

**halec**  
Herrnröther Str. 54  
63303 Dreieich  
Germany

[www.halec.de](http://www.halec.de)



# Handbuch

## roloFlash 2



Dokumentenversion 1.0.28 vom 2019-02-12  
(Stand der Software: 02.AA)

Copyright © 2009-2019 halec. Alle Marken, Logos und Bilder sind Eigentum der jeweiligen Hersteller bzw. Urheber. Änderungen und Irrtümer vorbehalten.

# Inhaltsverzeichnis

I	Vorwort.....	1
II	Verpackungsinhalt.....	3
III	Beschreibung.....	4
1	Programmier-Buchse.....	4
2	Pin-Belegung allgemein.....	4
3	Pullup- / Pulldown-Widerstände.....	5
4	Spannungsbereich.....	6
5	Elektrische Schutzmaßnahmen.....	6
6	Pin-Belegung JTAG-Interface.....	7
7	Pin-Belegung Atmel ISP-Interface.....	8
8	Pin-Belegung Atmel PDI-Interface.....	9
9	Pin-Belegung Atmel UPDI-Interface.....	10
10	Pin-Belegung UART-Interface 0.....	11
11	Pin-Belegung UART-Interface 1.....	11
12	Pin-Belegung GPIO-Interface.....	12
13	LEDs.....	13
14	microSD-Kartenslot.....	13
15	Vorbereitung der microSD-Karte am PC.....	14
16	Flashen der Targetboards.....	15
IV	Aktualisieren von roloFlash.....	16
V	Liste der mitgelieferten roloBasic-Skripte.....	19
VI	roloFlash-API (Liste der Prozeduren und Funktionen).....	22
1	Interne Datenbank.....	23
1.1	DB_getHandle.....	23
1.2	DB_get.....	24
2	Öffnen und Schließen von Bussen und Targets.....	25
2.1	Bus öffnen.....	26
2.2	Bus schließen.....	27
2.3	Target öffnen.....	28
2.4	Target schließen.....	29
3	JTAG-Bus.....	30
3.1	Bus öffnen.....	30
3.2	Busgeschwindigkeit ändern.....	33
3.3	Busgeschwindigkeit abfragen.....	34
3.4	JTAG-Chain scannen.....	35
4	Atmel ISP-Bus.....	36
4.1	Bus öffnen.....	36
4.2	Busgeschwindigkeit ändern.....	39
4.3	Busgeschwindigkeit abfragen.....	40
4.4	Reset-Mode einstellen.....	41
5	Atmel PDI-Bus.....	42

5.1	Bus öffnen.....	42
5.2	Busgeschwindigkeit ändern.....	44
5.3	Busgeschwindigkeit abfragen.....	45
6	Atmel UPDI-Bus.....	46
6.1	Bus öffnen.....	46
6.2	Busgeschwindigkeit ändern.....	48
6.3	Busgeschwindigkeit abfragen.....	50
7	UART-Interface.....	50
7.1	Bus öffnen.....	50
7.2	Baudrate ändern.....	56
7.3	Baudrate abfragen.....	57
7.4	Schreiben.....	58
7.5	Lesen.....	59
8	GPIO-Interface.....	59
8.1	Bus öffnen.....	59
8.2	Mode ändern.....	61
8.3	Schreiben.....	62
8.4	Lesen.....	63
9	Target allgemein.....	64
9.1	getTargetPresent.....	64
9.2	setTargetMode.....	66
9.3	restartTarget.....	68
10	Target-Memorymap lesen/schreiben.....	71
10.1	Wert in ein Property einer Speicherart schreiben.....	71
10.2	Wert aus einem Property einer Speicherart lesen.....	73
10.3	clearMemoryLayout.....	74
11	Loader.....	75
12	Target löschen, schreiben, lesen und verifizieren.....	76
12.1	EraseFlash.....	76
12.2	writeFileToTarget.....	77
12.3	readFileFromTarget.....	79
12.4	writeRawDataToTarget.....	81
12.5	readRawDataFromTarget.....	82
13	Target STM32.....	84
14	Target Atmel AVR (ISP-Interface).....	85
14.1	getSignature.....	85
14.2	readBits.....	86
14.3	writeBits.....	87
14.4	setExtendedAddressMode.....	89
15	Target Atmel AVR (PDI-Interface).....	90
15.1	getDeviceId.....	90

15.2	readBits.....	91
15.3	writeBits.....	92
16	Target Atmel UPDI (UPDI-Interface).....	94
16.1	getDeviceId.....	94
16.2	readBits.....	95
16.3	writeBits.....	96
17	Dateien.....	98
17.1	fsCreate.....	99
17.2	fsRemove.....	99
17.3	fsMkDir.....	100
17.4	fsFileExists.....	101
17.5	fsFileSize.....	102
17.6	fsOpen.....	103
17.7	fsRead.....	103
17.8	fsWrite.....	104
17.9	fsTruncate.....	105
17.10	fsClose.....	106
17.11	fsSync.....	107
18	LEDs.....	108
18.1	ledOn.....	108
18.2	ledOff.....	109
18.3	ledBlink.....	110
18.4	ledRunningLight.....	110
18.5	ledRunningLightOutstanding.....	111
19	Sonstige.....	112
19.1	setLogMode.....	112
19.2	print.....	114
19.3	delay.....	115
19.4	getSystemTime.....	116
19.5	getTargetBoardVoltage.....	116
19.6	setCpuClock.....	117
19.7	getCpuClock.....	118
VII	Abfrage von roloFlash-Eigenschaften.....	119
1	Versionsnummern etc.....	119
2	Seriennummer.....	119
VIII	Exceptions.....	121
1	Exceptions des roloBasic.....	121
2	Exceptions des Dateisystems.....	122
3	Exceptions des roloFlash.....	124
4	Vom Benutzer ausgelöste Exceptions.....	127
IX	Bedeutungen von LED-Codes.....	128

1	Normaler Betrieb.....	128
1.1	Keine microSD-Karte gefunden.....	128
1.2	Exception aufgetreten.....	128
2	roloFlash-Aktualisierung.....	129
2.1	Wartet auf microSD-Karte für Aktualisierung.....	129
2.2	Aktualisierung läuft.....	130
2.3	Aktualisierung mit Erfolg abgeschlossen.....	130
2.4	Aktualisierung fehlerhaft: Dateifehler.....	130
2.5	Aktualisierung fehlerhaft: Datei nicht gefunden.....	131
2.6	Aktualisierung fehlerhaft: Mehrere Dateien gefunden.....	131
2.7	Aktualisierung fehlerhaft: Sonstiges.....	132
X	Spezifikation.....	133
1	Unterstützte Controller von ST Microelectronics.....	133
1.1	STM32F1.....	133
1.2	STM32F2.....	134
1.3	STM32F3.....	134
1.4	STM32F4.....	135
2	Unterstützte Controller von Atmel.....	136
2.1	AVR (ISP-Interface).....	136
2.2	AVR (PDI-Interface).....	139
2.3	AVR (UPDI-Interface).....	139
3	Technische Daten.....	140

# I Vorwort

- Mit roloFlash können Sie mobil und unabhängig vom PC Ihre Produkte mit verschiedenen Mikrocontrollern flashen. Unter bestimmten Bedingungen können auch mehrere Mikrocontroller in Ihrem Produkt geflasht werden. Eine Liste der aktuell unterstützten Mikrocontroller finden Sie im Kapitel „Spezifikation“.
- Sie können ungelerntes Personal oder Kunden Ihre Produkte flashen lassen, da Fehlbedienungen prinzipbedingt ausgeschlossen sind.
- Dazu sind kein PC und keine spezifischen Tool-Chains (z. B. von Mikrocontroller-Herstellern) nötig.
- Nutzen Sie roloFlash im Feldeinsatz, in Ihrem Kundenumfeld bzw. zur Serien- und Kleinserienfertigung.
- Gewinnen Sie Freiräume, indem Sie auf einen einheitlichen Prozeß für alle unterstützten Mikrocontroller-Familien zurückgreifen.

## **Begriff „Atmel“**

Die Firma Atmel wurde von Microchip übernommen. Es wird jedoch weiterhin der Name „Atmel“ verwendet (in Dokumentation und Software), um Verwechslungen mit anderen Controllern von Microchip (z.B. PIC-Familien) zu vermeiden.

## **Begriff „Targetboard“**

Unter „Targetboard“ verstehen wir Ihre zu flashenden Produkte. Die Produkte enthalten den bzw. die zu flashenden Mikrocontroller. Diesen Begriff verwenden wir von nun an regelmäßig in diesem Dokument.

## **Begriff „Target“**

Unter „Target“ verstehen wir den bzw. die zu flashenden Mikrocontroller (falls mehrere vorhanden sind, z.B. JTAG-Chain).

## **Begriff „zu flashende Mikrocontroller“**

Außer „Flashen“ können Sie Ihre Mikrocontroller (Target) auch auslesen (und z. B. als HEX-Datei speichern), verifizieren (z. B. gegen eine HEX-

Datei), löschen oder modifizieren. Aus Gründen der Verständlichkeit wird oft nur das „Flashen“ erwähnt, ohne die anderen Möglichkeiten jedesmal zu wiederholen.

### **Zeichen „<“ und „>“**

Bei den Beschreibungen der Funktionen und Prozeduren werden die Parameter oft mit „<“ und „>“ eingerahmt. Dieses soll symbolisieren, daß an dieser Stelle ein sinnvoller Wert (ohne die spitzen Klammern) verwendet werden soll:

Beispiel:

```
delay <duration>
```

Hier können Sie z. B.

```
delay 1000
```

schreiben.

## II Verpackungsinhalt

Bitte überprüfen Sie sorgfältig den Lieferumfang:

- roloFlash 2
- microSD-Karte
  - vorbereitet für den Einsatz in Ihrem roloFlash, mit Dokumentation, Beispielen, Firmware und roloBasic-Compiler
  - zum Einlegen in roloFlash

Hinweis: Die microSD-Karte befindet sich entweder im roloFlash eingesteckt oder liegt bei.



# III Beschreibung

---

## 1 Programmier-Buchse

Die 10-polige Programmier-Buchse wird entweder auf einen passenden Stecker des zu programmierenden Targetboards gesteckt oder über einen passenden Adapter mit dem zu programmierenden Targetboard verbunden.

Sie finden auf der Vorderseite des roloFlash direkt über der Buchse eine Pin-1-Markierung.

Das Rastermaß der Buchse ist 2,54 mm (0,1 Zoll).

---

## 2 Pin-Belegung allgemein

Die Anschlüsse des roloFlash können je nach verwendetem Bus verschiedene Bedeutungen haben.

Die Standard-Belegung ist für JTAG. Falls Anschlüsse als GPIO verwendet werden, dann ist diese die Grundlage für die Benennung der Anschlüsse.

Die Belegung der Buchse ist zu der Belegung des 10-poligen JTAG-High-Density-Steckers mit Rastermaß 1,27 mm identisch (1:1), abgesehen vom Rastermaß 2,54 mm.

Signal	Pin	Pin	Signal
V <sub>targetboard</sub>	1 ●	● 2	TMS
GND	3 ●	● 4	TCK
GND	5 ●	● 6	TDO
RTCK	7 ●	● 8	TDI
GND Detect	9 ●	● 10	Reset

Abbildung 1: Draufsicht passender Stecker auf dem Targetboard

#### Hinweis:

Es gibt folgende Adapter, um die Pin-Belegung des roloFlash auf verschiedene übliche Programmierstecker-Belegungen anzupassen; diese werden bei den entsprechenden Bussen aufgezählt.

Darüber hinaus gibt es einen Universal-Adapter:

Bezeichnung	Pins	Reihen	Rastermaß [mm]
roloFlash-2-Universal-Adapter	1-20	1-2	1,27 (SMD) und 2,54 (thru-hole)

## 3 Pullup- / Pulldown-Widerstände

Um auf allen Leitungen einen definierten Pegel zu haben, besitzt roloFlash interne Pullup- und Pulldown-Widerstände:

Signal	Pin	Pin	Signal
Vtargetboard	1 ●	● 2	TMS Pullup 1MΩ
GND	3 ●	● 4	TCK Pulldown 1MΩ
GND	5 ●	● 6	TDO Pullup 1MΩ
RTCK Pullup 1MΩ	7 ●	● 8	TDI Pullup 1MΩ
GND Detect Pullup 1MΩ	9 ●	● 10	Reset Pullup 1MΩ

Abbildung 2: Draufsicht passender Stecker auf dem Targetboard

## 4 Spannungsbereich

roloFlash wird vom Targetboard aus über Pin 1 ( $V_{\text{targetboard}}$ ) versorgt. Dabei paßt roloFlash alle Datenleitungen auf diese Spannung an.

*Spannungsbereich: 2,0 Volt - 5,5 Volt*

## 5 Elektrische Schutzmaßnahmen

roloFlash ist geschützt gegen:

- Verpolung der Versorgungsspannung: Die Verbindung wird aufgetrennt.
- Überspannung auf der Versorgungsspannung: Bei Spannungen höher als 5,7 V schaltet eine Schutzschaltung ab.
- Alle Datenleitungen sind mit Polyswitches abgesichert.
- Der zweite GND-Anschluß auf Pin 5 ist zum Schutz des Targetboards nur über einen Polyswitch an GND auf Pin 3 verbunden.

- Alle Leitungen sind mit ESD-Schutzbauteilen ausgestattet, die IEC 61000-4-2 Level 4 (15 kV (air discharge) , 8 kV (contact discharge)) erfüllen.

Diese Maßnahmen bieten einen weitgehenden Schutz bei Fehlbedienungen wie verpoltes Aufstecken etc. Trotzdem ist nicht ausgeschlossen, daß bei Fehlbedienungen Schäden am Targetboard oder an roloFlash entstehen können.

## 6 Pin-Belegung JTAG-Interface

Wenn Sie das JTAG-Interface benutzen, dann werden folgende Anschlüsse für den Bus verwendet:

Signal	Pin	Pin	Signal
$V_{\text{targetboard}}$	1 ●	● 2	TMS
GND	3 ●	● 4	TCK
GND	5 ●	● 6	TDO
	7 ●	● 8	TDI
	9 ●	● 10	

Abbildung 3: Draufsicht passender Stecker auf dem Targetboard

### Hinweis:

Die Belegung der Buchse ist zu der Belegung des 10-poligen JTAG-High-Density-Steckers mit Rastermaß 1,27 mm identisch (1:1), abgesehen vom Rastermaß 2,54 mm.

### Hinweis:

Die übrigen Signale werden für JTAG nicht unbedingt benötigt und stehen anderen Bussen zur Verfügung. Sie können z. B. den Reset-Anschluß (Pin 9) als GPIO ansprechen und damit das Target resetten.

### Hinweis:

Es gibt folgende Adapter, um auf bestimmte übliche Programmiersteckerbelegungen zu adaptieren:

Bezeichnung	Pins	Reihen	Rastermaß [mm]
roloFlash-2-Target-Adapter ARM JTAG 20p	20	2	2,54
roloFlash-2-Target-Adapter ARM Cortex Debug 10p HD	10	2	1,27
roloFlash-2-Universal-Adapter	1-20	1-2	1,27 (SMD) und 2,54 (thru-hole)

## 7 Pin-Belegung Atmel ISP-Interface

Wenn Sie das ISP-Interface benutzen, dann werden folgende Anschlüsse für den Bus verwendet:

Signal	Pin	Pin	Signal
$V_{\text{targetboard}}$	1 ●	● 2	MISO
GND	3 ●	● 4	SCK
GND	5 ●	● 6	
	7 ●	● 8	MOSI
	9 ●	● 10	RST

Abbildung 4: Draufsicht passender Stecker auf dem Targetboard

### Achtung:

Die Pin-Belegung ist nicht zum 10-poligen ISP-Anschluß von Atmel kompatibel, auch wenn es mechanisch passen sollte.

### Hinweis:

Die übrigen Signale werden für ISP nicht benötigt und stehen anderen Bussen zur Verfügung. Sie können diese z. B. als GPIO ansprechen.

### Hinweis:

Es gibt folgende Adapter, um auf bestimmte übliche Programmierstecker-Belegungen zu adaptieren:

Bezeichnung	Pins	Reihen	Rastermaß [mm]
roloFlash-2-Target-Adapter Atmel ISP/TPI 6p (Hinweis: TPI-Unterstützung ist geplant)	6	2	2,54
roloFlash-2-Target-Adapter Atmel ISP/TPI 10p (Hinweis: TPI-Unterstützung ist geplant)	10	2	2,54
roloFlash-2-Universal-Adapter	1-20	1-2	1,27 (SMD) und 2,54 (thru-hole)

## 8 Pin-Belegung Atmel PDI-Interface

Wenn Sie das PDI-Interface benutzen, dann werden folgende Anschlüsse für den Bus verwendet:

Signal	Pin	Pin	Signal
$V_{\text{targetboard}}$	1 ●	● 2	Data
GND	3 ●	● 4	CLK
GND	5 ●	● 6	
	7 ●	● 8	
	9 ●	● 10	

Abbildung 5: Draufsicht passender Stecker auf dem Targetboard

### Achtung:

Die Pin-Belegung ist nicht zu dem 10-poligen ISP-Anschluss von Atmel kompatibel, auch wenn es mechanisch passen sollte.

### Hinweis:

Die übrigen Signale werden für PDI nicht benötigt und stehen anderen Bussen zur Verfügung. Sie können diese z. B. als GPIO ansprechen.

### Hinweis:

Es gibt folgende Adapter, um auf bestimmte übliche Programmierstecker-Belegungen zu adaptieren:

Bezeichnung	Pins	Reihen	Rastermaß [mm]
roloFlash-2-Target-Adapter Atmel PDI 6p	6	2	2,54
roloFlash-2-Universal-Adapter	1-20	1-2	1,27 (SMD) und 2,54 (thru-hole)

## 9 Pin-Belegung Atmel UPDI-Interface

Wenn Sie das UPDI-Interface benutzen, dann werden folgende Anschlüsse für den Bus verwendet:

Signal	Pin	Pin	Signal
$V_{\text{targetboard}}$	1 ●	● 2	UPDI_DATA
GND	3 ●	● 4	
GND	5 ●	● 6	
	7 ●	● 8	
	9 ●	● 10	

Abbildung 6: Draufsicht passender Stecker auf dem Targetboard

### Achtung:

Die Pin-Belegung ist nicht zu dem 10-poligen ISP-Anschluss von Atmel kompatibel, auch wenn es mechanisch passen sollte.

### Hinweis:

Die übrigen Signale werden für UPDI nicht benötigt und stehen anderen Bussen zur Verfügung. Sie können diese z. B. als GPIO ansprechen.

### Hinweis:

Es gibt folgende Adapter, um auf bestimmte übliche Programmierstecker-Belegungen zu adaptieren:

Bezeichnung	Pins	Reihen	Rastermaß [mm]
roloFlash-2-Target-Adapter Atmel PDI 6p	6	2	2,54
roloFlash-2-Universal-Adapter	1-20	1-2	1,27 (SMD) und 2,54 (thru-hole)

## 10 Pin-Belegung UART-Interface 0

Wenn Sie das UART-Interface benutzen, dann werden folgende Anschlüsse für den Bus verwendet:

Signal	Pin	Pin	Signal
$V_{\text{targetboard}}$	1 ●	● 2	TX
GND	3 ●	● 4	
GND	5 ●	● 6	RX
	7 ●	● 8	
	9 ●	● 10	

Abbildung 7: Draufsicht passender Stecker auf dem Targetboard

### Hinweis:

Sofern es keine Überschneidungen mit anderen Pins gibt, können andere Busse zeitgleich geöffnet werden. Dieses ist bei dieser UART bei JTAG, Atmel PDI und Atmel ISP nicht der Fall. Sie können jedoch UART-Interface 1 verwenden.

### Hinweis:

Die übrigen Signale werden für UART-Interface 0 nicht verwendet und stehen anderen Bussen zur Verfügung. Sie können diese z. B. als GPIO ansprechen.

## 11 Pin-Belegung UART-Interface 1

Wenn Sie das UART-Interface benutzen, dann werden folgende Anschlüsse für den Bus verwendet:



Signal	Pin	Pin	Signal
Vtargetboard	1 ●	● 2	
GND	3 ●	● 4	
GND	5 ●	● 6	
RX	7 ●	● 8	
TX	9 ●	● 10	

Abbildung 8: Draufsicht passender Stecker auf dem Targetboard

**Hinweis:**

Sofern es keine Überschneidungen mit anderen Pins gibt, können andere Busse zeitgleich geöffnet werden. Dieses ist bei dieser UART bei JTAG, Atmel PDI und Atmel ISP der Fall.

