# Firmware und Hardware Aalyse

# Basics

# Firmware Analyse

- Umfasst die Analyse jeglicher Daten wie z.B. FW-Images, Disk-Images, Flash-Dumps etc.
- Der Aufbau ist nicht einheitlich und teilweise verschlüsselt oder durch verschiedenen Methoden verschleiert.
- Meist ist der Vorgang nahe an einer forensischen Analyse - nur ohne den ganzen prozeduralen overhead wie Dokumentation, 4-Augen-Prinzip, Ablaufprotokolle usw.
    - Nichtsdestotrotz sollte sorgfältig und mit einer Kopie des Originals gearbeitet werden.
- Tools:
    - Hexeditor, GNU file, GNU strings, binwalk, foremost usw.

# FritzBox FW-Image

1. Schritt: Ermitteln des Datei-Types

   $ file FRITZ.Box_Fon_WLAN_7170.29.04.88.image
   FRITZ.Box_Fon_WLAN_7170.29.04.88.image: **POSIX tar archive (GNU)**

2. Schritt: Entpacken des FW-Images mit Hilfe des tar-Befehls:
   $ tar -xvf FRITZ.Box_Fon_WLAN_7170.29.04.88.image

```
./var/
./var/flash_update.ko
./var/install
./var/chksum
./var/info.txt
./var/tmp/
./var/tmp/filesystem.image
./var/tmp/kernel.image
./var/flash_update.o
./var/regelex
./var/signature
```

# FritzBox FW-Image

3. Schritt: Analyse der entpackten Daten (kleine Auswahl):

   $ file chksum
   chksum: **ELF 32-bit LSB executable, MIPS, MIPS32 version 1 (SYSV), statically linked, stripped**
   $ file install
   install: **POSIX shell script, ISO-8859 text executable**
   $ file regelex
   regelex: **ELF 32-bit LSB executable, MIPS, MIPS32 version 1 (SYSV), statically linked, stripped**
   $ file signature
   signature: **data**
   $ file tmp/kernel.image
   tmp/kernel.image: **data**

## FritzBox FW-Image

- Manchmal bringt binwalk keinen Erfolg:

  $ binwalk kernel.image
  DECIMAL HEXADECIMAL DESCRIPTION
  ---

- Problem: AVM Images haben einen sogenannten NMI vector, der entfernt werden muss.
  - Details siehe https://github.com/Freetz/freetz/blob/master/tools/remove-nmi-vector
- Sobald dieser entfernt wurde, kann mit unsquashfs das kernel.image entpackt werden.
  - Am einfachsten ist es eine entsprechende Freetz-Umgebung einzurichten und damit zu arbeiten.
- Wer interesse daran hat, kann dies gerne als Selbststudium weiter führen ;-)
  - Mind. Tool-Set von freetz, wenn man nicht die ganze freetz-Toolchain installieren möchte: remove-nmi-vector + freetz_bin_functions und SFK - The Swiss File Knife Multi Function Tool

# Zugriff auf Hardware

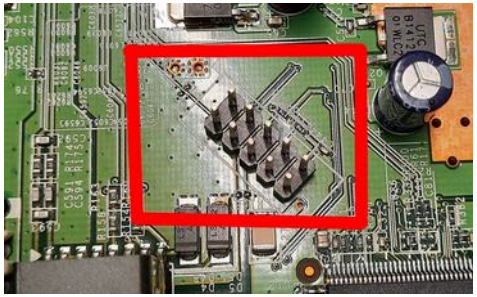
- Oft findet man keine Firmware zum herunterladen oder möchte ein System direkt auf der Hardware analysieren
- Strike, wenn eine Serielle Konsole (UART) auf dem System verfügbar ist - was sehr oft vorkommt :-)
- Evtl. ist auch JTAG (Debug- und Programmier-Schnittstelle) vorhanden
  - geht auch, meist benötigt man aber weitere Tools aus der Build-Chain des Prozessors etc. um z.B. auf den Flash zugreifen zu können.
- Manche Systeme nutzen einen Serielle Schnittstelle, die auch zum Programmieren genutzt werden kann, wie z.B. Arduinio-, ESP32- und ES8266-Boards.
- Wenn alle Stricke reißen, muss man direkt an den Speicher ran - was sehr aufwendig werden kann.

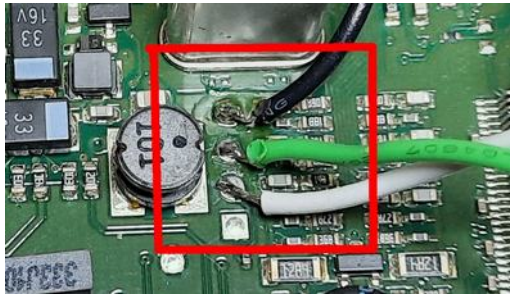
# UART bzw. Serielle Schnittstelle

- UART => Universal Asynchronous Receiver Transmitter
- “Eine UART-Schnittstelle dient zum Senden und Empfangen von Daten über eine Datenleitung und bildet den Standard der seriellen Schnittstellen”
  - aus Wikipedia - Details siehe https://de.wikipedia.org/wiki/Universal_Asynchronous_Receiver_Transmitter
- Früher eine weit Verbreitete Schnittstelle bei PC und anderen Computersystemen zum Datenaustausch (RS-232 oder RS-422)
  - Direkt über sog. Null-Modem-Kabel oder über Modems undTelefonleitung
- Andere weit verbreitete Serielle Schnittstellen:
  - CAN-BUS, Ethernet, RS-485 usw.
- Weiterführende Infos: https://de.wikipedia.org/wiki/Serielle_Schnittstelle

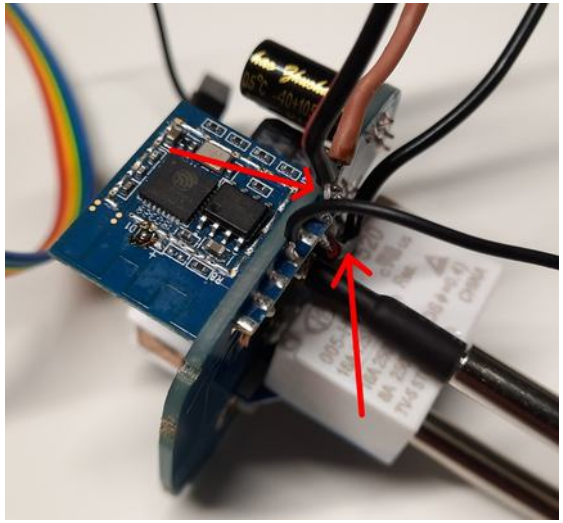
# UART auf Platinen / Embedded Geräten

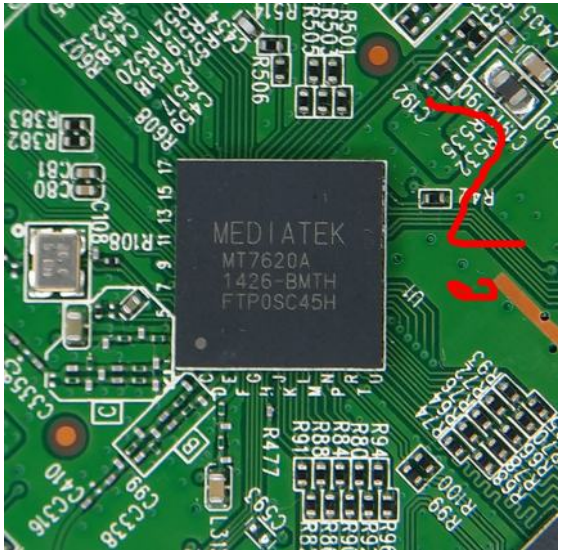
- Meist ist keine Sub-D Stecker wie die RS-232 bei einem PC verfügbar
- Wenn man Glück hat existiert ein Pfostenstecker oder zumindest einen entsprechende Vorsehung auf der Platine
- Ansonsten muss man die ICs auf der Platine analysieren und ggfls. Kabel direkt an die Pins der ICs oder andere taktisch gut gelegene Stellen auf der Platine anlöten.





