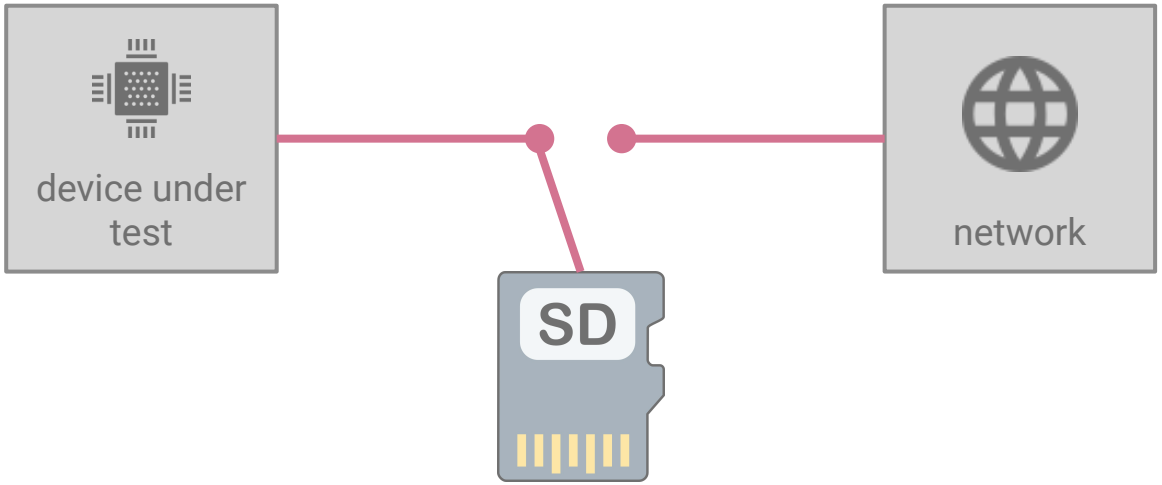


HPI Hasso Plattner Institut
Digital Engineering · Universität Potsdam

NetSD - Networked SD Card

Remote Access to Integrated SD Cards of Embedded Devices

1st International Workshop on Testing Distributed Internet of Things Systems, Oct 4th 2021

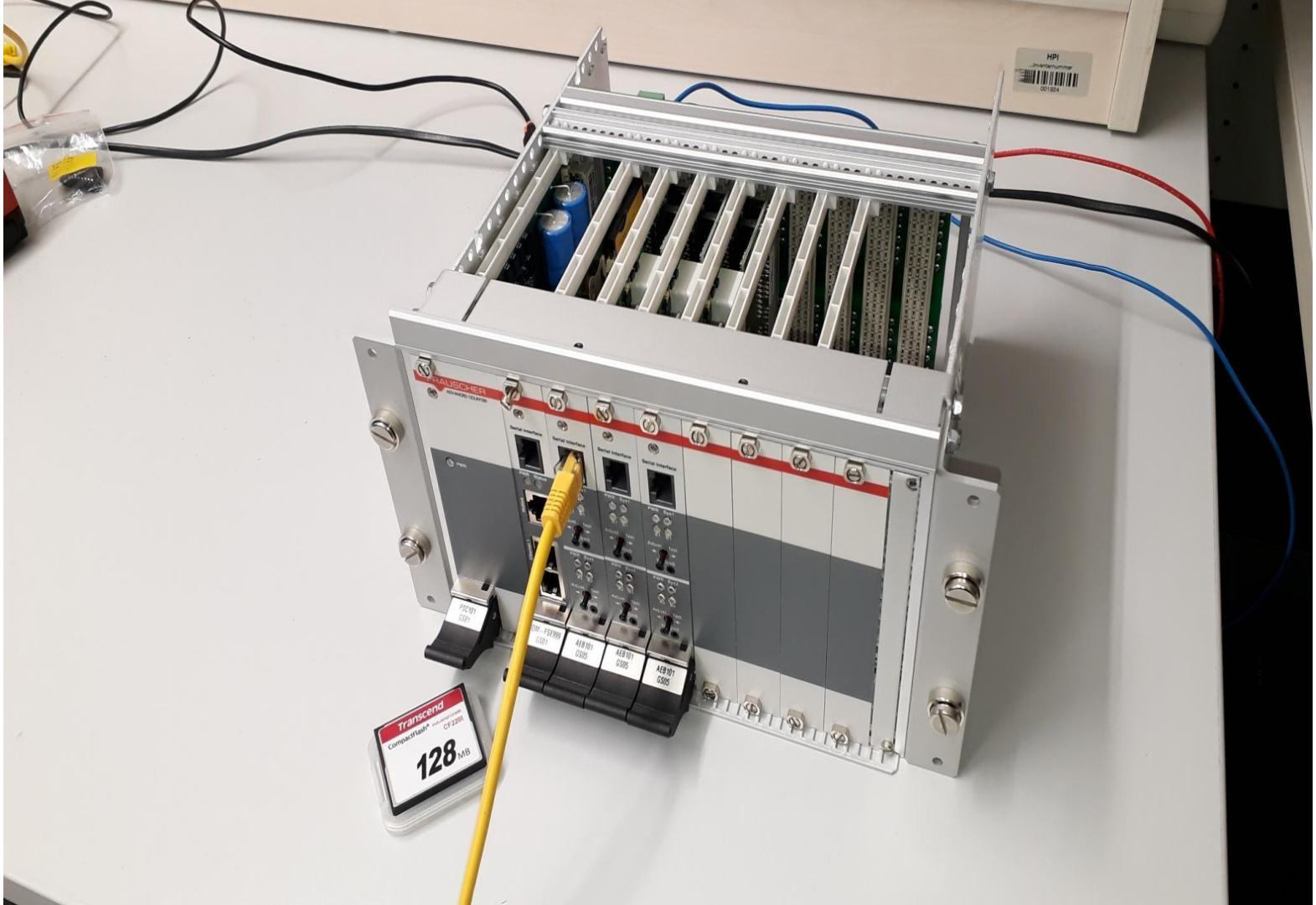
Valentin Schröter, Arne Bookmeyer, Lukas Pirl
Professorship for Operating Systems and Middleware (Prof. Andreas Polze)
Hasso Plattner Institute, University of Potsdam

Research Question

How can the configuration via storage cards of embedded devices be automated?

Or: How would you automate a “power off, flash, power on, test, repeat” cycle for a single-board computer without PXE, UART, etc.?

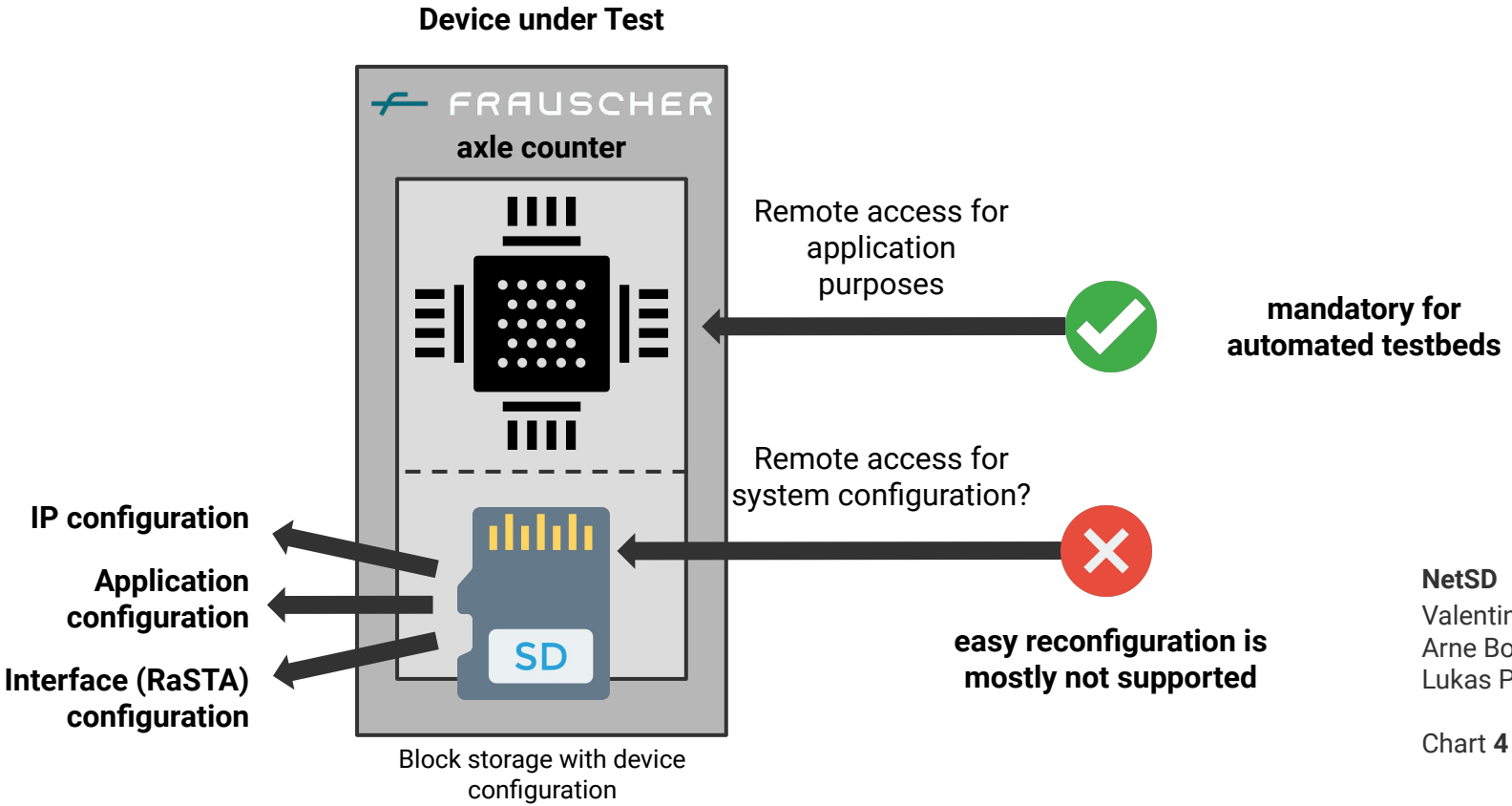
- especially for hybrid testbeds
  - e.g., proprietary hardware in the HPI IoT lab
- remove need for physical access
  - e.g., “lights out lab”, online teaching, etc.



HPI Hasso Plattner Institut

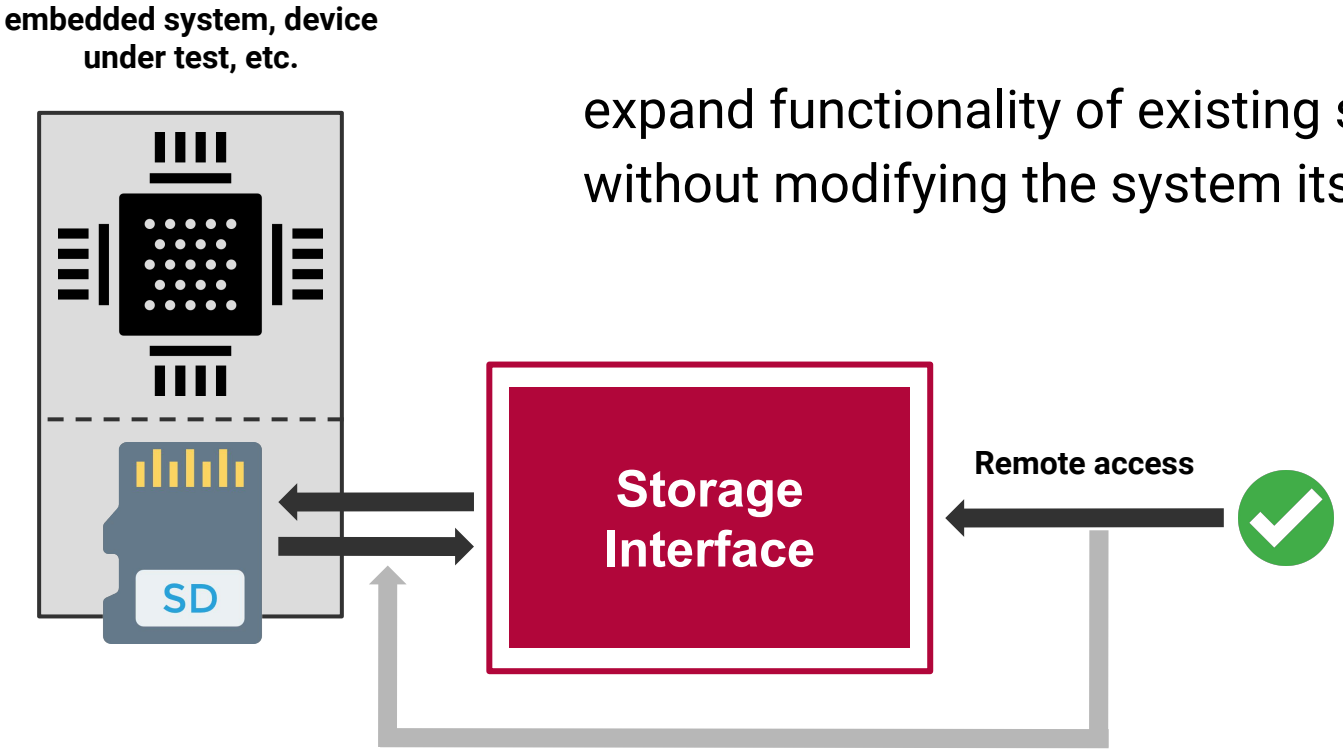
IoT Lab

Use case for networked SD cards: **Hybrid testbeds**



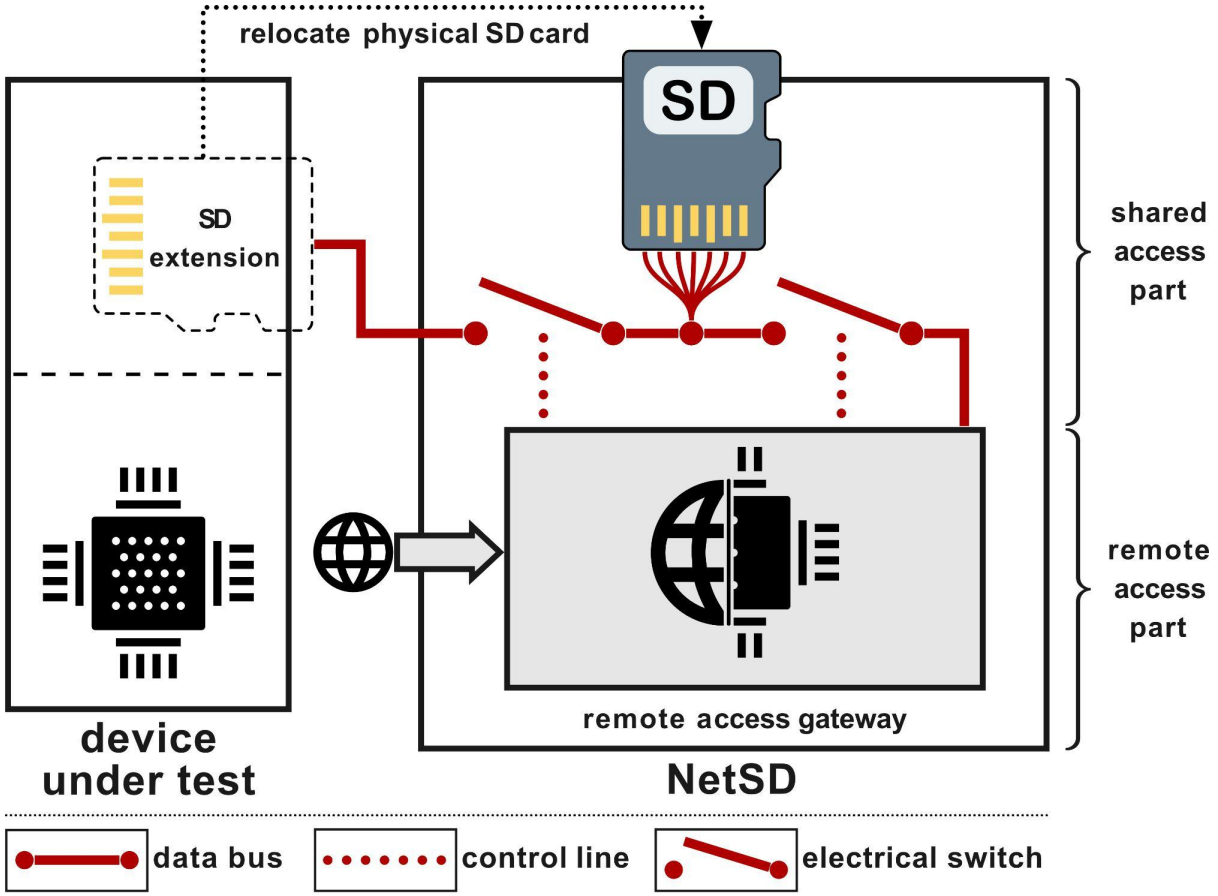
NetSD
Valentin Schröter
Arne Boockmeyer
Lukas Pirl

