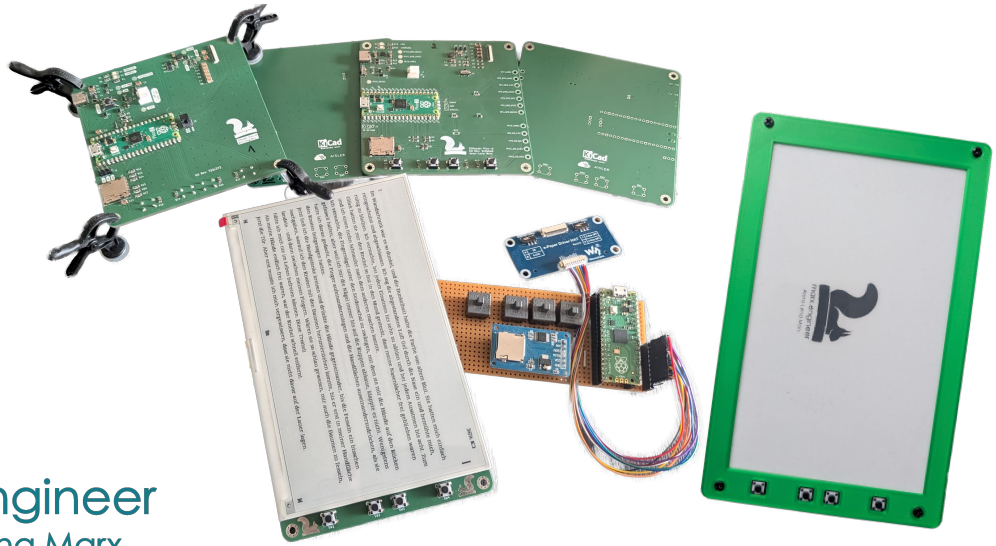
The ZEReader Project

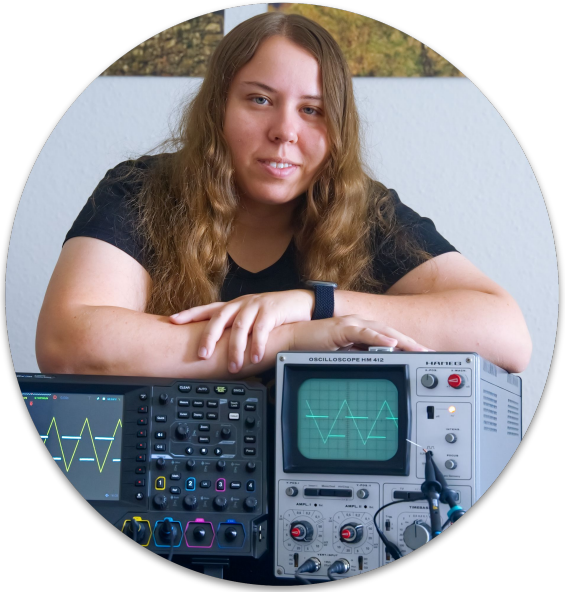
Adopting Agile and Software-First Methods in Open Hardware



marx.engineer
Anna-Lena Marx

Hi, I’m Anna

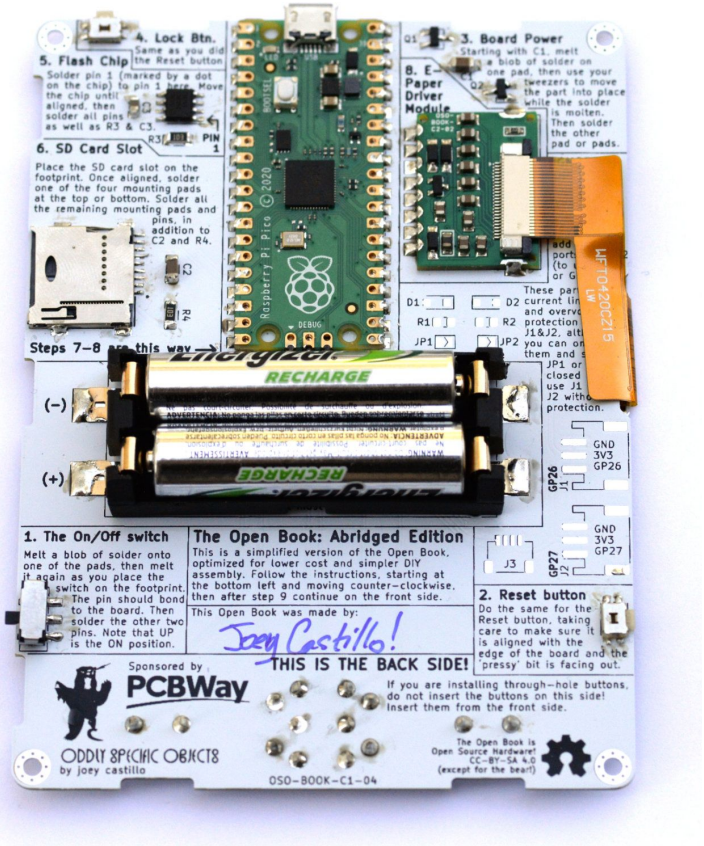
- Senior Embedded Linux Engineer /
  Tech Lead Embedded Linux Systems
  - B.Sc. Computer Science
  - M.Sc. Embedded Systems
  - Focus: Yocto, Linux, System Design, Kernel, ...