**Hinweis:**

Die übrigen Signale werden für UART Interface 0 nicht verwendet und stehen anderen Bussen zur Verfügung. Sie können diese z. B. als GPIO ansprechen.

---

## 12 Pin-Belegung GPIO-Interface

Es gibt für jeden als GPIO nutzbaren Pin ein eigenes Interface. Zur Verfügung stehen dazu:

Signal	Pin	Pin	Signal
	1 ●	● 2	TMS
	3 ●	● 4	TCK
	5 ●	● 6	TDO
RTCK	7 ●	● 8	TDI
GND Detect	9 ●	● 10	Reset

Abbildung 9: Draufsicht passender Stecker auf dem Targetboard

**Hinweis:**

Die Pins TDO und RTCK können nicht als Ausgang benutzt werden.

**Hinweis:**

Sofern es keine Überschneidungen mit anderen Pins gibt, können andere Busse zeitgleich geöffnet werden.

**Beispiel:**

Bei JTAG gehört das Reset-Signal bei roloFlash nicht zum JTAG-Bus dazu, da es für das eigentliche JTAG-Protokoll nicht benötigt wird. Sie können parallel zum geöffneten JTAG-Bus den Reset-Pin als GPIO öffnen und damit den Reset des Targets steuern.

## 13 LEDs

Fünf programmierbare zweifarbige (rote und grüne) LEDs. Mit den LEDs können Sie z. B.

- ein grünes Lauflicht laufen lassen, das den Flashvorgang darstellt.
- mit rot Fehlermeldungen ausgeben.

## 14 microSD-Kartenslot

Für eine microSD- oder microSDHC-Karte, die das abzuarbeitende Skript (RUN.BIN) sowie die zu flashenden Dateien enthält.

### **Ablauf / Verwendung:**

Der Ablauf gliedert sich in zwei Teile:

- Vorbereitung der microSD-Karte am PC (z. B. in der Entwicklung)
- Flashen der Targetboards (z. B. ungeschultes Personal in der Produktion, Kunde oder Techniker im Feldeinsatz)

---

## **15 Vorbereitung der microSD-Karte am PC**

### **Z. B. in der Entwicklung**

Maßgeblich ist immer die Datei "RUN.BIN", die von roloFlash ausgewertet wird, um den darin kodierten Programmablauf abzuarbeiten.

- Falls Sie eine microSD-Karte formatieren wollen, benutzen Sie dazu Windows 7 oder höher (Windows XP ist nicht geeignet).
- Sie erstellen den gewünschten Ablauf in roloBasic. Dazu können Sie ein Beispielskript verwenden oder anpassen. Im Kapitel „Spezifikation“ finden Sie eine Liste der exakten Namen der bekannten Controller, die Ihnen roloFlash dazu zur Verfügung stellt. Ihre erzeugte Datei sollte die Dateiendung „.BAS“ haben.
- Ihre Basic-Datei muß mit einer Magic-Cookie-Zeile anfangen. Diese lautet:

#### **#roloFlash 2**

Wenn diese Zeile fehlt oder einen anderen Inhalt hat, dann wird der Compiler das Übersetzen verweigern.

- Ihr Skript kann dabei auf übliche „.HEX“-Dateien (Intel-HEX-Format: „I8HEX“, „I16HEX“ und „I32HEX“) oder auf „.RAW“-Dateien verweisen, die auf das Targetboard geflasht werden sollen.
- Sie rufen auf dem PC den mitgelieferten Compiler „rbc.exe“ auf. Der Compiler erzeugt eine gleichnamige kompilierte Datei mit der Endung „.BIN“.
- Sie benennen die Datei in „RUN.BIN“ um oder rufen statt „rbc.exe“ die Batchdatei `compile.bat` auf, welche aus „RUN.BAS“ ein „RUN.BIN“ erzeugt. Danach legen Sie die Datei „RUN.BIN“ zusammen mit den vom Skript aus benötigten Dateien (z.B. eine „.HEX“-Datei, eventuell benötigte Loader) auf der microSD-Karte ab.

Sie können die Dateien mit den Skripten („BAS“), die kompilierten Dateien („BIN“) und den Compiler nach eigenem Ermessen auf dem PC und/oder auf der microSD-Karte speichern. Zum Flashen ist lediglich die Datei „RUN.BIN“ (sowie die durch den Code referenzierten Dateien) relevant.

---

## 16 Flashen der Targetboards

### Z. B. ungeschultes Personal in der Produktion

Hier ist der Ablauf denkbar einfach:

- Targetboard mit Energie versorgen.
- roloFlash auf den passenden Stecker des Targetboards aufstecken oder die Verbindung mit einem Adapter herstellen.
- roloFlash wird vom Targetboard mit Energie versorgt und beginnt automatisch mit der Abarbeitung der Datei „RUN.BIN“. Hierdurch wird üblicherweise das Flashen vorgenommen. Währenddessen kann z. B. ein grünes Lauflicht den Flashvorgang signalisieren.
- Wenn die RUN.BIN abgearbeitet wurde, was üblicherweise durch eine grün leuchtende LED 5 angezeigt wird, roloFlash abziehen – fertig.

## IV Aktualisieren von roloFlash

roloFlash verfügt selbst über eine eigene Firmware, die aktualisiert werden kann.

### Versionsnummern

Die Versionsnummer setzt sich aus `major` und `minor` zusammen:

- `major`:  
Major wird angepaßt wenn:
  - sich die Basic-Schnittelle ändert
  - grundlegende neue Funktionen hinzugefügt wurden
- `minor`:  
Minor wird angepaßt wenn:
  - sich die Änderungen auf das Beseitigen von Bugs beschränken
  - kleinere neue Funktionen hinzugefügt wurden

Daraus folgt, daß solange `major` nicht geändert wurde, auch kein Update des roloBasic-Compilers notwendig ist und bereits kompilierte RUN.BIN-Dateien gültig bleiben.

### Dateinamen für das Firmware-Update

Der Dateiname für das Firmware-Update hält sich an die 8.3-Konvention und ist wie folgt aufgebaut:

RF2\_aabb.HMP mit:

- `aa` = major (als Zahl, z.B. „01“)
- `bb` = minor (als Buchstaben, z.B. „AA“)

### Starten der Aktualisierung

- Zum Aktualisieren muß sich exakt eine Firmware-Datei im Hauptverzeichnis der microSD-Karte befinden. Sind mehrere Dateien vorhanden, wird die Aktualisierung nicht gestartet.
- Das Aktualisieren wird ausgelöst, wenn
  - der roloFlash **ohne** microSD-Karte auf ein beliebiges Targetboard aufgesteckt und anschließend die microSD-Karte eingesteckt wird.

- oder eine vorherige Aktualisierung fehlgeschlagen war. In diesem Fall ist es unerheblich, ob erst der roloFlash auf ein Targetboard aufgesteckt wird und dann die microSD-Karte eingesteckt wird oder die microSD-Karte schon eingesteckt ist.
- Es erfolgt keine Überprüfung, ob die Firmware auf der microSD-Karte neuer oder älter ist. Damit ist es auch möglich, zu einer älteren Version zurückzukehren, falls das erforderlich sein sollte.

Hinweis: Bei Auslieferung befindet sich die aktuelle Version in einem Unterverzeichnis. Die Datei wird erst dann zum Flashen herangezogen, wenn sie in das Hauptverzeichnis der microSD-Karte verschoben bzw. kopiert wurde.

### **Der Vorgang des Aktualisierens**

- Das Targetboard dient dabei nur zur Energieversorgung.
- Der Vorgang wird mittels der LEDs angezeigt, siehe Kapitel „roloFlash-Aktualisierung“.
- Solange die microSD-Karte noch nicht eingesteckt ist, leuchtet LED 1 rot.
- Während der Aktualisierung blinken LED 2 und LED 3 im Wechsel.  
**roloFlash sollte jetzt nicht abgezogen werden.**  
Falls roloFlash doch abgezogen worden sein sollte, kann es sein, daß die Firmware defekt ist. In diesem Zustand sollte roloFlash von selbst auf einer erneuten Aktualisierung bestehen.  
D. h. Bei der nächsten Energieversorgung wartet roloFlash solange, bis durch das Einschieben der microSD-Karte eine neue Firmware zur Verfügung steht. Diese wird erneut geflasht.  
**Falls eine Aktualisierung unterbrochen wurde, führen Sie auf jeden Fall eine erneute Aktualisierung durch, auch wenn Sie den Eindruck haben, daß die Aktualisierung eventuell doch erfolgreich war.**
- Bei Erfolg leuchten anschließend LED 1 und LED 2 grün.
- roloFlash bleibt in diesem Zustand, bis er abgezogen wird. Bitte ziehen Sie roloFlash nun ab.
- Ab dem nächsten Einstecken läuft roloFlash mit der aktualisierten Firmware.

Falls die Aktualisierung nicht erfolgreich gewesen sein sollte, verwenden Sie bitte eine frisch unter Windows 7 oder höher mit FAT32 formatierte microSD-Karte, auf der sich ausschließlich die Datei für die Firmware-Aktualisierung befindet.

**Hinweis:**

Für eine Produktion oder die Weitergabe des roloFlash an Ihre Kunden wird empfohlen, keine Datei für eine Firmware-Aktualisierung auf der microSD-Karte zu belassen.

# V Liste der mitgelieferten roloBasic-Skripte

- **Erase-and-Flash**

- scripts\STM\STM32F1-F4\JTAG\erase-and-flash\RUN.BAS
- scripts\Microchip\_Atme1\AVR\ISP\erase-and-flash\RUN.BAS
- scripts\Microchip\_Atme1\AVR\PDI\erase-and-flash\RUN.BAS

## **Vorbereitung:**

- Das Skript gibt es jeweils in einer Version für STM32-, Atmel-ISP- und Atmel-PDI-Controller.
- Zum Verwenden kopieren Sie bitte die passende Version als RUN.BAS in das Hauptverzeichnis der microSD-Karte.
- Passen Sie bitte anschließend in der Datei den Namen Ihres Targets und die Dateinamen der HEX-Datei an (bei Atmel: Zusätzlich zur Angabe einer HEX-Datei für den Flashspeicher können Sie auch eine weitere HEX-Datei für das EEPROM angeben).
- Optional können Sie auch die Busgeschwindigkeit sowie die Geschwindigkeit des roloFlashs anpassen.
- Starten Sie den Compiler mittels „compile.bat“, um aus der RUN.BAS die benötigte RUN.BIN zu erzeugen.

## **Funktion:**

- Startet ein Lauflicht von LED 1 zu LED 4, um einen Flash-Vorgang darzustellen.
- Löscht eine eventuell vorhandene vorherige LOG.TXT-Datei.
- Öffnet den jeweiligen Bus zum Target.
- Liest aus der internen Datenbank des roloFlash spezifische Informationen für den von Ihnen angegebenen Controller, darunter die ID in Form einer Signature (bei Atmel ISP), einer Device-ID (bei Atmel PDI) oder eines oder mehrerer IDCODEs (bei einem STM32-Controller zwei IDCODEs für Core- und BoundaryScan-Controller), sowie andere für das Flashen benötigte Parameter, aus.



- Liest die ID(s) des angeschlossenen Targets aus und vergleicht diese mit den Werten aus der Datenbank.
  - Wenn die ID(s) nicht stimmen sollte(n) (z. B. anderer Controller), dann wird der weitere Ablauf mit Ausgabe einer Fehlermeldung abgebrochen
  - Löscht das Target (erase).
  - Wenn von Ihnen angegeben: Ihre HEX-Datei wird in das Flash des Targets geschrieben und verifiziert.
  - Nur Atmel: Wenn von Ihnen angegeben: Ihre HEX-Datei wird in das EEPROM des Targets geschrieben und verifiziert.
  - Währenddessen läuft ein grünes Lauflicht, am Ende bleibt bei Erfolg LED 5 auf grün.
  - Schreibt die Ergebnisse ins Log-File (LOG.TXT)
- **Read**
    - scripts\STM\STM32F1-F4\JTAG\read\RUN.BAS
    - scripts\Microchip\_Atmel\AVR\ISP\read\RUN.BAS
    - scripts\Microchip\_Atmel\AVR\PDI\read\RUN.BAS

#### **Vorbereitung:**

- Das Skript gibt es jeweils in einer Version für STM32-, Atmel-ISP- und Atmel-PDI-Controller.
- Zum Verwenden kopieren Sie bitte die passende Version als RUN.BAS in das Hauptverzeichnis der microSD-Karte.
- Passen Sie bitte anschließend in der Datei den Namen Ihres Targets und die Dateinamen der HEX-Datei an (bei Atmel: Zusätzlich zur Angabe einer HEX-Datei für den Flashspeicher können Sie auch eine weitere HEX-Datei für das EEPROM angeben).
- Optional können Sie auch die Busgeschwindigkeit sowie die Geschwindigkeit des roloFlashs anpassen.
- Starten Sie den Compiler mittels „compile.bat“, um aus der RUN.BAS die benötigte RUN.BIN zu erzeugen.

#### **Funktion:**

- Startet ein Lauflicht von LED 4 zu LED 1, um einen Lese-Vorgang darzustellen.
- Löscht eine eventuell vorhandene vorherige LOG.TXT-Datei.
- Öffnet den jeweiligen Bus zum Target.
- Liest aus der internen Datenbank des roloFlash spezifische Informationen für den von Ihnen angegebenen Controller, darunter die ID in Form einer Signature (bei Atmel ISP), einer Device-ID (bei Atmel PDI) oder eines oder mehrerer IDCODEs (bei einem STM32-Controller zwei IDCODEs für Core- und BoundaryScan-Controller), sowie andere für das Lesen benötigte Parameter (Größe des Speichers), aus.
- Liest die ID(s) des angeschlossenen Targets aus und vergleicht diese mit den Werten aus der Datenbank.
- Wenn die ID(s) nicht stimmen sollte(n) (z. B. anderer Controller), dann wird der weitere Ablauf mit Ausgabe einer Fehlermeldung abgebrochen
- Wenn von Ihnen angegeben: Das Flash des Targets wird komplett ausgelesen und in die von Ihnen angegebene HEX-Datei geschrieben.
- Nur Atmel: Wenn von Ihnen angegeben: Das EEPROM des Targets wird komplett ausgelesen und in die von Ihnen angegebene HEX-Datei geschrieben.
- Währenddessen läuft ein grünes Lauflicht, am Ende bleibt bei Erfolg LED 5 auf grün.
- Schreibt die Ergebnisse ins Log-File (LOG.TXT)

# VI roloFlash-API (Liste der Prozeduren und Funktionen)

(API = Application Programming Interface)

Damit ist die Schnittstelle gemeint, durch die roloBasic Zugriff auf alle roloFlash-spezifischen Prozeduren und Funktionen erlangt.

Sie können die Prozeduren und Funktionen direkt aufrufen.

## Prozeduren:

Prozeduren haben keinen Rückgabewert. Die übergebenen Parameter müssen ohne Klammern angegeben werden.

Beispiel:

```
delay 1000
```

## Funktionen:

Funktionen haben einen Rückgabewert. Die übergebenen Parameter müssen in Klammern gesetzt werden.

Beispiel:

```
handle = fsOpen(0, "TEST.TXT")
```

Hat die Funktion keine Parameter, dann können die Klammern weggelassen werden.

Beispiel:

```
value = getTargetBoardVoltage
```

oder

```
value = getTargetBoardVoltage()
```

## 1 Interne Datenbank

In roloFlash ist eine Datenbank integriert, die zu vielen Targets Informationen enthält. Die Informationen dienen folgenden Zwecken:

- Um in roloBasic zu prüfen, ob das gewünschte Target wirklich angeschlossen ist (z. B. JTAG IDCODE, Atmel ISP-Signature oder Atmel PDI-Device-ID).
- Um für das Flashen benötigte Daten zur Verfügung zu stellen.

Über den gewünschten Namen des Controllers kann man von der Datenbank ein Handle bekommen und mit diesem die weiteren Informationen abfragen. Das Handle muß anschließend nicht geschlossen werden.

### 1.1 DB\_getHandle

Unter Angabe eines Target-Namens kann dazu ein passendes Handle ermittelt werden.

```
dbHandle = DB_getHandle(<name>)
```

#### **Vorbedingung:**

- keine

#### **Parameter:**

##### **name**

Name des Targets. Es kann sein, daß der Name in der Datenbank verkürzt abgespeichert ist. Das ist dann der Fall, falls es mehrere Targets gibt, die sich z.B. nur in der Gehäuseform unterscheiden und ansonsten gleiche Parameter haben. Bitte schauen Sie für Ihren Controller in der Liste nach, wie die korrekte Schreibweise ist. Bitte achten Sie auch auf die dort verwendete Groß-/Kleinschreibung.

#### **Rückgabewert:**

- ein Datenbank-Handle. Dieses kann benutzt werden, um mittels DB\_get Informationen zu diesem Target abzufragen.

**Exceptions:**

unknownTarget  
apiTypeFault

Target nicht bekannt  
Unzulässiger Typ für name

## 1.2 DB\_get

Unter Angabe eines bereits mittels DB\_getHandle erhaltenden Handles können weitere Informationen abgefragt werden.

Value = DB\_get(<dbHandle>, <property>)

**Vorbedingung:**

- gültiges dbHandle

**Parameter:**

**dbHandle**

Handle zum Zugriff auf die Datenbank, siehe DB\_getHandle

**property**

Angabe, welche Information ermittelt werden soll. Es stehen nicht für alle dbHandles alle Properties zur Verfügung. In dem Fall, daß die Information nicht ermittelt werden kann, wird eine Exception erzeugt.

Mögliche Werte für property sind:

**DB\_NAME:** Name des Targets. (Dieser kann kürzer sein als der Name, der zur Ermittlung des dbHandles angegeben wurde)

**DB\_FAMILY:** Ein Wert, der die Zugehörigkeit zu einer bestimmten Familie angibt. Dieser Wert wird zum Erhalten eines Target-Handles (siehe getTargetHandle) benötigt.

**DB\_COREIDCODE:** Im Fall eines JTAG-Device der IDCODE des Targets

**DB\_BOUNDARYSCANIDCODE :** Im Fall eines JTAG-Targets kann das Target ein zusätzliches Boundary-Scan JTAG Device beinhalten. In diesem Fall wird dessen IDCODE ausgegeben.

**DB\_FLASHSIZE:** Angaben über die Größe des Flashs in Bytes.

**DB\_FLASHPAGESIZE:** Angabe einer Page-Größe in Bytes für das Schreiben von Speicher mit bestimmten Page-Größen (z. B. Atmel AVR und Atmel Xmega)

**DB\_EEPROMSIZE:** Angaben über die Größe des EEPROMs in Bytes.

**DB\_EEPROMPAGESIZE:** Angabe einer Page-Größe in Bytes für das Schreiben von EEPROM mit bestimmten Page-Größen (z. B. Atmel Xmega)

**DB\_SIGNATURE:** Angabe der Signature (z. B. Atmel AVR)

**DB\_DEVICEID:** Angabe der Device ID (z. B. Atmel Xmega)

### **Rückgabewert:**

- der abgefragte Wert für das Property

### **Exceptions:**

propertyNotFound

Der gewünschte Wert ist nicht bekannt oder existiert nicht (z.B. DB:COREIDCODE bei Nicht-JTAG-Targets)

apiTypeFault

Unzulässiger Typ für dbHandle oder property

---

## **2 Öffnen und Schließen von Bussen und Targets**

Bei roloFlash wird grundsätzlich jede Schnittstelle, über die ein Target geflasht werden kann, als Bus aufgefaßt. Dieses gilt auch, wenn an dieser Schnittstelle prinzipbedingt nur ein einziger Mikrocontroller angeschlossen sein kann (z. B. wird die ISP-Schnittstelle für Atmel AVR als Bus aufgefaßt).

- Grundsätzlich muß ein Bus erst geöffnet werden.
- Beim Öffnen wird geprüft, ob der Bus zur Verfügung steht. Sollte der Bus schon geöffnet sein, wird eine Exception erzeugt (resourceUnavailable). Die gleiche Exception erhalten Sie, falls Sie schon einen anderen Bus geöffnet haben und die Signale oder interne Ressourcen sich überschneiden würden.

- Je nach Bus steht dazu eine passende Funktion zur Verfügung, die in dem jeweiligen Kapitel beschrieben ist.
- Ein an dem Bus angeschlossener Mikrocontroller (Target) kann erst angesprochen werden, wenn man von dem Bus ein Target-Handle erhalten hat.
- Die Verbindung zu einem Target-Handle kann auch wieder geschlossen werden.
- Ein Bus kann auch geschlossen werden. In diesem Fall werden die betroffenen Leitungen wieder hochohmig.