Objective of NetSD



expand functionality of existing system without modifying the system itself

System Overview



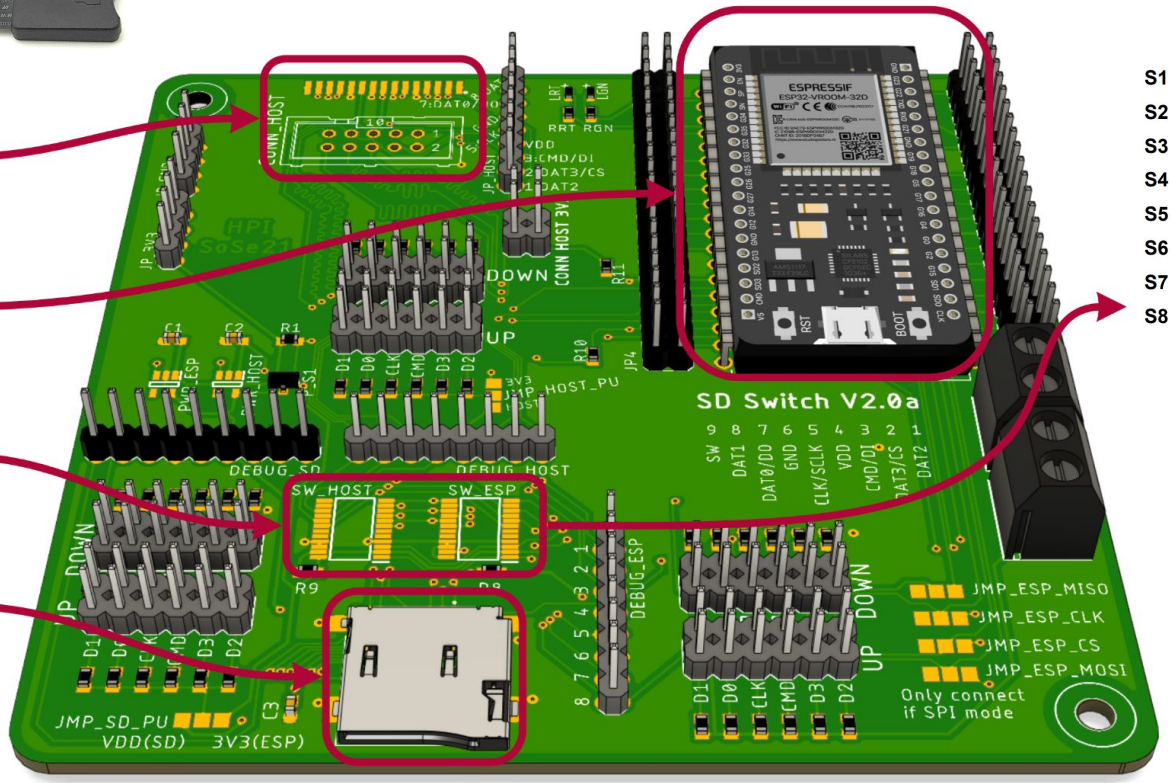
System Overview: **Evaluation Board**

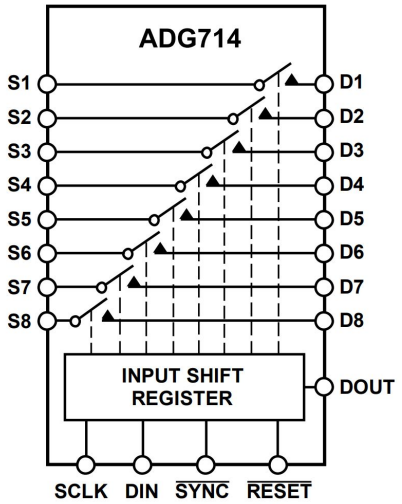
Connection for testbed device

ESP32

Bus Switches

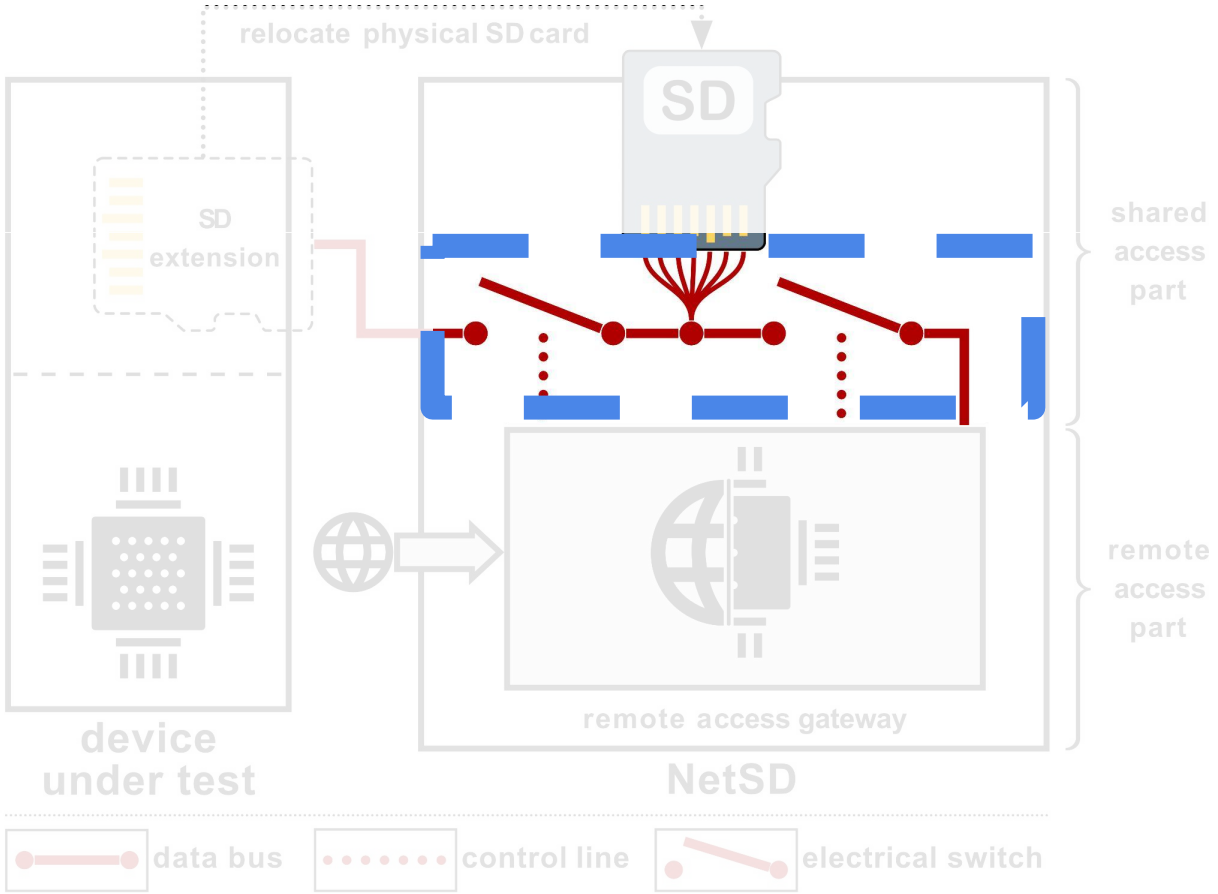
SD Card Slot





NetSD
Valentin Schröter
Arne Boockmeyer
Lukas Pirl

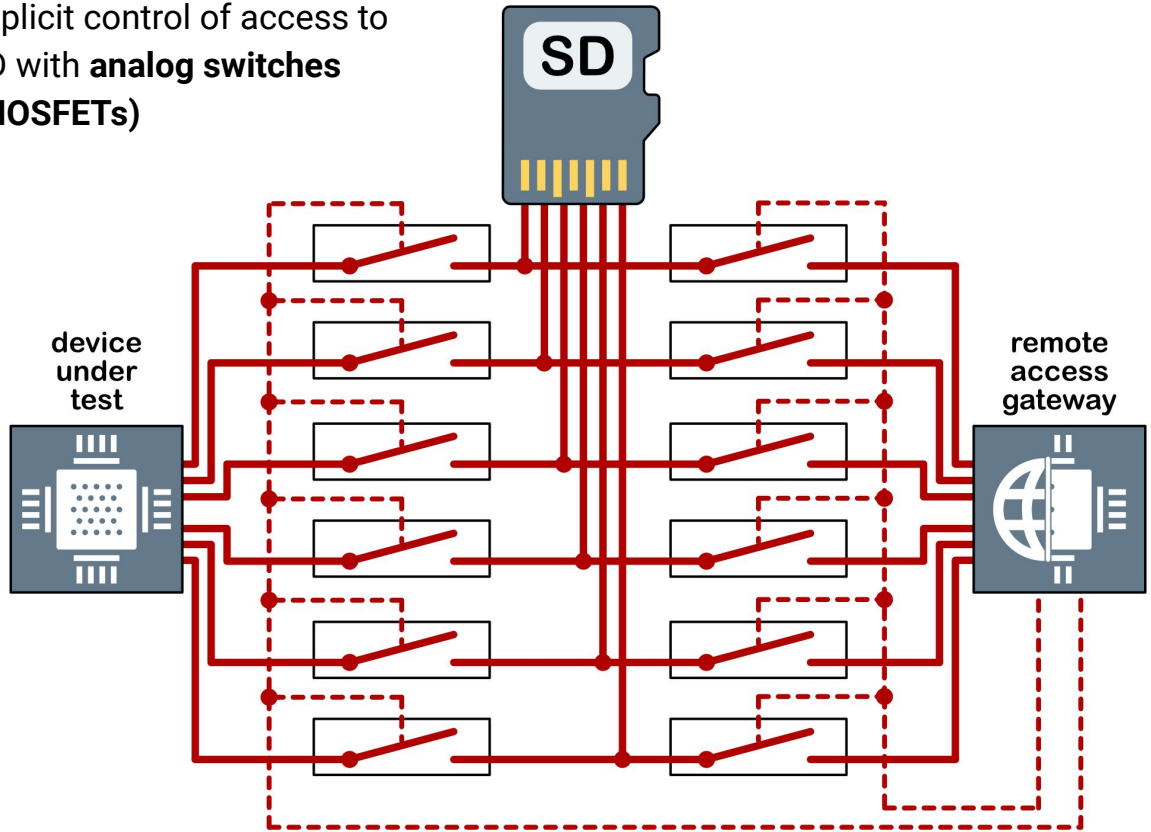
System Overview

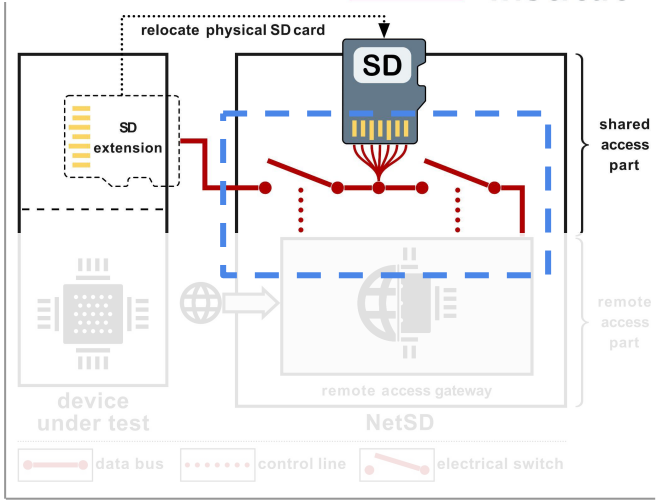


NetSD
Valentin Schröter
Arne Boockmeyer
Lukas Pirl

Shared Access to SD Card

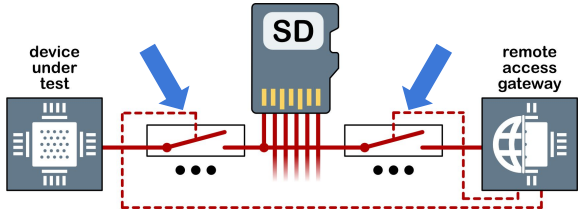
Explicit control of access to SD with **analog switches (MOSFETs)**

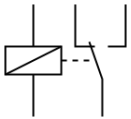
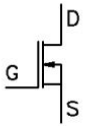


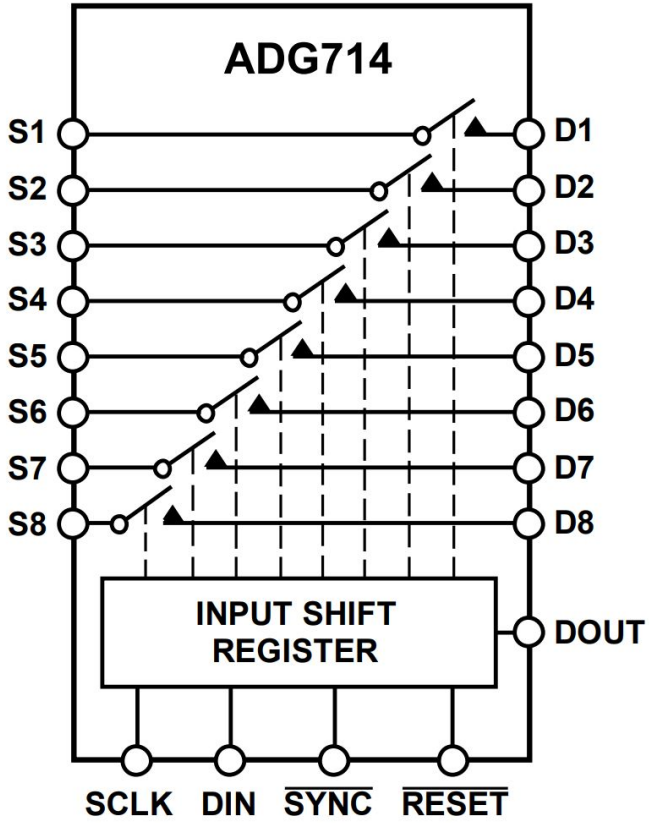


NetSD
Valentin Schröter
Arne Boockmeyer
Lukas Pirl

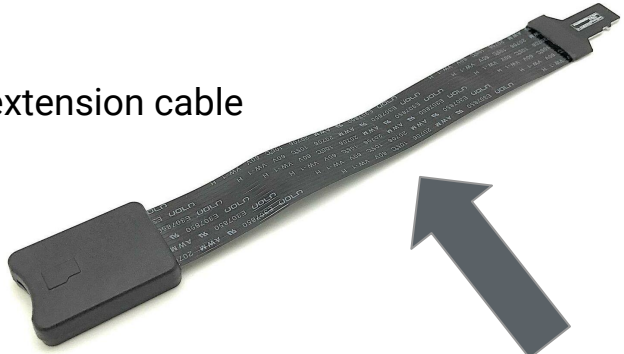
Switch Technology



	**Relays**	→ ease of design → certainty of electr. separation	→ slow control speed → energy consumption → big component size
	**Transistor**	→ fast control speed → small component size	→ **decrease of signal voltage** → complex electrical network for bidirectional switch
	**MOSFET**	→ fast control speed → **no voltage changes** → small component size	→ complex electrical network for bidirectional switch



Physical Relocation of SD Card

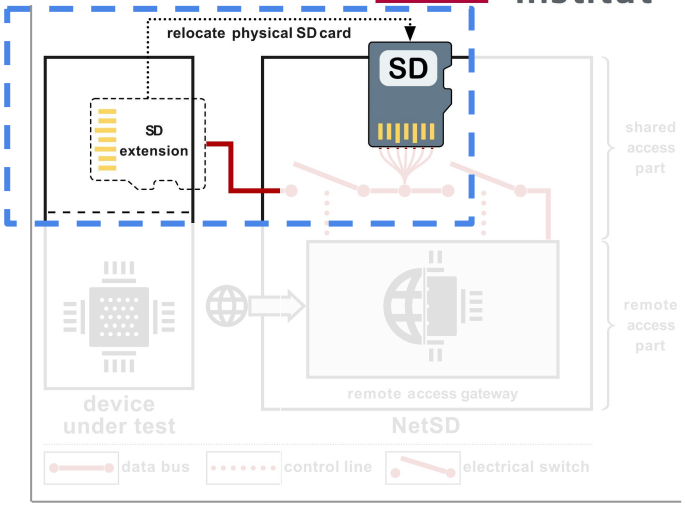


normal SD signal lines

adapted signal lines for improved transmission over extension cable

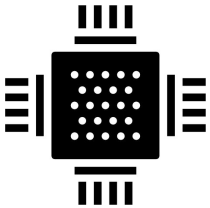
→ adjacent signal lines can influence each other (crosstalk)

