Das Software-Interface des Brenner8

Firmware V0.12

Autor: sprut (www.sprut.de)
Stand: 07.09.2008

-

1 Inhaltsverzeichnis

-

-

2 NUTZUNGSBEDINGUNGEN:

DIE SOFTWARE DARF OHNE ENTRICHTUNG EINER LIZENZGEBÜHR BENUTZT WERDEN. DAS GILT FÜR DIE PRIVATE UND GEWERBLICHE NUTZUNG.

DIE PUBLIKATION DER SOFTWARE ERFOLGT "AS IS". FÜR DIE EINHALTUNG ZUGESICHERTER EIGENSCHAFTEN ODER FÜR SCHÄDEN, DIE DURCH DEN EINSATZ ENTSTANDEN SEIN KÖNNTEN, ÜBERNIMMT DER AUTOR KEINERLEI HAFTUNG. SIE NUTZEN DIE SOFTWARE AUF EIGENE GEFAHR!

3 Einleitung

Dieses Dokument beschreibt das Softwareinterface des Brenner8. Diese Informationen gelten entsprechend auch für den Brenner9.

4 Das USB-Interface

Der Brenner8 ist ein bus-powered USB-2.0 Gerät.

Interface	USB2.0 full speed
Stromversorgung	bus powered
Stromaufnahme	< 100 mA
VID	0x04D8
PID	0xFF0B
Anzahl der USB-Konfigurationen	1
Anzahl der Interfaces	1
Anzahl der Endpunkte	2

Endpunkt1:

Datenrichtung	OUT
Betriebsart	BULK
Puffergröße	64 Byte

Endpunkt2:

Datenrichtung	IN
Betriebsart	BULK
Puffergröße	64 Byte

USB-Timeouts:

Schreibzugriffe	100 ms
Lsezugriffe	1000 ms

-

5 Bootloader und Firmware

Der Bootblock des Steuer-PIC (Adressen 0x0000 ... 0x07FF) wird von der Bootloader Software eingenommen. Der restliche Adressbereich des Steuer-PIC-Programmspeichers steht für die Firmware zur Verfügung.

Als Bootloader für den Brenner8 wird der Bootloader-0 verwendet. Bootloader und Firmware sind unabhängige und eigenständig lauffähige Programme. Beide initialisieren das USB-Interface auf identische Art und Weise. Unabhängig davon, ob der Bootloader oder die Firmware aktiv ist, identifiziert sich der Brenner8 gegenüber dem PC-Betriebssystem identisch.

Der Bootloader startet nach einem Reset des Steuer-PIC wenn mindestens eine der beiden folgenden Bedingungen erfüllt ist:

1. Das Pin 1 des Steuer-PIC liegt auf low-Pegel
2. Der Inhalt der EEPROM-Zelle ist der Adresse 0xFE ist 0xFF.

Ist keine der beiden Bedingungen erfüllt, startet beim Reset die Firmware des Brenner8.

Die Brennsoftware USBoot kann den Bootloader aktivieren und eine neue Firmware mit Hilfe des Bootloaders einspielen.
Alternativ kann sie Software USBoot verwendet werden.

-

6 Bootloader

6.1 Grundlagen

Der Bootloader im Brenner8 basiert auf den Microchip-Bootloader zum „PICDEM USB FS DEMOBOARD“.

Die Kommunikation erfolgt über beide Endpunkte und wird immer vom PC initiiert. Sie läuft immer nach folgendem Schema ab:

1. Der PC schreibt einen Datenblock in den out-Endpunkt.
2. Der Bootloader arbeitet daraufhin eine Aufgabe ab.
3. Der Bootloader schreibt Daten in den in-Endpunkt
4. Der PC list den Datenblock aus dem in-Endpunkt

Auch bei Aufgaben, die eigentlich keinen Datentransport zum PC erfordern, wird wenigstens ein 1 Byte-langer Datenblock als Quittung zum PC zurückgegeben.

Die nachfolgende Tabelle zeigt die Struktur der Datenblöcke, die zwischen PC und Bootloader ausgetauscht werden. Die maximale Länge eines Datenblocks ist auf 64 Byte begrenzt.
Die konkrete Länge hängt vom Kommando ab. Wenn ein Kommando weder eine Adresse noch Nutzdaten benötigt, dann kann ein Datenblock auch aus nur einem Byte (dem Kommando selbst) bestehen.

Wird vom PC ein Datenblock gesendet, der für das konkrete Kommando zu lang ist, dann werden die überflüssigen Bytes am Ende des Datenblocks ignoriert.
War der Datenblock zu kurz, und es fehlen essenzielle Daten am Ende des Datenblocks, dann werden die Daten benutzt, die sich zufällig im USB-Pufferspeicher befinden.

Generelle Datenstruktur :

	Adresse	**Bedeutung**
Kommando	0x00	Beschreibt die Aufgabe
Datenlänge	0x01	Anzahl der Datenbytes
Adresse low	0x02	Bits 0..7 des Adresse
Adresse high	0x03	Bits 8..15 des Adresse
Adresse upper	0x04	Bits 16..23 des Adresse
Datenbyte 1	0x05	
Datenbyte 1	0x06	
...	...	
Datenbyte n	0xFF	

Folgende Kommandos werden vom Bootloader unterstützt:

Name des Kommandos	**Code im Datenblock**
READ_VERSION	0x00
READ_FLASH	0x01

WRITE_FLASH	0x02
ERASE_FLASH	0x03
READ_EEDATA	0x04
WRITE_EEDATA	0x05
READ_CONFIG	0x06
WRITE_CONFIG	0x07
RESET	0xFF

Wird ein Datenblock mit einem anderen (nicht definiertem) Kommando empfangen, so wird der Datenblock vom Bootloader ignoriert. Es erfolgt auch keine Rücksendung eines Antwort-Datenblocks.

6.2 Befehle für den Bootloader

6.2.1 READ_VERSION

Da ich nur eine feste VID_PID für meine USB-Geräte habe, melden sich alle meine Geräte beim PC mit dieser VID_PID an. Mit dem READ_VERSION-Kommando kann man aus dem Gerät zwei Kennbytes auslesen. Eines identifiziert das Gerät, ein zweites kennzeichnet die Softwareversion/Firmwareversion des Gerätes.

PC -> Bootloader

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x00

Der Bootloader sendet den Gerätecode 0x01.

Bootloader -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x00
0x01	keine	-
0x02	Version	-
0x03	Gerät	0x01

6.2.2 READ_FLASH

Mit diesem Befehl kann der Inhalt des Programmspeichers des Steuer-PIC ausgelesen werden. Es können maximal 59 aufeinanderfolgende Bytes auf ein Mal ausgelesen werden. Die Adresse gibt an, ab welcher Adresse im Flash-Programmspeicher mit dem Auslesen begonnen werden soll

PC -> Bootloader

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x01
0x01	Datenlänge	1 .. 59
0x02	Adresse low	
0x03	Adresse high	
0x04	Adresse upper	

-

Der Bootloader sendet den gleichen Datenblock zurück, an den aber die gesuchten Bytes angehangen wurden.

PC -> Bootloader

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x01
0x01	Datenlänge	1 .. 59
0x02	Adresse low	
0x03	Adresse high	
0x04	Adresse upper	
0x05	Datenbyte 1	
...	...	...
0xXX	Datenbyte n	

6.2.3 WRITE_FLASH

Mit diesem Befehl kann der Programmspeichers des Steuer-PIC beschrieben werden.
Es ist zwingend nötig, vor dem Schreiben den Flash-Speicherbereich mit dem ERASE_FLASH-Kommando zu löschen. Ansonsten werden die neuen Daten mit den alten Speicherinhalten UND-Verknüpft.

Es werden genau 16 aufeinanderfolgende Bytes auf ein Mal geschrieben. Die Adresse gibt an, ab welcher Adresse im Flash-Programmspeicher mit dem Schreiben begonnen werden soll. Die Startadresse muss am Beginn eines 16-Byte-Blocks liegen. Folglich sind die unteren 4 Bits der Adresse „0000“.