Hinweis: die Funktionen zum Öffnen eines Busses finden Sie in dem jeweiligen Kapitel für den entsprechenden Bus.

## 2.1 Bus öffnen

Diese Funktion ist für jeden Bus spezifisch (z. B. für JTAG die Funktion JTAG\_open). Daher wird diese Funktion in den jeweiligen Kapiteln detailliert behandelt.

```
busHandle = <bus>_open(<index>, ...)
```

Öffnet den entsprechenden Bus <bus> und stellt ein busHandle zur Verfügung. Je nach Bus können hierbei Leitungen initialisiert werden.

Es kann je nach verwendeten Bus weitere Parameter geben.

### **Vorbedingung:**

- keine

### **Parameter:**

#### **index**

Gibt an, der wievielte Bus geöffnet werden soll. Der erste Bus hat den Index 0.

**Rückgabewert:**

- ein busHandle. Dieses kann benutzt werden, um weitere Funktionen wie z.B. getTargetHandle aufzurufen.

**Exceptions:**

apiValueRange  
apiTypeFault  
resourceUnavailable

Unzulässiger Wert für index  
Unzulässiger Typ für index  
Der Bus kann nicht geöffnet werden. Mögliche Gründe:  
- Der Bus wurde bereits geöffnet  
- ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich

## 2.2 Bus schließen

closeBus <busHandle>

Schließt den entsprechenden Bus. Die betroffenen Leitungen werden dabei abgeschaltet.

Sollten auf dem Bus noch geöffnete Targets vorhanden sein, dann werden diese abgetrennt und die Target-Handles ungültig.

**Vorbedingung:**

- gültiges BusHandle

**Parameter:**

**busHandle**

Das BusHandle auf den geöffneten Bus.

**Rückgabewert:**

- keiner (Prozedur)



**Exceptions:**

invalidHandle  
apiTypeFault

Handle ist schon geschlossen  
Unzulässiger Typ für busHandle

## 2.3 Target öffnen

```
targetHandle = getTargetHandle(<bushandle>, <index>,  
<family>)
```

Ermöglicht den Zugriff auf ein Target und liefert ein Target -Handle.

**Hinweis:**

Die Funktion prüft nicht, ob tatsächlich ein Target angeschlossen ist. Falls das geprüft werden soll, kann getTargetPresent verwendet werden.

**Vorbedingung:**

- gültiges BusHandle

**Parameter:**

**busHandle**

Das Bus-Handle auf den geöffneten Bus.

**index**

Gibt an, welches Target auf dem Bus geöffnet werden soll. Die Zählweise ist vom Bus abhängig. In den meisten Fällen sind die Targets durchnummeriert – das erste Target hat den Index 0.

Bei Bussen, die nur ein Target unterstützen, muß als Index immer die 0 angegeben werden.

**Hinweis:**

Bei JTAG (z.B. STM32) kann es sein, daß mehrere Targets einem Mikrocontroller entsprechen. Ein Mikrocontroller meldet z.B. auf dem JTAG-Bus ein JTAG-Device für den eigentlichen Controller und ein weiteres JTAG-Device für den Boundary-Scan-Controller an.

**Hinweis:**

Bitte bei ISP- und PDI-Bus immer 0 angeben.

### **family**

Mit diesem Parameter wird festgelegt, welcher Controller-Familie der Controller zugeordnet ist. Der Wert dazu kann direkt angegeben werden oder gegebenenfalls aus der internen Datenbank ausgelesen werden. Mögliche Familien: ATMELISP, ATMELPDI, STM32F1 etc.

### **Rückgabewert:**

- ein Target-Handle. Dieses kann benutzt werden, um weitere Funktionen wie z.B. getTargetPresent aufzurufen.

### **Exceptions:**

apiValueRange  
apiTypeFault  
invalidHandle

Unzulässiger Wert für index  
Unzulässiger Typ für busHandle oder index  
Das BusHandle ist ungültig (z.B. schon geschlossen)

## **2.4 Target schließen**

`closeTarget <targetHandle>`

Schließt das entsprechende Target.

### **Vorbedingung:**

- gültiges Target-Handle

### **Parameter:**

#### **targetHandle**

Das Target-Handle auf den geöffneten Bus.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

invalidHandle

Das Target-Handle ist schon geschlossen, oder der dazu gehörige Bus wurde schon geschlossen.

apiTypeFault

Unzulässiger Typ für targetHandle

---

## 3 JTAG-Bus

### 3.1 Bus öffnen

```
busHandle = JTAG_open(<index>, <speed>)
```

Öffnet den JTAG-Bus und initialisiert die Leitungen. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

**Vorbedingung:**

- keine

**Parameter:**

**index**

Muss 0 sein.

**speed**

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Geschwindigkeiten sind von der CPU-Clock (setCpuClock) des roloFlash abhängig.

Bei maximaler CPU-Clock = 120 Mhz werden die folgenden Geschwindigkeiten unterstützt:

15000000	7500000	5000000	3750000	3000000
2500000	2142857	1875000	1666666	1500000
1363636	1250000	1153846	1071428	1000000
937500	882352	833333	789473	750000
714285	681818	652173	625000	600000
576923	555555	535714	517241	500000
483870	468750	454545	441176	428571
416666	405405	394736	384615	375000
365853	357142	348837	340909	333333
326086	319148	312500	306122	300000
294117	288461	283018	277777	272727
267857	263157	258620	254237	250000
245901	241935	238095	234375	230769
227272	223880	220588	217391	214285
211267	208333	205479	202702	200000
197368	194805	192307	189873	187500
185185	182926	180722	178571	176470
174418	172413	170454	168539	166666
164835	163043	161290	159574	157894
156250	154639	153061	151515	150000
148514	147058	145631	144230	142857
141509	140186	138888	137614	136363
135135	133928	132743	131578	130434
129310	128205	127118	126050	125000
123966	122950	120967	119047	117187
115384	113636	111940	110294	108695
107142	105633	104166	102739	101351
100000	98684	97402	96153	94936
93750	92592	91463	90361	89285
88235	87209	86206	84745	83333
81967	80645	79365	78125	76923
75757	74626	73529	72463	71428
70422	69124	67873	66666	65502
64377	63291	62240	61224	60000
58823	57692	56603	55555	54545
53380	52264	51194	50167	49019
47923	46875	45871	44776	43731
42613	41551	40540	39473	38461
37406	36319	35294	34246	33185
32119	31055	30000	28957	27932
26929	25906	24875	23847	22831

21802	20775	19762	18750	17730
16722	15706	14705	13698	12690
11682	10676	9671	8670	7668
6666	5664	4662	3661	2660
1659				

Bei minimaler CpuClock = 24 Mhz werden die folgenden Geschwindigkeiten unterstützt:

1500000	750000	500000	375000	300000
250000	214285	187500	166666	150000
136363	125000	115384	107142	100000
93750	88235	83333	78947	75000
71428	68181	65217	62500	60000
57692	55555	53571	51724	50000
48387	46875	45454	44117	42857
41666	40540	39473	38461	36585
34883	33333	31914	30612	29411
28301	27272	25862	24590	23437
22388	21126	20000	18987	17857
16853	15789	14705	13636	12605
11538	10489	9433	8426	7425
6410	5395	4385	3378	2377
1376				

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet.

**Hinweis:**

Es kann sein, daß beim Scannen der JTAG-Chain langsamer als der angegebenen Wert gescannt wird. Der angegebene Wert ist für alle anschließenden Transfers relevant.

**Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels set - CpuClock den Takt des roloFlash ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie setCpuClock zuerst und öffnen dann erst den Bus.

- Oder setzen Sie nach setCpuClock die Busgeschwindigkeit erneut mittels JTAG\_setSpeed.

**Rückgabewert:**

- ein busHandle. Dieses kann benutzt werden, um weitere Funktionen wie z.B. getTargetHandle aufzurufen

**Exceptions:**

apiValueRange	Unzulässiger Wert für index oder speed.
apiTypeFault	Unzulässiger Typ für index oder speed
resourceUnavailable	Der Bus kann nicht geöffnet werden. Mögliche Gründe: - Der Bus wurde bereits geöffnet - ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich

### 3.2 Busgeschwindigkeit ändern

JTAG\_setSpeed(<busHandle>, <speed>)

Ändert bei einem bereits geöffneten JTAG-Bus die Busgeschwindigkeit. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von JTAG\_open erhaltenen Bus-Handle.

**speed**

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Werte sind im vorherigen Kapitel (JTAG\_open) angegeben.

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet.

**Hinweis:**

Es kann sein, daß beim Scannen der JTAG-Chain langsamer als der angegebenen Wert gescannt wird. Der angegebene Wert ist für alle anschließenden Transfers relevant.

**Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels `setCpuClock` den Takt des roloFlash ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie `setCpuClock` zuerst und öffnen dann erst den Bus.
- Oder setzen Sie nach `setCpuClock` die Busgeschwindigkeit erneut mittels `JTAG_setSpeed`.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

`apiValueRange`  
`apiTypeFault`

Unzulässiger Wert für `speed`.  
Unzulässiger Typ für `busHandle` oder `speed`

### 3.3 Busgeschwindigkeit abfragen

```
Speed = JTAG_getSpeed(<busHandle>)
```

Fragt bei einem bereits geöffneten JTAG-Bus die aktuelle Busgeschwindigkeit ab. Diese kann gleich oder geringer sein als die bei `JTAG_open` bzw. `JTAG_setSpeed` angegebene Busgeschwindigkeit.

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von JTAG\_open erhaltenen Bus-Handle.

**Rückgabewert:**

- Busgeschwindigkeit in Hz

**Exceptions:**

apiTypeFault

Unzulässiger Typ für busHandle

### 3.4 JTAG-Chain scannen

```
idCodes = JTAG_scan(<busHandle>)
```

Führt einen Scan auf dem JTAG-Bus durch und liefert ein Array mit den IDCODEs der gefundenen JTAG-Devices.

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von JTAG\_open erhaltene Bus-Handle.

**Rückgabewert:**

ein Array mit den IDCODEs. Für JTAG-Devices, die keinen IDCODE unterstützen, wird eine 0 eingetragen.



**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für index oder speed.  
Kein gültiges BusHandle auf den JTAG-Bus

## 4 Atmel ISP-Bus

### 4.1 Bus öffnen

```
busHandle = ISP_open(<index>, <speed>)
```

Öffnet den ISP-Bus und initialisiert die Leitungen. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

**Vorbedingung:**

- keine

**Parameter:**

**index**

Muß 0 sein.

**speed**

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Geschwindigkeiten sind von der CPU-Clock (setCpuClock) des roloFlash abhängig.

Bei maximaler CPU-Clock = 120 Mhz werden die folgenden Geschwindigkeiten unterstützt:

15000000	7500000	5000000	3750000	3000000
2500000	2142857	1875000	1666666	1500000
1363636	1250000	1153846	1071428	1000000
937500	882352	833333	789473	750000
714285	681818	652173	625000	600000

576923	555555	535714	517241	500000
483870	468750	454545	441176	428571
416666	405405	394736	384615	375000
365853	357142	348837	340909	333333
326086	319148	312500	306122	300000
294117	288461	283018	277777	272727
267857	263157	258620	254237	250000
245901	241935	238095	234375	230769
227272	223880	220588	217391	214285
211267	208333	205479	202702	200000
197368	194805	192307	189873	187500
185185	182926	180722	178571	176470
174418	172413	170454	168539	166666
164835	163043	161290	159574	157894
156250	154639	153061	151515	150000
148514	147058	145631	144230	142857
141509	140186	138888	137614	136363
135135	133928	132743	131578	130434
129310	128205	127118	126050	125000
123966	122950	120967	119047	117187
115384	113636	111940	110294	108695
107142	105633	104166	102739	101351
100000	98684	97402	96153	94936
93750	92592	91463	90361	89285
88235	87209	86206	84745	83333
81967	80645	79365	78125	76923
75757	74626	73529	72463	71428
70422	69124	67873	66666	65502
64377	63291	62240	61224	60000
58823	57692	56603	55555	54545
53380	52264	51194	50167	49019
47923	46875	45871	44776	43731
42613	41551	40540	39473	38461
37406	36319	35294	34246	33185
32119	31055	30000	28957	27932
26929	25906	24875	23847	22831
21802	20775	19762	18750	17730
16722	15706	14705	13698	12690
11682	10676	9671	8670	7668
6666	5664	4662	3661	2660
1659				

Bei minimaler CpuClock = 24 Mhz werden die folgenden Geschwindigkeiten unterstützt:

1500000	750000	500000	375000	300000
250000	214285	187500	166666	150000
136363	125000	115384	107142	100000
93750	88235	83333	78947	75000
71428	68181	65217	62500	60000
57692	55555	53571	51724	50000
48387	46875	45454	44117	42857
41666	40540	39473	38461	36585
34883	33333	31914	30612	29411
28301	27272	25862	24590	23437
22388	21126	20000	18987	17857
16853	15789	14705	13636	12605
11538	10489	9433	8426	7425
6410	5395	4385	3378	2377
1376				

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet.

**Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels `setCpuClock` den Takt des roloFlashs ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie `setCpuClock` zuerst und öffnen dann erst den Bus.
- Oder setzen Sie nach `setCpuClock` die Busgeschwindigkeit erneut mittels `ISP_setSpeed`.

**Rückgabewert:**

- ein `busHandle`. Dieses kann benutzt werden, um weitere Funktionen wie z.B. `getTargetPresent` aufzurufen.

**Exceptions:**

apiValueRange  
apiTypeFault  
resourceUnavailable

Unzulässiger Wert für index oder speed.  
Unzulässiger Typ für index oder speed  
Der Bus kann nicht geöffnet werden. Mögliche Gründe:  
- Der Bus wurde bereits geöffnet  
- ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich

## 4.2 Busgeschwindigkeit ändern

ISP\_setSpeed(<busHandle>, <speed>)

Ändert bei einem bereits geöffneten ISP-Bus die Busgeschwindigkeit. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von ISP\_open erhaltenen Bus-Handle.

**speed**

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Werte sind im vorherigen Kapitel „Bus öffnen“ angegeben.

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet.

**Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels set - CpuClock den Takt des roloFlashs ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie setCpuClock zuerst und öffnen dann erst den Bus.

- Oder setzen Sie nach setCpuClock die Busgeschwindigkeit erneut mittels ISP\_setSpeed.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für speed.  
Unzulässiger Typ für busHandle oder speed

### 4.3 Busgeschwindigkeit abfragen

Speed = ISP\_getSpeed(<busHandle>)

Fragt bei einem bereits geöffneten ISP-Bus die aktuelle Busgeschwindigkeit ab. Diese kann gleich oder geringer als die bei bei ISP\_open bzw. ISP\_setSpeed angegebene Busgeschwindigkeit sein.

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von ISP\_open erhaltene Bus-Handle.

**Rückgabewert:**

- aktuelle Busgeschwindigkeit in Hz.

**Exceptions:**

apiTypeFault

Unzulässiger Typ für busHandle

## 4.4 Reset-Mode einstellen

```
ISP_resetMode <busHandle> <resetMode>
```

Setzt für den ISP-Bus den Reset-Mode.

Nach dem Öffnen des ISP-Busses ist der ResetMode auf `pushpull` gesetzt. D.h.:

- Wenn kein Reset angelegt ist, ist die RST-Leitung aktiv high.
- Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv low

### **Vorbedingung:**

- gültiges `busHandle`

### **Parameter:**

#### **busHandle**

Das von `ISP_open` erhaltene Bus-Handle.

#### **rstMode**

- **PIN\_ACTIVELOW:**
  - Wenn kein Reset angelegt ist, ist die RST-Leitung hochohmig.
  - Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv low.
- **PIN\_ACTIVEHIGH:**
  - Wenn kein Reset angelegt ist, ist die RST-Leitung hochohmig.
  - Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv high.
- **PIN\_PUSHPULL:**
  - Wenn kein Reset angelegt ist, ist die RST-Leitung aktiv high.
  - Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv low.
- **PIN\_INVERTED:**
  - Wenn kein Reset angelegt ist, ist die RST-Leitung aktiv low.
  - Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv high.

**Hinweis:**

- Die rstModes PIN\_ACTIVEHIGH und PIN\_INVERTED sind gegenüber der üblichen Reset-Funktion invertiert und ziehen die Leitung für einen Reset auf high. Dieses ist nur für Controller sinnvoll, bei denen Reset high aktiv ist. In diesem Fall wird PIN\_INVERTED empfohlen.

**Rückgabewert:**

- keiner.

**Exceptions:**

apiTypeFault

Unzulässiger Typ für busHandle

---

## 5 Atmel PDI-Bus

### 5.1 Bus öffnen

```
busHandle = PDI_open(<index>, <speed>)
```

Öffnet den PDI-Bus und initialisiert die Leitungen. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

**Vorbedingung:**

- keine

**Parameter:**

**index**

Muß 0 sein.

**speed**

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Geschwindigkeiten sind von der CPU-Clock (setCpuClock) des roloFlash abhängig.

Bei maximaler CPU-Clock = 120 Mhz werden die folgenden Geschwindigkeiten unterstützt:

15000000	7500000	5000000	3750000	3000000
2500000	2142857	1875000	1666666	1500000
1363636	1250000	1153846	1071428	1000000
937500	882352	833333	789473	750000
714285	681818	652173	625000	600000
576923	555555	535714	517241	500000
483870	468750	454545	441176	428571
416666	405405	394736	384615	375000
365853	357142	348837	340909	333333
326086	319148	312500	306122	300000
294117	288461	283018	277777	272727
267857	263157	258620	254237	250000
245901	241935	238095	234375	230769
227272	223880	220588	217391	214285
211267	208333	205479	202702	200000
197368	194805	192307	189873	187500
185185	182926	180722	178571	176470
174418	172413	170454	168539	166666
164835	163043	161290	159574	157894
156250	154639	153061	151515	150000
148514	147058	145631	144230	142857
141509	140186	138888	137614	136363
135135	133928	132743	131578	130434
129310	128205	127118	126050	125000
123966	122950	120967	119047	117187
115384	113636	111940	110294	108695
107142	105633	104166	102739	101351
100000				

Bei minimaler CpuClock = 24 Mhz werden die folgenden Geschwindigkeiten unterstützt:

1500000	750000	500000	375000	300000
250000	214285	187500	166666	150000
136363	125000	115384	107142	100000



Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet. Die minimale Busgeschwindigkeit wird von Atmel mit 100 kHz angegeben. Bei Angabe von kleineren Werten wird auf 100 kHz aufgerundet.

**Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels `setCpuClock` den Takt des roloFlashs ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie `setCpuClock` zuerst und öffnen dann erst den Bus.
- Oder setzen Sie nach `setCpuClock` die Busgeschwindigkeit erneut mittels `PDI_setSpeed`.

**Rückgabewert:**

- ein `busHandle`. Dieses kann benutzt werden, um weitere Funktionen wie z.B. `getTargetPresent` aufzurufen.

**Exceptions:**

<code>apiValueRange</code>	Unzulässiger Wert für <code>index</code> oder <code>speed</code> .
<code>apiTypeFault</code>	Unzulässiger Typ für <code>index</code> oder <code>speed</code>
<code>resourceUnavailable</code>	Der Bus kann nicht geöffnet werden. Mögliche Gründe: - Der Bus wurde bereits geöffnet - ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich

## 5.2 Busgeschwindigkeit ändern

`PDI_setSpeed(<busHandle>, <speed>)`

Ändert bei einem bereits geöffneten PDI-Bus die Busgeschwindigkeit. Die maximale Busgeschwindigkeit wird auf „`speed`“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von PDI\_open erhaltenen Bus-Handle.

**speed**

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Werte sind im vorherigen Kapitel „Bus öffnen“ angegeben.

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet.

**Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels setCpuClock den Takt des roloFlashs ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie setCpuClock zuerst und öffnen dann erst den Bus.
- Oder setzen Sie nach setCpuClock die Busgeschwindigkeit erneut mittels PDI\_setSpeed .

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für speed.  
Unzulässiger Typ für busHandle oder speed

### 5.3 Busgeschwindigkeit abfragen

Speed = PDI\_getSpeed(<busHandle>)

Fragt bei einem bereits geöffneten PDI-Bus die aktuelle Busgeschwindigkeit ab. Diese kann gleich oder geringer als die bei PDI\_open bzw. PDI\_setSpeed angegebene Busgeschwindigkeit sein.

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von PDI\_open erhaltene Bus-Handle.

**Rückgabewert:**

- aktuelle Busgeschwindigkeit in Hz.