→ adjacent signal lines do not influence each other (**relevant for higher frequencies**)



NetSD
Valentin Schröter
Arne Boockmeyer
Lukas Pirl

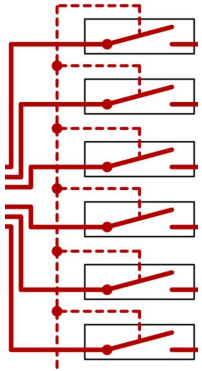
Hardware Evaluation

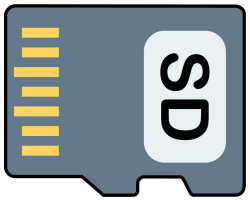


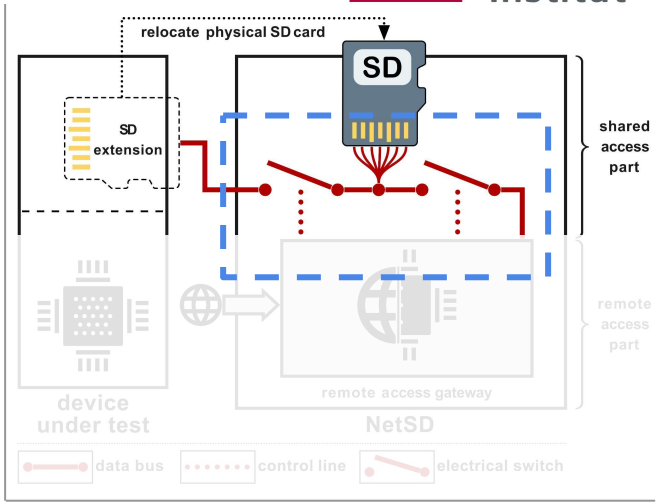
device under test

extension cable

analog switches



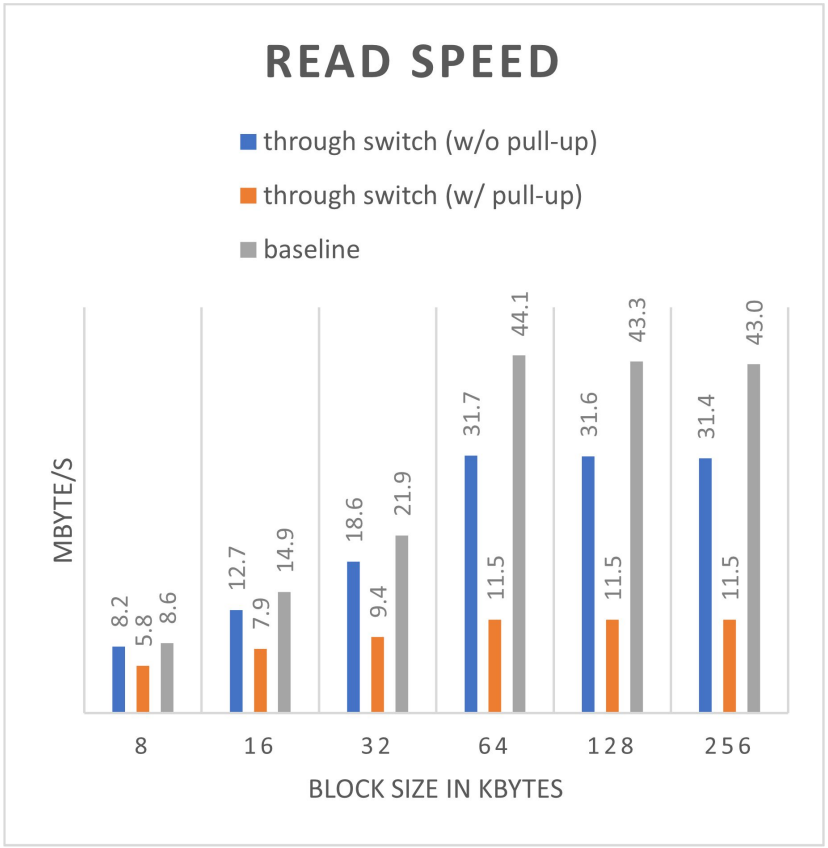


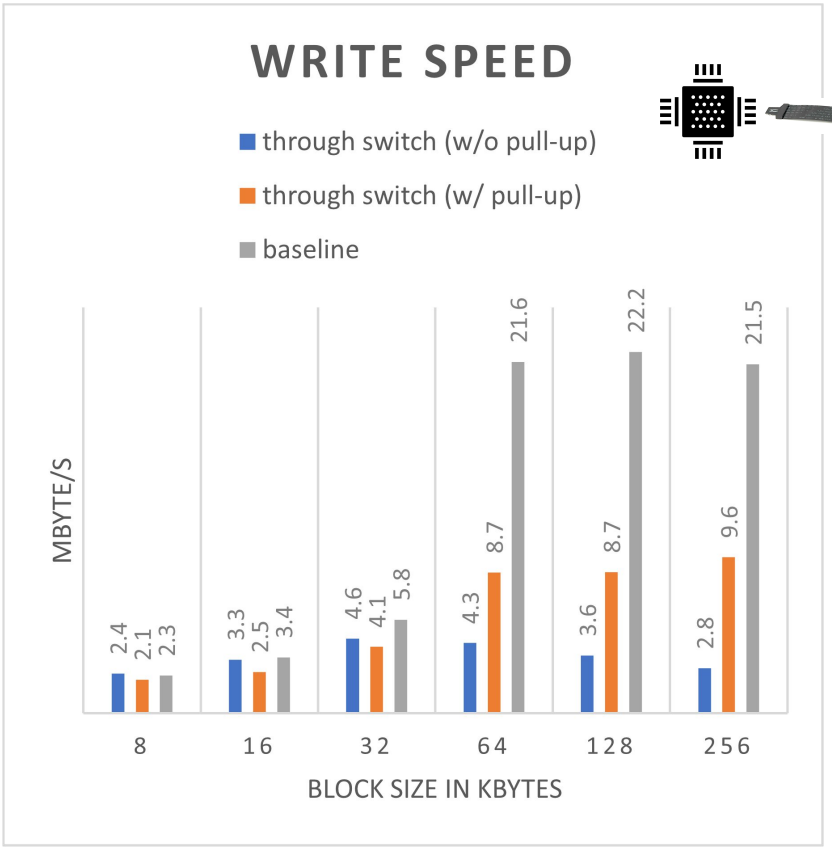


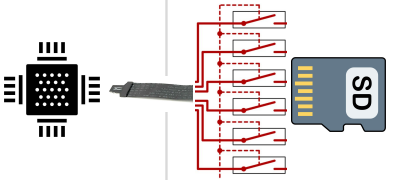
signal degradation?