- but - I also studied Electrical Engineering
  - as a hobby
  - besides work
  - and eventually needed a bachelor’s project / thesis

The Bachelor’s Project - My requirements

- Electronics / Hardware project
  - I was already a proven software engineer, no need doing a software thesis again
- Built something for myself
  - not just taking the easiest possible thesis
  - something I’d like to continue afterwards


4. Lock Btn.
Same as you did the Reset button
5. Flash Chip
Solder pin 1 (marked by a dot on the chip) to pin 1 here. Move the chip until aligned, then solder all pins as well as R3 & C3.
PIN 1
6. SD Card Slot
Place the SD card slot on the footprint. Once aligned, solder one of the four mounting pads at the top or bottom. Solder all the remaining mounting pads and pins, in addition to C2 and R4.
Steps 7–8 are this way →
3. Board Power
Starting with C1, melt a blob of solder on one pad, then use your tweezers to move the part into place while the solder is molten. Then solder the other pad or pads.
8. E-Paper Driver Module
Raspberry Pi Pico © 2020
DEBUG
D1
D2
R1
R2
JP1
JP2
(–)
(+)
RECHARGE
GND
3V3
GP26
GP27
J1
J2
J3
1. The On/Off switch
Melt a blob of solder onto one of the pads, then melt it again as you place the switch on the footprint. The pin should bond to the board. Then solder the other two pins. Note that UP is the ON position.
The Open Book: Abridged Edition
This is a simplified version of the Open Book, optimized for lower cost and simpler DIY assembly. Follow the instructions, starting at the bottom left and moving counter-clockwise, then after step 9 continue on the front side.
This Open Book was made by:
Joey Castillo!
2. Reset button
Do the same for the Reset button, taking care to make sure it is aligned with the edge of the board and the 'pressy' bit is facing out.
THIS IS THE BACK SIDE!
Sponsored by
PCBWay
If you are installing through-hole buttons, do not insert the buttons on this side! Insert them from the front side.
ODDLY SPECIFIC OBJECTS
by joey castillo
The Open Book is Open Source Hardware! CC-BY-SA 4.0 (except for the bear!)
OSO-BOOK-C1-04

The Idea - Building my own Open Source E-Reader

- Microcontroller-based design
  - too much fear of memory timings as a “beginner”
  - and isn’t a multicore processor a bit of overkill for just reading books?
- Bigger Display
- LiPo with USB-C charging
- Best case: one single PCB
- Native EPUB handling - no preparsing on a host computer

But Wait - This Involves Software Again

And I have seen lots of failing projects in my life because of

- Hardware was specified first
- No one asked for software requirements
- Wasted time, money and very long cycle times

And at this point, I didn't even know if my guess nowadays microcontrollers are sufficient to handle EPUB natively was correct....

Inverting the Workflow - Software First

- Start with a Software-focused Proof-of-Concept
  - Verify or disprove initial assumptions and the general concept
  - Use devboards, breadboard modules or building blocks to evaluate crucial components

Key (for me)

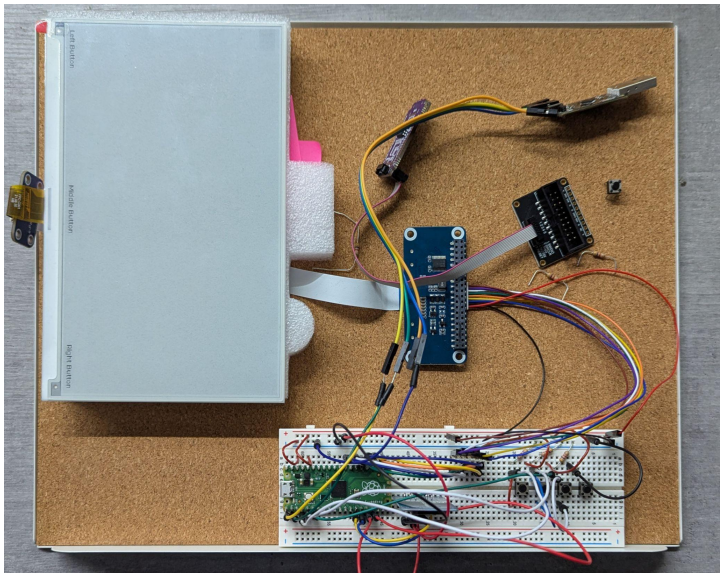
- Write as less code as possible
  → no time for writing each driver from scratch
- Flexible software with a good hardware abstraction
  → finale components were not yet selected