**Exceptions:**

apiTypeFault

Unzulässiger Typ für busHandle

---

## 6 Atmel UPDI-Bus

### 6.1 Bus öffnen

```
busHandle = UPDI_open(<index>, <speed>)
```

Öffnet den UPDI-Bus und initialisiert die Leitungen. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

**Vorbedingung:**

- keine

**Parameter:**

## index

Muß 0 sein.

## speed

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Geschwindigkeiten sind von der CPU-Clock (setCpuClock) des roloFlash abhängig.

Bei maximaler CPU-Clock = 120 Mhz werden die folgenden Geschwindigkeiten unterstützt:

500000	483870	468750	454545	441176
428571	416666	405405	394736	384615
375000	365853	357142	348837	340909
333333	326086	319148	312500	306122
300000	294117	288461	283018	277777
272727	267857	263157	258620	254237
250000	245901	241935	238095	234375
230769	227272	223880	220588	217391
214285	211267	208333	205479	202702
200000	197368	194805	192307	189873
187500	185185	182926	180722	178571
176470	174418	172413	170454	168539
166666	164835	163043	161290	159574
157894	156250	154639	153061	151515
150000	148514	147058	145631	144230
142857	141509	140186	138888	137614
136363	135135	133928	132743	131578
130434	129310	128205	127118	126050
125000	123966	122950	120967	119047
117187	115384	113636	111940	110294
108695	107142	105633	104166	102739
101351	100000	98684	97402	96153
94936	93750	92592	91463	90361
89285	88235	87209	86206	84745
83333	81967	80645	79365	78125
76923	75757	75000		

Bei minimaler CpuClock = 24 Mhz werden die folgenden Geschwindigkeiten unterstützt:

375000	300000	250000	214285	187500
--------	--------	--------	--------	--------

166666	150000	136363	125000	115384
107142	100000	93750	88235	83333
78947	75000			

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet. Die minimale Busgeschwindigkeit wird von Atmel mit 100 kHz angegeben. Bei Angabe von kleineren Werten wird auf 100 kHz aufgerundet.

#### **Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels `setCpuClock` den Takt des roloFlashs ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie `setCpuClock` zuerst und öffnen dann erst den Bus.
- Oder setzen Sie nach `setCpuClock` die Busgeschwindigkeit erneut mittels `UPDI_setSpeed`.

#### **Rückgabewert:**

- ein `busHandle`. Dieses kann benutzt werden, um weitere Funktionen wie z.B. `getTargetPresent` aufzurufen.

#### **Exceptions:**

<code>apiValueRange</code>	Unzulässiger Wert für <code>index</code> oder <code>speed</code> .
<code>apiTypeFault</code>	Unzulässiger Typ für <code>index</code> oder <code>speed</code>
<code>resourceUnavailable</code>	Der Bus kann nicht geöffnet werden. Mögliche Gründe:
	- Der Bus wurde bereits geöffnet
	- ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich

## 6.2 Busgeschwindigkeit ändern

```
UPDI_setSpeed(<busHandle>, <speed>)
```

Ändert bei einem bereits geöffneten UPDI-Bus die Busgeschwindigkeit. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von UPDI\_open erhaltenen Bus-Handle.

**speed**

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Werte sind im vorherigen Kapitel „Bus öffnen“ angegeben.

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet.

**Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels setCpuClock den Takt des roloFlashs ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie setCpuClock zuerst und öffnen dann erst den Bus.
- Oder setzen Sie nach setCpuClock die Busgeschwindigkeit erneut mittels UPDI\_setSpeed .

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für speed.  
Unzulässiger Typ für busHandle oder speed

## 6.3 Busgeschwindigkeit abfragen

```
Speed = UPDI_getSpeed(<busHandle>)
```

Fragt bei einem bereits geöffneten UPDI-Bus die aktuelle Busgeschwindigkeit ab. Diese kann gleich oder geringer als die bei bei UPDI\_open bzw. UPDI\_setSpeed angegebene Busgeschwindigkeit sein.

### **Vorbedingung:**

- gültiges busHandle

### **Parameter:**

#### **busHandle**

Das von UPDI\_open erhaltene Bus-Handle.

### **Rückgabewert:**

- aktuelle Busgeschwindigkeit in Hz.

### **Exceptions:**

apiTypeFault

Unzulässiger Typ für busHandle

---

## 7 UART-Interface

### 7.1 Bus öffnen

```
busHandle = UART_open(<index>, <baudrate>, <dataBits>,  
<parity>, <stopbits>)
```

Öffnet eine der beiden UART-Schnittstellen und initialisiert die Leitungen. Die Parameter werden wie angegeben eingestellt.

**Vorbedingung:**

- keine

**Parameter:**

**index**

- 0 für UART0-Interface
- 1 für UART1-Interface

**baudrate**

Baudrate, Angabe in Hz. Die möglichen Baudraten sind von der verwendeten UART und von der CpuClock abhängig.

Andere Baudraten, zwischen den üblichen, hier aufgezählten Baudraten, sind ebenso möglich. Sie müssen zwischen den angegebenen Maximal- und Minimalwerten liegen.



*Tabelle 1: UART0-Interface bei CPU-Clock = 120 Mhz*

Nominell [Hz]	Tatsächlich [Hz]	Abweichung in %
Maximal 3750000	3750000	0
3000000	3000000	0
2000000	2000000	0
1000000	1000000	0
500000	500000	0
250000	250000	0
230400	230769	0,16
115200	115163	-0,03
76800	76824	0,03
57600	57581	-0,03
38400	38412	0,03
28800	28804	0,01
19200	19200	0
14400	14398	-0,01
9600	9600	0
4800	4800	0
2400	2400	0
Minimal 1200	1200	0

*Tabelle 2: UART0-Interface bei CPU-Clock = 24 Mhz*

<b>Nominell [Hz]</b>	<b>Tatsächlich [Hz]</b>	<b>Abweichung in %</b>
Maximal 375000	375000	0
250000	250000	0
230400	230769	0,16
115200	115384	0,16
76800	76923	0,16
57600	57692	0,16
38400	38461	0,16
28800	28846	0,16
19200	19230	0,16
14400	14388	-0,08
9600	9600	0
4800	4800	0
2400	2400	0
Minimal 1200	1200	0

*Tabelle 3: UART1-Interface bei CPU-Clock = 120 Mhz*

<b>Nominell [Hz]</b>	<b>Tatsächlich [Hz]</b>	<b>Abweichung in %</b>
Maximal 1875000	1875000	0
1000000	1000000	0
500000	500000	0
250000	250000	0
230400	230769	0,16
115200	115384	0,16
76800	76726	-0,1
57600	57581	-0,03
38400	38412	0,03
28800	28790	-0,03
19200	19206	0,03
14400	14402	0,01
9600	9600	0
4800	4800	0
2400	2400	0
Minimal 1200	1200	0

*Tabelle 4: UART1-Interface bei CPU-Clock = 24 Mhz*

Nominell [Hz]	Tatsächlich [Hz]	Abweichung in %
Maximal 375000	375000	0
250000	250000	0
230400	230769	0,16
115200	115384	0,16
76800	76923	0,16
57600	57692	0,16
38400	38461	0,16
28800	28846	0,16
19200	19230	0,16
14400	14388	-0,08
9600	9600	0
4800	4800	0
2400	2400	0
Minimal 1200	1200	0

Falls die angegebene Frequenz nicht unterstützt wird, dann wird auf den nächsten möglichen Wert auf- oder abgerundet.

#### **datubits**

- 8 für 8 Datenbits

#### **parity**

- PARITY\_NONE: kein Parity
- PARITY\_EVEN: gerade Parity
- PARITY\_ODD: ungerade Parity

#### **stopbits**

- 1 für 1 Stopbit
- 2 für 2 Stopbits

#### **Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels `set - CpuClock` den Takt des roloFlash ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie `setCpuClock` zuerst und öffnen dann erst den Bus.

**Rückgabewert:**

- ein `busHandle`. Dieses kann benutzt werden, um weitere Funktionen wie z.B. `UART_write` aufzurufen.

**Exceptions:**

`apiValueRange`  
`apiTypeFault`  
`resourceUnavailable`

Unzulässiger Wert für einen der Parameter  
Unzulässiger Typ für einen der Parameter  
Das Interfaces kann nicht geöffnet werden.  
Mögliche Gründe:  
- Das Interface wurde bereits geöffnet  
- ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich

## 7.2 Baudrate ändern

`UART_setSpeed(<busHandle>, <baudrate>)`

Ändert bei einer bereits geöffneten UART-Schnittstelle die Baudrate. Die Informationen über maximale und minimale Baudraten, sowie über Abweichungen gegenüber üblichen Baudraten, finden Sie im vorherigen Kapitel „Bus öffnen“.

**Vorbedingung:**

- gültiges `busHandle`

**Parameter:**

**`busHandle`**

Das von `UART_open` erhaltenen `busHandle`.

**`baudrate`**

Die gewünschte Baudrate in [Hz]. Die unterstützten Werte sind im vorherigen Kapitel „Bus öffnen“ angegeben.

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert gerundet.

**Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels `setCpuClock` den Takt des roloFlashs ändern, dann ändert sich auch die Baudrate der UART. Daher wird empfohlen:

- Verwenden Sie `setCpuClock` zuerst und öffnen dann erst die UART.
- Oder setzen Sie nach `setCpuClock` die Baudrate erneut mittels `UART_setSpeed`.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

`apiValueRange`  
`apiTypeFault`

Unzulässiger Wert für `speed`.  
Unzulässiger Typ für `busHandle` oder `speed`

## 7.3 Baudrate abfragen

Baudrate = `UART_getSpeed(<busHandle>)`

Fragt bei einer bereits geöffneten UART-Schnittstelle die aktuelle Baudrate ab. Diese kann gleich oder ähnlich zu der bei `UART_open` bzw. `UART_setSpeed` angegebenen Baudrate sein.

**Vorbedingung:**

- gültiges `busHandle`

**Parameter:**

**busHandle**

Das von UART\_open erhaltene Bus-Handle.

**Rückgabewert:**

- aktuelle Baudrate in Hz.

**Exceptions:**

apiTypeFault

Unzulässiger Typ für busHandle

## 7.4 Schreiben

UART\_write <busHandle>, <text>

Gibt den Text auf der UART-Schnittstelle aus. Der Programmablauf wird dann weitergeführt, wenn die Ausgabe abgeschlossen ist.

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von UART\_open erhaltenen Bus-Handle.

**text**

der Text, der ausgegeben werden soll.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiTypeFault

Unzulässiger Typ für busHandle oder Text

## 7.5 Lesen

```
data = UART_read(<busHandle>)
```

Holt inzwischen eingelesene Daten ab und stellt diese in einem Char-Array in roloBasic zur Verfügung. Die Ausführung ist nicht blockierend. Es werden alle bis zu diesem Zeitpunkt eingelesenen Daten zurückgegeben. Falls es keine Daten gibt, dann wird ein Array mit der Länge 0 zurückgegeben.

### **Vorbedingung:**

- gültiges busHandle

### **Parameter:**

#### **busHandle**

Das von UART\_open erhaltenen Bus-Handle.

### **Rückgabewert:**

- Char-Array mit den gelesenen Daten

### **Exceptions:**

OutOfMemory

Es steht nicht genug Speicher zur Verfügung, um das roloBasic-Array anzulegen

apiTypeFault

Unzulässiger Typ für busHandle oder Text

---

## 8 GPIO-Interface

### 8.1 Bus öffnen

```
busHandle = GPIO_open(<index>, <mode>, <level>)
```



Öffnet eine der GPIO-Schnittstellen und initialisiert die Leitung. Der Mode wird wie angegeben eingestellt. Gegebenenfalls ändert sich der Zustand des Pins.

**Hinweis:**

- Die Pins TDO und RTCK können nur als Eingang verwendet werden.

**Vorbedingung:**

- keine

**Parameter:**

**index**

- 2 oder **GPIO\_TMS** für den TMS-Pin (Pin 2)
- 4 oder **GPIO\_TCK** für den TCK-Pin (Pin 4)
- 6 oder **GPIO\_TDO** für den TDO-Pin (Pin 6)
- 7 oder **GPIO\_RTCK** für den RTCK-Pin (Pin 7)
- 8 oder **GPIO\_TDI** für den TDI-Pin (Pin 8)
- 9 oder **GPIO\_GNDDDET** für den GND-Detect-Pin (Pin 9)
- 10 oder **GPIO\_RST** für den Reset-Pin (Pin 10)

**mode**

- **PIN\_INPUT**: diesen Pin als Eingang benutzen
- **PIN\_ACTIVELOW**: diesen Pin als Ausgang nutzen, jedoch nur low-aktiv treiben.
- **PIN\_ACTIVEHIGH**: diesen Pin als Ausgang nutzen, jedoch nur high-aktiv treiben.
- **PIN\_PUSHPULL**: diesen Pin als Ausgang benutzen

**level**

Dieser Wert darf bei mode = PIN\_INPUT nicht angegeben werden. Für die anderen Modi gibt der Wert den Ausgangszustand für den Pin an.

**Rückgabewert:**

- ein busHandle. Dieses kann benutzt werden, um weitere Funktionen wie z.B. GPIO\_set aufzurufen.

### **Exceptions:**

apiValueRange	Unzulässiger Wert für einen der Parameter
apiTypeFault	Unzulässiger Typ für einen der Parameter
functionNotSupported	Die Pins TDO und RTCK können nur als Input verwendet werden
BadArgumentCount	Die Anzahl der Argumente stimmt nicht. Tritt auf, wenn: - als Mode „PIN_INPUT“ und ein level angegeben wurde. - als Mode etwas anderes als „PIN_INPUT“ angegeben wurde und die Angabe für „level“ fehlt.
resourceUnavailable	Das Interface kann nicht geöffnet werden. Mögliche Gründe: - Das Interface wurde bereits geöffnet - ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich

## 8.2 Mode ändern

GPIO\_mode(<busHandle>, <mode>, <level>)

Ändert den Mode eines GPIO-Pins. Der Mode wird wie angegeben eingestellt. Gegebenenfalls ändert sich der Zustand des Pins.

### **Vorbedingung:**

- gültiges busHandle

### **Parameter:**

#### **busHandle**

Das von GPIO\_open erhaltenen Bus-Handle.

#### **mode**

- **PIN\_INPUT**: diesen Pin als Eingang benutzen
- **PIN\_ACTIVELOW**: diesen Pin als Ausgang nutzen, jedoch nur low-aktiv treiben.

- **PIN\_ACTIVEHIGH**: diesen Pin als Ausgang nutzen, jedoch nur high-aktiv treiben.
- **PIN\_PUSHPULL**: diesen Pin als Ausgang benutzen

#### **level**

Dieser Wert darf bei mode = PIN\_INPUT nicht angegeben werden. Für die anderen Modi gibt der Wert den Ausgangszustand für den Pin an.

#### **Rückgabewert:**

- keiner (Prozedur)

#### **Exceptions:**

apiValueRange  
apiTypeFault  
functionNotSupported  
BadArgumentCount

Unzulässiger Wert für einen der Parameter  
Unzulässiger Typ für einen der Parameter  
Die Pins TDO und RTCK können nur als Input verwendet werden  
Die Anzahl der Argumente stimmt nicht.  
Tritt auf, wenn:  
- als Mode „PIN\_INPUT“ und ein level angegeben wurde.  
- als Mode etwas anderes als „PIN\_INPUT“ angegeben wurde und die Angabe für „level“ fehlt.

## **8.3 Schreiben**

```
GPIO_set(<busHandle>, <level>)
```

Setzt den GPIO-Pin auf den angegebenen Level.

#### **Vorbedingung:**

- gültiges busHandle

#### **Parameter:**

##### **busHandle**

Das von GPIO\_open erhaltenen Bus-Handle.

**level**

Ausgangszustand für den Pin

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault  
functionNotSupported

Unzulässiger Wert für level  
Unzulässiger Typ für eine der Parameter  
Der Pin wird als Eingang genutzt, daher ist die  
Funktion nicht möglich

## 8.4 Lesen

GPIO\_get(<busHandle>)

Liest den GPIO-Pin ein.

**Vorbedingung:**

- gültiges busHandle

**Hinweis:**

Die Funktion steht nur zur Verfügung, wenn roloFlash die Leitung nicht selber treibt:

- bei mode = PIN\_INPUT
- bei mode = PIN\_ACTIVELOW und level = 1
- bei mode = PIN\_ACTIVEHIGH und level = 0

**Parameter:**

**busHandle**

Das von GPIO\_open erhaltenen Bus-Handle.

**Rückgabewert:**

- Ausgelesener Pin

**Exceptions:**

apiTypeFault  
functionNotSupported

Unzulässiger Typ für busHandle  
Der Pin kann nicht ausgelesen werden, da roloFlash den Pin aktiv treibt, siehe obigen Hinweis.

---

---

## 9 Target allgemein

Das Target kann sich in folgenden Betriebsmodi befinden:

**RunMode**

Target läuft ganz normal, als ob roloFlash nicht angeschlossen wäre.

**ProgramMode**

Target kann programmiert werden.

Die Prozedur setTargetMode wechselt den Betriebsmodus.

Andere Prozeduren bzw. Funktionen sind auf einen bestimmten Modus angewiesen. In diesem Fall steht das in der jeweiligen Beschreibung.

### 9.1 getTargetPresent

```
value = getTargetPresent(<targetHandle>)
```

Ermittelt, ob ein Target angeschlossen ist. Der Modus bleibt dabei unverändert. Es findet auf jeden Fall eine Kommunikation mit dem Target statt, so daß man eine aktuelle Information erhält.

**Hinweis Atmel ISP-Bus:**

Falls das Target im RunMode ist, dann wird bei dem Target vorübergehend ein Reset angelegt und es in den ProgramMode versetzt. Nach der Abfrage wird der Reset aufgehoben und der RunMode erreicht. Ein evtl. auf dem Target laufendes Programm wird dadurch neu gestartet.

Falls das Target schon im ProgramMode ist, dann wird ebenso eine Abfrage gestartet. Das Target bleibt im ProgramMode.

**Hinweis Atmel PDI-Bus und Atmel UPDI-Bus:**

Unabhängig davon, ob das Target im RunMode oder ProgramMode ist, wird über den PDI-Bus/UPDI-Bus eine Abfrage gestartet. Das Target verbleibt im jeweiligen Modus. Ein Reset findet nicht statt.

**Anmerkung:**

Bei roloFlash sollte immer ein Target angeschlossen sein, weil sonst roloFlash nicht mit Energie versorgt wäre. Die Funktion ist hauptsächlich für Programmiergeräte gedacht, die über eine eigene Energieversorgung verfügen.

Des weiteren ist es denkbar, daß roloFlash auf etwas anderes als ein Target aufgesteckt wird. Daher baut diese Funktion tatsächlich eine Kommunikation mit dem Target auf.

**Vorbedingung:**

- gültiges Target-Handle

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**Rückgabewert:**

0 = kein Target gefunden

1 = Target gefunden

**Exceptions:**

invalidHandle

Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.

apiTypeFault

Unzulässiger Typ des Target-Handles

## 9.2 setTargetMode

setTargetMode <targetHandle, targetMode>

Bringt das Target und roloFlash in den angegebenen Modus.

**Vorbedingung:**

- gültiges targetHandle

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**targetMode**

Angabe des gewünschten Modus:

**PROGRAMMODE:** Dieser Modus ist Voraussetzung für die meisten weiteren Funktionen mit dem Target, insbesondere für das Beschreiben des Flashspeichers. Dabei kann je nach Targetfamilie das Target gestoppt werden.

**RUNMODE:** Das Target läuft. Eine etwaige auf dem Target befindliche Software wird ausgeführt.

**Rückgabewert:**

- keiner (Prozedur)

#### **Hinweis JTAG-Bus:**

- `programMode`: Hat keinen Einfluss darauf, ob das Target gerade läuft oder steht. Es werden hier Initialisierungen für den Zugriff auf den Speicher des Targets durchgeführt.
- `runMode`: Startet das Target nur, wenn vorher keine Schreibzugriffe auf den Flashspeicher stattgefunden haben.

#### **Hinweis Atmel ISP-Bus:**

- `programMode`: Falls das Target im `RunMode` ist, dann wird das Target in den „Programming Enable Mode“ geschaltet und im Reset gehalten. Ein evtl. auf dem Target laufendes Programm wird dadurch angehalten.
- `runMode`: Der „Programming Enable Mode“ wird aufgehoben, der Reset wird weggenommen. Das Target läuft damit nach einem Reset los.

#### **Hinweis Atmel PDI-Bus:**

- `programMode`: Hat keinen Einfluss darauf, ob das Target gerade läuft oder steht. Es werden hier Initialisierungen für den Zugriff über den PDI-Bus durchgeführt.
- `runMode`: Der PDI-Takt wird gestoppt, folgend wird der „Programming Mode“ beendet. Das Target läuft nach einem Reset los.