NetSD
Valentin Schröter
Arne Boockmeyer
Lukas Pirl

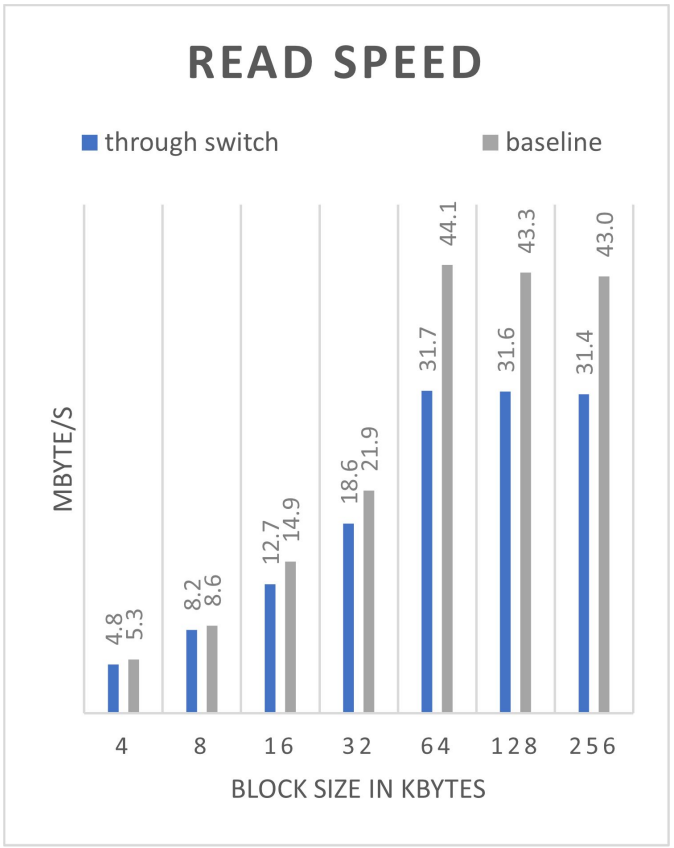
Access Throughput Evaluation

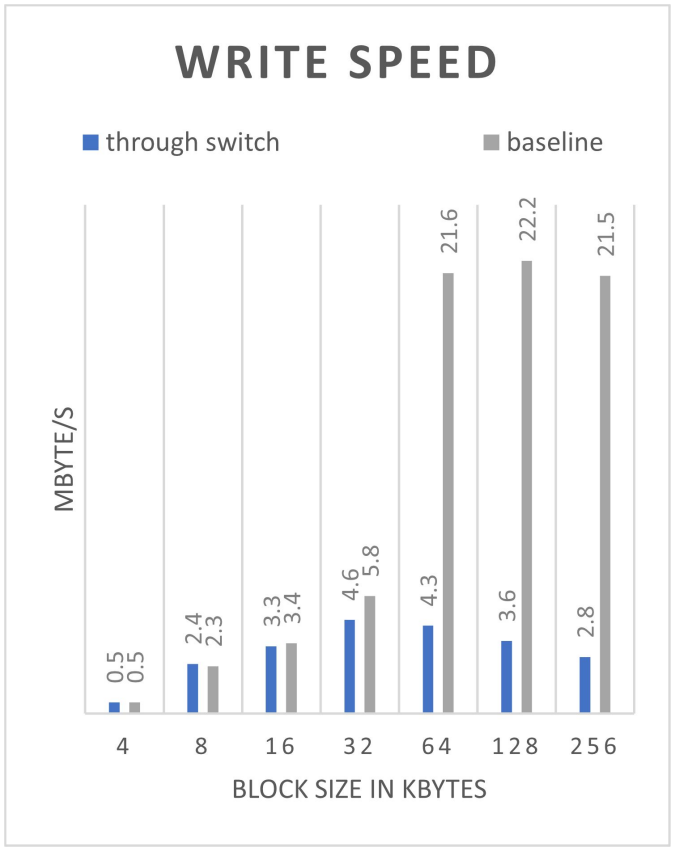


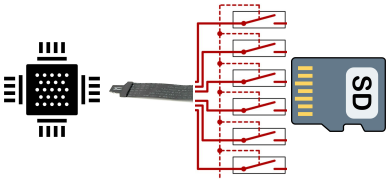




System Overview - data speed comparison



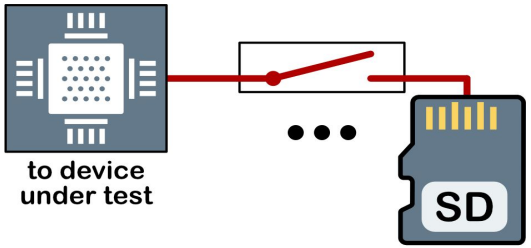




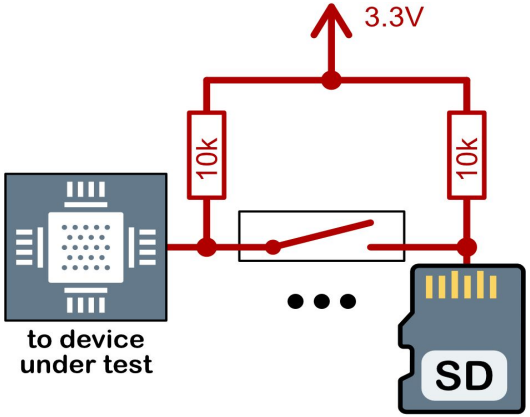
NetSD

Valentin Schröter
Arne Boockmeyer
Lukas Pirl

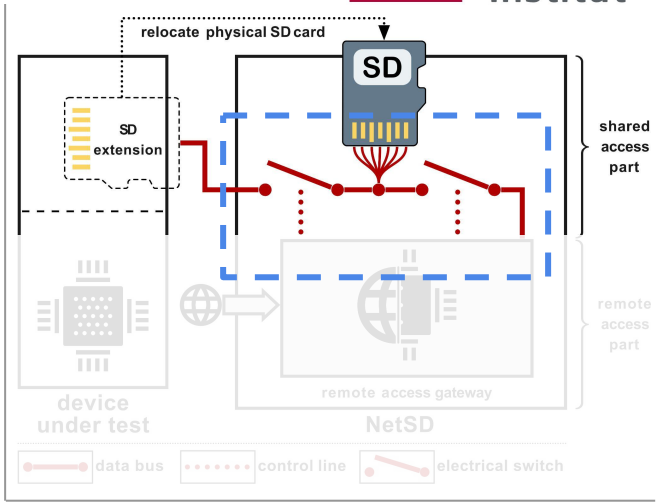
System Overview - data speed comparison



With implicit
pull-up resistors
from host device

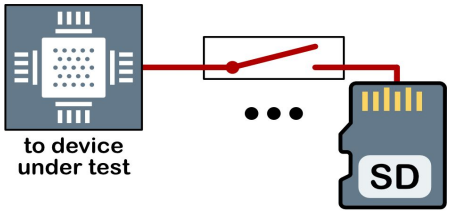


With explicit
pull-up resistors
on NetSD



NetSD
Valentin Schröter
Arne Boockmeyer
Lukas Pirl

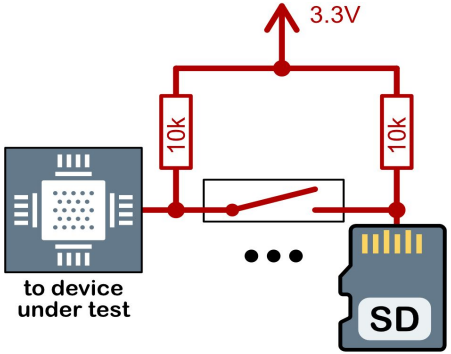
System Overview - Hardware Quality



With implicit
pull-up resistors
from host device

no explicit
pull-up resistors
→ 1.8 Volt

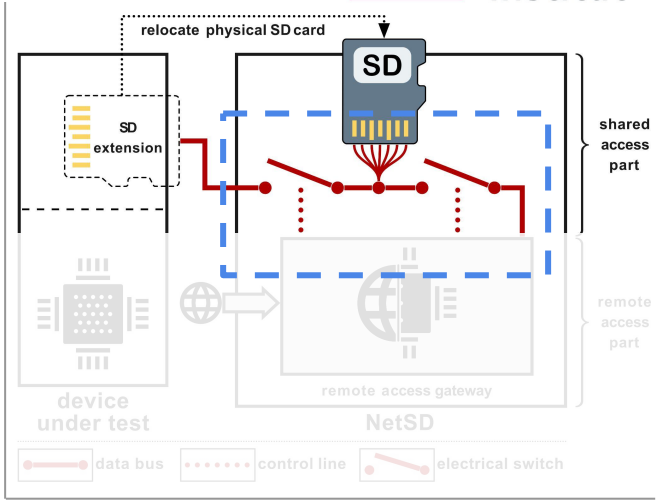
for UHS mode and
fast read speed



With explicit
pull-up resistors
on NetSD

explicit
pull-up resistors
→ 3.3 Volt

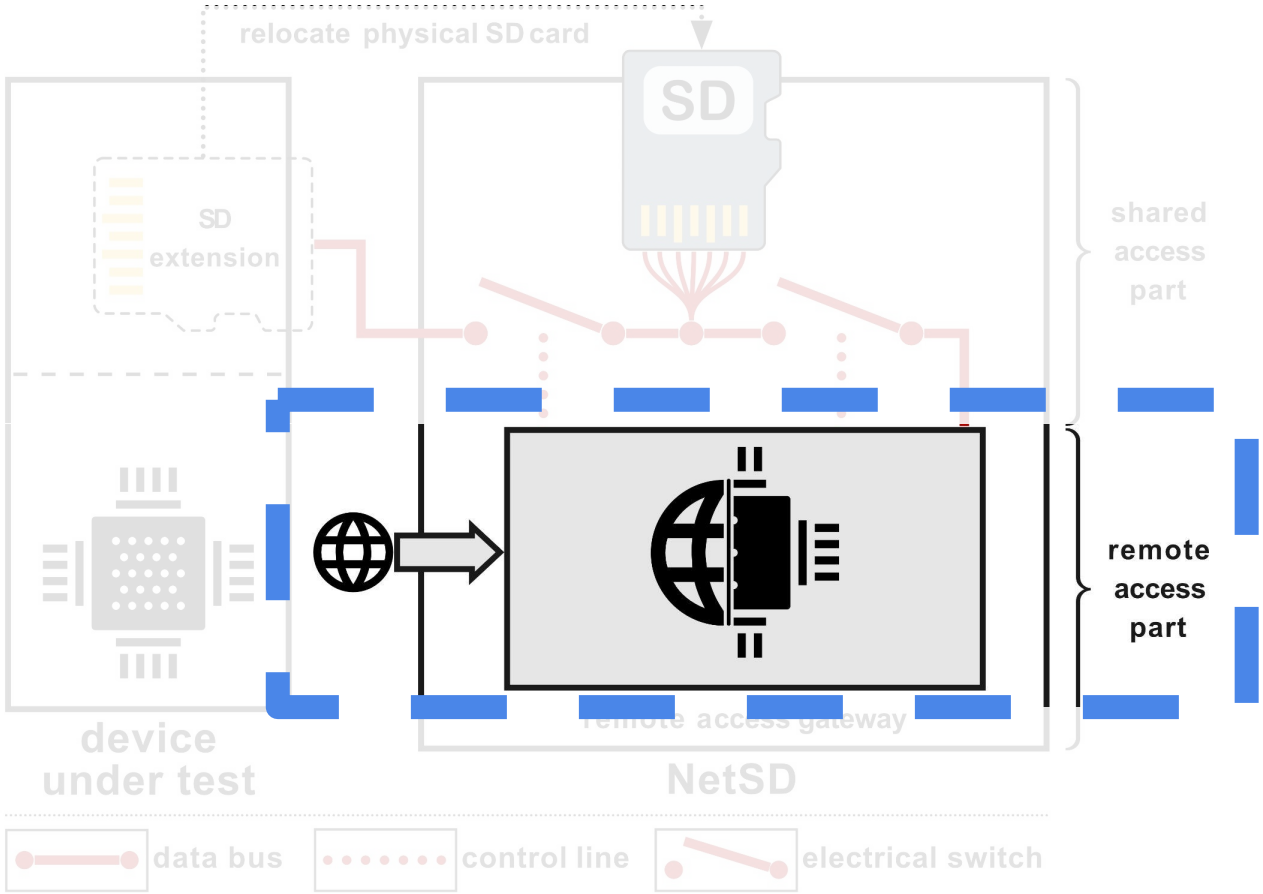
for reliable write
transmissions



NetSD
Valentin Schröter
Arne Boockmeyer
Lukas Pirl

Future Work: allowing both modes simultaneously with host sensitive adaption

Remote Access

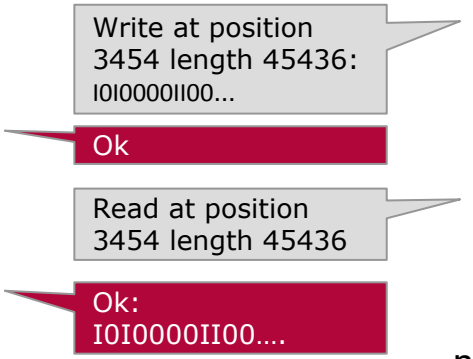


Remote Access

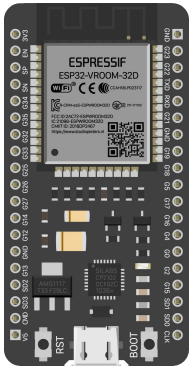
Block-level: Network Block Device (NBD)

NBD Protocol (Data Phase):

- card exposed as block device on Linux client
- server waiting for commands:

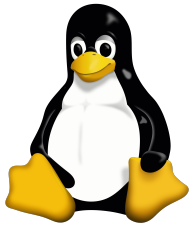


TCP Server
Port 10809

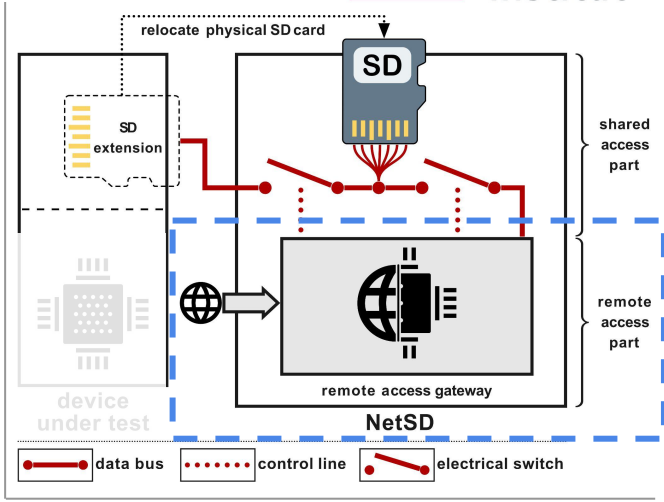


ESP32

TCP Client



kernel module +
ndb-client command line tool

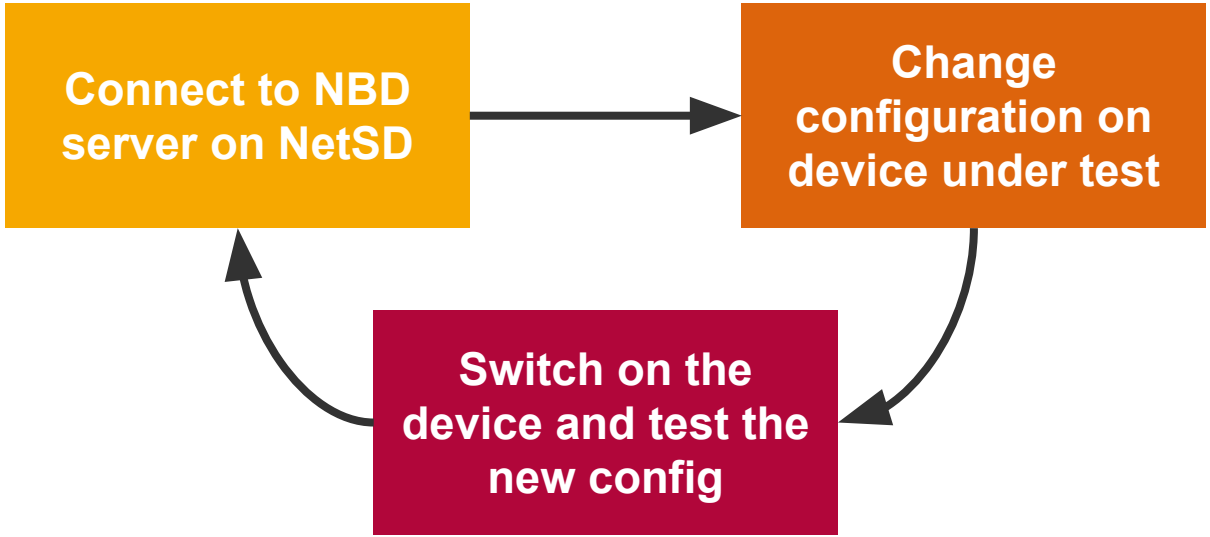


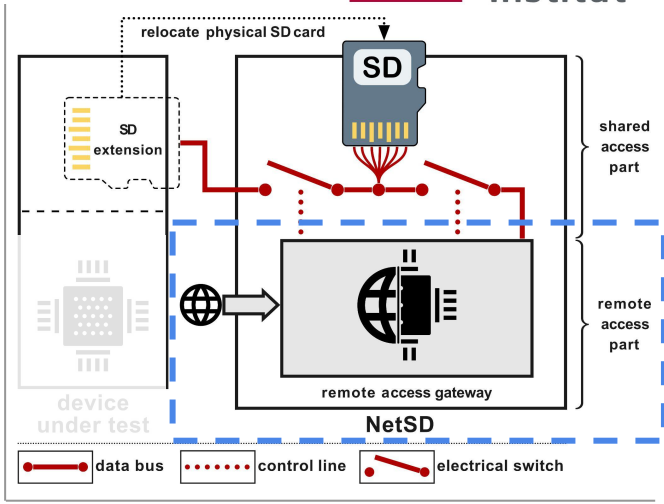
File-level: Web server

NetSD
Valentin Schröter
Arne Boockmeyer
Lukas Pirl

Example Process

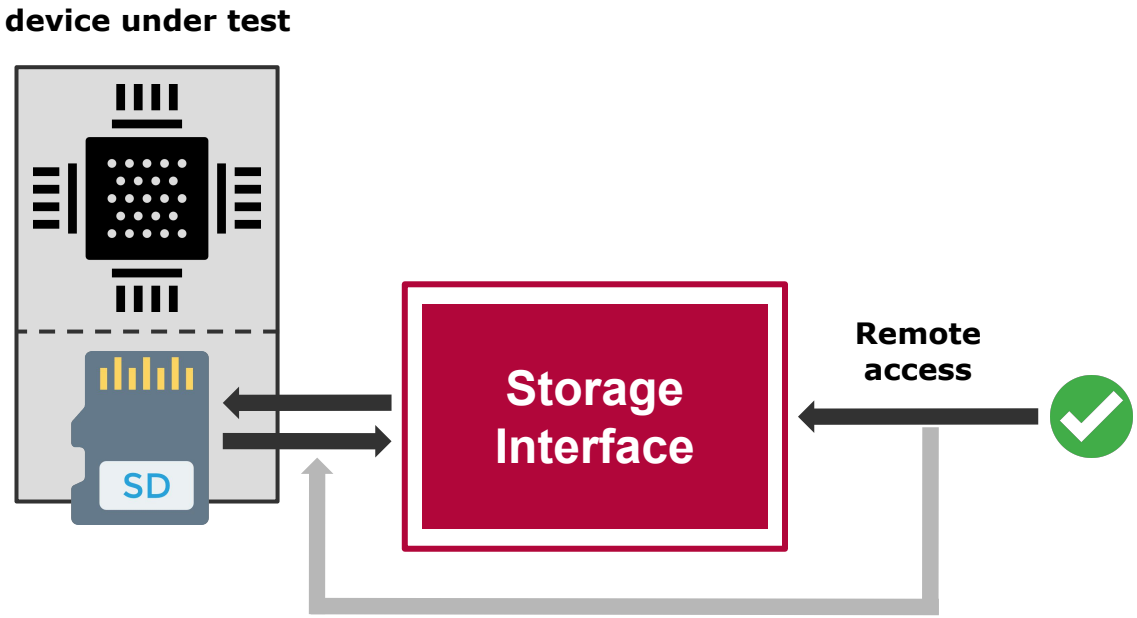
Block-level remote access with **NBD**

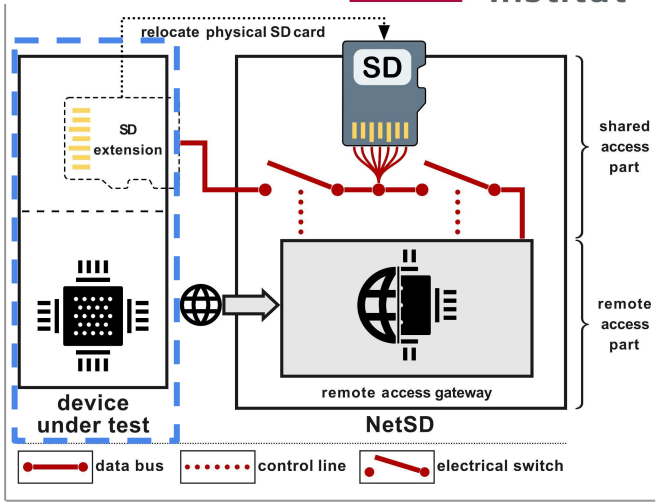




change configuration without physically touching the block device

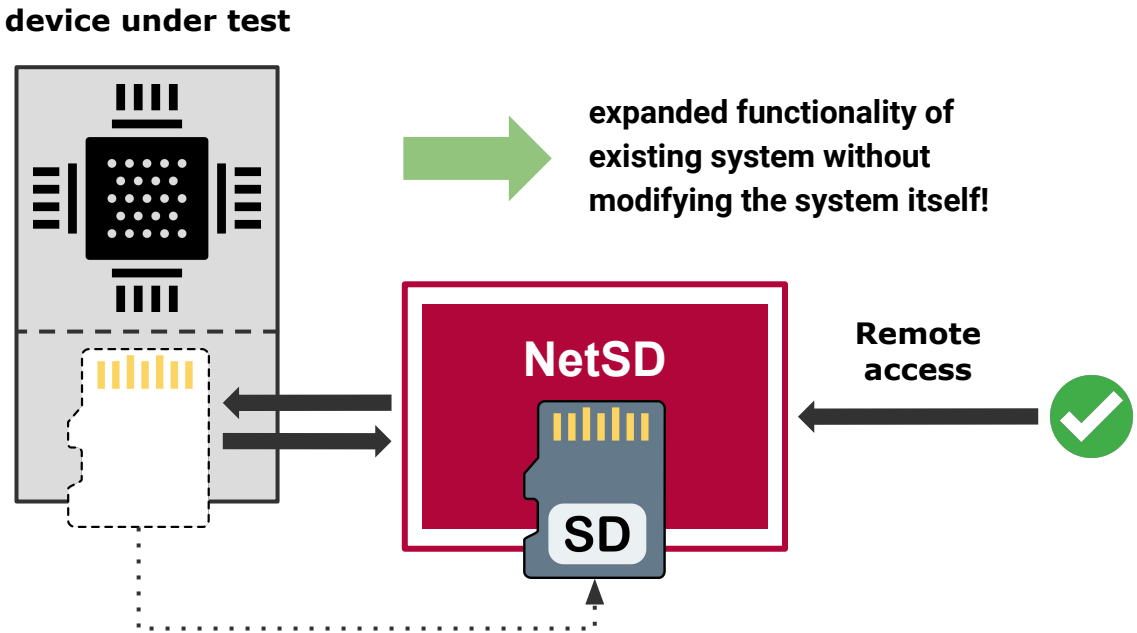
System Overview: **device under test**

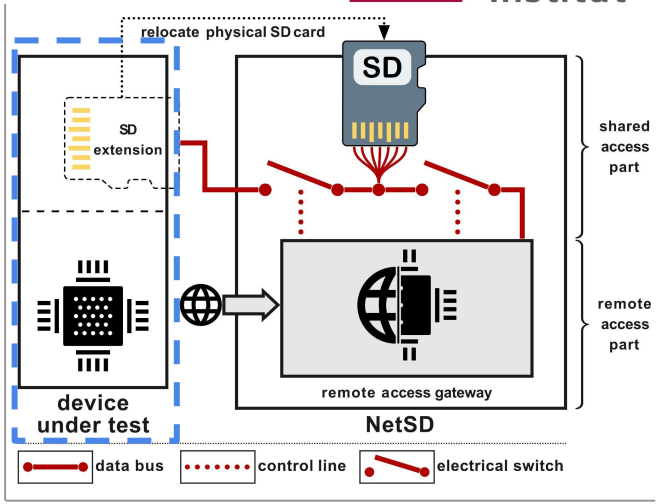




NetSD
Valentin Schröter
Arne Boockmeyer
Lukas Pirl

System Overview: **device under test**





limitations: fails on parallel access, no concurrent access

NetSD
Valentin Schröter
Arne Boockmeyer
Lukas Pirl

Summary: **Capabilities and Future Work**

Hasso Plattner Institut

General purpose remote access to block storage

Fault Injection

Storage multiplexing for embedded systems

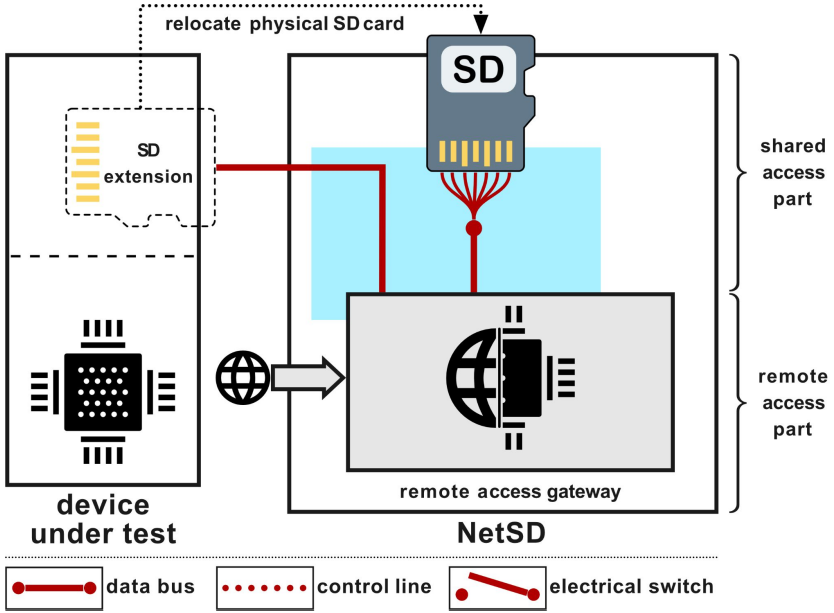
Concurrent access to an SD card from multiple devices

