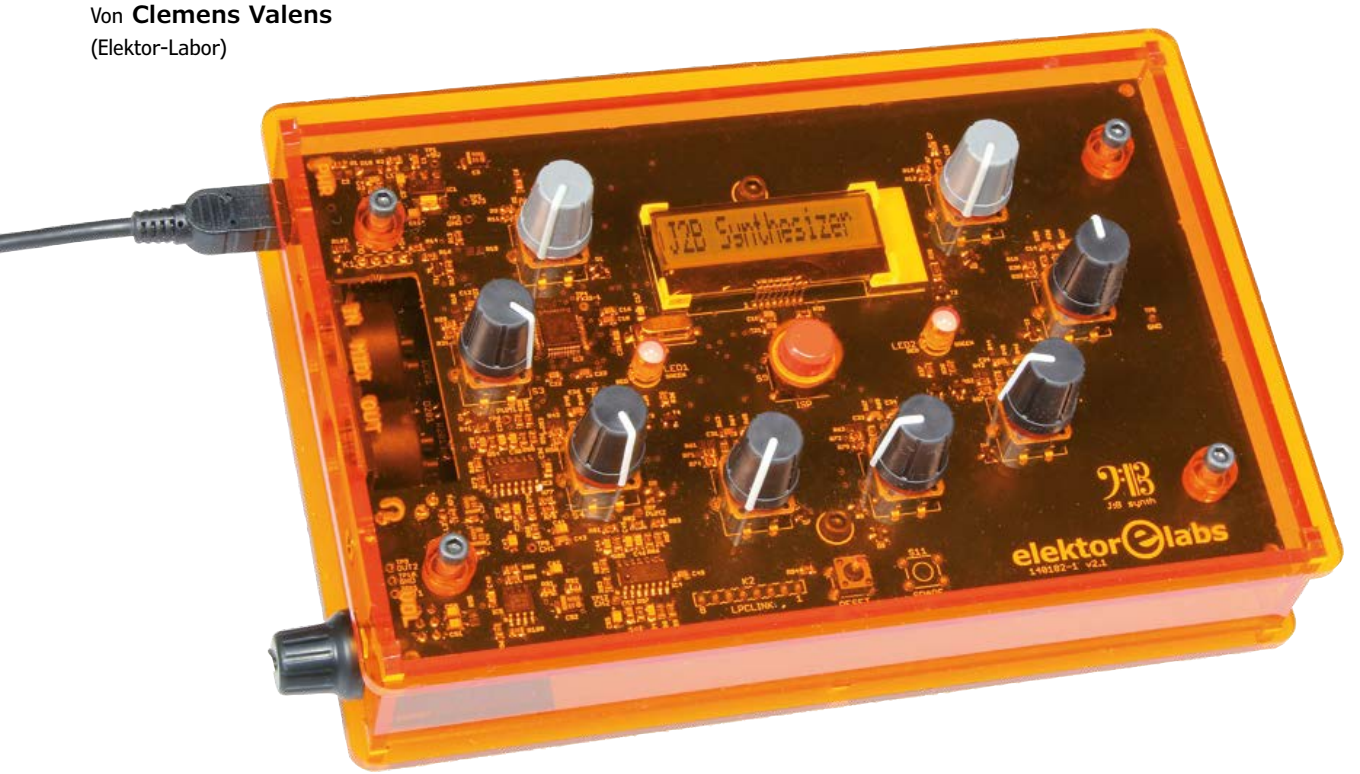
J²B-Synthesizer

Eine offene digitale Musikplattform

Von **Clemens Valens**
(Elektor-Labor)



Der Musik-Synthesizer Atmegatron von Soulsby Synthesizer [1] ist mit einem ATmega328 8-bit-AVR-Controller von Atmel aufgebaut, dem gleichen Mikrocontroller, der auch den Arduino Uno steuert. Der Atmegatron weist einige Aspekte auf, die mich sehr interessierten und schließlich zu diesem Projekt bewogen.

Bevor Sie in das Projekt eintauchen, lassen Sie mich eine Warnung an alle aussprechen, die hautsächlich an der Klangsynthese eines Synthesizers interessiert sind. Nachfolgend wird beschrieben, wie ich den Atmegatron in meine eigene Hardware-Plattform portiert habe. Die Sound-Engine selbst wird nur kurz und bündig beschrieben. Wenn dies Ihr einziges Interesse ist, werfen Sie

Eigenschaften

- Monophoner 9-bit-Synthesizer
- 32 feste und benutzerdefinierte Wellenformen
- 15 Filtertypen
- 2 Hüllkurven-Generatoren
- LFO mit 16 Wellenformen
- 15-Muster-Arpeggiator
- 16 Patch-Memories
- 6 Live-Steuerungen
- MIDI
- Patch sichern/laden über MIDI
- Mikrocontroller NXP LPC1347 32-bit-ARM-Cortex-M3
- 2 Ausgangskanäle
- Open-Source & Open-Hardware-Design

einen Blick auf Bild 2; danach sollten Sie den Source-Code in [2] studieren.

Atmegatron

Der Mikrocontroller im Atmegatron (**Bild 1**) kann im Prinzip alles, von der Erzeugung von Klängen bis zur Interaktion mit dem Anwender. Er kann sich sogar mit anderen Geräten über seinen MIDI-Anschluss unterhalten. All diese Funktionen passen in den 32 KB großen Programmspeicher, was, wie ich finde, schon eine Leistung ist, zumal die Software als Arduino-Sketch in C/C++ geschrieben wurde. Die Software wird unter einer Open-Source-Lizenz veröffentlicht und kann kostenlos heruntergeladen werden. Leider liegt die Hardware des Atmegatrons nicht offen (obwohl man sie ganz einfach aus dem Studium der Software rekonstruieren kann).

Ein weiteres interessantes Feature ist, dass der Atmegatron für Live-Auftritte konzipiert wurde. Es besitzt sechs Potentiometer zur Steuerung, um zehn Klangparameter im laufenden Betrieb ändern zu können. Zusammen mit einem Funktionswähler und einem Funktionswert-Encoder besitzt der Synthesizer acht Steuerungen. Ein Taster wählt zudem zwischen zwei Betriebsarten. Grün/rote LEDs zeigen den Modus und die Werte an, ein alphanumerisches Display ist nicht vorhanden.

Für alle, die Sound „mit Biss“ süßem und sanftem Gedudel vorziehen, ist es interessant zu wissen, dass Atmegatron verschiedene Algorithmen bietet, um hässliche und entstellte Sounds zu produzieren. Atmegatron sollte daher nicht mit den Synthies der ganz großen Hersteller verglichen werden, Atmegatron hat seinen eigenen, ganz speziellen Sound.

Der Synthesizer verwendet das Pulsweitenmodulationsmodul (PWM) des Controllers, um Töne zu erzeugen. Ein externer Digital/Analog-Wandler (DAC) kann daher auf die Funktion eines Anti-Aliasing-Filters beschränkt sein.

Eine Open-Source-Sound-Engine, die von sechs Potentiometern und zwei Drehgebern gesteuert wird und eine Software, die in einen 32-KB-Programmspeicher passt: das hat mich neugierig gemacht, ob ich so etwas auf das J²B-Controllerboard, das ich vor ein paar Jahren in Elektor vorgestellt hatte, portieren könne [3]. J²B wurde um den 32-bit-ARM-Controller Cortex-M3 LPC1343 von NXP aufgebaut. Wie der ATmega328 besitzt er 32 KB Programmspeicher und alle andere Peripherie, die vom Synthesizer verwendet wird, einschließlich der ausgezeichneten PWM-Funktionen. Außerdem unterstützt das J²B-Board bis zu neun Drehencoder (oder acht und einen Druckknopf). Was allerdings fehlt, ist ein EEPROM, um die Sound-Presets zu speichern, aber es gibt Möglichkeiten, diese Klippe zu umschiffen.