#### **Hinweis Atmel UPDI-Bus:**

- `programMode`: Falls das Target im `RunMode` ist, dann wird das Target in den „Programming Enable Mode“ geschaltet und im Reset gehalten. Ein evtl. auf dem Target laufendes Programm wird dadurch angehalten.
- `runMode`: Der „Programming Enable Mode“ wird aufgehoben, der Reset wird weggenommen. Das Target läuft damit nach einem Reset los.
- Befindet sich das Target in dem „Programming Enable Mode“ und `roloFlash` wird abgezogen, dann verbleibt das Target in diesem Mode. Ein auf dem Target befindliches Programm startet nicht bis die Stromversorgung für das Target kurzzeitig unterbrochen wird. Das Starten des Targets kann erzwungen werden, indem vor dem Entfernen des `roloFlash` `setTargetMode` mit dem Parameter `runMode` aufgerufen wird. Alternativ kann auch das `targetHandle` mittels `closeTarget` geschlossen werden.



**Exceptions:**

targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für das Target-Handle

### 9.3 restartTarget

RestartTarget <targetHandle>

Startet das Target neu und nimmt wieder denselben Zustand ein:

**RunMode**

Es wird kurzzeitig der Reset aktiviert, dann deaktiviert. Das Target läuft daher vom Anfang los. Der RunMode wird beibehalten.

**ProgramMode**

Es wird ebenso ein Reset durchlaufen, dann wieder der ProgramMode hergestellt. Zwischenzeitlich kann eine evtl. auf dem Target befindliche Firmware bereits für kurze Zeit losgelaufen sein.

Es wird empfohlen, dieses Kommando nur einzusetzen, wenn dieses entweder nicht kritisch ist oder sich noch keine Firmware auf dem Target befindet.

**Hinweis JTAG-Bus:**

Da beim JTAG-Bus aus Sicht von roloFlash keine Resetleitung vorhanden ist, steht dieser Befehl nicht zur Verfügung. Sie können jedoch die Resetleitung als GPIO öffnen und selbstständig einen Reset auslösen. Da dieses für das Modul, welches für die JTAG-Kommunikation mit dem Target zuständig ist, nicht ersichtlich ist, verbleibt dieses in seinem Zustand, der dann evtl. nicht mehr zum Zustand des Targets passt. Für den Fall, daß Sie beim Target einen Reset auslösen und anschließend weiter mit dem

Target arbeiten möchten, schließen Sie bitte das Target-Handle zum Target und fordern Sie ein neues Target-Handle an:

```
! Reset aktivieren:  
handle = GPIO_open(GPIO_RST, PIN_PUSH_PULL, 0)  
! 100 ms im Reset bleiben:  
delay 100  
! GPIO schliessen, damit wird der Reset aufgehoben:  
closeBus handle  
! Schliessen und Neuoeffnen des Target-Handles zum  
! Zurücksetzen des Moduls für das Target im roloFlash:  
closeTarget targetHandle  
targetHandle = getTargetHandle(<bushandle>, <index>,  
<family>)
```

#### **Hinweis Atmel ISP-Bus:**

Der „Programming Enable Mode“ wird aufgehoben, der Reset wird weggenommen. Das Target läuft damit nach einem Reset los.

#### **RunMode**

Es wird kurzzeitig (100 ms) der Reset aktiviert, dann deaktiviert. Das Target läuft daher vom Anfang los. Der RunMode wird beibehalten.

#### **ProgramMode**

Der Reset wird kurzzeitig (3 ms) aufgehoben, dann wieder der ProgramMode hergestellt. Zwischenzeitlich kann eine evtl. auf dem Target befindliche Firmware bereits für kurze Zeit losgelaufen sein.

#### **Anwendungsbeispiel für Targets mit Atmel ISP-Interface:**

Die Prozedur wird benötigt, wenn man z. B. Fuses auf dem Target ändert und die Änderungen sofort aktiviert werden sollen. Das gilt insbesondere für das Aktivieren eines Quarzes für das Target, was dann anschließend eine höhere Programmiergeschwindigkeit ermöglicht:

```
! Quarz aktivieren, damit höhere  
! Programmiergeschwindigkeit möglich  
writeBits(targetHandle, FUSES_LOW, value)  
  
! Durch restartTarget die Änderung aktivieren  
restartTarget targetHandle  
  
ISP_setSpeed(bushandle, 1000000) ! z.B. 1 MHz  
writeFileToTarget ...
```

### **Hinweis Atmel PDI-Bus:**

Der Reset ist zwar Teil des PDI-Busses, wird aber bei PDI nicht als Reset genutzt. Folglich kann der Bus benutzt werden, ohne das Target im Reset zu halten.

### **RunMode**

Es wird kurzzeitig (100 ms) der Reset aktiviert, dann deaktiviert. Das Target läuft daher vom Anfang los. Der RunMode wird beibehalten.

### **ProgramMode**

Der PDI-Bus wird deaktiviert, ein Reset ausgelöst (100 ms), anschließend der PDI-Bus aktiviert und der ProgramMode wieder hergestellt. Das Target läuft vom Anfang los.

### **Hinweis UPDI-Bus:**

Da beim UPDI-Bus keine Resetleitung vorhanden ist, steht dieser Befehl nicht zur Verfügung. Falls das Resetsignal auf dem Programmierstecker zur Verfügung stehen sollte, dann können Sie die Resetleitung als GPIO öffnen und selbstständig einen Reset auslösen. Da dieses für das Modul, welches für die UPDI-Kommunikation mit dem Target zuständig ist, nicht ersichtlich ist, verbleibt dieses in seinem Zustand, der dann evtl. nicht mehr zum Zustand des Targets passt. Für den Fall, daß Sie beim Target einen Reset auslösen und anschließend weiter mit dem Target arbeiten möchten, schließen Sie bitte das Target-Handle zum Target und fordern Sie ein neues Target-Handle an

### **Vorbedingung:**

- gültiges Target-Handle

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

---

## 10 Target-Memorymap lesen/schreiben

Für verschiedene Speicherarten in den Targets unterstützt roloFlash eine sogenannte MemoryMap. Diese kann je nach Target und Speicherart Informationen über verschiedene Eigenschaften (sog. /Properties/) des Speichers zur Verfügung stellen bzw. diese Eigenschaften können hier eingestellt werden. Manche Eigenschaften müssen vor dem Flashen richtig gesetzt werden. Oft finden Sie die benötigten Werte in der Datenbank.

Ein guter Ansatzpunkt dazu sind die Beispielskripte.

### 10.1 Wert in ein Property einer Speicherart schreiben

```
setMemoryMap <targetHandle>, <memType>, <memProperty>  
<value>
```

Setzt für den angegebenen Speichertyp das entsprechende Property auf den angegebenen Wert.

**Vorbedingung:**

- gültiges Target-Handle

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**memType**

die gewählte Speicherart:

**FLASH:** Flash-Speicher

**RAM:** RAM-Speicher

**EEPROM:** EEPROM-Speicher

**memProperty**

das gewählte Property:

**MEM\_STARTADDR:** Startadresse des Speichers

**MEM\_SIZE:** Größe des Speichers in Bytes

**MEM\_PAGESIZE:** für bestimmte Targets: Größe einer Speicherseite

**value**

der zu setzende Wert

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
FunctionNotSupported	Unzulässige Kombination von MemType und Property
apiTypeFault	Unzulässiger Typ für einen der Parameter.
ValueNotAllowed	Unzulässiger Wert für value

## 10.2 Wert aus einem Property einer Speicherart lesen

Value = getMemoryMap(<targetHandle>, <memType>, <mem-Property>)

Setzt für den angegeben Speichertyp das entsprechende Property auf den angegebenen Wert.

### Vorbedingung:

- gültiges targetHandle

### Parameter:

#### **targetHandle**

Das Target-Handle auf das anzusprechende Target

#### **memType**

die gewählte Speicherart:

**FLASH:** Flash-Speicher

**RAM:** RAM-Speicher

**EEPROM:** EEPROM-Speicher

#### **memProperty**

das gewählte Property:

**MEM\_STARTADDR:** Startadresse des Speichers

**MEM\_SIZE:** Größe des Speichers in Bytes

**MEM\_PAGESIZE:** für bestimmte Targets: Größe einer Speicherseite

### Rückgabewert:

- gelesener Wert

**Exceptions:**

targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
FunctionNotSupported	Unzulässige Kombination von MemType und Property
apiTypeFault	Unzulässiger Typ für einen der Parameter.
valueUnknown	Der Wert kann nicht ermittelt werden

### 10.3 clearMemoryLayout

clearMemoryLayout

Löscht ein bereits vorhandenes Speicherlayout (Flash- und EEPROM-Layout).

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
invalidHandle	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für das Target-Handle.

## 11 Loader

Manche Controllerfamilien werden mit Hilfe eines Loaders angesprochen. Dabei muß vor dem Nutzen bestimmter Funktionen erst ein Loader in den RAM-Speicher des Targets gebracht und gestartet werden. Je nach Target ist der Loader bereits innerhalb des roloFlash vorhanden oder muss auf der SD-Karte gespeichert sein:

Controllerfamilie	Loader	Funktion, die den Loader benötigen
STM32	<extern>	eraseFlash writeFileToTarget mit memType = Flash writeRawDataToTarget mit memType = Flash
Atmel ISP	<kein Loader>	
Atmel PDI	<kein Loader>	

Die Verwendung des passenden Loaders passiert automatisch, sobald eine Funktion benutzt wird, die einen Loader benötigt (siehe obige Tabelle). In diesem Fall bringt roloFlash den Loader auf das Target und startet diesen direkt vor dem Ausführen der gewünschten Funktion.

Daraus folgt:

- Sie brauchen sich um den Loader nicht zu kümmern, er muß lediglich auf der microSD-Karte vorhanden sein (falls es ein externer Loader ist).
- Eine eventuell laufende Applikation auf dem Target wird gestoppt.
- Nach einer solchen Funktion kann der Loader aktiv bleiben und steht zur Verfügung, falls noch weitere Funktionen den Loader benötigen.
- Bestimmte Teile des RAMs werden verändert.



---

## 12 Target löschen, schreiben, lesen und verifizieren

### 12.1 EraseFlash

eraseFlash

Löscht das gesamte Flash des Targets. Bei manchen Targets wird dabei automatisch auch das EEPROM gelöscht (z.B. Atmel-Fuse „EESAVE“). Informationen dazu finden Sie im jeweiligen Datenblatt des Targets.

#### **Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

#### **Parameter:**

##### **targetHandle**

Das Target-Handle auf das anzusprechende Target

#### **Rückgabewert:**

- keiner (Prozedur)

#### **Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für das Target-Handle.

## 12.2 writeFileToTarget

```
writeFileToTarget <targetHandle>, <filesystem>, <file-  
name>, <fileformat>, <memType>, <verify>, <startAddr>
```

Schreibt eine Datei in den Speicher des Targets.

### **Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

### **Parameter:**

#### **targetHandle**

Das Target-Handle auf das anzusprechende Target

#### **filesystem**

Der Parameter wird ignoriert und sollte mit 0 angegeben werden.

#### **filename**

Es gelten die Bedingungen für Dateinamen, siehe Kapitel „Dateien“.

#### **fileformat**

Gibt an, in welchem Format die Datei vorliegt. Mögliche Werte sind:

**HEX:** Intel-HEX-Format (ASCII-Datei)

**RAW:** Raw-Format (Binärdatei mit Rohdaten ohne Adreßangabe)

#### **memType**

Gibt an, welcher Speicher beschrieben werden soll. Die Angabe ist für die jeweilige Targetfamilie spezifisch und ist in den jeweiligen Kapiteln beschrieben.

#### **verify**

Gibt an, ob ein Verify durchgeführt werden soll. Mögliche Werte sind:

**WRITEONLY:** Schreiben ohne Verify

**VERIFYONLY:** Die Daten werden nur verifiziert (kein Schreiben auf dem Target)

## **WRITEVERIFY:** Schreiben und Verifizieren

### **startAddr**

(optional). Der Parameter ist ausschließlich für das Raw-Format vorgesehen. Da im Raw-Format in der Datei keine Adresse angegeben ist, muß hiermit die Startadresse spezifiziert werden.

### **Rückgabewert:**

- keiner (Prozedur)

### **Hinweis zu Verify = WRITEVERIFY**

Die geschriebenen Daten werden vom Target zurückgelesen und mit den aus der Datei gelesenen Daten im roloFlash verglichen. Die Daten werden dazu nicht ein zweites Mal von der microSD-Karte gelesen und dekodiert. Etwaige Lesefehler von der microSD-Karte werden so nicht erkannt. Allerdings werden bei Hex-Dateien auch die CRC-Werte ausgelesen und überprüft, so daß Lesefehler dementsprechend unwahrscheinlich sind.

Wenn Sie die Sicherheit weiter erhöhen wollen, dann benutzen Sie bitte zwei Aufrufe: ein Aufruf mit verify = WRITEONLY und den zweiten Aufruf mit verify = VERIFYONLY. Dieses Vorgehen kann länger dauern als ein einziger Aufruf mit verify = WRITEVERIFY.

**Exceptions:**

targetMemoryLayout,  
hexFileSize,  
hexFileCRC  
targetWrongMode  
targetCommunication

Siehe Kapitel „Exceptions des roloFlash“.

targetError  
apiTypeFault  
invalidHandle

Target ist nicht im "ProgramMode".  
Die Kommunikation mit dem Target funktioniert nicht.

targetVerify

Es gibt einen Fehler auf dem Target  
Unzulässiger Typ für einen der Parameter.  
Das Target-Handle ist schon geschlossen, oder  
der Bus wurde schon geschlossen.

Beim Verify wurden andere Daten gelesen.

Mögliche Ursachen:

- Kommunikationsprobleme
- Zu hohe Datenrate
- Das Target ist vorher nicht gelöscht worden  
(betrifft vornehmlich Flash-Speicher)

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 12.3 readFromFileFromTarget

readFromFileFromTarget <targetHandle>, <filesystem>, <file-  
name>, <fileformat>, <memType>, <startAddr>, <length>

Liest aus dem Speicher des Targets, erzeugt eine neue Datei und schreibt  
die Daten im angegebenen Format in die Datei.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target.

**filesystem**

Der Parameter wird ignoriert und sollte mit 0 angegeben werden.

**filename**

Es gelten die Bedingungen für Dateinamen, siehe Kapitel „Dateien“. Falls die Datei schon existieren sollte, dann wird sie überschrieben.

**fileformat**

Gibt an, in welchem Format die Daten geschrieben werden sollen. Mögliche Werte sind:

**HEX:** Intel-HEX-Format (ASCII-Datei)

**RAW:** Raw-Format (Binärdatei mit Rohdaten ohne Adreßangabe)

**memType**

Gibt an, welcher Speicher gelesen werden soll. Die Angabe ist für die jeweilige Targetfamilie spezifisch und ist in den jeweiligen Kapiteln beschrieben.

**startAddr**

Bestimmt die erste Adresse, ab der gelesen werden soll.

**length**

Bestimmt die Anzahl in Bytes, die gelesen werden sollen.

**Rückgabewert:**

- keiner (Prozedur)

**Hinweis zu Verify beim Lesen**

Um ein Verify, ähnlich wie beim Schreiben ins Target, zu erreichen, kann man die gelesene Datei anschließend mittels `writeFileToTarget` mit `verify = verifyOnly` überprüfen.

**Exceptions:**

targetMemoryLayout, hexFileSize, hexFileCRC	Siehe Kapitel „Exceptions des roloFlash“.
targetWrongMode targetCommunication	Target ist nicht im "ProgramMode". Die Kommunikation mit dem Target funktioniert nicht.
targetError apiTypeFault invalidHandle	Es gibt einen Fehler auf dem Target Unzulässiger Typ für einen der Parameter. Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
<diverse Exceptions des Dateisystems>	Siehe Kapitel „Exceptions des Dateisystems“.

## 12.4 writeRawDataToTarget

writeRawDataToTarget <targetHandle>, <dataArray>, <mem-Type>, <verify>, <startAddr>

Schreibt ein Daten-Array aus dem roloBasic in den Speicher des Targets.

**Vorbedingung:**

- gültiges targetHandle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**dataArray**

Ein Char-Array mit den zu schreibenden Daten.

**memType**

Gibt an, welcher Speicher beschrieben werden soll. Die Angabe ist für die jeweilige Targetfamilie spezifisch und ist in den jeweiligen Kapiteln beschrieben.

## verify

Gibt an, ob ein Verify durchgeführt werden soll. Mögliche Werte sind:

**WRITEONLY**: Schreiben ohne Verify

**VERIFYONLY**: Die Daten werden nur verifiziert (kein Schreiben auf dem Target)

**WRITEVERIFY**: Schreiben und Verifizieren

## startAddr

Die Adresse, an die Daten im Target geschrieben werden sollen.

## Rückgabewert:

- keiner (Prozedur)

## Exceptions:

targetMemoryLayout  
targetWrongMode  
targetCommunication

Siehe Kapitel „Exceptions des roloFlash“.

Target ist nicht im "ProgramMode".

Die Kommunikation mit dem Target funktioniert nicht.

targetError  
apiTypeFault  
invalidHandle

Es gibt einen Fehler auf dem Target

Unzulässiger Typ für einen der Parameter.

Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.

targetVerify

Beim Verify wurden andere Daten gelesen.

Mögliche Ursachen:

- Kommunikationsprobleme

- Zu hohe Datenrate

- Das Target ist vorher nicht gelöscht worden (betrifft vornehmlich Flash-Speicher)

<diverse Exceptions des Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 12.5 readRawDataFromTarget

```
dataArray = readRawDataFromTarget( <targetHandle>,  
<memType>, <startAddr>, <length>
```

Liest aus dem Speicher des Targets, erzeugt ein Char-Array im Basic und füllt dieses Array mit den gelesenen Daten.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target.

**memType**

Gibt an, welcher Speicher gelesen werden soll. Die Angabe ist für die jeweilige Targetfamilie spezifisch und ist in den jeweiligen Kapiteln beschrieben.

**startAddr**

Bestimmt die erste Adresse, ab der gelesen werden soll.

**length**

Bestimmt die Anzahl in Bytes, die gelesen werden sollen.

**Rückgabewert:**

- Char-Array mit den ausgelesenen Daten

**Hinweis zu Verify beim Lesen**

Um ein Verify, ähnlich wie beim Schreiben ins Target, zu erreichen, kann man die gelesene Datei anschließend mittels `writeRawDataToTarget` mit `verify = VERIFYONLY` überprüfen.



### **Exceptions:**

OutOfMemory	Es steht nicht genug Speicher zur Verfügung, um das Basic Array anzulegen
targetMemoryLayout	Siehe Kapitel „Exceptions des roloFlash“.
targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
targetError	Es gibt einen Fehler auf dem Target
apiTypeFault	Unzulässiger Typ für einen der Parameter.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
<diverse Exceptions des Dateisystems>	Siehe Kapitel „Exceptions des Dateisystems“.

---

## 13 Target STM32

Momentan werden folgende Unterfamilien unterstützt:

- STM32F1
- STM32F2
- STM32F3
- STM32F4

Es werden alle Funktionen der Kapitel „Target allgemein“ bis „Target löschen, schreiben, lesen und verifizieren“ inklusive aller Unterkapitel unterstützt.

Es kommt ein externer gespeicherter Loader zum Einsatz, wenn folgende Funktionen aufgerufen werden:

- eraseFlash
- writeFileToTarget mit memType = FLASH
- writeRawDataToTarget mit memType = FLASH

Unterstützte memTypes beim Schreiben:

- Flash
- Ram

Unterstützte memTypes beim Lesen:

- Flash

- Ram
- ReadMemory (für Zugriff auf RAM- und Flash-Speicher)

---

## 14 Target Atmel AVR (ISP-Interface)

Es werden alle Funktionen der Kapitel „Target allgemein“ bis „Target löschen, schreiben, lesen und verifizieren“ inklusive aller Unterkapitel unterstützt.

Es wird kein Loader verwendet.

Unterstützte memTypes beim Schreiben:

- Flash
- EEPROM

Unterstützte memTypes beim Lesen:

- Flash
- EEPROM

### 14.1 getSignature

```
s = getSignature(<targetHandle>)
```

Liest die Signature des Targets. Anhand dieser lassen sich die verschiedenen Controller unterscheiden.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das targetHandle auf das anzusprechende Target

**Rückgabewert:**

Ausgelesene Signature. Die Signature wird in einem Byte-Array mit 3 Bytes geliefert. Sie können die Signature mit einer Signature aus der Datenbank vergleichen.