# UART auf Platinen / Embedded Geräten



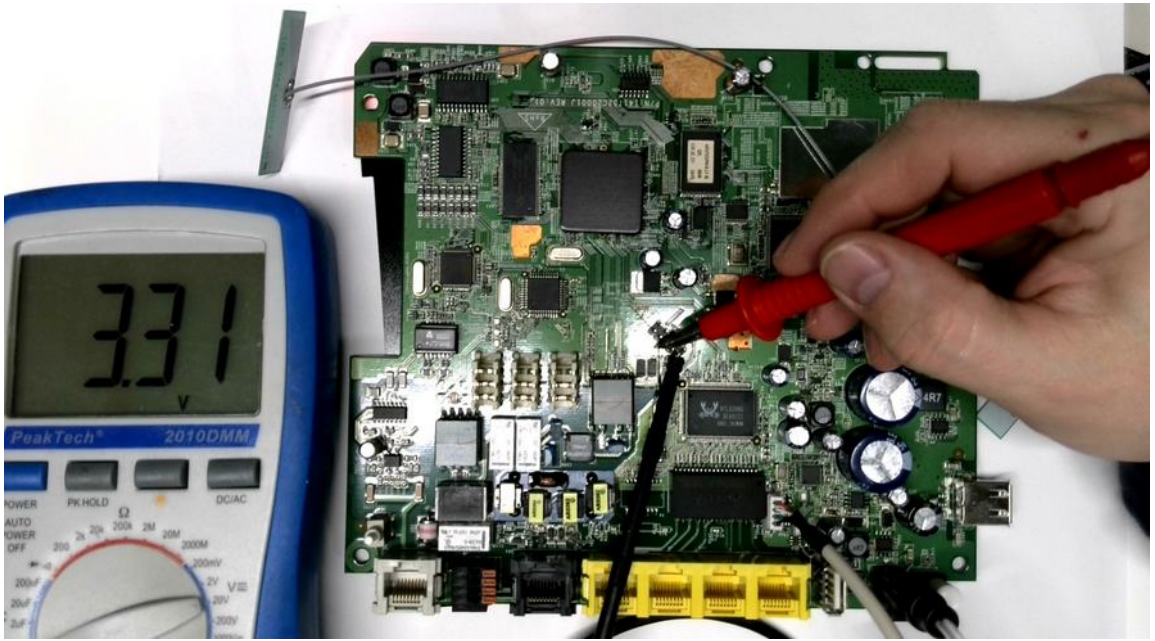
## UART auf Platinen / Embedded Geräten

- Mit das wichtigste Tool für den UART Zugriff sind entsprechende Adapter
- RS-232 arbeitet mit mind +/-12 V => daher kein USB-TTL Adapter nutzen!
- Ansonsten sollte man auf darauf achten, mit welchen Pegel gearbeitet wird.
  - TTL Standard nutzt 5,0 V
  - Neuere Systeme nutzen oft 3,3 V oder sogar 1,8 V
  - Daher: Eine zu hohe Spannung für TX (transmit data) kann den UART-Anschluss auf dem System oder des Adapters zerstören.

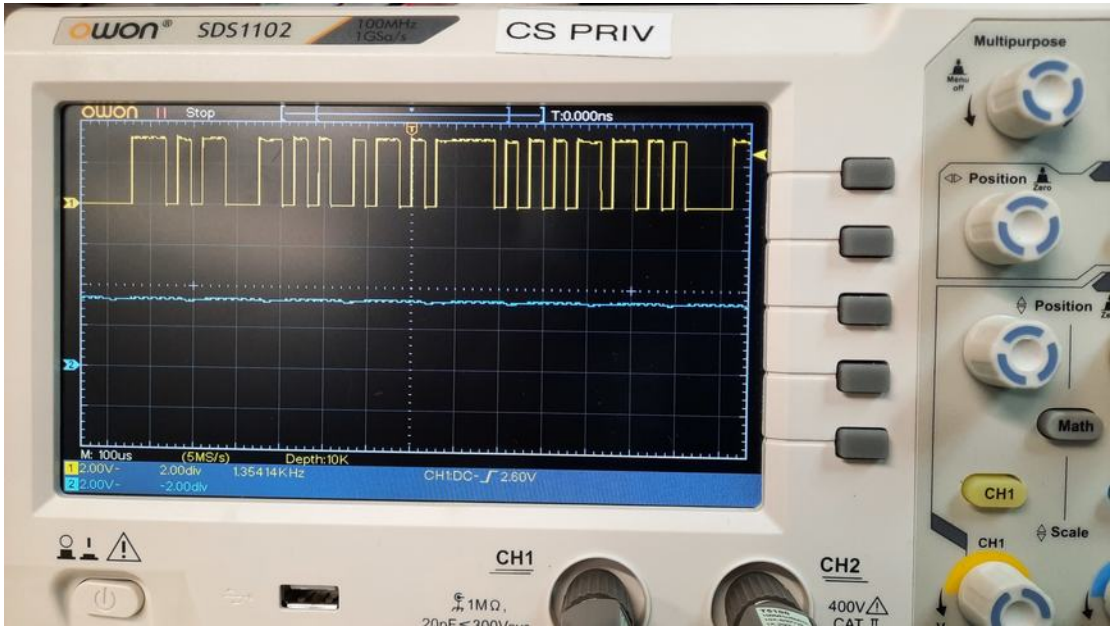
# Was ist, wenn ich das Pin-Layout nicht kenne?

- Mit etwas Glück findet man zu den meisten Systemen eine Info über das Pin-Layout der UART Schnittstelle - Google und Co. sind dein Freund.
- Ansonsten muss man die Pins durchtesten:
  - Identifikation der Ground-Pins
  - Spannungswerte an den Pins können hilfreich sein (TTL UART hat meistn zwischen 2-5 V Spannung anliegen)
  - Ggfls. der Widerstandswert der einzelnen Pins gegen Ground (TTL Schnittstellen nutzen meist Pull-Up oder -Down Widerstände)
  - Logic-Analyse und/oder Oszilloskop sind nützlich um TX seitens des Microcontroller zu finden

# Messung der Spannung mit Multimeter

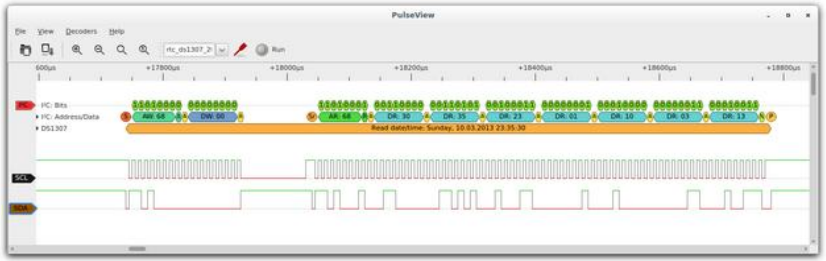


# Analyse des Signals mit Oszilloskop



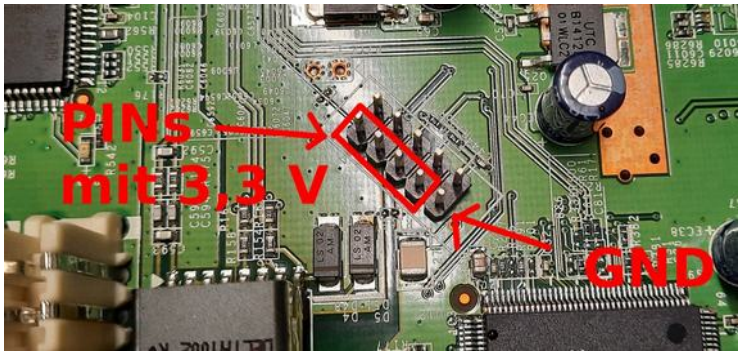
# Analyse des Signals mit Logic-Analyser

- Da ein Oszilloskop nicht gerade günstig ist, bietet sich die kostengünstige Version Spannungsmessung und “15,- Euro” 24 MHz Logic-Analyser an (China Saleae Fake) an.
- Unter Linux kann dieser Logic-Analyser mit Hilfe der Sigrok-Tools und PulseView einfach genutzt werden
  - Sigrok-cli: https://sigrok.org/wiki/Sigrok-cli
  - PulseView (GUI): https://sigrok.org/wiki/PulseView



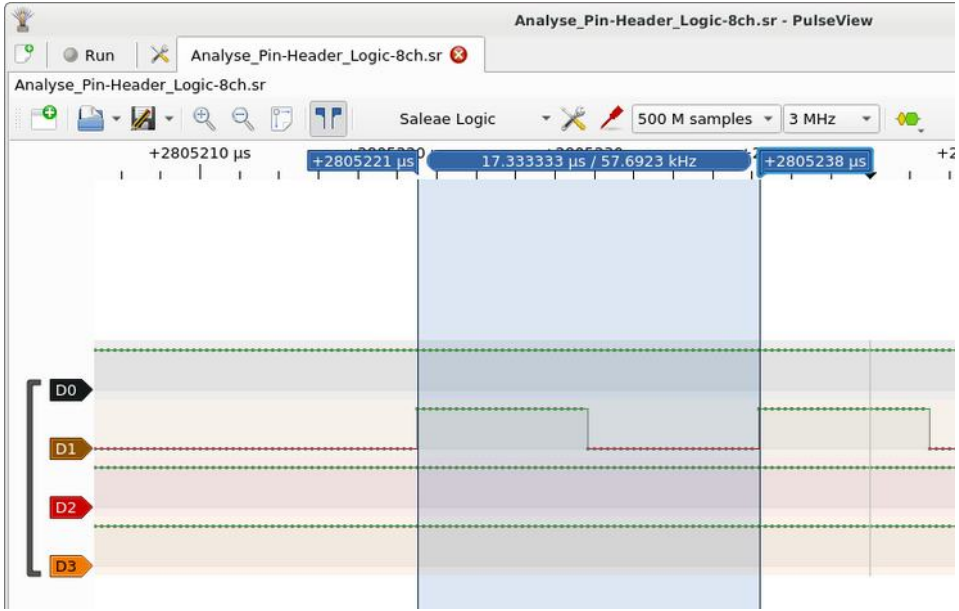

# PulseView und Serielle Schnittstelle EasyBox

- Auf der Platine ist ein Pin-Header vorhanden, zu dem keine Infos bekannt sind.
- Mit Hilfe eines Durchgangsprüfer und Spannungsmesser konnte GND und 4 Pins mit 3,3 V festgestellt werden.
- Zur weiteren Analyse wird ein Logic-Analyser mit GND und den vier übrigen PINs verbunden.
- Da unklar ist, mit welcher Geschwindigkeit die Schnittstelle arbeitet, muss eine ausreichend hohe Abtastrate genutzt werden.
- Nyquist-Shannon-Abtasttheorem beachten!