Der Atmegatron verwendet zweifarbige LEDs, um Werte, Positionen und Modi ohne alphanumerisches Display anzuzeigen. Das J²B-Board verfügt aber über ein LCD, man kann sogar unter drei Größen wählen. So habe ich beschlossen, die LEDs durch ein LCD zu ersetzen.

Die Portierung lief auf drei Hauptaufgaben hinaus:

1. Portierung des Arduino-Sketches für den ATmega328 zu einem Eclipse/LPCXpresso-Projekt mit dem LPC1343;
2. Ersetzen der sechs analogen Potis durch sechs digitale Drehencoder;
3. Ersetzen der zweifarbigen LEDs durch ein LCD.

Bild 1.
Der Atmegatron, der Ursprung dieses Projekts.

Aufgabe 1

Die Portierung eines Codes von einem Mikrocontroller zum anderen kann mehr oder weniger entmutigend sein, je nachdem, wie die ursprüngliche Software geschrieben wurde. Ein gut strukturiertes Projekt ist viel einfacher umzusetzen als Spaghetti-Code. Auch die Abstraktion der Hardware spielt eine Rolle. Code, der Funktionen aufruft, um Register und Peripherie zu ändern, ist viel einfacher zu portieren als wenn wir es mit einer direkten Interaktion mit der Hardware zu tun hätten. Darüber hinaus ist eine Datei, die all diese Funktionen gruppiert, einfacher zu handhaben als Code, der diese Funktionen über die gesamte Software verteilt. Zum Glück ist der Atmegatron-Code gut strukturiert, gut dokumentiert und sehr „portierfreundlich“, so dass ich einen großen Teil der Arbeit erledigen konnte, während ich einen Bruce-Willis-Film im Fernsehen verfolgte. Die meiste Arbeit ergab sich bei der Erstellung der Header-Dateien für das Eclipse/LPCXpresso-C-Projekt, Arduino-Sketches kommen nämlich ohne Header-Dateien aus. Typische Fallstricke in dieser Phase sind die inkonsequente Zuordnung der Datentypen in der Software und inkompatible Datentypen zwischen den Compilern (AVR GCC für Arduino und ARM GCC für Eclipse/LPCXpresso). Beide Probleme führen zu Fehlern wie Datenüberlauf (Variablen passen nicht mehr in den für sie reservierten Speicherbereich) und unbeabsichtigte Vorzeichenwechsel. Solche Probleme kann man durch die konsequent richtige Definition der Datentypen (mit deutlicher Angabe der Größe in bits und ob sie „signed“ oder „unsigned“ sind) vermeiden. Verwenden Sie beispielsweise `uint8_t` für einen 8-bit-Ganzzahl-Wert statt `unsigned char`. Verwenden Sie `int16_t` oder `int32_t` statt `int`, da die Größe des `int` meist plattformabhängig ist. Noch besser ist es, wenn Sie Datentypen verwenden, die auch zeigen, um welche Art von Daten es sich handelt. Erstellen Sie beispielsweise einen Datentyp `sample_t` und verwenden Sie ihn nur, um Sample-Werte zu speichern, und dies durchgängig.

Viele schwierige Situationen bei der Portierung entstehen durch die Unterschiede der Hardware. Timer sind relativ einfache Peripherieblöcke, die bei den meisten CPUs mehr oder weniger gleich verwendet werden. Sie müssen also nur wissen, wie der Timer funktionieren soll. Aber was, wenn die Zielplattform nicht über die Funktion verfügt, die die Quellplattform einsetzt? Oder wenn die Peripherie sich ein wenig unterschiedlich verhält? Dies war beispielsweise beim PWM-Modul des LPC1343 der Fall, das seine Arbeit nicht ganz genau so erledigte wie das PWM-Modul des ATmega328, so dass ein verzerrter Klang die Folge war. Lassen Sie es mich erklären!

Beim AVR vergleicht das PWM-Modul ständig einen Zähler mit einem Schwellwert, setzt seinen Ausgang entsprechend und bestimmt so das Tastverhältnis des PWM-Signals. Wenn der Zählerwert gleich oder höher als der Schwellwert ist, ist der PWM-Ausgang Low (oder High, je nach Konfiguration), wenn der Zählerwert dagegen unter dem Schwellwert liegt, ist der PWM-Ausgang High. Wenn man im zweiten Fall aber den Schwellwert unter den Zähler bewegt, wird der PWM-Ausgang unmittelbar Low. Ein C-artiger Pseudo-Code sähe so aus:

```
counter = counter + 1
if (counter == max) then counter = 0
if (counter >= threshold) then PWM = 0
else PWM = 1
```

Beim LPC1343 der Mechanismus ist nahezu identisch. Nahezu. Anstelle der ständigen Kontrolle des Zählerstands ändert dieser Controller den PWM-Ausgang nur dann, wenn Zähler und Schwellwert exakt den gleichen Wert aufweisen (ein „match“). Eine Änderung der Schwelle hat daher keine unmittelbare Wirkung, der Ausgang wechselt nur, wenn der Zähler den neuen Schwellwert erreicht. Auch dies in Pseudocode:

```
counter = counter + 1
if (counter == max) then
  {
    counter = 0
    PWM = 1
  }
if (counter == threshold) then PWM = 0
```

Um die Auswirkungen zu verstehen, müssen wir einen genaueren Blick auf die Sound-Engine des Atmegatron werfen.

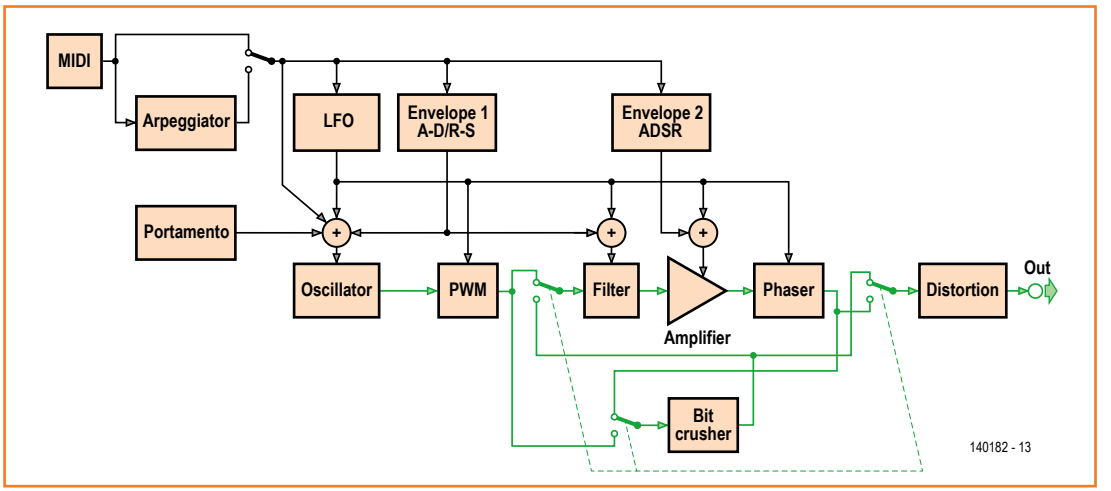


Bild 2.
Die Sound-Engine des Synthesizers, eingefangen in einem Blockdiagramm. Die Blöcke PWM, Bit-Crusher und Distortion sind für die ursprünglichen, rauen Töne verantwortlich.

