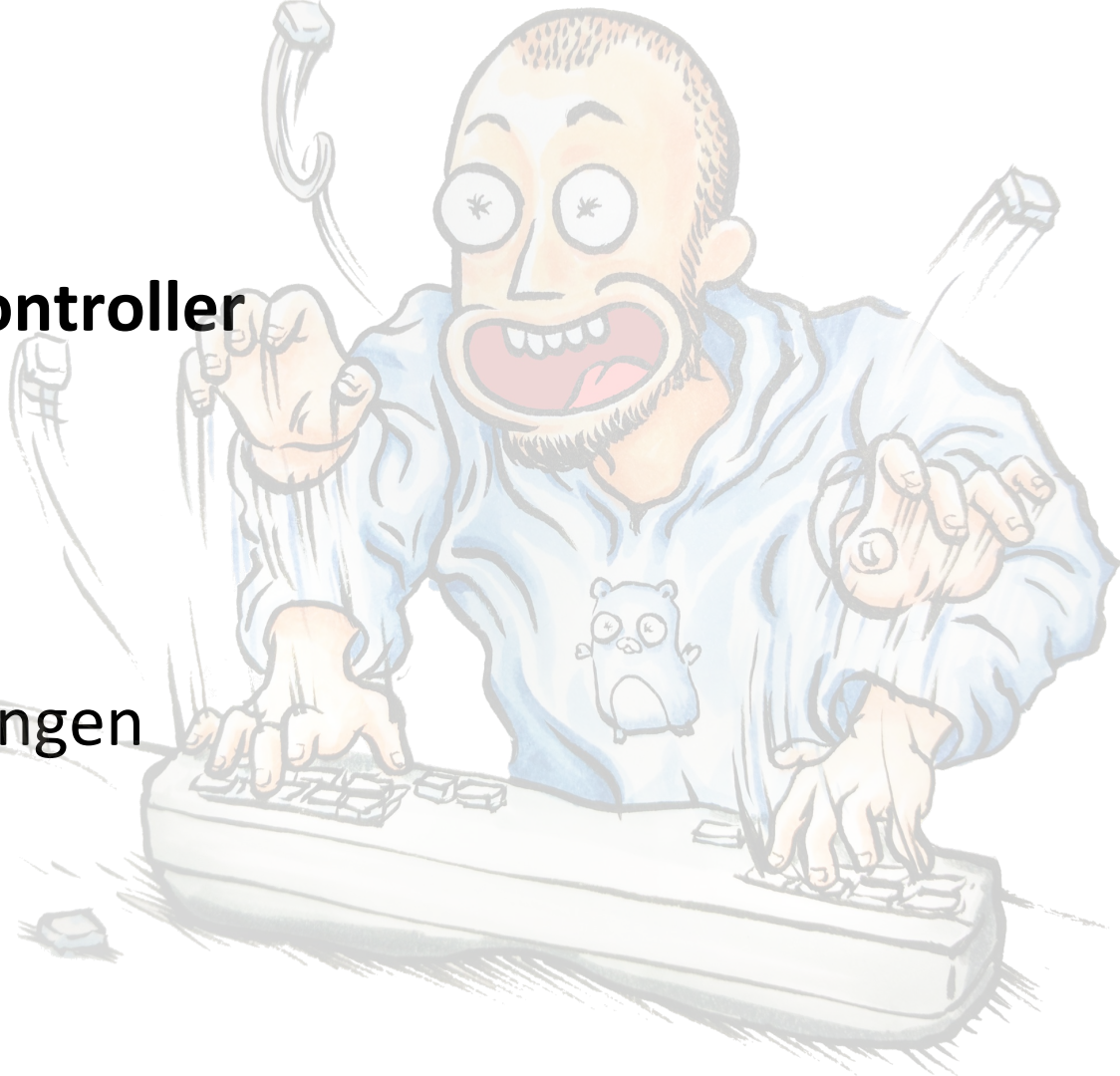
kinX: keyboard hacking

Michael Stapelberg

2018-05-12

Inhalt

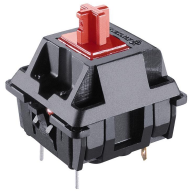
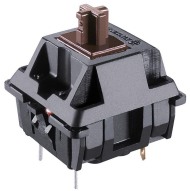
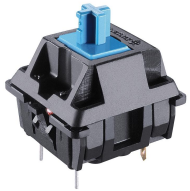
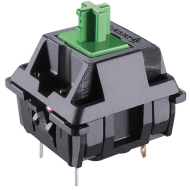
- **eigener keyboard controller**
- eigener USB hub
- input latency Messungen