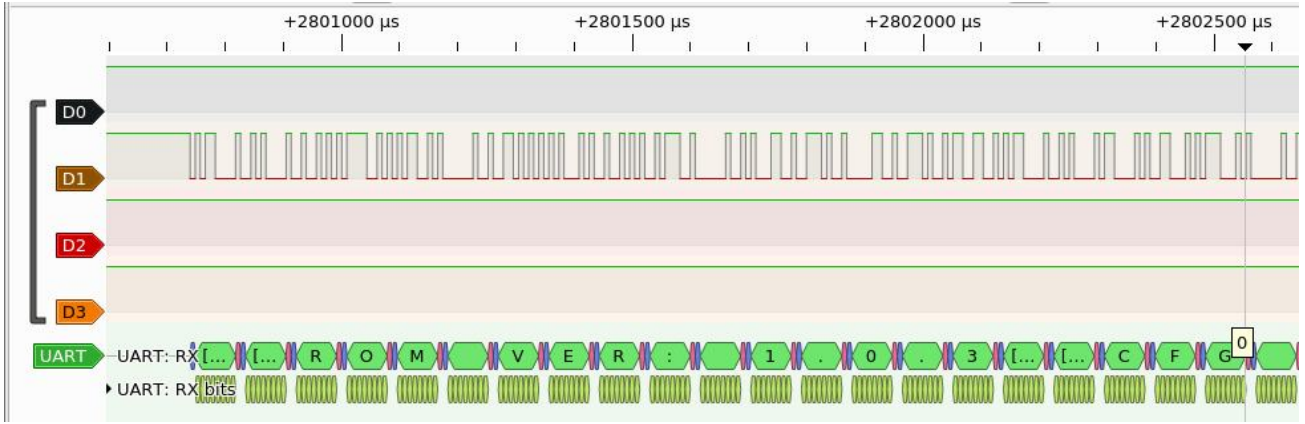
# PulseView und Serielle Schnittstelle EasyBox

- Mit Hilfe von PulseView können nun die Logikpegel analysiert werden.
- Nur der Kanal D1 liefert kontinuierlich Daten. Das ist mit hoher Wahrscheinlichkeit ein TX-Ausgang
- Die Pulslänge (high+low = 2 bit) beträgt ca. 17,33 µS
  => für die Baudrate muss dieser Wert noch durch 2 geteilt werden: 8,66 µS (115,384837 kHz) was 115.200 bit/s entspricht.



# PulseView und Serielle Schnittstelle EasyBox

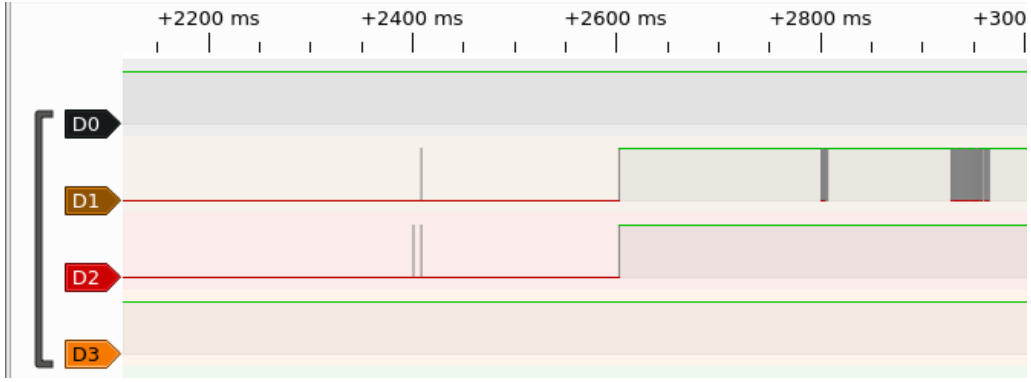
- Mit Hilfe der Protokoll-Decoder von PulseView kann das Signal als UART und korrekten Baudrate decodiert werden:




Ergebnis: “...ROM VER: 1.0.3...CFG…”

# PulseView und Serielle Schnittstelle EasyBox

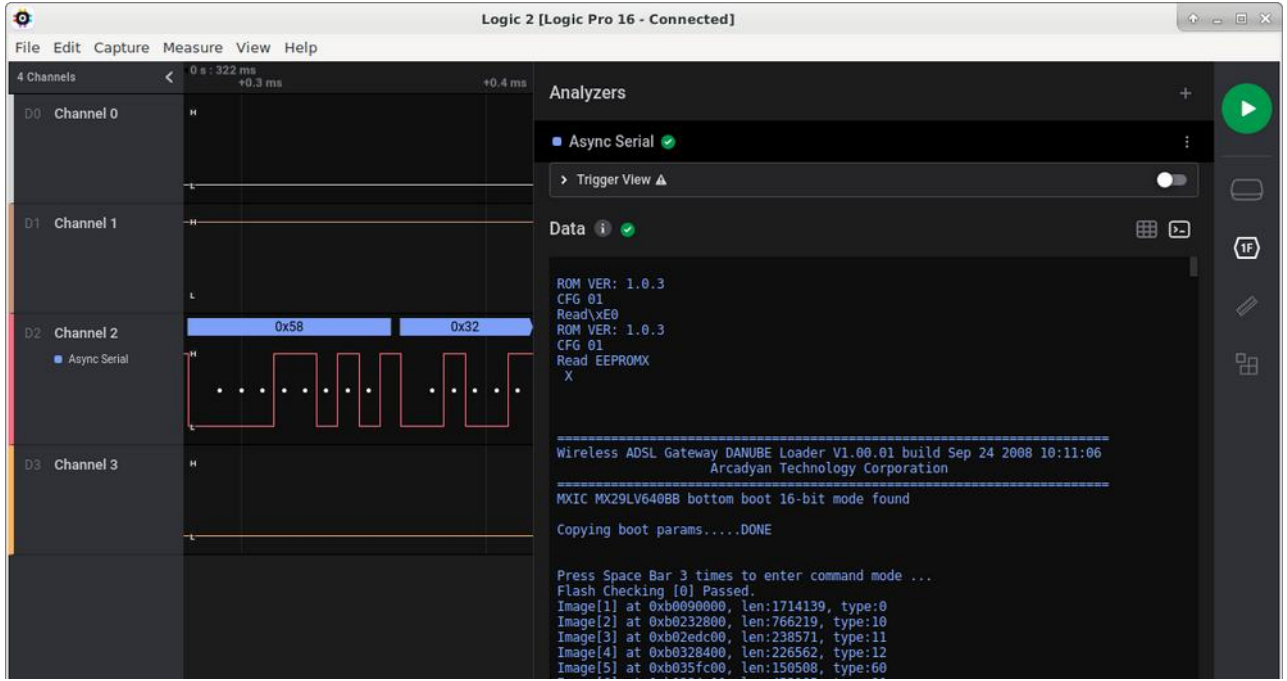
- Weitere Information mit Hilfe des Logic-Analysers:
  - Neben dem kontinuierlichen Signal auf Kanal D1 ist ein konstanter Pegelanstieg auf Kanal D2 zu erkennen.
    Dies lässt auf den möglichen RX-Eingang schließen.
  - Auf den anderen Kanälen D0 und D3 sind keinen Pegeländerungen zu erkennen



# PulseView und Serielle Schnittstelle EasyBox

- Mit Hilfe des sigrok-cli Tools und der Sigrok-Session Datei aus PulseView kann die komplette Session als UART nach ASCII decodiert und ausgegeben werden:
  ROM VER: 1.0.3
  CFG 01
  Readï¿½
  ROM VER: 1.0.3
  CFG 01
  Read EEPROMX
  X
  ==========
  Wireless ADSL Gateway DANUBE Loader V1.00.01 build Sep 24 2008 10:11:06
  Arcadyan Technology Corporation
  ==========

# Logic-Analyser - Luxusversion



# Übung PulseView

# Übung PulseView

- Gehen sie in Moodle zur Aufgabe “PulseView / Sigrok Analyse” und laden Sie die Sigrok Session Datei herunter.
- Analysieren Sie die Datei in PulseView und nutzen Sie den UART Decoder
- Nutzen sie sigrok-cli um den Inhalt der Datei so zu decodieren, dass die UART-Kommunikation im Klartext angezeigt werden kann.
  Hinweis: Nutzen SIe die entsprechenden Funktionen und Parameter von sigrok-cli ( https://sigrok.org/wiki/Sigrok-cli)

# Übung PulseView

- Welche Einstellungen müssen Sie für den UART Decode wählen, um das Serielle Signal fehlerfrei decodiert angezeigt zu bekommen?
  - Die Baudrate muss auf 115200 eingestellt werden.

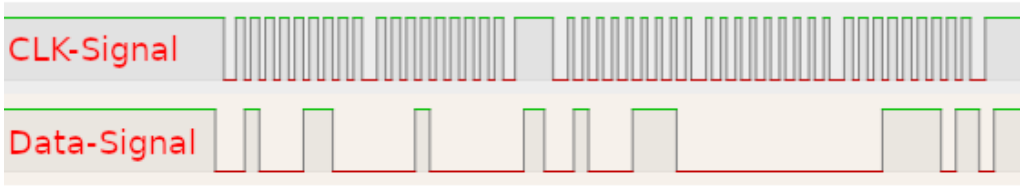
- Mit welchen sigrok-CLI Parameter konnten sie das Signal decodiert in der Konsole darstellen?
  - $ sigrok-cli -i Analyse_Pin-Header_Logic-8ch.sr -P uart:baudrate=115200:format=ascii:tx=D1 -B uart=tx
    führt zum gewünschten Ergebnis.