Wird eine andere Adresse angegeben, dann schreibt der Bootloader die Daten trotzdem ab dem Block-Anfang in den Flash. Ist die Datenlänge kleiner als 16 Byte, dann wird nichts geschrieben.

PC -> Bootloader

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x02
0x01	Datenlänge	0x10
0x02	Adresse low	0xX0
0x03	Adresse high	
0x04	Adresse upper	
0x05	Datenbyte 1	
...	...	...
0x14	Datenbyte 16	

Der Bootloader sendet als Quittung nur das Kommando zurück.

PC -> Bootloader

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x02

-

6.2.4 ERASE_FLASH

Mit diesem Kommando kann Flash-Speicher gelöscht werden. Das Löschen erfolgt immer in 64-Byte großen Speicherblöcken. Es können mit einem Mal mehrere aufeinanderfolgende 64-Byte-Blöcke gelöscht werden.
Der erste gelöschte Block ist derjenige auf den die Adresse verweist. Die niederwertigen 6 Bit der Adresse werden dabei ignoriert.

PC -> Bootloader

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x03
0x01	Anzahl der Blöcke	
0x02	Adresse low	
0x03	Adresse high	
0x04	Adresse upper	

Der Bootloader sendet als Quittung nur das Kommando zurück.

PC -> Bootloader

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x03

6.2.5 RESET

Mit diesem Kommando wird ein Reset des Steuer-PIC ausgelöst. Vorher meldet sich der Bootloader korrekt beim USB-Controller im PC ab.

PC -> Bootloader

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0xFF

Der Bootloader sendet KEINE Quittung an den PC zurück.

-

6.3 Ein Beispiel für die Nutzung des Bootloaders

Folgender Delphi-Code zeigt die Nutzung des Bootloaders. Sie benutzt die Prozedur **Sende_Empfange(NrS, NrE).**
Diese Routine sendet **NrS** Bytes aus dem Byte-Array **send_buf** zum USB-Device und empfängt anschließend **NrE** Bytes aus dem USB-Device in das Byte-Array **receive_buf**.
Beide Byte-Arrays sind je 64 Byte lang.

Der Bootloader wird mit READ_VERSION erkannt.
Anschließend wird der Programmspeicher des Steuer-PIC im Bereich vom 0x0800 bis 0x7FFF mit neuem Code beschrieben. Der neue Code stammt aus dem Array **Hexin.Flash**.
Danach wird der neu beschriebene Bereich des Steuer-PIC wieder ausgelesen, und mit **Hexin.Flash** verglichen, um eventuelle Fehler zu erkennen.
War alles fehlerfrei, dann wird die EEPROM-Zelle 0xFE des Steuer-PIC mit 0 beschrieben, und der Steuer-PIC neu „gebootet“.

```
// ist denn da ein Bootloader ?
send_buf[0]:=READ_VERSION;    //0
Sende_Empfange(1, 4);
if (receive_buf[0] <>  READ_VERSION) or (receive_buf[3]<>1) then exit;

//brennen
Adresse    := $800;
Endadresse :=$7FFF;
while Adresse<Endadresse do begin
  //64 byte löschen
  send_buf[0]:= ERASE_FLASH;
  send_buf[1]:= 1;                              // 1 x 64 byte
  send_buf[2]:=  Adresse and $0000FF;           // low
  send_buf[3]:= (Adresse and $00FF00) shr  8;  // high
  send_buf[4]:= (Adresse and $FF0000) shr 16;  // upper
  Sende_Empfange(5,1);
  //4 x 16 byte schreiben
  for k:=0 to 3 do begin
    send_buf[0]:= WRITE_FLASH;
    send_buf[1]:= 16;                             // länge 16 Byte
    send_buf[2]:=  Adresse and $0000FF;           // low
    send_buf[3]:= (Adresse and $00FF00) shr  8;  // high
    send_buf[4]:= (Adresse and $FF0000) shr 16;  // upper
    for L:=0 to 15 do  send_buf[5+L]:=Hexin.Flash[Adresse+L] and $FF;
    Sende_Empfange(21,1);
    Adresse:=Adresse+16
  end;
end;

//prüfen
Fehler:=0;
Adresse    := $800;
Endadresse :=$7FFF;
while Adresse<Endadresse do begin
```

```
  send_buf[0]:= READ_FLASH;
  send_buf[1]:= 16;                              // länge
  send_buf[2]:=  Adresse and $0000FF;            // low
  send_buf[3]:= (Adresse and $00FF00) shr  8;    // high
  send_buf[4]:= (Adresse and $FF0000) shr 16;    // upper
  Sende_Empfange(5,send_buf[1]+5);

  for k:=0 to receive_buf[1]-1 do begin
    if (receive_buf[k+5] and $FF) <> (Hexin.Flash[Adresse+k] and $FF)
    then begin
      inc(Fehler);
    end;
  end;
  Adresse:=Adresse+16
end;

if Fehler=0 then begin
  // EEPROM-Zellen 0xFE und 0xFF mit 0 beschreiben
  send_buf[0]:= WRITE_EEDATA;
  send_buf[1]:= 2;            // länge 1 Byte
  send_buf[2]:= $FE;          // low
  send_buf[3]:= 0;            // high
  send_buf[4]:= 0;            // upper
  send_buf[5]:= 0;
  send_buf[6]:= 0;
  Sende_Empfange(7,1);

  // neu booten
  send_buf[0]:= RESET;
  Sende_Empfange(1,0);

end else Memo.lines.add('Flash-Error');
```


-

7 Firmware

Die Firmware des Brenner8 Nimmt per USB Befehle vom PC entgegen, führ sie aus und quittiert sie per USB. Sie liegt im Steuer-PIC des Brenner8 ab der Adresse 0x0800.

Die Kommunikation erfolgt über beide Endpunkte und wird immer vom PC initiiert. Sie läuft immer nach folgendem Schema ab:

1. Der PC schreibt einen Datenblock in den out-Endpunkt.
2. Die Firmware arbeitet daraufhin eine Aufgabe ab.
3. Die Firmware schreibt Daten in den in-Endpunkt
4. Der PC list den Datenblock aus dem in-Endpunkt

Auch bei Aufgaben, die eigentlich keinen Datentransport zum PC erfordern, wird wenigstens ein 1 Byte-langer Datenblock als Quittung zum PC zurückgegeben.

In jedem Datenblock ist das erste Byte das Kommando. Es beschreibt die Aufgabe für den Steuer-PIC. Die folgende Tabelle enthält eine Liste aller existierenden Kommandos:

Kommando	**Code**	**Bedeutung**
READ_VERSION	0x00	Lesen der Gerätekennung und der FW-Version
LED_ONOFF	0x31	Ein/Ausschalten von LEDs
RD_ADC	0x37	Messen einer Spannung
SET_AN	0x38	Auswahl des ADC-Eingangs
SET_PWM	0x39	Einstellen eines PWM-Verhältnisses
SET_SOC	0x3A	Auswahl-PIC-Sockel
SET_SIGNAL	0x3B	ICSP-Leitungen einzeln ein/aus-Schalten
SET_DIR	0x3C	ICSP-Datenleitung auf lesen/schreiben schalten
READ_DATA	0x3D	Lesen der ICSP-Datenleitung
~~ICSP_WRITE~~	~~0x3E~~	
~~ICSP_READ~~	~~0x3F~~	
SET_KERN	0x40	Auswahl der PIC-Familie
SET_PICTYPE	0x41	Alle nötigen Informationen über den PIC liefern
SET_ADRESS	0x42	Einstellen eines Adressbereichs
SET_VPP	0x43	Einstellen der Vpp-Spannung und der Stabilisierung
READ_EDATA	0x44	EEPROM des Steuer-PIC auslesen
WRITE_EDATA	0x45	EEPROM des Steuer-PIC beschreiben
READ_CHIPID	0x50	Aus dem Target die Chip-ID-Auslesen
READ_FLASH	0x51	Programmspeicher des Target lesen
READ_EEPROM	0x52	EEPROM des Target lesen
READ_IDDATA	0x53	ID-Daten des Target lesen
READ_CONFIG	0x54	Configuration des Target lesen