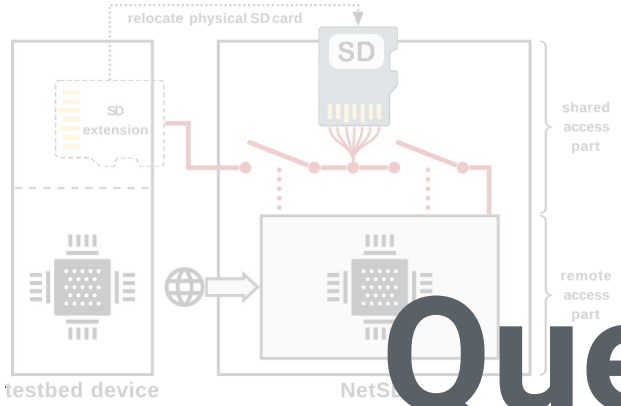
(future work)

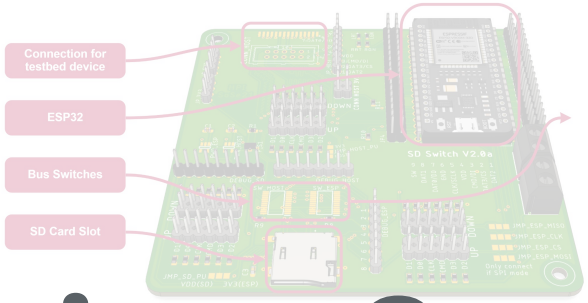
Future Work - Implementation Strategy

Concurrent access to an SD card from multiple devices

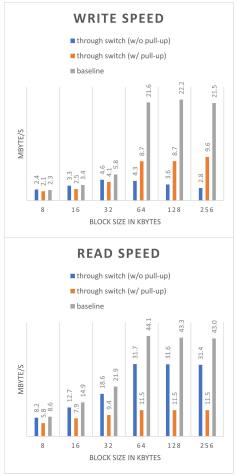
Exposing proper clock speed info for our device







Questions?



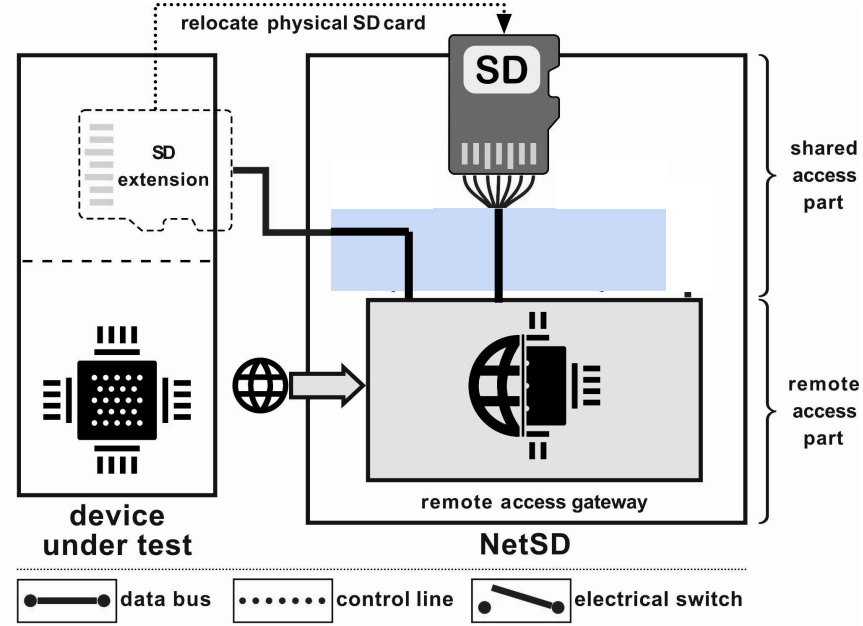
HPI Hasso Plattner Institut
Digital Engineering · Universität Potsdam

NetSD - Remote Access to Integrated SD Cards of Embedded Devices
1st International Workshop on Testing Distributed Internet of Things Systems, Oct 4th 2021
Valentin Schröter, Arne Bookmeyer, Lukas Pirl
Professorship for Operating Systems and Middleware (Prof. Andreas Polze)
Hasso Plattner Institute, University of Potsdam

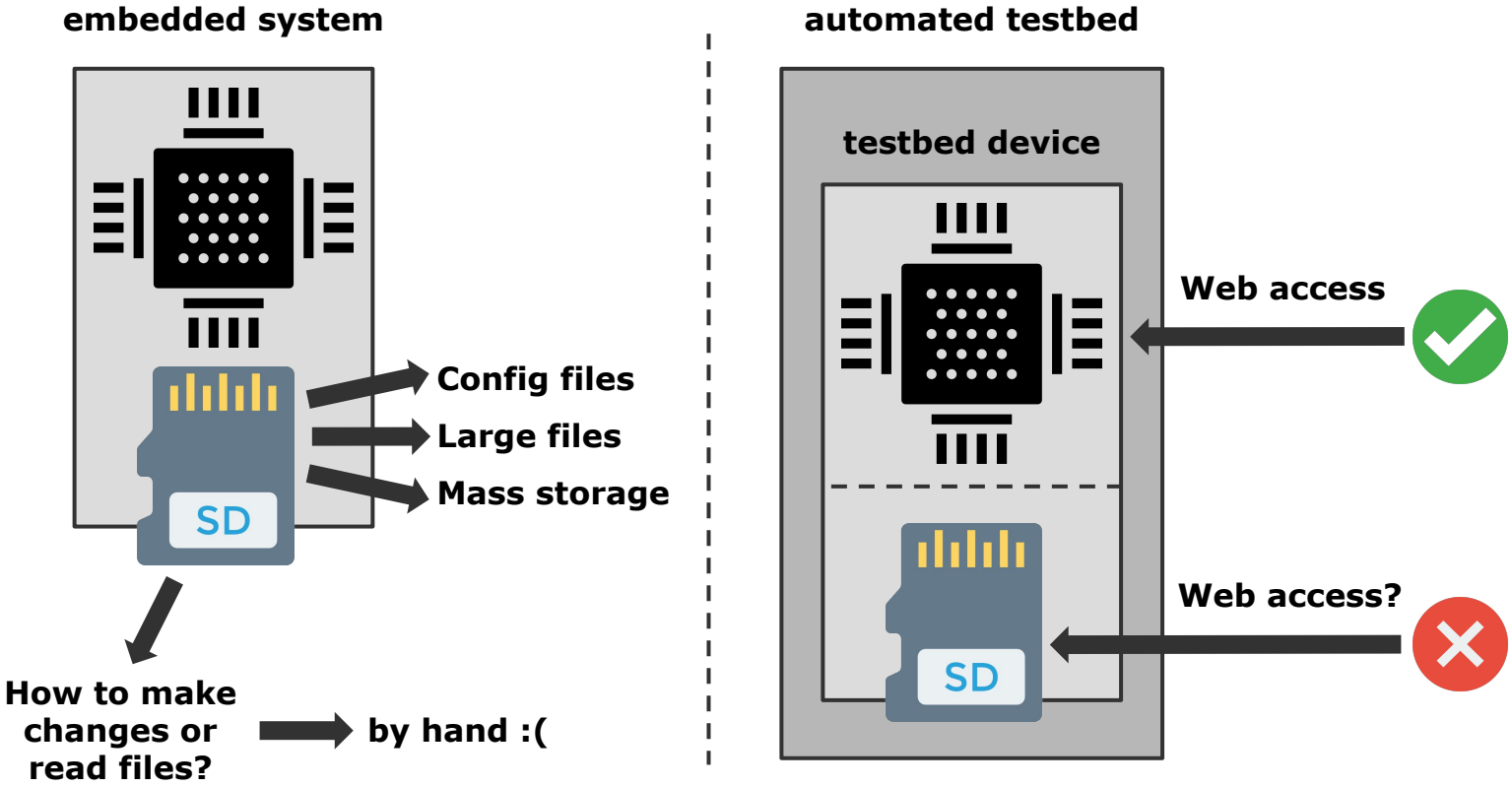
Future Work - Implementation Strategy

Concurrent access to an SD card from multiple devices

Exposing proper clock speed info for our deivce

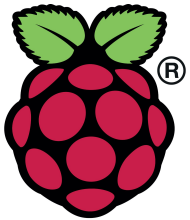


Motivation for a networked SD Card

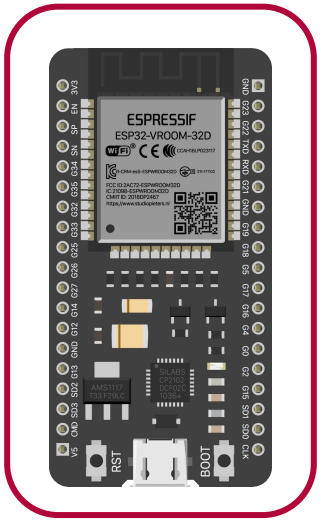


Hardware: the **Remote** part

device with Ethernet/WiFi capabilities



Raspberry Pi



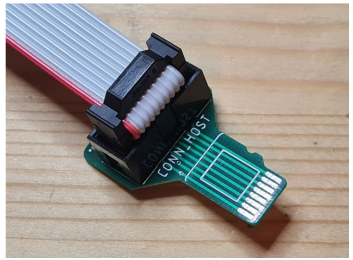
ESP32

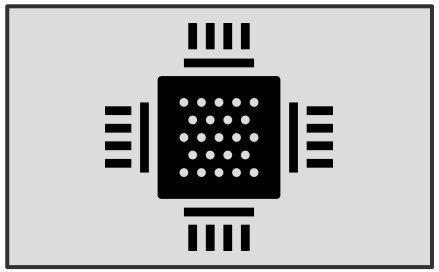
→ **simple WiFi connectivity**

→ **sufficient performance**

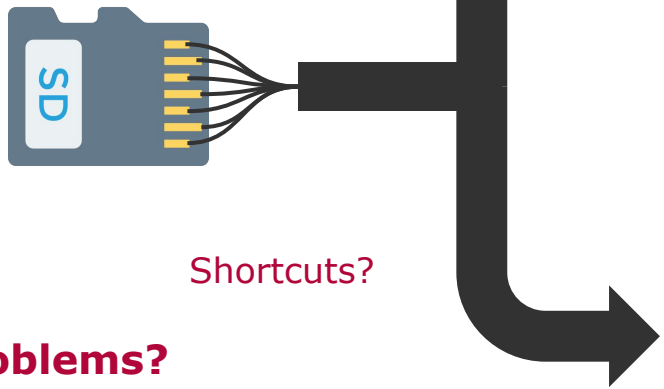
Hardware: the **Shared Access** part

First idea:
hardwired electrical connection of both devices with SD





testbed device

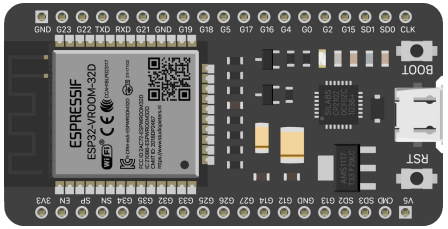


Shortcuts?

Problems?

Read/Write conflicts?

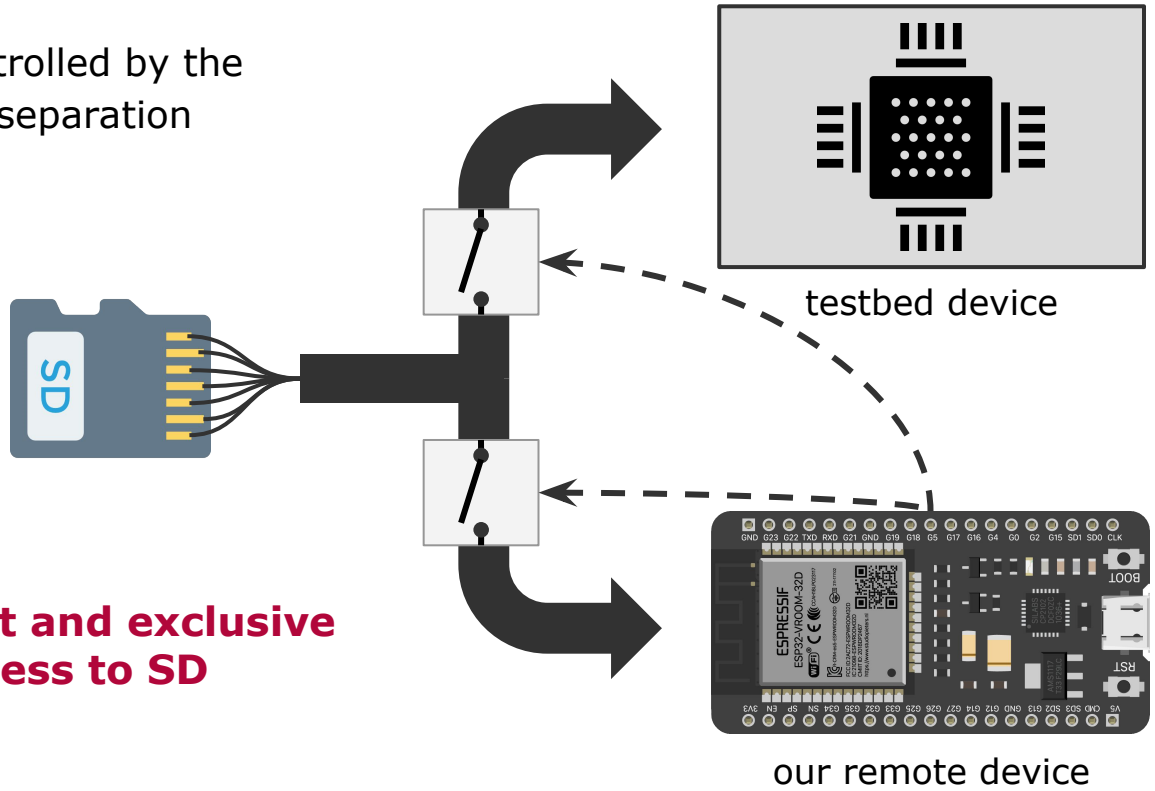
Different voltage levels?



our remote device

Hardware: **SD Switch**

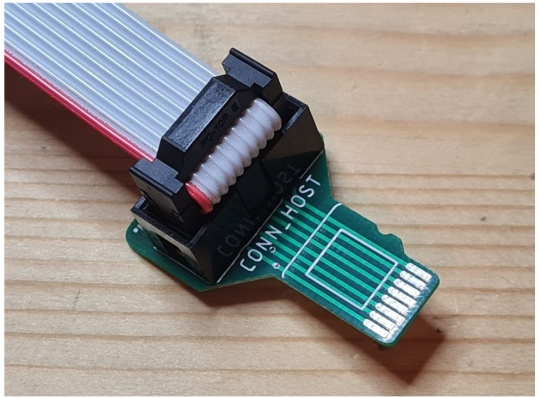
Better:
Bus Switches controlled by the ESP for electrical separation



