ftPwrDrive

ein fischertechnik-kompatibler I²C Stepper und Servo-Controller

Bedienungsanleitung

Christian Bergschneider
Stefan Fuss
Björn Gundermann

17.09.2019

Projekt-Homepage: https://github.com/elektrofuzzis/ftPwrDrive

Kontakt: information@gundermann.org

Forum: https://forum.ftcommunity.de/

Inhalt

Einleitung

Der ftPwrDrive ist ein Schrittmotor- und Servocontroller, der für den Betrieb mit den fischertechnik Controllen TX und TXT sowie dem ftDuino designed wurde. Durch die Ansteuerung über den I²C-Bus kann er genauso einfach in jedem Arduino[1]- oder Raspberry-Projekt eingesetzt werden.

Der Controller selbst basiert auf einem Arduino Leonardo und hat die notwendige Elektronik zur Ansteuerung von Schrittmotoren, Servos und Endlagenschaltern integriert. Über den I²C-Bus erhält der ftPwrDrive von einem Mastercontroller wie dem TXT Befehle zu Ansteuerung seiner Hardware. So schickt z.B. der Mastercontroller an den ftPwrDrive das Kommando den Motor M1 um 10.000 Schritte zu verfahren. Der Mastercontroller kann nach diesem Kommando sein Programm fortsetzen, die Firmware des ftPwrDrive führt das Kommando aus und bewegt den Motor autark.

Auch wenn Sie jetzt so schnell wie möglich Ihren ersten Motor anschließen möchten, nehmen Sie sich bitte etwas Zeit um zunächst die Arbeitsweise von Schrittmotoren und Servos sowie die Anschlüsse des ftPwrDrive kennenzulernen.

[1]) Die Schnittstellenimplementierung für ftDuino kann auf jedem Arduino-Clon eingesetzt werden.

Schrittmotoren

Bei den normalen Gleichstrommotoren von fischertechnik wird die Drehzahl über die am Motor anliegende Spannung geregelt. Eine höhere Spannung ergibt eine höhere Drehzahl und eine niedrigere Spannung ergibt eine geringere Drehzahl. Um festzustellen, wie viele Umdrehungen sich ein Motor bewegt hat, muss die Bewegung mit externen Encodern gemessen werden.

Schrittmotoren benötigen eine aufwendigere Ansteuerung. Die einzelnen Spulen des Motors werden nacheinander ein- und ausgeschaltet. Der Motor bewegt sich so schrittweise von einer zur nächsten Position. Eine Messung der realen Bewegung ist nicht notwendig, die Anzahl der ausgelösten Schritte bestimmt wie viele Umdrehungen der Motor durchgeführt hat. Der Winkel, um die sich ein Schrittmotor bei einem Schritt bewegt, ist sehr klein. Die zum ftPwrDrive passenden NEMA-14-Motoren bewegen sich je Schritt um 1.8° (200 Schritte pro Umdrehung).

Um den Motor effizient anzusteuern, wird nicht nur der Strom einer Spule ein- und ausgeschaltet. Vielmehr wird der Strom der durch die Spule fließt durch die Endstufe geregelt. Mit der Stromregelung und dem gleichzeitigen Ansteuern von mehreren Spulen lassen sich Positionen zwischen zwei Vollschritten realisieren. In diesem Fall spricht man von Microstepping. Der ftPwrDrive kann neben dem Vollschrittbetrieb, halbe, viertel, achtel und sechzehntel Schritte. Im Sechzehntelschrittbetrieb bewegt sich der Motor je Schritt nur noch um 0.1125°.

Durch die Regelung des Spulenstroms ist es wichtig, diesen auf den maximalen Strom der Motoren einzustellen. Der Maximalstrom kann beim ftPwrDrive über ein Potentiometer eingestellt werden. Im Auslieferzustand ist dieser auf 0.7A eingestellt.

Zieht man im laufenden Betrieb die Stecker für die Schrittmotoren ab, so wird es durch die Stromregelung zu Funken an den Steckern kommen. Um den ftPwrDrive nicht zu beschädigen, dürfen die Motoren immer nur bei ausgeschalteter Endstufe angeschlossen werden.

Der ftPwrDrive nutzt A4988 Stepper Treiber für die Ansteuerung der Schrittmotoren. Dies weit verbreitete Endstufe ist weitgehend kurzschlusssicher und kann in der Auslegung beim ftPwrDrive Motoren mit bis zu 1A Spulenstrom ansteuern.

Der Controller kann die Endstufen der Schrittmotoren einzeln ein- und ausschalten. Im geschalteten Zustand fließt auch ohne Verfahrbewegung ein Spulenstrom und blockiert somit den Motor. Mit dieser Funktion kann z.B. ein Roboterarm in Position gehalten werden. Der anliegende Spulenstrom führt zu einer deutlichen Erwärmung des Motors und ggf. zu hochfrequentem „Pfeifen“. Wenn nicht explizit notwendig, sollte nach einer Motorbewegung der Spulenstrom ausgeschaltet werden.