# Übung Bus-Systeme: I²C und SPI

# I²C - Zum Einstieg

# I²C - Zum Einstieg

- I²C oder I2C (Inter-Integrated Circuit) ist ein 1982 entwickelter serieller Datenbus mit 2 Signalleitungen
    - Taktleitung: SCL = Serial Clock
    - Datenleitung: SDA = Serial Data
- Die einzelnen Teilnehmer werden im Master/Slave Modus über Adressen angesprochen.
- Folgende Geschwindigkeiten werden unterstützt:
    - 0,1 (Standard), 0,4, 1,0, 2,0, 3,4 und 5.0 (Ultra Fast Mode) MBit/s
- Weitere Infos und Details zum Protokoll unter https://de.wikipedia.org/wiki/I%C2%B2C

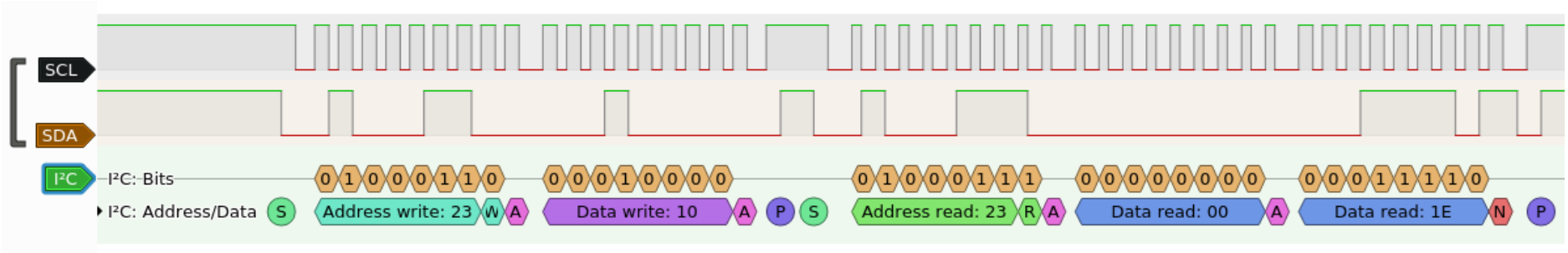


# I²C Übung

- Gehen sie in Moodle zur Aufgabe “Daten Bus I2C” und laden Sie das Archiv mit den Übungsdaten herunter.
  - Inhalt: zwei Sigrok-Sessiondateien und ein PDF (Datasheet)
- Öffnen Sie als erste i2c_sniff.sr in Pulseview und nutzen Sie den I²C Decoder zum analysieren der Daten.
- Sie können auch zusätzlich die Daten mit sigrok-cli extrahieren - siehe UART-Übung (nur diesmal mit dem I2C Decoder).
- Fragen:
  - Woran lässt sich der SCL (Clock) eindeutig vom SDA (Datensignal) unterscheiden?
  - Welche Werte wurden übermittelt? (Siehe Funktion der I²C Teilnehmers aus Datenblatt)
- Öffnen Sie als zweites i2c_sniff_2.sr
  - Was fällt auf und was ist Ursache dieses Verhaltens?

# I²C Übung - Lösung

Anzeige “i2c_sniff.sr” in Pulseview mit I²C Decoder:




Sigrok-cli Kommandos:

$ sigrok-cli -i i2c_sniff.sr -P i2c:scl=D0:sda=D1 | less **oder**

$ sigrok-cli -i i2c_sniff.sr -P i2c:scl=D0:sda=D1 > i2c_sniff.sigrok_out_1.txt

$ sigrok-cli -i i2c_sniff.sr -P i2c:scl=D0:sda=D1 -B i2c | hexdump -C **oder**

$ sigrok-cli -i i2c_sniff.sr -P i2c:scl=D0:sda=D1 -B i2c > i2c_sniff.sigrok_out_2.raw

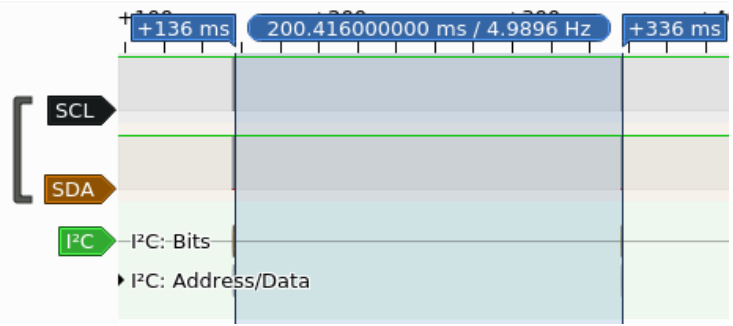
# I²C Übung - Lösung

- Woran lässt sich der SCL (Clock) eindeutig vom SDA (Datensignal) unterscheiden?
  - SCL ist ein 8-Takte langes und regelmäßiges Signal. Es sind immer kleine Pausen zwischen den 8 Takten zu erkennen. SDA variiert bzgl. der abgefragten und empfangenen Daten.
- Welche Werte wurden übermittelt? (Siehe Funktion der I²C Teilnehmers aus Datenblatt)
  - Der Teilnehmer ist ein BH1750 Lichtsensor und gibt die aktuellen Lichtwerte in LUX aus (folgen immer nach Address read),
  - Es wird der Modus 0x10 (0001000b) genutzt (H-Resolution Mode)
- Öffnen Sie als zweites i2c_sniff_2.sr / Was fällt auf und was ist Ursache dieses Verhaltens?
  - Alle Werte sind identisch (0x03) - scheinbar änderten sich diese nie (was unrealistisch ist, außer bei absoluter Dunkelheit oder maximal messbare Helligkeit.
  - Die Abfrage der Werte erfolgt alle 100 ms - da der Sensor H-Resolution Mode nutzt (120-180 ms timing), führt dies zu einem Timing-Fehler und somit wird immer der zuerst ermittelte Wert ausgelesen.

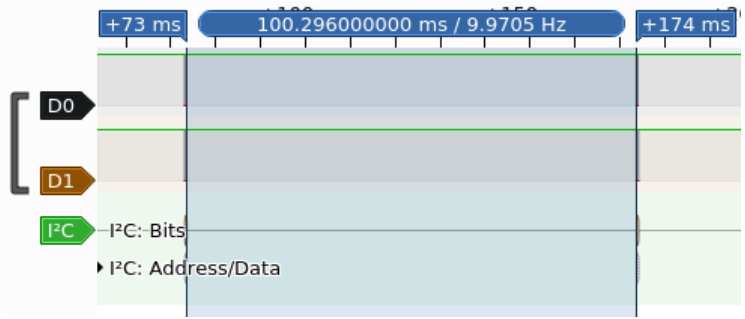
# I²C Übung - Lösung

Abstand zwischen dem Datenabruf

I2c_sniff.sr (200 ms - OK)




i2c_sniff_2.sr (100 ms - NOK)