→ explicit and exclusive access to SD

Hardware Problems: **Extension Cable**

Our Cable

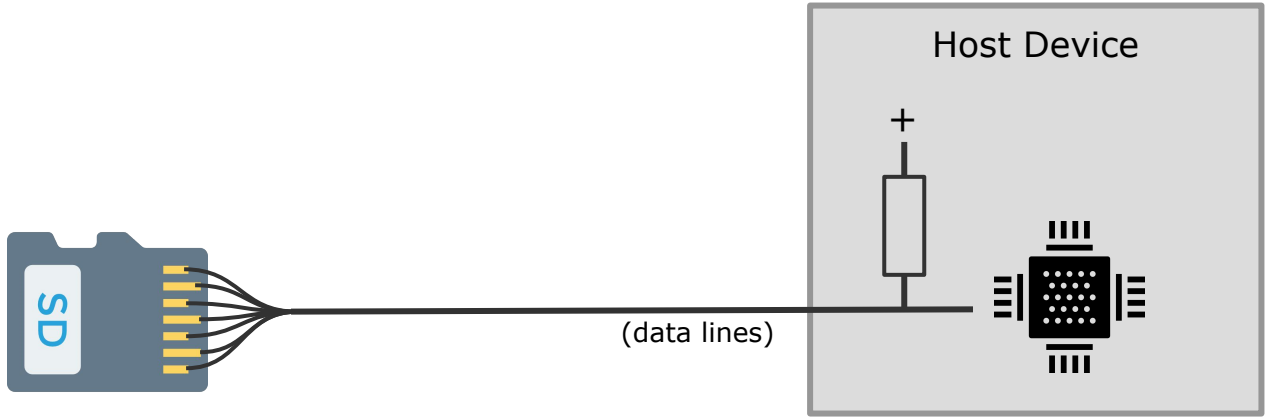


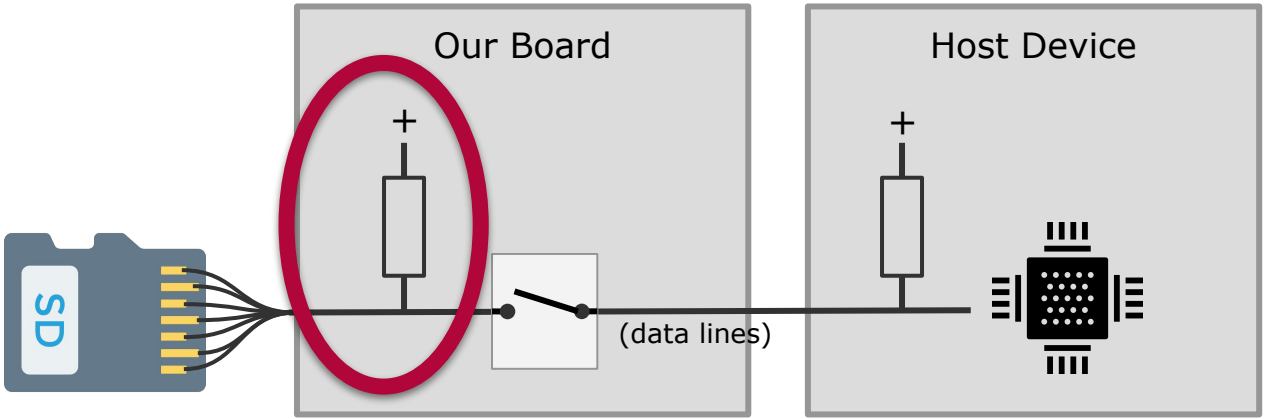
DAT2
DAT3
CMD
VDD
CLK
GND
DAT0
DAT1

Commercial Cable

DAT2
GND
DAT3
GND
CMD
GND
VDD
GND
CLK
GND
DAT0
GND
DAT1

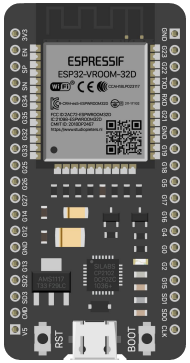
Hardware Problems: **SD Pull-ups through the Switch**





Software: **Remote Access**

First Idea:
REST API to read and write files over **HTTP**
file system level access using a library on the ESP

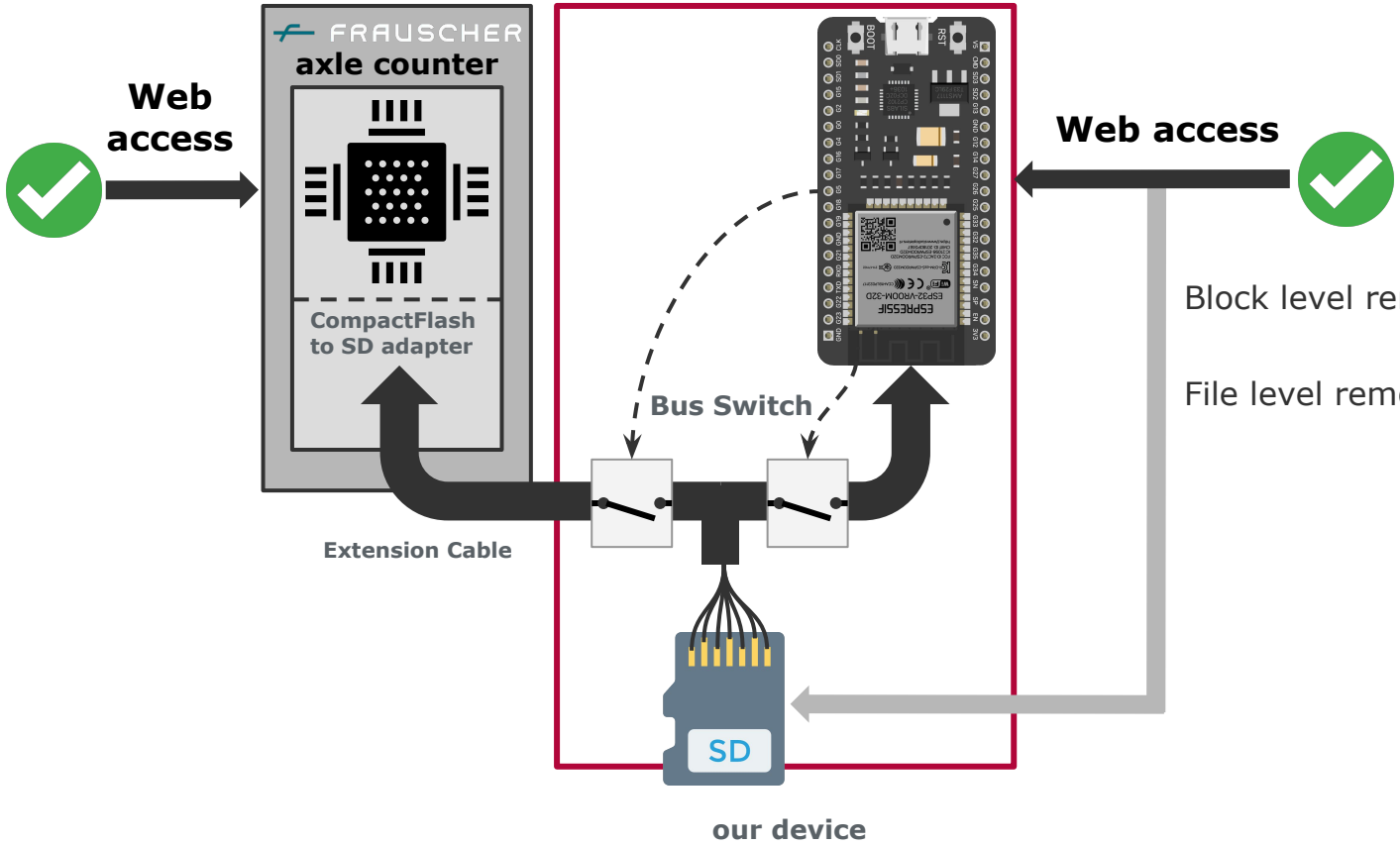


HTTP Client

Limitations

- different file systems
- writing complete disk images
- partitions
- intentionally breaking the file system for testing purposes

Time for **questions**



Block level remote access with **NBD**

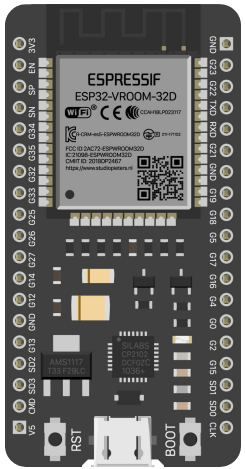
File level remote access with **REST-API**

Networked SD Card

Valentin Schröter,
Tobias Zagorni

Software: Block level remote access with **NBD**

TCP Server
Port 10809



ESP32

NBD Protocol (Data Phase):

- card exposed as block device on Linux client
- server waiting for commands:

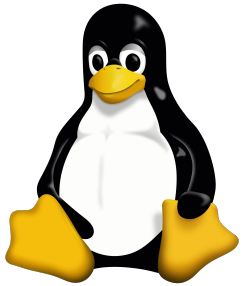
Write at position 3454
length 45436:
I0I0000II00…

Ok

Read at position 3454
length 45436

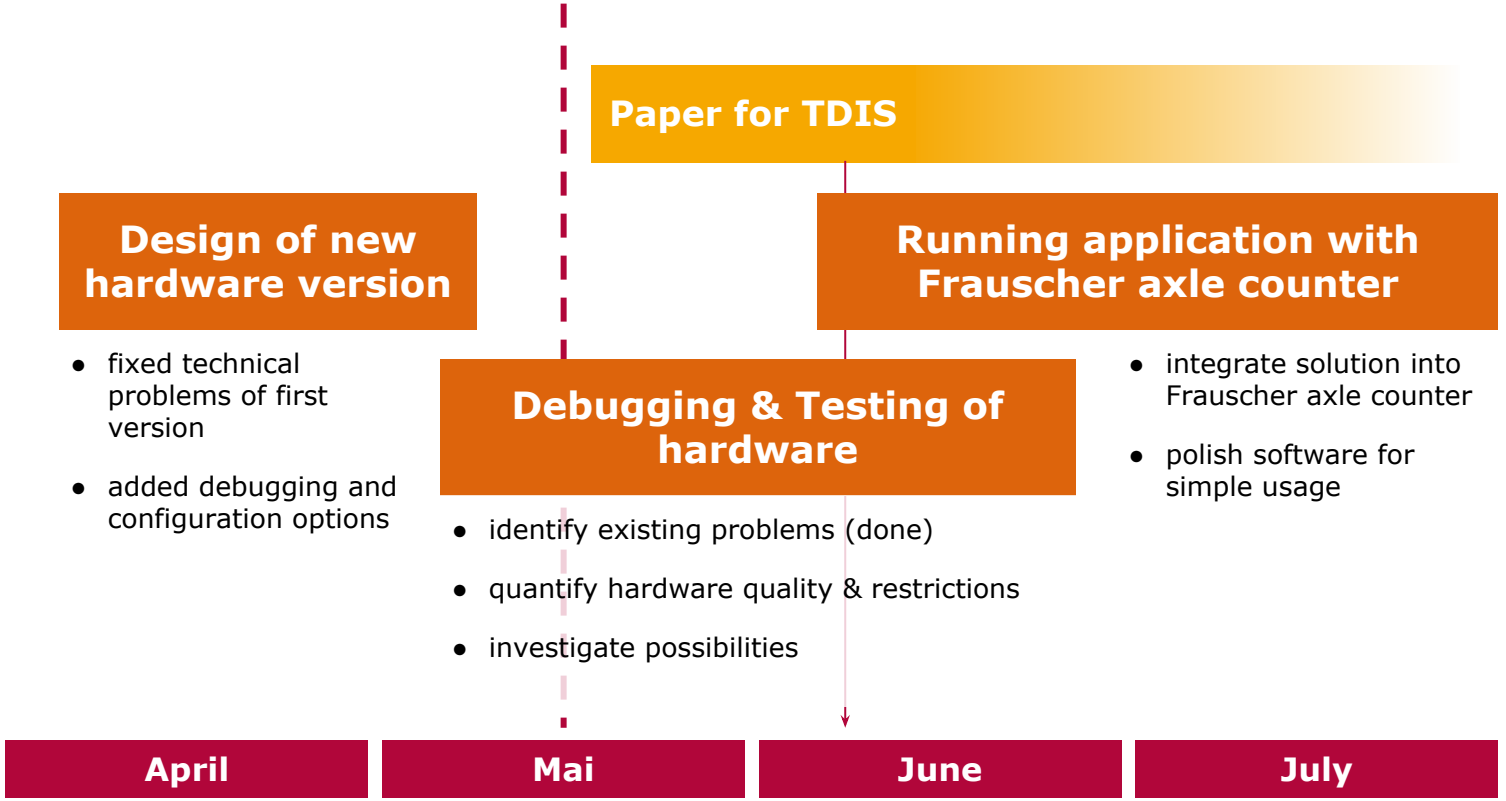
Ok:
IOI0000II00….

TCP Client



kernel module +
ndb-client command line tool

Project timeline



Paper outline

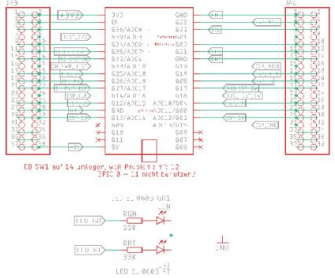
ca. 6 pages

- **Initial situation & motivation**
  - general motivation
  - use case: Frauscher
- **Technical implementation**
  - requirements & system overview
  - conceptual options
  - technical difficulties
- **Capabilities of the system**
  - general purpose usage for emb. dev.
  - “real” parallel access to block storage
  - fault injection
- **Future plans**
  - unfinished ToDos
  - possible extensions

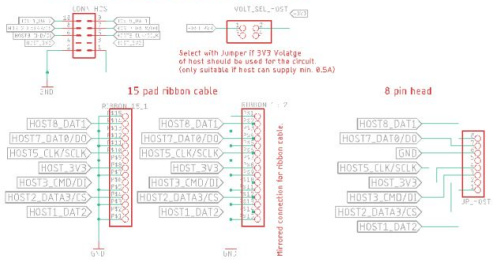
Power Connections

GP Power Pins

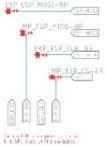
ESP



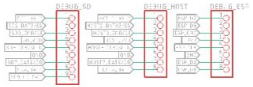
Host Device Connections



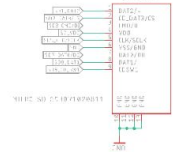
ESP SD Interface Selector (Default: native interface, connect pads for SPI Interface)