-

WRITE_FLASH	0x60	Programmspeicher des Target beschreiben
WRITE_EEPROM	0x61	EEPROM des Target beschreiben
WRITE_IDDATA	0x62	ID-Daten des Target beschreiben
WRITE_CONFIG	0x63	Konfiguration des Target beschreiben
ERASE	0x70	Target löschen
REMOVECP	0x71	Codeprotection des Target löschen (Bulk Erase)
SUPPORTED	0x72	Abfrage der von der Firmware unterstützten Typen
RESET	0xFF	Einen Reset des Steuer-PIC auslösen

7.1 Ablauf

Die Brennsoftware geht mit dem Brenner8 wie folgt um:

- Mit READ_VERSION wird überprüft, ob überhaupt ein Brenner8 vorhanden ist.
- Mit SET_AN und READ_ADC wird die Spannung der Z-Diode gemessen, und daraus die Betriebsspannung errechnet, die auch ADC-Referenzspannung ist. Ein ADC Korrekturwert für spätere ADC-Messungen wird errechnet.
- Danach wird mit SET_SOC die elektrische Verbindung zum Target festgelegt.
- Mit SET_KERN wird die PIC-Familie festgelegt.
- Nun wird mit SET_PWM eine Standard-Programmierspannung (12..13V) eingestellt und mit SET_AN und READ_ADC überprüft.

- Mit READ_CHIPID wird der Typ des Target identifiziert.
- Für das identifizierte Target werden alle nötigen Daten aus der PIC-Database ausgelesen, und an den Brenner8 mit SET_PICTYPE übertragen.
- Mit SET_PWM / SET_VPP wird die korrekte Programmierspannung eingestellt und mit SET_AN und READ_ADC überprüft.

- Mit ERASE und/oder REMOVECP wird das Target gelöscht.
- Mit SET_ADRESS und vielen WRITE_FLASH wird der Programmspeicher des Target beschrieben.
- Mit SET_ADRESS und vielen READ_FLASH wird der Programmspeicher des Target ausgelesen, und mit dem Sollinhalt verglichen.
- Sinngemäß passiert das gleiche mit EEPROM, ID und CONFIG.

- fertig

7.2 Allgemeine Kommandos

7.2.1 READ_VERSION

Da ich nur eine feste VID_PID für meine USB-Geräte habe, melden sich alle meine Geräte beim PC mit dieser VID_PID an. Mit dem READ_VERSION-Kommando kann

-

man aus dem Gerät zwei Kennbytes auslesen. Eines identifiziert das Gerät, ein zweites kennzeichnet die Softwareversion/Firmwareversion des Gerätes.

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x00

Der Brenner8 sendet den Gerätecode 0x00. Als Version wird die Firmwareversion gesendet (0x06 entspricht V 0.6).

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x00
0x01	keine	-
0x02	Version	-
0x03	Gerät	0x00

7.2.2 LED_ONOFF

Mit diesem Kommando können die beiden LED des Brenner8 ein- und ausgeschaltet werden.

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x31
0x01	Status	0, 1, 4, oder 5

Für das Status-Byte sind folgende Werte definiert:

Status-Wert	Wirkung
0	LED1 einschalten
1	LED2 einschalten
4	LED1 ausschalten
5	LED2 ausschalten

Bei einem gültigen Status-Wert liefert der Brenner8 nur das Kommando als Quittung. Ein ungültiger Status-Wert wird nicht quittiert.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x31

7.2.3 RD_ADC

Es wird die Spannung am gerade selektierten Eingang (siehe SET_AN) des ADC gemessen.

PC -> Brenner8

Adresse	Bedeutung	Inhalt

-

0x00	Kommando	0x37

Der Brenner8 sendet das 10-Bit Resultat des ADC. ADRESL sind die unteren 8 Bit des Ergebnisses und ADRESH die beiden oberen Bits.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x37
0x01	ADRESL	-
0x02	ADRESH	-

Als positive Referenzspannung für den ADC dient die Betriebsspannung, die bei etwa 4,7V liegt. Ihr präziser Wert lässt sich bestimmen, wenn man zum Vergleich die (bekannte) Z.-Dioden-Spannung misst.
Die Vpp-Spannung wird dem ADC-Pin über einen Spannungsteiler mit einem Teilverhältnis von ca. 3.14:1 zugeführt.
Der genaue Wert der Z-Spannung sowie des Spannungsteilerverhältnisses wird durch Kalibrierung ermittelt, und im EEPROM des Steuer-PIC abgespeichert.

7.2.4 SET_AN

Der ADC kann wahlweise benutzt werden die Programmierspannung Vpp oder die Z-Diodenspannung Uz zu messen. Bevor die eigentliche Messung mit RD_ADC vorgenommen wird, muß mit SET_AN der richtige ADC-Eingang ausgewählt werden.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x38
0x01	ADC-Eingang	0 – Vpp 1 - Uz

Ist „ADC-Eingang = 0“, dann wird der Vpp-Eingang (RA1/AN1) ausgewählt. Ansonsten wird der Zu-Eingang (RA3/AN3) selektiert.

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x38

7.2.5 SET_PWM

Die Programmierspannung wird im Brenner8 mit einem Boost-Konverter erzeugt, der vom PWM-Kanal 1 „gepumpt“ wird.
Die erzeugte Ausgangsspannung hängt vom Taktverhältnis der PWM-Kanals sowie von der Last an der Vpp-Leitung ab. Mit SET_PWM wird das Taktverhältnis eingestellt, wobei gilt DC = 120/pwm_off. Die erzeugte Spannung steigt mit dem pwm_off Wert an.

- bei pwm_off=0 beträgt sie ca. 4,5V.

-

- bei pwm_off=70 erreicht sie ihr Maximum (25 .. 30V).

pwm_off ist der primär verwendete PWM-Wert. Je nach Betriebsart der Vpp-Erzeugung (siehe SET_VPP) kann beim Zuschalten von Vpp zum Target auf pwm_on umgeschaltet werden:

In der Theorie beträgt die erzeugte Spannung:

Vpp = 4,7V * 120 /(120-pwm_off)

In der Praxis werden die für eine bestimmte Spannung (VppSoll) nötigen pwm-Werte wie folgt ermittelt:

```
pwm_off:= VppSoll * gain_off - pwm0V_off
pwm_on := VppSoll * gain_on  - pwm0V_on;
```

Die Werte gain_off, gain_on, pwm0V_off und pwm0V_on werden durch Kalibrierung ermittelt, und im EEPROM des Steuer-PIC abgespeichert.

Beim Reset wird ein Default-Wert von pwm_off=12 benutzt, um eine ungefährliche Vpp von deutlich unter 13V zu erzeugen.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x39
0x01	pwm_off	0 70
0x02	pwm_on	0..70

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x39

7.3 Kommandos für Testfunktionen

7.3.1 SET_SIGNAL

Zum Programmieren des Target ist dieser Befehl nicht nötig. Er dient nur zum Test der Hardware bei der Fehlersuche, sowie zur Kalibrierung.

Mit diesem Kommando können die 5 Signale des ICSP-Anschlusses am Testsockel bzw. am ICSP-Steckverbinder einzeln ein- und ausgeschaltet werden.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x3B
0x01	Befehl	0x01, 0x02, 0x04, 0x08 0x10, 0x20, 0x40, 0x80

-

Für das Befehl-Byte sind folgende Werte definiert:

Befehl-Wert	Wirkung
0x01	Vdd (&Vss) einschalten
0x10	Vdd (&Vss)ausschalten
0x02	Vpp einschalten
0x20	Vpp ausschalten
0x04	PGD high
0x40	PGD low
0x08	PGC high
0x80	PGC low

Trotz der bitweisen Codierung kann immer nur ein Pin geschaltet werden. Welche Pins genau geschaltet werden hängt von der mit SET_SOC gemachten Einstellung ab. Wurde mit SET_SOC die Einstellung „8/14-Pin“ gemacht, dann wird mit Vdd (+5V) auch Vss (0V) geschaltet.
Bei einem gültigen Befehl-Wert liefert der Brenner8 nur das Kommando als Quittung. Ein ungültiger Befehl-Wert wird nicht quittiert.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x3B

7.3.2 SET_DIR

Zum Programmieren des Target ist dieser Befehl nicht nötig. Er dient nur zum Test der Hardware bei der Fehlersuche.