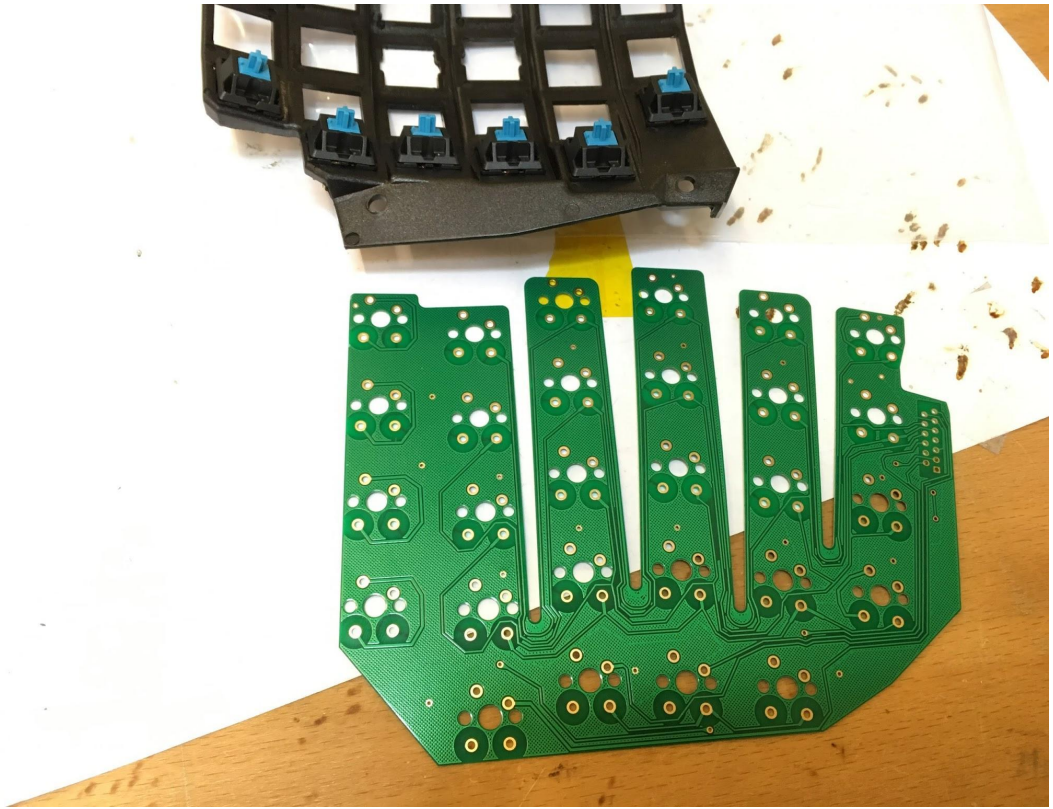
Eigener controller: Motivation

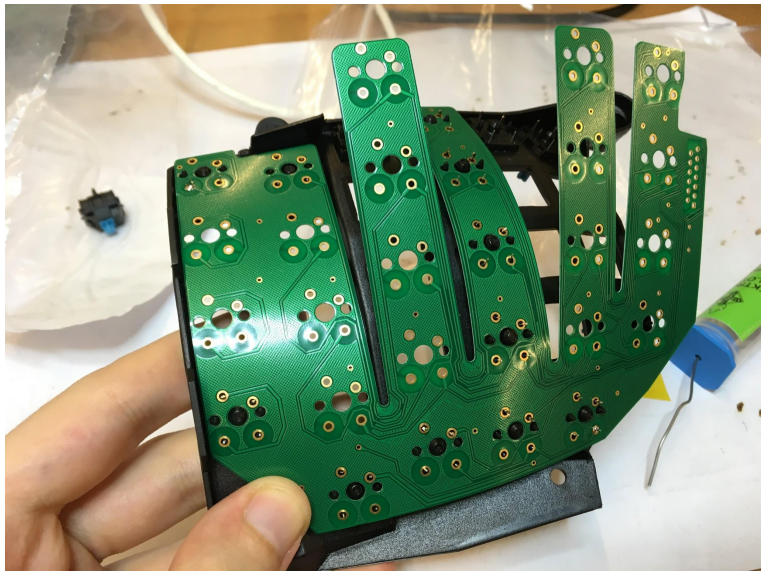
- Kinesis Advantage-Nutzer seit 2008

- Super Tastatur! Aber:
  - braune Cherry MX
  - stuck modifier Bug

	linear	tactile	clicky
soft			
medium			
stiff			

Kinesis mit eigenen Tasten





Tastatur-Matrix

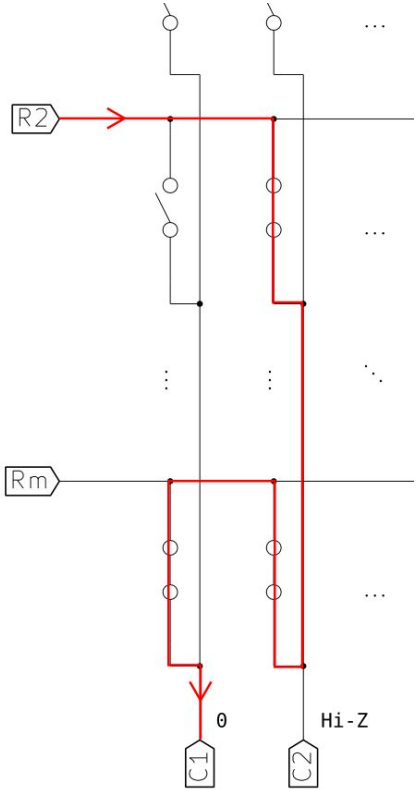


image source: http://blog.komar.be/how-to-make-a-keyboard-the-matrix/

Kinesis Tasten brauchen Dioden

- Cherry MX1A-E1**D**W mit Diode

- Schwierig zu bekommen! (nicht in großen Shops)
  - samples von Cherry
  - Gruppenbestellung im deskthority-Forum
  - selbst Dioden in jede Taste einbauen

Stuck modifier-Bug

- Shift drücken, a drücken, a loslassen, Shift loslassen
  - Tastatur verwirft “Shift loslassen”
    → Shift bleibt gedrückt
  - betrifft auch andere Modifier (Caps Lock)

- laut Foren weiß Kinesis von dem Problem

Stuck modifier-Bug (2)

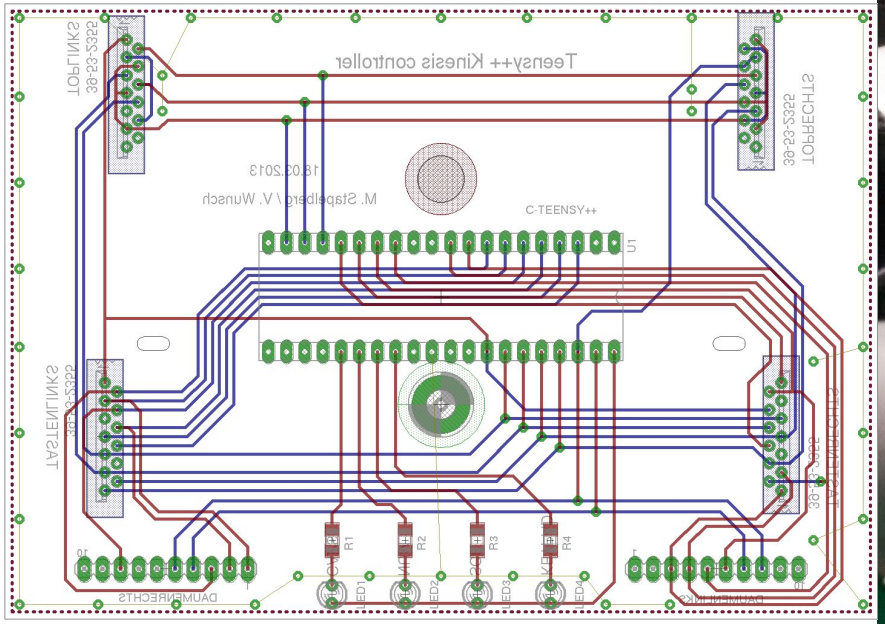
- tech support schickt mir ein “firmware update”
- d.h. ein neuer Microcontroller per Post!
- Shift seltener betroffen, Caps Lock immer noch :-(
  - wichtig wenn man das NEO layout nutzt

