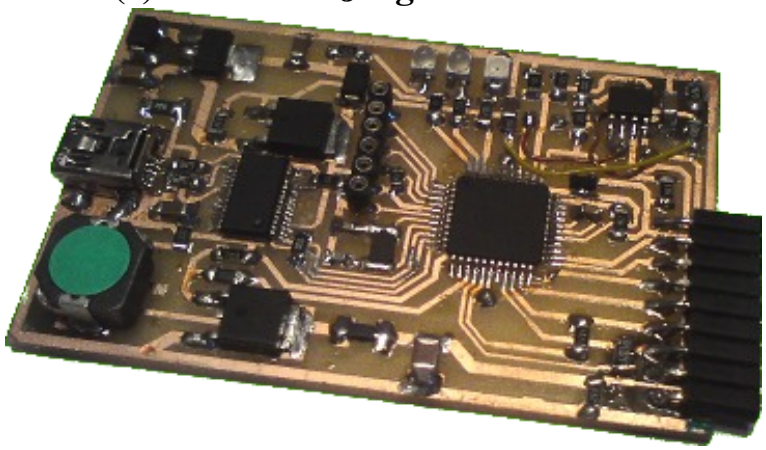
UPROG2: Universeller Programmer für Linux

V1.40 (c) 2017-2021 Jörg Wolfram



[0.5cm]

[0.5cm]

1 Hinweis und Lizenz

Das Programm unterliegt der GPL (GNU General Public Licence) Version 3 oder höher, jede Nutzung der Software/Informationen nonkonform zur GPL oder ausserhalb des Geltungsbereiches der GPL ist untersagt!

Die Veröffentlichung dieses Projekts erfolgt in der Hoffnung, daß es Ihnen von Nutzen sein wird, aber OHNE IRGENDEINE GARANTIE, auch ohne die implizite Garantie der MARKTREIFE oder der VERWENDBARKEIT FÜR EINEN BESTIMMTEN ZWECK.

Alle im Text genannten Marken sind Eigentum des entsprechenden Inhabers.

Die Software wurde unter und für Linux entwickelt, Unterstützung für andere Betriebssysteme ist nicht vorgesehen.

2 Allgemeines

Angefangen hat es vor ein paar Jahre damit, dass es keine gescheite Software für den TBDML (BDM-Programmer für Freescale/NXP S12(x)) gab, die auch die S12XD und S12XE unterstützt. Nach einigen (vergeblichen) Versuchen, die vorhandene Software anzupassen, habe ich mich letztendlich dazu entschlossen mir einen eigenen Programmer zu bauen. Inzwischen sind einige andere Controllerfamilien dazugekommen, irgendwann war der Mega168 zu klein und ich habe ein komplettes Re-Design mit einem Mega644 gemacht.
Und da ich wohl nicht er Einzige bin der sich mit deratigen Problemen rumschlägt, habe ich mich entschlossen, das Projekt zu dokumentieren und zu veröffentlichen.

2.1 Unterstützte Controller- und Speicherfamilien

Die Liste der unterstützten Devices ist fest in das Programm integriert und schon recht lang, zusätzliche Devices könen nur durch Neukompilation hinzugefügt werden. Derzeit können Devices aus folgenden Familien programmiert werden:

- Atmel AVR (SPI)
- Atmel AVR0, AVR1 (UPDI)
- Atmel ATxmega (PDI)
- Atmel AT89S8252

- Cypress PSOC4 (SWD)
- Microchip PIC10xx/PIC12xx/PIC16xx
- Microchip PIC18xx
- Microchip dsPIC33xx
- NXP/Freescale MPC56xx (BAM)
- NXP/Freescale HCS08
- NXP/Freescale HCS12(X)
- NXP/Freescale S12Z
- NXP/Freescale S32K
- NXP/Freescale K9EA
- NXP/Freescale MPC574x
- Renesas R8C
- Renesas 78K0R
- Renesas RL78
- Renesas V850
- Renesas RH850
- ST Micro SPC56xx (BAM)
- ST Micro SPC56xx, SPC58xx (JTAG)
- ST Micro ST7FLITE
- ST Micro STM8 (SWIM)
- ST Micro STM32 (SWD)
- TI MSP430 (SBW)
- TI CC2540, CC2541
- TI CC2640
- Silabs EFM32/EFR32
- XILINX XC9500/XL
- Atmel Dataflash (AT45DBxx)
- SPI-Flash (25xx)
- SPI-EEPROM (250xx)
- I2C-EEPROM (24xx)
- ONEWIRE-EEPROM (DS28E07)
- Magnetsensoren (Melexis, Infineon)