**Hinweis:**

Zur besseren Kompatibilität zu anderen Atmel-Controller kann gleichwertig auch die Funktion `getDeviceId` verwendet werden.

**Exceptions:**

<code>targetWrongMode</code>	Target ist nicht im "ProgramMode".
<code>targetCommunication</code>	Die Kommunikation mit dem Target funktioniert nicht.
<code>invalidHandle</code>	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
<code>apiTypeFault</code>	Unzulässiger Typ für Target-Handle.

## 14.2 readBits

`values = readBits(<targetHandle>, <index>)`

Liest die angegebenen Fuses bzw. Lock-Bits aus.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**index**

Gibt an, welche Fuses bzw. Lock-Bits gelesen werden sollen. Es gibt dazu die Konstanten FUSES\_LOW, FUSES\_HIGH, FUSES\_EXT und LOCK\_BITS.

Bei Controllern, die keine Extended-Fuses haben, ist der ausgelesene Wert bei FUSES\_EXT unbestimmt (es wird keine Exception erzeugt).

**Rückgabewert:**

Ausgelesene Fuses bzw. Lock-Bits.

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index.
invalidHandle	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

### 14.3 writeBits

writeBits <targetHandle>, <index>, <values>

Schreibt die angegebenen Fuses bzw. Lock-Bits.

**Vorsicht:**

- Setzen Sie die Lock-Bits erst, nachdem Sie alle anderen Zugriffe auf den Chip ausgeführt haben.
- Falls Sie einen durch Lock-Bits gesperrten Chip bearbeiten wollen, führen Sie als erstes ein eraseFlash aus. Dieses setzt auch die Lock-Bits wieder zurück.

**Vorbedingung:**

- gültiges Target-Handle

- das Target muß im ProgramMode sein.

### **Hinweis:**

Manche Änderungen der Fuses sind erst nach einem Reset wirksam bzw. mittels `readBits` sichtbar. Näheres finden Sie dazu im Handbuch des jeweiligen Targets. Sie können dazu das Kommando `restartTarget` verwenden.

### **Parameter:**

#### **targetHandle**

Das Target-Handle auf das anzusprechende Target

#### **index**

Gibt an, welche Fuses bzw. Lock-Bits beschrieben werden sollen. Es gibt dazu die Konstanten `FUSES_LOW`, `FUSES_HIGH`, `FUSES_EXT` und `LOCK_BITS`.

Bei Controllern, die keine Extended-Fuses haben, findet bei `FUSES_EXT` kein Schreibvorgang statt (es wird keine Exception erzeugt).

#### **values**

Zu schreibende Werte.

### **Rückgabewert:**

- keiner (Prozedur)

### **Exceptions:**

<code>targetWrongMode</code>	Target ist nicht im "ProgramMode".
<code>targetCommunication</code>	Die Kommunikation mit dem Target funktioniert nicht.
<code>apiValueRange</code>	Unzulässiger Wert für <code>index</code> oder <code>value</code>
<code>invalidHandle</code>	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
<code>apiTypeFault</code>	Unzulässiger Typ für einen der Parameter.

## 14.4 setExtendedAddressMode

setExtendedAddressMode <targetHandle>, <value>

Für Controller mit 256 kB Flash oder mehr ist der normale Befehlssatz zum Programmieren über das ISP-Interface nicht mehr ausreichend. Es wird dann ein Extended Address Mode benötigt.

Beim Einstellen der Größe des Flash-Speichers (mittels setMemoryMap mit memType = flash und memProperty = mm\_size) wird der Wert automatisch passend gesetzt.

Mit dieser Funktion kann der Wert überschrieben werden.

### **Vorbedingung:**

- gültiges targetHandle
- das Target muß im ProgramMode sein.

### **Parameter:**

#### **targetHandle**

Das Target-Handle auf das anzusprechende Target

#### **value**

0: Extended address mode nicht verwenden  
sonst: Extended address mode verwenden

### **Rückgabewert:**

- keiner (Prozedur)

### **Exceptions:**

targetWrongMode  
invalidHandle

apiTypeFault

Target ist nicht im "ProgramMode".  
Das Target-Handle ist schon geschlossen oder  
der Bus wurde schon geschlossen.  
Unzulässiger Typ für einen der Parameter.

## 15 Target Atmel AVR (PDI-Interface)

Es werden alle Funktionen der Kapitel „Target allgemein“ bis „Target löschen, schreiben, lesen und verifizieren“ inklusive aller Unterkapitel unterstützt.

Es wird kein Loader verwendet.

Unterstützte memTypes beim Schreiben:

- Flash
- Eeprom

Unterstützte memTypes beim Lesen:

- Flash
- Eeprom

### 15.1 getDeviceId

```
id = getDeviceId(<targetHandle>)
```

Liest die Device-ID des Targets. Anhand dieser lassen sich die verschiedenen Controller unterscheiden.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**TargetHandle**

Das Target-Handle auf das anzusprechende Target

**Rückgabewert:**

Ausgelesene Device-ID. Die Device-ID wird in einem Byte-Array mit 3 Bytes geliefert. Sie können die Device-ID mit einer Device-ID aus der Datenbank vergleichen.

**Hinweis:**

Zur besseren Kompatibilität zu anderen Atmel-Controller kann gleichwertig auch die Funktion `getSignature` verwendet werden.

**Exceptions:**

<code>targetWrongMode</code>	Target ist nicht im "ProgramMode".
<code>targetCommunication</code>	Die Kommunikation mit dem Target funktioniert nicht.
<code>invalidHandle</code>	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
<code>apiTypeFault</code>	Unzulässiger Typ für TargetHandle.

## 15.2 readBits

```
values = readBits(<targetHandle>, <index>)
```

Liest die angegebenen Fuses bzw. Lock-Bits aus.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**index**

- 0:** Fuse-Byte 0
- 1:** Fuse-Byte 1



- 2: Fuse-Byte 2
- 3: <nicht zulässig>
- 4: Fuse-Byte 4
- 5: Fuse-Byte 5
- 6: <nicht zulässig>
- 7: Lock-Bits

Hinweis: für die Lock-Bits kann auch die Konstante LOCK\_BITS verwendet werden.

### **Rückgabewert:**

Ausgelesene Fuses bzw. Lock-Bits.

### **Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

## 15.3 writeBits

writeBits <targetHandle>, <index>, <values>

Schreibt die angegebenen Fuses bzw. Lock-Bits.

### **Vorsicht:**

- Setzen Sie die Lock-Bits erst, nachdem Sie alle anderen Zugriffe auf den Chip ausgeführt haben.
- Falls Sie einen durch Lock-Bits gesperrten Chip bearbeiten wollen, führen Sie als erstes ein eraseFlash aus. Dieses setzt auch die Lock-Bits wieder zurück.

### **Vorbedingung:**

- gültiges Target-Handle

- das Target muß im ProgramMode sein.

**Hinweis:**

Manche Änderungen der Fuses sind erst nach einem Reset wirksam bzw. mittels `readBits` sichtbar. Näheres finden Sie dazu im Handbuch des jeweiligen Targets. Sie können dazu das Kommando `restartTarget` verwenden.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**index**

- 0: Fuse-Byte 0
- 1: Fuse-Byte 1
- 2: Fuse-Byte 2
- 3: <nicht zulässig>
- 4: Fuse-Byte 4
- 5: Fuse-Byte 5
- 6: <nicht zulässig>
- 7: Lock-Bits

Hinweis: für die Lock-Bits kann auch die Konstante `LOCK_BITS` verwendet werden.

**values**

Zu schreibende Werte.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index oder value
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

---

## 16 Target Atmel UPDI (UPDI-Interface)

Es werden alle Funktionen der Kapitel „Target allgemein“ bis „Target löschen, schreiben, lesen und verifizieren“ inklusive aller Unterkapitel unterstützt.

Es wird kein Loader verwendet.

Unterstützte memTypes beim Schreiben:

- Flash
- Eeprom
- userSignature

Unterstützte memTypes beim Lesen:

- Flash
- Eeprom
- userSignature

### 16.1 getDeviceId

```
id = getDeviceId(<targetHandle>)
```

Liest die Device-ID des Targets. Anhand dieser lassen sich die verschiedenen Controller unterscheiden.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**TargetHandle**

Das Target-Handle auf das anzusprechende Target

**Rückgabewert:**

Ausgelesene Device-ID. Die Device-ID wird in einem Byte-Array mit 3 Bytes geliefert. Sie können die Device-ID mit einer Device-ID aus der Datenbank vergleichen.

**Hinweis:**

Zur besseren Kompatibilität zu anderen Atmel-Controller kann gleichwertig auch die Funktion `getSignature` verwendet werden.

**Exceptions:**

<code>targetWrongMode</code>	Target ist nicht im "ProgramMode".
<code>targetCommunication</code>	Die Kommunikation mit dem Target funktioniert nicht.
<code>invalidHandle</code>	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
<code>apiTypeFault</code>	Unzulässiger Typ für TargetHandle.

## 16.2 readBits

```
values = readBits(<targetHandle>, <index>)
```

Liest die angegebenen Fuses bzw. Lock-Bits aus.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**index** (aus Hersteller Dokumentation zu ATtiny417/817)

- 0: WDTCFG
- 1: BODCFG
- 2: OSCCFG
- 3: <nicht zulässig>
- 4: TCD0CFG
- 5: SYSCFG0
- 6: SYSCFG1
- 7: APPEND
- 8: BOOTEND
- 9: <nicht zulässig>
- 10: Lock-Bits

Hinweis: für die Lock-Bits kann auch die Konstante LOCK\_BITS verwendet werden.

**Rückgabewert:**

Ausgelesene Fuses bzw. Lock-Bits.

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

## 16.3 writeBits

writeBits <targetHandle>, <index>, <values>

Schreibt die angegebenen Fuses bzw. Lock-Bits.

**Vorsicht:**

- Setzen Sie die Lock-Bits erst, nachdem Sie alle anderen Zugriffe auf den Chip ausgeführt haben.
- Falls Sie einen durch Lock-Bits gesperrten Chip bearbeiten wollen, führen Sie als erstes ein `eraseFlash` aus. Dieses setzt auch die Lock-Bits wieder zurück.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Hinweis:**

Manche Änderungen der Fuses sind erst nach einem Reset wirksam bzw. mittels `readBits` sichtbar. Näheres finden Sie dazu im Handbuch des jeweiligen Targets. Sie können dazu das Kommando `restartTarget` verwenden.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**index** (aus Hersteller Dokumentation zu ATtiny417/817)

- 0:** WDTCFG
- 1:** BODCFG
- 2:** OSCCFG
- 3:** <nicht zulässig>
- 4:** TCD0CFG
- 5:** SYSCFG0
- 6:** SYSCFG1
- 7:** APPEND
- 8:** BOOTEND
- 9:** <nicht zulässig>
- 10:** Lock-Bits

Hinweis: für die Lock-Bits kann auch die Konstante `LOCK_BITS` verwendet werden.

### values

Zu schreibende Werte.

### Rückgabewert:

- keiner (Prozedur)

### Exceptions:

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index oder value
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

---

## 17 Dateien

### Dateinamen:

- Dateinamen müssen der 8.3-Regel folgen: „XXXXXXXX.YYY“.
- Es sind nur die Zeichen „A“ - „Z“, „0“ - „9“, sowie „\_“ und „-“ zulässig.
- Es dürfen nur Großbuchstaben verwendet werden.

### Verzeichnisnamen:

- Verzeichnisnamen dürfen maximal aus acht Zeichen bestehen: „XXXXXXXX“.
- Ansonsten gelten dieselben Konventionen wie bei Dateinamen.

### Aktuelles Verzeichnis ist immer das Hauptverzeichnis:

- Es gibt kein „change directory“. Der aktuelle Pfad bleibt immer das Hauptverzeichnis. Ein Dateiname muß daher immer den kompletten Pfad beinhalten.
- Als Trennzeichen zwischen Verzeichnissen und Dateiname werden „/“ und „\“ unterstützt.

## 17.1 fsCreate

```
fsCreate <filesystem>, <filename>
```

Erzeugt die angegebene Datei. Die Datei ist danach noch immer geschlossen. Falls die Datei schon existiert, hat die Prozedur keine Wirkung.

Wenn man eine Datei erzeugen und in diese etwas schreiben möchte, muß man die Datei zusätzlich noch öffnen:

```
fsCreate 0, "TEST.TXT"  
handle = fsOpen(0, "TEST.TXT")
```

### Vorbedingung:

- keine

### Parameter:

#### **filesystem**

Der Parameter wird ignoriert und sollte mit 0 angegeben werden.

#### **filename**

Es gelten die Bedingungen für Dateinamen, siehe Kapitel „Dateien“.

### Rückgabewert:

- keiner (Prozedur)

### Exceptions:

apiTypeFault	Unzulässiger Typ für filename.
<diverse Exceptions des Dateisystems>	Siehe Kapitel „Exceptions des Dateisystems“.

## 17.2 fsRemove

```
fsRemove <filesystem>, <filename>
```



Löscht die angegebene Datei oder das angegebene Verzeichnis, falls vorhanden.

**Vorbedingung:**

- keine

**Parameter:**

**filesystem**

Der Parameter wird ignoriert und sollte mit 0 angegeben werden.

**filename**

Es gelten die Bedingungen für Verzeichnis- bzw. Dateinamen, siehe Kapitel „Dateien“.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

fileNotFound

apiTypeFault

<diverse Exceptions des  
Dateisystems>

Die angegebene Datei existiert nicht.

Unzulässiger Typ für filename.

Siehe Kapitel „Exceptions des Dateisystems“.

## 17.3 fsMkDir

fsMkDir <filesystem>, <dirname>

Erzeugt das angegebene Verzeichnis. Falls das Verzeichnis schon existiert, hat die Prozedur keine Wirkung.

**Vorbedingung:**

- keine

**Parameter:**

**filesystem**

Der Parameter wird ignoriert und sollte mit 0 angegeben werden.

**dirname**

Es gelten die Bedingungen für Verzeichnisnamen, siehe Kapitel „Dateien“.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiTypeFault

<diverse Exceptions des  
Dateisystems>

Unzulässiger Typ für dirname.

Siehe Kapitel „Exceptions des Dateisystems“.

## 17.4 fsFileExists

```
bool fsFileExists(<filesystem>, <filename>)
```

Prüft, ob die angegebene Datei existiert.

**Vorbedingung:**

- keine

**Parameter:**

**filesystem**

Der Parameter wird ignoriert und sollte mit 0 angegeben werden.

**filename**

Es gelten die Bedingungen für Dateinamen, siehe Kapitel „Dateien“.

**Rückgabewert:**

0 = Datei existiert nicht

1 = Datei existiert

**Exceptions:**

apiTypeFault	Unzulässiger Typ für filename.
<diverse Exceptions des Dateisystems>	Siehe Kapitel „Exceptions des Dateisystems“.

## 17.5 fsFilesize

```
size = fsFilesize(<filesystem>, <filename>)
```

Ermittelt die Größe der angegebenen Datei.

**Vorbedingung:**

- Datei existiert.

**Parameter:**

**filesystem**

Der Parameter wird ignoriert und sollte mit 0 angegeben werden.

**filename**

Es gelten die Bedingungen für Dateinamen, siehe Kapitel „Dateien“.

**Rückgabewert:**

Es wird die Dateigröße zurückgegeben.

**Exceptions:**

apiTypeFault	Unzulässiger Typ für filename.
<diverse Exceptions des Dateisystems>	Siehe Kapitel „Exceptions des Dateisystems“.

## 17.6 fsOpen

```
filehandle = fsOpen(<filesystem>, <filename>)
```

Öffnet die angegebene Datei.

### **Vorbedingung:**

Die Datei muß bereits existieren. Soll eine neue Datei geöffnet werden, muß vorher `fsCreate` verwendet werden.

### **Parameter:**

#### **filesystem**

Der Parameter wird ignoriert und sollte mit 0 angegeben werden.

#### **filename**

Es gelten die Bedingungen für Dateinamen, siehe Kapitel „[Dateien](#)“.

### **Rückgabewert:**

Es wird ein Filehandle zum Zugriff auf die Datei (z. B. für `fsRead` und `fsWrite`) zurückgegeben. Das Filehandle wird außerdem zum Schließen der Datei (`fsClose`) benötigt.

### **Exceptions:**

<code>apiTypeFault</code>	Unzulässiger Typ für filename.
<code>&lt;diverse Exceptions des Dateisystems&gt;</code>	Siehe Kapitel „Exceptions des Dateisystems“.

## 17.7 fsRead

```
a = fsRead(<filehandle>, <position>, <count>)
```

Liest die angegebene Anzahl an Bytes aus der Datei.

### **Vorbedingung:**

- Gültiges Filehandle mittels fsOpen.

**Parameter:**

**filehandle**

Das von fsOpen erhaltene Filehandle.

**position**

Die Byte-Position, von der gelesen werden soll.

**count**

Die Anzahl der zu lesenden Bytes.

**Rückgabewert:**

Array of Byte mit den gelesenen Daten. Das Array hat die Größe von count. Falls nicht mehr genügend Daten zu lesen waren, ist das Array entsprechend kleiner. Wird am Dateiende oder darüber hinaus versucht zu lesen, wird ein leeres Array mit der Größe 0 zurückgegeben.

**Exceptions:**

apiValueRange	Unzulässiger Wert für filehandle, position oder count.
apiTypeFault	Unzulässiger Typ für filehandle, position oder count.
<diverse Exceptions des Dateisystems>	Siehe Kapitel „Exceptions des Dateisystems“.

## 17.8 fsWrite

fsWrite <filehandle>, <position>, <array>

Schreibt die übergebenen Daten in die Datei.

Sollte die Position außerhalb der momentanen Dateigröße sein, wird die Datei bis zu dieser Position mit zufälligen Werten aufgefüllt.

**Vorbedingung:**

- Gültiges Filehandle mittels `fsOpen`.

**Parameter:**

**filehandle**

Das von `fsOpen` erhaltene Filehandle.

**position**

Die Byte-Position, an die geschrieben werden soll.

**array**

Array of Byte mit den zu schreibenden Daten.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

<code>apiValueRange</code>	Unzulässiger Wert für <code>filehandle</code> , <code>position</code> oder <code>count</code> .
<code>apiTypeFault</code>	Unzulässiger Typ für <code>filehandle</code> , <code>position</code> oder <code>count</code> .
<diverse Exceptions des Dateisystems>	Siehe Kapitel „Exceptions des Dateisystems“.

## 17.9 fsTruncate

`fsTruncate <filehandle>, <len>`

Kürzt die Datei auf die angegebene Länge. Falls die Datei schon kleiner ist, ist die Prozedur ohne Wirkung.

**Vorbedingung:**

- Gültiges Filehandle mittels `fsOpen`.

**Parameter:**

**filehandle**

Das von fsOpen erhaltene Filehandle.

**len**

Die Länge, auf die die Datei gekürzt wird.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault  
<diverse Exceptions des  
Dateisystems>

Unzulässiger Wert für filehandle.  
Unzulässiger Typ für filehandle oder len.  
Siehe Kapitel „Exceptions des Dateisystems“.

## 17.10 fsClose

fsClose <filehandle>

Schließt die Datei. Das angegebene Filehandle wird dadurch ungültig und darf nicht mehr verwendet werden.

**Vorbedingung:**

- Gültiges Filehandle mittels fsOpen .

**Parameter:**

**filehandle**

Das von fsOpen erhaltene Filehandle.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange

apiTypeFault

<diverse Exceptions des  
Dateisystems>

Unzulässiger Wert für filehandle.

Unzulässiger Typ für filehandle.

Siehe Kapitel „Exceptions des Dateisystems“.

## 17.11 fsSync

fsSync <filesystem>

Stellt sicher, dass alle Daten, die evtl. noch nicht auf die Karte geschrieben wurden, nun geschrieben werden. Es wird empfohlen, wenn Schreibzugriffe auf die Karte stattfinden, diese Prozedur anschließend aufzurufen.

**Vorbedingung:**

- keine

**Parameter:**

**filesystem**

Der Parameter wird ignoriert und sollte mit 0 angegeben werden.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.



## 18 LEDs

### Immer nur eine LED an:

- Es ist aus roloBasic heraus nicht möglich, mehr als eine LED gleichzeitig einzuschalten. Dadurch können andere Fehlermeldungen vom System besser erkannt werden, da diese immer mehr als eine LED nutzen.

### Numerierung und Farben:

- Die LEDs sind in roloBasic genauso numeriert wie auf dem Gehäuse abgebildet: von 1 bis 5.
- Die LEDs können in Grün oder Rot leuchten. Dazu stehen die Konstanten `COLOR_GREEN` und `COLOR_RED` zur Verfügung.