Mit diesem Kommando kann die Datenleitung des ICSP-Anschlusses als Eingang oder Ausgang eingestellt werden. Benutzt wird der Befehl ausschließlich zu Test der Datenleitung im Rahmen der Fehlersuche. Soll mit dem READ_DATA Kommando der Pegel des PGD-Pins gelesen werden, dann ist das Pin des Steuer-PIC vorher mit dem Befehl 0x00 zum Eingang zu machen. Soll der Steuer-PIC aber mit SET_SIGNAL das PGD-Pin auf high oder low setzen, dann muss zuvor z.B. mit dem Befehl 0x01 das PGD-Pin des Steuer-PIC zum Ausgang gemacht werden. Welches Pin das PGD-Pin ist, dass hängt von der mit SET_SOC gemachten Einstellung ab.

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x3C
0x01	Befehl	0, other

Für das Befehl-Byte sind folgende Werte definiert:

Befehl-Wert	Wirkung
0x00	PGD wird Eingang des Steuer-PIC und Ausgang des Target
0x01 .. 0xFF	PGD wird Ausgang des Steuer-PIC

-

	und Eingang des Target

Bei einem gültigen Befehl-Wert liefert der Brenner8 nur das Kommando als Quittung. Ein ungültiger Befehl-Wert wird nicht quittiert.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x3C

7.3.3 READ_DATA

Zum Programmieren des Target ist dieser Befehl nicht nötig. Er dient nur zum Test der Hardware bei der Fehlersuche.

Mit diesem Kommando der Pegel der Datenleitung des ICSP-Anschlusses abgefragt werden:

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x3D

Der Brenner8 antwortet mit dem Kommando gefolgt von einem Byte. Falls am PGD-Pin low-Pegel anliegt ist dieses Byte=0, ansonsten ist es 1.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x3B
0x01	PGD-Wert	0,1

7.4 Kommandos für den Steuer-PIC

7.4.1 SET_SOC

Je nach Gehäusebauform des Target muss der Steuer-PIC das ICSP-Protokoll mit anderen Pins erzeugen. Deshalb muss dem Brenner8 vor dem ersten Zugriff auf ein Target die Gehäusegröße des Target im IC-Sockel mitgeteilt werden.

Die für Sockel zulässigen Werte sind 0, 1 und 2. Falls der ICSP-Steckverbinder benutzt werden soll, dann ist Sockel=1 zu wählen.
Beim Reset wird ein Default-Wert von Sockel=1 benutzt.

Wird ein ungültiger Sockel-Wert verwendet (z.B. 0xFF), dann erzeugt der Steuer-PIC keine ICSP-Signale. Die Pins 2, 4 und 5 des ICSP-Steckers sind hochohmig. Ab der Brenner-Revision 3 liegt am Pin 1 des ICSP-Steckers in diesem Fall 5V (hochohmig). Dadurch wird auf einer eventuell angeschlossenen Testplatine der Reset-Zustand beendet, und die Software im Target-PIC gestartet. Auf der Testplatine muß dann aber eine eigene Stromversorgung für Vdd aktiviert werden.

-

Fuer den Brenner8P/Brenner8miniP steht noch der Sockelwert 0xFE zur Verfügung. Er dient dem Start einer via ICSP angeschlossenen Testplatine, die keine eigene Stromversorgung hat.
Die Pins 4 und 5 des ICSP-Steckers sind hochohmig. Am Pin 2 werden 5V als Vdd-Spannung an den Target-PIC gelegt. Ab der Brenner-Revision 3 liegt am Pin 1 des ICSP-Steckers in diesem Fall 5V (hochohmig). Dadurch wird auf einer eventuell angeschlossenen Testplatine der Reset-Zustand beendet, und die Software im Target-PIC gestartet. Die Testplatine darf nicht mehr als 100 mA Betriebsstrom aus der Vdd-Leitung (Pin 2) aufnehmen.

Falls beim Empfang des Kommandos Vpp zum Target zugeschaltet war, so wird die Spannung deaktiviert.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x3A
0x01	Sockel	0x00 – 8 / 14 Pin 0x01 – 18Pin / ICSP 0x02 – 28 / 40 Pins 0xFE – Run-Vdd 0xFF – Run

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x3A

7.4.2 SET_KERN

Je nach PIC-Familie müssen völlig unterschiedliche Verfahren bei der Kommunikation mit dem Target verwendet werden. Deshalb muss dem Brenner8 vor dem ersten Zugriff auf ein Target die Befehlsbreite des PIC-Kerns des Target mitgeteilt werden.

Die für Kern zulässigen Werte sind 12, 14, 16, 30 und 33.
Die Routinen für Kern=12 sind aus Platzgründen z.Z. in der Firmware nicht enthalten.
Die Routinen für Kern=33 sind z.Z. in der Entwicklung. Wird die Funktion mit einem zulässigen Wert (12, 14, 16, 30, 33) aufgerufen, für den die Routinen aber in der firmware nicht enthalten sind, so wird im Antwortblock anstelle des Kommandos nur eine 0 übertragen.

Beim Reset wird ein Default-Wert von Kern=14 benutzt.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x40
0x01	Kern	12 – ~PIC12Fxxx 14 – ~PIC16Fxxx 16 – PIC18Fxxxx

-

		18 – PIC18FxxJxx 30 – dsPIC30Fxxxx 33 – PIC24/dsPIC33

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x40 / 0x00

7.4.3 SET_PICTYPE

Für das korrekte Flashen des Target benötigt der Brenner8 eine große Anzahl von Informationen. Diese werden dem Brenner8 mit dem Kommando SET_PICTYPE übertragen. Ab dem 2. Byte des Datensatzes (Adresse 0x01) steht eine 34 Byte lange Datenstruktur, deren Definition im grauen Feld unten beschrieben ist.

Die Daten stammen aus der PIC-Database.
Beim Reset werden alle Werte von *taktik* auf 0 gesetzt, und power mit 40 beschrieben.

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x41
0x01 ... 0x22	PICtype	

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x41

```
typedef struct T_taktik          // 7 Byte
{
                                 byte         flash;
                                 byte         eeprom;
                                 byte         id;
                                 byte         config;
                                 byte         erase;
                                 byte         cp;
                                 byte         read_eeprom;
} T_taktik;


typedef struct T_latches         // Schreibpuffergrößen in Byte
                                 // 5 Byte
{
                                 byte         pgm ;
                                 byte         eedata;
```


-

```
                                byte          userid;
                                byte          cfg;
                                byte          rowerase;    // Löschbereich
                                                           // in Byte
} T_latches;


typedef struct T_wait           // alle Brenn-Zeiten in Mikrosekunden
                                // 14 Byte
{
                                word          pgm;
                                word          lvpgm;
                                word          eedata;
                                word          cfg;
                                word          userid;
                                word          erase;
                                word          lverase;
} T_wait;

typedef struct T_PICtype        // 8 + 7 + 5 + 14 = 34 Byte
{
                                byte                 kommando;
                                byte                 cpu;
                                byte                 power;
                                unsigned int  blocksize;
                                byte                 pins;
                                byte                 vpp;
                                word                 panelsize;

                                T_taktik      taktik;
                                T_latches     latches;
                                T_wait         wait;
} T_PICtype;
```

7.4.4 SET_ADRESS

Mit diesem Befehl wird festgelegt, auf welchen Adressbereich sich die nächsten Schrei- und Leseoperationen zum und vom Target beziehen. Deshalb ist SET_ADRESS vor dem Zugriff auf den Flash-Programmspeicher, EEPROM, ID-Bereich und Config anzuwenden.