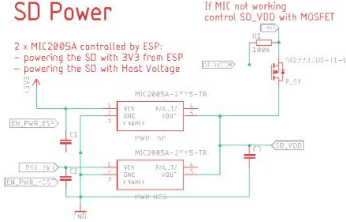
Debug Pins for Oszi



Micro SD Slot



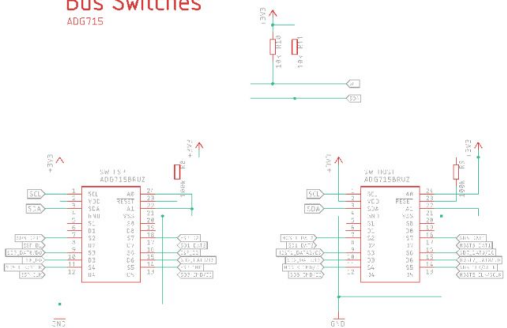
SD Power



Mounting

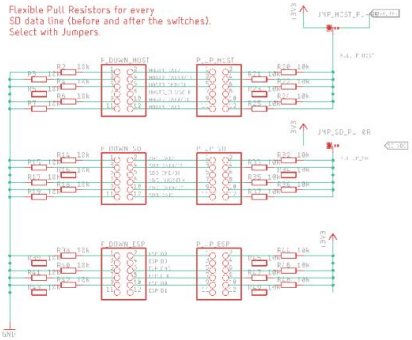
Bus Switches

ADG715

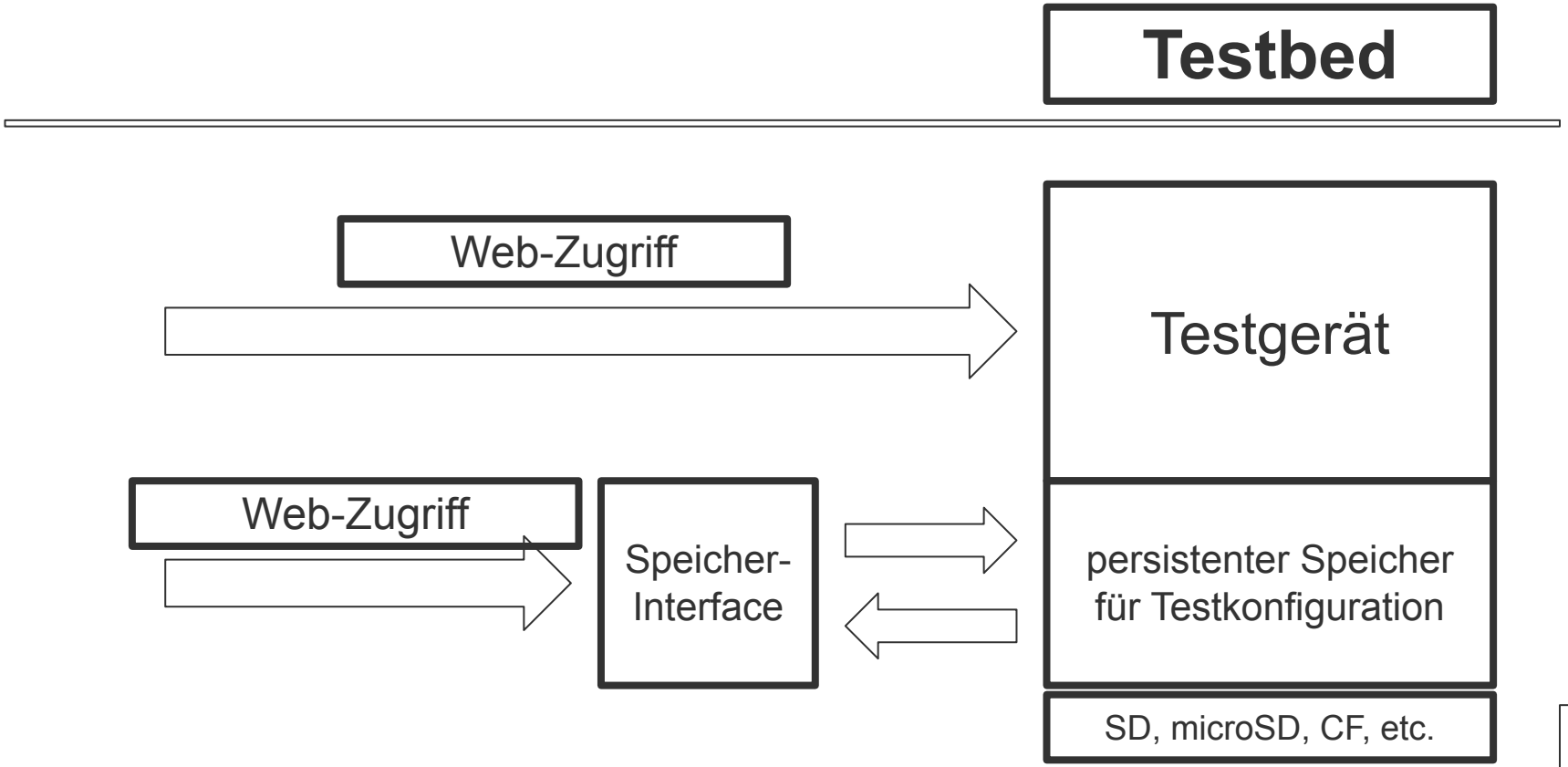


Pull Up & Pull Down

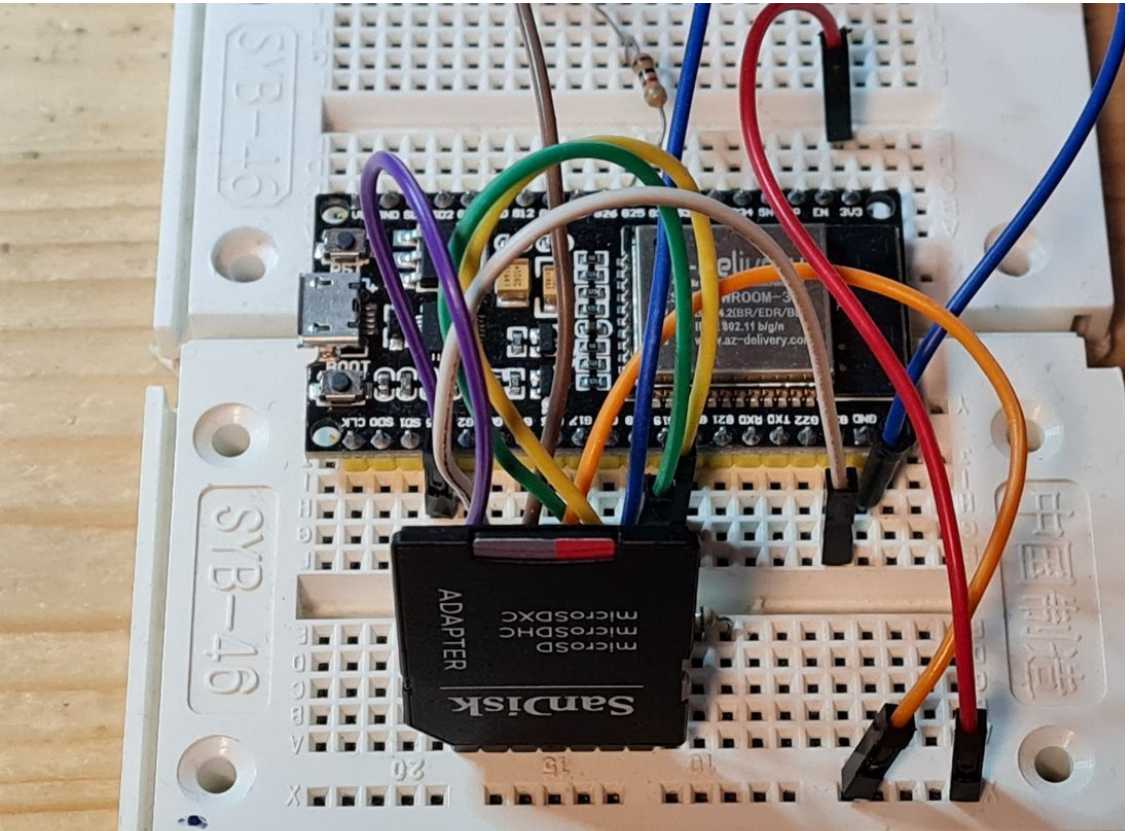
Flexible Pull Resistors for every SD data line (before and after the switches). Select with Jumpers.

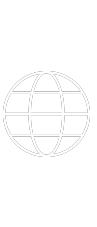


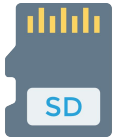
Ausgangssituation & Ziel



Was wurde bisher gemacht?
Verknüpfung ESP32 & SD-Karte & Web-API



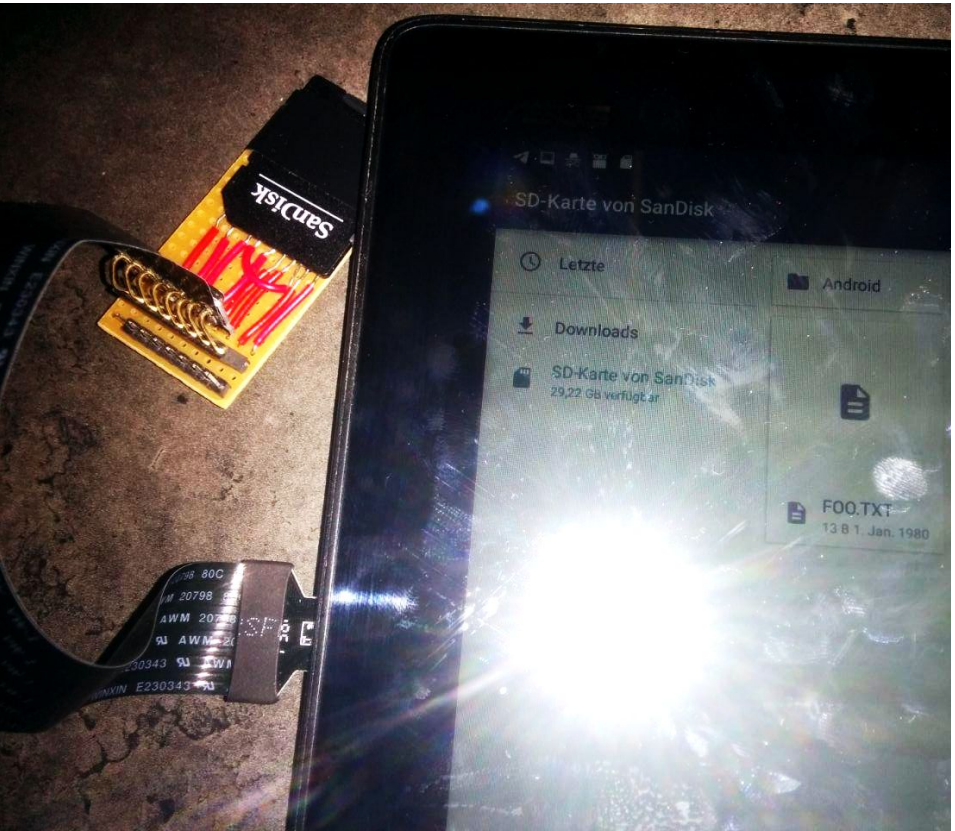




Verknüpfung SD-Karte über SPI-Interface

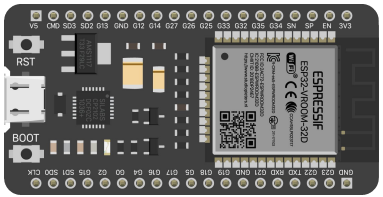
- Anzeigen, Lesen, Schreiben & Löschen von Dateien mittels des ESPs
- Funktionen über REST-API bereitgestellt

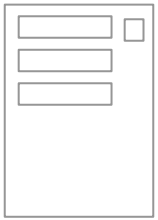
Was wurde bisher gemacht?
Verknüpfung Host-Gerät & SD-Karte über Adapter

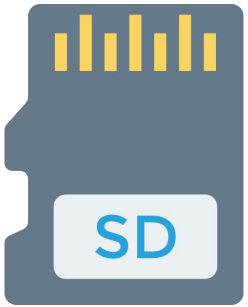


Verknüpfung SD-Karte mit Host-Gerät über Adapter

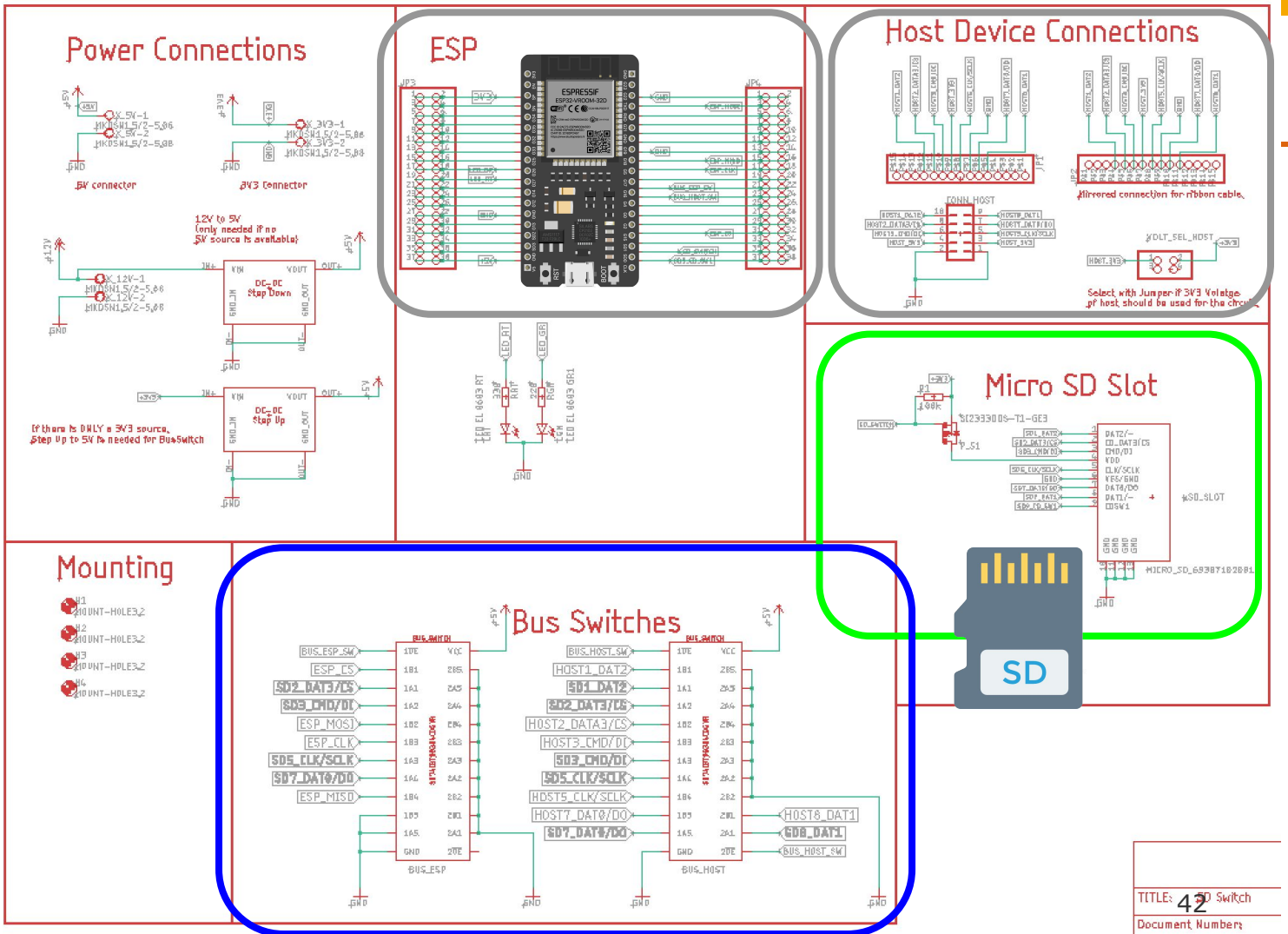
Was wurde bisher gemacht?
ESP-Host-Switch