Wenn man **uprog2 LIST** aufruft, werden die aktuelle unterstützten Devices ausgegeben. Geplant sind derzeit noch dsPIC30, LPCxxxx von NXP, Freescale ColdFire und TI C2000 (TMS32F28xx). Wann ich letztendlich dazukomme, steht allerdings in den Sternen. Und natürlich die Erweiterung der vorhandenen Listen um weitere Devices.

2.2 Konzept

Das Steuerprogramm auf dem PC ist ein einzelnes Binary, es werden keine witeren Dateien benötigt/angelegt. Auf dem System müssen libbluetooth und libftdi (bzw. libftdi1) vorhanden sein.
Das mit der fest eincompilierten Typenliste ist zwar erstmal etwas unflexibel, für mich aber völlig ausreichend. Beim Start prüft das Programm ob der **uprog2d**-Dämon schon läuft, ansonsten wird der Prozess geforkt. Die Kommunikation mit dem Dämon findet über Shared Memory statt, Auf die gleiche Weise könnte dann auch ein Debugger über Programmer mit seinem Device kommunizieren.
Updates werden automatisch auf den Programmer übertragen, wenn die PC-Software eine veraltete Version feststellt.

2.3 Installation

2.3.1 Auf dem PC

Momentan werden Intel 32-bit und x86-64 unterstützt. Je nach System sollte **uprog-32** oder **uprog2-64** nach **/usr/local/uprog2** kopiert werden, das war es dann auch schon. Ausserdem müssen **libftdi** und **libbluetooth** auf dem Rechner installiert sein.

2.3.2 Auf dem AVR

Zuerst sollten die Fuses prorammiert werden. Für den Mega644 ist folgende Einstellung vorgesehen (**Änderung ab Version 1.32**):

- LOW FUSE: 0xe6
- HIGH FUSE: 0xd0
- EXT FUSE: 0xff

Danach muss das entsprechende Hexfile (main-bt.hex, main-usb.hex) programmiert werden. Die Bluetooth-Variante führt beim erstmaligen Start eine Initialisation des Bluetooth-Modules durch, danach muss das Pairing durchgeführt werden. Als Pin ist 55309 fest im Programm codiert. Soll diese geändert werden, muß das AVR-Programm anschließend neu übersetzt werden.

2.4 Compilieren und Erweitern

Eine Erweiterung auf zusätzliche Controller ist derzeit nur durch eine Neucompilation möglich. Soweit möglich, werde ich entsprechende Anfragen in zukünftigen Softwareversionen berücksichtigen.
Bei vielen Controllerfamilien lassen sich als „Workaround" ähnliche Typen finden. Wenn dabei die ID-Abfrage zu einem Abbruch führt, lässt sich diese meist auch über das Kommando „ii" (ignore ID) entschärfen.
Für das Übersetzen der Firmware benötigt man AVRA, für die PC-Seite ist eine Standard-Toolchain mit make und gcc ausreichend. Für die Übersetzung der EXEC-Files (z.B. PowerPC und STM32) wird dafür auch noch die entsprechende Binutils-Variante benötigt.

2.5 Änderung der PID beim FTDI232RL

Ab Version 1.35 nutzt der Programmer eine neue PID, nämlich 0x6661 anstelle 0x6001. Das hat hauptsächlich den Grund, dass man einen USB-Seriell-Wandler mit einem zweiten FT232RL parallel zum Programmer betreiben kann. Beim Testen von Schaltungen war es einfach umständlich, jedes Mal das terminal zu schließen, USB umstecken, programmieren, USB wieder umstecken, Terminal neu starten...
Um den UPROG2 umzuprogrammieren, sollte nur der Programmer am PC amgesteckt sein. Außerdem werden die libftdi-Tools benötigt. Nun muss nur noch das Script **prog_ftdi** im **sys**-Verzeichnis gestartet werden. Mit dem Script **install_udev_rule** wird dann noch die passende UDEV-Rule installiert, damit man als normaler User auf den Programmer zugreifen kann.