Servos

Servomotoren bestehen aus kleinen Elektromotoren mit einer integrierten Positionsregelung. Der Servo erhält über ein PWM-Steuersignal des ftPwrDrive eine Winkelvorgabe, die Positionsregelung fährt diesen Winkel an und hält anschließend die Winkelposition ein.

Die für fischertechnik typischen Microservos können mit maximal 6V betrieben werden. Im Ruhezustand und ohne mechanische Belastung benötigen die Servos nur ca. 150mA. Das Verstellen der Servos oder eine mechanische Belastung führen zu deutlich höherer Leistungsaufnahme.

Der ftPwrDrive besitzt eine interne 5V-Spannungsversorgung mit einer Maximalleistung von 800mA für die Servomotoren. Damit lassen sich in der Regel bis zu zwei Servos ansteuern. Alternativ können die Servomotoren über ein USB-Steckernetzteil mit max. 5V/2.5A versorgt werden. Somit steht genug Leistung für den Betrieb von bis zu vier Servos zur Verfügung.

Endlagenschalter & Notaustaster

Der Controller verfügt über 5 Tastereingänge, an die Schließer angeschlossen werden können. An diesen Eingängen können ausschließlich Taster angeschlossen werden, Reedkontakte und Phototransistoren werden nicht unterstützt.

Zu jedem Motor gibt es einen Endlagenschaltereingang. Wird dieser ausgelöst, so erkennt die Firmware das Überfahren einer Endposition und stoppt die Verfahrbewegung. Über eine Parallelschaltung können zwei Taster die beiden Endpositionen einer Achse dem Controller signalisieren. Der Endlagenschalter kann von der Firmware auch zur Anfahrt einer Referenzposition genutzt werden.

Der mittlere Tastereingang ist ein Notauseingang. Wird dieser ausgelöst, so stoppt die Firmware alle Verfahrbewegungen und schaltet die Motorendstufen ab.

I²C Bus

Der ftPwrDrive wird über den I²C-Bus an den Mastercontroller angeschlossen. Über den I²C-Bus erhält der ftPwrDrive vom Master seine Verfahrbefehle, der Master kann darüber auch den Zustand des Controllers abfragen.

Da am I²C-Bus mehrere Geräte angeschlossen werden können, hat jedes Gerät am Bus eine eigene Adresse. Werkseitig ist der ftPwrDrive auf die Adresse 32 bzw. 0x20 eingestellt. Ist diese Adresse durch ein anderes Gerät belegt, oder sollen mehrere ftPwrDrive angeschlossen werden, so kann die I²C-Adresse im Maintenance Modus geändert werden.

Die I²C-Busadresse lässt sich auch durch I²C-Scanner ermitteln. Dabei testet der Mastercontroller alle gültigen I²C-Adressen ab, ob sich dort ein Gerät meldet. Im Softwarepaket des ftDuinos wird ein I²C-Scanner in den Beispielen mitgeliefert:

```
Datei ▷ Beispiele ▷ FtduinoSimple ▷ I2C ▷ I2cScanner
```

Da es beim I²C-Bus verschiedene Logikspannungen gibt, muss man beim Anschluss des ftPwrDrive auf die richtige Logikspannung achten. Der ftPwrDrive benutzt 5V als Logikspannung. Der fischertechnik TX-Controller und der ftDuino arbeiten ebenfalls mit 5V. Der fischertechnik TXT-Controller arbeitet mit 3.3V. Die meisten Arduino-Clone arbeiten ebenfalls mit 5V, einige neuere Modelle benutzen wie der TXT 3.3 V.

Innerhalb der fischertechnik-Controller kann man bei den Logikspannungen keinen Fehler machen. Alle 5V-Geräte haben einen 6-poligen Stecker, diesen verbindet man über ein 1:1-Kabel mit dem ftPwrDrive.

Zum Anschluss des TXT-Controllers muss die Logikspannung mit einem Levelshifter wie dem ftExtender angepasst werden. Der TXT-Controller hat einen 10-poligen Stecker, der wiederum mit einen 1:1-Kabel an den ftExtender angeschlossen wird. Der ftPwrDrive wird mit einem 1:1-Kabel an den 6-poligen Stecker des ftExtenders angeschlossen. Zusätzlich benötigt der ftExtender über ein normales ft-Kabel eine 9V-Stromversorgung.

Wer mag, kann auch handelsübliche Levelshifter selbst mit Kabeln für den ftPwrDrive konfektionieren.

Stromversorgung

Schrittmotoren benötigen in der Regel größere Netzteile als Gleichstrommotoren. Da der ftPwrDrive bis zu vier Schrittmotoren mit bis zu 1A Spulenstrom ansteuern kann, wird ein leistungsfähiges Netzteil benötigt.

Am ftPwrDrive wird ein Netzteil mit 12V / 5A und einen 5.5mm Hohlstecker benötigt. Über dieses Netzteil wird die interne Stromversorgung für die Servos, die Schrittmotoren und der integrierte Arduino Leonardo versorgt.

Die Stromversorgung des ftPwrDrive über die USB-Buchse ist nicht möglich.

