3-Phasen Frequenzumrichter mit AVR

von matzetronics

Dieser Artikel nimmt am Artikelwettbewerb 2012/2013 teil.

In diesem Artikel wird ein universeller 3-Phasen Frequenzumrichter mit einem einzigen ATMega 88/168/328P beschrieben. Der ATMega übernimmt dabei die komplette Steuerung der Bedienelemente, des LC-Displays und die Erzeugung der 3 Phasen. Das Projekt sollte auf fertigen Boards wie dem Arduino 2009 oder dem Uno auch laufen, das wurde aber nicht ausprobiert. Im Gegensatz zu anderen Lösungen wird hier der Sinus nicht errechnet, sondern aus einer Tabelle gewonnen. Das spart sowohl Rechenzeit als auch Speicherplatz und ermöglicht dem MC, die gesamte Steuerung als Einzelprozessor (Single Chip) zu erledigen. Gleitkommaberechnungen kommen nicht vor.

Die Frequenz und Amplitude der Ausgangssignale sind über 3 Tasten einstellbar und können permanent in das EEPROM des MC gespeichert werden. Ebenso ist eine externe Steuerung über 2 analoge Eingänge vorgesehen. Die Drehrichtung der 3 Phasen wird über eine Steckbrücke oder Schalter festgelegt.

Eine einstellbare V/f Charakteristik erlaubt die Anpassung an viele Motoren und andere Verbraucher. Ein integrierter PID Regler für die analogen Eingänge wurde ebenso programmiert, die PID Parameter können im EEPROM abgelegt werden. Die Totzeit ist einstell- und abspeicherbar.

Im groben Zügen basiert die Software auf der Application Note AVR447 von Atmel/Microchip, ist aber erheblich geändert, um sie für den hier beschriebenen Regler zu benutzen. Ebenso wird die LCD Bibliothek von Peter Fleury benutzt, herzlichen Dank an ihn für diese schöne Software. Leider passt das Programm beim derzeitigen Stand nicht ganz in einen ATMega48, mit ein wenig Optimieren in den Stringfunktionen könnte es aber klappen. Um etwaigen Fragen vorzubeugen - nein, das Ganze läuft nicht in einem (veralteten) ATMega 8. Diesem fehlen sowohl die Timerfunktionen als auch die Pinchange Interrupts.

Daten:

Frequenzbereich: regelbar von ca. 0,6 Hz - 162 Hz

Ausgangsspannung: regelbar von 0V - 325 Vpp, maximal die vorhandene Zwischenkreisspannung

Wellenformen: Entweder Motorkurven ('Popokurven') oder Sinus, wählbar vor dem Kompilieren

Inhaltsverzeichnis

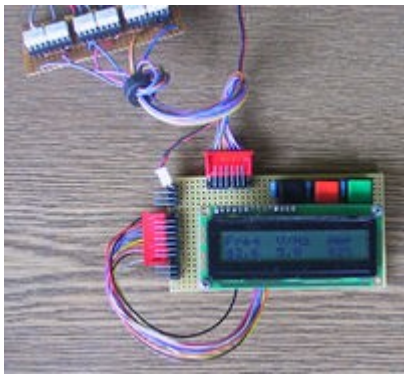
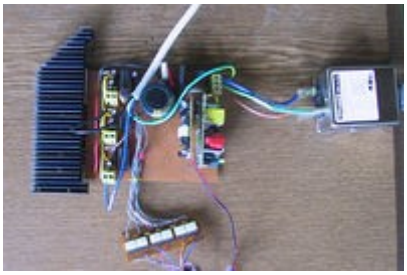
Beschreibung

Der MC ist mit einem LC-Display des HD44780 Industriestandard im 4-bit Modus und 3 einfachen (Digi-)Tastern ausgerüstet. Beim Einschalten prüft der MC das Vorhandensein einer Steckbrücke (Jumper) und schaltet dann entweder in den internen oder externen Steuermodus. Die komplette MC Steuerplatine ist galvanisch von der Motorendstufe getrennt, hierzu dienen 6 Optokoppler vom Typ HCPL3120. Diese Optokoppler liefern die Signale für klassische Halbbrücken Treiber vom Typ IR2110/2113, welche wiederum die MOSFet/IGBT Endstufen ansteuern. Die gezeichnete Endstufe ist nur als Beispiel zu verstehen, natürlich sind auch andere Konfigurationen denk- und machbar.