Anfang: Adresse der ersten Zelle die gebrannt oder gelesen werden soll
Ende: Adresse der letzten Zelle die gebrannt oder gelesen werden soll

Anfang und Ende sind 24-Bit lange Werte, die in jeweils 3 Byte übertragen werden.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x42
0x01	Anfang_low	Bit 0..7
0x02	Anfang_high	Bit 8..15
0x03	Anfang_upper	Bit 16..23
0x04	Ende_low	Bit 0..7
0x05	Ende_high	Bit 8..15
0x06	Ende_upper	Bit 16..23

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x42

7.4.5 SET_VPP

Dieser Befehl steuert die Stabilisierung der Programmierspannung, und ist momentan noch eine Baustelle.
Prinzipiell wird ein 10-Bit-Wert übergeben. Er entspricht dem Messergebnis, das der ADC beim Messen von Vpp liefern sollte. Die niederwertigen 8 Bit sind VppL und die oberen 2 Bit in VppH. Die 6 höherwertigen Bits in VppH stehen für Steuerzwecke zur Verfügung.

Es gibt 4 Modes, von denen aber z.Z. nur 2 Modes zur Verwendung zugelassen sind:

- Mode 0: keine Regelung (für Test und Kalibrierung)
- Mode 1: kontinuierliche Regelung
- Mode 3: Begrenzung (beim Lesen und Brennen von Target-PICs)

Mode0: Keine Regelung
Wird VppL=VppH=0 übergeben, wird jegliche Regelung und Überwachung von Vpp deaktiviert. Die mit SET_PWM eingestellten Werte werden unverändert benutzt. Das ist z.B. zu Testzwecken oder zur Kalibrierung nötig.

Mode1: Kontinuierliche Regelung
In diesem Mode wird die erzeugte Spannung kontinuierlich stabil auf einem Sollwert gehalten. Dabei wird ein festes PWM-Tastverhältnis von 40% eingehalten. Dieser Mode wird erst ab der Firmware 0.20 zugelassen.
In schneller kontinuierlicher Folge wird die Vpp-Spannung kontinuierlich gemessen. Das Messergebnis wird mit einem Sollwert verglichen. Überschreitet die Spannung den Sollwert, dann wird die PWM-Schaltung vorrübergehend deaktiviert. Unterschreitet die Spannung einen Minimalwert, dann wird die PWM-Schaltung wieder aktiviert.
Der Sollwert wird als 10-Bit ADC- Sollwert (ADCsoll) mit dem Befehl an die Firmware übergeben. Die unteren 8 Bit stehen in VppL, die oberen beiden Bits stehen in VppH. Der Minimalwert entspricht „Sollwert – 8“. Da die ADC-Auflösung etwa 15 mV (Vpp) entspricht, liegt der Minimalwert ca. 0,12V unter dem Sollwert.
Der ADC-Sollwert kann nach folgender Formel ermittelt werden:

ADCsoll = Vpp / ((Vusb / 1024) x Div)

mit: Vpp - gewollte Vpp-Spannung
Vusb - momentane Brenner8-Betriebsspannung (ca. 4,9V)
Div - Teilerverhältnis des Vpp-Spannungsteilers (ca. 3,12)

Mode 1 wird aktiviert, wenn ein ADCsoll-Wert von mehr als 0 übertragen wird, und die oberen 6 Bits von VppH alle „0“ sind. Es gilt dann:
VppL = ADCsoll & 0x00FF
VppH = (ADCsoll & 0x0300) / 0x100

Falls durch eine Fehlfunktion oder eienn falschen Befehl die Vpp auf über ca.14V (ADC-Wert>1000) ansteigen sollte, erfolgt eine Notabschaltung von Vpp und die grüne LED wird dauerhaft eingeschaltet

Mode3: Begrenzung
Der Begrenzungsbetrieb (Mode 3) wird aktiviert, wenn in VppH Das Bit 6 auf 1 gesetzt wird.
Der Brenner8 benutzt nun die bei SET_PWM übergebenen Werte für den Boost-Konverter zur Vpp-Erzeugung. Solange Vpp nicht zum Target zugeschaltet ist wird vpp_off verwendet. Wird Vpp zum Target geschaltet wird vpp_on verwendet. Regelmäßig wird die Vpp-Spannung mit dem ADC gemessen, und der Messwert mit VppH_VppL verglichen. Ist die Spannung zu hoch, wird sie durch Verringerung von vpp_on bzw. vpp_off verkleinert. Eine Heraufsetzung der Vpp-Spannung erfolgt dagegen nicht.
Falls Vpp über ca.14V (ADC-Wert>1000) ansteigen sollte, erfolgt eine Notabschaltung von Vpp und die grüne LED wird dauerhaft eingeschaltet.
Dieser Mode liefert tendenziell zu kleine Spannungen. Das dient der Sicherheit der Tasrget-PICs, hkann aber auch zu Problemen bei Löschen von PICs mit viel Flash-Speicher führen

Mittelfristig wird eine verbesserte Vpp-Regelung eingeführt.

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x43
0x01	VppL	0xXX
0x02	VppH	0x4X

Die Verwendung eines nicht zugelassenen Modes kann zur Zerstörung des Target führen!

Mode	Regelverfahren	Aktivieren durch	zugelassen
Mode 0	Keine Regelung	VppL=VppH=0	**Ja**
Mode 1	Kontinuierliche Regelung	VppH & 0xFC=0	Nein
Mode 2	Regelung nur nach Vpp-Schaltung	VppH & 0x80=0x80	Nein
Mode 3	Begrenzung	VppH & 0x40=0x40	**ja**

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x43

7.4.6 READ_EDATA

Mit diesem Befehl kann der EEPROM des Steuer-PIC ausgelesen werden. Das ist z.B. nötig, wenn man die im Brenner abgelegten Kalibrierdaten auslesen muss. Übertragen wird mit dem Kommando die Startadresse im EEPROM als 16-Bit-Wert. Da der Steuer-PIC PIC18F2550 nur die EEPROM-Adressen 0x00 bis 0xFF kennt,

-

bleibt in der Praxis der High-Teil der Adresse unbenutzt. Es können mit einem Befehl maximal 59 Bytes ausgelesen werden.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x44
0x01	Anfang _low	Bit 0..7
0x02	Anfang_high	0x00
0x03	Zahl der Bytes	0x01 ..0x3B

Der Brenner8 die vom PC empfangenen 4 Byte, ergänzt um die gewünschten Datenbytes aus dem EEPROM. Der Antwortblock ist im Übrigen immer 63 Bytes lang. In den von den Daten nicht benötigten Zellen stehen Zufallswerte.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x44
0x01	Anfang _low	Bit 0..7
0x02	Anfang_high	0x00
0x03	Zahl der Bytes	0x01 ..0x3B
0x04	Datenbyte 1	
...		
0xXX	Letztes Datenbyte	
0x3E	-	-

7.4.7 WRITE_EDATA

Mit diesem Befehl können Daten in den EEPROM des Steuer-PIC geschrieben werden. Das ist z.B. nötig, wenn man Kalibrierdaten im Brenner ablegen möchte, oder um den Bootloader zu aktivieren.
Übertragen wird mit dem Kommando die Startadresse im EEPROM als 16-Bit-Wert. Da der Steuer-PIC PIC18F2550 nur die EEPROM-Adressen 0x00 bis 0xFF kennt, bleibt in der Praxis der High-Teil der Adresse unbenutzt.

Es können mit einem Befehl maximal 59 Bytes geschrieben.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x45
0x01	Anfang _low	Bit 0..7
0x02	Anfang_high	0x00
0x03	Zahl der Bytes	0x01 ..0x3B
0x04	Datenbyte 1	
...		
0xXX	Letztes Datenbyte	
0x3E	-	-

-

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x45

7.4.8 RESET

Mit diesem Kommando wird ein Reset des Steuer-PIC ausgelöst. Vorher meldet sich der Brenner8 korrekt beim USB-Controller im PC ab.