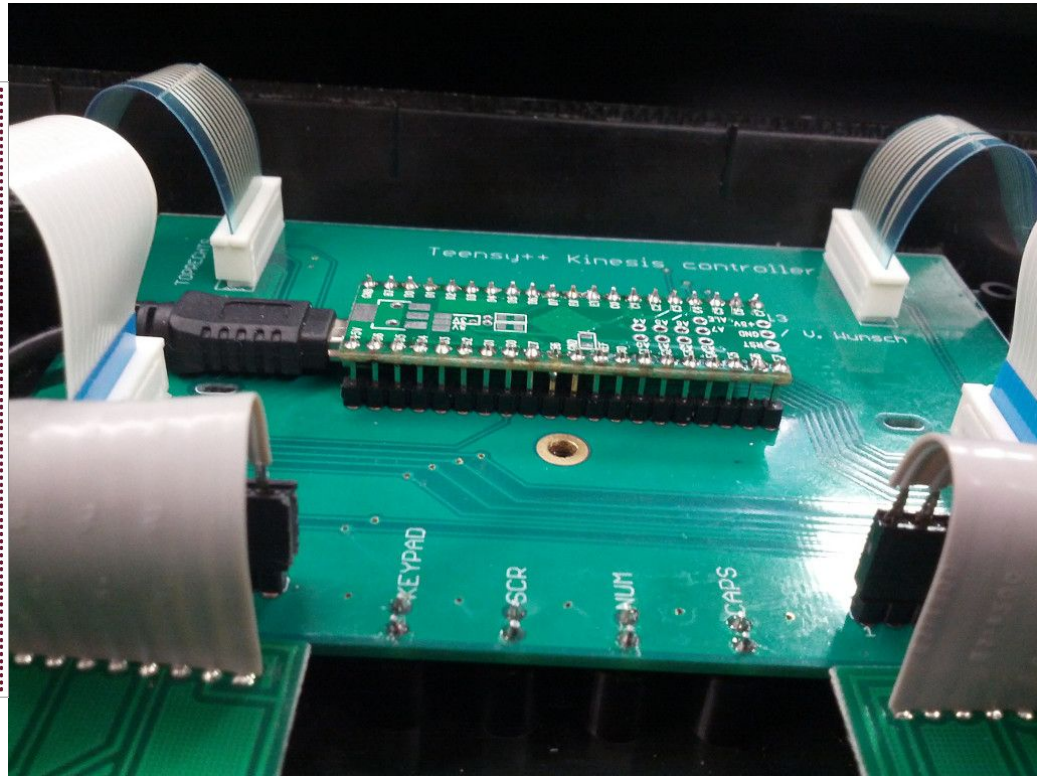
Eigener keyboard controller

- humblehacker blog posts
  - super reverse-engineering (ältere Kinesis)
  - eigene Firmware, ohne stuck modifier-Bug

- ich habe das Projekt 2013 nachgebaut

Eigener keyboard controller (2)





Eigener keyboard controller (3)

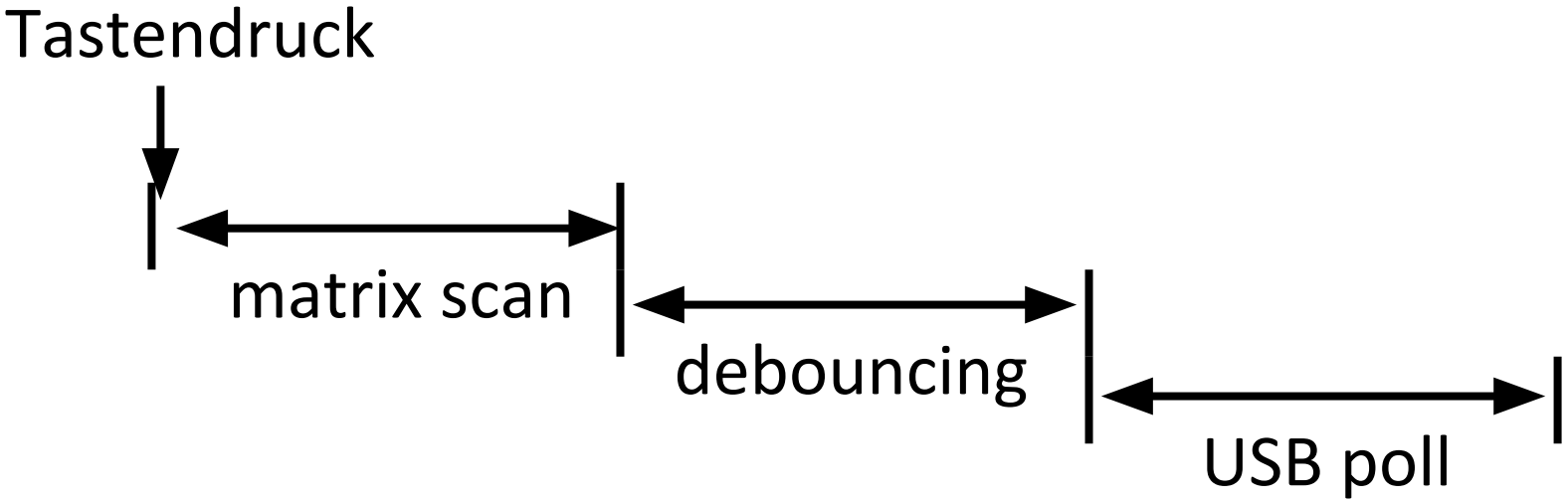
- vorsichtig vorgehen:
  Teensy++ mit humblehacker-Firmware nutzen

- hat für 4 Jahre gut funktioniert

input latency

- Leute schreiben über input latency
  - super Artikel: Pavel Fatins “typing with pleasure”
- Was ist die input latency meiner Tastatur?
- Was kann ich ändern? Wie weit kann ich gehen?

Quellen von input latency



humblehacker-Firmware input latency

- scannt die Matrix nach jedem USB poll (?)
- debouncing: wartet bis zu 8 Zyklen hintereinander
- standardmäßig 10ms USB polling interval
  - auf 1ms reduziert

humblehacker-Firmware input latency (2)

- Verbesserungsversuch des debouncing scheitert

- Offenbar muss ich mehr über das Thema lernen
  - also bauen wir eine Firmware von Grund auf!

- mit einem neueren Teensy
  - das Design wird dann nicht so schnell obsolet

Einwurf: Teensy (von PJRC)

