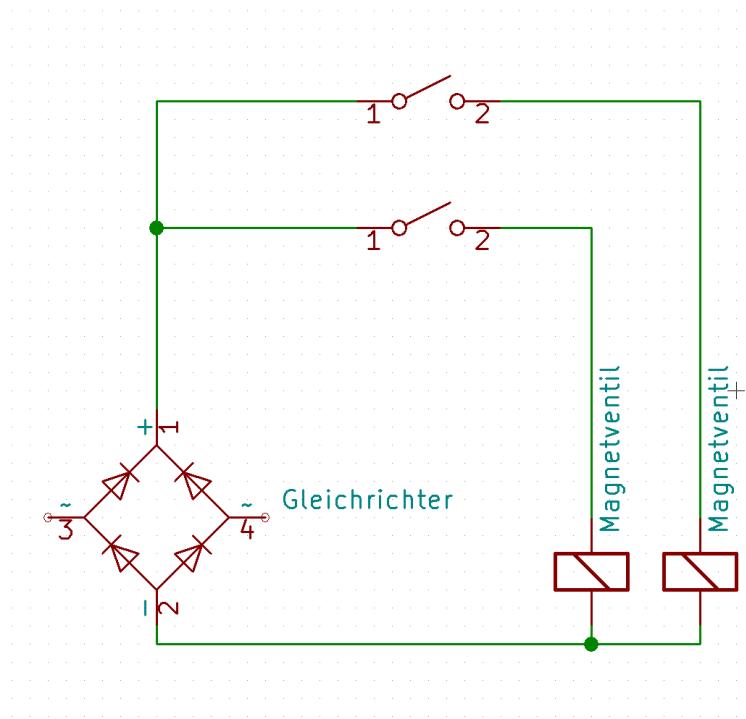


MIDI für Kirchenorgel – 1

Publiziert am 20. Juni 2019 von adminedw

Bei der Renovierung der Steinmeyer-Orgel meiner Kirchengemeinde hatte ich erstmals näheren Kontakt mit der Technik dieses Instruments. Anders als die historischen Kirchenorgeln (1), bei der Tastendrücke über mechanische Elemente die Ventile der Pfeifen betätigen, arbeitet dieses Instrument aus den 1950er Jahren mit elektrischen Schaltkontakten.



Über einen Transformator mit Gleichrichter (2) wird eine 12V-Spannung erzeugt, die über die Tasten im Spieltisch an die Magnetventile zu den Orgelpfeifen geführt wird (3) und diese ansteuert (4). Vorteil dieses Systems gegenüber der mechanischen Ansteuerung ist u. A. die unproblematische räumliche Verteilung von Spieltisch und Pfeifen.

Die elektrische Ausführung unserer Orgel brachte mich sofort auf die Idee, die Orgel mit einer MIDI-Schnittstelle auszurüsten. Damit könnte man

- Über MIDI-Out mittels eines Orgel-Expanders (Viscount CM-100) zusätzliche Klänge erzeugen.
- Die Orgel über MIDI-In mit anderen Tasteninstrumenten von Ferne spielen (um beim Instrumentenwechsel in der Kirche Laufwege zu sparen).
- Koppeln bereitstellen, die die Orgel nicht hat: P<I, I<II etc.
- Über einen MIDI-Player die Orgel alleine spielen lassen.
- Tastendrücke mit einer Fernsteuern simulieren zum Stimmen der Zungenregister ohne menschliche Hilfe.

Kaufen oder Basteln?

Fertige MIDI-Schnittstellen für den Einbau in Geräten gibt es von verschiedenen Anbietern, z.B. von [Doepfer](#). Hier gibt es Module zur Umsetzung von MIDI in 64 Einzelleitungen als Eingang bzw. Ausgang im TTL(5V)-Pegel (kostet ca. 100 €). Für den Einsatz in der Orgel wäre zusätzlich eine Umsetzung von 12V in 5V (Spannungsteiler

mit 2 Widerständen) und 5V in 12V (Transistoren) nötig. Die für 3 Manuale à 56 Tasten, das Pedal mit 30 Tasten und die 29 Registermagnete wären aufgerundet 256 Leitungen nötig. Zusätzlich zu den gekauften Modulen wäre also in jedem Fall ein erheblicher Aufwand an Elektronik nötig.

Ferner ist die Konfiguration der Module nur bedingt an die Notwendigkeiten in der Orgel-Installation geeignet, so hätte jedes 64-Bit-Modul 3 eigene 5-polige MIDI-Schnittstellen. Für eine wirklich sinnvolle Nutzung wäre zudem eine Bedieneinheit nötig (Anzeige, Tasten), die in jedem Fall selbst zu entwickeln wäre.

Insgesamt wäre der Entwicklungs-, Herstellungs- und Verkabelungs- Aufwand bei Verwendung einer Fertiglösung also nicht viel geringer, als wenn man gleich alles selbst macht. Ausnahme wäre der Einbau einer Komplettlösung, wie z.B. das [Viscount Pipe Interface](#), das genau für die Verwendung in Pfeifenorgeln konzipiert wurde – allerdings preislich in einer anderen Liga spielt.

Damit war meine Entscheidung zugunsten einer reinen Selbstbaulösung gefallen.

Anmerkungen

- (1) Heutige gebaute Orgeln sind wieder mechanisch. Angeblich hat das u. A. spieltechnische Gründe.
- (2) Der Transformator arbeitet mit „Drehstrom“, d.h. 3-phäsig, ebenso der Gleichrichter, so dass ohne jede weitere Glättung eine Gleichspannung mit nur 6% Restwelligkeit erzeugt wird.
- (3) Je nach Orgelbauer wurden zwischen 12V und 24V verwendet und der Schalter entweder in die „+“ oder „-,-“ Seite („Hi-Side“/„Low-Side“) eingefügt.
- (4) Genau genommen steuert das relativ schwache Magnetventil nur den Luftstrom in ein zweites, pneumatisches Ventil, dass dann den Luftstrom in die Pfeife freigibt, sofern auch der Luftstrom für die Pfeifenreihe dieses Registers durch ein Magnetventil freigegeben wurde.

[Teil 2](#)

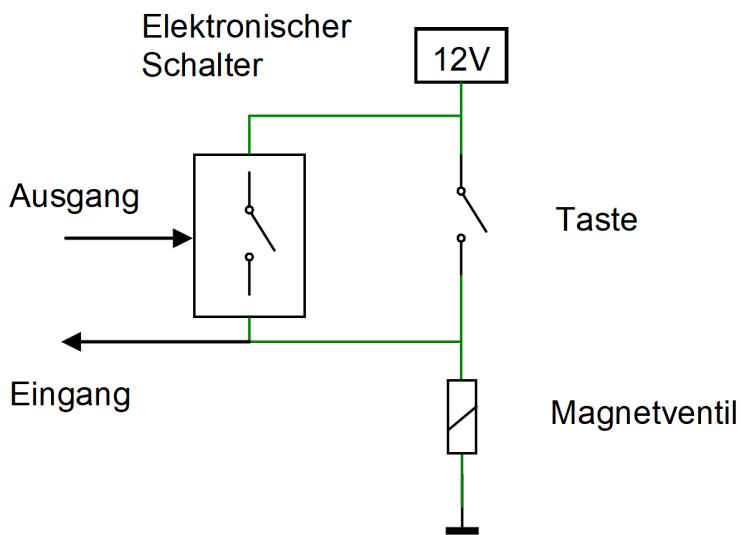
Dieser Beitrag wurde unter [Basteln](#), [Mikrocontroller/Arduino](#) abgelegt und mit [Kirchenorgel](#), [MIDI](#) verschlagwortet. Setze ein Lesezeichen auf den [Permalink](#).

MIDI für Kirchenorgel – 2

Publiziert am 23. Juni 2019 von adminedw

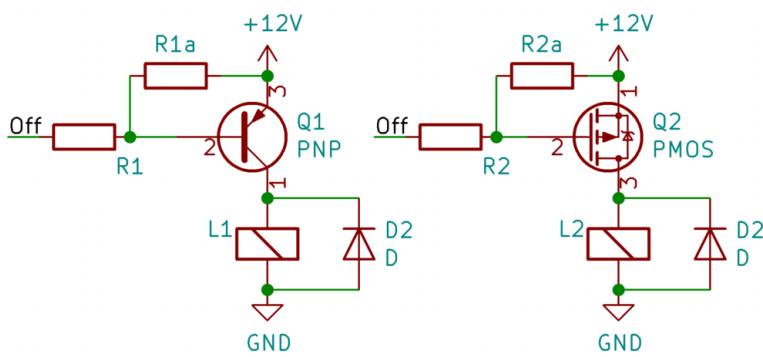
Ansteuerung und Abfrage der Pfeifenmagnete

Für die Kirchenorgel sind 2 Ergänzungen der Funktionsweisen machbar. Ansteuerung der Pfeifen durch die Elektronik (z.B. zur Realisierung eines MIDI-Eingangs) und Abfrage des Zustands der Tasten am Spieltisch (u.A. zur Weiterführung an den MIDI-Ausgang). Geschaltet und „gelesen“ werden müssen dabei die 12V, mit denen die Magnetventile gesteuert werden:



Schalten der 12V

Das Schalten der 12V könnte durch PNP-Tranistoren oder P-Kanal-Mosfets erfolgen (1)(2):

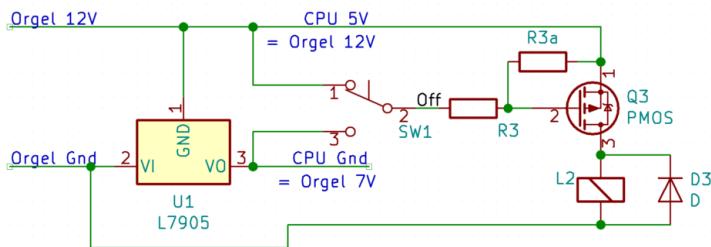


Hier ist am Basiswiderstand R1 bzw. Gatewiderstand R2 ein 12V-Signal „Off“ nötig (12V=Transistor sperrt, 0V=Transistor leitet). Wegen der geringen Verluste habe ich mich für die MOSFETS entschieden.

