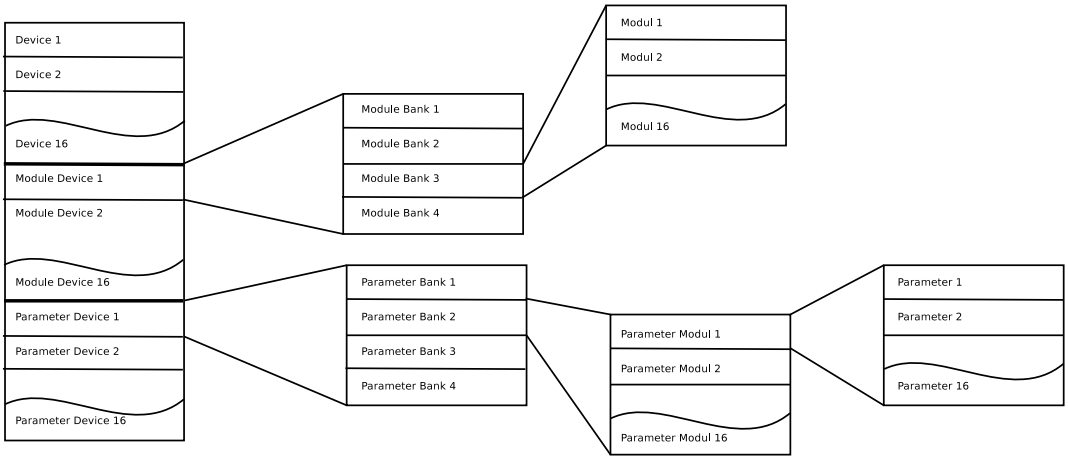

Abbildung 4.35: Aufteilung des statischen Speichers

In den Datenstrukturen für Devices und Module sind außer den Namen, die bei Bedarf auf die Displays geschrieben werden, nur eine 16 Bit Adresse für den dynamischen Speicher enthalten (siehe Abbildung 4.8). Als Erweiterung für diese Datenstrukturen kommen nur SysEx Aufrufe in Frage, die im Zusammenhang mit einer Editbuffer Anfrage stehen (siehe Tabelle 4.10).

Datentyp	Feldname	Aufgabe
char[20]	name	zweizeiliger Name der auf dem Display angezeigt wird
unsigned int	dump_addr	Adresse auf eine Datenstruktur im dynamischen Speicher, um einen Editbuffer anzufordern

Tabelle 4.8: Device- und Modul-Struktur

Die Parameterstruktur wird in Kapitel 4.4 ausführlich erklärt und soll hier nur der Vollständikeit halber angeführt werden. Diese Datenstruktur kann bis zu 2 Adressen auf den dynamischen Speicher enthalten, die es zum einen erlaubt, den Parameter bei Bedarf mit einem SysEx String zu verbinden und zum anderen, Parametern verschiedene Wertnamen zuordnet. Da nicht jeder Parameter sich direkt mit Zahlen assoziieren lässt, besteht die Möglichkeit, den verschiedenen Werten eines Parameters Bezeichnungen zuzuordnen (Beispielsweise Oszillatorform: Dreieck,Sinus,Rechteck).

4.3.2 dynamische Speicheraufteilung

Die Aufteilung des dynamischen Speichers in sechzehn Blöcke begründet sich in der Art der Datenübertragung (siehe Kapitel 4.5) und der Bestrebung, aufwendige und fehleranfällige Defragmentierungsfunktionen zu vermeiden. Da bei der Übertragung immer vollständige Devices auch mit deren dynamischem Speicher transferiert werden sollen, sind die Algorithmen zum Aufteilen und Referenzieren dieses Speicherbe-

reichs in die Software verschoben. Aus Sicht der MIDIbox ist somit die Bezeichnung dynamischer Speicher irreführend. Variabel adressierter Speicher träfe es besser.

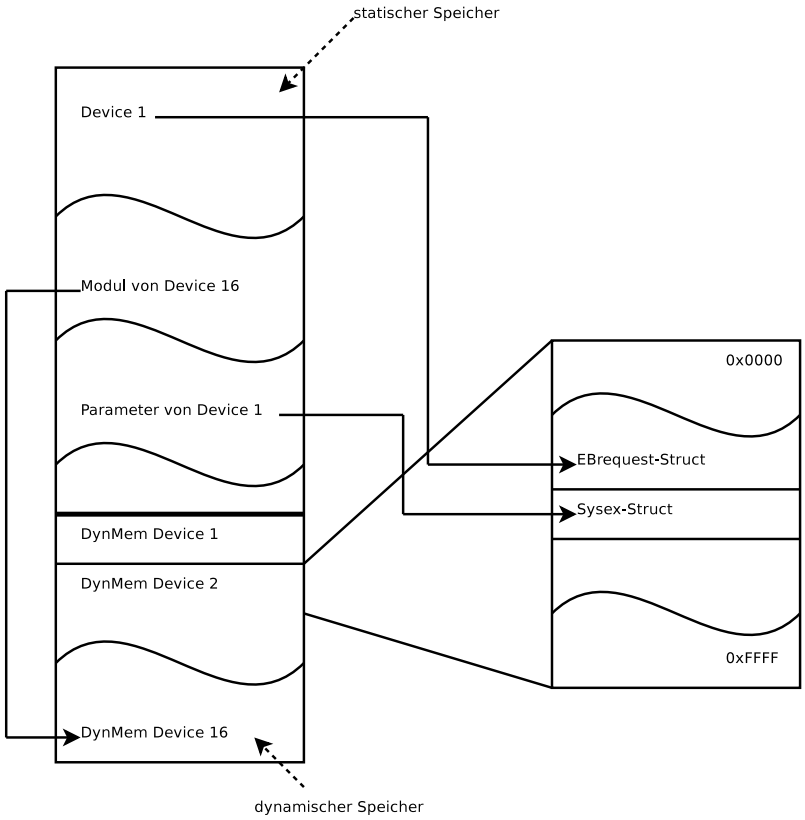

Abbildung 4.36: Aufteilung und Referenzierung des dynamischen Speichers

Im dynamischen Speicher existieren drei Datenstrukturen die im Folgenden genauer erläutert werden.

- **SysEx-Struktur** Die SysEx Struktur (siehe Tabelle 4.9) ermöglicht durch Ändern eines Parameters eine SysEx Nachricht an den Synthesizer zu verschicken. Um dies zu erreichen, muss sowohl der String selbst im Speicher abgelegt werden, als auch Informationen, wie der Parameterwert in ihn eingetragen wird. Die SysEx Nachricht ist direkt im Anschluss der SysEx Datenstruktur im Eprom abgelegt. Ihre Länge wird durch das *length* Feld bestimmt. Um ein Datum in den String zu speichern, muss sowohl eine Datenkonvertierungsfunktion als auch die Position des Datums im String bekannt sein. Datenkonvertierungsfunktionen konvertieren[1] und kopieren den aktuellen Wert des Parameters in den SysEx String.

- **Sswitch-Struktur** Die Sswitch Struktur ist keine physische sondern eine imaginäre Datenstruktur, um Wertnamen der Parameter im dynamischen Speicher abzulegen. Die statische Parameterstruktur zeigt dabei auf die Anfangsadresse mehrerer 6 Byte langer Strings. Die Anzahl der Strings errechnet sich aus

[1]Die Konvertierung ist nötig, da viele Hersteller verschiedene Datenformate einsetzen (z.B. little Endian,big Endian,...)

Datentyp	Feldname	Aufgabe
unsigned char	convert_func	numerischer Bezeichner der Konvertierungsfunktion
unsigned char	convert_func_addr	Adresse an der die Konvertierungsfunktion den Wert des Parameters schreiben soll
unsigned char	length	Länge der im Speicher folgenden Daten

Tabelle 4.9: SysEx-Struktur

dem maximalen minus dem minimalen Wert des Parameters geteilt durch den Skalierungsfaktor.
$Anzahl_{Strings} = \frac{valueMax - valueMin}{scale}$

- **Dump-Struktur** Die Dump Struktur (siehe Tabelle 4.10) ermöglicht das Abfragen des Editbuffers. Zur Abfrage wird, wie bei der SysEx Struktur, ein SysEx String benötigt, der auch an die Struktur im dynamischen Speicher angehängt ist. Um alle Parameterwerte aus dem Editbuffer auslesen und schreiben zu können, wird wieder eine Datenkonvertierungsfunktion benötigt (*eb_convert*). Des Weiteren ist das Bilden einer Checksumme beim Versenden des Puffers zurück an den Synthesizer erforderlich, was über die Checksum Funktion realisiert wird. Beim Übermitteln des Editbuffers wird ein Header vom Synthesizer vorausgeschickt, der mithilfe dessen Längeangabe (*wrong_header_length*) beim Empfang abgeschnitten wird. Zum Zurückschicken des Editbuffers wird ein neuer Header benötigt, der dem SysEx String im dynamischen Speicher folgt und die Länge *header_length* hat. Zum Schluss ist noch zu erwähnen, dass es mit der Dump Struktur auch möglich ist, der Editbufferanfrage einen SysEx String vorauszuschicken (z.B. zum Stellen einer MIDI Kreuzschiene). Da nur auf eine Nachricht gezeigt werden kann, sind die Dumpnachrichten in einer Liste organisiert. Durch Setzen eines Flags in der *typ* Variablen, ist es möglich auf eine noch folgenden Dumpstruktur aufmerksam zu machen. Diese kann nach Ablauf des *Timeouts* abgearbeitet werden. Beim Typ SysEx Dump (*typ*=0x01) stellt dieser Timeout die Verzögerung in 10'er Millisekundenschritten bis zum Senden des nächsten Dumps dar. Bei der Abfrage des Editbuffers (*typ*=0x00) gibt er die Wartezeit in Sekunden an, bis die Editbufferanfrage als gescheitert angesehen werden soll.

Datentyp	Feldname	Aufgabe
unsigned char	typ	Art der Nachricht die verschickt werden soll (Editbuffer Request, SysEx Nachricht) : Timeout (Zeit bis nächster Dump der Kette abgearbeitet wird : Flag welches Anzeigt ob diesem Dump ein weiterer folgt
unsigned char	eb_convert	numerischer Bezeichner der Konvertierungsfunktion
unsigned char	wrong_header_length	Die Länge des vom Synthesizer zurückgegebenen Headers
unsigned char	edit_buffer_length	Die Länge des Editbuffers
unsigned char	length	Länge der im Speicher folgenden Daten
unsigned char	header_length	Länge des Editbuffer Headers der nach dem SysEx String im Speicher steht

Tabelle 4.10: Dump-Struktur

4.4 Parameterstruktur

Im Folgenden wird die Parameterstruktur erklärt die die Informationen zu Erstellung von MIDI Nachrichten speichert. Die Tabelle 4.11 zeigt die Speicherverteilung innerhalb der Struktur. Das erste Bit einer Variablen ist immer 0 da per MIDI nur 7 Bit Dateninformationen übermittelt werden können, andernfalls wird das Byte als Statusbyte interpretiert. In diesem Abschnitt werden außerdem die Bedeutungen der Einstellungen der einzelnen Bytes beschrieben. Grundsätzlich unterstützt die MIDIbox folgende MIDI Nachrichten: „Note On/Off“, „Aftertouch Polyphon/Channel Pressure“, „Pitchbend“, „Control Change“, „RPN“ und „NRPN“. Außerdem ünterstützt die MIDIbox das Versenden von MIDI SysEx Nachrichten und Editbuffers (siehe dazu Abschitt 4.5 und 4.7).

Datentyp	Feldname	Aufgabe
char[10]	name	einzeiliger Name der auf dem Display angezeigt wird
unsigned char	messageChannel	enthält sowohl den MIDI Kanal auf dem die Nachricht versendet werden soll, als auch in den Nachrichtentyp (im höher wertigen Nibble)
unsigned char	modeSelect	In diesem Byte wird die Art der Controllerbewegung gespeichert (z.B. absolut, relativ...)
unsigned char[2]	number	Controller, Key, NRPN Nummer bzw. falls die Nachricht eine SysEx Nachricht ist zeigen diese Variablen die SysEx Struktur 4.9 im dynamischen Speicher
unsigned char[2]	valueMin	untere Schranke des Parameters
unsigned char[2]	valueMax	obere Schranke des Parameters
unsigned char[2]	value	Wert des Parameters
unsigned char	scale	Skalierungsfaktor, um welche der aktuelle Wert inkrementiert oder dekrementiert werden kann
unsigned int	edit	Flag- und Adressvariable zum Erfragen des Editbuffers
unsigned int	sswitch_addr	Adresse von Parameterwertnamen im dynamischen Speicher

Tabelle 4.11: Aufbau der Parameterstruktur

Die Variable *name* ist 10 Byte lang und speichert den Namen des Parameters.