2.6 Benutzung des Programmes

2.6.1 Aufruf über Kommandozeile

Die Benutzung über die Kommandozeile erfolgt nach einem festen Schema und ist dafür ausgelegt, möglichst einfach in Makefiles integriert zu werden. Das erste Argument ist immer der Controller- oder Speichertyp. Danach folgen die Kommandos mit einem vorangestellten Minuszeichen. Zwischen den einzelnen Kommandos sind keine Leerzeichen etc. zugelassen. Je nach auszuführenden Kommandos muss noch ein dritter Parameter folgen. Dies ist meist ein Dateiname oder ein numerischer Wert. Die möglichen Kommandos können mittels **uprog2 CONTROLLERTYP -help** angezeigt werden.

```
>uprog2 ATMEGA328P -help

################################################################################
#                                                                              #
#  UNI-Programmer UPROG2 V1.40                                                 #
#                                                                              #
#  (c) 2012-2021 Joerg Wolfram                                                 #
#                                                                              #
#  usage: uprog2 device -commands [up to 4 files/data]                         #
#         uprog2 KILL                          kill active daemon              #
#         uprog2 LIST                          for device list                 #
#         uprog2 device -help                  for device specific commands    #
#                                                                              #
#   xxx types in database                                                      #
#                                                                              #
################################################################################
>> USB programmer is active.

*** ATMEGA328P can be programmed with: (1=VSS  2=VDD  3=RST  4=SCK  5=MISO  6=MOSI) ***

Version =  1.6
Sysver  =  0013
V-Ext   =  0.0V
V-PROG  =  4.9V
-- 5v -- set VDD to 5V
-- ls -- low spi speed
-- ea -- chip erase
-- pm -- main flash program
-- vm -- main flash verify
-- rm -- main flash readout
-- pe -- eeprom program
-- ve -- eeprom verify
-- re -- eeprom readout
-- lf -- set low fuse
-- hf -- set high fuse
-- ef -- set ext fuse
-- lb -- set lock bits
-- st -- start device
-- ii -- ignore wrong ID
-- d2 -- switch to device 2
```

Soweit möglich, sind die Kommandos für alle Typen einheitlich. Wenn sich Kommandos gegenseitig ausschließen (z.B. Verify und Readout), wird das Kommando, welches Dateien überschreiben würde, deaktiviert.

2.6.2 Ein Beispiel

Will man z.B. einen ATMega328 mit der Datei main.hex flashen mit anschließendem Verify, würde die Kommendozeile folgendermaßen lauten: **uprog2 ATMEGA328P -eapmvm main.hex**

```
>uprog2 ATMEGA328P -eapmvm main.hex

################################################################################
#                                                                              #
#  UNI-Programmer UPROG2 V1.40                                                 #
#                                                                              #
#  (c) 2012-2021 Joerg Wolfram                                                 #
#                                                                              #
#  usage: uprog2 device -commands [up to 4 files/data]                         #
#         uprog2 KILL                          kill active daemon              #
#         uprog2 LIST                          for device list                 #
#         uprog2 device -help                  for device specific commands    #
#                                                                              #
#   xxx types in database                                                      #
#                                                                              #
################################################################################
>> USB programmer is active.

*** ATMEGA328P can be programmed with: (1=VSS  2=VDD  3=RST  4=SCK  5=MISO  6=MOSI) ***

Version =  1.6
Sysver  =  0013
V-Ext   =  0.0V
V-PROG  =  4.9V
## using high speed spi mode
## Action: chip erase
## Action: flash program using main.hex
## Action: flash verify using main.hex

INIT
READ ID
SIGNATURE  = 1E 95 0F
FUSE LOW       = 0xE6
FUSE HIGH      = 0xD0
FUSE EXT       = 0xFD
Lock Bits      = 0xFF
Calibration    = 0x93
CHIP ERASE
MAIN PROG    |************************************************|
MAIN READ    |************************************************|

OK
```

Bei länger dauernden Operationen wird, soweit möglich, ein Fortschrittsbalken angezeigt.