Teensy++ (16 MHz)
Atmel AVR

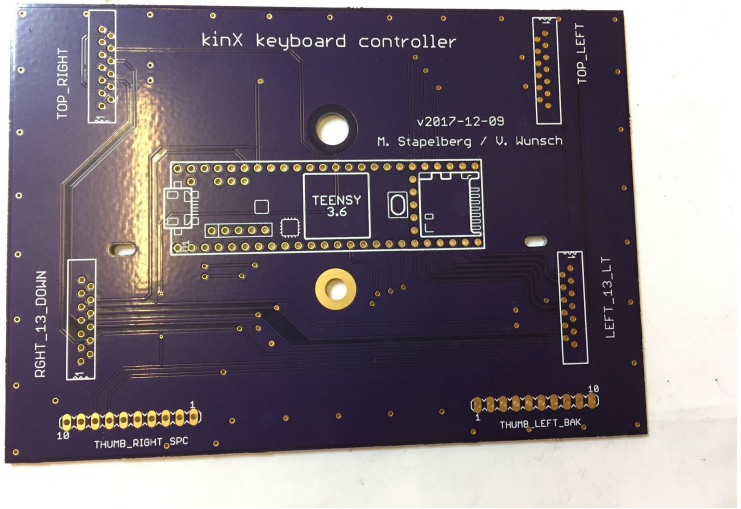
Teensy 3.6 (180 MHz)
NXP ARM Cortex M4

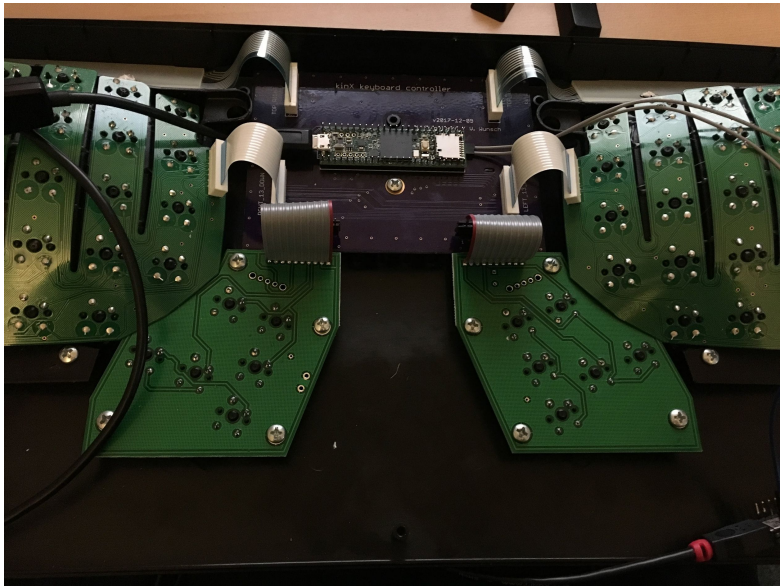
Bauen wir einen keyboard controller!

Bauen wir einen keyboard controller! (2)

- Schaltplan und Layout aufräumen:
  - Kinesis PCB Namensschema nutzen
  - Platzierungsprobleme beheben (Loch)
  - keyboard-Matrix umlegen: von humblehacker auf ein bequemeres Layout

Bauen wir einen keyboard controller! (3)





Bare metal-Firmware

- fast von grund auf-Firmware
  - nutzt gcc-arm-none-eabi, libnewlib-arm-none-eabi
  - teensy3 linker script wiederverwendet
  - eigener startup, USB, keyboard code

- bare metal damit nichts dazwischenfunkt

firmware: scanning (bis zu 0.1ms delay)

- Teensy 3.6 auf 180 MHz betreiben
- scan() in einer Schleife (nicht nur bei USB poll)
- alle Spalten mit einem Speicherzugriff lesen
  - daher alle an einen GPIO port verkabelt

firmware: debouncing

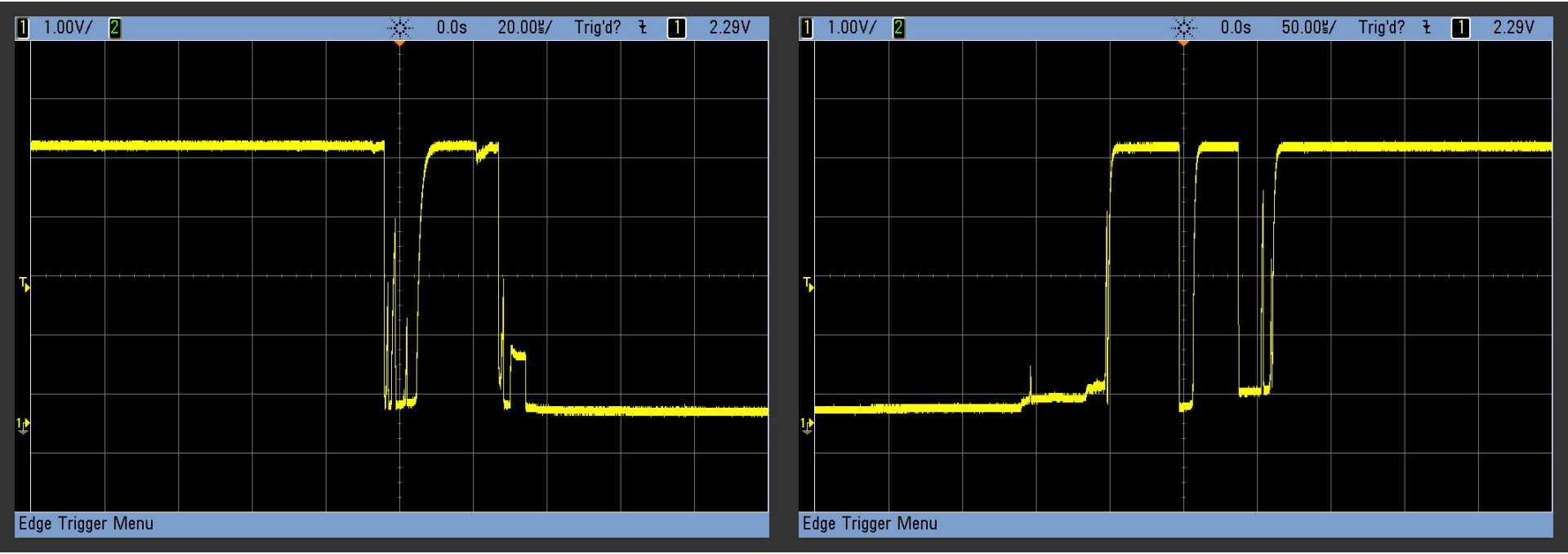
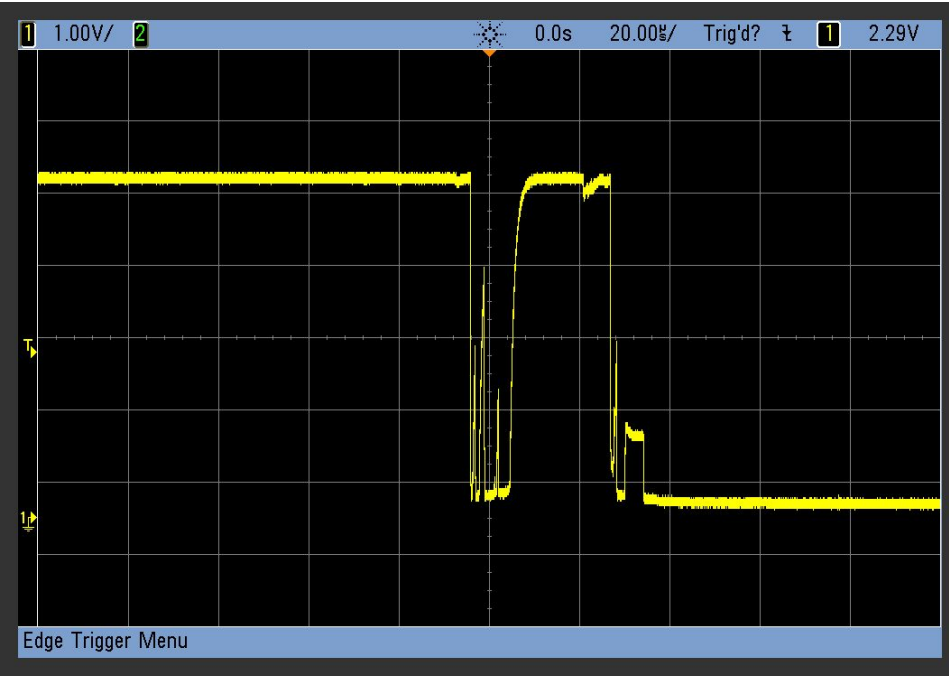


image source:
https://hackaday.com/2015/12/10/embed-with-elliot-debounce-your-noisy-buttons-part-ii/

debounce: langsame scan rate (> 5ms)