Die Variable *messageChannel* (siehe 4.12) ist in zwei Teile geteilt. Der erste Teil legt den MIDI Nachrichtentyp fest, der bei der Bedienung des entsprechenden Encoders in der Parameterebene gesendet werden soll. Der zweite Teil, legt den MIDI Kanal fest auf dem die MIDI Nachricht gesendet werden soll.

Die Variable *modeSelect* (siehe Tabelle 4.13) ist mit relativ vielen Einstellungen

7 6 5 4 3 2 1 0	
0 X X X 0 0 0 0	*messageChannel*
Bitkombination	MIDI Nachrichtentyp
0 0 0	Off
0 0 1	Note On/Off, Aftertouch, Pitchbend
0 1 0	*** frei ***
0 1 1	Control Change
1 0 0	Program Change
1 0 1	NRPN
1 1 0	RPN
1 1 1	SysEx und Editbuffer

Tabelle 4.12: *messageChannel* - MIDI Nachrichtentyp

belegt die jeweils von der Einstellung der Variablen *messageChannel* abhängen. Ist dort als Nachrichtentyp „Note On/Off, Aftertouch, Pitchbend“ gewählt, steht in *modeSelect* die genaue Unterscheidung zwischen den Nachrichtentypen. „Note On/Off“, „Aftertouch“, „Pitchbend“ besitzen keine speziellen Controller Modi und können deshalb zusammengefasst werden. Wenn „Control Change“, „NRPN“ oder „RPN“ als Nachrichtentyp gewählt ist, bestimmt *modeSelect* einen der Controller Modi „Absolute“, „Relativ1“(2er Komplement), „Relativ2“(Binär Offset), „Relativ3“(Vorzeichenbit) oder „Inc/Dec“. Bei dem ausgewählten Nachrichtentyp „Program Change“ wird mit *modeSelect* eingestellt ob einer „Program Change“ Nachricht eine „Bankselect“ Nachricht vorausgehen soll. Das Bit 3 von *modeSelect* (siehe Tabelle 4.14) legt fest ob die Encoderdrehfunktion oder die Encodertastenfunktion ausgewertet wird. Bit 2 von *modeSelect* hat verschiedene Bedeutungen, abhängig von den zuvor gesetzten Bits. Ist dir Encoderdrehfunktion aktiv, gibt an, ob die Beschleunigung „Acceleration“ eingeschaltet ist. Bei Encodertasten im „Absolute“ Controller Mode steht Bit 2 für „Toggle Off/On“. Bei „Toggle Off“ wird beim Drücken der Wert aus $valueMax$ und beim Loslassen direkt der Wert aus $valueMin$ gesendet. Bei „Toggle On“ wird beim Loslassen keine Nachricht gesendet, sondern erst beim erneuten Drücken. So kann man die Encodertaste als Schalter oder Taster verwenden. In den relativen Controller Modi legt dieses Bit fest ob beim Drücken eine Incrementer oder eine Decrementer MIDI Nachricht versendet werden soll. Mit Bit 1 von *modeSelect* stellt man die Auflösung der gewünschten MIDI Nachricht ein. Das Bit 0 zeigt an, ob der Wert in $valueMin$ ein negatives Vorzeichen hat. Diese Angabe ist wichtig beim Anzeigen von negativen Wertebereichen.

In *number* wird entsprechend des MIDI Nachrichtentyps die „NRPN/RPN Nummer“, „Control Change Nummer“, „Notennummer“ oder die „Bankselect Nummer“ gespeichert. $valueMin$ speichert die untere Grenze des anzuzeigenden Wertebereichs und $valueMax$ die obere Grenze. *value* speichert den Wert des Parameters. 7 Bit Werte stehen immer Byte[0], bei 14 Bit Werten steht das MSB in Byte[0] und das

7 6 5 4 3 2 1 0	
0 X X X 0 0 0 0	*modeSelect*
Bitkombination	Note On/Off, Aftertouch, Pitchbend
0 0 0	Note On/Off
0 0 1	Aftertouch Channel Pressure
0 1 0	Aftertouch Polyphon Pressure
0 1 1	Pitchbend
	Controller Mode bei NRPN, RPN und Control Change
0 0 0	Absolute
0 0 1	Relativ1 (2er Komplement)
0 1 0	Relativ2 (Binär Offset)
0 1 1	Relativ3 (Vorzeichenbit)
1 0 0	Inc/Dec
	Bankselect bei Program Change
0 0 0	senden ohne Bankselect
0 0 1	senden mit Bankselect (MSB)
0 1 0	senden mit Bankselect (LSB)
0 1 1	senden mit Bankselect (MSB und LSB)

Tabelle 4.13: *modeSelect* - Bitkombinationen Teil 1

LSB in Byte[1] des Arrays. Die Variable *scale* enthält den Skalierungsfaktor. Sollen SysEx Nachrichten versendet werden, wird in *number* die Adresse zur der entsprechenden SysEx Struktur (siehe Abschnitt 4.3) gespeichert.

Zu dem Thema Wertebereiche haben wir die Begrifflichkeiten MIDI Skala und User Skala eingeführt. Zum einen gibt es die MIDI Skala; das sind die Zahlenwerte, die real über die MIDI Schnittstelle gesendet werden. Diese Skala hat die Grenzen 0 bis 127 bei 7 Bit und 0 bis 16383 bei 14 Bit. Zum anderen gibt es die User Skala, deren Grenzen vom Benutzer im Javaprogramm (siehe Literaturverzeichnis [3]) eingestellt werden, und auch negativ sein können. Diese Werte werden im Display der MIDIbox angezeigt und sollen der Darstellung am zu steuernden Gerät (Synthesizer...) entsprechen. Zwischen diesen beiden Skalen muss umgerechnet werden, da es z.B. unmöglich ist, über die MIDI Schnittstelle negative Werte zu versenden. Hierbei gibt es einige Einschränkungen:

- Der Wertebereich in der User Skala muss kleiner oder gleich dem der MIDI Skala sein.

- Das User Maximum/Minimum kann höchstens MIDI Maximum/Minimum geteilt durch Scale Faktor betragen.

Je nach Einstellung in der Variable *scale* eines Parameters wird die Umrechnung zwischen MIDI Skala und User Skala anders vorgenommen. Bei allen Modi gilt: ist das User Minimum Null oder positiv, so ist die User Null gleich der MIDI Null.

7 6 5 4 3 2 1 0	
0 0 0 0 X 0 0 0	*modeSelect*
Bitkombination	Encoder oder Encodertaste
0	Encoder aktiv
1	Encodertaste aktiv
0 0 0 0 0 X 0 0	*modeSelect*
Bitkombination	Acceleration bei Encoder
0	Acceleration nicht aktiv
1	Acceleration aktiv
	Toggle Off/On bei Encodertaste und Controller Mode „Absolute“
0	Toggle Off
1	Toggle On
	Inc/Dec bei Encodertaste und Controller Mode „Relativ1“, „Relativ2“, „Relativ3“, „Inc/Dec“
0	Incrementer Mode
1	Decrementer Mode
0 0 0 0 0 0 X 0	*modeSelect*
Bitkombination	7 Bit oder 14 Bit Auflösung
0	7 Bit
1	14 Bit
0 0 0 0 0 0 0 X	*modeSelect*
Bitkombination	Ist der Wert in $valueMin$ negativ?
0	$valueMin$ positiv
1	$valueMin$ negativ

Tabelle 4.14: *modeSelect* - Bitkombinationen Teil 2

Ist das User Minimum negativ, entspricht die User Null dem MIDI Wert 64 (bei 7 Bit; 8192 bei 14 Bit), der sich um die positiven und negative Werte erhöht und verringert.

scale=0 hat also eine „Auffächerung“ der User Skala auf die MIDI Skala zur Folge, in allen anderen Modi beschreibt der Wert in *scale* die Schrittweite in der MIDI Skala pro User Wertesprung. Siehe hierzu Tabelle 4.15

In der Integer Variablen *edit* sind 12 Bit für die Adresse reserviert in der der Initalwert des Parameters im eingelesenen Editbuffer ausgelesen wird. Das Bit 6 legt fest ob bei einer Parameteränderung der Editbuffer an den Synthesizer gesendet werden soll. Bit 7 bestimmt ob im dynamischen Speicher Text Strings abgelegt sind, die alternativ zum eingestellten Wert auf dem Display angezeigt werden (siehe Abschnitt 4.9). Wenn das $Sswitch$ Bit gesetzt ist folgt in der Integer Variable $sswitch_addr$, die Adresse im dynamischen Speicher ab der die anzuzeigenden Text

scale=0:	Der gesamte MIDI Bereich wird auf die User Skala gemapped. Das bedeutet, dass beim Drehen eines Encoders kontinuierlich alle Werte der MIDI Skala gesendet werden, aber die Anzeige nur an bestimmten Stellen auf den nächsten User Wert umspringt.
scale=1:	Es gibt genau so viele MIDI Werte wie User Werte. Beim Drehen eines Encoders entspricht die Werteänderung der User Skala der der MIDI Skala.
scale=2-64:	Die User Skala verändert sich mit jedem Drehschritt um + -1, die MIDI Skala verändert sich mit jedem Drehschritt um den in *scale* eingestelleten Wert. Dies ist sinvoll, um Wertebereiche in der MIDI Skala zu überspringen, und weniger User Werte anzusteuern.

Tabelle 4.15: *scale* - Unterschiedliche Skalierungseinstellungen

Bits	*edit*
12	Editbufferadresse des Werts
1	Sende Editbuffer
1	Sswitch soll verwendet werden

Tabelle 4.16: *edit* - Bitaufteilung

Strings abgelegt sind.

Wert	NRPN Nachricht
messageChannel	(bitweise)
0	***MIDI bedingt frei
1 0 1	NRPN
0 0 0 0	MIDI Kanal 0
modeSelect	(bitweise)
0	***MIDI bedingt frei
0 0 0	Absolut Modus
0	Encoder aktiv
1	Bescheunigung aktiv
1	14 Bit
1	valueMin ist negativ
number	(byteweise)
0x01	NRPN NR MSB (number[0])
0x04	NRPN NR LSB (number[1])
valueMax	(byteweise)
0x0F	valueMax[0]
0x50	valueMax[1]
valueMin	(byteweise)
0x0F	valueMin[0]
0x50	valueMin[1]

Tabelle 4.17: Beispiel für die Bitkombination einer 14 Bit NRPN Nachricht mit NRPN Nr. 132 auf MIDI Kanal 1, betrieben als absoluter Drehregler in den Grenzen -2000 und +2000, Beschleunigung aktiviert

Wert	Control Change Nachricht
messageChannel	(bitweise)
0	***MIDI bedingt frei
0 1 1	Control Change
0 1 0 1	MIDI Kanal 5
modeSelect	(bitweise)
0	***MIDI bedingt frei
0 0 0	Absolut Modus
1	Encodertaster aktiv
1	Toggle On (Schalter)
0	7 Bit
0	valueMin ist positiv
number	(byteweise)
0x12	CC Nr. (number[0])
valueMax	(byteweise)
0x10	valueMax[0]
valueMin	(byteweise)
0x00	valueMin[0]

Tabelle 4.18: Beispiel für die Bitkombination einer 7 Bit Control Change Nachricht mit der CC Nr. 18 auf MIDI Kanal 6, betrieben als absoluter Schalter zwischen den Grenzen 0 und 16

4.5 MIDI Empfang

Das MIDI Empfangsmodul implementiert drei wichtige Funktionen der MIDIbox. Die erste und auch einfachste ist ein MIDI Through. Dieser gewährleistet, dass alle ankommenden MIDI Bytes, die nicht für die Box bestimmt sind, am MIDI Out wieder ausgegeben werden. Da die Box nur SysEx Nachrichten verarbeitet, können alle anderen ohne weitere Analyse verschickt werden. Um auch systemexklusive MIDI Nachrichten, die nicht für die Box bestimmt sind, weiterzuleiten, bedarf es einer Analyse des Inhaltes und einer Zwischenspeicherung der empfangenen Bytes bis eine Entscheidung getroffen werden kann. Fällt diese negativ aus, werden die gecachten Daten und alle weiteren Daten bis zum Beginn der nächsten SysEx Nachricht auf dem MIDI Out ausgegeben.