2.6.3 Zwei Devices programmieren

Seit Version 1.30 können an den UPROG2 zwei Devices angeschlossen werden. Ausnahme sind die PIC-Controller, da dafür die Programmierspannung „missbraucht" wird. Mit einem 12V-Umschaltrelais (sofern es nicht allzuviel Strom braucht) an Pin 9 und der Opion **d2** kann das Programmierinterface temporär auf ein zweites Device umschaltet werden. Ist **d2** im Programmierkommando enthalten, wird VPP auf 15V eingestellt und an Pin 9 aktiviert. Die Spannung bricht zwar zusammen, hat aber bei mir für ein 4-fach Umschaltrelais (NAiS DS4E-M-DC12V) ausgereicht.
Auf diese Weise kann man z.B. zwei Controller auf einem Board „in einem Rutsch" programmieren, ohne umstecken zu müssen. Für diese Funktion muss allerdings der Bootloader (>=V1.4) vorhanden sein oder mit einem externen Tool neu geflasht werden. Andernfalls lässt sich diese Funktion nicht nutzen.

3 Debug-Funktionen für ARM Cortex-M

Ab Version 1.39 sind für verschiedene Controller-Familien mit Cortex-M Kernen und SWD-Interface rudimentäre Debug-Funktionen vorhanden. Damit könnnen Programme im Flash oder RAM schrittweise abgearbeitet, Breakpoints gesetzt und Registerinhalte sowie Speicherzellen angezeigt und auch modifiziert werden. Ein Debugging auf Hochsprachenlevel ist damit nicht möglich, könnte aber später durch eine Anbindung z.B. an OpenOCD realisiert werden.

Aktuell werden folgende Controllerfamilien unterstützt:

- NXP/Freescale S32K
- NXP/Freescale K9EA
- ST Micro STM32
- Silabs EFM32/EFR32

Die Debug-Funktionen erreicht man mit den Kommandos **df** (debug in flash) und **dr** (debug in RAM). Beim Debuugen im Flash werden Startadresse und Stackpointer automatisch aus dem Flash ermittelt, für das Debuggen aus dem RAM muss wie für das **rr** Kommando ein geeignetes Hexfile angegeben werden. Nach der Ausführung des Kommandos erhält man einen Debug-Prompt:

```
DBG>
```

Gibt man hier nur eine leere Zeile ein (nur ENTER), so werden die verfügbaren Befehle aufgelistet:

```
q            : Quit
s            : Step
c            : Continue
g addr       : Go
b addr       : Set breakpoint and continue
r0= data     : Set processor register 0
r15= data    : Set processor register 15 (PC)
rb addr      : Read memory (16 bytes from addr)
rw addr      : Read memory (8 words from addr)
rl addr      : Read memory (4 longs from addr)
wb addr data : Write memory (1 byte)
ww addr data : Write memory (1 word)
wl addr data : Write memory (1 long)

DBG>
```

Nach jedem Kommando außer Lesen und Schreiben im Speicher/IO wird der aktuelle Inhalt der Prozessor-Register angezeigt. Neben dem Wert für den Programm-Counter stehen auch noch die nächsten 3 Worte im Speicher, auf die das PC Register zeigt. Beispiel:

```
START CODE AT 0x0800c070
R0 : 00000000    R1 : 00000000    R2 : 00000000    R3 : 00000000
R4 : 00000000    R5 : 00000000    R6 : 00000000    R7 : 00000000
R8 : 00000000    R9 : 00000000    R10: 00000000    R11: 00000000    R12: 00000000
SP : 10010000
LR : FFFFFFFF
PC : 0800C070 --> 4B10 2200 4293

DBG>s
R0 : 00000000    R1 : 00000000    R2 : 00000000    R3 : 0800C870
R4 : 00000000    R5 : 00000000    R6 : 00000000    R7 : 00000000
```

```
R8 : 00000000    R9 : 00000000    R10: 00000000    R11: 00000000    R12: 00000000
SP : 10010000
LR : FFFFFFFF
PC : 0800C072 --> 2200 4293 D008
```

Für die Parameter **addr** und **data** werden Hexadezimalwerte erwartet, Groß- und Kleinschreibung wird dabei ignoriert. Mit den **r0=** bis **r15=** lassen sich Registerwerte ändern:

```
DBG>r6= 1234
R0 : 00000000    R1 : 00000000    R2 : 00000000    R3 : 0800C870
R4 : 00000000    R5 : 00000000    R6 : 00001234    R7 : 00000000
R8 : 00000000    R9 : 00000000    R10: 00000000    R11: 00000000    R12: 00000000
SP : 10010000
LR : FFFFFFFF
PC : 0800C072 --> 2200 4293 D008

DBG>
```