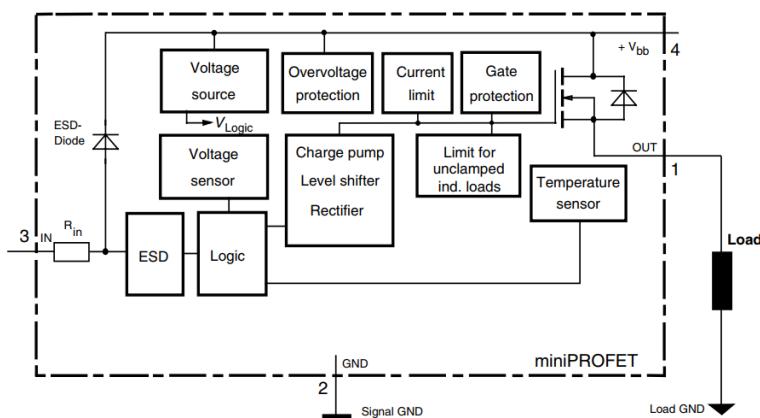
Verwendet man diese Standard-Schaltungstechnik, so ist schon am Widerstand R1/R2 ein 12V Signal nötig oder ein Logikausgang, der nur gegen Masse („Gnd“) schaltet und unempfindlich für 12V ist (z.B. Open-Collector). Letzteres würde jedoch ein zusätzliches Bauteil bedeuten, da Ports oder Schieberegister mit solchen Ausgängen nicht verfügbar sind.

Es gäbe noch 2 Möglichkeiten, auf 12V in den Schaltkreisen für die Porterweiterung zu verzichten:

- Statt gemeinsame Masse von Orgel und Mikrocontroller gemeinsame Versorgungsspannung, dafür „Anhebung“ der Masse der Orgel auf die der CPU. Aus Zum Durchschalten der Transistoren würde die CPU Gnd = Orgel 7V ausreichen. (3)



- Verwendungen von High-Side-Schaltern, die z.B. in der Automobiltechnik oft eingesetzt werden.



BSP 452: 40V – 0,7A, 0,2Ohm, TTL-Eingang, Kurzschlussfest, Thermisch gesichert

Zum Entwicklungszeitpunkt waren solche Schalter aber für Hobbyzwecke schwer erhältlich bzw. recht teuer. Daher habe ich mich gegen ihre Verwendung entscheiden. Rückblickend wäre jedoch der obige BSP 452 dafür gut geeignet. Dieser würde gegenüber der obigen Schaltung Transistor, Diode und 2 Widerstände einsparen und zudem die Sicherung, die oben nicht eingezeichnet ist. Ferner könnte alles mit 5V statt mit 12V betrieben werden. Die Eingangsschaltung müsste jedoch weiter 12V auf 5V reduzieren.

512 Port-Bits

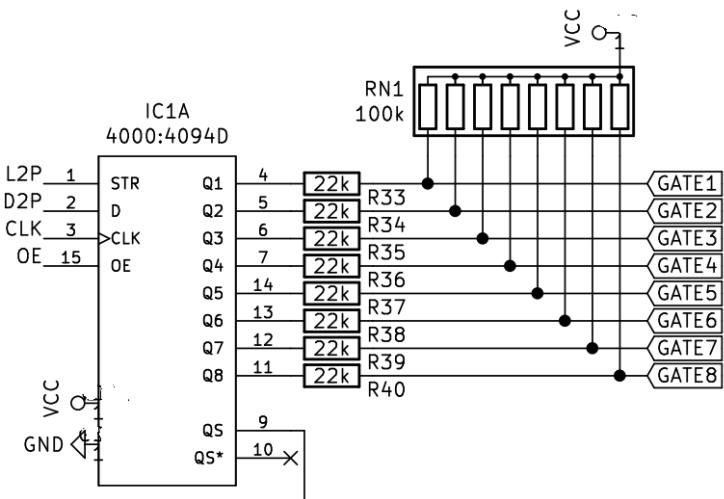
256 Ein-/Ausgänge in 12V sind schon eine gewisse Herausforderung, zumal für Ein- und Ausgang jeweils getrennte Leitungen nötig sind. Kein normaler Mikrocontroller verfügt über so viele Ports. Ein Porterweiterung wäre möglich durch

- seriell angeschlossene Schieberegister (Vorteil: Einfach anzusteueren, leicht kaskasierbar; Nachteil: Keine Fehlertoleranz im Protokoll)
- I2C Portbausteine (mittlerer Aufwand in Software, halbwegs robustes Protokoll)
- Slave-CPU's über SPI o.ä. (Vorteil: hohe Störsicherheit bei entsprechendem Protokoll, Nachteil: hoher Software-Aufwand, Extra Slave-System nötig)

Die Wandlung von 5V auf 12V scheint zunächst in jeder Variante gleich aufwändig zu sein; dies ist jedoch nicht der Fall: Die Schieberegister sind in können in der 4000er-CMOS-Ausführung direkt mit 12V betrieben werden,

so dass nur die wenigen Schnittstellen-Leitungen zur CPU eine 5V-12V-Wandlung benötigen, statt der 256 Leitungen zur Ansteuerung der Transistoren. Dies war letztlich ausschlaggebend für die Entscheidung zugunsten der Schieberegister 4094.

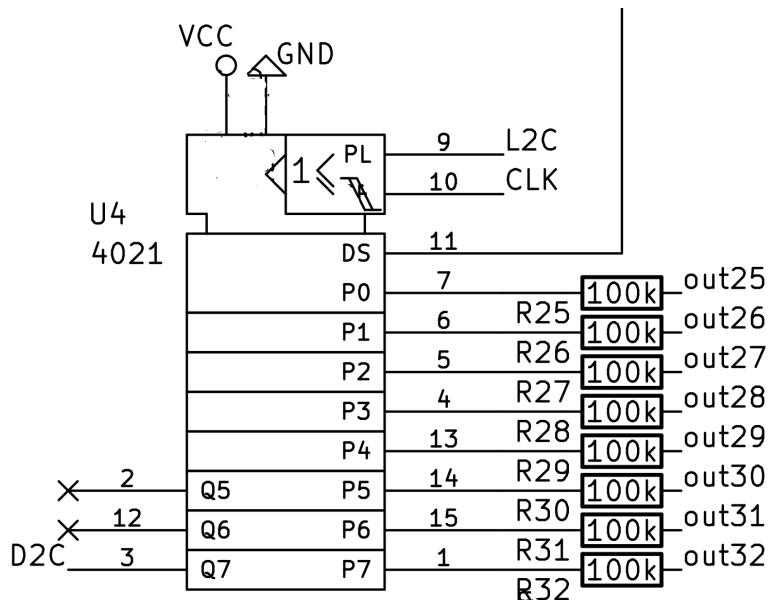
Die serielle Anordnung macht es nötig, immer alle Daten zu schreiben, nicht nur die, die sich geändert haben. Ebenso die Taktrate soll 1 MHz nicht überschreiten, wegen der recht großflächigen Anordnung der vielen Platinen und den sich ergebenden Leitungslängen. Würde man alle 256 seriell übertragen, würde das 0,256 ms dauern (plus Software-Overhead), das wäre für die angestrebte Übertragung/Abfrage jede Millisekunde schon recht lang. Da die Schaltung zur Ansteuerung der 256 Leitungen sowieso auf mehrere Platinen (etwa 8) verteilt werden muss, war es nahe liegend, die Datenleitungen D2C (Lesen: Daten an CPU) und D2P (Schreiben: Ansteuerung der Transistoren durch die CPU) für jede Platine einzeln zu realisieren. So können 8 Bit in einem Taktzyklus parallel gelesen und geschrieben werden. Dadurch wird die Länge der seriellen Übertragung auf 32 Bit reduziert.



Schaltbild-Auszug: 4094 Schieberegister zur Ansteuerung der Transistoren

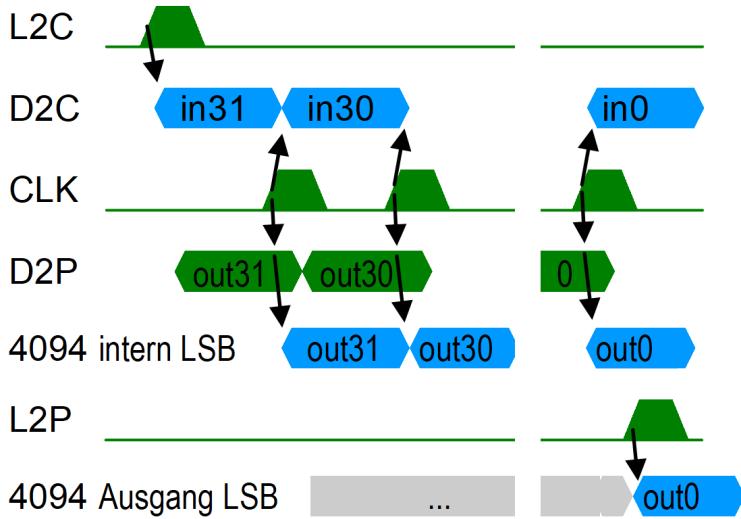
In den Eingang „D“ kommen über D2P die seriellen Daten von der CPU (beim ersten Schieberegister) bzw. vom vorherigen Schieberegister (Ausgang QS) bei den folgenden. Mit einer steigenden Flanke an CLK werden diese eingelesen. Sind alle Bits übertragen, so schreibt ein kurzer Impuls auf L2P die Daten in das Ausgangsregister, so dass diese auf Q1-Q8 erscheinen und die Transistoren über R33-40 ansteuern. OE schaltet die Ausgänge frei. Solange die Ansteuerung noch nicht arbeitet wird diese Leitung von der CPU auf Low gehalten, so dass die Transistoren über RN1 gesperrt werden.