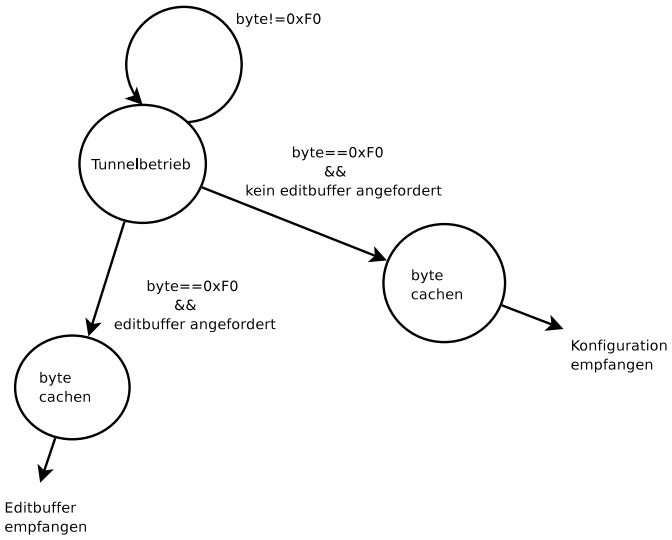

Abbildung 4.37: Finite State Digramm das die Arbeit des MIDI Throughs veranschaulicht. Die eingehenden Kanten am State "Tunnelbetrieb" sollen das Ausgeben der eingehenden Nachrichten am MIDI Out symbolisieren.

Die gesamte Verarbeitung der MIDI Nachrichten findet im Event Handler *MPROC_NotifyReceivedByte(unsigned char byte)* statt. Dieser ist als deterministischer finiter Automat implementiert (siehe Abbildung 4.37). Der Ausgangszustand, zu dem immer zurück gekehrt wird, ist der Tunnelbetrieb. Die beiden anderen Möglichkeiten und ihre einzelnen Zustände werden im Folgenden beschrieben.

4.5.1 Programmierung der MIDIbox mittels Java Software

Zur Programmierung der Box wird ein einfaches Speicherbeschreibungsprotokoll verwendet. Dieses ermöglicht der Programmiersoftware das Eprom der MIDIbox wahlfrei zu beschreiben. Zur Verfügung stehen, wie im Kapitel Speicheraufteilung beschrieben, 768 Kilobyte statischer und dynamischer Speicher. Um eine Programmierung durchführen zu können, muss die Java Software die Aufteilung des Speichers genau kennen. Die Programmiersoftware ist auch dazu angehalten, immer ein

komplettes Device mit allen Modulen, Parametern und dem zum Device gehörenden dynamischen Speicher zu übertragen, da sonst Speicherinkonsistenzen auftreten könnten. So wäre es beispielsweise vorstellbar, nur einen Teil der Parameter, Module und des dynamischen Speichers zu übertragen, wenn die Programmiersoftware Kenntnis[2] über den aktuellen Speicherinhalt besäße. Dies würde aber ein unvertretbar aufwendiges (und zum Schluss auch langsames) Protokoll oder das Nutzen nur eines Programmiercomputers voraussetzen.

Feld	Inhalt	Aufgabe
0	0xF0	SysEx Start
1	0x55	Hersellercode
2	0x4d	Hersellercode (erweitert 1)
3	0x42	Hersellercode (erweitert 2)
4	0x00-0x7F	Empfängercore
5	0x00-0x7F	Funktion
6	0x00-0x7F	Protokollversion MAJOR
7	0x00-0x7F	Protokollversion MINOR
8	0x00-0x7F	Speicheradresse1
9	0x00-0x7F	Speicheradresse2
10	0x00-0x7F	Speicheradresse3
11	n*0x00-0x7F	Daten
12	0xF7	SysEx Ende

Tabelle 4.19: Nachricht von der Java Software zur MIDIbox

Das Protokoll besteht aus einer Speicheranfrage zu sehen in Tabelle 4.19 und einer Antwort zu sehen in Tabelle 4.20. Das Java Programm schickt ohne vorherigen Handshake die Anfrage inklusiv der zu speichernden Daten. Die Anfrage besteht ihrerseits aus einigen Feldern, die es der Box erlauben, festzustellen, dass dies eine an sie gerichtete Speicheranfrage ist (Herstellercodes, Empfänger-Core und Funktion). Des Weiteren enthält sie Versionsnummern, die es der Box erlauben, festzustellen, ob die Versionen der Speicheraufteilung gleich sind. Bei einem Konflikt der *MINOR* Versionsnummer werden die Daten trotzdem gespeichert, aber der Sender davon in Kenntnis gesetzt (siehe Tabelle 4.21). Bei einem *MAJOR* Versionskonflikt wird die Nachricht verworfen und der Sender mit der entsprechenden Antwort darüber informiert.

Die drei Adressfelder im Protokoll bedürfen einer genaueren Erklärung (siehe auch Abbildung 4.38. Das erste Adressbyte enthält sowohl die fünf höchstwertigen Bits der Adresse als auch die beiden höchsten Bits der beiden niederwertigen Adress-

[2]Dies ist mit einer Ausnahme jedoch meist nicht der Fall. Wenn der Benutzer gerade das komplette Device übertragen hat, ist es der Software, solang sie davon ausgehen kann, dass das Gerät nicht von einer anderen Instanz überschrieben wurde oder sogar vom Nutzer ausgetauscht wurde, möglich nur Teile der Konfiguration zu übertragen. Dies kann beim Ändern nur eines Parameters erheblich Zeit sparen und erlaubt dem Nutzer ein Online Update und somit auch Testfunktionalität.

Feld	Inhalt	Aufgabe
0	0xF0	SysEx Start
1	0x54	Hersellercode
2	0x00-0x7F	Antwortcode (siehe Tabelle 4.21)
3	0x00-0x7F	Checksum
4	0xF7	SysEx Ende

Tabelle 4.20: Antwort der MIDIbox auf eintreffende Nachrichten

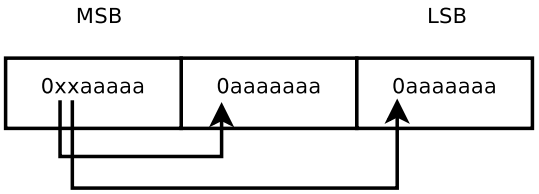

Abbildung 4.38: Aufteilung der Adressbits für die Programmierung der Software.

bytes. Dies ist notwendig, da innerhalb von SysEx Nachrichten das 8. Bit immer 0 sein muss. Aus den 25 Adressbits ergibt sich die Möglichkeit, bis zu 2 Megabyte zu adressieren. Diese 2 Megabyte sind zur Zeit noch nicht ausgeschöpft, bieten aber die Möglichkeit, den dynamischen Speicher der Box auf 1,5 Megabyte zu vergrößern[3].

Code	Bedeutung
0x00	OK
0x01	Daten inkorrekt
0x02	kein Speicherplatz vohanden bzw. ausserhalb des adressierbaren Speicherbereiches
0x03	Protokollversionen stimmen nicht überein (MINOR)
0x04	Protokollversionen stimmen nicht überein (MAJOR)

Tabelle 4.21: Antwort-Codes

Nach Übermittlung der Adressbytes folgen die im Eprom zu speichernden Daten. Deren Länge ist auf maximal 64 Bytes begrenzt, um ein Überlaufen des MIDI In Puffers zu vermeiden. Falls ein zu schreibender Speicherbereich größer als 64 Bytes ist, muss die Speicherung in mehrere Blöcke zerlegt werden. Grund für die 64-Byte Grenze ist zum einen, dass eine MIDI SysEx Nachricht nicht länger als 256 Byte sein sollte als auch, dass sich 64 Byte Blöcke mit maximaler Geschwindigkeit in die BankSticks übertragen lassen. Eine Quittierung der Daten bietet sich zudem auch nach 64 Bytes an, da sich zu diesem Zeitpunkt die Checksumme der in den

[3]Der versierte Nutzer kann, indem er neue Bankstickplatinen in das Gerät implementiert (siehe 3.3.4) und an wenigen Stellen im Quellcode der Box Änderungen vornimmt (siehe 4.1), den Speicher erweitern.

Speicher übertragenen Daten schnell[4] errechnen lässt. Da sich die zu übertragenden Speicherblöcke nicht immer in ganzzahlige Vielfache von 64 Bytes aufteilen lassen, sind auch kleinere Nachrichten erlaubt. Beendet wird die Übertragung durch Senden des SysEx Endbytes (0xF7). Auf dieses folgend sendet die MIDIbox eine Quittung, zu sehen in Tabelle 4.20.

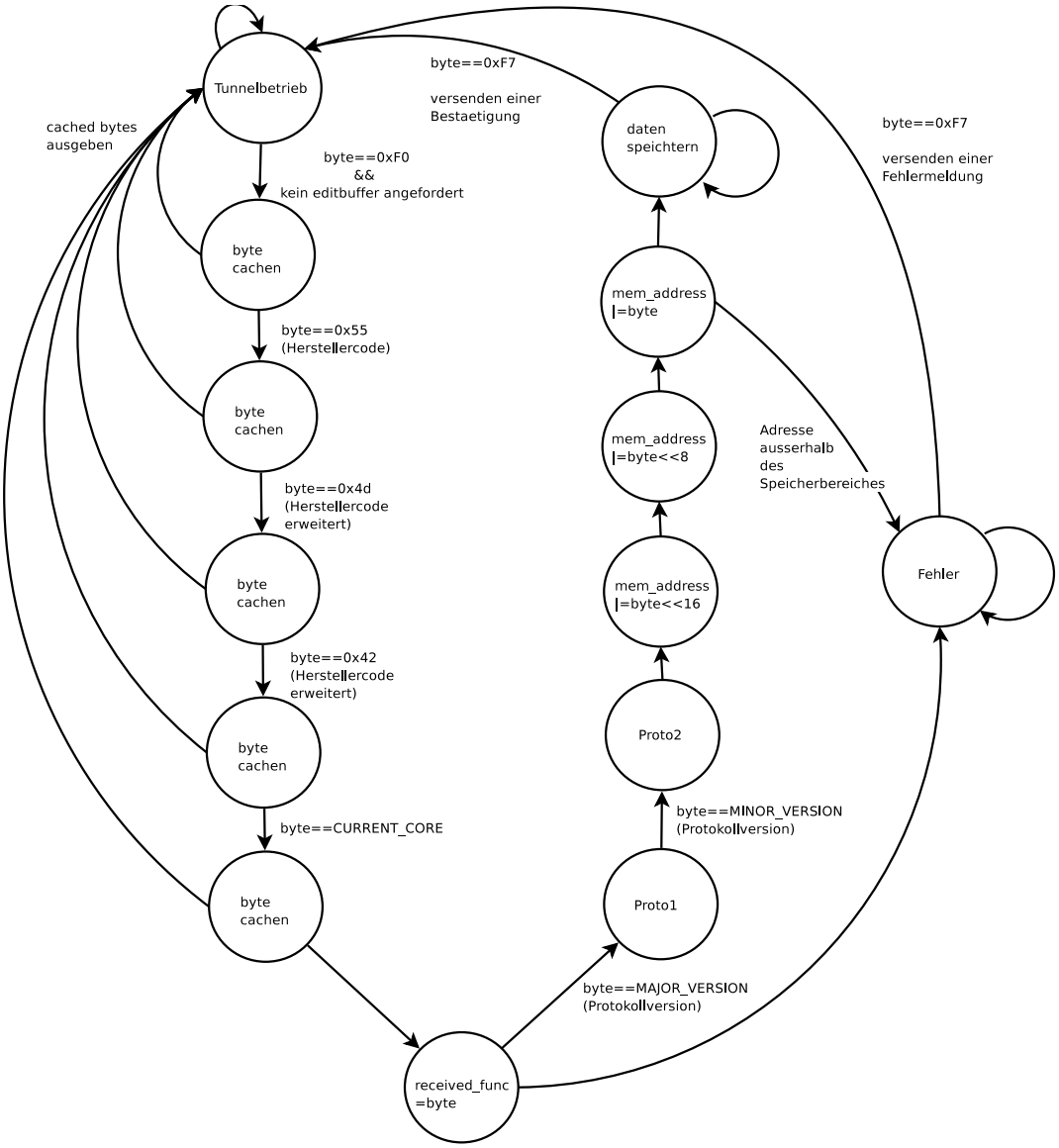

Abbildung 4.39: Finite State Digramm das die Erkennung und Speicherung neuer Programme visualisiert.