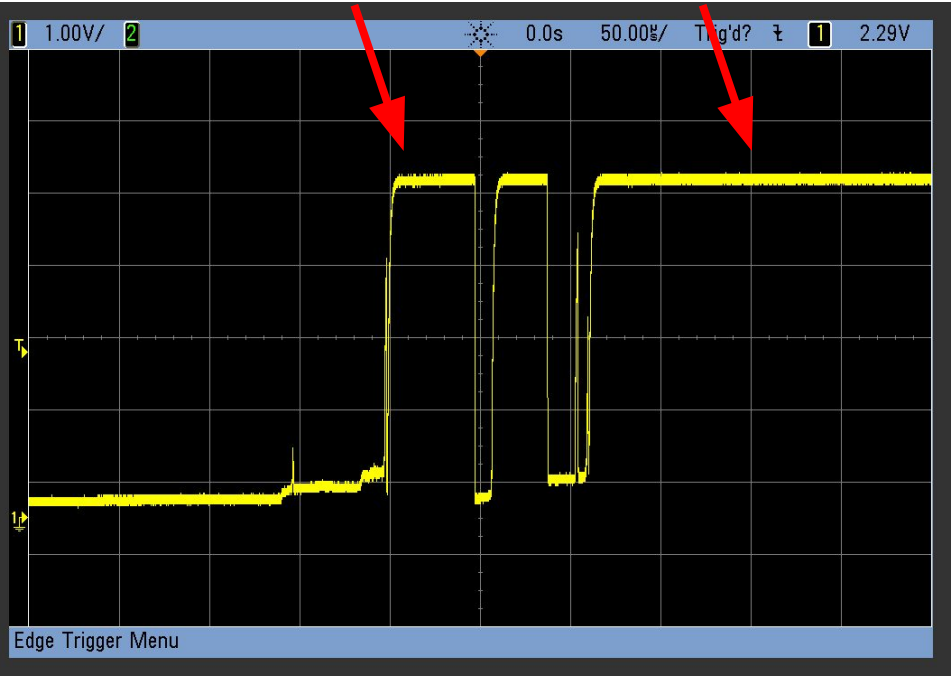


image source:
https://hackaday.com/2015/12/10/embed-with-elliot-debounce-your-noisy-buttons-part-ii/

debounce: langsame scan rate: worst case

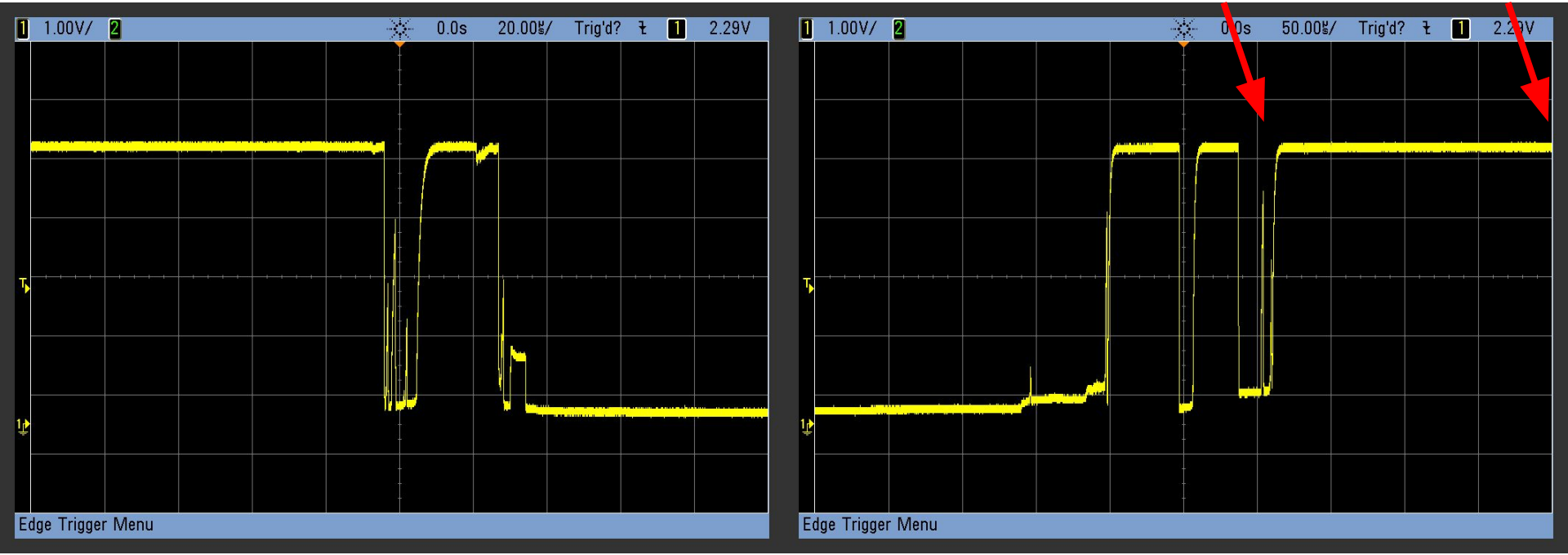
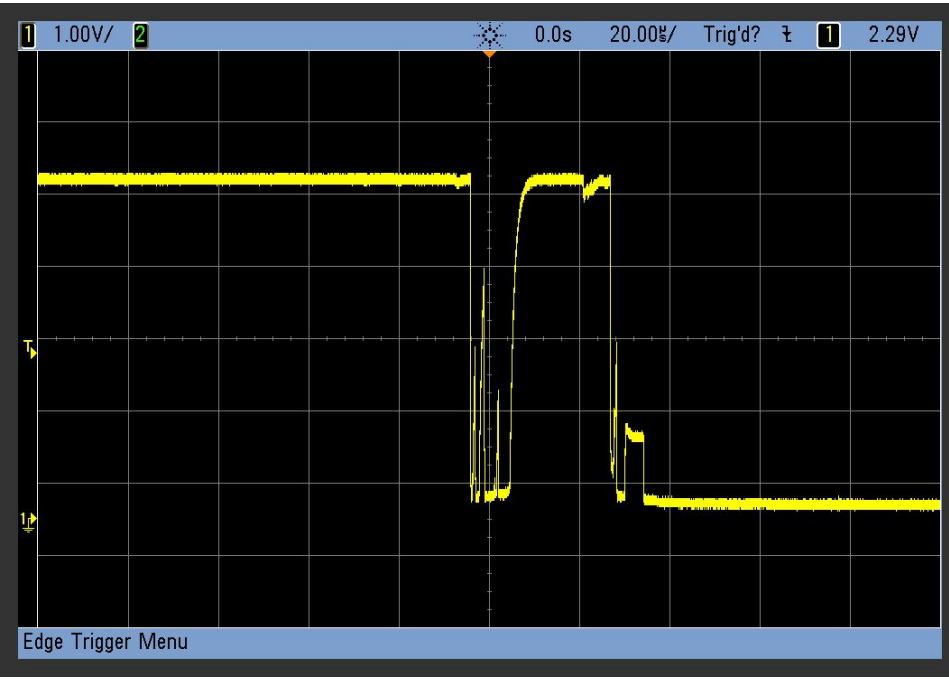