exklusiver Zugriff auf die SD-Karte durch elektr. Abkopplung

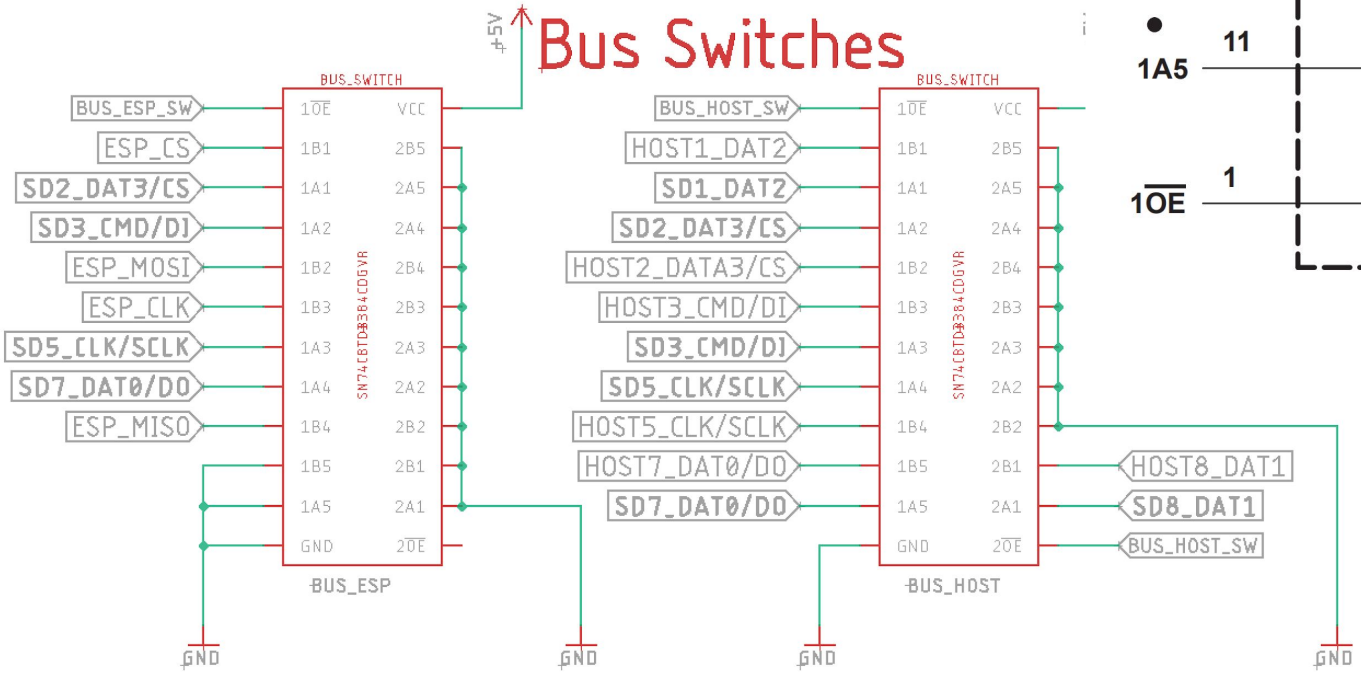


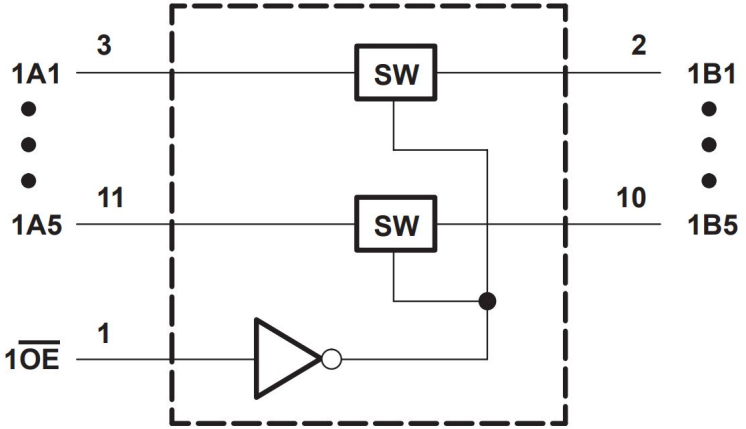
Hasso Plattner Institut

tplan

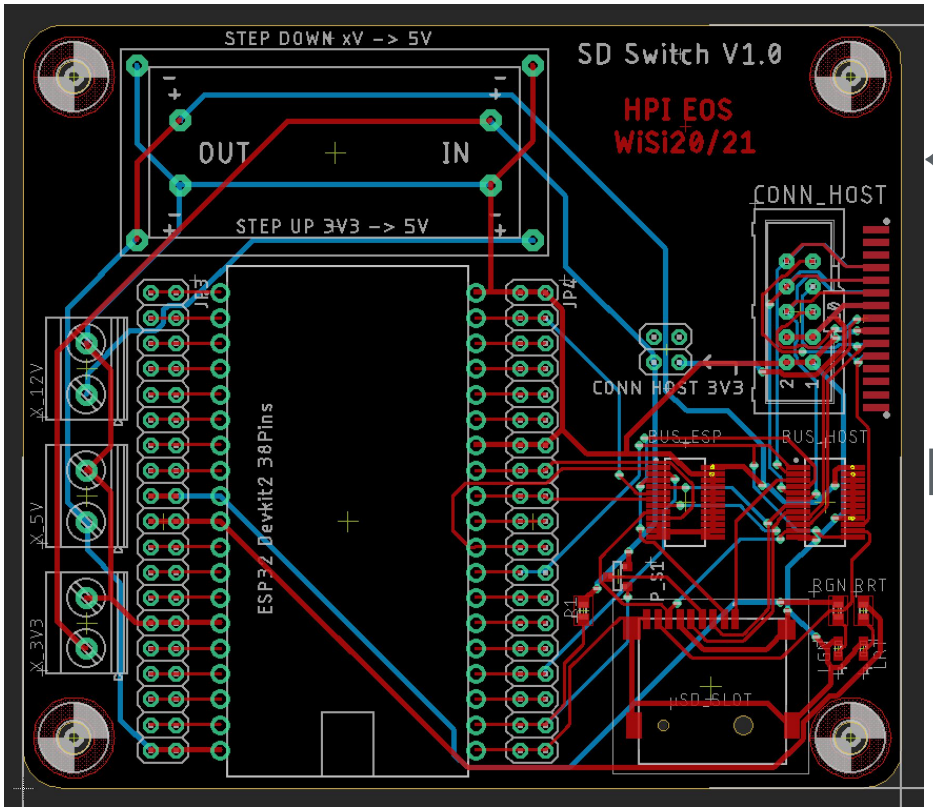
ESP-Host-Switch

Was wurde bisher gemacht?
ESP-Host-Switch



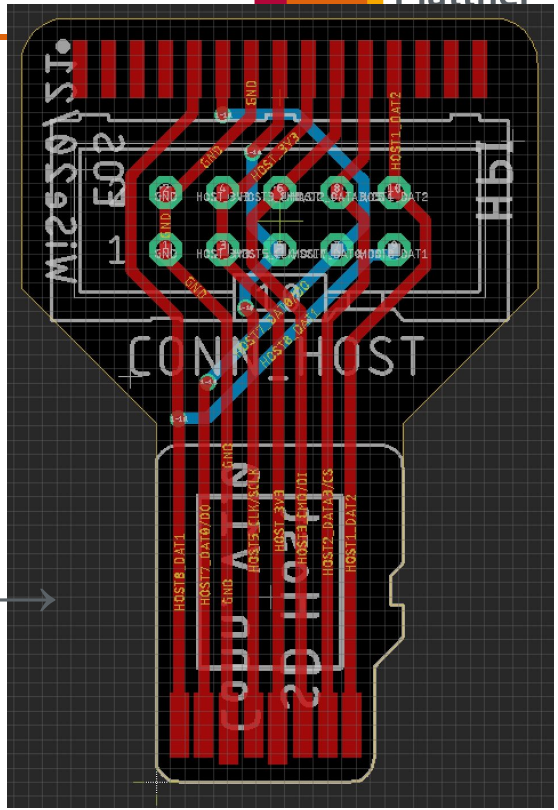


Was wurde bisher gemacht?
ESP-Host-Switch



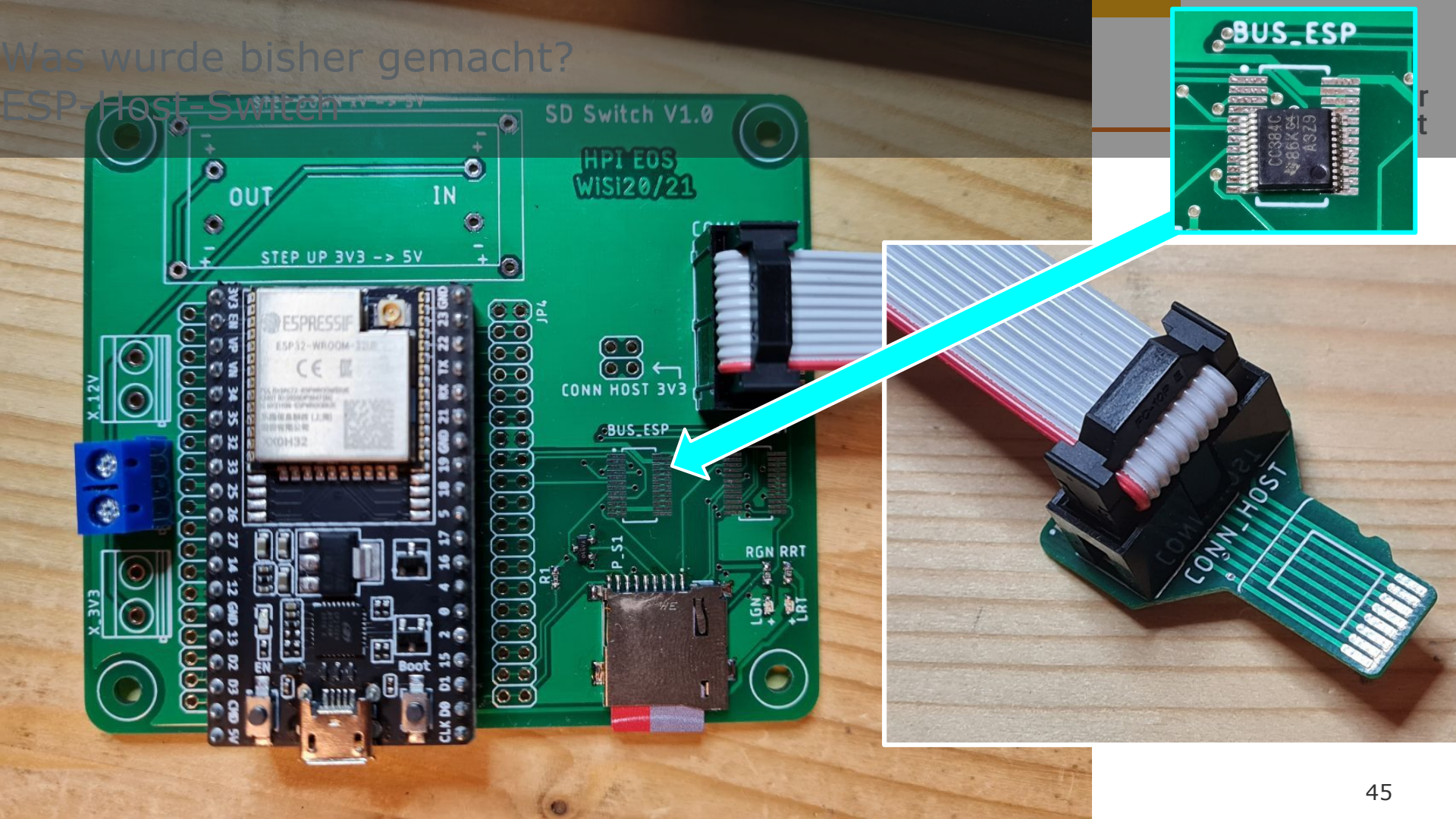
← Switch

Host-Conn. →

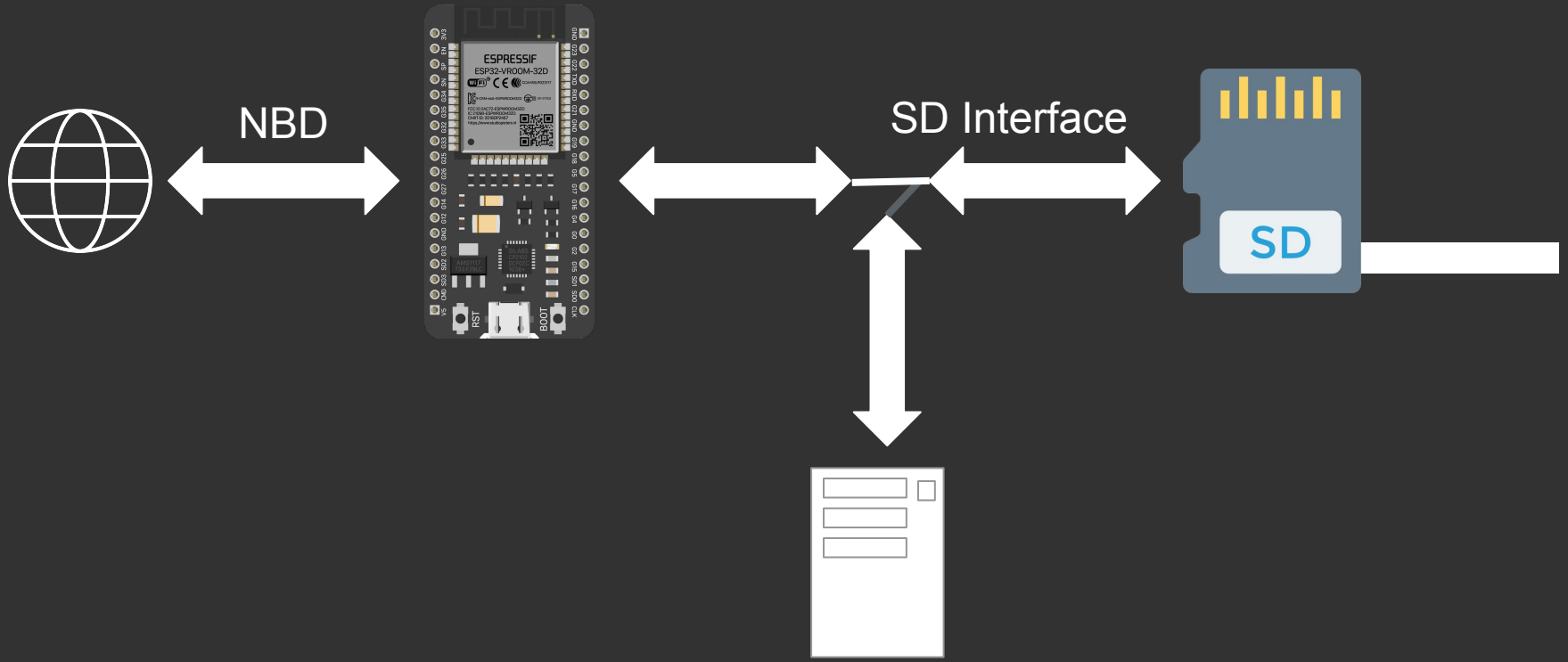


Was wurde bisher gemacht?

ESP-Host-Switch

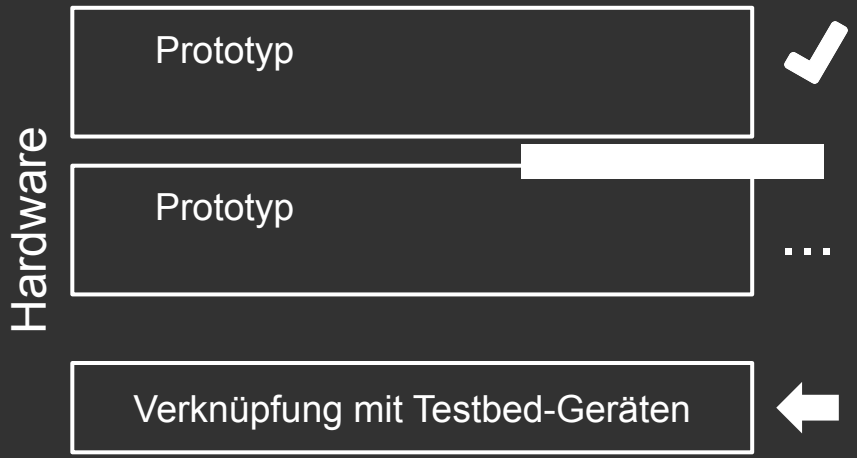


Aufbau: ESP NBD Server



NDB Live Demo

Zusammenfassung -
Was wurde geschafft? Was kommt als nächstes?



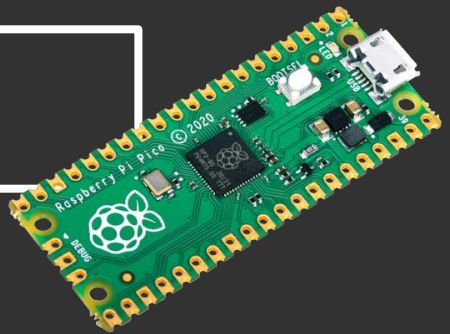
Zusammenfassung - Future Plans

Hardware

Prototyp → embedded Gerät
(kleiner, spezialisierter, funktionaler)

Integration Energiemanagement
(Ein- und Ausschalten)

Raspberry Pi Pico
SD-Karten-Simulator



Software

Einbinden der Funktionalitäten ins bestehende Testbed / Backend

Testen & Validieren aller Funktionalitäten

Verknüpfungspunkte zur Lehrveranstaltung

Hasso Plattner Institut

Verfügbare Ressourcen & Speichermanagement

- ESP32 → **4** MByte Flash & **320** KByte RAM
  - aktuell Flash: 260KB / 4MB (~6%) & RAM: 70KB / 320KB (~22%)
  - Programm: hauptsächlich statische Allokation
    - kein explizites *malloc()* etc.
    - aber: Verarbeitung Web-Requests → dynamische Allokation
    - Prinzip: direkte Verarbeitung Requests & Freigabe Speicher
    - Arbeitsspeicher begrenzt Verarbeitung größerer Pakete

Verknüpfungspunkte zur Lehrveranstaltung

Hasso Plattner Institut

Performance & Task-Management

- ESP32 → **160MHz Dual Core**
  - Aufgabe: Web-Requests verarbeiten und SD-Karte lesen/schreiben → Performance vollkommen ausreichend
  - Echtzeitbetriebssystem FreeRTOS, aber wir nutzen die Funktionen davon nicht direkt
  - keine direkten Echtzeitanforderungen:
    - stark aperiodische Tasks, laufen alle im selben Prozess
    - keine direkten Deadlines
- GPIO-Frequenz (im Allgemeinen) zu gering für natives SD-Interface

Verknüpfungspunkte zur Lehrveranstaltung

Hasso Plattner Institut

IDEs / Programmiersprache / Debugging

- Entwicklung mit VS Code (PlatformIO) & QT Creator (ESP-IDF)
- Programmiersprache: C/C++
- Debugging:
  - Debugging mit Breakpoints, Steps, etc. über JTAG-Connector möglich: → nicht verfügbar, daher kein richtiges Debugging
  - typisches Embedded-Debugging:
    blinkende LEDs, serielle Ausgaben & direktes Prüfen der Funktionen über Web-Zugriff
  - Wireshark für NDB Protokoll

Abschlusspräsentation 10.02.2021

Tobias Zagorni & Valentin Schröter