Abschließend soll grob der Aufbau der Speicherfunktion anhand der Abbildung 4.39 erläutert werden. Als deterministischer finiter Automat implementiert, lassen sich klar die Zustände im Diagramm erkennen. Mit dem Empfang jedes Bytes kann der Zustand geändert werden. Dabei gilt, dass, solange das Protokoll eingehalten ,wird alle empfangenen Daten zwischengespeichert werden. Am Anfang geschieht

[4]Um Speicherfehler zu erkennen, werden die Daten zurück aus dem Eprom in den RAM gelesen, um dort die Checksumme zu bilden. Der Bottleneck dieser Operation ist also das Rücklesen der Daten, welches eine maximale Geschwindigkeit bei zugleich geringem Platzbedarf im RAM bei 64 Byte hat.

dies, um bei einer möglichen Abweichung vom Protokoll die MIDI Through Funktionalität zu gewährleisten. Später dient das Caching der Speicherung im Eprom, welche, wenn möglich, in 64 Byte Blöcken statt findet.

4.5.2 Empfangen des Editbuffers

Nachdem eine Editbufferanfrage an einen Synthesizer geschickt wurde, wartet die MIDIbox auf Antwort des Synthesizers. Da als Antwort nur SysEx Nachrichten in Frage kommen, werden andere Nachrichten weiterhin getunnelt. Beim Beginn einer SysEx Nachricht wechselt die MIDIbox in einen Prüfmodus, um zu erkennen, ob die Nachricht vom betreffenden Gerät kommt (Herstellercode). Ist dies der Fall, wird angenommen, dass es sich um den Editbuffer des Gerätes handelt, der, nachdem der Header dieses Buffers abgeschnitten wurde, in den RAM geschrieben wird. Beendet ist die Übertragung durch Empfang einer SysEx End (0xF7) Nachricht.

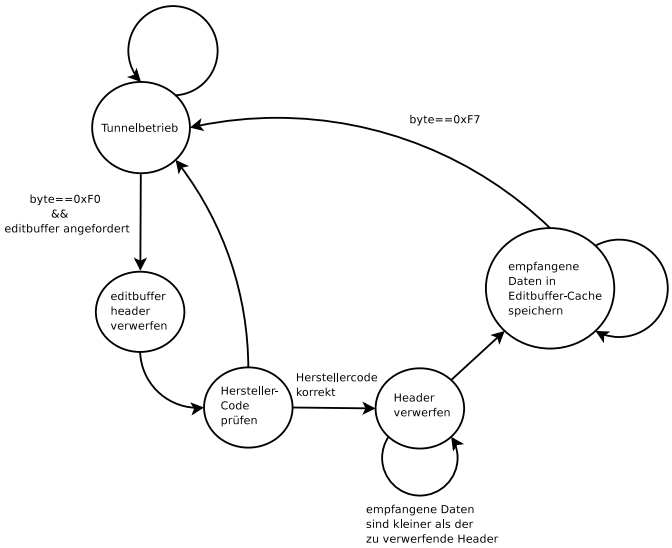

Abbildung 4.40: Finite State Digramm welches das Eintreffen eines Editbuffers zeigt.

4.6 Eingabe und Navigation

Zum Bedienen der *MIDI*box stehen dem Benutzer verschiedene Tasten und Encoder mit Tastenfunktion auf der Frontplatte zur Verfügung (siehe Abschnitt 2.1). Dieser Abschnitt erklärt die Funktionen der Software die zum Abfragen der Eingabemodule ausgeführt werden. Außerdem werden die Funktionen, die für die Navigation in der Menüstruktur (siehe Abschnitt 2.2) aufgerufen werden, beschrieben.

Für die Werteübergabe und Zwischenspeicherung der Encoderwerte benutzen wir ein Integer Array *potis*. In diesem Array sind von *potis*[0] bis *potis*[15] die eingestellten Werte der aktuellen Parameterseite gespeichert. Diese Werte werden in der *writeBar* Funktion für die Display-Anzeige (siehe Kapitel 4.9) und in der *sendMessage* für die Generierung von MIDI Nachrichten verwendet. Verändert werden die Werte ausschließlich in der $DIN\_NotifyToggle$ Funktion und $ENC\_Notify\ Change$ Funktion, hier werden die Werteänderungen unter Berücksichtigung der Skalierung und der Wertegrenzen errechnet. *potis*[16] ist eine Flag Variable, die angibt, welcher der Werte in *potis*[0] bis *potis*[15] verändert wurde, damit die Display Funktionen das Display an der richtigen Stelle aktualisieren können. *potis*[17] ist auch eine Flag Variable, die speichert, welcher der Werte verändert wurde. Jedoch ist diese Angabe für die $DIN\_NotifyToggle$ Funktion wichtig, da sie das Senden von MIDI Nachrichten beim Loslassen einer Encodertaste direkt nach dem Betreten der Parameterebene verhindert.

Die Funktion $DIN\_NotifyToggle$ ist im MIOS enthalten und wird bei einer Änderung der Tasterzustände ausgeführt. Sie übergibt die Variablen *pin* für die gewählte Taste und *pin_value* für seinen Zustand. Beim Drücken wird eine 0 übergeben und beim Loslassen eine 1. Das Diagramm in Abbildung 4.41 zeigt schematisch den Ablauf der $DIN\_NotifyToggle$ Funktion. Beim Aufruf wird entschieden, ob eine Menütaste oder eine Encodertaste geändert wurde. Die Menütasten sind am zweiten Digital In Modul (siehe 3.2.2) an den Pins 52 bis 55 angeschlossen. Die Encodertasten, die ebenfalls am zweiten Digital In Modul angeschlossen sind, haben die Pinnummern 32 bis 47. Grundsätzlich rufen Menütasten erst beim Loslassen eine Funktion auf und Encodertasten schon beim Drücken. Ist eine Menütaste im Modulmodus oder Parametermodus gedrückt, wird in der Variable *knob* ein Bit für diese Taste gesetzt. Wird er losgelassen, wird die Funktion $initModulModus$ aufgerufen. Werden die Menütasten k0 und k1 gleichzeitig gedrückt, wird die Variable *knob* auf 0x10 gesetzt und beim Loslassen wird die Funktion $initDeviceModus$ ausgeführt. Die Navigation und die verwendeten Funktionen werden in Abbildung 4.43 veranschaulicht. Wird eine Encodertaste gedrückt führt die $DIN\_NotifyToggle$ zuerst einen Remapping Algorithmus durch, da die tatsächliche und die logische Nummerierung der Encoder nicht übereinstimmt (siehe dazu Kapitel 3.2.2). Danach folgt eine *switch* Anweisung die unterscheidet in welcher Ebene, in welchem Modus, sich die MIDIbox befindet. In der Deviceebene wird die Variable für das gewählte Device

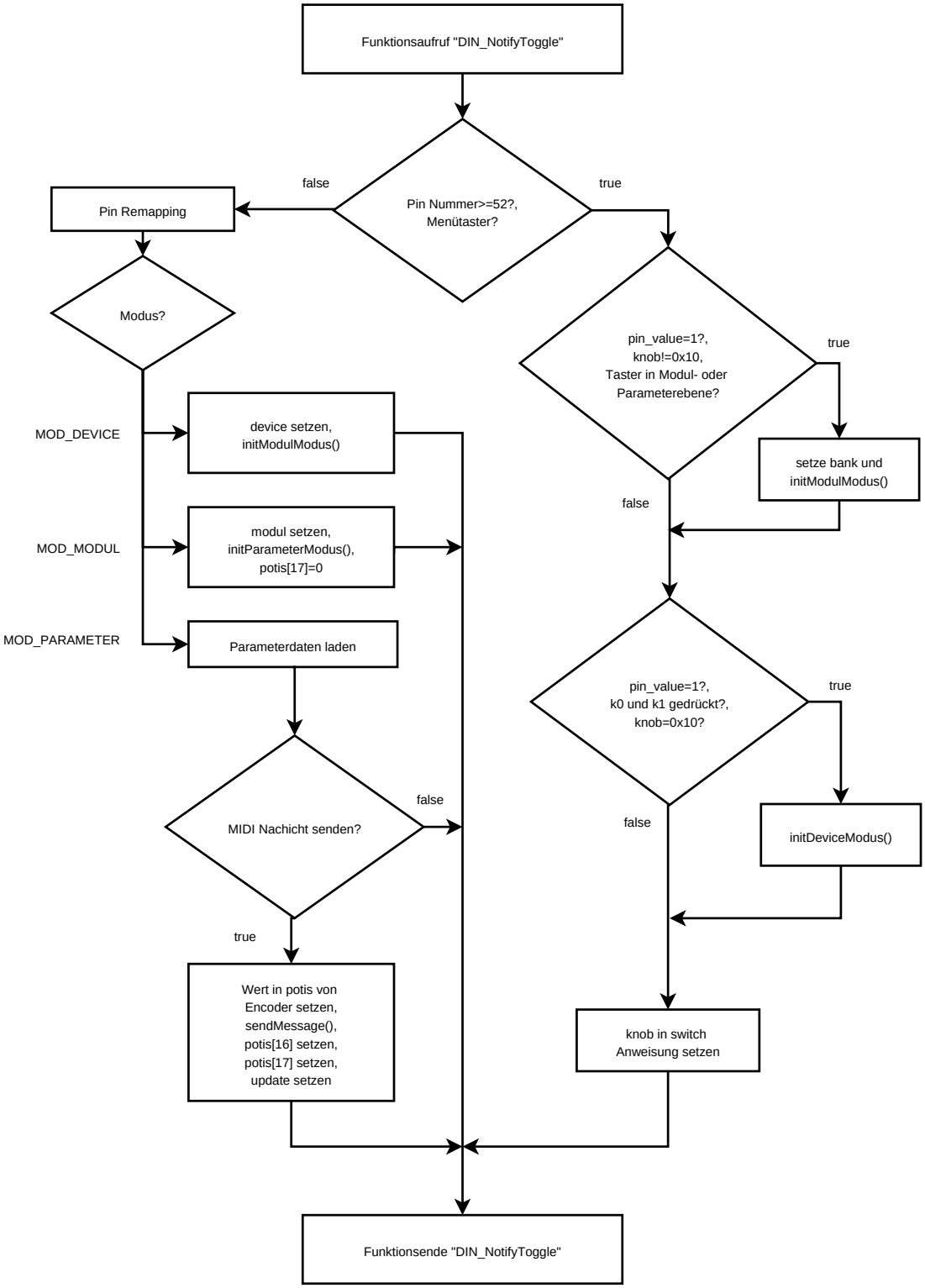

Abbildung 4.41: Diagramm der DIN_NotifyToggle Funktion

gesetzt und die $initModulModus$ Funktion aufgerufen. In der Modulebene wird die die Variable für das gewählte Modul gesetzt und die $initParameterModus$ Funktion aufgerufen. In der Parameterebene wird zuerst die entsprechende Parameterstruktur geladen und gegebenenfalls die $sendMessage$ aufgerufen um eine MIDI Nachricht zu versenden.