Bei den Leseoperationen werden z.Zt. immer 16 Bytes ab der angegebenen Adresse ausgelesen und angezeit, die Formatierung bei der Anzeige ist jedoch abhängig vom Kommando:

```
DBG>rb 8000000
0x08000000  :  00 00 01 10 71 C0 00 08 AD C1 00 08 AD C1 00 08
DBG>rw 8000000
0x08000000  :  0000 1001 C071 0800 C1AD 0800 C1AD 0800
DBG>rl 8000000
0x08000000  :  10010000 0800C071 0800C1AD 0800C1AD
DBG>
```

Bei den Schreiboperationen >1 Byte (Word, Long) werden die Adressen passend gerundet, wenn der eingegebene Wert für data den Bereich für die zu schreibende Bitbreite überschreitet, werden die ”überhängenden” Bits ignoriert.

```
DBG>rl 20000000
0x20000000  :  544CABB1 F612E6C4 66DDE632 E6F542DB
DBG>wl 20000000 0
DBG>rl 20000000
0x20000000  :  00000000 F612E6C4 66DDE632 E6F542DB
DBG>wb 20000001 1234
DBG>rl 20000000
0x20000000  :  00003400 F612E6C4 66DDE632 E6F542DB
DBG>
```

Nach den Go/Continue/Breakpoint Befehlen läuft der Prozessor weiter und hält ggf. beim Breakpoint an. Mit der Enter-Taste (Go, Continue) sowie der ESC-Taste (Breakpoint) wird der Prozessor angehalten und wieder in den Debug-Mode gewechselt. Das ist z.B. sinnvoll, falls der eingestellte Breakpoint nicht erreicht wird.

4 Debug-Funktionen für HCS08

Ab Version 1.40 sind für HCS08 Controllern rudimentäre Debug-Funktionen vorhanden. Damit könnnen Programme im Flash oder RAM schrittweise abgearbeitet, Breakpoints gesetzt und Registerinhalte sowie Speicherzellen angezeigt und auch modifiziert werden. Ein Debugging auf Hochsprachenlevel ist damit nicht möglich, könnte aber später durch eine Anbindung z.B. an OpenOCD realisiert werden.

Die Debug-Funktionen erreicht man mit den Kommandos **df** (debug in flash) und **dr** (debug in RAM). Beim Debuugen im Flash werden Startadresse und Stackpointer automatisch aus dem Flash ermittelt, für das Debuggen aus dem RAM muss wie für das **rr** Kommando ein geeignetes Hexfile angegeben werden. Nach der Ausführung des Kommandos erhält man einen Debug-Prompt:

```
DBG>
```

Gibt man hier nur eine leere Zeile ein (nur ENTER), so werden die verfügbaren Befehle aufgelistet:

```
q            : Quit
s            : Step
c            : Continue
g addr       : Go
b addr       : Set breakpoint and continue
A= data      : Set processor register A
HX= data     : Set processor register HX
SP= data     : Set processor stack pointer
rb addr      : Read memory (16 bytes from addr)
rw addr      : Read memory (8 words from addr)
wb addr data : Write memory (1 byte)
ww addr data : Write memory (1 word)

DBG>
```

Nach jedem Kommando außer Lesen und Schreiben im Speicher/IO wird der aktuelle Inhalt der Prozessor-Register angezeigt. Neben dem Wert für den Programm-Counter stehen auch noch die nächsten 3 Worte im Speicher, auf die das PC Register zeigt. Beispiel:

```
START CODE AT 0xC000
A : 00          CCR: 68 (v**hInzc)
HX: FF FF       SP : 00FF
PC: C000  -> 45 04 80 94

DBG>s
A : 00          CCR: 68 (v**hInzc)
HX: 04 80       SP : 00FF
PC: C003  -> 94 CD C2 AE

DBG>
```

Für die Parameter **addr** und **data** werden Hexadezimalwerte erwartet, Groß- und Kleinschreibung wird dabei ignoriert. Mit den **r0=** bis **r15=** lassen sich Registerwerte ändern:

```
DBG>HX= 1234
A : 00          CCR: 68 (v**hInzc)
HX: 12 34       SP : 00FF
PC: C003  -> 94 CD C2 AE

DBG>
```