Analog werden zum Lesen der Zustände der Tasten/Magnete Schieberegister 4021 verwendet (4):



Schaltbild-Auszug: 4021 Zum Lesen des Zustands der Tasten/Transistor-Ausgänge

Über R25-32 sind die parallelen Eingänge mit den Tasten/ Magneten/ Transistor-Ausgängen verbunden. Der aktuelle Status wird durch einen Impuls auf „L2C“ im Schieberegister gespeichert. Das oberste Bit (P7) erscheint an Q7 und wird an die CPU geführt („D2C“). Mit jedem Taktimpuls auf „CLK“ wird das nächste gespeicherte Bit an Q7 weitergeschoben. Über „DS“ wird Q7 vom vorherigen 4021 nachgeschoben (5).



Lesen: L2C lädt das Schieberegister 4021. Das oberste Bit liegt jetzt am 4021-Ausgang „D2C“. Mit jedem von 31 folgenden Taktimpulsen wird hier das nächste Bit durchgeschoben und kann von der CPU gelesen werden. Beim Schreiben legt die CPU das Datenbit (beginnend mit dem obersten) vor dem CLK-Impuls an „D2P“ an. Die 32 Bits werden von unten nach oben geschoben und nach dem letzten CLK mit dem L2P-Signal in das Ausgangsregister übernommen (6).

Die Anzahl der Leitungen zum Datenaustausch wird dadurch verringert auf zweimal 8 für die Daten (Lesen und Schreiben finden ja gleichzeitig statt) und 4 für die Steuersignale L2C, L2P, CLK und OE, also 20 Leitungen.

Teil 3

Anmerkung

- (1) Die Widerstände von Gate/Basis zu 12V sollen einen definierten Zustand sicherstellen, wenn kein Signal anliegt.

- (2) Die Freilaufdioden sind zum Schalten der induktiven Last nötig.
- (3) Orgel Gnd wäre dann aber aus Sicht der CPU -7V, was bei der Eingangsschaltung zu berücksichtigen wäre.
- (4) Die Symbole für dieses IC stammen aus einer anderen Bibliothek, bei der die Bits 0...7 heißen, statt 1...8 wie beim 4094
- (5) Am ersten 4021 wird QS des letzten 4094 über DS hineingeschoben. Somit erscheint nach dem 32. Taktimpuls das oberste der zuletzt in die 4094 geschrieben Bits am Lese-Ausgang D2C. Dadurch kann beim Selbsttest mit Testmustern geprüft werden, ob die Kette der Schieberegister in einem Modul funktioniert.
- (6) Kleiner Fehler im Diagramm: Mit dem 31. CLK-Impuls erscheint das letzte Bit „ino“ des 4021 an D2C. Aber erst mit dem 32. CLK-Impuls wird das letzte zu schreibende Bit „outo“ in den 4094 übernommen

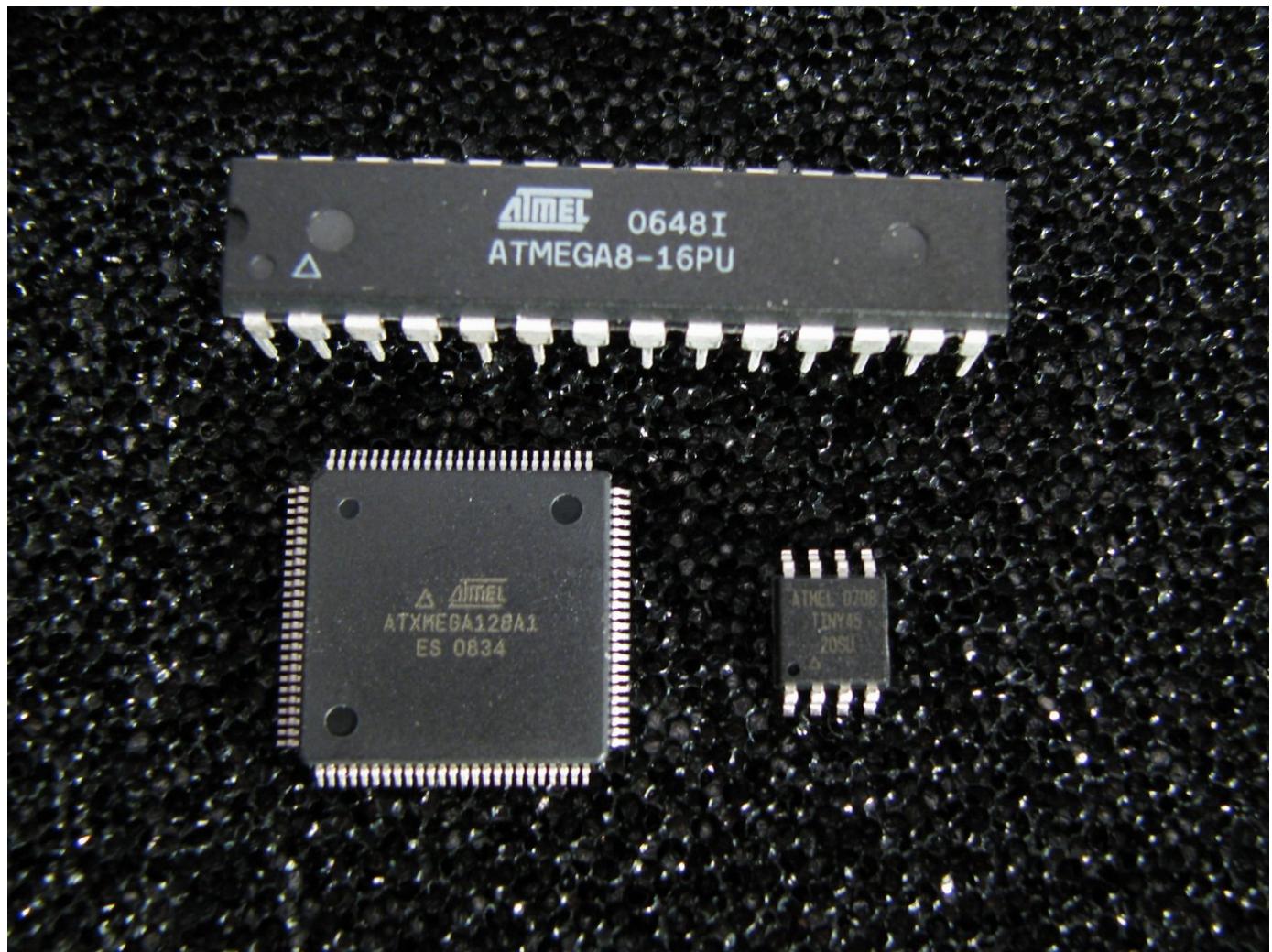
Dieser Beitrag wurde unter [Basteln](#), [Mikrocontroller/Arduino](#) abgelegt und mit [Kirchenorgel](#), [MIDI](#) verschlagwortet. Setze ein Lesezeichen auf den [Permalink](#).

MIDI für Kirchenorgel – 3

Publiziert am 25. Juni 2019 von adminedw

Mikrocontroller und Systemarchitektur

Schon seit vielen Jahren arbeite ich in meiner Freizeit mit den CPUs von ATMEL (inzwischen übernommen von Microchip). Die Entwicklungswerzeuge sind mir halbwegs vertraut, für üblichen Steuerungsaufgaben reicht die inzwischen angestaubte 8-bit-Architektur mit bis zu 20 MHz aus.



ATMEL CPUs, oben in der der bastelfreundlichen DIP-Ausführung, unten SMD

Von Springob – Eigenes Werk, CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=5054949>

Warum ATMEL?

Ich habe in meiner Bastelkiste auch STM32, ESP8266, ESP32-Boards herumliegen. Diese sind viel leistungsfähiger, und teilweise mit WLAN besser an fortschrittliche Bedienung mit Smartphone und Tablets anzupassen. Aber: sie haben zu wenige Ports (s.u.) und sind für mich bis auf einige „Hello World“-Projekte Neuland. Mit dieser Hypothek wollte ich so ein Großprojekt nicht zusätzlich belasten.