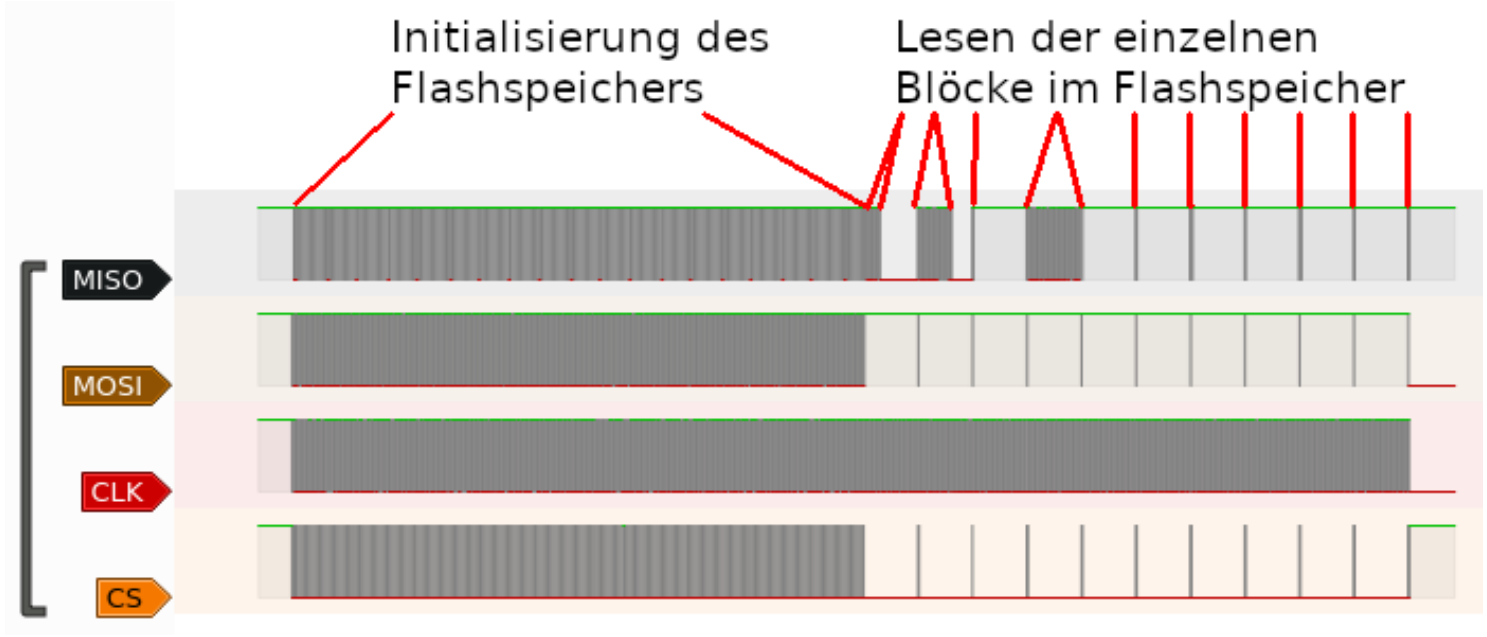


# SPI - Daten Lesen und Schreiben

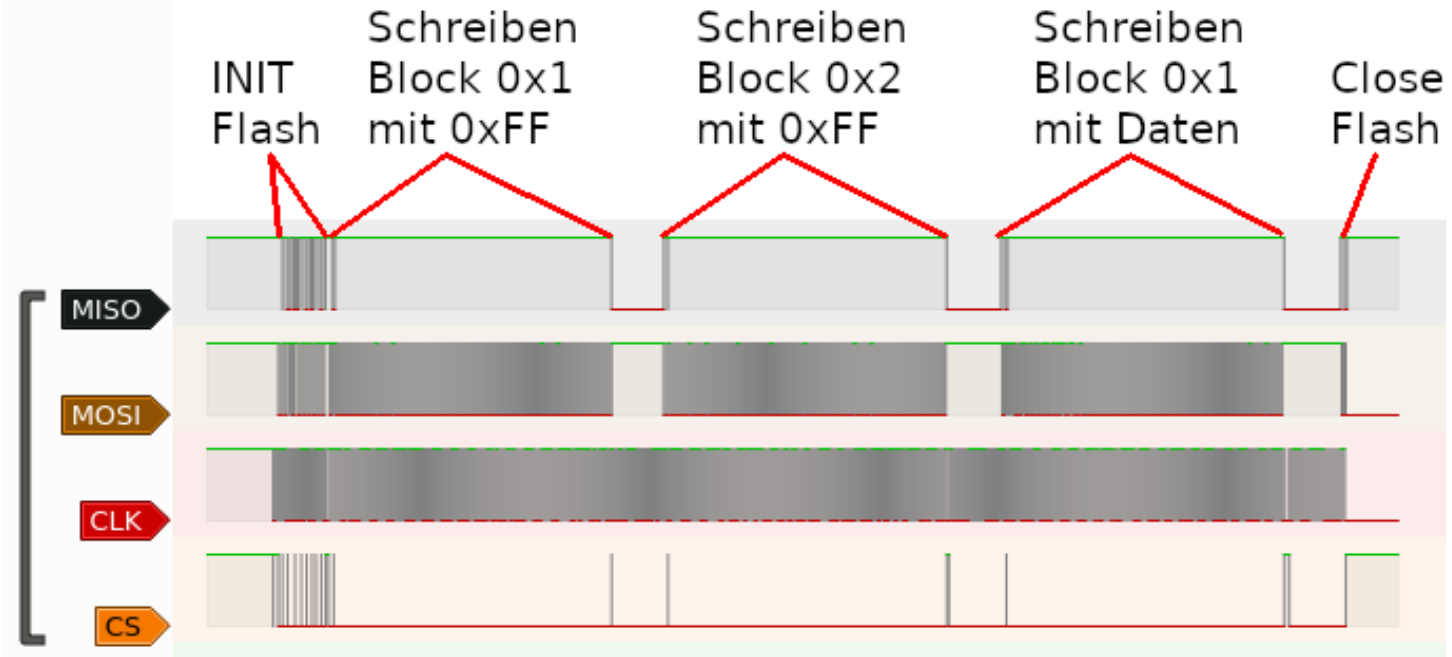
# SPI Daten lesen und schreiben - Übung

- Gehen sie in Moodle zur Aufgabe “SPI Daten lesen und schreiben” und laden Sie das Archiv mit den Übungsdaten herunter.
  - Inhalt: drei Sigrok-Session-Dateien und ein Dump des Flash-Inhalts
- Öffnen Sie Session-Dateien in Pulseview und nutzen Sie den SPI Decoder zum analysieren der Daten (MOSI, MISO, CLK und CS sind bereits vorgegeben)
  **Tipp:** Von hinten nach vorne analysieren.
  - Was ist der Unterschied zwischen MOSI und MISO? Wie könnte man diese identifizieren?
  - Was bedeutet das beim Lesen und beim Schreiben in den Flash?
  - Was ist der Overhead neben den eigentlichen Roh-Daten beim Lesen und Schreiben?
  - Warum ist dieser Overhead im Dump der Flashspeichers nicht vorhanden?
  - In welchem Abstand liegen die Daten im Flaschspeicher?
- Sie können auch zusätzlich die Daten mit sigrok-cli extrahieren - siehe UART-Übung (nur diesmal mit dem SPI Decoder).
- Nutzen Sie zur Analyse des Flashspeicher-Dumps einen Hex-Viewer/Editor.

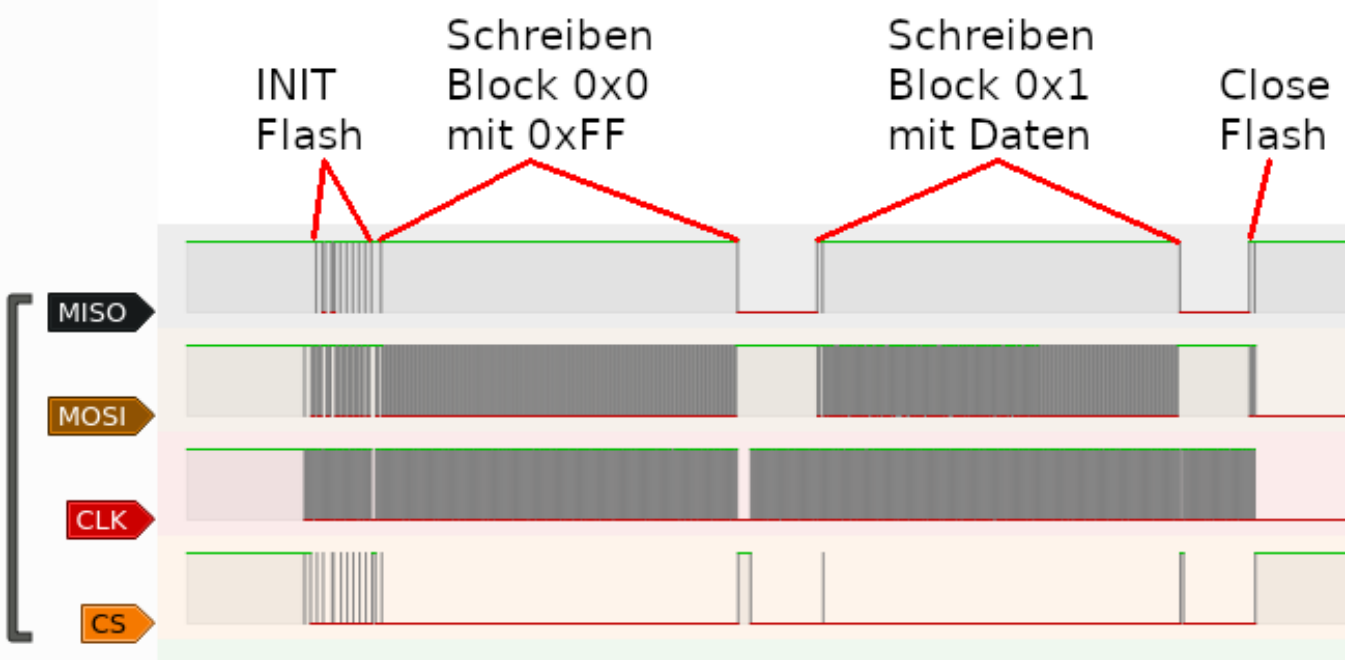
# SPI Daten lesen - Lösung (sd-card_raw_read.sr)



# SPI Daten schreiben - Lösung (sd-card_raw_write_01.sr)



# SPI Daten schreiben - Lösung (sd-card_raw_write_02.sr)



# SPI Daten lesen und schreiben - Lösung

**Was ist der Unterschied zwischen MOSI und MISO?**

- MOSI => MasterOutSlaveIn: Alle Daten, die vom SPI Master an den SPI Slave gehen.
- MISO => MasterInSlaveOut: Alle Daten, die vom SPI Slave an den SPI Master gehen.

**Was bedeutet das beim Lesen und beim Schreiben in den Flash?**

- Die Richtung und damit Nutzung der MOSI/MISO Datenleitung wird anders genutzt, um die Daten auf den Flasch zu speichern bzw. vom Flash zu lesen.

**Was ist der Overhead neben den eigentlichen Rohdaten beim Lesen und Schreiben?**

- Das sind einmal die Initialisierungsdaten für den Flash-Zugriff, und die Adressierung der Daten zu lesen- und schreibenden Daten.