Über die Sound-Engine

Die Sound-Engine (**Bild 2**) ist für die Ausgabe von Sounds verantwortlich. In unserem Fall berechnet sie den Ausgangssound in Blöcken á 32 Samples. Ein solcher Block entspricht immer einer Periode der Kurvenform am Ausgang (was bedeutet, dass die Abtastrate nicht konstant ist). Die Berechnungen beginnen mit einer Wavetable, die 32 Samples einer Periode einer Kurvenform enthält (es sind 32 Wellenformen vorgefertigt, aber Sie können auch Ihre eigenen definieren). Diese Samples werden gefiltert, auf verschiedene Weise berarbeitet und dann in einem Ausgangspuffer gespeichert. Dieser Puffer wird so schnell, wie es die Hauptschleife des Arduino-Programms erlaubt, aktualisiert, was bedeutet, dass dies eigentlich wie ein Task im Hintergrund läuft, ohne eine wirkliche Priorität. Die Aktualisierung wird halt durchgeführt, wenn das Programm an dieser Stelle ankommt. Sich dynamisch ändernde Parameter wie Lautstärke-Hüllkurven und Modulationssignale werden von einem Millisekunden-Timer synchronisiert, so dass sie von der Ausführungsgeschwindigkeit der langsamen Hauptschleife unabhängig sind.

Die Tonhöhe wird von einem Timer bestimmt, dessen Frequenz 32 Mal höher ist als der Ton selbst, um die Größe der Wavetable mit ihren 32 Samples auszugleichen. Wenn der Timer seinen von der Tonhöhe abhängigen Endwert erreicht hat, wird das nächste Sample vom Ausgangspuffer erfasst und die PWM-Schwelle (also das Tastverhältnis) gemäß der neuen Abtastung neu festgesetzt. So, wie der AVR arbeitet, folgt das Tastverhältnis den Änderungen der Samples nahezu ohne Verzögerung.

Beim LPC1343 wird das Tastverhältnis nur erneuert, wenn der PWM-Timer den neuen Schwellwert erreicht, was bis zu einer PWM-Periode dauern kann (wenn der neue Schwellwert genau unter den aktuellen Zählerwert gesetzt wird). Diese variable Verzögerung produziert hörbare Störungen und weiches rhythmisches Klicken.

Die Lösung des Problems war es, Interrupts beim PWM-Modul des LPC einzusetzen. Obwohl das PWM-Signal eine Frequenz von 140,625 kHz aufweist, kann der Controller dabei problemlos mit Interrupts umgehen. Der Interrupt-Routine des Pitch-Timers geht durch den Sample-Ausgangspuffer wie zuvor, doch anstatt den PWM-Schwellwert direkt dem neuen Abtastwert anzupassen, wird eine Variable als Zwischenspeicher verwendet. Die PWM-Interrupt-Routine liest diese Zwischengröße und benutzt sie, um das PWM-Tastverhältnis zu aktualisieren. Nun wird der PWM-Update ordentlich auf das PWM-Signal synchronisiert und die Klicks sind eliminiert.

Zu wenig Speicher

Als meine Portierung beinahe erledigt war, stellte sich ein anderes Problem ein: zu wenig Speicher! Das war überraschend, da doch beide Controller die gleiche Menge an Programmspeicherplatz (32 KB) zur Verfügung stellen. Zudem hatte ich erwartet, dass der LPC-Code effizienter ist, weil er als so genannter Thumb-Code kompiliert wird, extra um Programmspeicherplatz zu sparen. Ist der Arduino-Compiler denn wirklich so gut? Oder der AVR so codeeffizient? Wie auch immer, schuld am Übergewicht meiner Portierung war zum Teil die Bibliothek für die Hardware-Abstractions von LPCXpresso für den LPC1343 und zum Teil die in

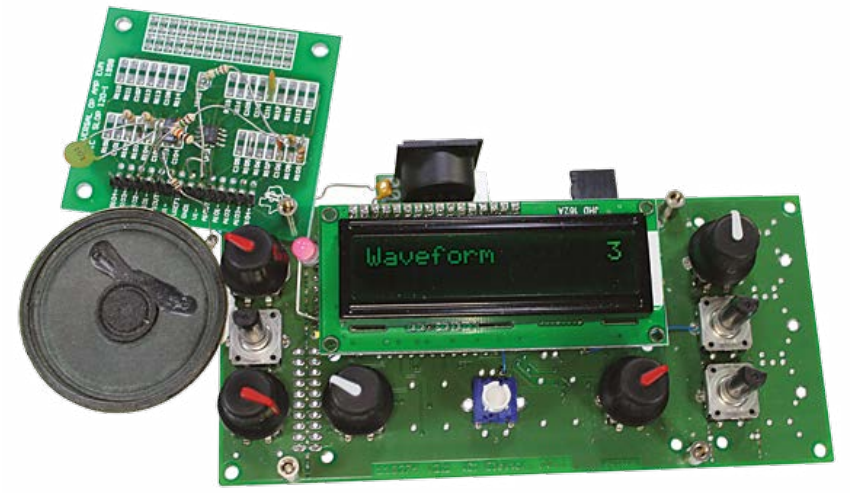


Bild 3.
Der J²B-basierte Prototyp mit seinen acht Drehencodern und dem 2x16-LC-Display.

einem LPCXpresso-Projekt standardmäßig eingebaute newlib-Bibliothek. Abgesehen davon, dass dies die Standard-C-Bibliothek ist, bietet sie auch printf-artige Debug-Unterstützung, auch im Release-Modus. Diese Debug-Unterstützung kann man entfernen, wenn sie nicht nötig ist. Die Option ist schwer zu finden, aber die Mühe lohnt sich: Gehen Sie zu den „Properties“ dieses Projekts und öffnen Sie „C/C++ Build“, gefolgt von „Settings“. Wählen Sie dann in der Liste der „Tool Settings“ das „Managed Linker Script“, um Zugang zur Bibliothek zu erhalten (Sie müssen „MCU Linker“ expandieren). Nach ein bisschen Experimentieren fand ich, dass die Bibliothek „Redlib (none)“ gute Ergebnisse erzielte, genau so gut wie jede andere „(none)“-Bibliothek. Dieser Trick befreit viel Speicherplatz, aber ich musste immer noch die EEPROM-Unterstützung und einen Teil der Benutzerschnittstelle implementieren.

Aufgabe 2

Das Ersetzen von Potentiometern durch Drehencoder mag trivial erscheinen, ist es aber nicht. Ein Poti, das man mit einer schnellen Fingerbewegung von Min nach Max dreht, fühlt sich ganz anders an als ein Drehencoder. Im Gegensatz dazu ermöglichen Drehencoder eine sehr präzise Steuerung der Parameter, aber es sind viele Umdrehungen nötig, um den gesamten Parameterbereich abzudecken. Man muss tricksen, irgendeine Art von Beschleunigungsmechanismus einsetzen, damit man wieder ein Poti „fühlt“, ohne dabei jedoch auf die Präzision des Drehencoders verzichten zu müssen.

Obwohl ich mich eher zu den „Trial-and-error“-Menschen zähle, bin ich an das Problem „wissenschaftlich“ herangegangen. Ich habe den von mir gewählten Drehgeber (24 Takte pro Umdrehung) an mein Oszilloskop angeschlossen und Drehgeschwindigkeiten ausgemessen: Wenn man den Encoder so schnell wie möglich dreht, erzielt man eine Impulsrate von etwa 100 Hz. Mein Ziel war es aber, den gesamten Bereich von 0...255 in *einer* schnellen Umdrehung statt in zehn zu durchlaufen. Da ich nun über praxiserprobte Impulsraten Bescheid wusste, konnte ich einen Algorithmus für die Messung der Taktrate und die Umrechnung in einen Beschleunigungsfaktor entwerfen. Das Resultat war erstaunlich gut!