Folgende Ein-/Ausgänge sind nötig:

- 2 x 8 Bit Datenleitungen für die serielle Ansteuerung der Schieberegister plus 4 Steuerleitungen (20 Leitungen)
- 4 Datenleitungen plus 2 Steuerleitungen für das LCD
- 1 Leitung A/D für die Tasten (erzeugen Spannung über geschaltetes Widerstandsnetzwerk, die eingelesen wird)
- 2-3 x 2 serielle Leitung (RxD/TxD) für MIDI, RS232 (PC-Anschluss), Kommunikation mit anderem Mikrocontroller (als Option)

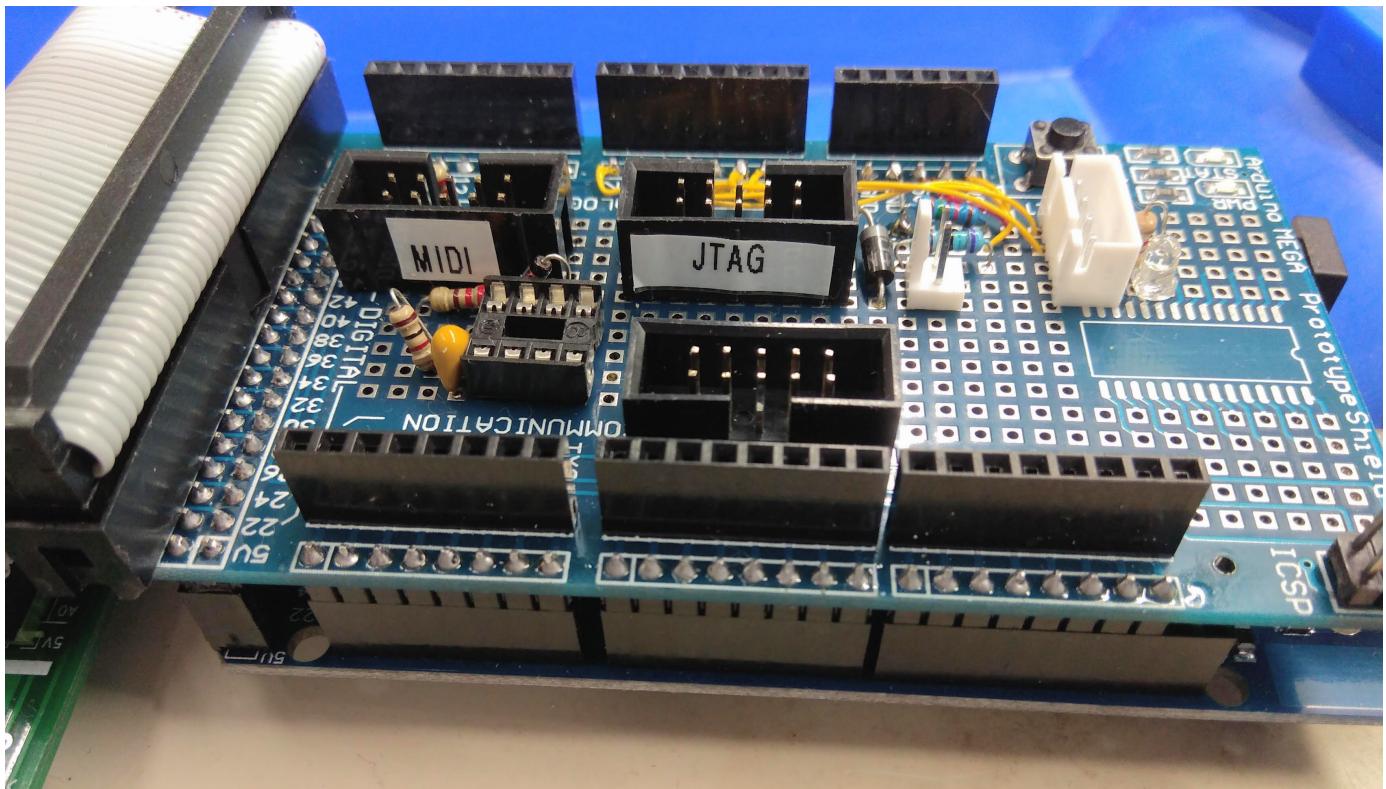
Das macht minimal 31 Port-Leitungen, wobei ein paar Reserven für spätere Erweiterungen und eigene Leitungen für das JTAG Interface (4x) sinnvoll wären.

Die DIP-Version ATMegas mit maximal 40 Pins kommen damit nicht in Frage, da diese damit jetzt von vornherein komplett ausgelastet wären und ferner keine 3 seriellen Schnittstellen haben. Kurzfristig plante ich, die Aufgaben auf 2 CPUs zu verteilen, die miteinander kommunizieren (z.B. Ein-/Ausgabe zu den Tasten/Magneten und Benutzerinterface. Der erhöhte Entwicklungsaufwand schreckte mich aber doch ab.

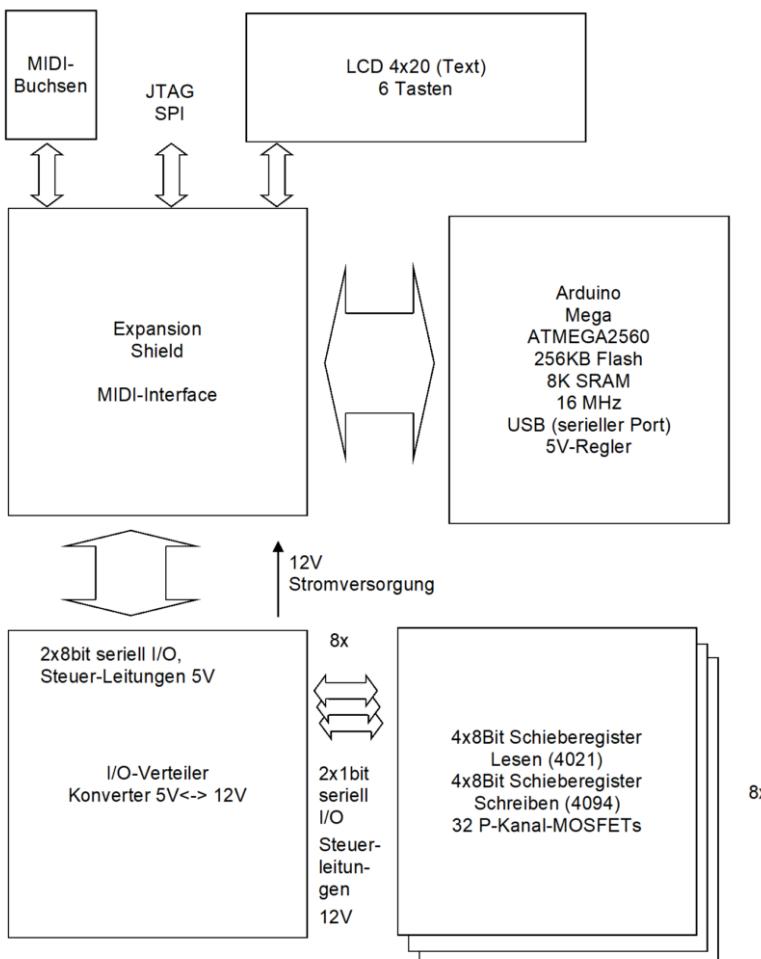
Arduino

Es bleiben also die ATMega CPUs in SMD-Ausführung, was zum schnellen Selbstbau leider ungeeignet ist. Auf der Suche nach Fertigmodulen z. B. vom ATMega 1280 stieß ich unerwartet auf einen alten Bekannten: Den [Arduino](#), hier in Form des Arduino Mega. Ich finde die Arduino-Idee großartig, denn sie ermöglicht einen niederschwelligen Zugang zur Arbeit mit Mikrocontrollern und bietet eine unglaubliche Menge an Zubehör mit Bibliotheken. Für ernsthafte Projekte hatte ich jedoch noch nie Arduinos eingesetzt: Die Entwicklungsumgebung ist wenig leistungsfähig im Vergleich zum AVR Studio, v. a. was Debugging angeht. In Echtzeit-Systemen habe ich zudem nicht viel übrig Code, in denen massiv „delay“-Schleifen enthalten sind und der auf Portbits mit [aufwändigen](#) Unterprogrammen statt mit schneller Bitmanipulation. Ich wollte daher keine Arduino-Bibliotheken verwenden, zumal das dort verwendete C++ auf einem Mikrocontroller auch nicht sehr performant ist und ich reines „C“ bevorzuge. Wie ich jedoch herausfand, kann man die Arduinos auch perfekt als reine Hardware benutzen, d.h. direkt per SPI/JTAG programmieren und den Bootloader deaktivieren.

Daher setzte ich bald einen Arduino Mega mit einem Prototype-Shield (kleine Lötfläche für eigene Schaltungen) als Zentrale ein:



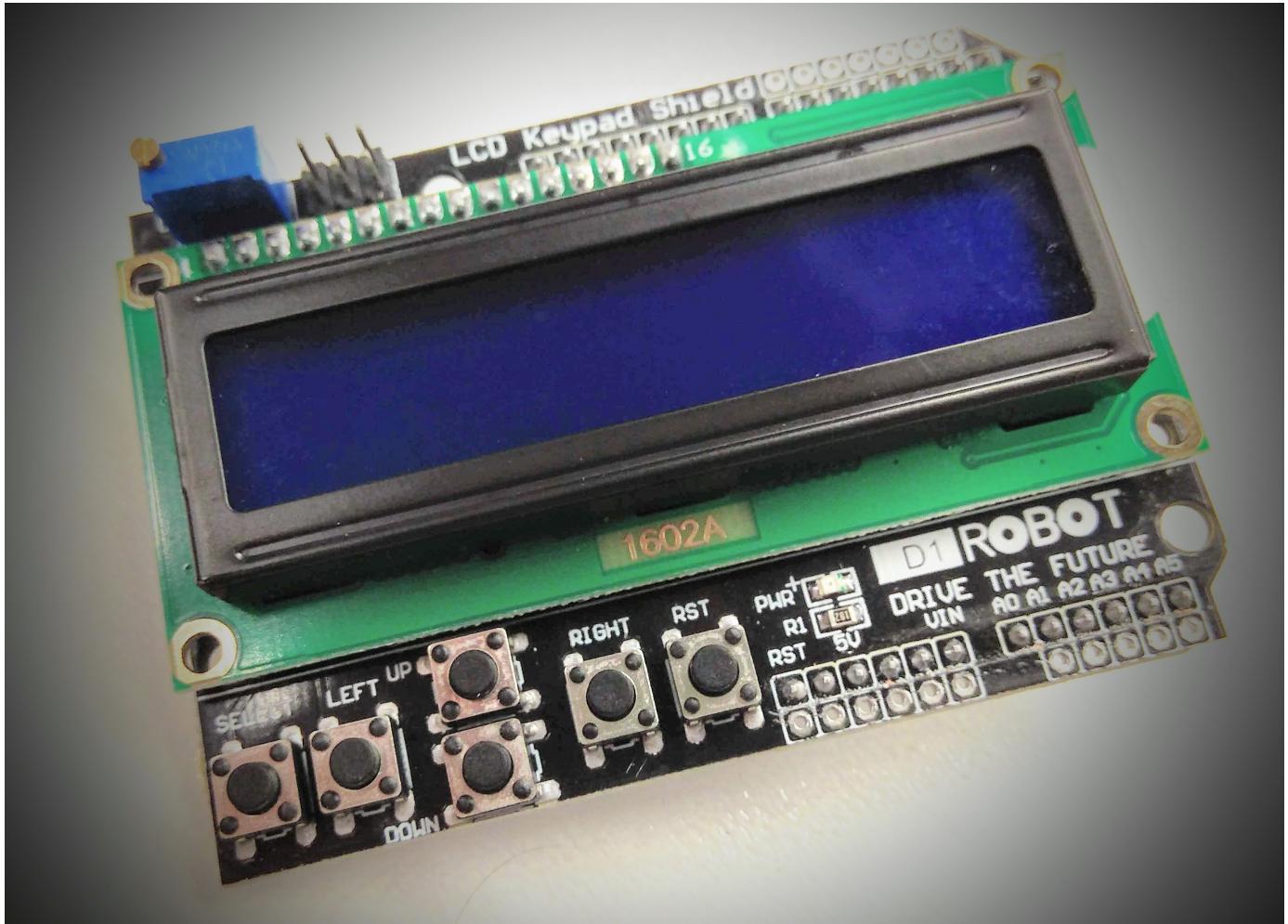
Der Arduino Mega befindet sich unter dem aufgesteckten Prototype-Shield