Anschlüsse

M1..M4: Hier werden die Schrittmotoren angeschlossen. Die Farbmarkierungen im Bild beziehen sich auf die Kabelfarben bei 14HS13-0804S Motoren.

Bei anderen Farbcodierungen wird Spule A des Motors an die beiden oberen Anschlüsse (schwarz und grün) angeschlossen. Spule 2 wird dementsprechend an die beiden unteren Anschlüsse (rot und blau) angeschlossen.

Motoren immer nur bei ausgeschalteten Endstufen anstecken oder abziehen! Sonst „funkt“ es und kann zu Beschädigungen des ftPwrDrive führen.

Verwenden Sie bitte die roten und grünen ft-Stecker zum Konfektionieren der Kabel. Die Gehäuse der Märklin-Stecker sind sehr dick, je nach Serie besteht die Möglichkeit über die Schrauben zweier Stecker eine Spule kurzzuschließen.

E1..E4: Endlagenschalter zu den Motoren. Ist der Schalter geschlossen, ist die Endlage erreicht und der Controller schaltet die Verfahrbewegung aus. Die Endlagenschalter können auch für Referenzpositionen genutzt werden.

N: Notaus. Ist der Schalter geschlossen werden alle laufenden Verfahrbewegungen der Motoren beendet und die Endstufen abgeschaltet.

P1: Über dieses Poti kann man im Maintenance Modus den maximalen Spulenstrom der Schrittmotoren einstellen. Dazu muss der Deckel

abgeschraubt werden. Die Geräte werden mit einer Stromstärke von 0,7A ausgeliefert.

USB: Im Normalbetrieb können hier die Servos von einem USB-Netzteil mit bis 2.5A Leistung versorgt werden.

Im Maintenance-Mode kann hier die I2C-Adresse und der maximale Spulenstrom für die Motoren eingestellt werden.

Außerdem können hierüber FW Updates eingespielt werden.

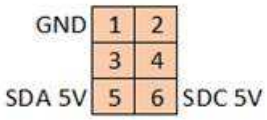
12V: 12V-Spannungsversorgung für den ftPwrDrive. Je nach eingesetzten Motoren und Servos sollte das Netzteil für bis zu 5A ausgelegt sein.

Betrieb: Rote LED, die konstant leuchtet, wenn die Spannungsversorgung angeschlossen ist.

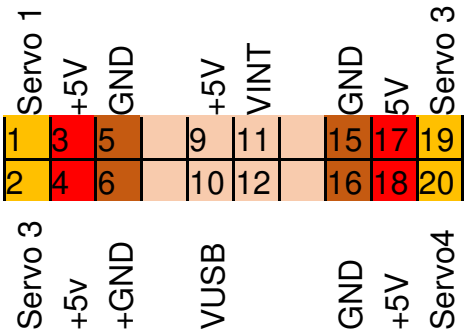
Status: Rote LED, die im Normalbetrieb einfach kontinuierlich leuchtet. Ist der Maintenance-Modus aktiviert oder stimmt die Stromeinstellung des Poti P1 nicht mehr, so blinkt die LED.

I2C: Standard TX/ftDuino 6poliger Stecker, 5V Logikpegel. Der 6-polige I2C-Busstecker ist kompatibel zum EXT-Anschluss des TX und ftDuinos, so dass hier die beiden Geräte direkt angeschlossen werden können. Für den Anschluss reicht ein 1:1 belegtes 6- poliges Kabel.

Wegen den unterschiedlichen Logikspannungen muss beim TXT ein Levelshifter wie der ft-Extender zwischen TXT und ftPwrDrive geschaltet werden.



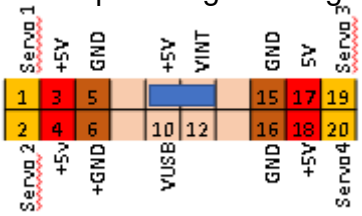
Servo: Hier können bis zu 4 Servos angeschlossen werden. Allgemeine Steckerbelegung, die Kabelfarben entsprechen den ft-Servos:



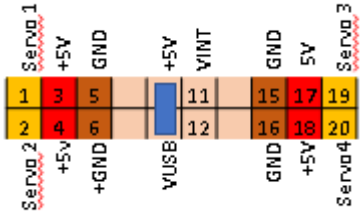
Die +5V Versorgungsspannung können entweder aus dem internen Spannungsregler mit max. 800mA kommen oder von einem USB-Netzteil

an der USB-Buchse. Dazu wird ein Jumper auf den Pins 9/11 bzw. 9/10 gesteckt:

Interne Spannungsversorgung:



USB Netzteil:



ftDuino-Interface

Für den ftDuino wird eine Bibliothek für die Arduino IDE geliefert. Diese kann in github heruntergeladen werden.

Das ZIP-File mit der Library kann dann in der Arduino-IDE unter „Sketch/IncludeLibrary/Add .ZIP Library“ (Sketch/Bibliothek einbinden/Zip Bibliothek hinzufügen) eingelesen werden.