Die 3 Timer des ATMega laufen synchron und es werden die OC (Output Compare) Ausgänge aller Timer benutzt. Einer der Timer (Timer 1) liefert einen Overflow Interrupt, der dazu benutzt wird, die Schrittgeschwindigkeit der Sinuserzeugung einzustellen und danach neue PWM Werte in alle 3 Timer zu schreiben. Der Quellcode erhält dazu auch ausführliche Kommentare. Die Schrittgeschwindigkeit ist direkt proportional der erzeugten Frequenz und wird vom Benutzer eingestellt. Mehr dazu findet sich im Abschnitt "Sinuserzeugung".

Aus Gründen der Übersichtlichkeit errechnet der MC aber daraus die Frequenz in Hertz und zeigt diese auf dem Display. Ebenso wird die Amplitude der Wellenzüge über die Pulsbreite der PWM Signale eingestellt und auf dem Display in Prozent der Zwischenkreisspannung dargestellt. Da Drehstrommotore meistens mit einer V/f Charakteristik angesteuert werden, wird auch diese errechnet, bevor die Timer neu geladen werden.

Hier Fotos vom Prototyp. Der Kontrollteil wurde auf 2,54 mm Punktraster in Fädeltechnik gebaut, der Leistungsteil auf Streifenleiterplatine mit 5,08 mm Raster. Fädeltechnik ist meine bevorzugte Technik für Einzelstücke und Prototypen. Der MC befindet sich unter dem LC Display. Ganz rechts auch noch ein Bild eines meiner treuen Drehstrom Motore. Dieser Motor hat 550 Watt und arbeitete seit 1969 in meiner Brunnenpumpe. Die Pumpe ist völlig verrostet und nun ausser Betrieb, aber der Motor läuft noch wunderbar. Gefertigt wurde er von den Stephan-Werken in Hameln.


Software, Bedienung und Kompilieren

Die Software ist so konzipiert, das die PWM Erzeugung komplett in Interrupts läuft. Die Hauptschleife des Programmes erledigt die Benutzerführung mit Tastenabfrage und LCD Ansteuerung. Ingesamt gibt es zwei Anzeigemodi: Modus 1 ist das normale Menü, in dem per Taste 1 zwischen V/f Einstellung und Frequenzjustage gewechselt werden kann. Taste 2 und 3 verringern bzw. erhöhen den gewählten Wert. Der Cursor des Displays steht dabei unter dem editierbaren Wert. Eine einfache Autorepeat Funktion mit Beschleunigung erleichtert das Einstellen.

Modus 2 wird durch gleichzeitiges Drücken der Tasten 1 und 2 erreicht. Dieses Menü dient dazu, die PID Werte des Reglers und die Totzeit der Brückensteuerung einzustellen und sowohl diese als auch die Startwerte für V/f und Frequenz ins EEPROM zu schreiben. Auch hier wechselt Taste 1 zwischen den Parametern und Taste 2 und 3 dienen zum Einstellen. Gleichzeitiges Drücken von 1 und 2 speichert die Werte, gleichzeitiges Drücken von 1 und 3 verlässt den Modus ohne Abspeichern. Es werden dabei die Werte für V/f und Frequenz gespeichert, die im Menü Modus 1 gerade eingestellt sind.

Im Betrieb mit externen Signal zeigt Modus 1 die Werte an, sie lassen sich aber nicht per Taster ändern. Modus 2 hingegen ist anwählbar und funktioniert auch bei externer Kontrolle, um PID Werte und Totzeit zu justieren. Ein Interrupt wird durch den Analog-Digital Wandler ausgelöst. Dieser liest den Wert des ADC aus

und speichert ihn in globalen Variablen. Das 'switch-case' Konstrukt erlaubt es, weitere Kanäle des ADC zu nutzen, wenn das erforderlich werden sollte. Die DIP Variante des ATMega88/168 bietet diese Möglichkeit zwar nicht, hier sind alle Anschlüsse des ADC benutzt. Allerdings hat die QFP Ausführung noch die ADC Kanäle 6 und 7, diese könnten genutzt werden.

Die hier beschriebene Schaltung ermöglicht es, sowohl Frequenz als auch V/f Werte als analoge Signale anzulegen. Ohne externe Spannungsteiler wird eine Eingangsspannung von 0-5 Volt an den Eingängen erwartet.