Aufgabe 3

Die dritte Aufgabe war es, die zweifarbigen LEDs durch ein alphanumerisches LC-Display zu ersetzen. Ich hätte die LEDs erhalten können, aber dies mit einem zusätzlichen Port-Expander am Controller erkauft. Außerdem war es mein Bestreben, so nah wie möglich am J²B-Board zu bleiben. Zudem können auf einem LCD viel mehr Informationen übermittelt werden, was dem Benutzer erspart, dauernd im Handbuch nachschlagen zu müssen. Acht Drehregler am J²B-Board erfordern ein 2x16-LCD, sonst gibt es nicht genug Platz für alle Infos (**Bild 3**). Gute Texte für ein Display zu finden ist schon schwierig, noch komplizierter ist es aber, Werte oder Positionen der sechs Live-Steuerungen sinnvoll und deutlich wiederzugeben. Am einfachsten wäre es, zwei Reihen mit je drei Werten anzuzeigen, dann wäre aber die Position des Wertes (zwischen einer Min- und einer Max-Einstellung) ziemlich undeutlich. Es hat mich allerhand Gehirnschmalz gekostet, aber schließlich habe ich die wohl optimale Lösung in Form kleiner Schieberegler-Symbole gefunden. In einem

Standard-LCD können bis zu acht 5x7-Punkt-Zeichen vom Benutzer definiert werden, und das ist gerade genug, um einen vertikalen Schieberegler mit sieben Positionen zu erstellen. Über zwei Zeilen (zwei Zeichen übereinander) ergibt das 14 und zusammen mit zwei speziellen Zeichen für 0 und Maximum dann 16 Postionen.
Ein besonderer Algorithmus ermittelt, ob der Benutzer eine der sechs Live-Steuerungen oder einen Funktions/Wert-Encoder (zum Beispiel zur Einstellung der Wellenform) nutzt, so dass die Software automatisch zwischen zwei Seiten umschalten und die relevanten Informationen anzeigen kann. Ich habe die Anzeige des roten und grünen Modus des Atmegatron einfach durch eine zweifarbige LED realisiert, obwohl ich noch lieber die Hintergrundbeleuchtung des LCD für diesen Zweck eingesetzt hätte.

Auf der Hardwareseite

Nun hatte ich einen arbeitenden, wenn auch nicht voll funktionsfähigen Prototyp auf mein J²B-Board (Bild 3) portiert und es war an der Zeit, sich um das Hardware-Design des Synthesizers zu kümmern. Ich wollte zwar das J²B-Board als Basis benutzen, aber während der Experimente wurde die unergonomische Positionierung der Drehgeber auf der Platine offenbar. Sie lagen zu dicht beieinander. Da ich ohnehin ein Add-on-Board für das Antialiasing-Filter, die MIDI-Schnittstelle, den Kopfhörerverstärker und ein EEPROM entwerfen musste, entschloss ich mich zu einer großen Hauptplatine mit mehr Platz für die Drehencoder. Dann gäbe es auch genug Platz, um so weit wie möglich Durchsteck-Bauteile aus der Bibliothek *Elektor Labs Preferred Parts* (ELPP) einzusetzen. Und das J²B-Hirn würde sich dann auf einer Tochterplatine befinden.
Der Entwurf der Baugruppe hat nicht lange gedauert. Für das Antialiasing hat mir das freie FilterLab von Microchip ein Tschebyscheff-Filter fünften Ordnung mit einer Grenzfrequenz von 15 kHz (in meinem Alter hört man darüber nichts mehr) berechnet. Das hat etwa 30 Sekunden gedauert. Der „Rest“ dauerte etwas länger, aber nach zehn Tagen lagen die geätzte Platine und alle Bauteile auf meinem Labortisch. Ich konnte mit der Montage des neuen Prototyps beginnen.
Leider dämmerte es mir jetzt (viel zu spät), dass mein Hardware-Ansatz eine einzige Katastrophe war. Es gab einfach zu viele Verbindungen zwischen J²B- und Hauptplatine. Und um es noch schlimmer zu machen, waren die meisten der Verbindungen fast unzugänglich. Nach einigen vergeblichen (dummen) Bemühungen, das Problem zu lösen, habe ich dann aufgegeben.
Ich hatte jetzt eine unbrauchbare mechanische Konstruktion mit einem immer noch vorhandenen leichten Speicherproblem. Das hat mich nachdenklich gemacht. Um die Dinge richtig zu machen, sollte ich das J²B-Board erst einmal vergessen und das Design komplett neu beginnen. Aber, wenn ich schon alles neu machen muss, warum dann an dem LPC1343 kleben? Seit diese Controllerfamilie vor einigen Jahren erschien, sind neue Bauteile wie der LPC1347 aufgetaucht, der nicht nur die doppelte Speichergröße (64 KB) besitzt, sondern auch 4 KB internes EEPROM. Er hatte auch den ausgezeichneten USB-Bootloader, also warum nicht zum neuen 1347 wechseln?

Mk.II

Mein neues Design (V2 oder Mk.II, wie Sie wollen) basiert daher auf dem LPC1347 und ist durch-

Anmerkung zur Klangqualität

Die Atmegatron produziert keine qualitativ hochwertige Musik, sondern ist ein LoFi-8-bit-Synthesizer und klingt auch so. Er ist dank seiner speziellen Verzerrungsalgorithmen in der Lage, aggressive und hässliche Geräusche und alle Arten von „Robotersounds“ zu erzeugen. Es klingt ein bisschen wie ein Casio-Keyboard der 80er Jahre auf Speed. Wenn dies berücksichtigt wird, kann man einige hervorragende Sounds erzeugen und damit jede Menge Spaß haben. Der Arpeggiator ist ein nettes zusätzliches Feature. Ich habe die Klangqualität des J²B-Ports ein „bitchen“ verbessert: Die PWM des LPC1347 ist 16 bit tief, während Atmegatron nur 8 bit erlaubt (der 16-bit-PWM-Modus des AVR ist zu langsam für diese Anwendung). Ich habe die PWM-Tiefe um ein Bit erhöht, so dass es jetzt ein 9-bit-Synthesizer ist. Ich habe auch die PWM-Frequenz mehr als verdoppelt, so dass das Anti-Aliasing-Filter (5. Ordnung anstelle 3. Ordnung beim Atmegatron) bessere Ergebnisse liefert, was zur weiteren Verbesserung der Klangqualität führt.
Es gibt noch viele weitere Möglichkeiten zur Klangverbesserung. Die Sound-Engine-Algorithmen neigen dazu, ihre Ausgaben bei acht Bit abzuschneiden. Bei einem 32-bit-Controller ist dies ein wenig schade. Die Wavetables könnten länger sein. Die Filterung verwendet jetzt eine Gleitkomma-Arithmetik, was zwar ok ist, aber auch aber zeit- und speicheraufwändig. Ein Übergang zu Festkomma-Arithmetik würde kostenlos eine Menge Rechenleistung für andere Sound-Algorithmen frei machen.
Eine weitere interessante Erweiterung wäre, virtuelle Synthesizer-Funktionen (zum Beispiel zur Steuerung eines Software-Synthesizers auf dem PC) über den USB-Port hinzufügen und/oder MIDI über USB.
Der J²B-Synthesizer ist eine preiswerte Plattform, um mit computerbasierter Klangsynthese direkt am Gerät zu experimentieren. Das Stichwort ist Prog’n’Play!

Bild 4.
Schaltung der Hauptplatine des J²B-Synthesizers.