Unter „File/Examples/Examples From Custom Libraries/ftPwrDrive“ (Datei/Beispiele/Beispiele aus eigenen Bibliotheken/ftPwrDrive Library) befinden sich dann Beispielprogramme.

Aktuell sind diese

01_simple_motor

02_microstepping

03_servo

04_everything

Inbetriebnahme der Hardware

Wir empfehlen zum Einstieg zunächst die Inbetriebnahme über die Beispielprogramme. So wird auf einfache Art die Korrektheit des Aufbaus und der notwendigen Tools sichergestellt.

Das ftPwrDrive wird ohne jeglichen Anschluss erst einmal auf den Tisch gelegt.

Führen Sie nun die folgenden Schritte durch:

- Schließen Sie 1-4 Schrittmotoren an. Achten Sie dabei auf die Kabelfarben.
- Verbinden Sie den I²C-Stecker des ftPwrDrive über ein 6poliges 1:1-Kabel mit dem ftDuino.
- Schließen Sie ein Netzteil mit 12V/5A als Stromversorgung an.
- Schalten Sie ftDuino und ftPwrDrive an. Betriebs-LED und Status-LED des ftPwrDrive sollten kontinuierlich eingeschaltet sein. Die Schrittmotoren „zucken“ kurz beim Start des ftPwrDrive. Die Motorendstufen sind abgeschaltet, so dass sich die Achsen der Schrittmotoren bewegen lassen.
- Installieren Sie – sofern noch nicht erfolgt – die ftDuino-Library.

Beispielprogramm 01_simple_motor

Laden Sie in der Arduino IDE das Beispielprogramm 01_simple_motor. Wählen Sie als Board „ftDuino“ aus und stellen unter Port die serielle Schnittstelle des ftDuino aus.

Compilieren Sie das Programm und laden es auf den ftDuino. Das Programm wartet zunächst darauf, dass Sie in der Arduino IDE den seriellen Monitor starten. Anschließend wird auf der seriellen Console Informationen über die Verfahrbewegungen ausgegeben und die Motoren starten.

Zunächst werden die Motoren nacheinander jeweils um 1000 Schritte bewegt. Danach werden alle Motoren gleichzeitig angesprochen. Das Programm beendet sich, wenn alle Motoren zum Stillstand gekommen sind.

Schließen Sie nun einen Endlagentaster für einen beliebigen Motor an. Starten Sie das Programm erneut und drücken den Endlagentaster. Der Motor stoppt augenblicklich.

Zunächst muss das Interface initialisiert werden:

```
// create an instance of ftPwrDrive and set the I2C address
ftPwrDrive Drive = ftPwrDrive(32);
```

Um nun einen Motor zu starten, müssen Sie ihm zunächst eine Zielposition vorgeben und anschließend den Motor starten:

```
// never use Drive.setRelDistance( i, 1000 ); because i is not equal to FTPWRDRIVE_M[i]
Drive.setRelDistance( FTPWRDRIVE_M[i], 1000 );
Drive.startMoving( FTPWRDRIVE_M[i] );
```

Bitte beachten Sie, dass die Motoren im Interface nicht über 0..3 bzw. 1..4 angesprochen werden. Nutzen Sie deshalb die Konstanten `FTPWRDRIVE_M1` bis `FTPWRDRIVE_M4` oder die Enumeration der Motoren `FTPWRDRIVE_M[]`.

Da der Mastercontroller nach diesen Kommandos direkt weiterarbeitet, müssen Sie den Status der Bewegung pollen:

```
while (Drive.isMoving( FTPWRDRIVE_M[i] )) {
      print_StepsToGo();
      delay(100);
}
```

Beispielprogramm 02_micostepping

Laden Sie das Programm auf den ftDuino uns starten es. Das Programm wartet wiederum, bis eine serielle Console zur Visualisierung eingeschaltet ist. Es wählt verschiedene Microstepmodi aus und lässt dann jeweils M1 im Full-, Half-, Quad-, Eigth- und Sixteenstepmodus laufen.

Beispielprogramm 03_servo

Wenn Sie das ftPwrDrive auch für Servo Motoren nutzen wollen, dann laden Sie das Beispiel 03_servo

Vor dem anschließen der Servos bitte die Stromversorgung vom ftPwrDrive trennen.

Verkabeln Sie die Servo Motoren bitte wie im Kapitel Anschlüsse beschrieben. Bitte nicht vergessen den Jumper für die Spannungsversorgung entsprechend zu setzen.

Danach den ftPwrDrive Controller wieder bestromen, das Beispiel in den ftDuino hochladen und ausführen lassen. Auch hier werden wieder Statusausgaben über die Monitor Ausgabe vorgenommen.