Bei den Leseoperationen werden z.Zt. immer 16 Bytes ab der angegebenen Adresse ausgelesen und angezeit, die Formatierung bei der Anzeige ist jedoch abhängig vom Kommando:

```
DBG>rb c000
0xC000  :  45 04 80 94 CD C2 AE 27 03 CC C0 7A 45 00 00 65
DBG>rw c000
0xC000  :  0445 9480 C2CD 27AE
DBG>
```

Bei den Schreiboperationen >1 Byte (Word) werden die Adressen passend gerundet, wenn der eingegebene Wert für data den Bereich für die zu schreibende Bitbreite überschreitet, werden die ”überhängenden” Bits ignoriert. Im nachfolgenden Beispiel wrden zuerst Byte 5 (PTCDD) und Byte 4 (PTCD) mit 0x0C beschrieben, danach sollten PTC2 und PTC3 HIGH-Pegel haben.

```
DBG>rb 0
0x0000  :  00 00 00 00 00 00 07 00 00 00 00 00 00 00 00 00
DBG>wb 5 0c
DBG>wb 4 0c
DBG>rb 0
0x0000  :  00 00 00 00 0C 0C 07 00 00 00 00 00 00 00 00 00
DBG>
```

Nach den Go/Continue Befehlen läuft der Prozessor weiter und kann mit der Enter-Taste wieder angehalten werden. Im Breakpoint-Modus lässt sich der Prozessor mittels der ESC-Taste anhalten und in den Debug-Mode wechseln. Das ist z.B. sinnvoll, falls der eingestellte Breakpoint nicht erreicht wird. Im Go- und Continue-Modus ist der Berakpoint nicht aktiv.

5 Changelog

12.6.2021 Version 1.40

- System-Version ist jetzt 0018, Update erfolgt automatisch
- Feature: Debug-Funktionen für HCS08 Controller
- Feature: Checksummen-Berechnung für die RL78 Familie
- Neue Devices: ATTiny 1xxx
- Neue Devices: ATMEGA über JTAG
- Erstmals Binary für Raspberry Pi
- Beschleunigung der Datenübertragung beim Auslesen
- Feature: Margin Check / Dataflash für S32K Familie
- Bugfix: Program->Verify für S32K Familie las erste 16 Bytes falsch
- Bugfix: Programmierung S32K: FSEC etc. wurde vorher nicht gelöscht

12.1.2021 Version 1.39

- System-Version ist jetzt 0017, Update erfolgt automatisch
- Feature: Debug-Funktionen für ARM Cortex basierte Controller
- Feature: Dump-Funktionen für die RL78 Familie
- Feature: ID-Funktionen für die RH870 Familie
- Neue Devices: STM32F7xx
- Neues Device: DS28E07
- Modellpflege: Neue Devices beim R8C
- Bugfix: Beim STM32F4xx blieb das Erase manchmal in einer Endlosschleife
- Bugfix: falsche Maske für Option-Bytes beim STM32F4xx
- diverse kleinere Bugfixes

26.10.2020 Version 1.38

- System-Version ist jetzt 0016, Update erfolgt automatisch
- Bugfix: Bei S08 und S12X wurde die BDM-Frequenz zu niedrig erkannt (nur ca. 1/3)
- Bugfix: Dateiname bei make install korrigiert

28.9.2020 Version 1.37

- System-Version ist jetzt 0015, Update erfolgt automatisch
- Die 32-Bit Version (X86 Binary) habe ich eingestellt
- Anpassungen an GCC10
- Neue Devices: NXP/Freescale S12Z
- Neue Devices: NXP/Freescale MPC574x
- Neue Devices: STM SPC584cc
- Neue Devices: Silabs EFM32/EFR32

- diverse kleinere Bugfixes

27.1.2020 Version 1.36

- System-Version ist jetzt 0014, Update erfolgt automatisch
- Bugfix: Programmierung XC95xxXL funktionerte nicht richtig
- Feature: Bootstrapping beim RH850
- Feature: Config lesen/schreiben beim SPI Flash
- Neue Devices: AT89S8252
- Modellpflege: neue Devices beim RH850
- Modellpflege: neue Devices beim SPI-Flash, Änderung der Standardtypen