PC -> Bootloader

Adresse	Bedeutung	Inhalt
0x00	Kommando	0xFF

Der Brenner8 sendet KEINE Quittung an den PC zurück.

7.5 Kommandos zum Programmieren des Target

7.5.1 READ_CHIPID

Jeder moderne Flash-PIC (alle außer PIC16F84/83) besitzt einen Identifizierungscode, den man auslesen kann. Damit lässt sich der PIC-Typ identifizieren.
Die Brennersoftware liest die ChipID aus, und entnimmt auf Grundlage der ChipID aus der PIC-Database alle nötigen Informationen über den PIC. Diese werden dann mit SET_PICTYPE an den Brenner8 übertragen.

Vor diesem Kommando müssen sowohl Sockel (SET_SOC) als auch Kern (SET_KERN) korrekt eingestellt sein.

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x50

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x50
0x01	ChipID_low	
0x02	ChipID_high	

7.5.2 READ_FLASH

Mit diesem Kommando lässt sich ein beliebiger Teil des Flash-Programmspeicher des Targets in kurzer Zeit auslesen.

-

Bevor man den Programmspeicher mit READ_FLASH auslesen kann, muss man den interessierenden Speicherbereich mit SET_ADRESS festgelegt haben. Danach startet man die Leseprozedur mit dem Aufruf von READ_FLASH

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x51

Der Brenner8 sendet daraufhin einen Datenblock mit zurück. Er besteht aus einem 3 Bytes langem Kopf und einem angehängten Datenblock. Die Länge des übertragenen Datenblocks (in Byte) steht an der Adresse 0x01.

PIC16Fxxx
Da der Programmspeicher eines 14-Bit-PIC breiter als 1 Byte ist, belegt eine Flash-Zelle immer 2 Bytes. Die Übertragung erfolgt little endian (Low-Byte zuerst).

PIC18Fxxx
Der Programmspeicher eines 16-Bit-PIC ist im Grunde byteweise aufgebaut, so dass jedes Byte einer Flash-Adresse entspricht, es aber auch ungerade Flash-Adressen gibt.

dsPIC30Fxxx
Da der Programmspeicher eines 24-Bit-PIC (dsPIC30F) breiter als 1 Byte ist, belegt eine Flash-Zelle immer 3 Bytes. Die Übertragung erfolgt little endian (Low-Byte zuerst).

Die Adresse 0x02 ist nur dann „Null“, wenn der mit dem aktuellen Datenblock das Ende des mit SET_ADRESS festgelegten Adressbereichs erreicht wurde. Ist das Endekennzeichen aber nicht „Null“, dann können mit weiteren READ_FLASH-Befehlen weitere Datenblöcke ausgelesen werden Ein weiteres SET_ADRESS ist nicht nötig.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x51
0x01	Anzahl der Bytes	
0x02	Ende-Kennzeichen	0 – Ende
0x03	Byte Nr1	
...		
0xXX	letztes Byte	

7.5.3 READ_EEPROM

Dieser Befehl funktioniert analog zum READ_FLASH-Befehl. Es wird aber der EEPROM des Target ausgelesen.
Bevor man den EEPROM mit READ_ EEPROM auslesen kann, muss man den interessierenden Speicherbereich mit SET_ADRESS festgelegt haben. Danach startet man die Leseprozedur mit dem Aufruf von READ_ EEPROM.

-

dsPIC30F
Da der EEPROM-Speicher eines 24-Bit-PIC (dsPIC30F) breiter als 1 Byte ist, belegt eine EEPROM-Zelle immer 2 Bytes. Die Übertragung erfolgt little endian (Low-Byte zuerst). Die nichtexistierenden ungerade Adressen dieses Typs werden nicht ausgelesen.
Man könnte es auch so sehen, dass der EEPROM eines 24-Bit-PIC ist im Grunde byteweise aufgebaut ist, so dass jedes Byte einer EEPROM-Adresse entspricht, es aber auch ungerade Flash-Adressen gibt.

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x52

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x52
0x01	Anzahl der Bytes	
0x02	Ende-Kennzeichen	0 – Ende
0x03	Byte Nr1	
...		
0xXX	letztes Byte	

7.5.4 READ_IDDATA

Dieser Befehl funktioniert analog zum READ_FLASH-Befehl. Es wird aber der ID-Bereich des Target ausgelesen.
Als Adressen sind die echten ID-Adresssen des PIC zu verwenden (stehen in der Database). Das sind :

- für PIC16Fxxx: 0x2000 ... 0x2003
- für PIC18Fxxxx: 0x200000 ... 0x200007

Da dsPIC30F-Typen keinen ID-Bereich haben, liefern sie einen leeren Datenblock zurück.

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x53

Da die ID immer in einem Block übertragen werden kann, ist das Endekennzeichen eigentlich überflüssig. Es ist aus Gründen der Vereinheitlichung aber vorhanden.

Wie beim Flash erfolgt für 14-Bit-PICs die Übertragung einer ID-Zelle mit 2 Bytes (low-Byte zuerst), auch wenn von den übertragenen 2 Bytes in Wirklichkeit nur die untersten 4 Bit für die ID nutzbar sind.

-

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x53
0x01	Anzahl der Bytes	
0x02	Ende-Kennzeichen	0 – Ende
0x03	Byte Nr1	
...		
0xXX	letztes Byte	

7.5.5 READ_CONFIG

Dieser Befehl funktioniert analog zum READ_FLASH-Befehl. Es wird aber der Config-Bereich des Target ausgelesen.
Als Adressen sind die echten Config-Adresssen des PIC zu verwenden (stehen in der Database). Das sind :

- für PIC16Fxxx: 0x2007 ... 0x2007
- für PIC18Fxxxx: 0x300000 ... 0x30000D
- für dsPIC30Fxxx: 0xF80000 ... 0xF8000D

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x54

Da die Config immer in einem Block übertragen werden kann, ist das Endekennzeichen eigentlich überflüssig. Es ist aus Gründen der Vereinheitlichung aber vorhanden.

PIC16Fxxx
Wie beim Flash erfolgt für 14-Bit-PICs die Übertragung einer Config-Zelle mit 2 Bytes (low-Byte zuerst).

dsPIC30Fxxx
Die 16-Bit langen Config-Worte dieser Typen werden mit jeweils 2 Byte (low Byte zuerst) übertragen. Für die (eigentlich nicht existierenden) ungeraden Adressen wird 0x0000 übertragen.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x54
0x01	Anzahl der Bytes	
0x02	Ende-Kennzeichen	0 – Ende
0x03	Byte Nr1	
...		
0xXX	letztes Byte	

7.5.6 WRITE_FLASH

Mit diesem Kommando lässt sich ein beliebiger Teil des Flash-Programmspeicher des Targets in kurzer Zeit beschreiben.
Bevor man den Programmspeicher mit WRITE_FLASH beschreiben kann, muss man den interessierenden Speicherbereich mit SET_ADRESS festgelegt haben. Danach startet man die Schreibprozedur mit dem Aufruf von WRITE_FLASH.
Wurde mit SET_ADRESS versehentlich eine Start-Adresse hinter der End-Adresse festgelegt, dann verweigert WRITE_FLASH die Arbeit.
Bei der Festlegung des Adressbereichs muss nicht auf eventuelle Block- oder Flash-Zeilengrenzen Rücksicht genommen werden. (im Gegensatz zum Bootloader)

Das an den Brenner8 übertragene Packet besteht aus einem 3 Byte langem Kopf und angehängtem Datenblock. An der Adresse 0x01 steht die Länge dieses Datenblocks in Byte.
An der Adresse 0x02 steht das Endekennzeichen. Dieses ist nötig, da sich mit einem USB-Transfer nur ein Bruchteil der Flash-Daten übertragen lässt. Solange das Endekennzeichen „1“ ist, beendet der Brenner das Brennen nicht, sondern wartet auf weitere WRITE_FLASH-Befehle mit weiteren Datenblöcken. Ist das Endekennzeichen aber „0“, dann beendet der Brenner nach dem Brennen dieses Datenblocks die Brennfunktion.
Sollte das Ende des mit SET_ADRESS definierten Speicherbereichs erreicht sein, ohne dass das Endekennzeichen gesetzt ist, wird das Brennen ebenfalls abgebrochen.