Kommandoumfang der Firmware

```
// Microstep modes

static const uint8_t FTPWRDRIVE_FULLSTEP = 0, FTPWRDRIVE_HALFSTEP = 4, FTPWRDRIVE_QUARTERSTEP
= 2, FTPWRDRIVE_EIGTHSTEP = 6, FTPWRDRIVE_SIXTEENTHSTEP = 7;

// motor numbers

static const uint8_t FTPWRDRIVE_M1 = 1, FTPWRDRIVE_M2 = 2, FTPWRDRIVE_M3 = 4, FTPWRDRIVE_M4 =
8;

// number of motors

static const uint8_t FTPWRDRIVE_MOTORS = 4;

// enumeration of motor numbers

static const uint8_t FTPWRDRIVE_M[ FTPWRDRIVE_MOTORS ] = { FTPWRDRIVE_M1, FTPWRDRIVE_M2,
FTPWRDRIVE_M3, FTPWRDRIVE_M4 }

// flags, i.e. used in getState

static const uint8_t FTPWRDRIVE_ISMOVING = 1, FTPWRDRIVE_ENDSTOP = 2, FTPWRDRIVE_EMERCENCYSTOP
= 4, FTPWRDRIVE_HOMING = 8;

class ftPwrDrive {

  public:

    ftPwrDrive( uint8_t myI2CAddress );

      // constructor

    void Watchdog( long w );

      // set wartchog timer

    void setMicrostepMode( uint8_t mode );

      // set microstep mode - FILLSTEP, HALFSTEP, QUARTERSTEP, EIGTHSTEP, SIXTEENTHSTEP

    uint8_t getMicrostepMode( void );

      // get microstep mode - FILLSTEP, HALFSTEP, QUARTERSTEP, EIGTHSTEP, SIXTEENTHSTEP

    void setRelDistance( uint8_t motor, long distance );

      // set a distance to go, relative to actual position

    void setAbsDistance( uint8_t motor, long distance );

      // set a absolute distance to go

    long getStepsToGo( uint8_t motor );

      // number of needed steps to go to distance

    void setMaxSpeed( uint8_t motor, long speed );

      // set a max speed
```

```
    long getMaxSpeed( uint8_t motor);
      // get max speed

    void startMoving( uint8_t motor, boolean disableOnStop = true );
      // start motor moving, disableOnStop disables the motor driver at the end of the
movement

    void startMovingAll( uint8_t maskMotor, uint8_t maskDisableOnStop =
FTPWRDRIVE_M1|FTPWRDRIVE_M2|FTPWRDRIVE_M3|FTPWRDRIVE_M4 );
      // same as StartMoving, but using uint8_t masks

    boolean isMoving( uint8_t motor );
      // check, if a motor is moving

    uint8_t isMovingAll( );
      // return value is uint8_tmask, flag 1 is motor#1, flag2 is motor#2, ...

    uint8_t getState( uint8_t motor );
      // 8754321  - flag 1 motor is running, flag 2 endstop, flag 3 EMS, flag 4 position
overrun, flag 5 homing

    boolean endStopActive( uint8_t motor );
      // check, if end stop is pressed

    boolean emergencyStopActive( void );
      // check, if emergeny stop is pressed

    void setPosition( uint8_t motor, long position );
      // set position

    void setPositionAll( long p1, long p2, long p3, long p4 );
      // set position of all motors

    long getPosition( uint8_t motor );
      // get position

    void getPositionAll( long &p1, long &p2, long &p3, long &p4 );
      // get position of all motors

    // don't use acceleration yet - just stub implementation

    void setAcceleration( uint8_t motor, long acceleration );
      // set acceleration

    void setAccelerationAll( long a1, long a2, long a3, long a4 );
      // set accelerationof all motors

    long getAcceleration( uint8_t motor );
      // get acceleration
```

```
    void getAccelerationAll( long &a1, long &a2, long &a3, long &a4 );
      // get acceleration of all motors

    void setServo( uint8_t servo, long position );
      // set servo position

    long getServo( uint8_t servo );
      // get servo position

    void setServoAll( long p1, long p2, long p3, long p4 );
      // set all servos positions

    void getServoAll( long &p1, long &p2, long &p3, long &p4 );
      // get all servo positions

    void setServoOffset( uint8_t servo, long offset );
      // set servo offset

    long getServoOffset( uint8_t servo );
      // get servo offset

    void setServoOffsetAll( long o1, long o2, long o3, long o4 );
      // set servo offset all

    void getServoOffsetAll( long &o1, long &o2, long &o3, long &o4 );
      // get all servo offset

    void setServoOnOff( uint8_t servo, boolean on );
      // set servo pin On or Off without PWM

    void homing( uint8_t motor, long maxDistance, boolean disableOnStop = true );
      // homing of motor using end stop

  private:
    uint8_t i2cAddress = 32;
};
```

Maintenance Modus

Im Maintenance Modus können Sie sowohl die I2C Adresse des ftPwrDrive Controllers als auch die maximale Stromstärke anpassen. Der Maintenance Modus wird automatisch gestartet, wenn die gewünschte Stromstärke nicht zur aktuell ausgewählten passt.

Manueller Start des Maintenance Modus

Verbinden Sie den ftPwrDrive Controller direkt mit dem PC und wählen Sie dann als Board Arduino Leonardo aus. Die Port Adresse ist dann der bei ihnen erkannte COM Port.