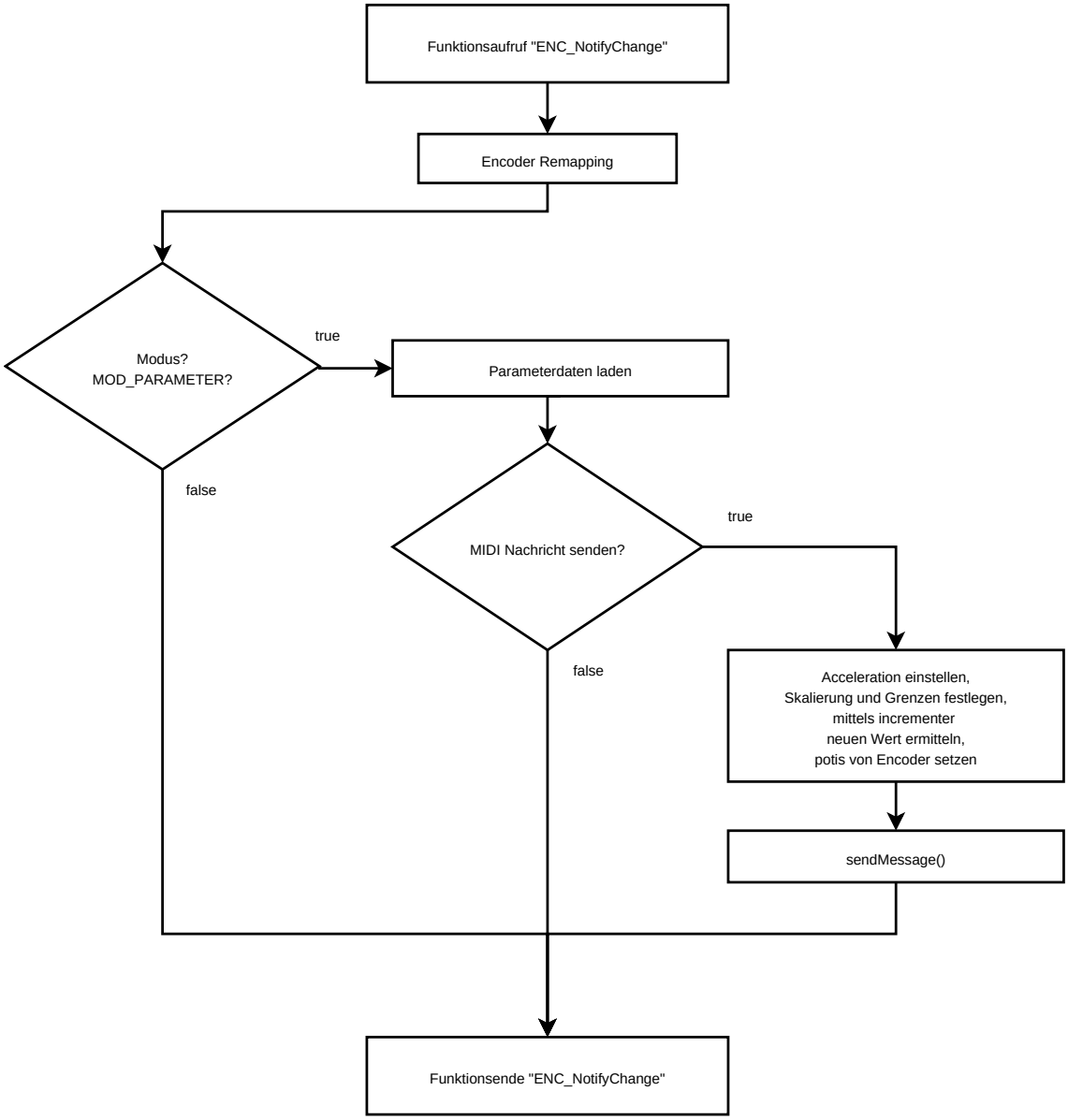

Abbildung 4.42: Diagramm der ENC_NotifyChange

Ebenfalls enthält das MIOS die Funktion $ENC\_NotifyChange$, welche beim Drehen eines Encoders ausgeführt wird (siehe 4.42). Sie übergibt beim Funktionsaufruf die Variablen *encoder*, die für die Nummer des gewählten Encoders steht, und die Variable *incrementer*, die angibt, um wieviel die Position des Encoders verändert wurde. Vorausgesetzt die MIDIbox befindet sich in der Parameterebene, werden nach dem Remapping der Encoder Nummern (siehe dazu Kapitel 3.2.2) die Parameterdaten des gewählten Encoders geladen. Soll eine MIDI Nachricht versendet werden, wird unter Berücksichtigung der Skalierung, der Wertegrenzen und der

gewählten Auflösung mittels des übergebenen *incrementer* die Werteänderung von *potis*[] durchgeführt und die *sendMessage* Funktion aufgerufen, die die programmierte MIDI Nachricht versendet. Veranschaulicht wird die *sendMessage* Funktion in Abbildung 4.46.

Sämtliche Funktionen, die bei der Navigation durch die Menüstruktur verwendet werden, sind von uns entwickelt und programmiert. Das Diagramm in Abbildung 4.43 zeigt, welche Funktionen zwischen den verschiedenen Ebenen ausgeführt werden. Um den Display Funktionen (beschrieben in 4.9) zu signalisieren, dass in eine neue Ebene in der Menüstruktur navigiert wurde, also ein anderer Modus gewählt wurde den das Display anzeigen soll, haben wir die Variable *update* eingeführt. Sie signalisiert den Display Funktionen, was angezeigt werden soll. Das Setzen von *update* geschieht grundsätzlich in den Funktionen $initDeviceModus$, $initModulModus$ und $initParameterModus$. Nach dem Systemstart der MIDIbox wird die Funtion $initDeviceModus$ ausgeführt. Diese initalisiert die LEDs und setzt den Modus auf MOD_DEVICE. Außerdem wird in $initDeviceModus$ die Variable *bank* 0 gesetzt, so dass beim Auswählen eines Devices immer die Modulbank 0 angezeigt wird. Durch Drücken einer Encodertaste im Devicemodus wird die Funktion $initModulModus$ ausgeführt. In der $initModulModus$ werden mit $loadDevice$ die spezifischen Devicedaten geladen, die zum Anfordern und Empfangen des Editbuffers des Synthesizers notwendig sind. Wenn der Editbuffer geladen werden soll, wird dies ausgeführt und der Modus wird auf MOD_MODUL gesetzt. Im Modulmodus kann der Benutzer mit den Menütasten und Encodertasten navigieren. Drückt er eine der vier Menütasten um in eine andere Modulbank zu gelangen, wird die entsprechende Bank in Variable *bank* gesetzt und erneut $initModulModus$ ausgeführt. Drückt er die Menütaster k0 und k1 zusammen, wird $initDeviceModus$ ausgeführt. Um auch Synthesizer mit mehreren Editbuffers bedienen zu können, gibt es die Möglichkeit auch beim Auswählen eines einzelnen Moduls den Editbuffer anzufordern. Ob und mit welcher MIDI Nachricht der Editbuffer angefordert wird, kann der Benutzer mit der Java Software (siehe Literaturverzeichnis [3]) einstellen. Drückt der Benutzer im Modulmodus eine Encodertaste zum Auswählen eines Moduls, wird die $initParameterModus$ Funktion ausgeführt. In der $initParameterModus$ werden die spezifischen Moduldaten geladen und gegebenenfalls der Editbuffer angefordert. Danach werden mit der Funktion $loadParameterPage$ alle Parameterstrukturen der anzuzeigenden Parameter geladen und in den Arrays $parametersleft$ und $parametersright$ gespeichert. Der Grund, dass wir die Parameter auf zwei Arrays aufgeteilt haben liegt darin, dass der Compiler keine Arrays größer als 256 Byte zulässt. Da eine Parameterstruktur aber eine Größe von 25 Byte hat, passen nicht alle sechzehn Parameterstrukturen in ein Array. Im Parametermodus $MOD\_PARAMETER$ kann der Benutzer mit den Menütasten zurück in den Modulmodus oder Devicemodus springen. Drückt er

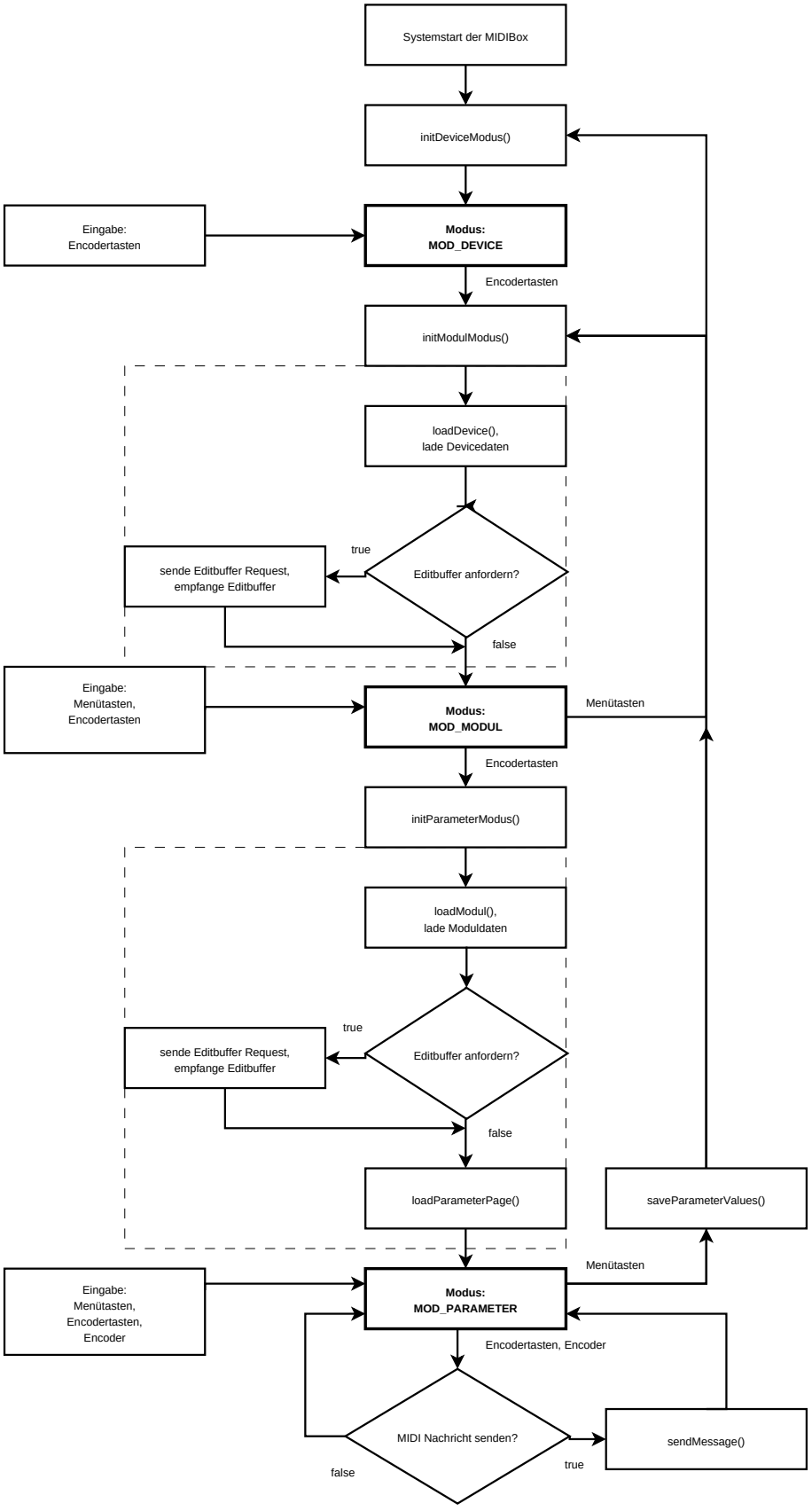

Abbildung 4.43: Verwendete Funktionen bei der Navigation

eine einzelne Menütaste, wird die Variable *bank* gesetzt und die *initModulModus* Funktion wird ausgeführt. Drückt er die Menütaster k0 und k1 zusammen, wird, wie Modulmodus, *initDeviceModus* ausgeführt. Die Encodertasten und die Encoder haben in diesem Modus keine Navigationsfunktion. Ist der entsprechende Encoder mit einer MIDI Nachricht programmiert, wird bei Bedienung des Encoders die *sendMessage* Funktion (siehe Abbildung 4.46) ausgeführt und die gewünschte MIDI Nachricht gesendet.

4.7 MIDI Messages senden

Zur Generierung der erwünschten MIDI Nachrichten (siehe 4.44 und 4.45, entnommen aus [3]) werden die vom Benutzer mit der Java Software am PC erstellten Parameterdatensätze aus den Eprom Speicherplatinen durch die Funktion *loadParameter Page* (siehe Abschnitt 4.6) geladen und durch die Funktion *sendMessage* decodiert und gesendet. Abbildung 4.46 zeigt den schematischen Ablauf der *sendMessage* Funktion. Über den Aufbau der festgelegten MIDI Nachrichten gab die Internetseite www.zem-collage.de/midi über den Zeitraum des Projekts Auskunft [2].

Nach der Variablendeklaration und dem Laden der aktuellen Parameterdaten läuft das Programm in eine *Do-While* Schleife. In dieser wird eine *switch* Anweisung ausgeführt die das Byte *messageChannel* des gewählten Parameters ausgewertet (siehe hierzu Abschnitt 4.4). Für die verschiedenen unterstützten MIDI Nachrichten „Note, Aftertouch, Pitchbend“, „RPN“, „NRPN“, „Control Change“ und „Program Change“ werden, abhängig von der Variablen *modeSelect*, die drei Variablen des lokalen *char evnt*[3] Arrays gesetzt. Am Ende der Schleife werden die drei Byte per MIDI gesendet. Es gibt drei Ausnahmen dass eine MIDI Nachricht nicht aus drei Bytes besteht und zwar bei „Channel Aftertouch“, „Program Change“ und „SysEx“ Messages. In diesen Fällen wird die Message direkt an der Stelle verschickt, an der sie zusammengesetzt wird und die Funktion wird mit dem *return* Befehl beendet. Wenn eine Nachricht aus mehreren Teilnachrichten besteht, wie es zum Beispiel bei einer NRPN Nachricht der Fall ist, merkt sich die *sendMessage* Funktion in der lokalen Variable *sendMessageMemory*, dass noch weitere Nachrichten zu senden sind. Wenn *sendMessageMemory* nicht null ist, wird die Funktion nicht beendet, sondern springt an den Anfang der *Do-While* Schleife und wird wiederholt. Mittels der *sendMessageMemory* Variable erkennt die Funktion beim wiederholten Durchlauf dann, welche Nachricht als nächstes gesendet werden soll. Die Schleife wird so lange wiederholt, bis alle Teilnachrichten einer MIDI Nachricht versendet wurden.