Die LCD Bibliothek musste leicht modifiziert werden, da hier zwei unterschiedliche Ports für Daten- und Steuerleitungen benutzt werden. Die Änderungen befinden sich in lcd.h und sollten so übernommen werden, da die PWM Ausgänge des MC nicht verändert werden können.

Übrigens wurden bis auf den PID Regler und die LCD Routinen alle Softwareteile in einer Datei zusammengefasst, um dem Compiler die beste Optimierung zu ermöglichen. Als Optimierungseinstellung hat sich -Os bewährt. Vor dem Kompilieren sollte nicht vergessen werden, die benutzte Quarzfrequenz in Hz in die Projekteinstellungen einzutragen. Unter Linux sollte das Makefile mit einem -DF_CPU 16000000 ergänzt werden, in AS4 unter "Projekt Settings -> General". Wird das mitgelieferte *.aps als Projektvorlage benutzt, ist das jedoch unnötig. Das Software Paket enthält alle Dateien, um mit AVR Studio 4 das Projekt zusammenzubauen. Der Sourcecode besteht aus folgenden Dateien:

main.c, pid.c, lcd.c

Dazu kommen die Header:

vfd.h, vfdtables.h, pid.h, lcd.h

Besondere Aufmerksamkeit verdient vfd.h. Hier sind alle Deklarationen für das Projekt zusammengefasst. Wer main.c studiert, wird eine Menge Konstanten finden, sie alle werden in vfd.h deklariert. Die am häufigsten benutzten Variablen sind die Schrittgeschwindigkeit (inco), der Zeiger in die Sinustabelle(sinTableIncrement), die Amplitude (amplitude) und das V/f Verhältnis (VperHz). Diese Variablen werden direkt in Registern gehalten, um die Arbeitsgeschwindigkeit zu erhöhen. Vermutlich wäre das nicht nötig, aber als Beispiel zur Verwendung direkter Registervariablen ist es evtl. auch für andere Projekte hilfreich. Eine weitere direkte Registervariable ist (fastFlags). Dieses struct wird bitweise benutzt, allerdings sind hier nur 4 bits benutzt worden. Ein Bit für die Umschaltung zwischen externer und interner Kontrolle, dann 2 Bits für die benutzte Wellenform (entweder 'undefined' oder 'Sinus') und dann noch ein Bit für die gewünschte Drehrichtung. Die Anzeigeparameter werden über Integermathematik errechnet, da Gleitkommazahlen vermieden werden sollten. Eine regelmässige Aufgabe des Timerinterrupts ist das Aufrufen der Routine 'SpeedController' (nach 200 Timerüberläufen). Hier wird im externen Kontrollmodus der PID Regler aufgerufen und ausserdem die Anzeigeparameter errechnet. Die Anzeige auf dem LCD erfolgt im Kommandointerpreter.

Nach dem Reset werden zuerst das LCD und die Ports initialisiert und alle Variablen vorbelegt. Im Anschluss folgen die Initialisierung der Timer, der Pinchange Interrupts, des ADC und des PID Reglers. Nun werden die Jumper abgefragt und das EEPROM eingelesen. Zum Schluss erfolgt die Freigabe der globalen Interrupts und damit der Frequenzerzeugung.

Während des Vorganges werden Diagnosezahlen auf dem LCD ausgegeben, um etwaige Probleme mit 'steckengebliebenen' Tasten oder andere Probleme zu melden. Am Ende folgt die Hauptschleife, in der der 'Kommandointerpreter' (execCommand) läuft.

Der Kommandointerpreter tut nichts anderes, als erstmal nach gedrückten Tasten zu schauen. Findet er eine oder mehrere Tasten gedrückt, wird das switch-case Konstrukt abgearbeitet. Anschliessend wird das LCD aktualisiert. Eine zweite Kommandoebene tut das für das erweiterte Menü, die Arbeitsweise ist die gleiche. Bei allen Einstellungen wird dafür gesorgt, das die Werte nicht 'überschlagen' können, da das oft fatale Folgen hätte. So ist es z.B. nicht möglich, durch Drücken der - Taste die Arbeitsfrequenz von 0,1 Hz direkt auf 162 Hz umzuschalten oder die Totzeit auf 0 zu stellen.