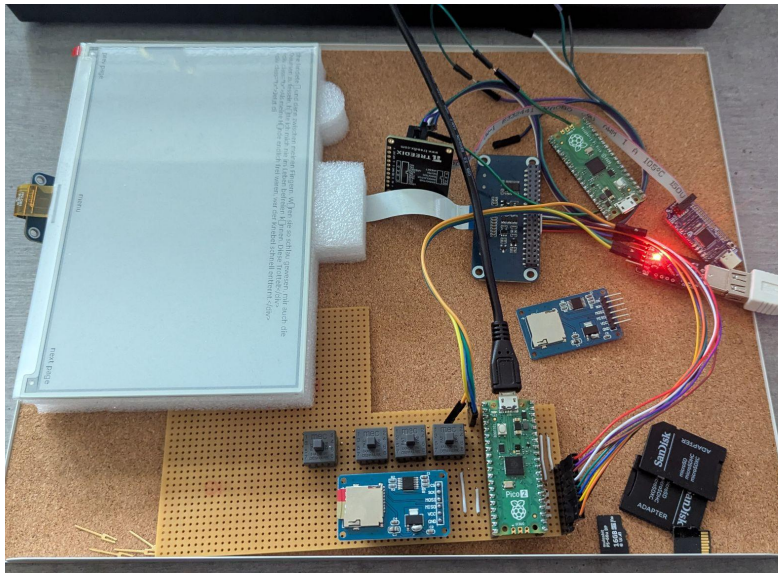
Inverting the Workflow - Utilizing Zephyr RTOS

- Hardware Abstraction
  - uses Device Trees (similar to Linux) to describe exact hardware
  - clearly defined API for subsystem drivers
  - → changing components is (best case) just adjusting a Device Tree

- Large Driver Ecosystem
  - select hardware components based on available drivers
    - → saves development time
    - → Allows focusing on the use-case and hardware development

Inverting the Workflow
Building a PoC with Off-the-Shelf Components





Inverting the Workflow
Building a PoC with Off-the-Shelf Components

```
/ {

  chosen {
    zephyr,display = &uc8179_waveshare_epaper_gdew075t7;
  };

  mipi_dbi_waveshare_epaper_gdew075t7 {
    compatible = "zephyr,mipi-dbi-spi";
    spi-dev = <&spi0>; //spi@0 -> GPIO 16-19
    dc-gpios = <&gpio0 20 GPIO_ACTIVE_HIGH>;
    reset-gpios = <&gpio0 21 GPIO_ACTIVE_LOW>;
    write-only;
    #address-cells = <1>;
    #size-cells = <0>;

    uc8179_waveshare_epaper_gdew075t7: uc8179@0 {
      compatible = "gooddisplay,gdew075t7", "ultrachip,uc8179";
      mipi-max-frequency = <4000000>;
      reg = <0>;
      width = <800>;
      height = <480>;
      busy-gpios = <&gpio0 22 GPIO_ACTIVE_LOW>;

      softstart = [ 17 17 28 17 ];

      full {
        pwr = [ 07 07 3f 3f 09];
        cdi = <0x07>;
        tcon = <0x60>;
        pll = <0x30>;
      };
```

```
  keys: keys {
    compatible = "gpio-keys";
    button1: button1 {
      gpios = <&gpio0 2 GPIO_ACTIVE_LOW>;
      zephyr,code = <INPUT_KEY_1>;
    };

    button2: button2 {
      gpios = <&gpio0 3 GPIO_ACTIVE_LOW>;
      zephyr,code = <INPUT_KEY_2>;
    };

    button3: button3 {
      gpios = <&gpio0 4 GPIO_ACTIVE_LOW>;
      zephyr,code = <INPUT_KEY_3>;
    };

    button4: button4 {
      gpios = <&gpio0 5 GPIO_ACTIVE_LOW>;
      zephyr,code = <INPUT_KEY_4>;
    };
  };

  lvgl_button_input: lvgl_button_input {
    compatible = "zephyr,lvgl-button-input";
    input = <&keys>;
    input-codes = <INPUT_KEY_1 INPUT_KEY_2 INPUT_KEY_3 INPUT_KEY_4>;
    coordinates = <10 470>, <360 470>, <450 470>, <770 470>;
  };
};
```