Soll eine SysEx Nachricht versendet werden, wird an der Adresse aus der Variablen *number* plus der Länge der SysEx Struktur der zu versendenden SysEx String, dessen Länge in der SysEx Struktur selbst steht, ausgelesen und in einen SysEx Sendepuffer geschrieben. Nun wird einer synthesizerspezifischen Konvertierungsfunktion der Zeiger auf diesen Puffer, die Stelle an der der Wert eingesetzt werden soll und der Wert selbst übergeben. Hat diese den String konvertiert wird er direkt aus dem Puffer versendet.

Ein Spezialfall von SysEx Nachrichten ist es den gesamten Editbuffer mit modifizierten Werten wieder an den Synthesizer zurückzusenden. Dies wird durch Aufrufen einer Konvertierungsfunktion, die in der Dumpstruktur kodiert ist, realisiert, die den aktuellen Wert konvertiert und an die in der *edit* Variablen der Parameterstruktur enthaltenen Adresse im Editbuffer schreibt. Ist der Editbuffer nun aktualisiert, wird er nach Vorausgehen eines neuen Headers an den Synthesizer gesendet.

ENC / SWITCH	MIDI DATA TYPE	MIDI CHANNEL	PARA. NAME	VAL. 1
ENCODER	**Off**			
SWITCH				
ENCODER	**PROGRAM CHANGE**	1 – 16	Off, Bank Sel MSB	Off, Bank Sel LSB
ENCODER	**CONTROLCHANGE**	1 – 16	CC 0-127	Min.Val.: -64 - 127 -8192 - 16383
SWITCH		1 – 16	CC 0-127	On Val.: -64 - 127 -8192 - 16383
ENCODER	**NRPN/RPN**	1 – 16	NRPN-Nr.	Min.Val.: -64 - 127 -8192 - 16383
SWITCH		1 – 16	NRPN-Nr.	On Val.: -64 - 127 -8192 - 16383
SWITCH		1 – 16	Note Nr.	On Val.: Velocity fixed
ENCODER	**AFTERTOUCH**	1 – 16	Key Nr. 0 - 127, All (All=Chan.Aftertouch)	Min.Val.: -64 - 127
SWITCH		1 – 16	Key Nr. 0 - 127, All (All=Chan.Aftertouch)	On Val.: -64 - 127
ENCODER	**PITCHBEND**	1 – 16	Range	Range: 0 - 127 -8192 - 16383
ENCODER	**SYSEX**	1 – 16	Position 1 + 2	Position 1 , 2: 1 - 15
SWITCH		1 – 16	Position 1 + 2	Position 1 , 2: 1 - 15
ENCODER	**EDITBUFFER**	1 – 16	Position	Position: 1 - 15
SWITCH		1 – 16	Position	Position: 1 - 15

Abbildung 4.44: Parametertabelle Teil 1

ENC / SWITCH	VAL. 2	CTL. MODE	NAME	SCALING	ACCEL.	CONV.	TABLE	HEADER
ENCODER								
SWITCH								
ENCODER	Off, 0 - 127	Abs	X					
ENCODER	Max.Val.: 0 - 127 0 - 16383	Abs, Abs (14) Rel1, Rel2, Rel3 Rel1, Rel2, Rel3 (14) INC / DEC	X	0, 1, 2, 4, 8, 16, 32, 64	X		X	
SWITCH	Off Val.: 0 - 127 0 - 16383	Abs, Abs (14) Rel1, Rel2, Rel3 Rel1, Rel2, Rel3 (14) Toggle On / Off INC / DEC	X					
ENCODER	Max.Val.: 0 - 127 0 - 16383	Abs, Abs (14) Rel1, Rel2, Rel3 Rel1, Rel2, Rel3 (14) INC / DEC	X	0, 1, 2, 4, 8, 16, 32, 64	X			
SWITCH	Off Val.: 0 - 127 0 - 16383	Abs, Abs (14) Rel1, Rel2, Rel3 Rel1, Rel2, Rel3 (14) Toggle On / Off INC / DEC	X					
SWITCH		Abs	X					
ENCODER	Max.Val.: 0 - 127	Abs	X					
SWITCH	Off Val.: 0 - 127	Abs	X					
ENCODER		Abs, Abs (14)	X	0, 1, 2, 4, 8, 16, 32, 64	X			
ENCODER	Min: / Max: -64 - 127 / 0 - 127	Abs, Abs (14)	X	0, 1, 2, 4, 8, 16, 32, 64	X	0 - 255	X	15 Byte (Hex)
SWITCH	Min: / Max: -64 - 127 / 0 - 127	Toggle On / Off	X			0 - 255	X	15 Byte (Hex)
ENCODER	Min: / Max: -64 - 127 / 0 - 127	Abs	X	0, 1, 2, 4, 8, 16, 32, 64	X	0 - 255	X	
SWITCH	Min: / Max: -64 - 127 / 0 - 127	Abs	X			0 - 255	X	

Abbildung 4.45: Parametertabelle Teil 2

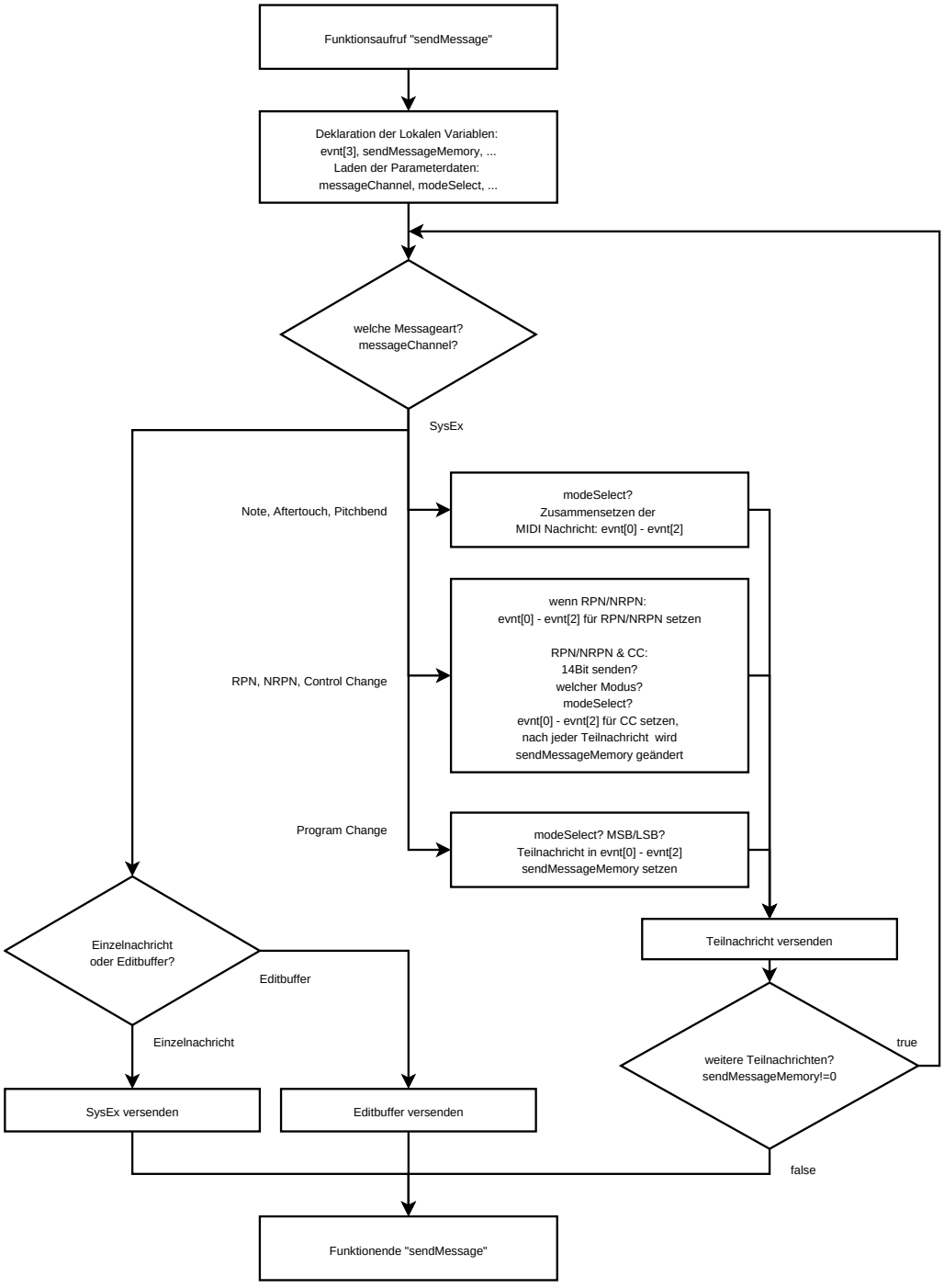

Abbildung 4.46: Diagramm der sendMessage Funktion

Die folgende Tabelle zeigt eine 14 Bit NRPN Nachricht mit der NRPN Nummer 132 und dem Wert 735 auf MIDI Kanal 0. NRPN Nachrichten sind auf der Basis von Control Change Nachrichten aufgebaut, sie haben auch das Statusbyte 0xBx. Das x steht für den MIDI Kanal.

MIDI Nachricht	Status	Daten1	Daten2	Beschreibung
NRPN MSB	0xB0	0x63	0x01	Datenbyte1 ist die Control Change Nummer für NRPN MSB Nachrichten. Datenbyte2 enthält die oberen 7Bit der 14Bit NRPN Nummer
NRPN LSB	0xB0	0x62	0x04	Datenbyte1 ist die Control Change Nummer für NRPN LSB Nachrichten. Datenbyte2 enthält die unteren 7Bit der 14Bit NRPN Nummer
DATA ENTRY MSB	0xB0	0x06	0x05	Datenbyte1 ist die Control Change Nummer für DATA ENTRY MSB Nachrichten. Datenbyte2 enthält die oberen 7Bit des 14Bit Wertes
DATA ENTRY LSB	0xB0	0x26	0x5F	Datenbyte1 ist die Control Change Nummer für DATA ENTRY LSB Nachrichten. Datenbyte2 enthält die unteren 7Bit des 14Bit Wertes

Tabelle 4.22: Beispiel für eine 14 Bit NRPN Nachricht

4.8 Display-Tunnel

Die Verwendung zweier Displays in unserem MIDI Controller ist nicht ganz unproblematisch. Die MIDIbox Hardware Plattform sieht nur den Anschluss eines GLCDs an ein Core Modul vor, was aus Performancegründen auch sehr sinnvoll ist. Also muss man für ein zweites Display auch ein zweites Core Modul verwenden, was dann die Frage der Kommunikation zwischen den beiden Cores aufwirft. Die einfachste Möglichkeit ist, sie über die MIDI Schnittstelle kommunizieren zu lassen, was aber wegen der begrenzten Zahl an MIDI In/Out Ports nur unidirektional funktioniert. Das bedeutet, alle Bedienelemente müssen an das erste Core Modul angeschlossen sein und dort müssen auch alle Ein- und Ausgaben verarbeitet werden. Das erste Core Modul gibt dann alle Display Ausgaben, die für das zweite Display bestimmt sind über MIDI SysEx an den zweiten Core weiter. Dieser hat dann die Aufgabe, alle SysEx Nachrichten, die für ihn bestimmt sind, auf dem Display anzuzeigen, und alle anderen (an Synthesizer oder Computer) an den Ausgang zu tunneln. Damit das Beschreiben beider Displays einheitlich funktioniert haben wir „Wrapper“-Funktionen geschrieben, die nun immer anstatt den MIOS_LCD Funktionen eingesetzt werden (zu finden in wrapper.c). Sie bekommen neben den normalen Parametern für die MIOS Display Funktionen einen zusätzlichen Parameter, das „Display“ übergeben. Ist das Display 0 ausgewählt wird direkt die entsprechende MIOS_LCD Funktion aufgerufen, ist das Display 1 ausgewählt wird die darzustellende Nachricht automatisch über MIDI SysEx versendet. Damit das zweite Core Modul die ankommenden SysEx Nachrichten unterscheiden kann, muss den Nachrichten die für das Display bestimmt sind ein spezieller Header vorausgehen. Dieser besteht nach dem obligatorischen SysEx Start Byte „F0“ aus unserer Hersteller ID „55“, und den Buchstaben „M“ und „B“ für „MIDIbox“ (siehe Abbildung 4.47). Der zweite Core empfängt sie, wertet den Header aus und ruft nun an seinem Display die entsprechende MIOS_LCD Funktion auf (siehe Abbildung 4.48). Bis zu drei Bytes einer SysEx Übertragung werden im inc_buffer_1-3 aufgehoben, und entweder verworfen, wenn der spezielle Header empfangen wurde, oder zusammen mit allen folgenden Bytes (abgeschlossen mit F7) an den Ausgang getunnelt, sollte der Header nicht empfangen worden sein. Empfängt Core 1 diesen Header, ist sicher, dass die folgende Nachricht für das Display bestimmt ist. Die folgenden Bytes sind dann die Bezeichnung der aufgerufenen Displayfunktion (entsprechende defines sind in common\include\tunnel.h zu finden), das Display, das der Wrapperfunktion übergeben wurde und anschließend die Funktionsparameter, die zur MIOS Funktion gehören. Wird ein *int* Parameter verwendet, so wird er zuerst manuell in zwei Bytes aufgeteilt. Die Displaynummer wird übergeben, um theoretisch noch weitere Cores und Displays anschließen und ansteuern zu können. Hierfür müsste lediglich in den nachfolgenden Cores der Sourcecode aus dem Core 1 mit der entsprechenden Änderung der Displaynummer ($\#define\ CURENT\_DISPLAYx$) verwendet werden. Bis jetzt ist leider in keinem