Noch eine Anmerkung zu AVR Studio 4 - die letzte Version ist 4.19 und um Kompilierfehler zu vermeiden, sollte man die AVR Toolchain 3.3.1 - 1020 benutzen, sonst wird man mit Fehlern belästigt. (Vielen Dank an kabelbieger). Alternativ ist WinAVR geeignet - die Version 20100110 funktioniert bspw. recht gut.

Sinuserzeugung

Die drei Phasen für den Sinusausgang werden aus einer Tabelle geholt, diese befindet sich in der Datei 'vfdtables.h'. Hier liegen in einem Array die Werte für U,V und W (Heute auch oft L1, L2 und L3 genannt, früher auch als R, S und T bezeichnet). Die erzeugte Frequenz wird dadurch bestimmt, nach wieviel Aufrufen der 'TIMER1_CAPT' ISR der Zeiger in die Tabelle inkrementiert wird. Wenn 'inco' z.B. 1 ist, wird der Zeiger in die Tabelle erst nach 255 Durchläufen inkrementiert und der nächste Tabellenwert in die Timer geladen. Da die Tabelle 192 Werte lang ist, resultiert das in einer Ausgangsfrequenz von 0,635 Hertz, die sich aus der PWM Wiederholrate von 31,25 kHz/256 = 8,192ms * 192 ergibt. Ist 'inco' bei 255, wird der nächste Tabellenwert bereits nach 1/31,25kHz = 32µs geladen und das resultiert in einer Ausgangsfrequenz von 1/(32µs * 192) = 162,7 Hz. Mein Dank geht an CNCler, der das richtig erklärt hat.

Die Drehrichtung wird einfach dadurch gewechselt, das die Werte für V und W vertauscht werden, bevor die Timer geladen werden. Tabellenzugriff und das Laden der PWM Timer erfolgen gesammelt in der Timer 1 Overflow Interrupt Routine.

Diese Routine stellt somit den Kern der Sinuserzeugung dar. Der Betrieb von Halbbrücken erfordert das Einfügen einer Totzeit, in der weder der obere noch der untere Schalter der Halbbrücke angeschaltet sein darf. Würde das der Fall sein, wird die Versorgungsspannung kurzgeschlossen, was auf keinen Fall passieren darf. (Dieser Fall wird oft als 'Shoot-through' bezeichnet). Da die Totzeit stark von den verwendeten Halbbrücken abhängig ist, wurde sie einstellbar gemacht. Die Routine 'InsertDeadband' errechnet für jeden Timer die erforderliche Totzeit, bevor die Werte für die PWM in den Timer geschrieben werden.

Die Totzeit wird im EEPROM abgespeichert. Aus Sicherheitgründen wird bei EEPROM Lesefehlern oder zu kleinen Werten für die Totzeit ein Minimum statt des ungültigen Wertes angewendet. Eine zu grosse Totzeit ist nicht schädlich, begrenzt aber den nutzbaren PWM Bereich und die Effektivität des Frequenzumrichters. Es schadet aber natürlich nicht, bei der Erstinbetriebnahme von einer sehr hohen Totzeit auszugehen und diese dann unter Beobachtung der Stromaufnahme zu verringern. Die Einheit der Totzeit (DEAD_TIME_HALF) bei der Anzeige im Display sind MC-Zyklen. Die resultierende Totzeit über die Halbbrücke ist doppelt so hoch wie die eingestellten MC-Zyklen. Bei der gezeigten Schaltung ist ein Wert von 20 verwendet worden, resultierend in einer Totzeit von (2 * 20 * 62,5ns = 2500ns). Bei schnellen MOSFets oder IGBTs kann dieser Wert möglicherweise noch verringert werden. Die Stromaufnahme kann am Shunt R7 gemessen werden. **Achtung: Dieser Teil ist nicht netzgetrennt und sollte deswegen mit Vorsicht behandelt werden. Eine Verbindung zwischen GNDI und PE führt unweigerlich zur Zerstörung von Bauteilen im Netzeingang.**