image source:
https://hackaday.com/2015/12/10/embed-with-elliot-debounce-your-noisy-buttons-part-ii/

debounce: 5ms warten (→ 5ms delay)



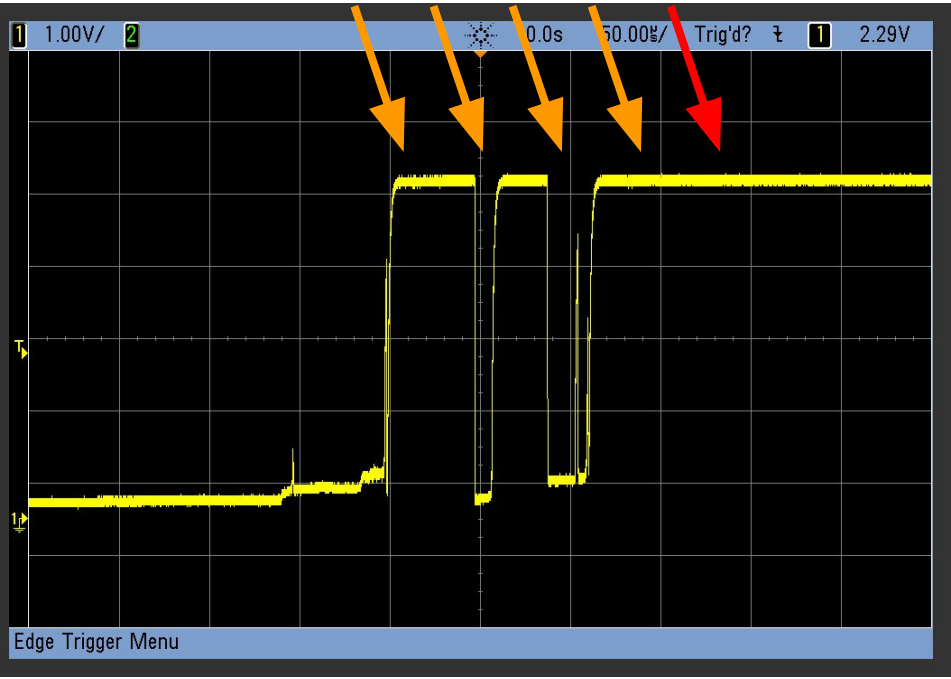
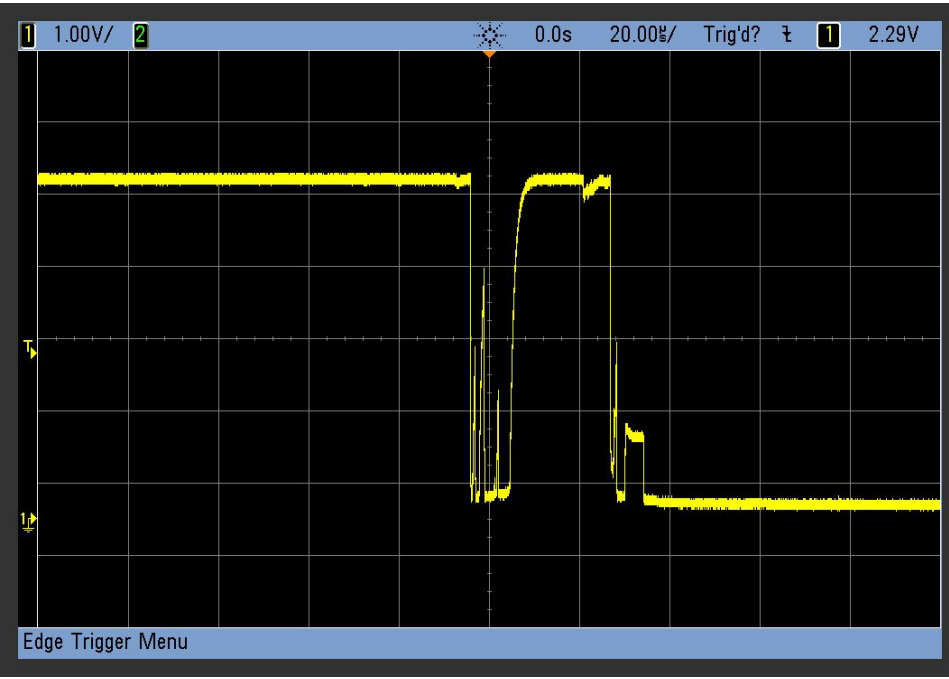


image source:
https://hackaday.com/2015/12/10/embed-with-elliot-debounce-your-noisy-buttons-part-ii/

debounce: trigger, dann warten (→ 0ms)