Die Flash-Worte von 14-Bit-PICs werden immer in 2 Bytes übertragen (low Byte zuerst). Folglich ist dann die Byteanzahl (Adresse 0x01) das Doppelte der Flash-Zellenzahl im Datenblock). US-Burn überträgt pro USB-Transfer jeweils 60 Datenbyte.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x60
0x01	Zahl der Bytes	
0x02	Endekennzeichen	0 – letztes Packet
0x03	Byte N. 1	
...		
0xXX	letztes Byte	

Der Brenner8 sendet nach dem Brennen des Blocks nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x60

7.5.7 WRITE_EEPROM

Dieser Befehl funktioniert analog zum Befehl WRITE_FLASH.

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x61
0x01	Zahl der Bytes	
0x02	Endekennzeichen	0 – letztes Packet
0x03	Byte N. 1	
...		
0xXX	letztes Byte	

Der Brenner8 sendet nach dem Brennen des Blocks nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x61

7.5.8 WRITE_IDDATA

Dieser Befehl funktioniert analog zum Befehl WRITE_FLASH.

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x62
0x01	Zahl der Bytes	
0x02	Endekennzeichen	0
0x03	Byte N. 1	
...		
0xXX	letztes Byte	

Der Brenner8 sendet nach dem Brennen des Blocks nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x62

7.5.9 WRITE_CONFIG

Dieser Befehl funktioniert analog zum Befehl WRITE_FLASH.

PC -> Brenner8

Adresse	Bedeutung	Inhalt
0x00	Kommando	0x63
0x01	Zahl der Bytes	
0x02	Endekennzeichen	0
0x03	Byte N. 1	

-

...		
0xXX	letztes Byte	

Der Brenner8 sendet nach dem Brennen des Blocks nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x63

7.5.10 ERASE

Dieser Befehl löst ein Erase des Target aus. Dabei wird der Programmspeicher und EEPROM (und je nach Typ auch Config und ID) gelöscht.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x70

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x70

7.5.11 REMOVECP

Dieser Befehl löst ein Bulk-Erase des Target aus. Dabei wird der Programmspeicher und der EEPROM (je nach Typ auch Config und ID) gelöscht, und ein eventuell aktives Codeprotection aufgehoben.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x71

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x71

7.5.12 SUPPORTED (ab Fw. 0.11 / 3.11)

Dieser Befehl fragt ab, welche PIC-Typen die im Brenner vorhandene firmware unterstützt.

PC -> Brenner8

Adresse	**Bedeutung**	**Inhalt**

-

0x00	Kommando	0x72

Der Brenner8 sendet nur das Kommando als Quittung.

Brenner8 -> PC

Adresse	**Bedeutung**	**Inhalt**
0x00	Kommando	0x71
0x01	Typen	0x00 .. 0xFF

Die Firmware meldet ein 8-Bit-Wort zurück, in dem jedes Bit für eine Gruppe von PICs steht. Eine“1“ bedeutet, das diese Gruppe unterstützt wird, eine „0“ weist auf fehlende Unterstützung für diese Gruppe hin:

Bit 6	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
PIC18FxKx	PIC24x	PIC18FxJx	dsPIC30Fx	PIC18Fx	PIC16Fx	PIC10Fx
	dsPIC33Fx					

-

8 EEPROM des Steuer-PIC

Im EEPROM des Steuer-PIC werden eine Reihe Kalibrierwerte gespeichert.

Außerdem steuert die Zelle 0xFE den Bootloader. Ist ihr Wert 0xFF, das startet beim Reset der Bootloader.

Firmware 0.7

von	bis	Zahl der Bytes	Datentyp	Wert	Standardwerer
0x00	0x07	8	Real	Referenzspannung (Z-Diode)	3,3
0x08	0x0F	8	Real	VppSpannungsteilerverhältnis	3,14
0x10	0x17	8	Real	gain_off	2,6
0x18	0x1F	8	Real	pwm0V_off	16
0x20	0x27	8	Real	gain_on	5,6
0x28	0x2F	8	Real	pwm0V_on	35
0x30	0x30	1	Byte	Prüfsumme von 0x00 bis 0x2F	-
0xFE	0xFE	1	Byte	Bootloaderkennzeichen	0

Die Prüfsumme in Zelle 0x30 sind die unteren 8 Bit der Summe aller Werte aus den Zellen 0x00 bis 0x2F plus die Zahl dieser Zellen (also 0x30).

Firmware 0.8

von	bis	Zahl der Bytes	Datentyp	Wert	Standardwerer
0x00	0x07	8	Real	Referenzspannung (Z-Diode)	3,3
0x08	0x0F	8	Real	VppSpannungsteilerverhältnis	3,14
0x10	0x17	8	Real	gain_off	2,6
0x18	0x1F	8	Real	pwm0V_off	16
0x20	0x27	8	Real	gain_on	5,6
0x28	0x2F	8	Real	pwm0V_on	35
0x30	0x37	8	Real	Vdd wärend Kalibrierung	5
0x38	0x38	1	Byte	Prüfsumme von 0x00 bis 0x37	-
0xFE	0xFE	1	Byte	Bootloaderkennzeichen	0

Die Prüfsumme in Zelle 0x38 sind die unteren 8 Bit der Summe aller Werte aus den Zellen 0x00 bis 0x37 plus die Zahl dieser Zellen (also 0x38).

-

9 Database-Struktur

9.1 Aufbau der picdef03.dat

Die für den Brenner8 relevanten Daten stehen im File **picdef03.dat** der PIC-Database. Diese File besteht aus Datenblöcken vom Typ **TPicDef**.
Die Datensätze liegen in der Datei nicht im packed-Format vor. Dass bedeutet, dass jedes neue Element der Datenstruktur an einer seiner Größe entspreched „geraden“ Adresse beginnt. Dadurch entstehen einige Lücken im Speicherbereich, der den Datensatz enthält.
Die Struktur dieses Typs ist nachfolgend aufgelistet.:

```
TPicDef = record
  name      : string[20]; // z.B. 'PIC18F8722A '
  cpu       : byte;       // 12, 14, 16, 24 Bit
  power     : byte;       // vdd-vpp-Einschaltreihenfolge
  config    : integer;    // eintrag in der config-Datei für erste Config
  software  : byte;       // 1-Flashover, 2-PBrenner, 4-P18, 5-dsProg
  blocksize : integer;    // größe der CP-Blöcke im flash
  pins      : byte;       //Zahl der Anschluß-Pins
  ExtraStr  : string[16];
  ExtraInt  : integer;
  Extrabool : boolean;
  interfaces : record
    io       : byte;
    adc      : byte;
    adctyp   : byte;
    uart     : byte;
    spi      : byte;
    i2c      : byte;
    can      : byte;
    usb      : byte;
    timer    : byte;
    compare  : byte;
    capture  : byte;
    pwm      : byte;
    ccp      : byte;
    eccp     : byte;
    ssp      : byte;
    ex2      : byte;
  end;
  vpp : record            // Programmierspannung in Volt
    min   : real;
    max   : real;
    deflt : real;
  end;
  vdd : record         // Betriebsspannung in Volt
    min     : real;
    max     : real;
    dfltmin : real;
    dfltmax : real;
    nominal : real;
  end;
  pgming : record
```

```
  memtech : byte;    // ee=1 oder eprom=2
  ovrpgm  : byte;    // nur für Eprom  ansonsten Pin-Zahl
  tries   : byte;    // Zyklenzahl für eine Zelle (bei Flash=1)
  lvpthresh : real;
  panelsize : word;  // panel size of zero means single panel
end;
wait : record        // alle Brenn-Zeiten in Mikrosekunden
  pgm     : word;
  lvpgm   : word;
  eedata  : word;
  cfg     : word;
  userid  : word;
  erase   : word;
  lverase : word;
end;
latches : record     // Schreibpuffergrößen in Byte
  pgm      : byte;
  eedata   : byte;
  userid   : byte;
  cfg      : byte;
  rowerase : byte;   // Löschbereich in Byte
end;
pgmmem : record      // Flash-Adressen
  min : longint;
  max : longint;
end;
eedata : record      // EEPROM-Adressen
  min : longint;
  max : longint;
end;
extpgm : record      // externer Speicherschaltkreis
  min : longint;
  max : longint;
  modeaddr : longint;
end;
cfgmem : record
  min : longint;
  max : longint;
end;
calmem : record
  min : longint;
  max : longint;
end;
userid : record
  min : longint;
  max : longint;
end;
devid : record
  min    : longint;
  max    : longint;
  idmask : word;
  id     : word;
end;
taktik : record
  flash  : byte;
  eeprom : byte;
  id     : byte;
  config : byte;
  erase  : byte;
```

```
      cp     : byte;
      read_eeprom : byte;
    end;
  end;
```