Die Wellenform der Phasen ist bei der gezeigten Tabelle dem Betrieb von Motoren angepasst. Andere Wellenformen lassen sich durch Austausch der Tabelle erreichen. Seit dem Update wird nun die Option '#define PURE_SINE_DRIVE' in vfd.h angeboten. Es werden in diesem Fall 3 echte Sinuskurven mit 120° Phasenverschiebung erzeugt, ohne die typischen 'Popokurven' für BLDC ähnliche Ansteuerung. Beim Austausch gegen andere Tabellen sollte beachtet werden, das die Länge der Tabelle mit der Originaltabelle übereinstimmt, da die Berechnung der Schrittweite davon abhängt. Wenn der MC mit einer anderen Frequenz als 16Mhz betrieben wird, sollte auch der Parameter 'DIVISIONEER' in der Datei 'vfd.h' angepasst werden, damit die Darstellung am Display korrekt ist. Irgendwie kriege ich es nicht hin, das der Wert direkt aus F_CPU errechnet wird, wäre schön, wenn mir das mal jemand beibringt. Wer lediglich einen Wechselstrom Ausgang benötigt, sollte die Tabelle durch echte Sinuswerte ersetzen (wie jetzt im Update vorgesehen) und nur 2 Endstufen aufbauen. Zwischen den Ausgängen dieser Endstufen ist dann ein regelbares Wechselspannungspotential.

Natürlich sollte die Drehrichtung eines mit voller Geschwindigkeit laufenden Motors nicht einfach umgekehrt werden. In jedem Fall ist es ratsam, den Motor zu stoppen und erst dann die Drehrichtung zu ändern.

V/f

Die V/f Charakteristik ist die meist angewandte Methode bei Frequenzumrichtern, um die aufgenommene Leistung eines Motors mit seiner Drehzahl zu regeln. 'Volt per Frequenz' Charakteristik heisst, das ein Motor bei z.B. 100% seiner Drehzahl zwar schon mit der vollen Nennspannung betrieben werden darf, das aber bei reduzierter Drehzahl auch die Betriebsspannung heruntergeregelt werden muss. Bei z.B. 50% der Nenndrehzahl sollten auch nicht mehr als 50% der Betriebsspannung angelegt werden usw. Die genauen Werte sind unterschiedlich und sollten deswegen entweder dem Datenblatt des Motors entnommen werden

oder durch Ausprobieren ermittelt werden.

Ein Hinweis liefert dabei das erzielte Drehmoment (die "Kraft") des Motors. Selbstverständlich sollte es dabei sein, dem Motor nicht mehr abzuverlangen, als er liefern kann. Lautes Brummen ist z.B. ein Zeichen für zu viel Spannung, das V/f Verhältnis sollte kleiner gemacht werden. Oft ist es aber ausreichend, der Maschine bei Nennfrequenz (z.B. 50 Hz) gerade 100% Betriebsspannung zu liefern, bei einer reduzierten Frequenz reduziert der MC dann auch proportional die mittlere Spannung an den Motorwicklungen.

Übrigens erlaubt diese Steuerung auch, den Umrichter als 'regelbares Netzteil' zu missbrauchen. Ich habe z.B. mit einem schnellen Gleichrichter auch schon einen DC-Motor aus einer Waschmaschine mit einer Spannung von 24 Volt (Waschen) bis 200 Volt (Schleudern) betrieben.

Schaltbild

Das Schaltbild (siehe unten) bietet keine grossen Besonderheiten. Was hier angeberisch als 'DC/DC Wandler' beschrieben wird, ist im Fall des 5 Volt Wandlers ein Schaltnetzteil vom Typ Handylader und im Fall des 18 Volt Wandlers ein kleines Schaltnetzteil für LED Lichterketten, das durch geringfügiges ändern des Regelkreises von 12 auf 18 Volt hochgedreht wurde. Der benötigte Strom ist abhängig von der gewählten Treiberschaltung und den Endstufen und liegt bei der gezeichneten Variante so um die 150mA für alle 3 Endstufen. Wichtig ist wie immer eine gute Entkopplung möglichst direkt an den Treiberbausteinen. Die impulsartige Belastung durch den Ladevorgang der Gates sollte durch Elkos aufgefangen werden, wie im Schaltbild gezeigt. Der 5 Volt Zweig wird ohne Hintergrundbeleuchtung des LCD mit etwa 40-80mA belastet, wobei der Löwenanteil für die LEDs der Optokoppler draufgeht. Übrigens sind auch Kleinnetzteile mit Einweggleichrichtung am Markt zu finden, die auf keinen Fall aus der Zwischenkreisspannung gespeist werden dürfen. Solche Netzteile sind auf der Wechselstromseite des Netzeingangs anzuschliessen.