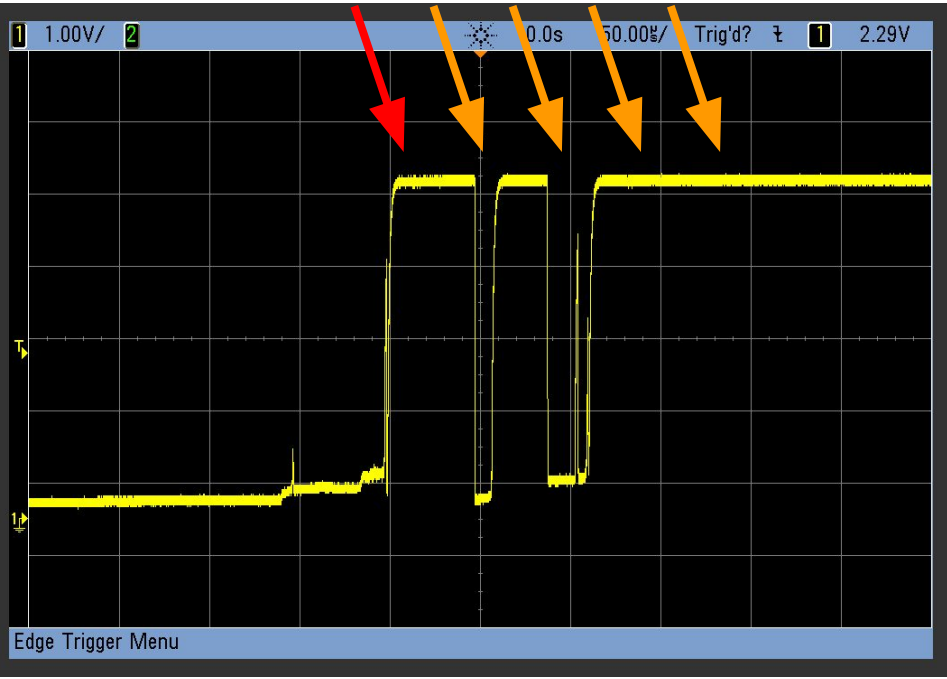
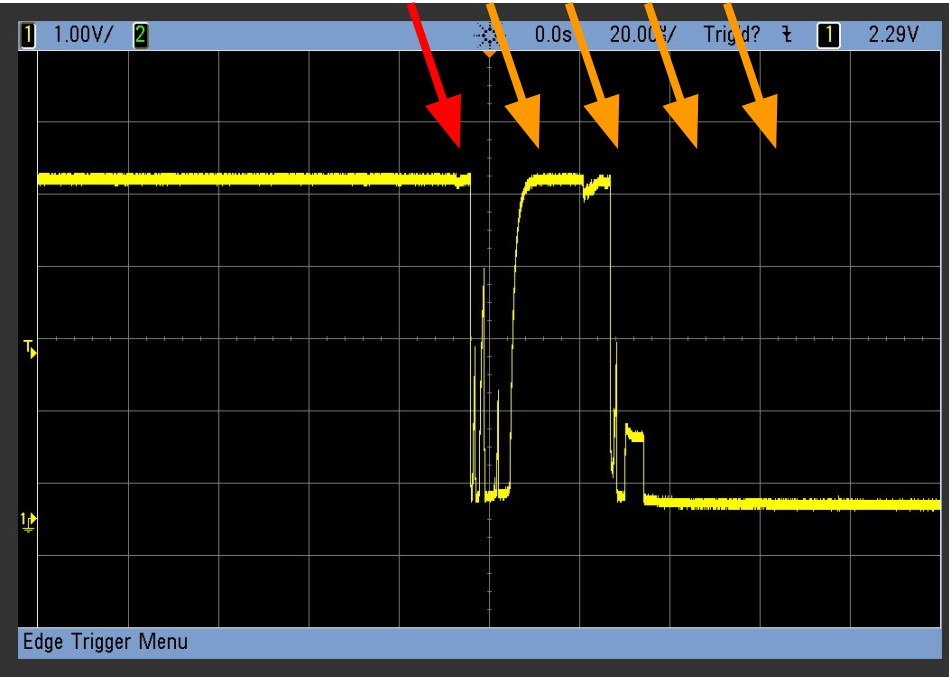


image source:
https://hackaday.com/2015/12/10/embed-with-elliot-debounce-your-noisy-buttons-part-ii/

debounce: Taste drücken



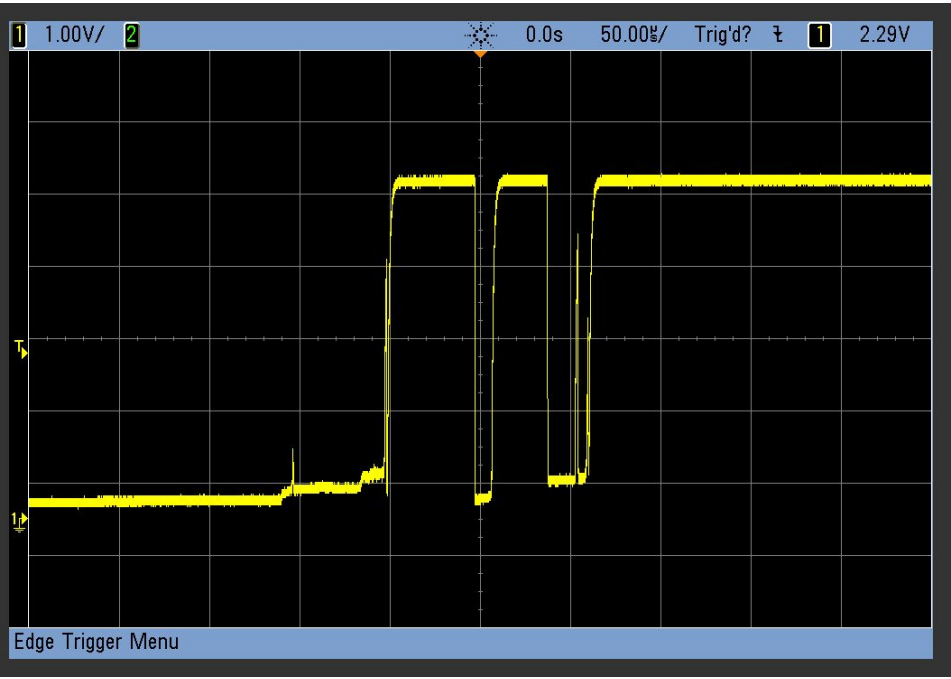


image source:
https://hackaday.com/2015/12/10/embed-with-elliot-debounce-your-noisy-buttons-part-ii/

firmware: Fazit

- ≤0.1ms scan latency
- 0ms debounce latency
- ≤1ms USB poll latency
- → input latency bewegt sich in [0ms, 1.1ms]

- Können wir das weiter reduzieren?