# SPI Daten lesen und schreiben - Lösung

**Warum ist dieser Overhead im Dump der Flashspeichers nicht vorhanden?**

- Die Daten sind direkt in eindeutigen Blöcken im Flaschspeicher abgelegt

**In welchem Abstand liegen die Daten im Flaschspeicher?**

- 512 kb => Damit ist ein Datenblock 512 kb groß

**sigrok-cli Befehle:**

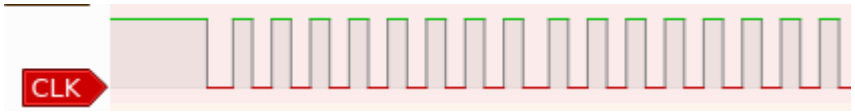
- $ sigrok-cli -i sd-card_raw_read.sr -P spi:wordsize=8:miso=MISO:mosi=MOSI:clk=CLK:cs=CS -B spi=miso | hexdump -C
- $ sigrok-cli -i sd-card_raw_read.sr -P spi:wordsize=8:miso=MISO:mosi=MOSI:clk=CLK:cs=CS -B spi=mosi | hexdump -C

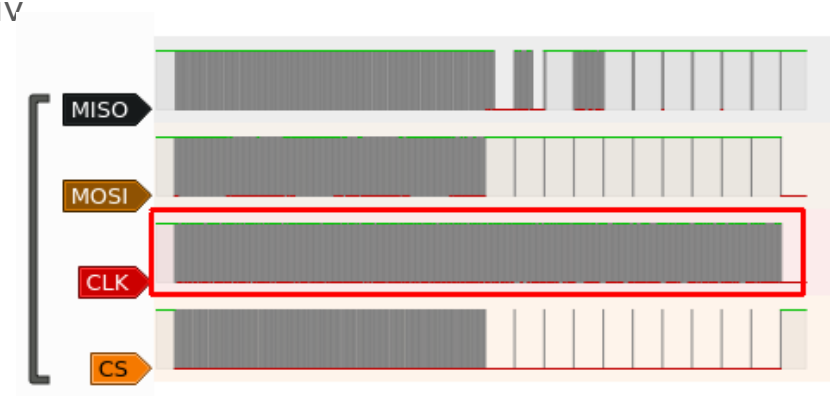
# SPI Daten lesen und schreiben - Lösung Zusatz

Wenn MOSI, MISO, CLK und CS unbekannt sind, kann man diese recht einfach erkennen

**CLK:**

- Ist immer so lange aktiv, wie der Zugriff aktiv Ist - siehe rote Umrandung.
- Zusätzlich ist das Signal kontinuierlich - mit kurzen Pausen.
- Damit kann auch die Word Size abgelesen werden - hier im Beispiel 8 bit.

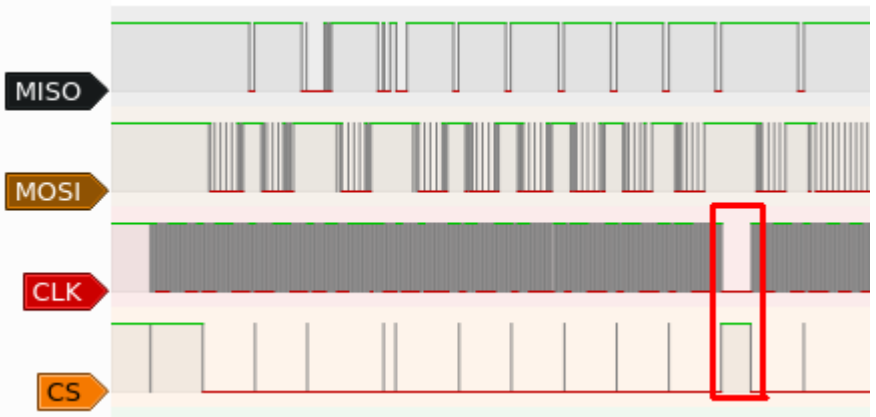


# SPI Daten lesen und schreiben - Lösung Zusatz

**CS (Chip Select):**

- CS ist entsprechend aktiv bei der Datenübertragung (active low)
- Und pausiert bei längeren CLK Pausen zwischen dem Daten-austausch (siehe roter Kasten)

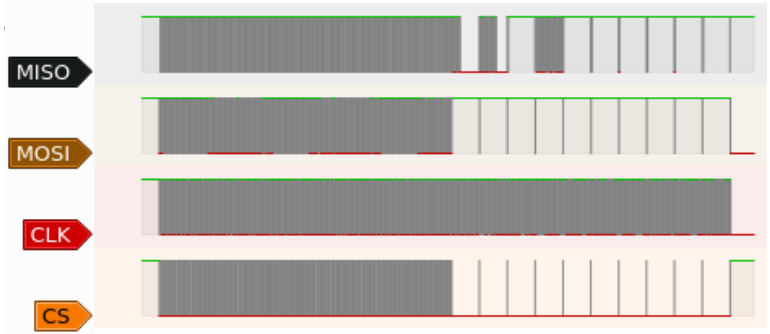


# SPI Daten lesen und schreiben - Lösung Zusatz

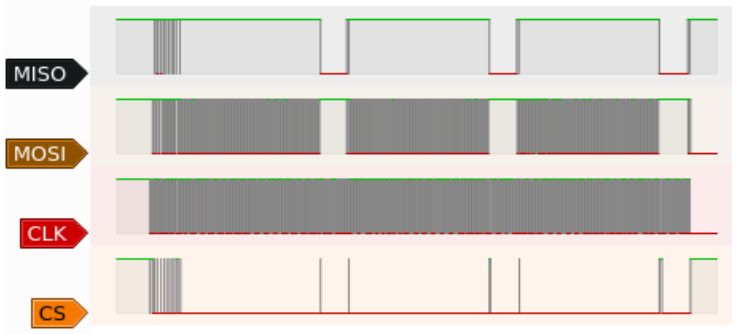
**MISO und MOSI:**

- Muss aus dem Kontext evaluiert werden - da meist asynchron zugegriffen wird, kann es an den übertragenen Datenmengen abgeleitet werden ;-)

Lesender Zugriff: **MIS0 aktiver**




Schreibender Zugriff: **MOSI**



# SPI - SD Card Zugriff

# SPI SD-Card Zugriff - Übung

In einem System ist eine fest verbaute SD-Card, die für die Datenspeicherung genutzt wird und über SPI angesprochen wird.

- Gehen sie in Moodle zur Aufgabe “SPI SD-Card Zugriff” und laden Sie das Archiv mit den Übungsdaten herunter.
  - Inhalt: eine Sigrok-Session-Dateien und eine Konsolenausgabe.
- Öffnen Sie Session-Dateien in Pulseview und nutzen Sie den SPI Decoder zum analysieren der Daten (MOSI, MISO, CLK und CS sind bereits vorgegeben)
- Nutzen Sie auf jeden Fall für die Analyse der Daten die sigrok-cli mit dem SPI Decoder - die reine Analyse in PulseView ist sehr mühsam.

# SPI SD-Card Zugriff - Übung

**Fragen:**

- Was ist auffällig bei der Analyse der Sigrok-Session Datei?
- Welches Dateisystem auf der SD-Karte wird genutzt?
- Sind die drei in der Konsole aufgelisteten Daten die einzigen Daten auf der SD-Card?
  - Wenn ja, wie heist die Datei und können Sie den Inhalt ermitteln?
- Welche Inhalte können Sie aus der Sigrok-Session Datei extrahieren?

# SPI SD-Card Zugriff - Lösung

**Was ist auffällig bei der Analyse der Sigrok-Session Datei?**

- Es sind sehr viel mehr Daten zwischen dem System und der SD-Card ausgetauscht worden.
- Der Overhead besteht aus mehr Informationen, als beim direkten SPI Zugriff - was auf ein Dateisystemzugriff zurückzuführen ist,

**Welches Dateisystem auf der SD-Karte wird genutzt?**

- FAT32
  Mit Hilfe sigrok-cli z.B. die MISO Daten extrahieren - folgender Eintrag ist zu finden:
  00002010  00 80 00 29 f8 4e 16 90  4e 4f 20 4e 41 4d 45 20  |...).N..NO NAME |
  00002020  20 20 20 46 41 54 33 32  20 20 20 00 00 00 00 00  |  **FAT32**  .....|

# SPI SD-Card Zugriff - Lösung

**Sind die drei in der Konsole aufgelisteten Daten die einzigen Daten auf der SD-Card?**

- Nein, es ist noch eine Datei mit dem Namen “ONFIG TXT” vorhanden - siehe MISO Daten:
  00002270  a1 b7 52 04 00 4f 02 00  00 e5 63 00 6f 00 6e 00  |..R..O....**c.o.n.**|
  00002280  66 00 69 00 0f 00 39 67  00 2e 00 74 00 78 00 74  |**f.i...9g...t.x.t**|
  00002290  00 00 00 00 00 ff ff ff  ff e5 4f 4e 46 49 47 20  |..........**ONFIG** |
  000022a0  20 54 58 54 20 00 31 fb  6d b8 52 b8 52 00 00 fb  | **TXT** .1.m.R.R...|
- Auf den Inhalt der Datei kann über die Sigrok-Session nicht zugegriffen werden, Es handelts sich um eine gelöschte Datei, die in der FAT als gelöscht markiert wurde. Ein Flash-Dump könnte ggfl. den Inhalt der Datei rekonstruieren, falls diese noch nicht überschrieben wurde.