Öffnen Sie nur einen seriellen Monitor mit 9600 Baud. Tippen Sie ein Zeichen ein. Es sollte die folgende Ausgabe zu sehen sein und die LED neben der USB-Buchse blinkt.

```
 __ _   _____                _____       _
  / _| | |  __ \\              |  __ \\     (_)
 | |_| |_| |__) |_      ___ __| |  | |_ __ ___   _____
 |  _| __|  ___/\\ \\ /\\ / / '__| |  | | '__| \\ \\ / / _ \\
 | | | |_| |     \\ V  V /| |  | |__| | |  | |\\ V /  __/
 |_|  \\__|_|      \\_/\\_/ |_|  |_____/|_|  |_| \\_/ \\___|


(C) 2019 Christian Bergschneider & Stefan Fuss
Version x.yy
Serial number:      xx
I2C-Address:         yy
max. motor current:  x.yyA /  a.bbA
```

Je nach Stellung des Potentiometers erfolgt jetzt einen Fehlerhinweis zur Motoren Stromstärke oder nicht.

I2C Adresse

Als erstes werden Sie nach der I2C Adresse gefragt. Sie können die bisherige Adresse, 32 bei Auslieferung, akzeptieren indem Sie einfach die Enter Taste drücken. Alternativ geben Sie eine andere Adresse ein und bestätigen diese dann mit der Enter Taste.

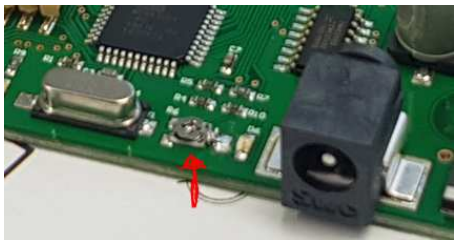
Stromstärke

Danach wählen Sie bitte eine der aufgelisteten max. Stromstärken aus. Die Geräte werden mit 0,7A Einstellung ausgeliefert.

Wenn die gewünschte Stromstärke nicht zur aktuellen Einstellung des Potentiometers passt, dann muss dieses entsprechend angepasst werden. In diesem Fall sehen Sie fortlaufend im Monitor die Stromstärke für die aktuelle Potentiometer Stellung.

Um das Potentiometer verstellen zu können entfernen Sie bitte die beiden Schrauben des Deckels 2. Dies ist der Deckel mit dem Aufkleber auf der Seite mit der USB Buchse. Deckel 1 deckt die ganzen Bundhülsen ab.

Das Potentiometer befindet sich zwischen der Strombuchse und dem Quartz.



Verstellen Sie das Potentiometer jetzt so lange bis der gewünschte Zielwert erreicht ist. Danach werden ihre Angaben im Controller gespeichert. Sie müssen dann den Controller von der Stromversorgung kurzzeitig trennen, um den Maintenance Modus zu verlassen.

Firmware Update

Die Firmware Versionen finden Sie in Git unter folgendem Link:

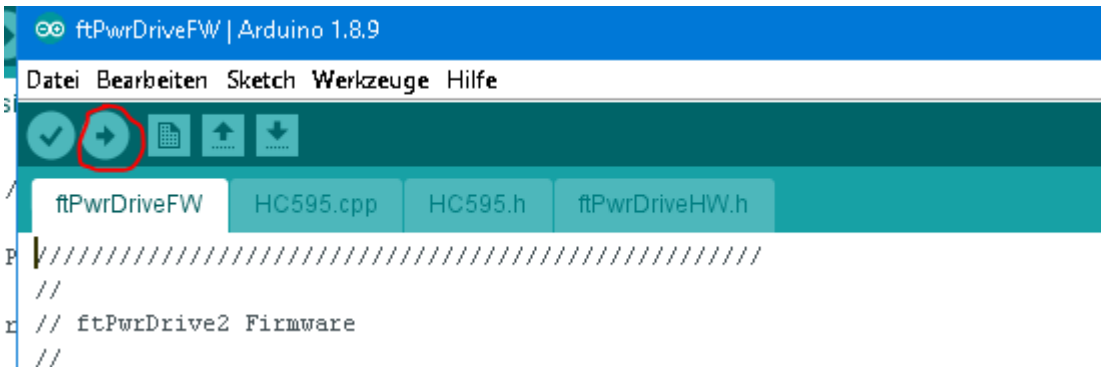
Sie benötigen dazu die Arduino IDE. Weiterhin müssen Sie zwei weitere Bibliotheken installieren.

Timer One: http://playground.arduino.cc/Code/Timer1

Timer Three: https://github.com/PaulStoffregen/TimerThree

Die herunter geladenen Dateien werden dann einfach als ZIP-Library installiert.

Nun einfach die neue Firmware in der Arduino IDE öffnen, Board „Arduino Leonardo“ auswählen und den passenden Port setzen. Vor dem Hochladen alle Peripherie (Motoren, Servos) abklemmen und die FW Hochladen.



Unterstütze Kommandos in der Firmware

Firmware Version: V0.92

Die Kommunikation erfolgt per I2C. D.h. die Kommandos können natürlich ganz einfach außerhalb der Arduino Umgebung ausgeführt werden. Oder wenn es aktuell für ihr Zielsystem noch keinen Support gibt.