Über das aufgesteckte Prototype-(im Bild „Expansion“)-Shield steuert der Mikrocontroller die gesamte Peripherie

Auf dem Expansion-Shield befinden sich außer der Standard-MIDI-Ansteuerung mit Optokoppler nur wenig Elektronik. Hauptsächlich werden die Portleitungen PAo...7, PB 0...7 etc. an die entsprechenden Buchsen zum Anschluss der Peripherie weitergegeben.

LCD



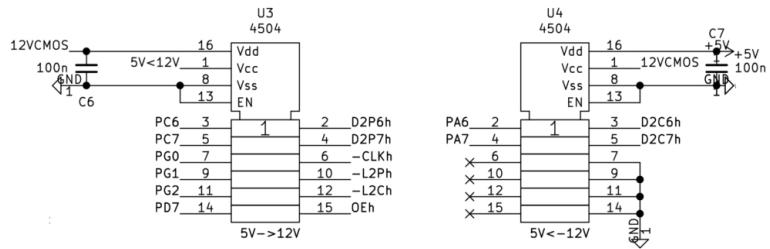
LCD-Keyboard-Shield. Die Tasten verändern über eine Widerstandsnetzwerk die Analogspannung an einem A/D-Eingang – nur eine Leitung für alle Tasten nötig!

Für die ersten Tests bot es sich an, ein fertiges LCD-Keyboard-Shield zu verwenden. Mit 2×16 Zeichen ist es aber knapp bemessen, so dass es für den wirklichen Einsatz durch ein 4×20-Zeichen-LCD ersetzt wurde. Für diese Standard-Text-LCDs mit dem [HD44780](#)-Controller gibt es viele Bibliotheken (1).

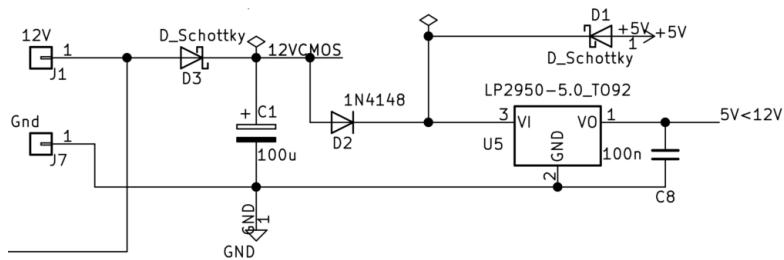
Schicker als so ein schnödes Text-Display wäre ein grafikfähiges, am besten mit Touch. Die gibt es und sie sind nicht teuer – aber belegen schon wieder sehr viele Ports und bedeuten natürlich viel mehr Softwareaufwand. Ausprobiert habe ich auch ein Display von [Nextion](#), das die Entwicklung der Benutzeroberfläche am PC erlaubt und mit überschaubarem Aufwand seriell mit dem Mikrocontroller interagiert. Letztlich beschloss ich jedoch, eine grafische Benutzeroberfläche als Option für später anzusehen, zumal das in der Orgel selbst schwer unterzubringen wäre.

Die Signalverteilung an die Schieberegister/MOSFETs

Der I/O-Verteiler besteht im wesentlichen aus den CMOS 4504 Konvertern, die 5V in 12V wandeln (links), bzw umgekehrt (rechts):



Hier mal Details zu Problemen der Hardware-Entwicklung: Anders als im Datenblatt versprochen, arbeiten die 4504, wenn nur eine von beiden Versorgungsspannungen anliegt, nicht zuverlässig (2). Daher wurden manchmal über das „OE“-Signal die Transistoren freigeschaltet wurden, wenn die 12V-, aber nicht die 5V-Versorgungsspannung anlag. Da aber die Schieberegister 4094 ohne die CPU (5V fehlt ja) nicht initialisiert werden, liegen an deren Ausgängen nach dem Einschalten zufälligen Werte an, die dann u.U. über die MOSFETs die Magnetventile aktivierten. Um dies zu vermeiden, wurden im Rahmen des Entwicklungsprozesses die entsprechenden 4504 von einem eigenen Spannungsregler (3) versorgt, der seine Spannung sowohl von den 5V der CPU als auch den 12V der Orgel bezieht, in der Schaltung als „5V<12V“ bezeichnet:



Der LP2950 versorgt einige der Konverter 5V->12V mit 5V

Leider gab es auch mit dieser Schaltung gelegentlich ein kurzes (< 0,1 s) Durchschalten der Transistoren beim Einschalten, so dass die 12V für die Transistoren letztlich doch durch eine kleine Relais-Schaltung von der CPU verzögert aktiviert werden mussten.

■ [Teil 4](#)

Anmerkung

(1) Leider viele davon nur für Arduino, wenige generisch für alle ATMEL CPUs in „reinem“ C. Zudem musste ich feststellen, dass viele davon sich nicht an die Timing-Vorgaben des Datenblatts halten und ich sie sicherheitshalber umgeschrieben habe.

(2) Wie sie nämlich auf der Seite ohne Versorgungsspannung das oV-Eingangssignal interpretieren, ist nicht sicher: Fast immer als „o“ (wie man das erwarten würde), leider gelegentlich aber auch als „1“. Zu dem Problem fand ich einen Eintrag im [TI-Support-Forum](#) einen Leidensgenossen: „the 5V power supply is not always present... the device ... output switching unexpectedly“. Ausweichende Antwort “The logic forum engineers are experts and more familiar with the [CD4504B](#). They will give your answers“. Deren Antwort konnte ich allerdings nicht finden, trotzdem wurde das Problem als erledigt gekennzeichnet.

(3) Über die Diodenschaltung D2/D1 erhält der Spannungsregler die höhere der beiden Eingangsspannungen. Bei 12V gar kein Problem, bei alleiniger 5V von der CPU fallen aber von den 5V (nur!) 0,08 V am LP2950 (bei 0,1 mA Stromentnahme) ab, hinzu kommen 0,28 V von der Schottky-Diode. Somit ist die Versorgungsspannung der 4504 um 0,36 V niedriger als die der CPU. Die Eingangssignale von der CPU können also um 0,36 V über der

Versorgungsspannung des 4504 liegen, was jedoch niedrig genug ist, um nicht die [Schutzschaltung](#) der Eingänge zu überlasten.

Dieser Beitrag wurde unter [Basteln](#), [Mikrocontroller/Arduino](#) abgelegt und mit [Kirchenorgel](#), [MIDI](#) verschlagwortet. Setze ein Lesezeichen auf den [Permalink](#).

Eine digitale Welt

Proudly powered by WordPress.

MIDI für Kirchenorgel – 4

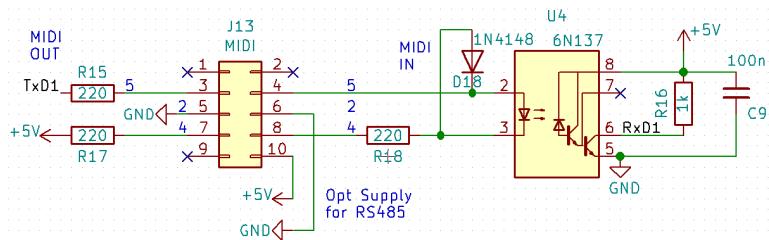
Publiziert am 14. Juli 2019 von adminedw

Meine erste „richtige“ Platine

Angesichts des hohen Verdrahtungsaufwands und der geringeren Zuverlässigkeit von Handverdrahtung auf Lochrasterplatinen war die Fertigung von Platinen unumgänglich. Weder hatte ich mich bisher ernsthaft mit Platinenentflechtung beschäftigt, noch hatte ich je welche fertigen lassen.