# SPI SD-Card Zugriff - Lösung

**Welche Inhalte können Sie aus der Sigrok-Session Datei extrahieren?**

- Es können über die **MISO Daten** die Inhalte der “config.cnf” und “logfile.log” extrahiert werden:

**config.cnf:**

```
[CONFIG]
version=1.1

[NETWORK]
ip=10.12.13.121

netmask=255.255.255.0
gateway=10.12.13.1
logserver=10.12.12.10

[LOGIN]
user=loguser
passwd=5V:I(-
```

**logfile.log:**

```
START-LOG-FILE
```

# SPI SD-Card Zugriff - Lösung

**Welche Inhalte können Sie aus der Sigrok-Session Datei extrahieren?**

- Es können über die **MOSI Daten** die Inhalte extrahiert werden, die in “logfile.log” geschrieben wurden:

**logfile.log:**

```
START-LOG-FILE
log::2021-05-12--12-21-42::readEntrySuccess
log::2021-05-12--12-21-43::readEntrySensorSuccess
log::2021-05-12--12-21-44::readEntrySensorValue::24.2
log::2021-05-12--12-21-45::readEntrySuccess
log::2021-05-12--12-21-46::readEntrySensorSuccess
log::2021-05-12--12-21-47::readEntrySensorValue::23.9
```

# SPI SD-Card Zugriff - Lösung

**Sigrok-cli Befehle zum extrahieren der Daten:**

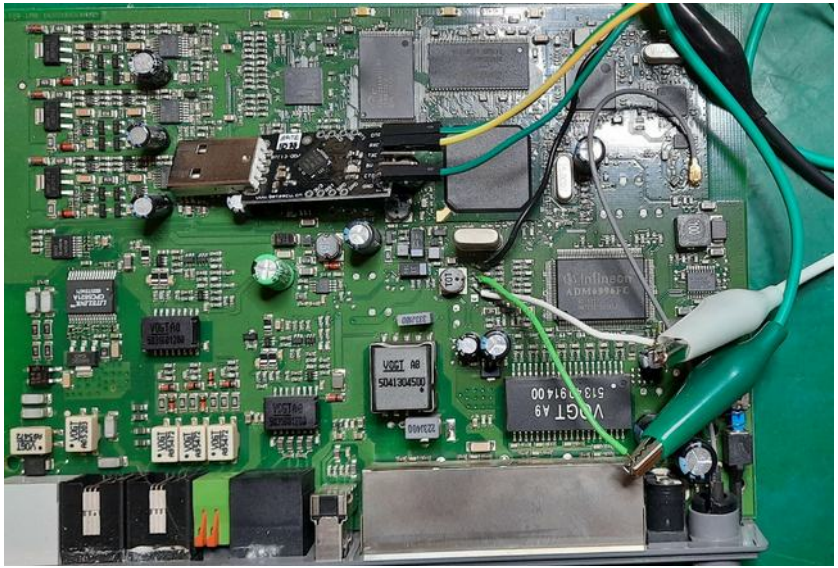
- **MOSI Daten** (Daten, die auf die SD-Card geschrieben werden - bzw. Befehle zur Karte/Slave):
  $ sigrok-cli -i sd-card-spi_config.sr -P spi:wordsize=8:miso=MISO:mosi=MOSI:clk=CLK:cs=CS -B spi=mosi | hexdump -C
- **MISO Daten** (Daten, die von der SD-Card gelesen werden):
  $ sigrok-cli -i sd-card-spi_config.sr -P spi:wordsize=8:miso=MISO:mosi=MOSI:clk=CLK:cs=CS -B spi=miso | hexdump -C

Alternativ können die Ausgaben auch in einen Datei mit Hilfe von ”> <dateiname>” anstatt “| hexdump -C” umgeleitet werden, und so von einem beliebigen anderen Programm geöffnet werden.

# FritzBox - Direkter Zugang über Serielle Konsole

# FritzBox und die Serielle Schnittstelle

- Die meisten FritzBoxen haben eine aktive Serielle Schnittstelle vorhanden, über die auf den Bootloader als auf ohne weiter Authentifizierung auf die OS-Konsole zugegriffen werden kann.
- Über diesen Zugang kann man sehr tief in das System eindringen und auf unterschiedliche Daten zugreifen - ohne das GUI-Passwort kennen zu müssen.



# FritzBox und die Serielle Schnittstelle

- Der Zugriff erfolgt per USB TTL Adapter und einem beliebigen Terminal-Programm wie z.B. mincom unter Linux.
  - 38400 baud, 8N1

```
(AVM) EVA Revision: 1.393 Version: 1393
(C) Copyright 2005 AVM Date: Nov 26 2007 Time: 14:05:07 (0) 2 0x0-0x41F

[FLASH:] SPANSION Top-MirrorBit-Flash 8MB 32 Bytes WriteBuffer
[FLASH:](Eraseregion [0] 127 sectors a 64kB)
[FLASH:](Eraseregion [1] 8 sectors a 8kB)
[SYSTEM:] OHIO on 211MHz/125MHz

Eva_AVM >AVM decompress Kernel:
.....executeProgram on 0x941DA000
[ohio_pre_init] System Clk = 62500000 Hz

LINUX started...
```

# DEMO FritzBox UART Zugriff

# Übung FritzBox

# FritzBox - Übung

Gehen sie in Moodle zur Aufgabe “FritzBox Dump” und laden Sie die Sigrok Session Datei herunter.

- Aufgabe: Extrahieren Sie die Inhalte des Flash-Speichers und die Kopie des Dateisystems.
  - Welche Daten sind in welcher Form zu finden?
  - Können Credentials in klartext oder verschlüsselt extrahiert werden?

- Die Lösungen werden wir gemeinsam Diskutieren.

# Hausarbeit Image Analyse
# Ein kleiner Ausflug in die Forensik

# Image-Forensik - Übung

Gehen sie in Moodle zur Aufgabe “Image-Forensik” und laden Sie das Archiv herunter und entpacken Sie diese in ein separates Verzeichnis.

Nutzen SIe für die Analyse der drei Images “binwalk” und “foremost” (beide Tools sind unter Kali-Linux vorhanden). Sie können gerne auch andere Forensik-Tool einsetzen.

- Was sind die Unterschiede bei der Extraktion der Daten mit den beiden Tools?
- Welche Daten können aus den Image *cf_disk_fat32_part_data_delete+wipe.img* und *cf_disk_fat32_part_data_after_new_format.img* wieder hergestellt werden?
- Was können Sie zur Sicherheit zu folgenden Befehlen/Aktionen aussagen im Hinblick zu den beiden Images?
    - Einsatz von rm (Linux) bzw. delete (Windows) von Dateien auf einem FAT-Dateisystem.
    - Einsatz von format als FAT32 Dateisystem (Unter Linux als auch Windows).
    - Einsatz von wipe (linux) mit und ohne der Option “-f”.

# Image-Forensik - Lösung

Folgt zur letzten Vorlesung.

# Fazit
# oder was hat das jetzt alle mit IT/Cyber Security zu tun???

# All about Safety - äh Security!

- Hacken von Hardware ist mittlerweile mit kleinem Geld möglich - also unter 1000,- EUR
  - und mit der Minimal-Austattung für um 100,- EUR (China Logic Analyser, Arduino, Multimeter, Lötkolben, TTY USB…)
- Security war lange Zeit nicht im Fokus - haupt Ausreden:
  - Das Equipment dafür ist zu teuer und das können ja nur Profis (ja - war mal so vor 15/20 Jahren).
  - Traut sich eh keine ran, weil ist ja in einem geschlossenen Gehäuse - kein Kommentar ;-)
  - Man muss ein Elektrotechnik Studium dafür machen, Assembler Code können und Mikrocontroller Programmieren können - ist von Vorteil… aber schon lange nicht mehr notwendig.
  - Öffnen des Gerätes ist $weil-überhaupt:
    - Garantieverlust (die endete auch irgendwann - oder billig genug)
    - Unmöglich, weil verklebt, “sonder” Schrauben, vernietet, vergossen - ja, das machts ätzend, usw.
    - Verboten, weil Garantieverlust (nun ja…), AGB/Nutzungsbedingungen (dann halt in der Bucht gebraucht kaufen), verplombt/versiegelt (nix ist unmöglich)
    - Kann es nur an 230/400V betreiben, und das ist dann ja Lebensgefährlich, wenn man es öffnet und daran bastel. **Ja, 230 V können einen töten!** Aber es gibt Trenntrafos und galvanische Trennung der Schnittstellen...

# IT/Cyber Security so wichtig wie noch auch im Embedded Umfeld!

# Keine IT/Cyber Security im Embedded Umfeld

- Die genutzten Schnittstellen sind öffentlich verfügbar - selten proprietäre Bus-System im Einsatz (Kostenfaktor) und Sicherheit durch Verschleierung hilft nur bedingt.
- Bit-Flip (7 bit ASCII z.B. mit 1xxx xxxx anstatt 0xxx xxxx übertragen) erkannt man einfach - und kann Bit-Flip einfach umrechnen.
- Kaum verschlüsselter Datenaustausch über BUS-Schnittstellen (SPI, I²C, UART, RS485 etc.)
  - Begründung: Ist doch nur ein interner Bus - und da kommt ja niemand dran…
  - Wenn gut umgesetzt, dann sind die Datenleitungen zwischen Multilayern auf der Platine versteckt. Das macht es schwierig ran zu kommen - und auch sehr schwer beim Platinenlayout und Routin.
- Daten liegen meist auch ungeschützt im Flash-Speicher des Systems
  - Bei SPI, I²C z.B. kommt man meist recht leicht an den Baustein ran und kann diesen direkt auslesen.
  - Einsatz von parallel ansprechbaren Flash-Speicher benötigt meist spezielle Hardware zum Auslesen!
    - Leider ist meist der OS-Zugriff nicht wirklich geschützt (FritzBox Beispiel) oder JTAG verfügbar...

# IT/Cyber Security im Embedded Umfeld

**Was kann ich besser machen?**

Das hängt davon ab, wie schützenswert die Daten in Rest und Transit sind!