Hinweise zu den Parametern.

Motor: Bitte verwenden Sie die Motorkonstanten.

Servos: Bitte verwenden Sie die Nummer 0 – 3

Distanz: Die Angabe erfolgt in Schritten

Speed: Die Angabe der Geschwindigkeit erfolgt in Schritten pro Sekunde.

Position Stepper: Die Angabe der Position erfolgt in Schritten zum gesetzten Nullpunkt.

Acceleration: Beschleunigungskommandos werden heute noch nicht unterstützt. Dies kommt in einer der folgenden FW-Releases.

Position Servo: Hier kann der Servo auf eine logische Position zwischen -60 und 60 gesetzt werden. Die Winkeldrehung ist servoabhängig.

Offset Servo: Mit diesem Wert kann die Mittenposition des Servos wie über die Stellschraube am Control-Set eingestellt werden.

Sie müssen nur in der Lage sein Daten per I2C zu senden. In der folgenden Tabelle werden nur die Kommandos gelistet. Ihr Kommando muss natürlich wie folgt aussehen:

I2C.sendData(I2CAddresse, Kommando, Parameter_1..Parameter_x);

I2C.sendData ist dann spezifisch ihrer Plattform.

Kommandoname	**Kom-man-do**	**Funktionsname mit Aufruf Parametern**	**Hinweistext**
SETWATCHDOG	0	void setWatchdog(long interval)	set watchdog timer
SETMICROSTEPMODE	1	void setMicrostepMode(mode)	set microstep mode - FULLSTEP, HALFSTEP, QUARTERSTEP, EIGTHSTEP, SIXTEENTHSTEP
GETMICROSTEPMODE	2	uint8_t getMicroStepMode(void)	get microstep mode - FULLSTEP, HALFSTEP, QUARTERSTEP, EIGTHSTEP, SIXTEENTHSTEP
SETRELDISTANCE	3	void setRelDistance(uint8_t motor, long distance)	set a distance to go, relative to actual position
SETABSDISTANCE	5	void setAbsDistance(uint8_t motor, long distance)	set a absolute distance to go

GETSTEPSTOGO	7	long getSetpsToGo(uint8_t motor)	number of needed steps to go to distance
SETMAXSPEED	8	void setMaxSpeed(uint8_t axis, long maxSpeed)	set max speed
GETMAXSPEED	9	long getMaxSpeed(uint8_t axis);	get max speed
STARTMOVING	10	void startMoving(uint8_t motor, boolean disableOnStop)	start motor moving, disableOnStop disables the motor driver at the end of the movement
STARTMOVINGALL	11	void startMovingAll(uint8_t maskMotor, uint8_t maskDisableOnStop)	same as StartMoving, but using uint8_t masks
ISMOVING	12	boolean isMoving(uint8_t motor)	check, if a motor is moving
ISMOVINGALL	13	uint8_t isMovingAll( )	return value is uint8_tmask, flag 1 is motor#1, flag2 is motor #2, ...
GETSTATE	14	uint8_t getState(uint8_t motor)	8754321 - flag 1 motor is running, flag 2 endstop, flag 3 EMS, flag 4 position overrun
SETPOSITION	15	void setPosition(uint8_t motor, long position)	set position
SETPOSITIONALL	16	void setPositionAll(long p1, long p2, long p3, long p4)	set position of all motors
GETPOSITION	17	long getPosition(uint8_t motor)	get position
GETPOSITIONALL	18	(long,long,long,long) getPositionAll(void)	get position of all motors
SETACCELERATION	19	void setAcceleration(uint8_t motor, long acceleration)	set acceleration
GETACCELERATION	20	long getAcceleration(uint8_t motor)	get acceleration
SETACCELERATIONALL	21	void setAccelerationAll(long acc1, long acc2, long acc3, long acc4	) set acceleration of all motors
GETACCELERATIONALL	22	(long,long,long,long) getAccelerationAll(void)	get acceleration of all motors
SETSERVO	23	void setServo(uint8_t servo, long position)	set servo position
GETSERVO	24	long getServo(uint8_t servo)	get servo position
SETSERVOALL	25	void setServoAll(long p1, long p2, long p3, long p4)	set all servos positions
GETSERVOALL	26	(long, long, long, long) getServoAll(void)	get all servo positions
SETSERVOOFFSET	27	void setServoOffset(uint8_t servo, long Offset)	set servo offset
GETSERVOOFFSET	28	long getServoOffset(void)	get servo offset
SETSERVOOFFSETALL	29	void serServoOffsetAll(long o1, long o2, long o3, long o4)	set servo offset all
GETSERVOOFFSETALL	30	(long, long, long, long) getServoOffsetAll(void)	get all servo offset

Beispiel:

Um die relative Distanz eines Motors zu setzen benötigen Sie das Kommando 3 „SETRELDISTANCE“.

i2c.sendData(i2cAddress, CMD_SETRELDISTANCE, motor, distance);