KiCAD

Beim Zeichnen von Schaltbildern hatte ich bisher Eagle eingesetzt, was in der kostenlosen Version jedoch sehr eingeschränkt ist, was das Platinendesign angeht. Da ich mit Eagle eh nie so richtig glücklich war und für ein reines Hobby-Projekt nicht die teurere Standard-Lizenz erwerben wollte, sah ich mir [KiCad](#) an. Open-Source, weit verbreitet und sehr leistungsfähig. Wieder einmal lernte ich damit auch die Schattenseiten freier Software kennen: Die Einzelmodule stammen von verschiedenen Entwicklern und weisen eine z.T. sehr unterschiedliche Bedienphilosophie (1) auf.



Wer an Stelle des Mauscursor (Kreuz bei R18) zum Selektieren klickt, bewirkt gar nichts. Stattdessen sind für Operationen mit der Maus nach dem Positionieren Tasten zu drücken.

Zudem arbeiten wohl die wenigsten KiCAD-Programmierer unter Windows und so nervte es mich als Windows-Nutzer anfangs ziemlich, dass das Benutzerinterface völlig unintuitiv ist.

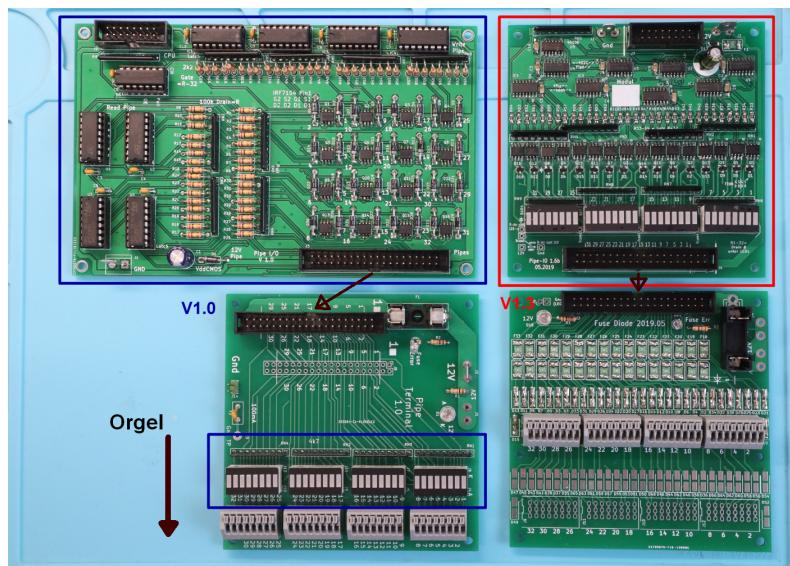
Lernt man das jedoch einmal, und auch mit den vielen Flaws umzugehen, dann hat man ein halbwegs beherrschbares System was wirklich sehr leistungsfähig ist. Eine automatisches Routing ist nicht dabei, kann aber über eine andere [kostenlose Software](#) per Export-/Import eingebunden werden. Insgesamt fühlte ich eine gehörige Portion Abenteuer bei der Erstellung der Platinen bis zum Hochladen der „Gerber“-Dateien.

Gespannt wartete ich die 15 Tage, bis meine Platinen von [JLCPCB](#) aus China kamen. Bei einem Preis von 16 € für 10 Platinen im Europa-Format erwartete ich nicht all zu viel, nicht von der Qualität und eigentlich auch nicht, dass meine Schaltung fehlerfrei war. Ich wurde positiv überrascht. Die Platinen waren von einwandfreier Qualität und funktionierten auf Anhieb.

Entwicklungsstufen

Die Verbindung zur Orgel geschieht mit einem Klingel/Telefon-Draht, der orgelseitig direkt an der Stifteleiste angelötet ist, die die Leitungen von der Tastatur mit denen zu den Pfeifenmagneten verbindet. Diesen

Klingeldraht wollte ich nicht direkt an die MOSFET-Platine, sondern an eine zwischengeschaltete Platine anschließen, die daran wiederum durch ein Flachbankabel angeschlossen ist. Das verbessert die räumliche Flexibilität für die MOSFET-Module und erleichtert deren Austausch, z. B. im Falle eines Transistordefekts.



V1.0 von 12/2018, V1.3 von 05/2019. Oben das MOSFET-Modul, unten die Schnittstelle zur Orgel mit den grauen Klemmen für den Klingeldraht. In V1.3 mit rückstellenden Sicherungen und (optionalen) Dioden für jede Leitung. für Die umrandeten Bereiche haben dieselbe Funktion!

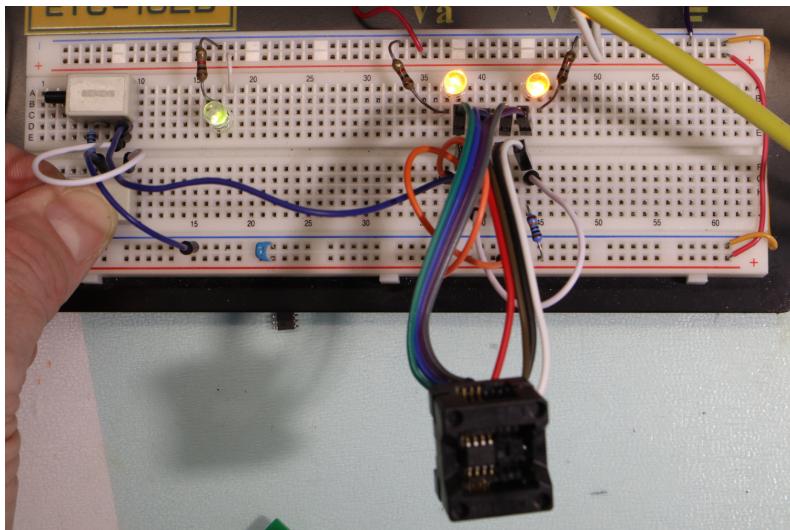
Mit zunehmender Erfahrung mit SMD-Löten stellte ich die Platine immer mehr aus SMD um, letztlich auch deswegen, weil das Europa-Format (160 mm x 100 mm) in der Orgel Platzprobleme hervorruft.

Probleme mit dem P-MOSFET

Nach der Entscheidung für den [IRF7104](#) als Schalt-Transistor bestellte ich zunächst einige Dutzend dieses Typen bei [Reichelt](#). Da sich die Schaltung zumindest anfangs als problemlos erwies, bestellte ich für die weitere Testmuster/Endprodukte immer wieder neue Exemplare bei verschiedenen Händlern, teilweise auch direkt in China (Ein Shop bei Aliexpress und LCSC). Möglicherweise zu wenig Aufmerksamkeit schenkte ich der fehlenden Antistatikverpackung der Aliexpress-Lieferung. Ich sammelte alle Transistoren in einer (Antistatik-)Box und bestückte daraus die neuen Platinen.

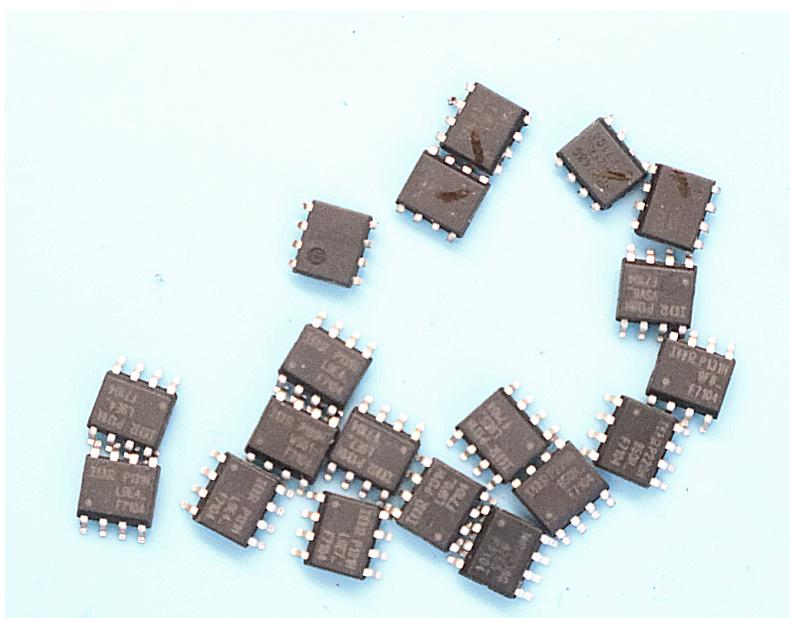
Überraschenderweise funktionierten nun einige Ausgänge nicht wie geplant. Teilweise waren sie permanent „High“, teilweise auch nur mit einem Strom von unter 1mA, was man an der schwach leuchtenden LED erkennen konnte. Andere schalteten gar nicht durch. Legte man das Gate manuell auf H/L, so flossen z.T. erhebliche Gateströme, was ja gar nicht sein dürfte.

Ich tauschte dies Transistoren aus, was mir nach dem ersten Fehlversuch sogar ohne Beschädigung der Platine gelang. Natürlich muss man die Pins am Gehäuse des MOSFETs abknipsen. Leider häuften sich die Ausfälle, so dass ich mich gezwungen sah, eine Testschaltung aufzubauen, um den Bestand in der Box zu überprüfen



Glücklicherweise gibt es für SOIC-8 Testadapter. Hier wird das Gate über 100k oder (mit dem unteren Taster) direkt an Masse gelegt. Interessant Manche defekte MOSFETs schalten nur bei letzterem, dann aber mit Gate-Strömen von 100mA!

Ein Drittel meines Bestandes erwiesen sich als defekt! Von diesen hatten 90% zwei von 6 verschiedenen Chargen-Bezeichnungen, die bei mir vorhanden waren.