25.10.2019 Version 1.35

- Loader-Version ist jetzt 1.6, das gesamte Image muss neu programmiert werden
- System-Version ist jetzt 0013
- **Wichtig: Änderung der USB PID auf 0x6661**
- externe Versorgung wird jetzt erst ab 1,5V erkannt
- Bugfix: Beim RH850 wurde der Data-Flash nicht komlett beschrieben
- Feature: Es können beim Programmieren/Verify bis zu 4 Dateien angegeben werden
- Feature: Security beim RH850 lesen/schreiben
- Neue Devices: AVR0 mit UPDI
- Neue Devices: TLE5014
- Modellpflege: neue Devices und Funktionen bei den SPI-Flashes

7.4.2019 Version 1.33

- System-Version ist jetzt 0012, Update erfolgt automatisch
- Bugfix: langsame SPI-Geschwindigkeit beim AVR war zu hoch
- Bugfix: beim R8C wurde Code>2K nicht im RAM gestartet
- Bugfix: Beim MSP430 POR vor Code-Transfer
- Bugfix: Beim HCS08 falscher Flash-bereich bei den 60K Varianten
- Feature: Programmierung über JTAG jetzt auch beim SPC56ELxx
- Feature: Option Bytes beim RH850 lesen/schreiben
- Modellpflege: neue Devices beim RL78

25.1.2019 Version 1.32

- Loader-Version ist jetzt 1.4, das gesamte Image muss neu programmiert werden
- System-Version ist jetzt 0011, Update erfolgt automatisch
- Verkleinerte Bootloader-Sektion, Fuse muss neu programmiert weden
- Bugfix: Prorammierung alter AVRs ohne Ready/Busy Bit
- Neue Devices: NXP K9EA

- Neue Devices: PIC16(L)F153xx
- Neue Devices: RH850/FK1M-S1

25.10.2018 Version 1.31

- System-Version ist jetzt 0010, Update erfolgt automatisch
- Bugfix: Quad-Program bei SPI-Flashes funktionierte teilweise nicht richtig
- WebGui-Funktion entfernt
- Neue Devices: NXP S32K

23.04.2018 Version 1.30

- Loader-Version ist jetzt 1.4, das gesamte Image muss neu programmiert werden
- System-Version ist jetzt 0009, Update erfolgt automatisch
- Bugfix: EEPROM-Programierung beim AVR ließ Bytes aus
- Bugfix: Binärpage-Erkennung bei den Dataflashes funktionierte nicht richtig
- Bugfix: Zu kleine RAM-Größen beim STM32F4xx, es wurden bei -rr nur max. 4K übertragen
- Neue Funktion, Umschaltung auf zweites Device
- Neue Devices: TI CC2640
- Neue Devices: STM32L4xx
- Neue Devices: PIC18F2xxx/PIC18F4xxx

17.12.2017 Version 1.29

- System-Version ist jetzt 0008, Update erfolgt automatisch
- Bugfix: Dataflash-Startadressen beim S12XE angepasst
- Trimm-Funktion für den internen Oszillator beim HCS08
- Neue Devices: Renesas V850/RH850
- Neues Device: Drucksensor LPS25H

5.10.2017 Version 1.28

- System-Version ist jetzt 0007, Update erfolgt automatisch
- Bugfix: Leerblock-Erkennung bei PIC16xxx entfernt (wird sequentiell programmmiert)
- MSP430F5xx fehlende Funktionalität beim Löschen/Schreiben integriert
- MSP430 Flash löschen mit/ohne INFO-A
- Single-Core SPC56xx können jetzt auch über JTAG programmiert werden
- SPC56xx Programmierung über BAM hat jetzt andere Typbezeichhnungen (SPC56XX-BL)
- Unterstützung für SPI-EEPROMs (AT25010...AT25640)
- Neue Funktion: Frequenzgenerator
- Neue Funktion: Web-Interface

25.2.2017 Version 1.26

- Anzeige über Debug-LEDs beim SPC56xx deaktiviert
- Bugfix: Pageoffset bei SPI-Flashes stimmte nicht
- Bugfix: beim MSP430 wurden falsche Programmerpins angesteuert
- Bugfix: Erase Dataflash beim RL78 brach mit Fehler ab, obwohl alles OK war
- 64MB SPI-Flash und Quad-Mode hinzugefügt
- MLX90363 readout (EEPROM write noch nicht implementiert

8.11.2016 Version 1.25a

- DOC: Fehler in der Controllerbeschaltung behoben (10K-Widerstand an PA3)

7.11.2016 Version 1.25

- Initiale Version