18 Volt werden übrigens nur deswegen verwendet, weil die gerade vorhandenen Optokoppler vom Typ 3120 mindestens 16 Volt Betriebsspannung benötigen. Hätte ich welche vom Typ HCPL3180 gehabt, hätten 12 Volt ausgereicht und das Netzteil hätte nicht manipuliert werden müssen. Leider sind die Optokoppler nicht ganz billig, theoretisch sollten es aber auch preiswertere schnelle Typen tun, wie z.B. der 6N137 oder HCPL2630.

Letzterer bietet 2 Koppler in einem Gehäuse. Dabei ist allerdings die Signaltreue zu beachten. Keinesfalls darf der Optokoppler die Phase invertieren! Dieser Fehler würde zum 'Shoot-Through' der Endstufen führen. Falls invertierende Koppler benutzt werden, müssen also die Polaritäten der OC Ausgänge in der Software umgedreht werden und mit entsprechenden Pulldowns und -Ups dafür gesorgt werden, das keine falschen Signale an den IR Treibern erscheinen. Die gezeigte Endstufe (**gezeigt wird nur eine Phase, die anderen beiden sind identisch**) besitzt mit dem Shunt R7 und den Transistoren T1 und T2 eine integrierte Überstromsicherung.

Bei einem zu hohen Strom (hier ca. 2 Ampere) leitet T1 und steuert T2 durch. Das Highsignal am Kollektor von T2 aktiviert die Shutdown Eingänge der Treiber ICs. L4 und L3 sollten hochstromfähig sein, da hier der Motorstrom herüberfliesst. Die kleinen Schaltnetzteile (aka "DC/DC Wandler") werden direkt aus der Zwischenkreisspannung versorgt. Diese liegt ja typisch bei ca. 325 Volt und ist damit genau richtig. Man könnte die internen Gleichrichter und Siebelkos entfernen, aus Faulheit ist das bei mir nicht geschehen.