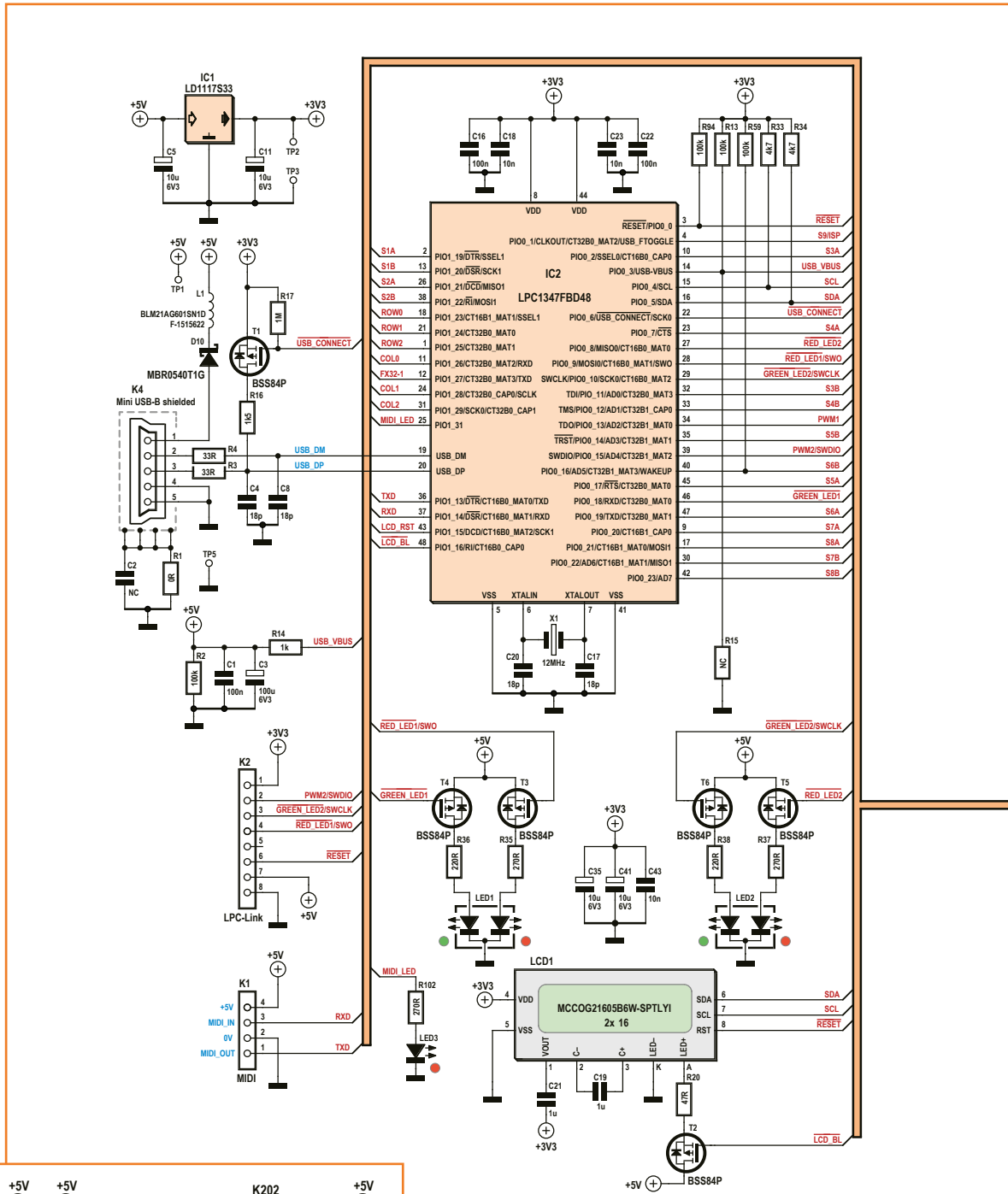
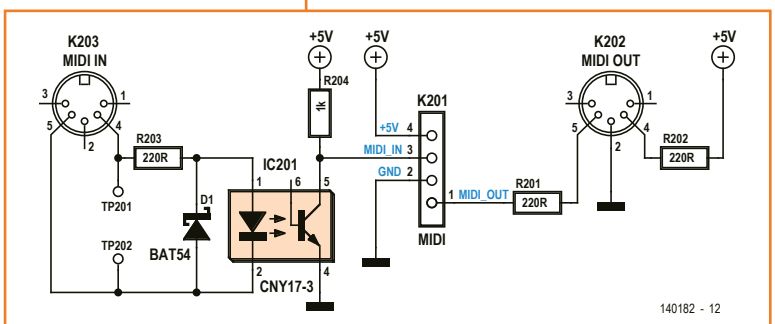


Bild 5.
Schaltung des MIDI-Interfaces. Es ist wegen der Größe der 5-poligen DIN-Buchse auf einer eigenen Platine untergebracht. Damit ist das Design mechanisch gesehen sehr flexibel.