Software First - A Recap

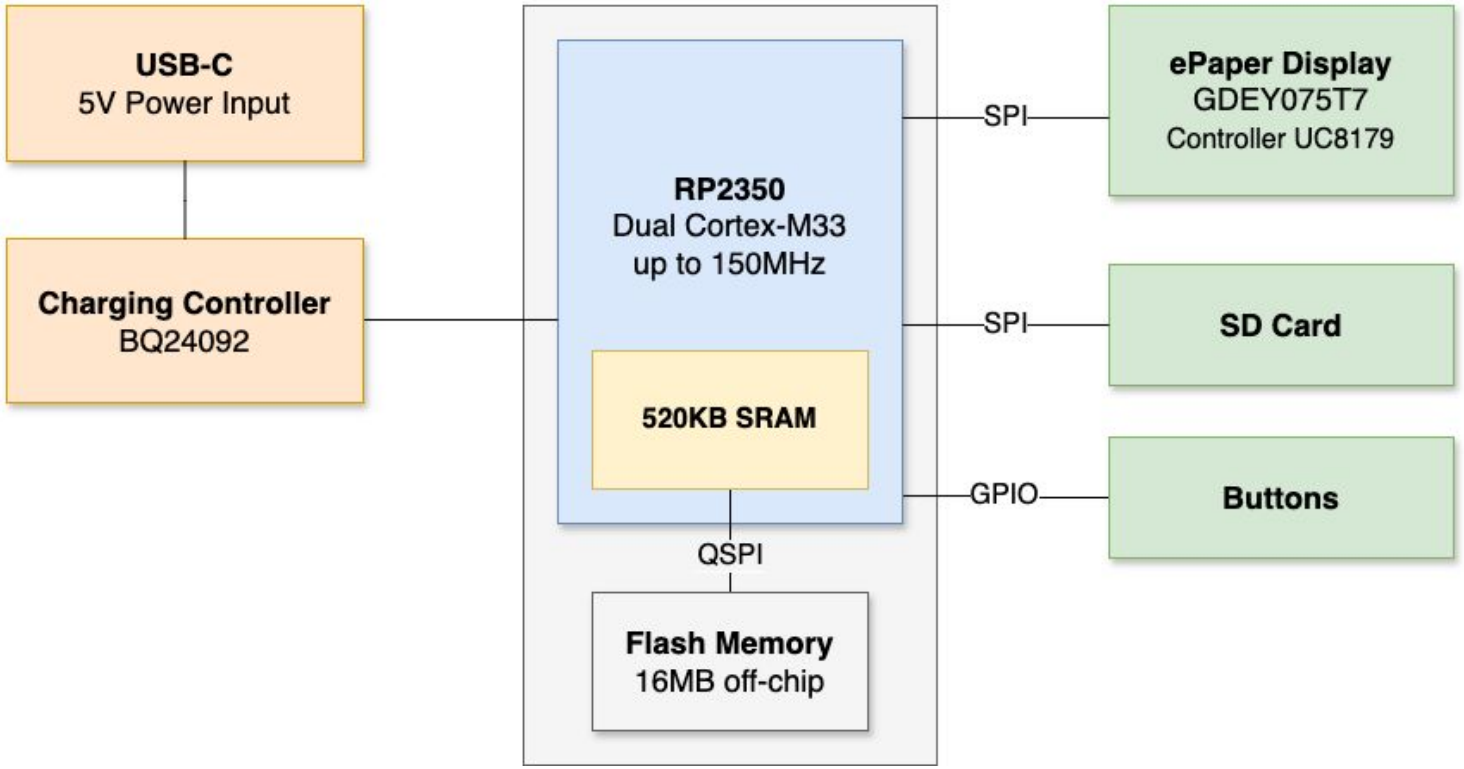
→ Confirmed the project's idea is viable
- but learned about potential problems, pitfalls, limitations
- … and RAM requirements!

→ Gained experience and confidence with components
- and a first selection for the hardware design

→ Minimal invest needed so far
- Off-the-Shelf components and Devboards from "The Drawer" → flexible and rapid
- Just bought "special" components, e.g. the e-paper display

→ Well-known software base to validate my upcoming PCB design

Let’s start with the fun part


USB-C
5V Power Input
Charging Controller
BQ24092
RP2350
Dual Cortex-M33
up to 150MHz
520KB SRAM
QSPI
Flash Memory
16MB off-chip
SPI
ePaper Display
GDEY075T7
Controller UC8179
SPI
SD Card
GPIO
Buttons

… But Wait - I’m still a Software Engineer

- Not that experienced with PCB design
  - With at least 5 design blocks I was not confident with
  - Just trying around and find out is still too expensive
  - And probably too slow iteration times (during the thesis)