- Informationsklassifizierung, Bedrohungsanalyse und Risk Assessment ist notwendig - und zwar vor dem Design! (Kleine(s) rapid development “Problem” bzw. Herausforderung ;-)
- Nicht immer anwendbar z.B. im kleinst Mikrocontroller Bereich aus performance Gründen. Allerdings stellt sich dann die Frage, ob dieser überhaupt für den Anwendungszweck die geeignete Wahl ist/war.

## IT/Cyber Security im Embedded Umfeld

**Folgende Mindestanforderungen sollten berücksichtigt werden:**

- Bootloader und Console per Passwort schützen
  - Kein Netzwerkboot zulassen! Auch kein Fallback darauf!
- Signaturen nutzen - beim booten und auch beim flaschen
- Verschlüsselung abhängig von der Hardware nutzen (AES128 können die meisten Mikrocontroller on the fly).
- Sicheres Konzept beim Ablegen von sensiblen Daten im Flash-Speicher erarbeiten und umsetzen.
- Read-Only Flash-Speicher nutzen (entweder als ROM oder über Flags im Flash und Mikrocontroller
- Debug Schnittstellen deaktivieren oder mit Passwort schützen.
  - Keine versteckten Schnittstellen, die über bestimmte Spannungspegel aktiviert werden nutzen - das kann gefuzzt werden

# Bests Friends Gehäuse öffnen...

# Gummihammer

Sehr zuverlässig bei geklebten Gehäusen - und richtet (meist) wenig Schaden an :-)

# Dremel / Mini-Flex / Puk-Säge

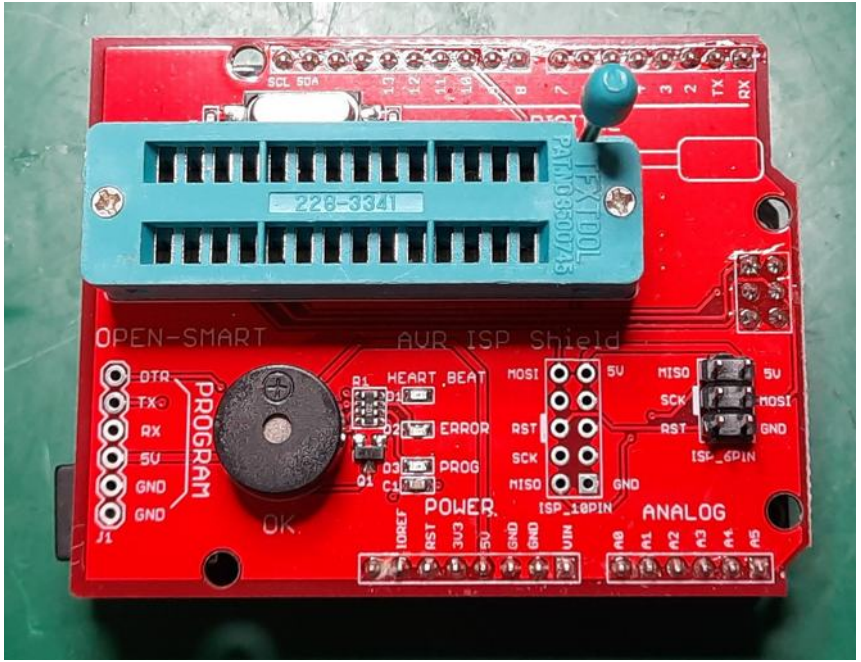
Wenn der Gummihammer nicht funktioniert - dann geht es mit ein wenig “Gewalt” ;-)

# Und ein paar nützliche weitere Tools…

# Universal-Programmer

Arduinio Uno in Kombination
mit dem AVR ISP Shield.

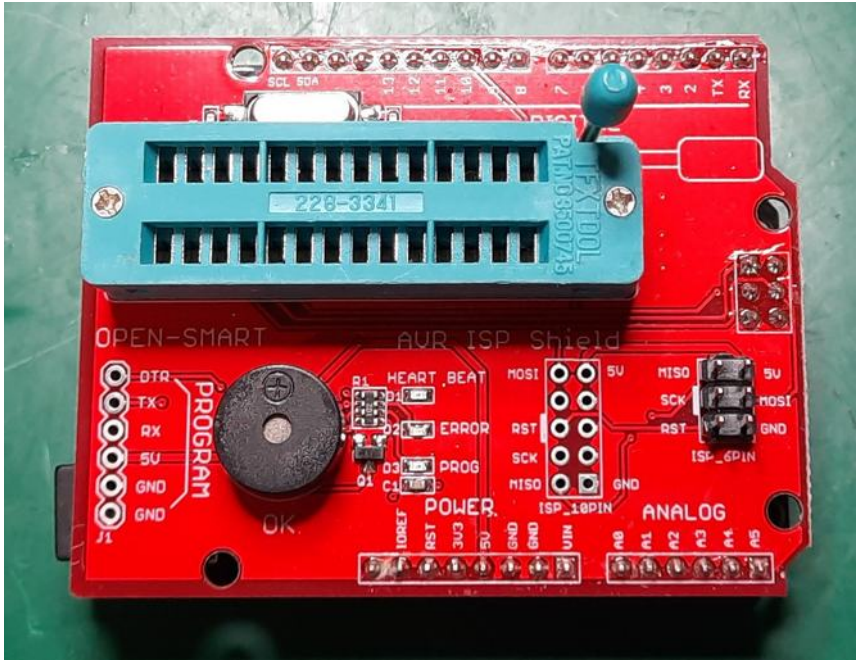
Sehr günstige Lösung für einen
Programmer mit großer
EEPROM usw. Unterstützung.



# Universal-Programmer

Arduinio Uno in Kombination
mit dem AVR ISP Shield.

Sehr günstige Lösung für einen
Programmer mit großer
EEPROM usw. Unterstützung.

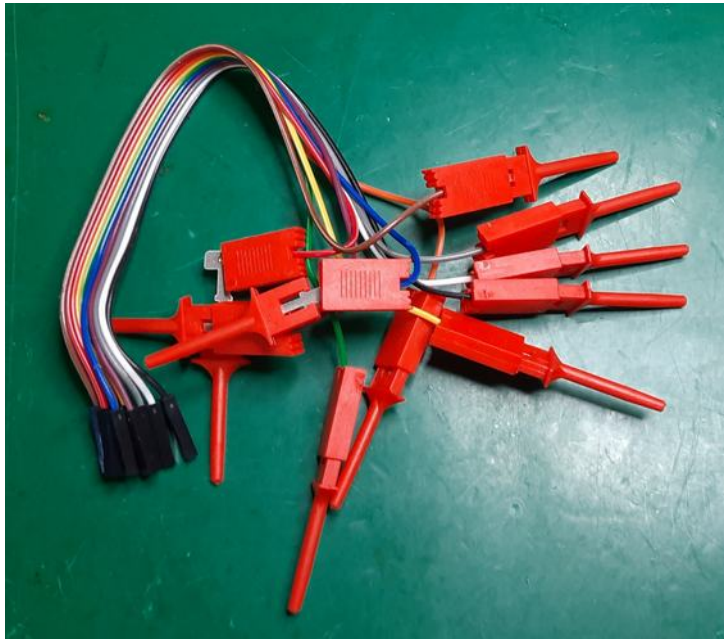


# Testclips und Adapter

Sammlung verschiedener Testclips und Adapter um aufgelötete oder SMD ICs mit dem Universal-Programmer oder Logic-Analyzer zu verbinden.

# Jumper-Kabel mit Testhaken

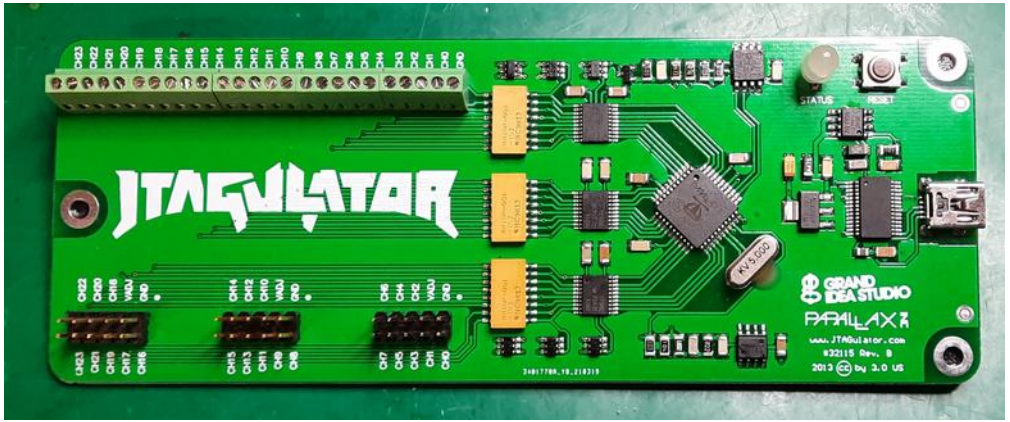
Oft sind Kroko-Klemmen zu „grob“ – Abhilfe schaffen Dupont/Jumper-Kabel mit Testhaken.



# JTagulator

Sobald man auf einer Hardware eine sehr große Anzahl an unbekannten, möglichen Debug- und/oder Daten-Schnittstellen vorfindet, ist die Analyse mit einem 8 oder 16 Port Logic-Analyzer oder Oszilioskop sehr Aufwändigt.

Abhilfe schafft hier der Jtagulator, der bis zu 24 Kanäle parallel analysieren kann.



# Fragen? Diskussion?