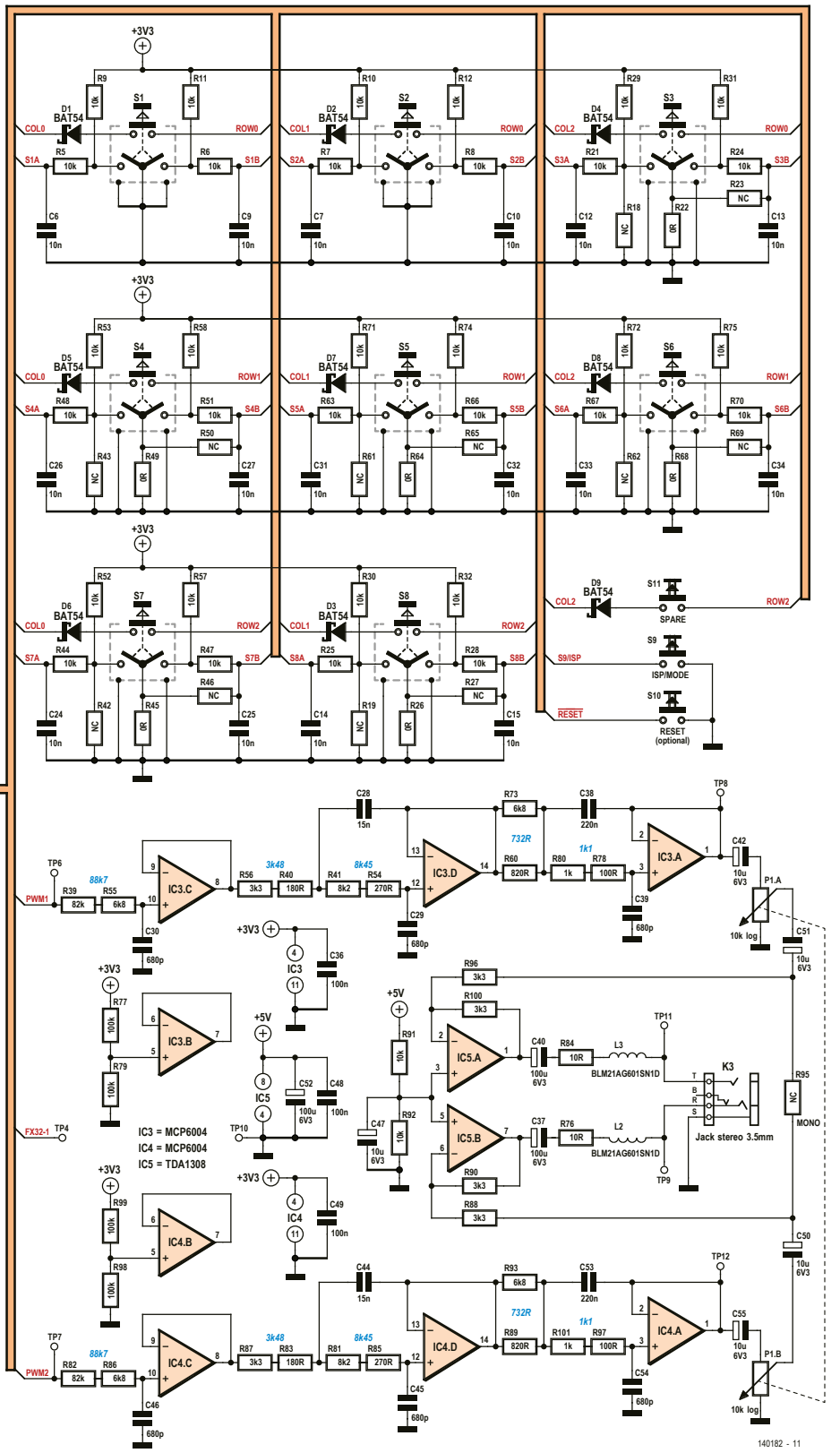

+3V3
S1
S2
S3
S4
S5
S6
S7
S8
S11
SPARE
S9
S9/ISP
ISP/MODE
S10
RESET
RESET (optional)
D1 BAT54
D2 BAT54
D3 BAT54
D4 BAT54
D5 BAT54
D6 BAT54
D7 BAT54
D8 BAT54
D9 BAT54
COL0
COL1
COL2
ROW0
ROW1
ROW2
IC3.A
IC3.B
IC3.C
IC3.D
IC4.A
IC4.B
IC4.C
IC4.D
IC5.A
IC5.B
IC3 = MCP6004
IC4 = MCP6004
IC5 = TDA1308
Jack stereo 3.5mm
MONO
P1.A
P1.B
10k log
PWM1
PWM2
FX32-1
BLM21AG601SN1D
140182 - 11

Bild 6.
Der J²B-Synthesizer in einem vom Laser geschnittenen Acrylgehäuse.

gängig mit SMDs bestückt (mit Ausnahme der Buchsen, Encoder und LEDs). Nicht aus Gründen der Symmetrie habe ich eine zweite zweifarbige LED hinzugefügt. In der Bedienungsanleitung des Atmegatron hatte ich übersehen und erst bei der Portierung gemerkt, dass eine LED eine MIDI-Aktivität anzeigt. Da der LPC1347 wie der LPC1343 über mehrere Kanäle pro PWM-Timer verfügt, habe ich beschlossen, ein zweites Antialiasing-Filter hinzuzufügen, um einen Zweikanal-Betrieb zu ermöglichen. Schließlich verwenden wir einen 32-bit-Cortex-M3, der mit 72 MHz getaktet wird. Da dürfte wohl ausreichend Rechenleistung vorhanden sein...

Die Schaltung

Beim Blick auf die Schaltung des Synthesizers (**Bild 4**) werden Sie keine Überraschungen erleben. Die acht Drehencoder belegen alle verfügbaren I/O-Ports, zwei pro Encoder. Die Widerstände bei den sechs Live-Steuerungen dienen der Flexibilität: Man kann durch die richtigen Widerstandswerte die Drehgeber durch Potentiometer ersetzen, da eine Seite der Encoder mit je einem Eingang des Analog/Digital-Wandlers (ADC) des Controllers verbunden ist. Nehmen Sie Encoder S5: Normalerweise wären R63, R64, R66, R71, R74, C31 und C32 vorhanden, um aber ein Poti

Stückliste

Hauptplatine

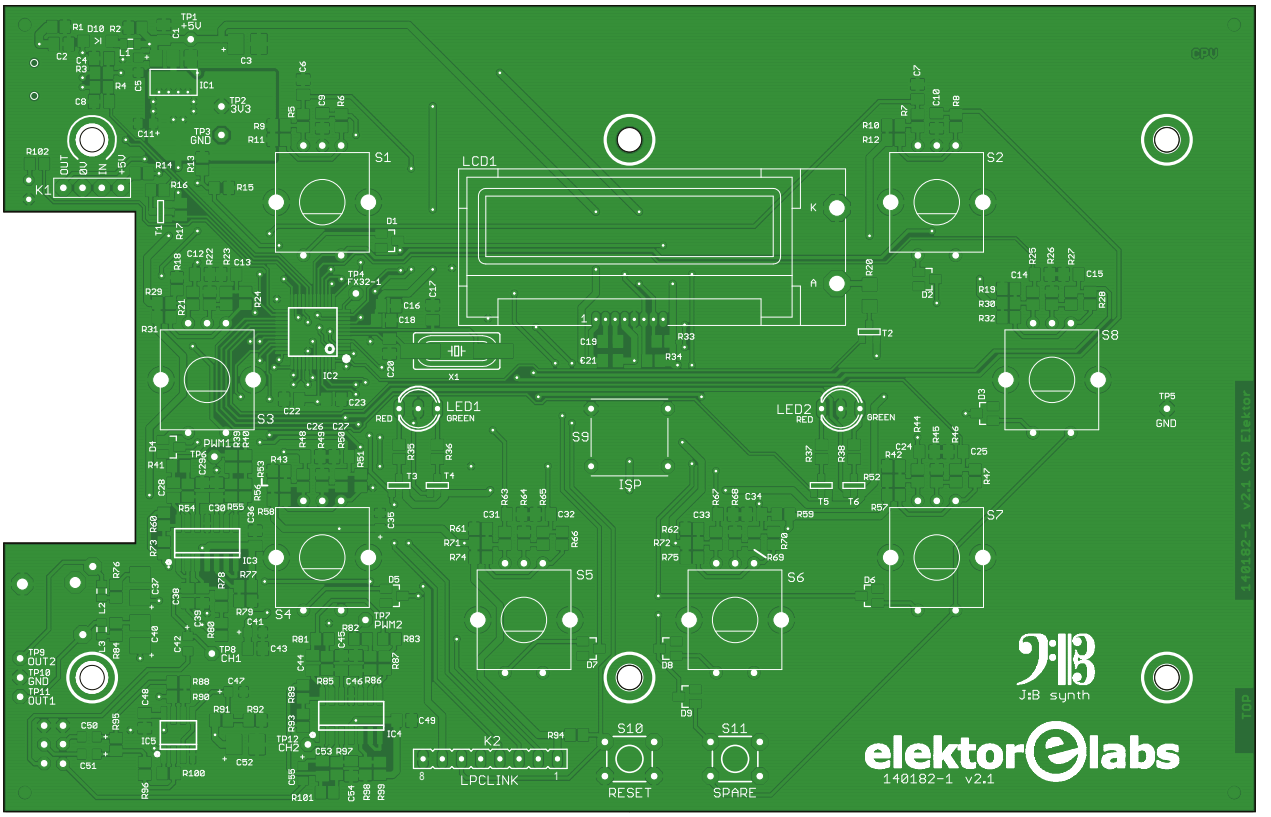
Widerstände:
alle SMD 0805, 5%, 0,1 W
R1,R22,R26,R45,R49,R64,R68 = 0 Ω
R2,R13,R59,R77,R79,R94,R98,R99 = 100 k
R3,R4 = 33 Ω
R5-R12,R21,R24,R25,R28..R32,R44,R47,R48,R51,R52,R53,R57,R58,R63,R66,R67,R70..R75,R91,R92 = 10 k
R14,R80,R101 = 1 k
R16 = 1k5
R17 = 1 M
R20 = 47 Ω, 1206, 0,25 W
R33,R34 = 4k7
R35,R37,R54,R85,R102 = 270 Ω
R36,R38 = 220 Ω
R39,R82 = 82 k
R40,R83 = 180 Ω
R41,R81 = 8k2
R55,R73,R86,R93 = 6k8
R56,R87,R88,R90,R96,R100 = 3k3
R60,R89 = 820 Ω
R78,R97 = 100 Ω
R76,R84 = 10 Ω
P1 = 10 k-Potentiometer, Stereo, logarithmisch
R15,R18,R19,R23,R27,R42,R43,R46,R50,R61,R62,R65,R69,R95 = nicht verwendet