→ **Modular approach needed!**

And, as a Software Engineer, I’d also like

- Version Control
- Traceability
- Automatisation
  - Testing
  - Generating Documentation
  - …
- Continuous Integration (CI) (and maybe even Continuous Delivery (CD)

Adopting Agile Approaches - Modularization

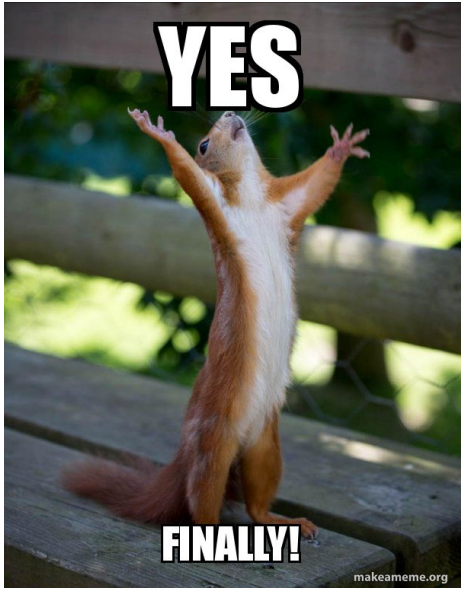
→ **Modular approach needed!**

- Split the design in functional blocks for risk minimization (and reusability?)
- Best Case: Build and verify each block (semi)-independently,
  e.g. as its own breakout board
  - Risk Minimization → verified in isolation → easier to track down design errors
  - Cost and roundtrip time is a limiting factor for hobbyists
- (Possible) Compromises:
  - Use the microcontroller dev board directly
    → most complex component, avoid risk until all other components are validated
  - 0-Ohms resistors to decouple functional components as possible
    → bring up one component at a time

Adopting Agile Approaches - Modularization in KiCAD

(Maybe?) a perfect fit for modularization in KiCAD 10:

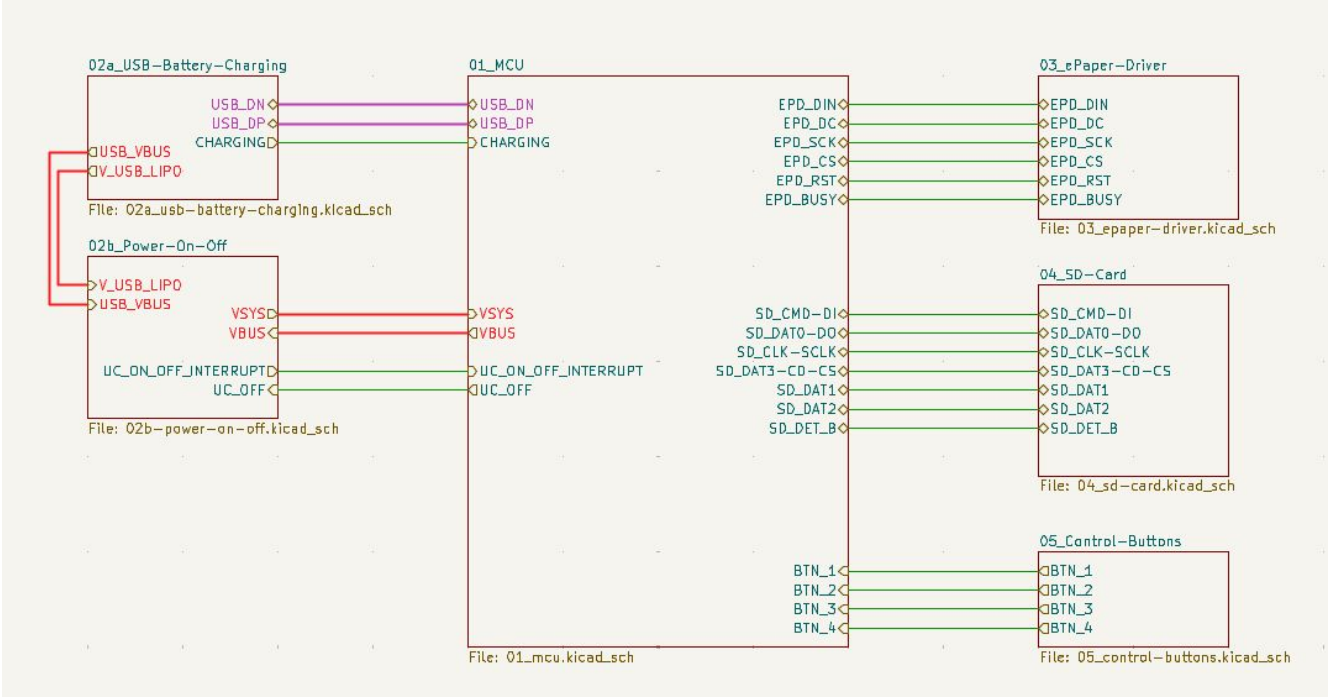
→ **Design Blocks**



But I started the first revision with KiCAD 8 and the second one with KICAD 9

Adopting Agile Approaches - Modularization in KiCAD

Best thing I had to start: **Hierarchical Schematics**



Adopting Agile Approaches - Version Control

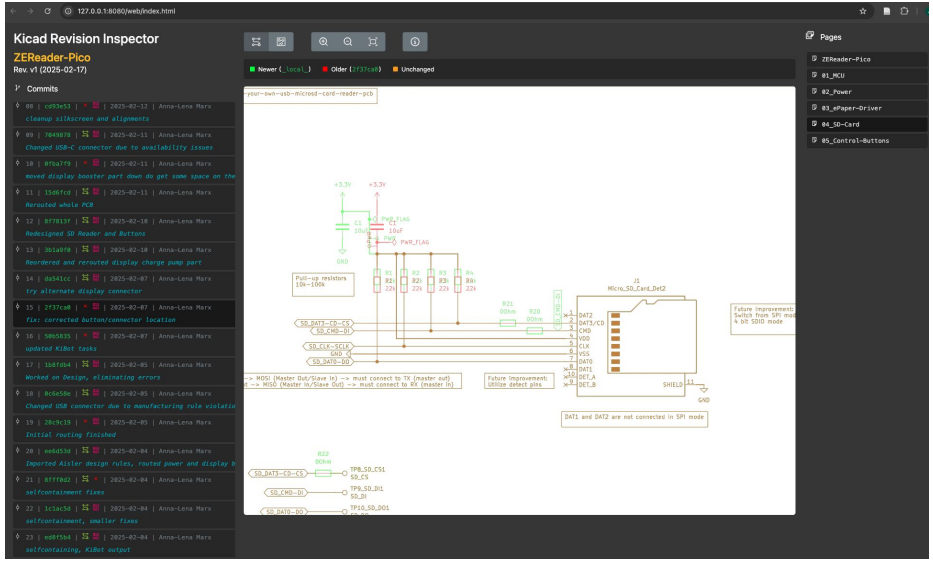
→ the base needed for

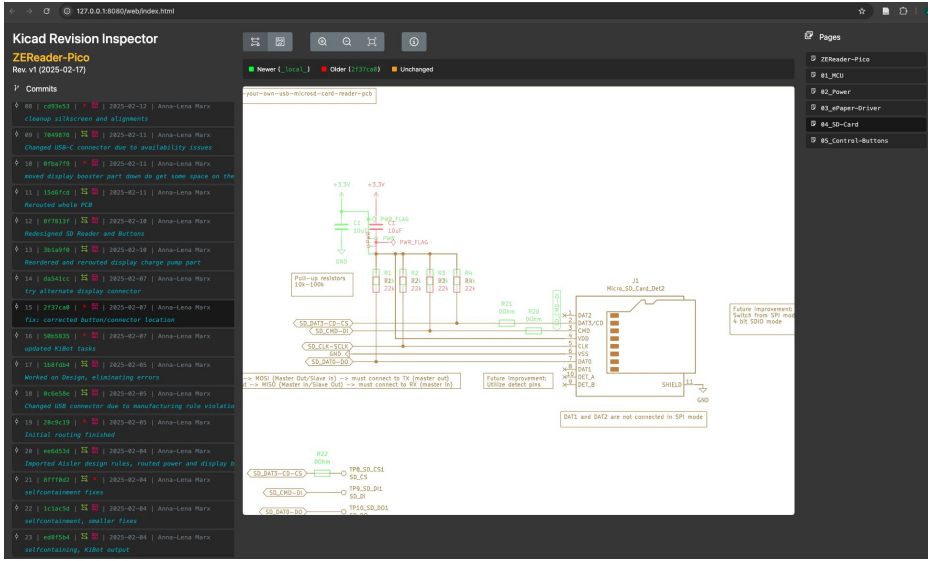
- Traceability
- Automation
- Documentation
- Continuous Integration (CI) (and maybe even Continuous Delivery (CD)

- Introduce **git** (version control system → GitHub, GitLab, Codeberg, ...)
  - KiCAD had no native git integration before 9.0
    → but not a problem as a software engineer

- Utilize **KiCAD Revision Inspector (KiRI)** for visual diffs!