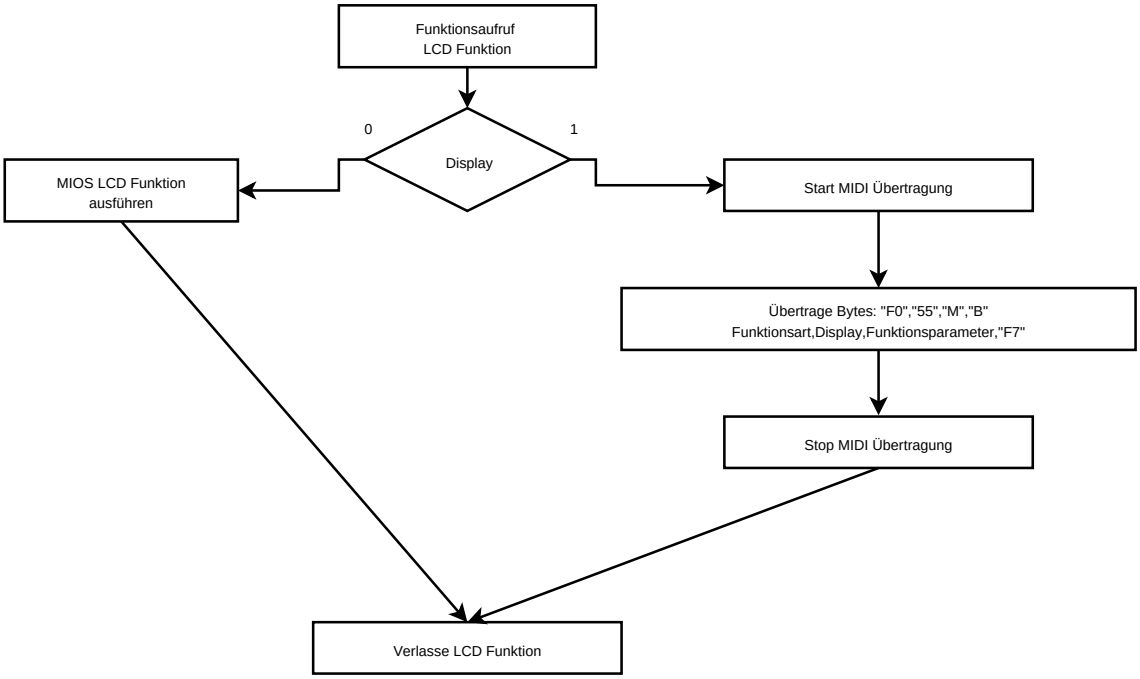

Abbildung 4.47: Display Tunnel in Core 0

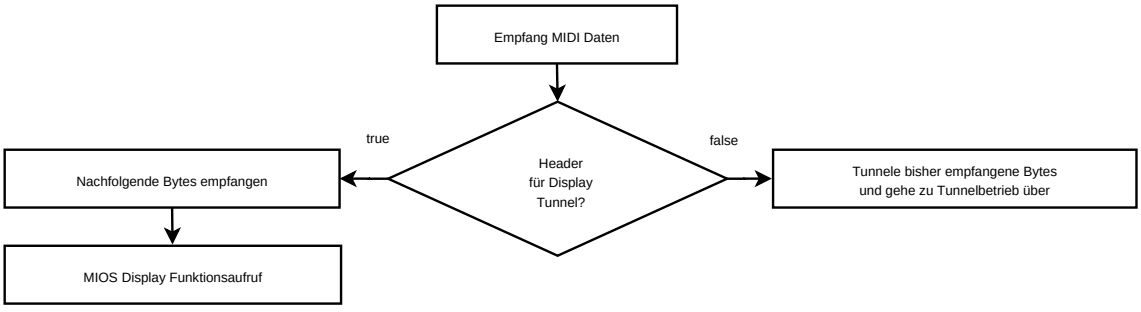

Abbildung 4.48: Display Tunnel in Core 1

anderen Teil des Programms eine Unterstützung weiterer Displays vorgesehen, aber es wären mit zusätzlichem Programmieraufwand zahlreiche Erweiterungen wie z.B. eine erweiterte Darstellung der Menüstruktur (Verzeichnisbaum...) denkbar.

Wir haben nur die MIOS Display Funktionen in den Wrapper aufgenommen, die wir auch tatsächlich verwenden. Falls benötigt, kann diese Liste im Nachhinein noch vervollständigt werden. Bis jetzt sind implementiert:

- MIOS_LCD_PrintChar => printChar
- MIOS_LCD_Clear => clearLCD
- MIOS_LCD_CursorSet => setCursor
- MIOS_LCD_PrintCString => printString
- MIOS_LCD_PrintBCD1 => printBCD1
- MIOS_LCD_PrintBCD2 => printBCD2
- MIOS_LCD_PrintBCD3 => printBCD3
- MIOS_LCD_PrintBCD4 => printBCD4
- MIOS_LCD_PrintBCD5 => printBCD5

Um zum Beispiel das Display am Core 1 zu löschen, müsste der C-Befehl lauten: $clearLCD(1)$. Die versendeten Bytes sind hierbei F0(SysEx Start) 55(Hersteller ID) 4D(M) 42(B) 00(FUNC_CLEAR_LCD) 01(Display 1) F7(SysEx End). Noch einmal ohne Kommentare: F0 55 4D 42 00 01 F7.

4.9 Display-Anzeige

In diesem Abschnitt wird beschrieben, wie die Ausgabefunktionen für die beiden Displays realisiert sind (zu finden in display.c). Da die beiden Displays stets eine funktionelle Einheit bilden, werden sie als ein großes Display behandelt.

Herzstück der Display Anzeige ist die MIOS Funktion $DISPLAY\_Tick$, die immer dann aus dem MIOS Mainloop aufgerufen wird, wenn gerade keine „temporary Message“ angezeigt wird, und somit zur Darstellung von Echtzeit Nachrichten geeignet ist. Wie in Abbildung 4.49 zu sehen, werden in der $DISPLAY\_Tick$ Funktion die vier „least significant bits“ der globalen Variable *update* ausgewertet. Diese Variable wird in allen Funktionen manipuliert die eine Display Ausgabe erzeugen wollen (z.B. $DIN\_NotifyToggle$) um dann in der $DISPLAY\_Tick$ die entsprechende Anzeige auszulösen. Wenn das erste Bit gesetzt ist, also *update*&1 wahr, steht das für eine

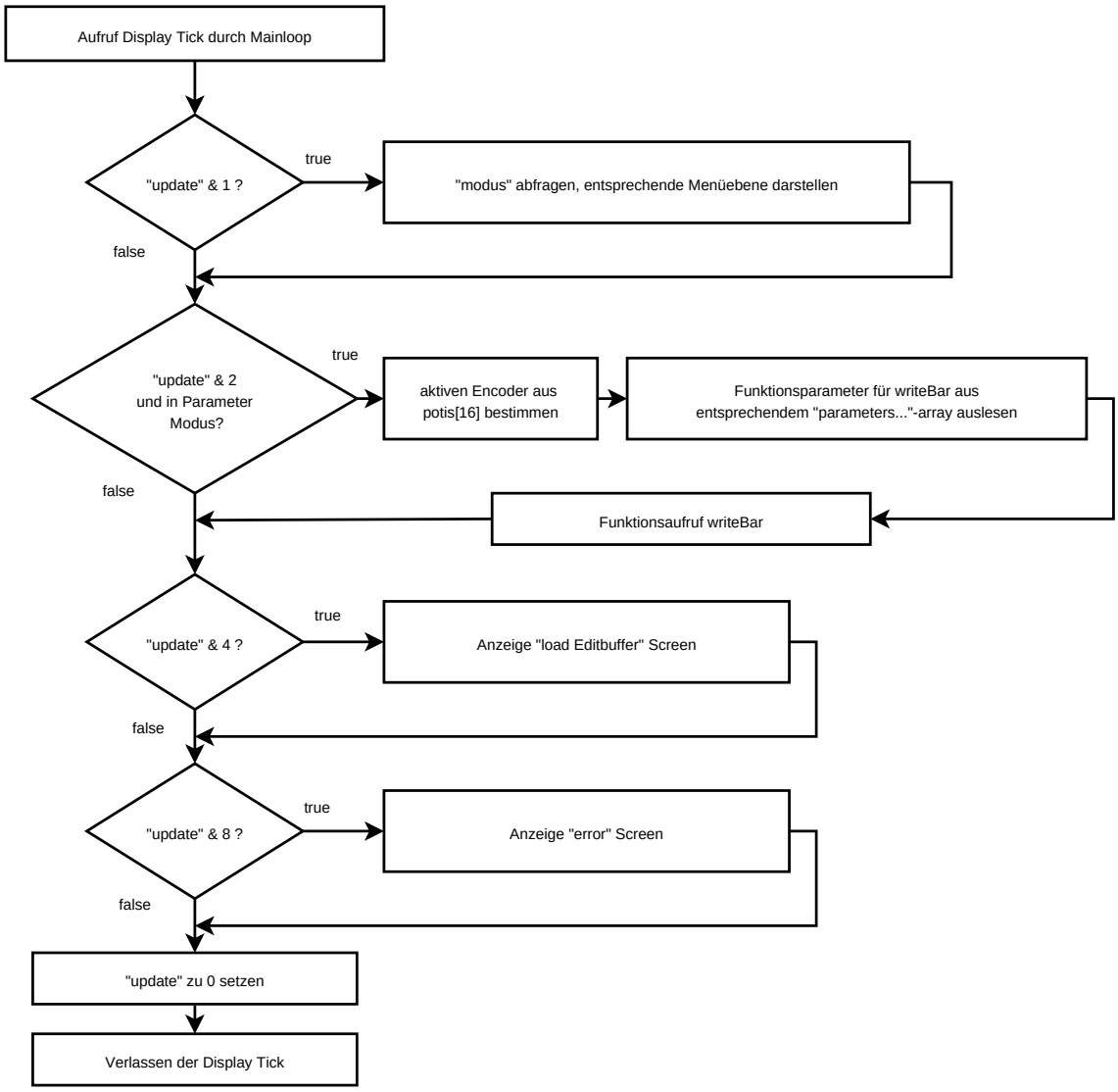

Abbildung 4.49: Diagramm der MIOS DISPLAY_Tick Funktion

Aktualisierung des kompletten Bildschirms, was bei einem Wechsel der Menüebene benötigt wird. Es wird die globale Variable *modus* abgefragt, in der die aktuel-

le Menüebene gespeichert ist, und die entsprechende Funktion zur Menü Darstellung aufgerufen. Diese sind *displayDevice*, *displayModul* und *displayParameter*, und sind sehr ähnlich aufgebaut. In jeder Funktion werden einige Überschriften mit statischen Zeilen geschrieben, und dann noch aus den Banksticks die Namen der jeweiligen Devices, Module und Parameter.