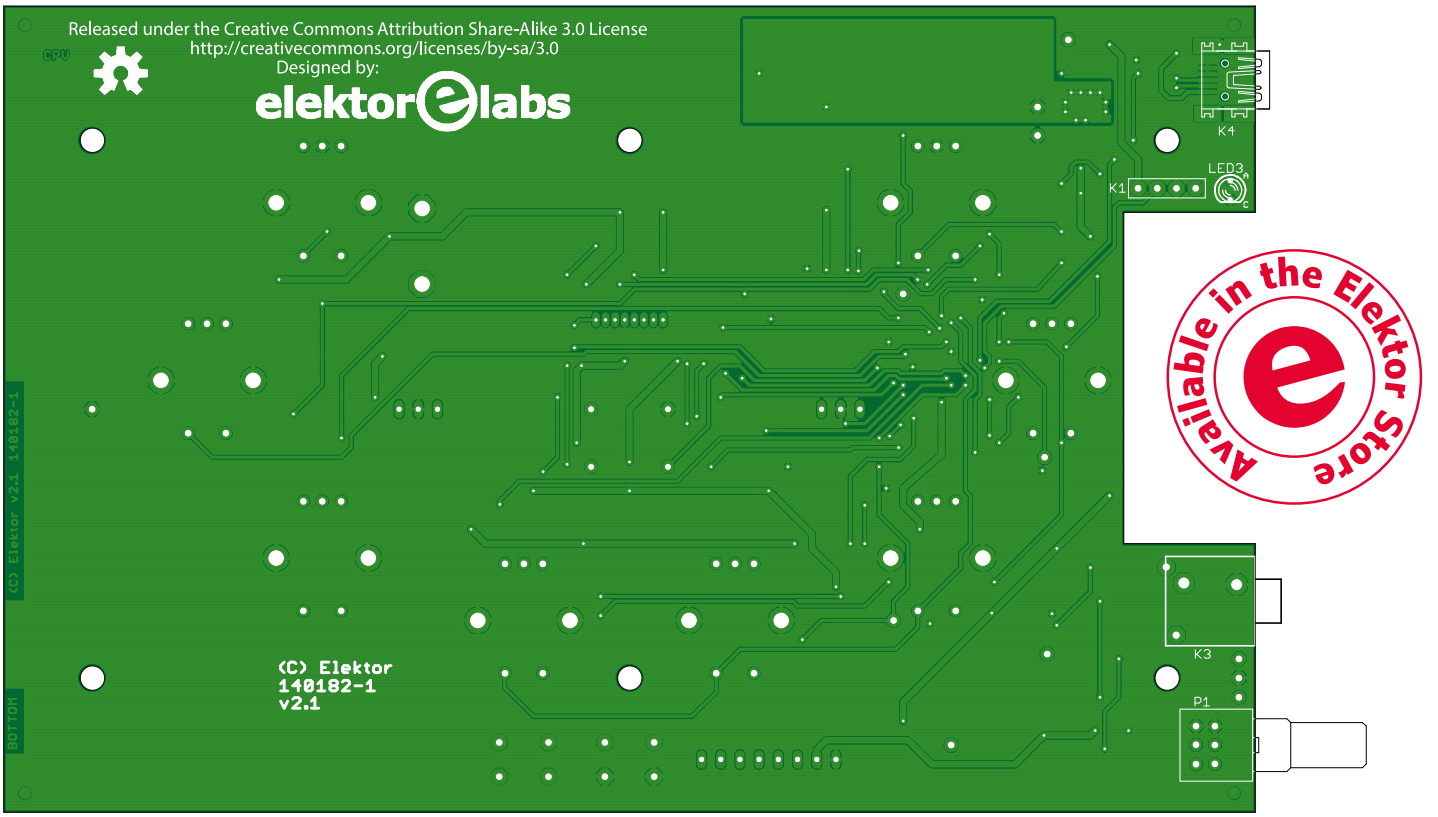
Kondensatoren:
alle SMD 0805
C1,C16,C22,C36,C48,C49 = 100 n
C3,C37,C40,C52 = 100 µ, 6V3 Tantal, Größe B
C4,C8,C17,C20 = 18 p
C5,C11,C35,C41,C42,C47,C50,C51,C55 = 10 µ, 6V3 Tantal
C6,C7,C9,C10,C12-C15,C18,C23-C27,C31-C34,C43 = 10 n
C19,C21 = 1 µ
C28,C44 = 15 n
C29,C30,C39,C45,C46,C54 = 680 p
C38,C53 = 220 n
C2 = nicht montiert

Induktivitäten:
L1,L2,L3 = Ferritperle, 0Ω21, 0,6 A, SMD 0805

Halbleiter:
D1...D9 = BAT54C (SOT-23)
D10 = MBR0540T1G
IC2 = LPC1347FBD48
IC1 = LD1117S33CTR
IC3,IC4 = MCP6004-I/SL
IC5 = TDA1308T/N2
LED1,LED2 = LED, zweifarbig rot/grün, CC, 5 mm
LED3 = LED, rot, 3 mm
T1...T6 = BSS84P (SOT-23)

Außerdem:
K1 = 4-polige Stiftleiste, Raster 0,1”
K2 = 8-polige Stiftleiste, Raster 0,1”
K3 = Klinkenbuchse 3,5 mm, Stereo
K4 = Mini-USB-B-Buchse, abgeschirmt
S1...S8 = Drehencoder
S9 = Drucktaster Multimec 3FTL6
LCD1 = LCD 2x16, I²C, z.B. Midas MCCOG21605B6W-SPTLYI
X1 = Quarz 12 MHz
BOX1 = Hammond 1597DGY oder lasergeschnitten.
Hauptplatine 140182-1 (Elektor-Shop)


LCD1
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
LED1
LED2
ISP
K1
K2
LPCLINK
RESET
SPARE
J:B synth
elektorlabs
140182-1 v2.1


Released under the Creative Commons Attribution Share-Alike 3.0 License
http://creativecommons.org/licenses/by-sa/3.0
Designed by:
elektorlabs
(C) Elektor
140182-1
v2.1
K1
K3
K4
P1
LED3
Available in the Elektor Store

einzusetzen, setzen Sie statt dessen R61, R65 und R74 (0 Ω) und vielleicht C32 ein. Signal S5B (der Schleifer) ist dann mit dem ADC-Eingang AD3 verbunden.

Die Drehencoder haben integrierte Drucktasten, die miteinander in Form einer 3x3-Matrix verbunden sind. Eine Stelle der Matrix ist (zurzeit) ohne Funktion. Ein spezieller Reset-Taster ist ebenso vorhanden wie ein eigener Modus-Taster (für die Modi Rot und Grün). Diese Taste wird auch verwendet, um den Controller in den Firmware-Update-Modus zu versetzen. Die rot/grünen LEDs werden von MOSFETs angesteuert, so dass eine Menge LED-Strom fließen kann, ohne dass der Controller überfordert wäre. Das LCD besitzt eine I²C-Schnittstelle. Es ist großartig, es passt in seiner Größe und Höhe zu den Drehencodern, kostet weniger als ein typisches LCD-Modul und belegt weniger Controlleranschlüsse!