Adopting Agile Approaches - Version Control





Adopting Agile Approaches
Version Control, Traceability, CI and Automation

- Each git commit triggers a CI run
  - Replaces a git commit hash placeholder with the actual commit hash
  - Generates (with KiBot or JobSets)
    - Production files
    - Documentation
    - Renders
    - ...
  - makes produced PCBs traceable
    → find the matching BoM, documentation, ...
    → trace development steps and errors

.github/workflows/kibot.yml

```yaml
jobs:
  kibot:
    name: "KiBot"
    runs-on: ubuntu-latest

    steps:
    - name: Checkout repo
      uses: actions/checkout@v4
      with:
        fetch-depth: '0'

    - name: Replace vars
      run: |
        DATE=$(TZ=Europe/Berlin date --iso-8601=minutes)
        sed -i "s|{kibot_date}|$DATE|g" *.kicad_*

        REV=$(git log -1 --format="%h" ZEReader-Pico.kicad_pcb)
        sed -i "s/{kibot_git-hash}/$REV/g" *.kicad_*

    - name: Run KiBot
      uses: INTI-CMNB/KiBot@v2_k8
      with:
        config: 'zereader-pico.kibot.yaml'
        dir: KiBot_Outputs
        schema: 'ZEReader-Pico.kicad_sch'

    - name: Upload results
      if: ${{ always() }}
      uses: actions/upload-artifact@v4
      with:
        name: KiBot_Outputs
        path: KiBot_Outputs
```

Adopting Agile Approaches
Version Control, Traceability, CI and Automation

zereader-pico.kibot.yaml

```yaml
    drc:
      enabled: true
      # Set some options to avoid stopping the KiBot rur
      dont_stop: true
      ignore_unconnected: false

    fill_zones: true
    update_xml: true

outputs:
  - name: 'IBoM'
    type: ibom
    dir: BoM

  - name: 'HTML BoM'
    type: bom
    dir: BoM
    options: &bom_options
      format: HTML

  - name: 'Schematic'
    type: pdf_sch_print
    dir: Documentation

  - name: 'PCB Sheet'
    type: pcb_print
    dir: Documentation
    options:
      frame_plot_mechanism: 'gui'
      force_edge_cuts: true
```

ZEReader-Pico.kicad_jobset

```
{
  "jobs": [
    {
      "description": "",
      "id": "dc510f61-bb82-45ae-8bab-ee4255af6610",
      "settings": {
        "description": "",
        "fail_on_error": true,
        "format": "report",
        "output_filename": "erc-report.rpt",
        "severity": 48,
        "units": "mm"
      },
      "type": "sch_erc"
    },
    {
      "description": "",
      "id": "3faa5922-afd4-4f68-8e52-57fd484a2ab2",
      "settings": {
        "description": "",
        "fail_on_error": true,
        "format": "report",
        "output_filename": "drc-report.rpt",
        "parity": true,
        "report_all_track_errors": false,
        "severity": 48,
        "units": "mm"
      },
      "type": "pcb_drc"
    },
    {
      "description": "Copy ZEReader-Pico.kicad_pcb",
```