### Nicht blockierend:

- Alle Prozeduren dieses Kapitels sind nicht blockierend. Das bedeutet, dass z. B. ein mit `ledRunningLight` aktiviertes Lauflicht parallel zur weiteren Ausführung des roloBasic läuft.

### 18.1 ledOn

```
ledOn <index>, <color>
```

Schaltet die LED auf die angegebene Farbe.

#### Vorbedingung:

- keine

#### Parameter:

**index**

Nummer der LED

**color**

COLOR\_GREEN bzw. COLOR\_RED

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für index oder color.  
Unzulässiger Typ für index oder color.

## 18.2 ledOff

ledOff

Schaltet alle LEDs aus.

**Vorbedingung:**

- keine

**Parameter:**

- keine

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

- keine

### 18.3 ledBlink

ledBlink <index>, <color>, <speed>

Die LED blinkt mit der angegebenen Geschwindigkeit.

**Vorbedingung:**

- keine

**Parameter:**

**index**

Nummer der LED

**color**

COLOR\_GREEN bzw. COLOR\_RED

**speed**

Geschwindigkeit des Blinkens in [ms]

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für index, color oder speed.  
Unzulässiger Typ für index, color oder speed.

### 18.4 ledRunningLight

ledRunningLight <from>, <to>, <color>, <speed>

Läßt ein Lauflicht laufen.

**Vorbedingung:**

- keine

**Parameter:**

**from, to**

Das Lauflicht läuft von LED 'from' bis LED 'to'.  
Ist 'from' kleiner als 'to', läuft das Lauflicht „anders herum“.  
Ist 'from' gleich 'to', leuchtet die eine LED ständig.

**color**

COLOR\_GREEN bzw. COLOR\_RED

**speed**

Geschwindigkeit des Lauflichts in [ms]

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für from, to, color oder speed.  
Unzulässiger Typ für from, to, color oder speed.

## 18.5 ledRunningLightOutstanding

```
ledRunningLightOutstanding <from>, <to>, <color>,  
<speed>, <outstandingLedNumber>
```

Läßt ein Lauflicht laufen, bei der eine bestimmte LED die jeweilig andere Farbe aufweist.

**Vorbedingung:**

- keine

**Parameter:**

**from, to**

Das Lauflicht läuft von LED 'from' bis LED 'to'.  
Ist 'from' kleiner als 'to', dann läuft das Lauflicht „anders herum“.  
Ist 'from' gleich 'to', leuchtet die eine LED ständig.

**color**

COLOR\_GREEN bzw. COLOR\_RED

**speed**

Geschwindigkeit des Lauflichts in [ms]

**outstandingLedNumber**

Nummer der LED, die mit der anderen Farbe aufleuchtet.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange

Unzulässiger Wert für from, to, color, speed oder outstandingLedNumber.

apiTypeFault

Unzulässiger Typ für from, to, color, speed oder outstandingLedNumber.

---

## 19 Sonstige

### 19.1 setLogMode

```
setLogMode <logMode>
```

Es wird der Mode für das Loggen (siehe folgendes Kapitel „[print](#)“) festgelegt.

Der Ausdruck erfolgt an das Ende der Datei „LOG.TXT“. Wenn die Datei noch nicht existiert, dann wird sie erzeugt.

**Vorbedingung:**

- keine

**Parameter:**

**LogMode:**

**LOGMODE\_OFF:** print-Ausgaben werden unterdrückt.

**LOGMODE\_NORMAL:** Die Datei wird geöffnet und bleibt geöffnet. Print-Ausgaben werden zwischengespeichert und gelegentlich in die Datei geschrieben. Am Ende des Skripts werden noch gespeicherte Daten geschrieben und die Datei geschlossen.

**LOGMODE\_IMMEDIATE:** Bei jeder Print-Ausgabe wird die Log-Datei geöffnet, die Ausgabe in die Datei geschrieben und die Datei wieder geschlossen. Somit ist sichergestellt, daß bei Ausführung der nächsten Skriptzeile die Print-Ausgabe auf der microSD-Karte gespeichert ist.

**Rückgabewert:**

- keiner (Prozedur)

**Hinweis:** Default ist der logMode LOGMODE\_NORMAL.

**Empfehlungen:**

Verwenden Sie LOGMODE\_IMMEDIATE nur bei der Fehlersuche. Da jede print-Ausgabe erneut die Datei öffnet, beschreibt und wieder schließt, wird auf der microSD-Karte jedesmal die FAT (file allocation table) beschrieben. Dies kann zu einem erhöhtem Verschleiß der microSD-Karte und deren Ausfall führen.

Wenn Sie Log-Ausgaben gar nicht benötigen, können Sie am Anfang des Skripts auf LOGMODE\_OFF wechseln. Sie können auch innerhalb des Skripts den LogMode wechseln.

Wenn Sie mit LOGMODE\_NORMAL arbeiten, werden die Log-Ausgaben eventuell erst nach Beendigung des Skripts auf die microSD-Karte geschrieben. Falls Sie in Ihren Skripten am Ende die letzte LED grün aufleuchten lassen, dann tun Sie das bitte erst am Ende, damit das abschließende Schreiben noch in der Reaktionszeit des Benutzers abgeschlossen werden kann. Vermutlich wird dieser roloFlash abziehen.

Wenn Sie über ein Interface (UART oder GPIO) einem anderen Gerät mitteilen, daß roloFlash fertig ist und dieses als Folge die Stromversorgung des Targets abschaltet, dann kann evtl. das Skript vor der Abschaltung nicht mehr beendet werden und somit Log-Ausgaben fehlen, bzw. das Dateisystem beschädigt werden. In diesem Fall sollten Sie den logMode unmittelbar vor der Mitteilung auf LOGMODE\_IMMEDIATE setzen.

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für logMode.  
Unzulässiger Typ für logMode.

## 19.2 print

```
print <a>, <b>, ...
```

Es werden die Parameter a, b etc. ausgedruckt. Die Anzahl der Parameter ist beliebig.

Der Ausdruck erfolgt an das Ende der Datei „LOG.TXT“. Wenn die Datei noch nicht existiert, dann wird sie erzeugt.

**Vorbedingung:**

- keine

**Parameter:**

**a, b, ...**

Hier können Zahlen und Arrays ausgegeben werden. Beispiel:

```
value = 42
```

```
print "Der Wert ist: ", value
```

Ist ein angegebener Parameter weder eine Zahl noch ein Char-Array, dann wird nichts ausgegeben.

**Rückgabewert:**

- keiner (Prozedur)

**Hinweis:** Die Ausgabe ist vom gewählten Logmode (siehe vorheriges Kapitel „setLogMode“) abhängig.

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

### 19.3 delay

delay <duration>

Es wird die angegebene Zeitdauer in ms abgewartet. Erst danach kehrt die Funktion zurück.

**Vorbedingung:**

- keine

**Parameter:**

**duration**

Zeitangabe in ms, die gewartet werden soll.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für duration.  
Unzulässiger Typ für duration.



## 19.4 getSystemTime

t = getSystemTime

Ermittelt die abgelaufene Zeit seit Systemstart in ms.

**Vorbedingung:**

- keine

**Parameter:**

- keine

**Rückgabewert:**

Systemzeit in [ms].

**Exceptions:**

- keine

## 19.5 getTargetBoardVoltage

u = getTargetBoardVoltage

Ermittelt die Spannung in mV, die das Targetboard liefert.

**Vorbedingung:**

- keine

**Parameter:**

- keine

**Rückgabewert:**

Ausgelesene Spannung in mV.

**Exceptions:**

- keine

## 19.6 setCpuClock

setCpuClock <frequency>

Es wird der interne CPU-Takt des roloFlash geändert.

- ein höherer Takt benötigt mehr Energie vom Targetboard
- ein niedriger Takt benötigt evtl. längere Zeit, um das roloBasic-Skript inkl. Flashen abzuarbeiten.

Der Takt bei Start des roloFlash ist für niedrigen Energieverbrauch auf 24 MHz eingestellt.

**Achtung:**

Bereits geöffnete Busse können dabei Ihren Takt ändern. Den aktuellen Takt können Sie abfragen.

**Empfehlung:**

Ändern Sie bei Bedarf den Takt am Anfang Ihres Skriptes.

**Vorbedingung:**

- keine

**Parameter:**

**frequency**

Angabe in Hz.

Unterstützte Werte:

- CPU\_CLOCKMAX: 120000000 (120 MHz)

- CPU\_CLOCKMIN: 24000000 (24 MHz)

Der Takt wird immer auf den nächstkleineren Takt, jedoch mindestens 24 Mhz eingestellt.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für frequency.  
Unzulässiger Typ für frequency.

## 19.7 getCpuClock

u = getCpuClock

Ermittelt den aktuellen Takt des roloFlash in Hz.

**Vorbedingung:**

- keine

**Parameter:**

- keine

**Rückgabewert:**

Ausgelesener Takt in Hz.

**Exceptions:**

- keine

# VII Abfrage von roloFlash-Eigenschaften

Mit folgenden Systemfunktionen / Systemkonstanten können Sie verschiedene Informationen Ihres roloFlashs ermitteln.

---

## 1 Versionsnummern etc.

Name	Wert / Bedeutung
companyName	„halec < <a href="http://halec.de">http://halec.de</a> >“
deviceName	„roloFlash 2“ bzw. „roloFlash 2 AVR“
softwareVersion	Versionsnummer der Firmware
hardwareVersion	Versionsnummer der Hardware
bootloaderVersion	Versionsnummer des Bootloaders
imageVersion	roloFlash erwartet das vom Compiler erzeugte Image in dieser Version. Bitte verwenden Sie daher den zur Firmware des roloFlash passenden Compiler.

---

## 2 Seriennummer

Name	Wert / Bedeutung
serialNumber	Ein String mit 24 Zeichen, jedes Zeichen '0' - '9' oder 'A' - 'F'

Die Seriennummer ist für jeden roloFlash eindeutig. Sie können somit rolo-Basic-Skripte erstellen, die nur auf bestimmten roloFlash ablaufen.

### **Beispiel:**

1. Einmalig die Seriennummer ermitteln:

```
print "serialNumber: ", serialNumber, "\r\n"
```

Aus Log-File entnehmen:

serialNumber: 1B9FE86E90B7660F08E387B

2. Ihr Skript soll nur auf diesem roloFlash laufen, andernfalls mit einer Exception abbrechen:

```
if serialNumber <> "1B9FE86E90B7660F08E387B"  
    print "Falscher roloFlash, Abbruch\r\n"  
    throw userException  
endif
```

**Hinweis:**

Für die Seriennummer wird intern eine vom Chiphersteller vorgegebene eindeutige Device-ID genutzt.

## VIII Exceptions

Im Handbuch für das roloBasic finden Sie die genaue Beschreibung, wie Exceptions ausgelöst und wieder gefangen werden können. Wird eine Exception nicht gefangen, wird die Exception mittels der LEDs angezeigt.

Falls dabei die Exception keine Zahl darstellt, wird die Exception „exceptionNotANumber“ ausgegeben. Details dazu finden Sie unter Kapitel „Exception aufgetreten“. Nur vom Benutzer ausgelöste Exceptions können nicht-numerisch sein.

Es gibt verschiedene Arten von Exceptions, die alle gleich behandelt werden:

- Exceptions des roloBasic
- Exceptions des Dateisystems
- Exceptions des roloFlash
- Vom Benutzer ausgelöste Exceptions

---

### 1 Exceptions des roloBasic

Diese Exceptions treten bei Fehlern auf, die nicht speziell etwas mit roloFlash zu tun haben, sondern mit der Abarbeitung des roloBasic. Ein typisches Beispiel dazu ist eine valueRange-Exception.

Sie finden diese Exceptions auch nochmal im Handbuch zu roloBasic.

Treten Fehler, wie bei den Exceptions valueRange, argumentFault und typeFault beschrieben, beim Aufruf an eine API-Funktion / -Prozedur auf, dann werden stattdessen die Exceptions apiValueRange, apiArgumentFault oder apiTypeFault erzeugt. Die jeweilige Nummer der Exceptions ist exakt 200 größer als die entsprechenden Exceptions des roloBasic.

Name	Nummer	Bedeutung
outOfMemory	1	Zu wenig freier Speicher vorhanden
rootstackOverflow	2	Interner Systemfehler
nullpointerAccess	3	Interner Systemfehler
valueRange	4	Wertbereichsüberschreitung, z.B. bei Zuweisung von Werten an Arrays
divisionByZero	5	Division durch 0. Kann bei div oder mod auftreten
argumentFault	6	Ungültige Anzahl Argumente beim Aufruf einer roloBasic-Funktion / Prozedur.
illegalFunction	7	Eine Variable wurde wie eine Funktion oder Prozedur aufgerufen, enthält jedoch keine gültige Funktion oder Prozedur.
indexRange	8	Indexbereichsüberschreitung bei Arrayzugriff
typeFault	9	Ein übergebener Parameter hat einen nicht passenden Type.

## 2 Exceptions des Dateisystems

Diese Exceptions treten bei Fehlern im Zusammenhang mit dem Dateisystem oder mit der microSD-Karte auf.

Name	Nummer	Bedeutung
deviceError	101	Es konnte nicht von der microSD-Karte gelesen bzw. auf sie geschrieben werden.
badCluster	102	Probleme innerhalb des Dateisystems. Das Dateisystem sollte auf dem PC auf Konsistenz geprüft werden.
notMounted	103	Es wurde versucht, auf die microSD-Karte zuzugreifen, obwohl sie nicht angemeldet ist. Dieses deutet auf ein Problem mit der microSD-Karte hin.
removeError	104	Die microSD-Karte wurde entfernt.
createError	105	Das Erzeugen einer Datei oder eines Verzeichnisses ist fehlgeschlagen.
fileNotOpen	106	Die Datei ist nicht geöffnet.
fileNotFound	107	Die angegebene Datei oder das Verzeichnis konnte nicht gefunden werden.
diskFull	108	Die microSD-Karte ist voll.
truncateError	109	Das Kürzen einer Datei mittels fsTruncate ist fehlgeschlagen.
illegalCluster	110	Probleme innerhalb des Dateisystems. Das Dateisystem sollte auf dem PC auf Konsistenz geprüft werden.
fileLocked	111	Es wurde versucht, eine bereits geöffnete Datei ein zweites Mal zu öffnen. Evtl. wurde ein fsClose vergessen.
outOfFileHandles	112	Die Anzahl der maximal geöffneten Dateien ist auf 3 begrenzt. Es wurde versucht, eine weitere Datei zu öffnen.
loaderNotFound	113	Der benötigte Loader wurde auf microSD-Karte nicht gefunden



## **3 Exceptions des roloFlash**

Name	Nummer	Bedeutung
exceptionIsNotANumber	200	<p>Es wurde eine Exception erzeugt, die keine Zahl ist, und innerhalb des roloBasic nicht mehr gefangen wurde. In diesem Fall wird die Original-Exception verworfen und durch diese Exception ersetzt.</p> <p>Das kann nur durch vom Benutzer erzeugte Exceptions auftreten, da alle anderen Funktionen ausschließlich die hier beschriebenen numerischen Exceptions nutzen. Beispiel: throw "Fehler"</p>
imageTooLarge	201	Das roloBasic-Skript ist zu groß. Es können maximal circa 65.000 Bytes geladen werden. Bitte überprüfen Sie die Dateigröße der vom Compiler erzeugten Datei.
imageWrongVersion	202	Der verwendete Compiler passt nicht zur Firmware auf dem roloFlash. Es wird empfohlen, immer jeweils den neuesten Compiler und neueste Firmware für den roloFlash zu verwenden.
productWrongVersion	203	Es wurde versucht, ein Image eines anderen Produktes auf den roloFlash zu laden. Beispielsweise wurde versucht, ein Image für roloFlash 1 auf einen roloFlash 2 zu laden.
apiValueRange	204	Wertbereichsüberschreitung eines Parameters beim Aufruf einer Api-Funktion / Prozedur . z.B. ledOn 6, COLOR_GREEN ! Es gibt nur 5 LEDs ( <b>Hinweis:</b> die Fehlernummer ist exakt 200 größer als die entsprechende Exception des roloBasic: valueRange)
imageNotFound	205	Die microSD-Karte konnte zwar eingebunden werden, jedoch nicht die Datei RUN.BIN gefunden werden.
apiBadArgumentCount	206	Ungültige Anzahl Argumente beim Aufruf einer Api-Funktion / Prozedur. ( <b>Hinweis:</b> die Fehlernummer ist exakt 200 größer als die entsprechende Exception des roloBasic: badArgumentCount)
apiTypeFault	209	Ein an eine API-Funktion / Prozedur übergebener Parameter hat einen nicht passenden Type. ( <b>Hinweis:</b> die Fehlernummer ist exakt 200 größer als die entsprechende Exception des roloBasic: typeFault)
targetWrongMode	210	Die aufgerufene Prozedur oder Funktion erfordert einen bestimmten Mode des Targets. Z. B. setzt die Prozedur setProgrammingSpeed den

		ProgramMode voraus.
targetCommunication	211	Ein Kommunikationsfehler mit dem Target.
targetMemoryLayout	212	Das Speicherlayout des Controllers ist nicht angegeben worden (setMemoryMap).
eraseError	213	Das Löschen des Targets hat nicht funktioniert.
targetVerify	214	Beim Zurücklesen von Daten wurde ein Unterschied festgestellt.
hexFileSize	230	Die Größe der angegebenen Hex-Datei ist nicht plausibel. Eventuell ist die Hex-Datei nicht in Ordnung oder leer.
hexFileCRC	231	Beim Parsen der Hex-Datei ist ein Prüfsummenfehler aufgetreten. Eventuell ist die Hex-Datei nicht in Ordnung.
invalidHandle	250	Das benutzte Handle ist nicht gültig. Das Handle wurde bereits geschlossen oder ein falscher Parameter wurde anstatt des Handles verwendet.
resourceUnavailable	251	Die angeforderte Ressource ist nicht verfügbar. Dieses kann beim Öffnen eines Busses passieren, falls schon ein anderer Bus geöffnet hat, der zumindest teilweise auf die gleichen Ressourcen zugreift. Insbesondere tritt diese Exception auf, wenn der gleiche Bus zweimal geöffnet wird.
unknownTarget	252	In der Datenbank ist der angefragte Controller nicht aufgeführt (siehe DB_getHandle)
propertyNotFound	253	Das angefragte Property ist für den angegebenen Controller nicht verfügbar (siehe DB_get)
familyNotSupported	254	Die angegebene Controller-Familie wird nicht unterstützt (siehe getTargetHandle)
FunctionNotSupported	255	Es wurde eine für das aktuelle Target nicht unterstützte Funktion aufgerufen. Z.B. wird die Prozedur WriteBits nur für Atmel-Controller unterstützt.
ValueUnknown	256	Es wurde versucht, einen Wert auszulesen, der nicht ermittelt werden kann (siehe getMemoryMap).
ValueNotAllowed	257	Es wurde versucht, einen unzulässigen Wert zu verwenden (siehe setMemoryMap).
TimeoutError	258	Die aufgerufene Funktion dauert zu lange. Evtl. liegt ein Problem mit dem Target vor. Falls nach einem solchen Fehler mit dem Target weitergearbeitet werden soll, kann es nötig sein, das Target-Handle vorab zu schließen und neu anzufordern.

TargetError	260	Das Target hat einen nicht weiter spezifizierten Fehler gemeldet. Bei ARM-Targets können das gesetzte Sticky-Bits sein.
writeProtectError	261	Der angesprochene Speicherbereich des Targets ist schreibgeschützt.
readProtectError	262	Der angesprochene Speicherbereich des Targets ist lesegeschützt.
writeError	263	Es gab einen Fehler beim Beschreiben des angesprochenen Speicherbereichs.
readError	264	Es gab einen Fehler beim Auslesen des angesprochenen Speicherbereichs.

## 4 Vom Benutzer ausgelöste Exceptions

- Der Benutzer kann mittels `throw` selbst Exceptions auslösen. Diese können numerisch sein und bestehende Werte mitnutzen, z. B.:  
`throw rangeError`
- Um vom Benutzer erzeugte Exceptions von den anderen Exceptions besser unterscheiden zu können, können andere Exceptionnummern verwendet werden. Es wird hierzu die Konstante `userException` mit dem Wert 1000 zur Verfügung gestellt. Der Vorteil dieses Wertes ist, dass sie im Blinkcode besonders gut zu erkennen ist, wenn die Exception nicht mehr gefangen wird. Die Konstante ist als Offset für eigenen Exceptions nutzbar, z.B.:  
`throw userException + 1`
- Es können auch nicht-numerische Exceptions erzeugt werden. Falls eine solche Exception nicht mehr gefangen wird, wird sie zum Schluss in die Exception `exceptionIsNotANumber` umgewandelt und der Code als Blinkcode ausgegeben, z. B.:  
`throw "error"`

# IX Bedeutungen von LED-Codes

---

## 1 Normaler Betrieb

### 1.1 Keine microSD-Karte gefunden

**LEDs:**

- 1: rot
- 2:
- 3:
- 4:
- 5:

**Bedeutung:**

Keine microSD-Karte gefunden, bzw. die Karte ist nicht mit FAT32 formatiert.