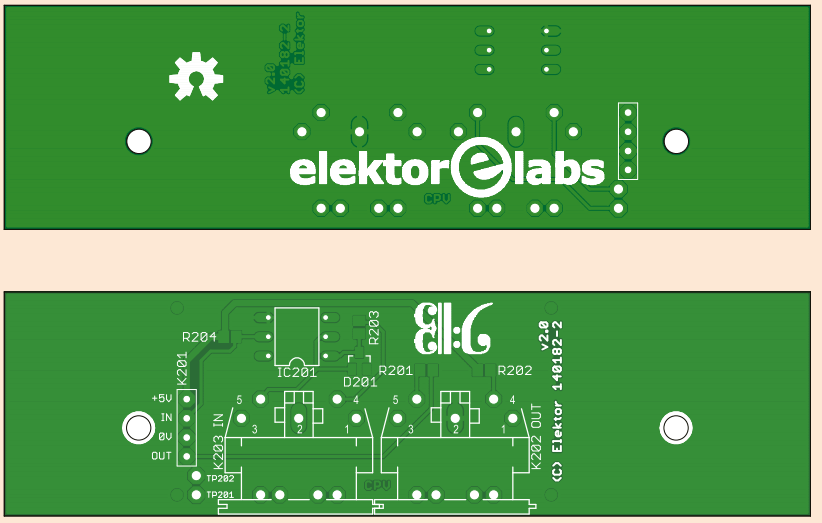
Die Anti-Aliasing-Filter bestehen jeweils aus drei Operationsverstärkern und einigen Widerständen und Kondensatoren. Die Widerstände sind in je zwei Werte geteilt, so dass die E12-Reihe verwendet werden kann. Je ein Operationsverstärker pro Kanal bleibt ungenutzt. Die Filter-Ausgänge, die über die Lautstärke-Potis geführt werden, sind mit zwei Kondensatoren von der Gleichspannung entkoppelt, so dass keine knisternden Geräusche auftreten können. Hi-End-Audio-Freaks mögen die Stirn runzeln, aber dies reicht für eine solche Anwendung aus. Ein Stereo-Kopfhörerverstärker stellt genügend Leistung für viele Anwendungen zur Verfügung.

Die MIDI-Schnittstelle (**Bild 5**) ist auf einer separaten Platine untergebracht, da die DIN-Anschlüsse zu hoch für das von mir gewählte Gehäuse sind. So können sie aber niedriger montiert werden und passen perfekt. Das Hardwaredesign erwies sich nun ebenfalls als perfekt, ich habe für Sie eine Bohrschablone [2] gezeichnet. Ich habe auch einen Versuch mit Laserschneiden gemacht (**Bild 6**). Auch diese Vorlagen stehen Ihnen unter [2] zur Verfügung.

Nochmal: Portierung

Ein neues Hardware-Design mit einem neuen Mikrocontroller erfordert unweigerlich eine zweite Runde der Code-Portierung. Wenn Sie nun denken, der neue Controller kommt ja aus der gleichen Familie wie der alte und die Portierung wäre ein Kinderspiel von einigen Minuten (das war es, was ich erwartet hatte), weit gefehlt!

In Wahrheit war es kompliziert, weil NXP beschlossen hatte, die Bibliothek für den Chip zu ändern, der eine Architektur aufweist, die näher an der Familie LPC11Cxx liegt als an der Familie LPC134x. Kurz gesagt, ist der LPC1347 weder pin- noch 100%-ig codekompatibel mit dem LPC1343. So musste ich in den sauren Apfel



Stückliste

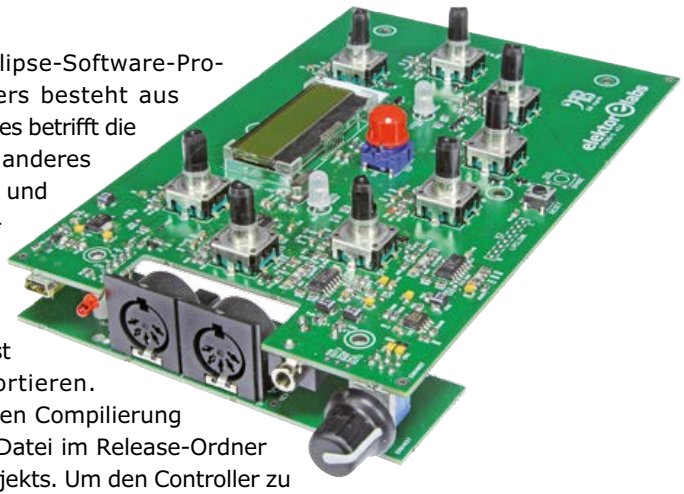
MIDI Board

Widerstände:
alle SMD 0805, 5%, 0,1 W
R201,R202,R203 = 220 Ω
R204 = 1 k

Halbleiter:
D201 = BAT54C (SOT-23)
IC201 = CNY17-3 (DIP-6)

Außerdem:
K201 = 4-polige Buchsenleiste, Raster 0,1“, vertikal
K202,K203 = 5-polige DIN-Fassung für Platinenmontage, 180°
MIDI-Platine 140182-2 (Elektor-Shop)

beißen und den Code durchackern, um die Kompatibilitätsprobleme zu beheben.



Do it yourself

Das LPCXpresso/Eclipse-Software-Projekt des Synthesizers besteht aus drei Teilprojekten. Eines betrifft die Chip-Bibliothek, ein anderes die Board-Bibliothek und ein drittes die Synthesizer-Anwendung selbst. Sie können das Paket als Zip-Datei (ohne zuerst auszupacken) importieren. Nach der erfolgreichen Compilierung finden Sie eine BIN-Datei im Release-Ordner des Synthesizers-Projekts. Um den Controller zu programmieren, brauchen Sie bloß den Synthesizer mit einem freien USB-Anschluss Ihres Windows-PCs zu verbinden, während Sie die Rot/Grün-Taste gedrückt halten. Der Synthesizer muss dabei ausgeschaltet sein, die Energie liefert nur der PC. Wenn alles gut geht, erkennt Windows einen USB-Speicherstick von 64 KB mit einer Datei im Inhalt. Löschen Sie diese Datei und kopieren Sie die BIN-Datei darauf. Drücken Sie die Reset-Taste oder unterbrechen Sie kurzzeitig die Stromversorgung des Synthesizers. Das Display sollte Sie mit der Meldung „J2B Synthesizer“ in großer Schrift begrüßen. Wenn die Anzeige verschwindet, ist der Synthesizer einsatzbereit. Schließen Sie ein MIDI-Keyboard und Kopfhörer an und das Spiel kann beginnen.

Alles ist offen

Dieses Projekt ist zu 100 % Open Source, inklusive der Hardware. Alle Design-Dateien, Hardware, Software und mechanische Unterlagen sind kostenlos erhältlich bei [2]. Ich lade Sie ein, einen Blick darauf zu werfen und zu versuchen, den Synthesizer weiter zu verbessern. Dazu gibt es sicher viele Möglichkeiten. Wenn Sie Ihren eigenen Synthesizer bauen, schicken Sie mir doch bitte ein Foto oder fügen einen Beitrag an die Projekt-Seite [2] an.

User Manual

Eine gute Sache bei der Portierung des Atmegatron (anstelle einer Neuentwicklung) ist, dass wir von der Bedienungsanleitung und dem Librian-Utility des Atmegatron profitieren können. Dies hat mir einen Haufen Schreibarbeit erspart. Laden Sie die Anleitung von [1] herunter und lesen Sie sie sorgfältig, sie ist tatsächlich umfassend. Der J²B-Synthesizer besitzt alle originalen Steuerungen des Atmegatron mit Ausnahme der Klangregelung. Die Drucktasten der sechs Live-Steuerungen, die ein schnelles Zurücksetzen der Werte eines Parameters emöglichen, sind Zugabe: Sie gibt es am Atmegatron nicht.

(140182)

Weblinks

[1] Atmegatron: http://soulsbysynths.com/

[2] Projekt-Downloads: www.elektor-magazine.de/140182

[3] J2B: Vielseitiges HMI-Modul mit ARM Cortex-M3. Elektor September 2011, www.elektor-magazine.de/110274; www.elektor-labs.com/node/3832