Adopting Agile Approaches
Version Control, Traceability, CI and Automation

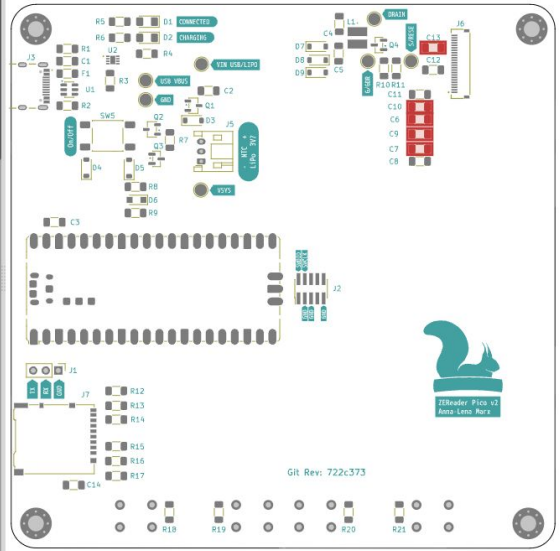
ZEReader-Pico
Anna-Lena Marx

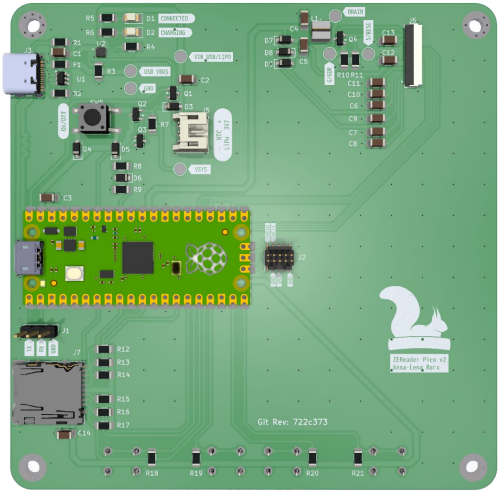
Rev: v2
2025-11-21T21:29+01:00

F | FB | B

Ref lookup | Filter

	Sourced	Placed	References	Value	Footprint	DigiKey	Manufacturer PartNo	Quantity
1	☐	☐	C6, C7, C9, C10, C13	1uF, 25V	C_1206_3216Metric_Pad1.33x1.80mm_HandSolder	1276-3091-1-ND	CL31B105KBHNFNE	5
2	☐	☐	C1, C2, C3, C14	10uF	C_1206_3216Metric_Pad1.33x1.80mm_HandSolder	1276-1804-1-ND	CL31B106KAHNNNE	4
3	☐	☐	C4, C5	4.7uF	C_1206_3216Metric_Pad1.33x1.80mm_HandSolder	1276-3179-1-ND	CL31B475KBHNFNE	2
4	☐	☐	C8	1uF, 50V	C_1206_3216Metric_Pad1.33x1.80mm_HandSolder	1276-3091-1-ND	CL31B105KBHNFNE	1
5	☐	☐	C11	1uF, 10V	C_1206_3216Metric_Pad1.33x1.80mm_HandSolder	1276-3091-1-ND	CL31B105KBHNFNE	1
6	☐	☐	C12	10uF, 10V	C_1206_3216Metric_Pad1.33x1.80mm_HandSolder	1276-1804-1-ND	CL31B106KAHNNNE	1
7	☐	☐	R5, R6, R7, R9, R10, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21	10k	R_1206_3216Metric_Pad1.30x1.75mm_HandSolder	311-10.0KFRCT-ND	RC1206FR-0710KL	15





Adopting Agile Approaches - Limitations and Wishes

- Cost and Time
  - Own “breadboard modules” was too much effort during a thesis (limited timeframe)

- Not all software workflows and tools are applicable
  - Standalone git is nearly useless for graphically coded contents → KiRI helps
  - No self-contained states while working on a new revision or a redesigning components
    → Best/good professional workflow? No clue right now
    → Maybe design blocks could help

Adopting Agile Approaches - Wishes

Still hoping for

- agile techniques in hardware development becoming more adopted by professional teams
- closer collaboration between hard- and software engineers

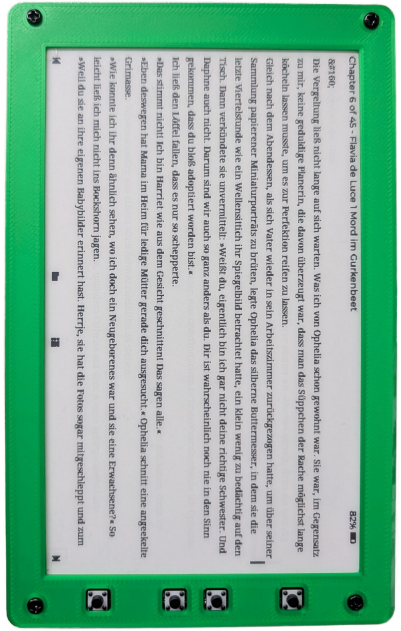
- More time to work on fun projects?

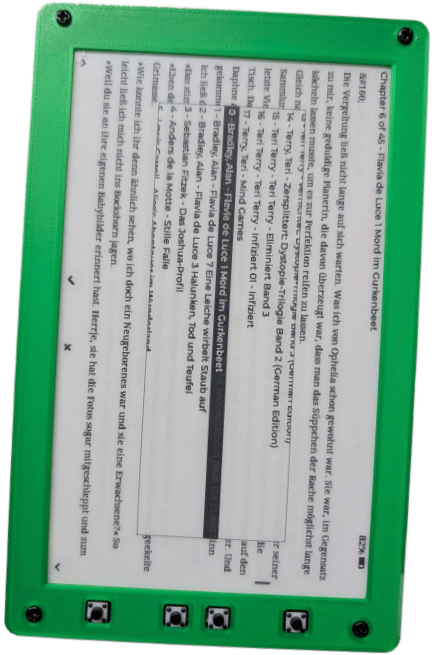
ZEReader - Current State

- Thesis finished since mid 2025
  → now: 100% hobby
  → most limiting factor: time
- Second PCB revision finished in winter 2025
  - Lots of hardware improvements
  - New issues found
  - And new ideas for improvements in mind and new KiCad features to adopt
- Software is a never-ending story
  - Major revisit to bring the PoC to a proper architectural state and stable base
  - Rotation for the E-Paper display needs to be implemented
  - EPUB handling is still rather basic and not standard conform
  - No picture support yet
  - And lots more
- First casings designed in spring 2026

  → Still just a prototype / pet project, but I read the first books using ZEReader

ZEReader Today





Interested?

https://marx.engineer/zereader

https://github.com/Allegra42/ZEReader

https://github.com/Allegra42/ZEReader-KiCad

And thanks to AISLER for supporting the project!