Hier ein kleiner Teil des Ausschusses: oben MOSFETs, die bereits eingebaut waren

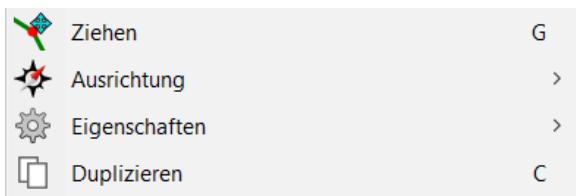
Meine nachfolgende Lieferung von [Mouser](#) habe ich wohlweislich völlig getrennt davon aufbewahrt. Eine mit diesen Transistoren bestückte Platine ging wieder auf Anhieb.

Anmerkungen

(1) Beispiele für die vielen kleinen Stolpersteine in KiCAD 5.0:

a) Gleiche Funktion, andere Tasten:

Ein Element markiert: Duplizieren = "C", Ziehen (mit Leitungen) = "G", Kopieren = nicht vorhanden;



Mehrere Elemente markiert: Duplizieren=keine Taste verfügbar (nur Kontextmenü), Ziehen="Tab", Kopieren="Ctrl-C". Warum?



b) Verschiedene Auswahl-Logik in verschiedenen Programmteilen:

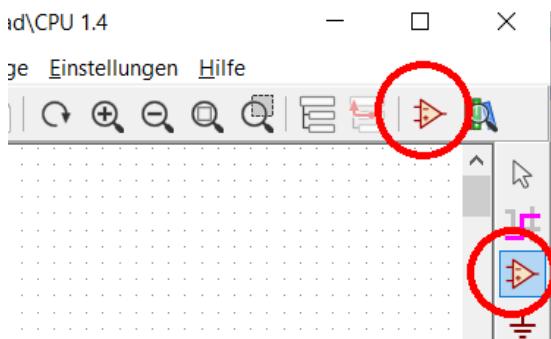
Beim Schaltplan- Editieren: Linksklick=keine Funktion, Operationen (am Element unter dem Mauspfeil) nur über Taste oder Rechtsklick (Kontextmenü).

Beim Platinen-Editieren: Linksklick=Selektieren, danach bewirkt Taste Funktion oder, falls (Achtung, Fehlerquelle!) nichts selektiert ist, bewirkt die Taste eine Operation am Element unter dem Mauspfeil, aber nur bei manchen Operationen – einige (wie z.B. Löschen) gehen nur mit der ersten Methode!

Ganz böse: Im Schaltplan-Editor kann mit der linken Maustaste ein Rahmen aufgezogen werden, der dann mehrere Elemente selektiert (dann darf also mit Links selektiert werden). Diese Auswahl wird aber dann ohne jede Rückfrage verschoben, wenn die Maus nur ein bisschen bewegt wird, weil sofort „Move“ als Standardfunktion ausgewählt ist. Also Maus absolut ruhig halten, bis mit einer Taste oder dem rechten Mausklick (jetzt nicht verwackeln!) über das Kontextmenü eine andere Funktion gewählt ist.

c) Doppelte Verwendung von Symbolen

Das OP-Amp-Symbol rechts fügt ein Bauelement ein. Dasselbe Symbol im gleichen Fenster(!) in der oberen Leiste startet den Bauelement-Editor.



d) ÜberflüssigeKlicks

- Erster Klick auf das Symbol „Bauelement-Einfügen“ rechts bewirkt gar nichts, außer der Auswahl dieser Schaltfläche.
- Klick auf die Zeichenfläche öffnet das Auswahlfenster bzw. lädt beim ersten Mal langwierig die Bibliotheken (wie wäre es mit Laden vorher im extra Thread?).
- Wählt man hier ein Element aus, dann erscheint dieses anstelle des Mauspfeils und kann durch Verschieben und Klicken positioniert werden, d.h. wird erst jetzt platziert.

Wozu war dann der erste Mausklick auf die Zeichenfläche gut?

Dieser Beitrag wurde unter [Basteln](#), [Mikrocontroller/Arduino](#) abgelegt und mit [KiCAD](#), [MIDI](#), [MOSFET](#), [Orgel](#) verschlagwortet. Setze ein Lesezeichen auf den [Permalink](#).

2 Kommentare zu **MIDI für Kirchenorgel – 4**



Abby sagt:

26. September 2019 um 10:09

JLCPCB is good; the quality is also excellent, there is a WellPCB in China, PCB assembly, cost-effective, I hope to give you a reference!

[Antworten](#)



adminedw sagt:

26. September 2019 um 10:28

Yes. My pcbs are from JLPCB. I am very satisfied about the quality.

[Antworten](#)

Eine digitale Welt

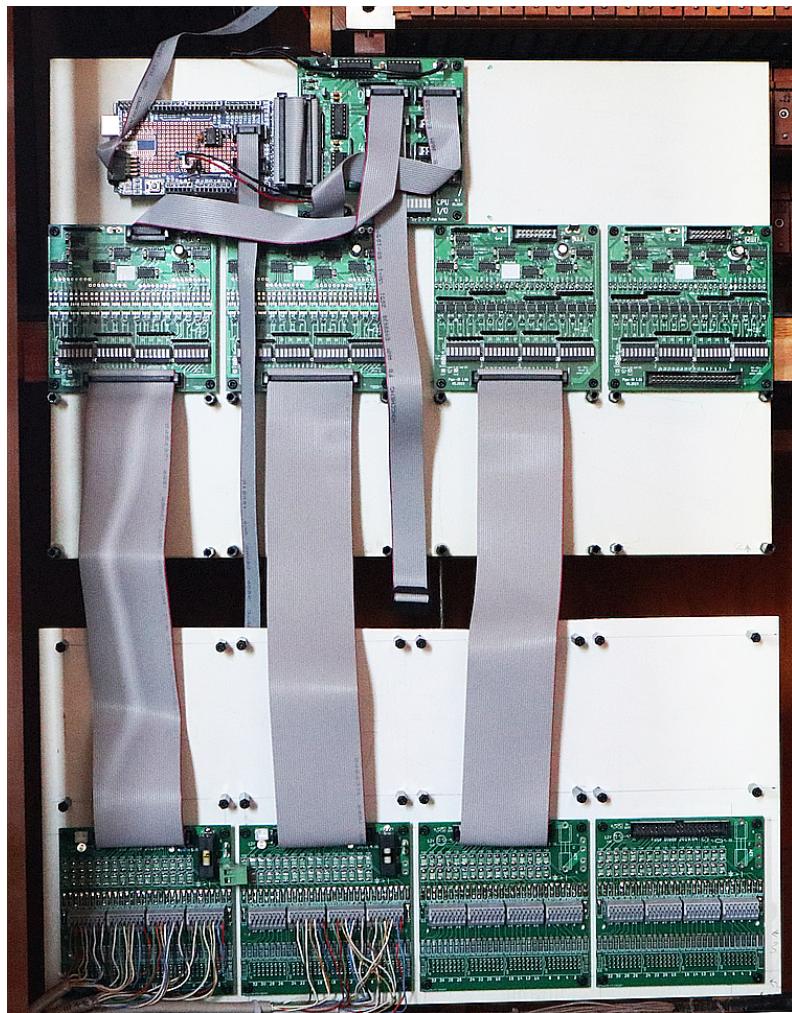
Proudly powered by *WordPress*.

MIDI für Kirchenorgel – 5

Publiziert am 29. Juli 2019 von [adminedw](#)

Einbau in die Orgel

So viele Platinen unterzubringen ist selbst in einen relativen „leeren“ Orgeltisch nicht ganz einfach. Wegen der besseren Zugänglichkeit bei Fehlersuche und Reparaturen entschied ich mich gegen ein Rack-Gehäuse zugunsten einer flachen aber ausladenden Anordnung fast direkt unter der rückwärtigen Wartungstüre.

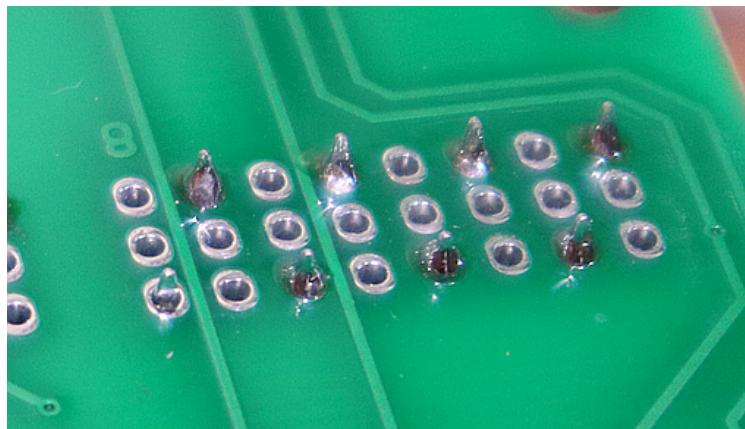


Oben links ganz klein: Die Arduino-Mega-CPU mit aufgesetztem Expansion-Shield, rechts daneben der I/O-Verteiler zu den MOSFET-Platinen. Die lange Flachbandkabel führen von dort zu den Buchsenleisten, wo die Leitungen zur Orgel eingesteckt sind.

In unserer Orgel ist das 3. Manual vorhanden und elektrisch funktionsfähig, aber die dafür geplanten Pfeifen wurden nie angeschafft. Es bot sich an, zunächst dessen Tastendrücke in MIDI umzusetzen, um damit einen vorhandenen externen Tonerzeuger („Expander“) anzusteuern, der Orgelklänge über eine Verstärker-/Lautsprecher-Anlage wiedergibt.

Die Funktionsfähigkeit der Schaltung an sich hatte ich hinreichend getestet, ebenso die Software zur Umwandlung von MIDI-Befehlen in 12V-Impulse und umgekehrt. Trotzdem war es ein spannender Moment, als ich die erste Ausbaustufe in die Orgel einbaute und in Betrieb nahm.