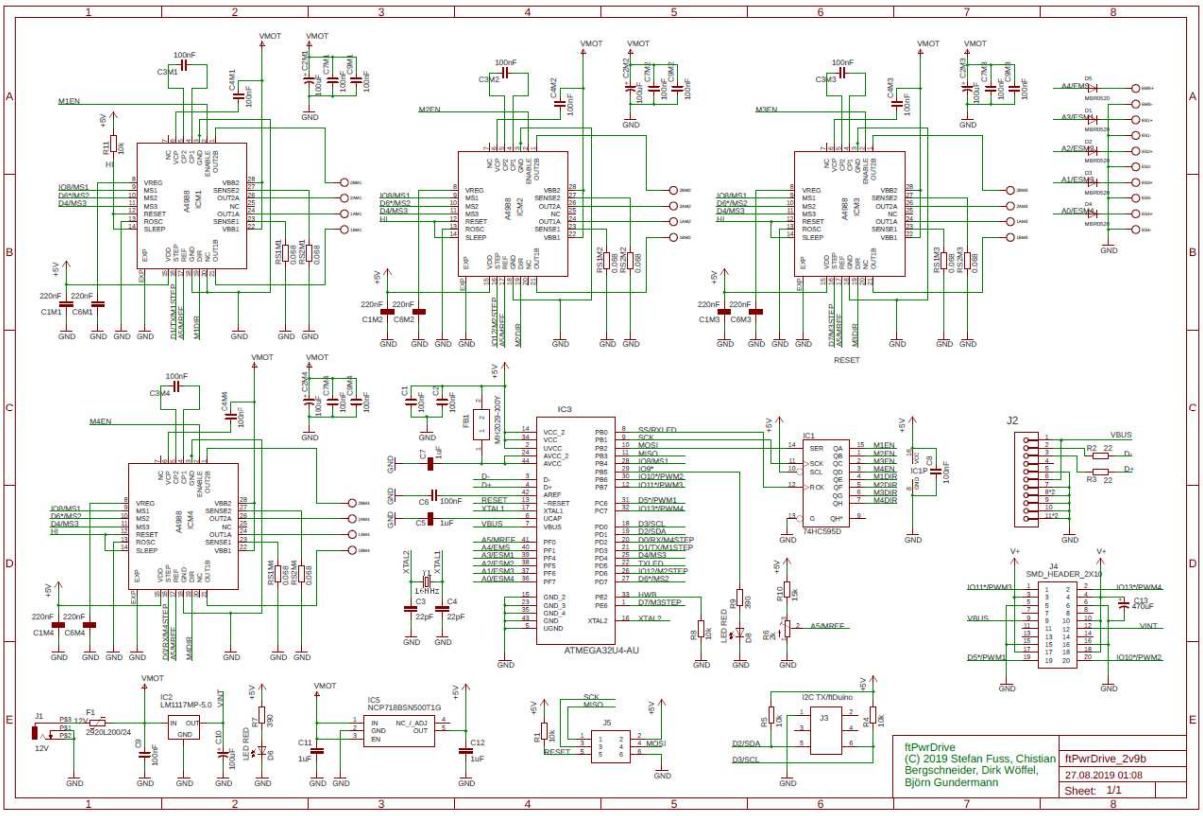
Der Sendeaufruf zum Setzen der Distanz 50 von Motor 1 sieht dann wie folgt aus.

➔ i2c.sendData(0x32, 3, 2, 50)

Hinweis: Bitte verwenden Sie die Motorkonstanten. Deswegen ergibt sich beim senden eine 2 für Motor 1.

Eine Liste mit den defines für die jeweiligen Kommandos befindet sind z.B. in der Datei „ftPwrDrive.cpp„. Diese ist Teil der Bibliothek für die Arduino Umgebung. Am besten übernehmen Sie diese defines für ihre Umgebung, um dann auch symbolisch darauf zuzugreifen. Die Kommando IDs werden sich zwar nicht ändern, dies ist aber der bessere Stil. Alternativ ersetzen Sie CMD_SETRELDISTANCE direkt durch eine 3

Anhang A: Schaltplan



Links

Hier eine Liste von sinnvollen Links zu Treibern und weiteren Informationen

ftDuino	fischertechnik compatible arduino	www.Ftduino.de info@ftduino.de Till Haarbaum
Fischertechnik	Hersteller des TX/TXT Controllers	www.fischertechnik.de
I²C	Serieller Datenbus	https://de.wikipedia.org/wiki/I%C2%B2C
I²C	Treiber Sammlung	https://github.com/ControlEverythingCommunity
Ft-Extender	Bilder/Video Fertigung	https://www.dropbox.com/sh/hyrzibvc3apt … –Zy8a?dl=0
Ft-Extender	Produktseite	http://www.gundermann.org
Ft-Extender	Technische Infos	https://github.com/elektrofuzzis/
ftDuinoTX-T	Nutzung des ftDuino als Erweiterung für den TX/TXT	https://github.com/elektrofuzzis/ftDuinoTX-T-
Pfostenbuchse	Beispiel Link von Reichelt	https://www.reichelt.de/Pfosten-Wannenstecker/PFL-6/3/index.html?ACTION=3&GROUPID=7437&ARTICLE=53153