Ist das zweite Bit gesetzt, also *update*&2 wahr, und befindet man sich im Parameter Modus, so werden die Werte der gerade veränderten Parameter aktualisiert. Zuerst wird aus dem Feld *potis*[16] (wird in den MIOS Eingabefunktionen gesetzt) bestimmt, welcher Encoder gedreht oder gedrückt wurde, da aus Performancegründen auch nur die Zeile neben diesem Encoder upgedated wird. Von dem Parameter dieser Zeile werden die zur Anzeige relevanten Eigenschaften aus den Arrays *parametersleft* oder *parametersright* ausgelesen, je nachdem auf welchem Display (links/rechts) sich der Parameter befindet. Diese Eigenschaften (7/14Bit, Text-Values, Skalierung, Vorzeichen) werden der Funktion *writeBar* übergeben, die dann eine entsprechende Darstellung des aktuellen Wertes auf dem Display erzeugt. Die Funktion wird aus Gründen der Übersichtlichkeit gesondert am Ende dieses Kapitels besprochen.

Ist das dritte Bit der *update* Variablen gesetzt, also *update*&4 wahr, so wird auf den Bildschirmen „load EB“ angezeigt. Dies wird während des Ladevorgangs des Edit Buffers benötigt.

Ist das vierte Bit gesetzt, also *update*&8 wahr, so wird ein „error“ Bildschirm angezeigt. Dies wird z.B. bei einer fehlerhaften Übertragung des Edit Buffer eingesetzt (siehe MIOS *Timer* Funktion in general.c).

Die Funktion *writeBar* hat zur Aufgabe, den Wert eines Parameters korrekt auf dem Display zu visualisieren. Da sie den Wert der User Skala anzeigen muss, aber immer den Wert der MIDI Skala übergeben bekommt, muss sie, basierend auf den Eigenschaften des Parameters, eine Umrechnung durchführen. Diese hängt eng mit der Berechnung des MIDI Wertes in den MIOS Eingabefunktionen DIN_NotifyToggle und ENC_NotifyChange zusammen. Die Funktion *writeBar* bekommt außer dem Display, der Zeile und dem Wert des Parameters noch seinen Bitmodus, seine Skalierung und sein Vorzeichen übergeben. Außerdem noch, ob der Parameter eine Textdarstellung seines Wertes besitzt, und wo diese im dynamischen Speicher zu finden ist. Die Eigenschaften werden ausgewertet, wie in Abbildung 4.50 zu sehen. Zuerst wird der Bitmodus ausgewertet, was Auswirkungen auf die benötigten Stellen der Werteanzeige und der Berechnung des angezeigten Wertes hat. Bei 7 Bit werden drei Stellen (bzw. zwei bei negativen Werten) zur Darstellung benötigt, bei 14 Bit 5 (bzw. 4). Der Offset, der im Falle von negativen Werten von dem MIDI Wert abgezogen wird, beträgt bei 7 Bit 64, bei 14 Bit 8192. Anschließend wird für den Wert ein Balken am Display Rand gezeichnet. Die verschiedenen Stufen des Balkens

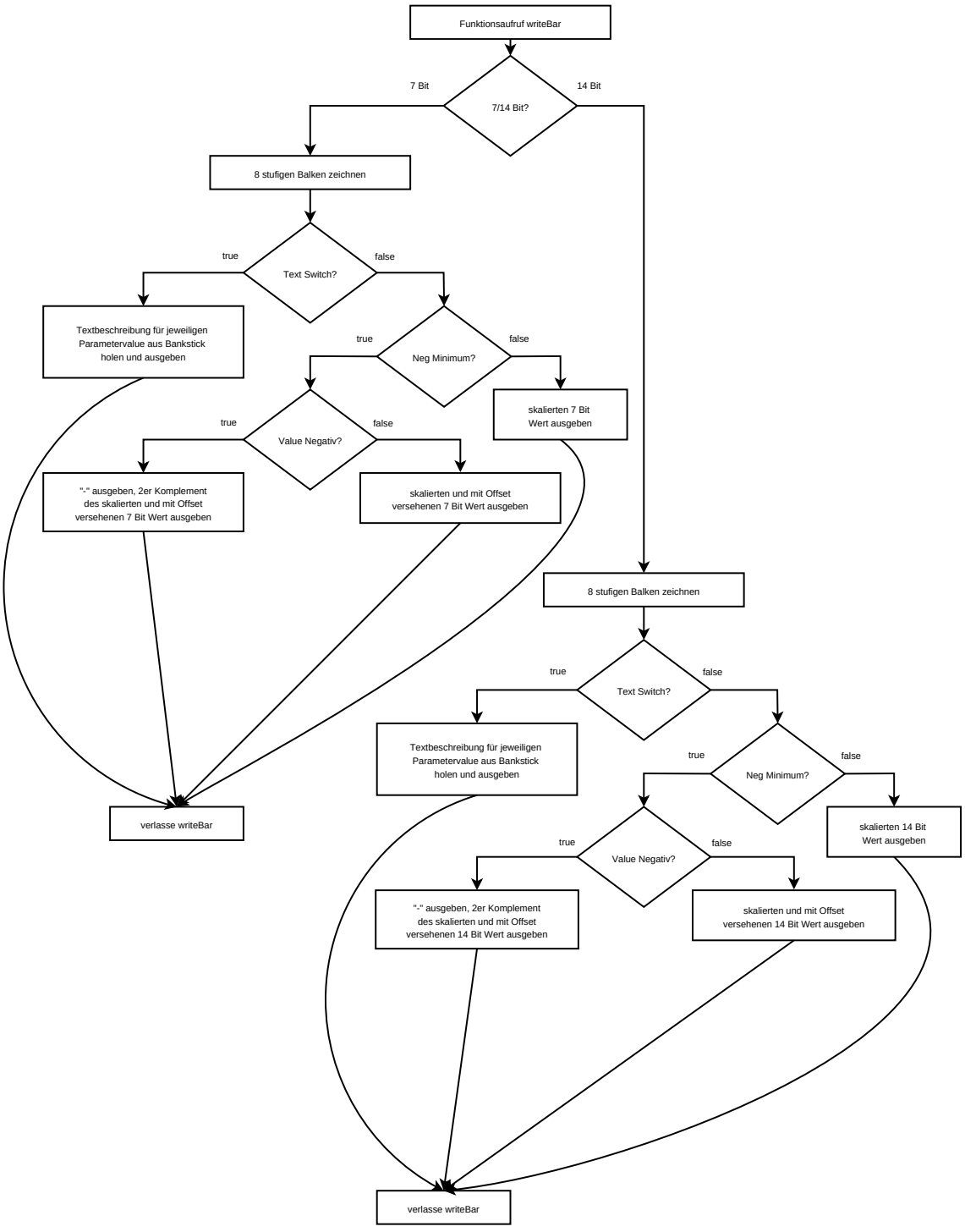

Abbildung 4.50: Diagramm der writeBar Funktion

finden sich in der MIOS Standardfont an Position 0 bis 7. Der 7 Bit Wert wird durch 16, der 14 Bit Wert durch 2048 geteilt, um das benötigte Zeichen in der Font zu ermitteln. Daraufhin wird ausgewertet, ob der Parameter eine textliche Beschreibung für seine Werte besitzt, was im Programm mit der Bezeichnung *sswitch* gekennzeichnet ist. Dies macht zum Beispiel bei der Wellenform eines Oszillators Sinn, da es hier intuitiver ist, die Values mit „Sine“, „Saw“ etc. anstatt mit Ziffern darzustellen. Ist dies der Fall, wird an die entsprechende Stelle im dynamischen Speicher gesprungen, die die *writeBar* Funktion durch die Variable *sswitch_addr* mitgeteilt bekam, und der Text dort ausgelesen. Die Textbeschreibungen sind jeweils statisch auf sechs Zeichen beschränkt, weshalb pro Wertesprung um 6 Stellen im Speicher weiter gesprungen wird. Liegt keine textliche Beschreibung der Values vor, so wird anschließend geprüft, ob der Parameter ein negatives Minimum besitzt. Falls nein, wird der skalierte 7 Bit Wert ausgegeben. Falls ja, wird vom MIDI Wert der Offset (siehe oben) abgezogen, was die angezeigte „Null“ in die Mitte der MIDI Skala rutschen lässt. Von diesem Wert aus kann man sich nun um die halbe MIDI Skala nach unten bewegen, was dann mit negativem Vorzeichen und als 2er Komplement des MIDI Wertes angezeigt wird, oder um die halbe MIDI Skala nach oben, was ohne weitere Umrechnung (Offset von vorhin ausgenommen) angezeigt werden kann. Um die Skalierung zu berücksichtigen, wird jeder angezeigte Wert entweder durch den Scale Wert geteilt, oder falls dieser „Null“ ist, durch die *stepwidth*. Diese wird in der ENC_NotifyChange berechnet, indem der gesamte verfügbare MIDI Wertebereich (0x7F oder 0x3FFF) durch die Anzahl der User Werte geteilt wird, und gibt dann an, nach wie vielen abgeschrittenen MIDI Werten wieder ein User Wert in der Anzeige weiter geschaltet werden muss. Zur Skalierung siehe auch Abschnitt 4.4.

5 Zusammenfassung

Die Aufgabenstellung dieses Medienprojekts war es, einen MIDI Controller zur intuitiven Programmierung von schlecht bedienbaren MIDI Synthesizern zu entwickeln.

Zu diesem Zweck haben wir, gemeinsam mit dem Medienprojekt „Entwurf einer plattformunabhängigen Konfigurationssoftware für die Bedienoberfläche eines Midicontrollers“ [3] ein Konzept für die Bedienung und Menüstruktur des Gerätes entworfen und anschließend den Hardwareteil und den Code für den Microcontroller umgesetzt. Wir haben die Anforderungen an die Hard- und Software analysiert, und für diese Anforderungen entsprechende Lösungen gefunden. Für die Hardware bauten diese zum größten Teil auf dem Open Source Projekt ucapps auf, aber auch auf eigenen Ideen, dort wo wir an die Grenzen des ucapps Projektes stießen. Für den Microcontroller Code bedienten wir uns der zur Verfügung stehenden Werkzeuge, wie z.B. des Betriebssystems MIOS mit seinen Funktionen, und haben mit diesen die Menüstruktur und sämtliche Funktionen des Controllers umgesetzt. Aufgrund der Natur der Aufgabenstellung drangen wir tief in das MIDI Protokoll ein und haben uns intensiv mit Synthesizern und deren Klangprogrammierung auseinandergesetzt. Wir haben eine Bandbreite von Synthesizern analysiert, um möglichst viele Fälle der Programmierung und Verwaltung der Datenstrukturen in unser System integrieren zu können. Abschließend haben wir unser komplettes Projekt, unsere Konzepte und Ausarbeitungen auf der ucapps Internetseite in einem Wiki veröffentlicht, um der Community, die dieses Projekt erst möglich gemacht hat, auch wieder etwas zurückzugeben.

Literatur

[1] www.ucapps.de, aufgerufen bis 31.09.2006

[2] www.zem-collage.de/midi, aufgerufen bis 31.09.2006

[3] Dümke, V.; Korn, J: „Entwurf einer plattformunabhängigen Konfigurationssoftware für die Bedienoberfläche eines Midicontrollers“, Medienprojekt

Erklärung:

Die vorliegende Arbeit ist eine Kollektivarbeit.
Von Herrn Holger Prang wurden folgende Abschnitte bearbeitet 3.3.2, 3.3.4, 3.4.1, 4.2, 4.4, 4.6, 4.7
Von Herrn Marcel Richter wurden folgende Abschnitte bearbeitet 4, 4.2, 4.3, 4.5
Von Herrn Christian Stöcklmeier wurden folgende Abschnitte bearbeitet 1, 2, 2.1, 2.2, 2.3, 3, 3.1, 3.2, 3.3.1, 3.3.3, 3.4.2, 4.1, 4.8, 4.9
Die enthaltenen Bilder 1.1, 2.4, 3.10, 3.12, 3.13, 3.15, 3.16, 3.17, 3.19, 3.21, 3.23, 3.25, 3.26, 3.30 wurden in Kolektivarbeit erstellt.
Von Herrn Holger Prang wurden folgende Abbildungen erstellt 2.6, 2.7, 3.24, 3.27, 3.28, 3.29, 4.41, 4.42, 4.43, 4.46
Von Herrn Marcel Richter wurden folgende Abbildungen erstellt 4.31, 4.32, 4.33, 4.35, 4.36, 4.37, 4.38, 4.39, 4.40
Von Herrn Christian Stöcklmeier wurden folgende Abbildungen erstellt 2.2, 2.3, 2.5, 2.8, 2.9, 3.11, 3.14, 3.18, 3.20, 3.22, 4.34, 4.47, 4.48, 4.49, 4.50
Wir erklären, dass wir diese Arbeit selbstständig durchgeführt und abgefasst haben. Quellen, Literatur und Hilfsmittel, die von uns benutzt wurden, sind als solche gekennzeichnet.

Unterschriften

Ilmenau, den 2. Oktober 2006