Bis auf ein wenige Löt- und Verdrahtungsfehler funktionierte alles auf Anhieb! Man konnte auf dem 3. Manual (oder die Koppeln) über den Expander ganz „normal“ Orgel spielen. Dadurch waren Klänge verfügbar, die es bisher auf dem altehrwürdigen Instrument nicht gab.



Schnell übersieht man einmal Lötstelle!

Nach einigen Wochen des störungsfreien Betriebs installierter ich die nächste Ausbaustufe für das 1. und 2. Manual. Auch hier gab es nur minimal Verdrahtungsprobleme am Anfang. Es war schon ein etweas erhebendes Gefühl, als ich vom Laptop mit USB-MIDI-Schnittstelle eine Bach-Invention auf der Pfeifenorgel wiedergeben konnte.

Dieser Beitrag wurde unter [Allgemein](#) veröffentlicht. Setze ein Lesezeichen auf den [Permalink](#).

Eine digitale Welt

Proudly powered by [WordPress](#).

MIDI für Kirchenorgel – 6

Publiziert am 25. April 2020 von adminedw

Debugging

Obwohl die Entwicklung gut vorwärts ging, ergaben sich durch die zunehmende Komplexität einige hartnäckige Probleme. Die Anbindung der Manuale an die Midi-In- und Midi-Out-Anschlüsse funktionierte zuverlässig, auch die Wiedergabe komplexer Midi-Dateien auf der Orgel lief störungsfrei, jedoch gab es dabei Fehlermeldungen, dass der MIDI-Out-, teilweise auch der In-Buffer überlief. Dies war sehr irritierend, da man keine fehlenden Töne oder Latenzen hörte; eine Vergrößerung des Puffers auf 256 Byte brachte nichts. Zunächst vermutete ich einen Fehler in der Puffer-Verwaltung.

MIDI-Buffer

Als Ring-Puffer dient ein Byte-Array midiTxBuffer[] mit den Indizes midiTxInIndex (zeigt auf nächste Position zum Schreiben in den Puffer) und midiTxOutIndex (zeigt auf die nächste aus dem Puffer auszugebende Byte. Anfangs sind beide Indizes Null; gleicher Wert für beide zeigt, dass nichts auszugeben ist.

Das Schreiben erfolgt in serial1MIDISend(). Der Index midiTxInIndex wird nach dem Schreiben erhöht. Durch die Und-Verknüpfung mit MIDI_TX_BUFFER_MASK wird der umlaufende Index immer im Rahmen der Arraygröße (die ergo eine 2er-Potenz sein muss) gehalten. Erreicht dieser Schreib-Index jedoch den Ausgabe-Index midiTxOutIndex sozusagen „von hinten“, so liegt ein Überlauf vor. In dem Fall wird der Index nicht erhöht, da sonst der komplette Puffer-Inhalt „gelöscht“ wäre. Falls danach noch ein Byte mit serial1MIDISend() gesendet würde, würde dies das vorige überschreiben (d.h. der Überlauf wird signalisiert, wenn er nur droht, ein Byte „Reserve“ gibt es noch.

```
void serial1MIDISend(uint8_t data) {
    ...
    midiTxBuffer[midiTxInIndex] = data;
    uint8_t newIndex = (midiTxInIndex+1) & MIDI_TX_BUFFER_MASK;
    if (newIndex == midiTxOutIndex) {
        // notify overflow to main by global var
        ...
    } else {
        // no overflow
        midiTxInIndex = newIndex;
    }
    UCSR1B |= (1 << UDRIE1); // Interrupt einschalten für "Senderegister leer"
}
```

Mit dem Schreiben von UCSR1B wird der Interrupt freigegeben, der ausgelöst wird, wenn das Senderegister leer ist, d. h. nach einem erfolgreichen Sendevorgang. Ist das Senderegister jetzt bereits leer, so wird der Interrupt sofort ausgelöst, falls noch ein Sendevorgang läuft, dann eben etwas später. Die eigentliche Interrupt-Senderoutine sieht dann so aus:

```
ISR(USART1_UDRE_vect) {
    if (MIDI_TX_BUFFER_NONEMPTY) {
        // there is a byte to sent
```

```

        UDR1 = midiTxBuffer[midiTxOutIndex];
        midiTxOutIndex = (midiTxOutIndex+1) & MIDI_TX_BUFFER_MASK;
        ...
    } else {
        // nothing to send
        UCSR1B &= ~(1 << UDRIE1);
        // Interrupt abschalten - wird in serial1MIDIWrite wieder gesetzt
    }
}

```

War in diesem selbst entworfenen Konzept, das ohne jedes Warten in while-Schleifen auskommt, fehlerhaft. Ich konnte bei der Analyse keinen Fehler finden und wandte mich einer anderen Möglichkeit zu.

LCD-Interface

Bereits bei der Entwicklung war mir die aus dem Internet übernommen Routine zur Ansteuerung des LCDs nicht sehr sympathisch. Zur Realisierung des Timings der recht langsamen Schnittstelle zum HD44780-kompatiblen Controller werden vielfach wait()-prozeduren verwendet, die zwar die Interrupt-Verarbeitung nicht blockieren, aber alles was im Hauptprogramm läuft, also v. A. die Verarbeitung des Midi-Eingangs und die Reaktion auf Tastendrücke etc. Allerdings dauert das Schreiben eines Zeichens nur ca. 0,1 ms – das schien ein zu vernachlässigender Wert zu sein. Allerdings wird wegen jeder empfangenen und gesendeten Noten-Information das Display aktualisiert, bei komplexen Stücken könnte das dann doch etwas mehr sein.

Leider erlaubt das einfache Debugging per JTAG keine Echtzeit-Analyse, oder eine Art Profiler, wie lang die CPU in bestimmten Prozeduren läuft. Zur Analyse nutzte ich daher (typisch für solche Mikrocontroller) einige der zahlreichen unbenutzten Port-Pins. Diese werden in bestimmten Routinen auf 1 gesetzt und danach wieder auf 0. Die gelbe Linie gibt die Zeit in der LCD-Routine wieder, blau ist die MIDI-Verarbeitung in main()



Die Sache schien also klar: die CPU ist zu sehr mit der Ausgabe auf dem LCD beschäftigt. Im Log zeigten sich dann massiv Fehlermeldungen, teilweise mehrmals pro Sekunde:



Oben Statuszeile: links: empfangene Note, Mitte: Betriebszeit (und „E“ für Fehler), rechts: gesendete Note

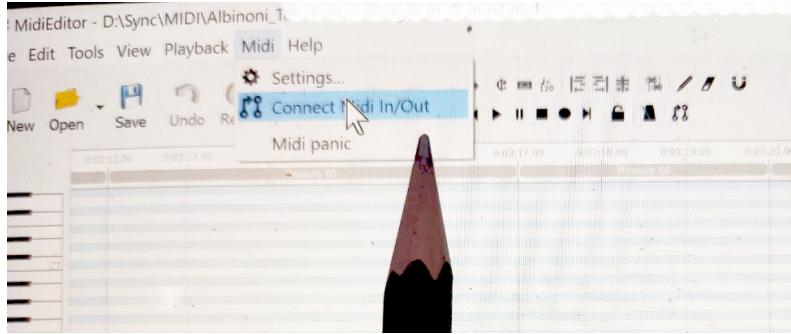
2. Zeile: Menü, 3. Zeile: Text zum Menüpunkte (hier Logeintrag), 4. Zeile: Bedeutung der 4 Softkeys.

Also reduzierte ich die Häufigkeit der Aktualisierung der empfangen bzw. gesendeten Noten in der Statuszeile auf 0,7 Sekunden. Damit waren fast alle Overflows verschwunden – aber nicht alle!

Exakt im Sekudentakt, aber erst nach ca. einer halben Minute erschien weiter ein Overflow, offensichtlich wenn die Uhr aktualisiert wird. Dies dauert nach Messung mit dem Oszilloskop ein paar Millisekunden. In der Zeit müsste ein 256 Byte großer Puffer bei einer 31250-Baud-Schnittstelle (ca 3 Bytes pro Millisekunde) aber locker ausreichen! Im Normalfall sollte der Puffer ja kaum gefüllt sein, da pro Sekunde selbst in schnellen Stücken maximal 70 Note-On/Off Befehle à 3 Byte empfangen bzw. gesendet werden, und das dauert nur 21ms.

Also überprüfte ich zunächst die Verarbeitung einzelner Midi-Befehle und musste überrascht feststellen, dass durch eine falsche Einstellung in der Midi-Konfiguration jeder Befehl 3-fach ausgegeben wurde. Das erklärte eine hohe überflüssige Last, aber nicht den Überlauf.

Die Lösung lag in einer unscheinbaren Einstellung der Midi-Player-Software beim Testen:



Ich hatte versehentlich (dank Software, die nicht mit HDPI umgehen kann) einmal eine Schleife Midi-In-Out in der Software aktiviert. Da aber in meiner Schaltung/Software empfangene Midi-Befehle an die Manuale ausgegeben und dann aber sofort als Tastendrücke registriert werden, wird jedes eingegangener Note-On/Off-Befehl wieder an Midi-Out ausgegeben (sofern der Midi-Ausgang aktiviert ist). Ergebnis ist eine Endlosschleife, die tatsächlich erst durch den Puffer-Überlauf unterbrochen wird! Nachdem ich den Menüpunkt im Player deaktiviert hatte, lief alles ohne Overflow!

Sicherheitshalber richtete ich noch eine Variable ein, in der die maximale Auslastung des Sende- und Empfangspuffers abgefragt werden kann. Sie liegt unter 5 Byte.

Dieser Beitrag wurde unter [Allgemein](#), [Mikrocontroller/Arduino](#) veröffentlicht. Setze ein Lesezeichen auf den [Permalink](#).