**Hinweis:**

Für den normalen Betrieb ist es Voraussetzung, daß beim Anschließen des roloFlash die microSD-Karte bereits eingelegt ist. Der Fall, die microSD-Karte erst später einzustecken, ist für die Firmware-Aktualisierung des roloFlash reserviert. Falls Sie den roloFlash normal verwenden wollen und lediglich die microSD-Karte nicht eingesteckt hatten, dann entfernen Sie bitte roloFlash, legen die microSD-Karte ein und schließen Sie roloFlash erneut an.

### 1.2 Exception aufgetreten

Wenn eine Exception aufgetreten ist und diese nicht aufgelöst (gefangen) wurde, wird die Nummer der Exception durch einen Blinkcode angezeigt.

**LEDs:**

- 1: rot: geht am Anfang des Blinkcodes kurz aus und wieder an
- 2: rot: blinkend, Anzahl entspricht 1000-er der Exception
- 3: rot: blinkend, Anzahl entspricht 100-er der Exception
- 4: rot: blinkend, Anzahl entspricht 10-er der Exception
- 5: rot: blinkend, Anzahl entspricht 1-er der Exception

**Bedeutung:**

Dieser Code kann entstanden sein, indem

- im Skript eine entsprechende „throw“-Anweisung ausgeführt wurde. Beispiel:

```
if getVoltage() > 4000
    throw 1234 !Exception 1234 erzeugen
endif
```
- eine Funktion / Prozedur Ihre Aufgabe nicht erfüllen konnte und eine Exception erzeugt hat.

---

## 2 roloFlash-Aktualisierung

Die Aktualisierung der roloFlash-Firmware ist im Kapitel „Aktualisieren von roloFlash“ beschrieben.

### 2.1 Wartet auf microSD-Karte für Aktualisierung

**LEDs:**

- 1: rot
- 2:
- 3:
- 4:
- 5:

**Bedeutung:**

Wenn beim Start des roloFlash keine microSD-Karte eingelegt ist, dann wird auf das Einstecken der microSD-Karte gewartet, um anschließend die Aktualisierung zu starten.

Falls Sie keine Aktualisierung durchführen wollen, dann starten Sie roloFlash mit vorab eingesteckter microSD-Karte.

## 2.2 Aktualisierung läuft

### LEDs:

- 1: rot
- 2: grün \ im Wechsel blinkend
- 3: grün /
- 4:
- 5:

### Bedeutung:

Die Aktualisierung läuft. Diese benötigt circa 10-15 Sekunden, bitte nicht abbrechen.

## 2.3 Aktualisierung mit Erfolg abgeschlossen

### LEDs:

- 1: grün
- 2: grün
- 3:
- 4:
- 5:

### Bedeutung:

Die Aktualisierung wurde mit Erfolg abgeschlossen. Nach dem Abziehen steht die neue Firmware bei der nächsten Nutzung zur Verfügung.

## 2.4 Aktualisierung fehlerhaft: Dateifehler

### LEDs:

- 1: rot
- 2: rot
- 3:
- 4:
- 5:

### Bedeutung:

Die Aktualisierung schlug mit einem Dateifehler fehl. Die alte Firmware steht evtl. noch zur Verfügung.

**Mögliche Abhilfe:**

- Aktualisierung nochmals versuchen.
- Aktualisierung mit einer anderen Firmware durchführen.

## 2.5 Aktualisierung fehlerhaft: Datei nicht gefunden

**LEDs:**

- 1: rot
- 2:
- 3: rot
- 4:
- 5:

**Bedeutung:**

Die Aktualisierung konnte nicht gestartet werden, da keine Datei für die Aktualisierung gefunden wurde. Die alte Firmware steht noch zur Verfügung.

**Mögliche Abhilfe:**

Datei für Firmware-Update auf die microSD-Karte kopieren, dann Aktualisierung nochmals versuchen.

## 2.6 Aktualisierung fehlerhaft: Mehrere Dateien gefunden

**LEDs:**

- 1: rot
- 2:
- 3:
- 4: rot
- 5:

**Bedeutung:**

Die Aktualisierung konnte nicht gestartet werden, da mehrere Dateien für die Aktualisierung gefunden wurde und dadurch nicht eindeutig ist, welche Datei verwendet werden soll. Die alte Firmware steht noch zur Verfügung.

**Mögliche Abhilfe:**



Es darf nur eine Datei vorhanden sein, die für eine Aktualisierung geeignet ist. Überflüssige Dateien bitte entfernen und dann Aktualisierung nochmals versuchen.

## 2.7 Aktualisierung fehlerhaft: Sonstiges

### **LEDs:**

- 1: rot
- 2:
- 3:
- 4:
- 5: rot

### **Bedeutung:**

Bei der Aktualisierung schlug etwas fehl. Die alte Firmware steht evtl. noch zur Verfügung.

### **Mögliche Abhilfe:**

- Aktualisierung nochmals versuchen.
- Aktualisierung mit einer anderen Firmware durchführen.

# X Spezifikation

## 1 Unterstützte Controller von ST Microelectronics

Folgende Controller sind in der Datenbank bekannt. Die hier angegebenen Namen können mit DB\_getHandle verwendet werden.

### 1.1 STM32F1

Anschluß über JTAG-Interface.

Unterstützte Controller:

STM32F100C4, STM32F100C6, STM32F100C8, STM32F100CB,  
STM32F100R4, STM32F100R6, STM32F100R8, STM32F100RB,  
STM32F100RC, STM32F100RD, STM32F100RE, STM32F100V8,  
STM32F100VB, STM32F100VC, STM32F100VD, STM32F100VE,  
STM32F100ZC, STM32F100ZD, STM32F100ZE, STM32F101C4,  
STM32F101C6, STM32F101C8, STM32F101CB, STM32F101R4,  
STM32F101R6, STM32F101R8, STM32F101RB, STM32F101RC,  
STM32F101RD, STM32F101RE, STM32F101RF, STM32F101RG,  
STM32F101T4, STM32F101T6, STM32F101T8, STM32F101TB,  
STM32F101V8, STM32F101VB, STM32F101VC, STM32F101VD,  
STM32F101VE, STM32F101VF, STM32F101VG, STM32F101ZC,  
STM32F101ZD, STM32F101ZE, STM32F101ZF, STM32F101ZG,  
STM32F102C4, STM32F102C6, STM32F102C8, STM32F102CB,  
STM32F102R4, STM32F102R6, STM32F102R8, STM32F102RB,  
STM32F103C4, STM32F103C6, STM32F103C8, STM32F103CB,  
STM32F103R4, STM32F103R6, STM32F103R8, STM32F103RB,  
STM32F103RC, STM32F103RD, STM32F103RE, STM32F103RF,  
STM32F103RG, STM32F103T4, STM32F103T6, STM32F103T8,

STM32F103TB, STM32F103V8, STM32F103VB, STM32F103VC,  
STM32F103VD, STM32F103VE, STM32F103VF, STM32F103VG,  
STM32F103ZC, STM32F103ZD, STM32F103ZE, STM32F103ZF,  
STM32F103ZG, STM32F105R8, STM32F105RB, STM32F105RC,  
STM32F105V8, STM32F105VB, STM32F105VC, STM32F107RB,  
STM32F107RC, STM32F107VB, STM32F107VC

## 1.2 STM32F2

Anschluß über JTAG-Interface.

Unterstützte Controller:

STM32F205RB, STM32F205RC, STM32F205RE, STM32F205RF,  
STM32F205RG, STM32F205VB, STM32F205VC, STM32F205VE,  
STM32F205VF, STM32F205VG, STM32F205ZC, STM32F205ZE,  
STM32F205ZF, STM32F205ZG, STM32F207IC, STM32F207IE,  
STM32F207IF, STM32F207IG, STM32F207VC, STM32F207VE,  
STM32F207VF, STM32F207VG, STM32F207ZC, STM32F207ZE,  
STM32F207ZF, STM32F207ZG, STM32F215RE, STM32F215RG,  
STM32F215VE, STM32F215VG, STM32F215ZE, STM32F215ZG,  
STM32F217IE, STM32F217IG, STM32F217VE, STM32F217VG,  
STM32F217ZE, STM32F217ZG

## 1.3 STM32F3

Anschluß über JTAG-Interface.

Unterstützte Controller:

STM32F301C6, STM32F301C8, STM32F301K6, STM32F301K8,  
STM32F301R6, STM32F301R8, STM32F302C6, STM32F302C8,  
STM32F302CB, STM32F302CC, STM32F302K6, STM32F302K8,  
STM32F302R6, STM32F302R8, STM32F302RB, STM32F302RC,  
STM32F302RD, STM32F302RE, STM32F302VB, STM32F302VC,  
STM32F302VD, STM32F302VE, STM32F302ZD, STM32F302ZE,  
STM32F303C6, STM32F303C8, STM32F303CB, STM32F303CC,

STM32F303K6, STM32F303K8, STM32F303R6, STM32F303R8,  
STM32F303RB, STM32F303RC, STM32F303RD, STM32F303RE,  
STM32F303VB, STM32F303VC, STM32F303VD, STM32F303VE,  
STM32F303ZD, STM32F303ZE, STM32F318C8, STM32F318K8,  
STM32F328C8, STM32F334C4, STM32F334C6, STM32F334C8,  
STM32F334K4, STM32F334K6, STM32F334K8, STM32F334R6,  
STM32F334R8, STM32F358CC, STM32F358RC, STM32F358VC,  
STM32F373C8, STM32F373CB, STM32F373CC, STM32F373R8,  
STM32F373RB, STM32F373RC, STM32F373V8, STM32F373VB,  
STM32F373VC, STM32F378CC, STM32F378RC, STM32F378VC,  
STM32F398VE

## 1.4 STM32F4

Anschluß über JTAG-Interface.

Unterstützte Controller:

STM32F401CB, STM32F401CC, STM32F401CD, STM32F401CE,  
STM32F401RB, STM32F401RC, STM32F401RD, STM32F401RE,  
STM32F401VB, STM32F401VC, STM32F401VD, STM32F401VE,  
STM32F4050E, STM32F4050G, STM32F405RG, STM32F405VG,  
STM32F405ZG, STM32F407IE, STM32F407IG, STM32F407VE,  
STM32F407VG, STM32F407ZE, STM32F407ZG, STM32F410C8,  
STM32F410CB, STM32F410R8, STM32F410RB, STM32F410T8,  
STM32F410TB, STM32F411CC, STM32F411CE, STM32F411RC,  
STM32F411RE, STM32F411VC, STM32F411VE, STM32F412CE,  
STM32F412CG, STM32F412RE, STM32F412RG, STM32F412VE,  
STM32F412VG, STM32F412ZE, STM32F412ZG, STM32F413CG,  
STM32F413CH, STM32F413MG, STM32F413MH, STM32F413RG,  
STM32F413RH, STM32F413VG, STM32F413VH, STM32F413ZG,  
STM32F413ZH, STM32F4150G, STM32F415RG, STM32F415VG,  
STM32F415ZG, STM32F417IE, STM32F417IG, STM32F417VE,  
STM32F417VG, STM32F417ZE, STM32F417ZG, STM32F423CH,

STM32F423MH, STM32F423RH, STM32F423VH, STM32F423ZH,  
 STM32F427AG, STM32F427AI, STM32F427IG, STM32F427II,  
 STM32F427VG, STM32F427VI, STM32F427ZG, STM32F427ZI,  
 STM32F429AG, STM32F429AI, STM32F429BE, STM32F429BG,  
 STM32F429BI, STM32F429IE, STM32F429IG, STM32F429II,  
 STM32F429NE, STM32F429NG, STM32F429NI, STM32F429VE,  
 STM32F429VG, STM32F429VI, STM32F429ZE, STM32F429ZG,  
 STM32F429ZI, STM32F437AI, STM32F437IG, STM32F437II,  
 STM32F437VG, STM32F437VI, STM32F437ZG, STM32F437ZI,  
 STM32F439AI, STM32F439BG, STM32F439BI, STM32F439IG,  
 STM32F439II, STM32F439NG, STM32F439NI, STM32F439VG,  
 STM32F439VI, STM32F439ZG, STM32F439ZI, STM32F446MC,  
 STM32F446ME, STM32F446RC, STM32F446RE, STM32F446VC,  
 STM32F446VE, STM32F446ZC, STM32F446ZE, STM32F469AE,  
 STM32F469AG, STM32F469AI, STM32F469BE, STM32F469BG,  
 STM32F469BI, STM32F469IE, STM32F469IG, STM32F469II,  
 STM32F469NE, STM32F469NG, STM32F469NI, STM32F469VE,  
 STM32F469VG, STM32F469VI, STM32F469ZE, STM32F469ZG,  
 STM32F469ZI, STM32F479AG, STM32F479AI, STM32F479BG,  
 STM32F479BI, STM32F479IG, STM32F479II, STM32F479NG,  
 STM32F479NI, STM32F479VG, STM32F479VI, STM32F479ZG,  
 STM32F479ZI

---

## 2 Unterstützte Controller von Atmel

Folgende Controller sind in der Datenbank bekannt. Die hier angegebenen Namen können mit `DB_getHandle` verwendet werden.

### 2.1 AVR (ISP-Interface)

Anschluß über ISP-Interface.

Unterstützte Controller:

AT90CAN128,	AT90CAN32,	AT90CAN64,
AT90PWM1,	AT90PWM2,	AT90PWM216,
AT90PWM2B,	AT90PWM3,	AT90PWM316,
AT90PWM3B,	AT90PWM81,	AT90S1200,
AT90S2313,	AT90S2323,	AT90S2343,
AT90S4414,	AT90S4433,	AT90S4434,
AT90S8515,	AT90S8535,	AT90SCR100H,
AT90USB1286,	AT90USB1287,	AT90USB162,
AT90USB646,	AT90USB647,	AT90USB82,
ATmega103,	ATmega128,	ATmega1280,
ATmega1281,	ATmega1284,	ATmega1284P,
ATmega1284RFR2,	ATmega128A,	ATmega128RFA1,
ATmega128RFR2,	ATmega16,	ATmega161,
ATmega162,	ATmega163,	ATmega164A,
ATmega164P,	ATmega164PA,	ATmega165,
ATmega165A,	ATmega165P,	ATmega165PA,
ATmega168,	ATmega168A,	ATmega168P,
ATmega168PA,	ATmega168PB,	ATmega169,
ATmega169A,	ATmega169P,	ATmega169PA,
ATmega16A,	ATmega16HVA,	ATmega16HVA2,
ATmega16HVB,	ATmega16HVBrevB,	ATmega16M1,
ATmega16U2,	ATmega16U4,	ATmega2560,
ATmega2561,	ATmega2564RFR2,	ATmega256RFR2,
ATmega32,	ATmega323,	ATmega324A,
ATmega324P,	ATmega324PA,	ATmega324PB,
ATmega325,	ATmega3250,	ATmega3250A,
ATmega3250P,	ATmega3250PA,	ATmega325A,
ATmega325P,	ATmega325PA,	ATmega328,
ATmega328P,	ATmega328PB,	ATmega329,
ATmega3290,	ATmega3290A,	ATmega3290P,

ATmega3290PA,	ATmega329A,	ATmega329P,
ATmega329PA,	ATmega32A,	ATmega32C1,
ATmega32HVB,	ATmega32HVBrevB,	ATmega32M1,
ATmega32U2,	ATmega32U4,	ATmega32U6,
ATmega48,	ATmega48A,	ATmega48P,
ATmega48PA,	ATmega48PB,	ATmega64,
ATmega640,	ATmega644,	ATmega644A,
ATmega644P,	ATmega644PA,	ATmega644RFR2,
ATmega645,	ATmega6450,	ATmega6450A,
ATmega6450P,	ATmega645A,	ATmega645P,
ATmega649,	ATmega6490,	ATmega6490A,
ATmega6490P,	ATmega649A,	ATmega649P,
ATmega64A,	ATmega64C1,	ATmega64HVE,
ATmega64HVE2,	ATmega64M1,	ATmega64RFR2,
ATmega8,	ATmega8515,	ATmega8535,
ATmega88,	ATmega88A,	ATmega88P,
ATmega88PA,	ATmega88PB,	ATmega8A,
ATmega8HVA,	ATmega8U2,	ATtiny12,
ATtiny13,	ATtiny13A,	ATtiny15,
ATtiny1634,	ATtiny167,	ATtiny22,
ATtiny2313,	ATtiny2313A,	ATtiny24,
ATtiny24A,	ATtiny25,	ATtiny26,
ATtiny261,	ATtiny261A,	ATtiny4313,
ATtiny43U,	ATtiny44,	ATtiny441,
ATtiny44A,	ATtiny45,	ATtiny461,
ATtiny461A,	ATtiny48,	ATtiny80,
ATtiny828,	ATtiny84,	ATtiny840,
ATtiny841,	ATtiny84A,	ATtiny85,
ATtiny861,	ATtiny861A,	ATtiny87,
ATtiny88		

## 2.2 AVR (PDI-Interface)

Anschluß über PDI-Interface.

Unterstützte Controller:

ATxmega128A1,	ATxmega128A1U,	ATxmega128A3,
ATxmega128A3U,	ATxmega128A4U,	ATxmega128B1,
ATxmega128B3,	ATxmega128C3,	ATxmega128D3,
ATxmega128D4,	ATxmega16A4,	ATxmega16A4U,
ATxmega16C4,	ATxmega16D4,	ATxmega16E5,
ATxmega192A3,	ATxmega192A3U,	ATxmega192C3,
ATxmega192D3,	ATxmega256A3,	ATxmega256A3B,
ATxmega256A3BU,	ATxmega256A3U,	ATxmega256C3,
ATxmega256D3,	ATxmega32A4,	ATxmega32A4U,
ATxmega32C3,	ATxmega32C4,	ATxmega32D3,
ATxmega32D4,	ATxmega32E5,	ATxmega384C3,
ATxmega384D3,	ATxmega64A1,	ATxmega64A1U,
ATxmega64A3,	ATxmega64A3U,	ATxmega64A4U,
ATxmega64B1,	ATxmega64B3,	ATxmega64C3,
ATxmega64D3,	ATxmega64D4,	ATxmega8E5

## 2.3 AVR (UPDI-Interface)

Anschluß über UPDI-Interface.

Unterstützte Controller:

ATmega3208,	ATmega3209,	ATmega4808,
ATmega4809,	ATtiny1614,	ATtiny1616,
ATtiny1617,	ATtiny212,	ATtiny214,
ATtiny3214,	ATtiny3216,	ATtiny3217,
ATtiny412,	ATtiny414,	ATtiny416,
ATtiny417,	ATtiny814,	ATtiny816,
ATtiny817		



---

## 3 Technische Daten

- Unterstützte Controller STM32F1 mit JTAG-Interface:
  - alle Derivate
- Unterstützte Controller STM32F2 mit JTAG-Interface:
  - alle Derivate
- Unterstützte Controller STM32F3 mit JTAG-Interface:
  - alle Derivate
- Unterstützte Controller STM32F4 mit JTAG-Interface:
  - alle Derivate
- Unterstützte Controller der Atmel-AVR-Serie mit ISP-Interface:
  - AT90
  - ATtiny (nur mit ISP-Interface)
  - ATmega
- Unterstützte Controller der Atmel-XMega-Serie mit PDI-Interface:
  - alle Derivate
- Programmierung des Mikrocontrollers über 10-polige JTAG-Buchse. Für andere JTAG-Anschlüsse sowie für das ISP- und PDI-Interface stehen entsprechende Adapter zur Verfügung.
- Spannungsversorgung über den zu programmierenden Mikrocontroller (2,0 - 5,5 Volt).
- Schreiben und Lesen von:
  - Flash
  - EEPROM (Atmel)
  - RAM (nur STM32)
  - Fusebits (Atmel)
  - Lockbits (Atmel)
- Unterstütztes Dateisystem: FAT32
- Unterstützte Dateiformate:
  - Intel HEX („.HEX“) (I8HEX, I16HEX, I32HEX) (ASCII-Datei)
  - RAW (Binärdatei mit Rohdaten ohne Adreßangabe)
- Unterstützte Speicherkarten-Formate: microSD, microSDHC