Ein kompatibler Datensatz in C++ sieht wie folgt aus (Alle mit fill... bezeichneten elemente dienen nur der Anpassung von packed auf unpacked.):

```
#pragma pack(push, 8)
/* set alignment to 8 -- really important */

//Struktur of the database
typedef struct {
      char name[21];
      __uint8_t cpu;
      __uint8_t power;
      __uint8_t fill1;        //unpacked
      __int32_t config;
      __uint8_t software;
      __uint8_t fill2;        //unpacked
      __uint8_t fill3;        //unpacked
      __uint8_t fill4;        //unpacked
      __int32_t blocksize;
      __uint8_t pins;
      char ExtraStr[17];
      __uint8_t fill5;        //unpacked
      __uint8_t fill6;        //unpacked
      __int32_t ExtraInt;
      __uint8_t ExtraBool;

      struct {
            __uint8_t io, adc, adctyp, uart, spi, i2c, can, usb,
                  timer, compare, capture, pwm, ccp, eccp, ssp, ext;
            __uint8_t fill7;  //unpacked
            __uint8_t fill8;  //unpacked
            __uint8_t fill9;  //unpacked
      } interfaces;

      struct {
            double min, max, deflt;
      } vpp;

      struct {
            double min, max, dfltmin, dfltmax, nominal;
      } vdd;

      struct {
            __uint8_t memtech, ovrpgm, tries;
            __uint8_t fill10; //unpacked
            __uint8_t fill11; //unpacked
            __uint8_t fill12; //unpacked
            __uint8_t fill13; //unpacked
            __uint8_t fill14; //unpacked
```

```
            double lvpthresh;
            __uint16_t panelsize;
            __uint8_t fill15; //unpacked
            __uint8_t fill16; //unpacked
            __uint8_t fill17; //unpacked
            __uint8_t fill18; //unpacked
            __uint8_t fill19; //unpacked
            __uint8_t fill20; //unpacked
      } pgming;

      struct {
            __uint16_t pgm, lvpgm, eedata, cfg, userid, erase, lverase;
      } wait;

      struct {
            unsigned char pgm, eedata, userid, cfg, rowerase;
            __uint8_t fill21; //unpacked
      } latches;

      struct {
            signed long int min, max;
      } pgmmem;

      struct {
            signed long int min, max;
      } eedata;

      struct {
            signed long int min, max, modeaddr;
      } extpgm;

      struct {
            signed long int min, max;
      } cfgmem;

      struct {
            signed long int min, max;
      } calmem;

      struct {
            signed long int min, max;
      } userid;

      struct {
            __uint32_t min, max;
            __uint16_t idmask, id;
      } devid;

      struct {
            __uint8_t flash, eeprom, id, config, erase, cp, read_eeprom;
      } taktik;
      __uint8_t fill21; //unpacked
      __uint8_t fill22; //unpacked
      __uint8_t fill23; //unpacked
      __uint8_t fill24; //unpacked
      __uint8_t fill25; //unpacked

} TPicDef;
```

```
#pragma pack(pop)
```

9.2 Auswahl des richtigen Datensatzes

Für jeden unterstützten PIC gibt es in der Database genau einen solchen Datensatz. Der für das Target relevante Datensatz kann auf zwei Wegen in der Database gefunden werden.

- Der Name des PICs ist im String **name** abgelegt. Im **name**-String werden generell Großbuchstaben verwendet.

- Die Chip-ID des PIC steht in jedem Datensatz im Record **devid**:

Hier noch einmal die Struktur von **devid** aus **TPicDef**.

```
devid : record
  min    : longint;
  max    : longint;
  idmask : word;
  id     : word;
end;
```

- **devid.min** und **devid.max** geben den Adressbereich im PIC an, in dem die ID steht.
- **devid.idmask** gibt an, welche Bits der ID für die Identifikation relevant sind Unwichtige Bits stehen in devid.idmask auf „0“.
- **devid.**id ist der identifikatioscode.

Beispiel: PIC18F2220

- Devid.min : 0x3FFFFE
- Devid.max: 0x3FFFFF
- Devid.idmask 0xFFE0
- Devid.id 0x0580)

Man liest den relevanten Speicherbereich aus:

- 0x3FFFFE = 0x83
- 0x3FFFFF = 0x05

Und erhält als ID

-

- ID = 0x0583

Diese Arbeit wird vom Befehl READ_CHIPID erledigt.

Der ermittelte Wert muss nun mit der **devid.mask** UND-Verknüpft werden. Das Ergebnis ist identisch mit **devid.id**. Folglich ist der PIC ein PIC18F220.

9.3 Datenauswahl

Nicht alle Werte aus der **TPicDef**-Struktur sind für die Firmware des Brenner8 relevant, und so müssen die wichtigen Werte in die **typedef**-Struktur des Kommandos 0x41 kopiert werden. Im folgenden Codebeispiel werden die Daten aus einer Variablen mit Namen **PICdata vom Typ TPicDEF** in eine Variable vom Typ **PICdata2 vom Typ typedef** übertragen. Letztere wird dann mit Kommando 0x41 der Firmware des Brenner8 übermittelt.:

```
  with PICdata2 do begin
    kommando  := SET_PICTYPE;
    cpu       := PICdata.cpu;
    power     := PICdata.power;
    blocksize := PICdata.blocksize;
    pins      := PICdata.pins;
    vpp       := round(PICdata.vpp.deflt*10); // 130;        //13,0V
    if (PICdata.pgming.panelsize<PICdata.pgmmem.max)
      then panelsize := PICdata.pgming.panelsize     //multipanel ?
      else panelsize := 0;                           //singelpanel!!

    with taktik do begin
      flash  := PICdata.taktik.flash;
      eeprom := PICdata.taktik.eeprom;
      id     := PICdata.taktik.id;
      config := PICdata.taktik.config;
      erase  := PICdata.taktik.erase;
      cp     := PICdata.taktik.cp;
      read_eeprom :=PICdata.taktik.read_eeprom;
    end;
    with latches do begin     // Schreibpuffergrößen in Byte
      pgm      := PICdata.latches.pgm;
      eedata   := PICdata.latches.eedata;
      userid   := PICdata.latches.userid;
      cfg      := PICdata.latches.cfg;
      rowerase := PICdata.latches.rowerase
    end;
    with wait do begin        // alle Brenn-Zeiten in Mikrosekunden
      pgm     := PICdata.wait.pgm;
      lvpgm   := PICdata.wait.lvpgm;
      eedata  := PICdata.wait.eedata;
      cfg     := PICdata.wait.cfg;
      userid  := PICdata.wait.userid;
      erase   := PICdata.wait.erase;
      lverase := PICdata.wait.lverase;
    end;
  end;
  // PATCH für DB10
  if PICdata.name='PIC18F1320' then PICdata2.wait.eedata:=6000;
  if PICdata.name='PIC18F1220' then PICdata2.wait.eedata:=6000;
```

Die letzten beiden Zeilen dienen nur der Korrektur zweier Fehler in der Database 10 und sind zukünftig nicht mehr relevant.

-