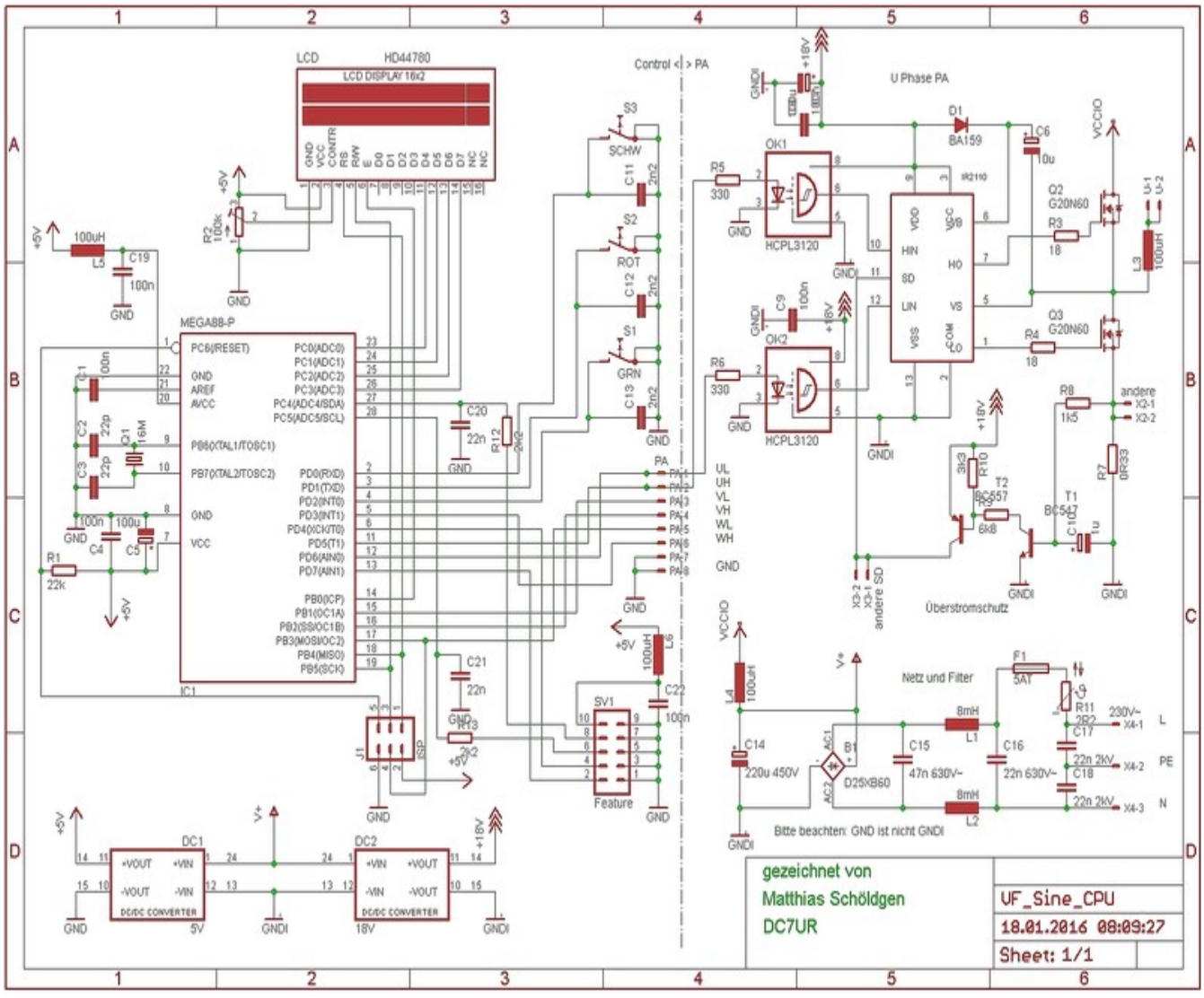
Auffällig ist das Netzeingangsfilter. Frequenzumrichter sind dafür bekannt, das sie harte Pulse mit hoher Frequenz erzeugen und um zu verhindern, das diese auf das Netz zurückwirken, wurde eine recht aufwendige Schaltung vorgesehen. Es ist auch ratsam, bei längeren Leitungen vom Ausgang der Endstufe zum Verbraucher diese Leitung entweder zu schirmen, oder über ein Filter direkt am Ausgang des FU zu führen. Die PWM Frequenz liegt bei ca. 31 kHz und hat steile Flanken, deswegen sollte der Abstrahlung von Störfeldern bzw. die Verhinderung derselben ein wenig Aufmerksamkeit geschenkt werden. C15 bis C18 sollten übrigens X2 Typen sein, wobei C17/C18 auch ein Y Typ sein kann. Bestens geeignet sind designierte Entstörkondensatoren, solange sie die nötige Spannungsfestigkeit haben. Die Drosseln können entweder stromkompensiert sein oder einzeln. Wie auf dem Bild oben zu sehen ist, habe ich ein fertiges Netzfilter mit Kaltgeräteeingang benutzt, das erspart den diskreten Aufbau eines Filters. R7 (der Shunt) muss natürlich den Motorstrom vertragen, ich verwand einen 5 Watt Betonwiderstand. Der hohe Einschaltstrom bei leerem Zwischenkreiskondensator wird durch R11, einen Hochstrom NTC, begrenzt, wie er auch in PC Netzteilen üblich ist. Die Bauform 644 von z.B. Philips ist hier gut geeignet.

Die Kondensatoren parallel zu den Tastern sind übrigens keineswegs zum Entprellen, obwohl sie diese

Funktion nebenbei auch erfüllen. Sie verhindern die Einstrahlung von Störsignalen durch naheliegende LCD- und PWM- Leitungen und sorgen für einen Mindeststrom in den Tastern. (Siehe auch: http://www.mikrocontroller.net/topic/287303). Die Taster werden zyklisch im 'Kommandointerpreter' der Software abgefragt und benötigen deswegen keine gesonderte Entprellung.

Neben dem ISP Anschluss ist ein als 'Feature' Anschluss bezeichneter Verbinder zu sehen. An diesen können die beiden Analogsignale (Pin 6 und 8) angelegt werden. Pin 6 (ADC Kanal 5) steuert das V/f Verhältnis während an Pin 8 (ADC-Kanal 4) das Frequenzregelsignal anliegt. Pin 4 dient zur Änderung der Drehrichtung. Eine nach Masse gesetzte Brücke wendet den Motor. Pin 2 schaltet zwischen externer (ADC) und interner Kontrolle (Taster und LCD) um. Bei gesetzter Brücke nach Masse wird auf externe Kontrolle umgeschaltet. Diese Prüfung wird nur nach einem Resetzyklus des MC durchgeführt. Änderungen im Betrieb sind also nicht vorgesehen.

Wichtig ist die Unterscheidung zwischen GND und GNDI: GND ist der netzgetrennte Bezugspunkt für die Steuerelektronik des MC, während GNDI direkt mit dem Netz verbunden ist. **Auf keinen Fall dürfen GND und GNDI miteinander verbunden werden!** Im Gegenteil, ausreichende Sicherheitsabstände sind auf jeden Fall einzuhalten. In meinem Prototyp sind deswegen Kontroll- und Leistungsplatine völlig voneinander getrennt und nur durch die Optokopplerplatine verbunden.



Fazit

Der gezeigte Frequenzumrichter ist durch die Einfachheit des Steuerungsteils und die Verwendung von Standardbauteilen ein dankbares Projekt, um die Möglichkeiten moderner Mikrocontroller zu demonstrieren und mit Motorsteuerungen zu experimentieren. Je nach verwendeter Endstufe können auch andere Motoren, z.B. E-Autos, Festplattenmotoren oder Spiegelmotoren aus Laserdruckern angesteuert werden. Ändern der PWM Frequenz z.B. ist in der Routine TimersInit() vorgesehen. Andere Vorteiler Einstellungen für alle 3 Timer

können hier mit Hilfe der im Moment auf 0 stehenden 'CSx1' eingefügt werden. Natürlich kann der MC auch mit dem internen 8 MHz Oszillator betrieben werden. Man erhält dadurch zwei weitere Portpins für z.B. Überstromrückmeldung oder Drehzahlüberwachung für den Preis einer niedrigeren PWM Frequenz.

Die externe Kontrolle über Analogeingänge ermöglicht z.B. auch, Servomotoren wie Capstan anzusteuern und mittels z.B. PLL zu stabilisieren. Der integrierte PID Regler mit einstellbaren Parametern erlaubt die Anpassung an verschiedenste Problemstellungen. Ich bin jedenfalls gespannt, was dem Leser an Anwendungen einfällt und freue mich über Anregungen und Ideen. Noch ein Hinweis: Es lohnt sich, die Preise der verschiedenen möglichen Controller zu vergleichen. Oft ist mittlerweile der 'grosse' ATMega328 billiger erhältlich als seine kleinen Brüder. Sollte ein anderer Controller verwendet werden (Mega88, 168 und 328 sind ja alle untereinander kompatibel), sollte das in den Projekteinstellungen berücksichtigt werden.

Downloads

Der Download enthält neben dem Source Code für AVR Studio 4 mit WinAVR auch ein Schaltbild und die Schaltung im Eagle Format (*.sch). Da die Anwendungen sich erheblich unterscheiden können, wurde auf die Erstellung eines Platinenlayouts verzichtet, ist aber mit dem Schaltplan für Eagle duchaus möglich. Die gewählten Bauformen für Teile sind nicht unbedingt die optimalsten, etwas Handarbeit und Anpassung könnten also nötig werden. AVR Studio 4 wurde wegen seiner Stabilität benutzt, der Code sollte mit evtl. geringen Anpassungen aber natürlich auch unter AS5, AS6 oder AS7 kompilieren. Wer auf jegliche Codeveränderungen verzichten möchte, findet fertige HEX Dateien im Sourcecode Ordner in Unterordner 'default', welche für Mega88, 168 und 328 vorkompiliert wurden, jeweils für Motorkurven- oder Sinuskurven Erzeugung.

- Datei:VF Sine Drive.zip Sourcecode

Siehe auch

- Diskussion zum Artikel
- Flash Animation zu Drehstrom bei der DWU - Zentrale für Unterrichtsmedien e.V.
- Atmel Application Note AVR447 - BLDC Motor Driver with Hallsensors and Sine Modulation
- Atmel Application Note AVR221 - A universal PID Controller with AVR
- Peter Fleurys Homepage mit der verwendeten LCD Bibliothek
- Infineon - Datenblätter und Applikationen für Treiber, MOSFet und IGBT

Kategorien: Wettbewerb | AVR-Projekte | Spannungsversorgung und Energiequellen