USB

Version	**Geschwindigkeit**	**micro frame**
USB 1.x	Full Speed (FS): 12 Mbit/s	1ms
USB 2.0	High Speed (HS): 480 Mbit/s	125µs
USB 3.0	SuperSpeed (SS): 5.0 Gbit/s	*interrupts*
…	…	…

Teensy 3.6: NXP MK66FX1M0VMD18

- 2 USB ports: USBFS (full speed), USBHS (high speed)

- komplett separate Software stacks
  - HS nutzt neue IP, alte für Kompatibilität präsent

- Teensy nutzt USBFS, HS port optional für host mode
  - Software noch in Kinderschuhen

Teensy 3.6: NXP MK66FX1M0VMD18 (2)

- NXP hat ein SDK mit Beispielen, inklusive USBHS

- Beispiele funktionieren nicht direkt auf dem Teensy:
  - falsche clock config (12 statt 16 MHz crystal)
  - Problem mit dem seriellen setup code

- USBHS hat bei mir nicht auf dem Teensy geklappt :-/

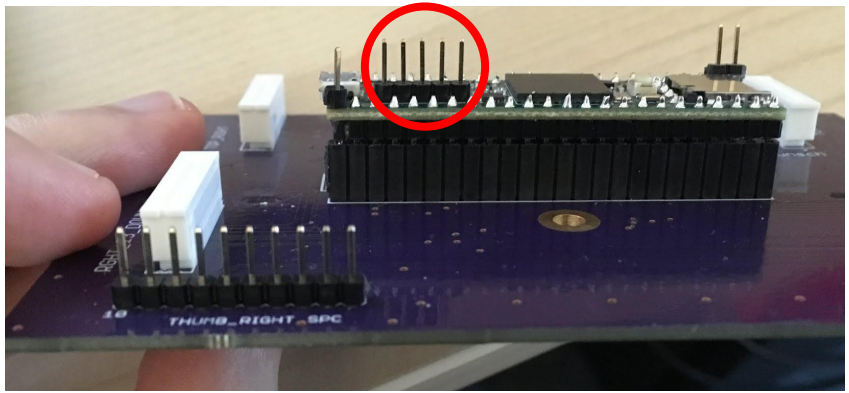
Teensy 3.6: NXP MK66FX1M0VMD18 (3)

- µTasker firmware läuft auf dem Teensy, nutzt USBHS

- aber recht schwergewichtig (Betriebssystem)
  - üblicherweise wohl schon okay,
    aber wir wollten möglichst bare metal bleiben

Teensy 3.6: USBHS port

- USBHS Port ist optional auf dem Teensy
  - man braucht ein Breakout board
  - …aber in der Tastatur ist nicht genug Platz :-(

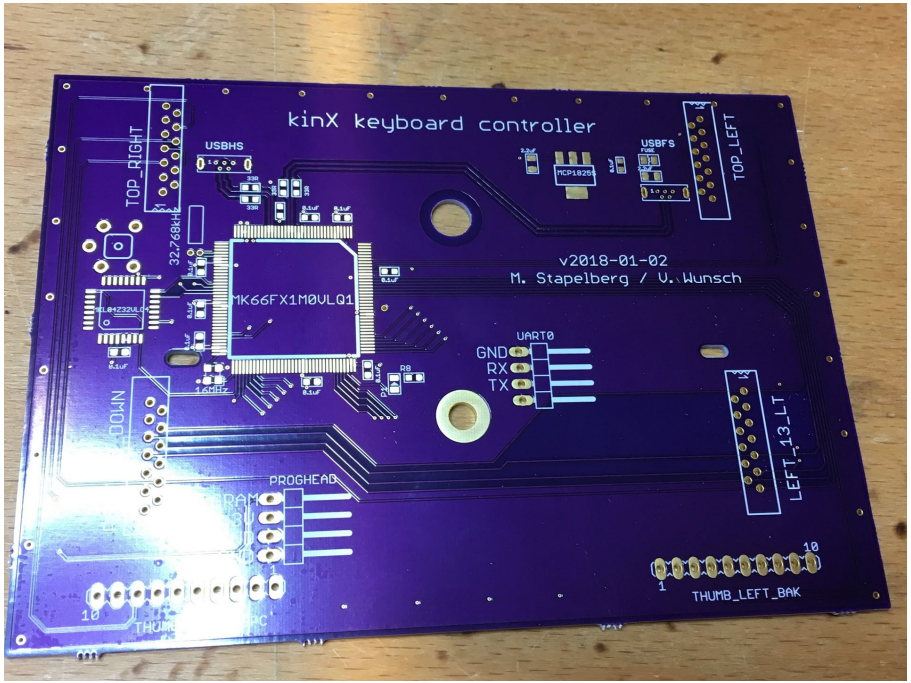


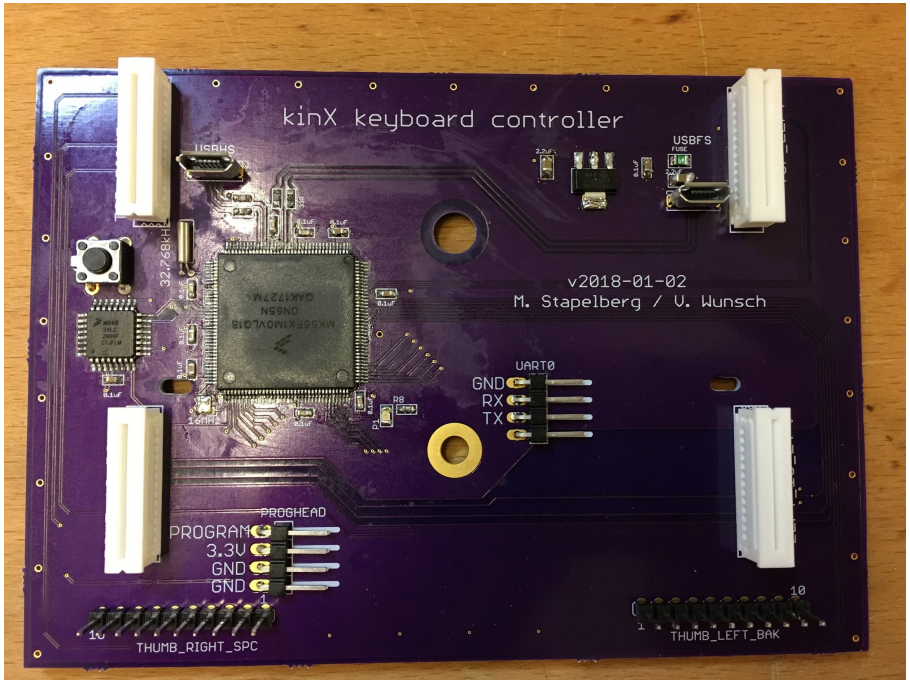
Bauen wir **noch einen** keyboard controller!

- NXP µc nutzen statt gesockelten Teensy

- Teensy bootloader chip für eigene Projekte
  - gleicher Programmier-mechanismus
  - minimale Änderungen zwischen Hardware revs!

Bauen wir **noch einen** keyboard controller!





Entwicklungs-umgebung

- links: usb2serial
- rechts: rebootor

Firmware (k66f)

- basiert auf NXP KSDK

  USBHS läuft auf meiner Hardware

  → müssen uns nicht mehr um bare metal kümmern

- implementiert mit der MCUXpresso IDE

  Erleichtert den Start, aber sonst nicht besonders gut

Firmware (k66f): Fazit

- ≤0.1ms scan latency
- 0ms debounce latency
- ≤0.125ms USB poll latency
- → input latency bewegt sich in [0ms, 0.225ms]

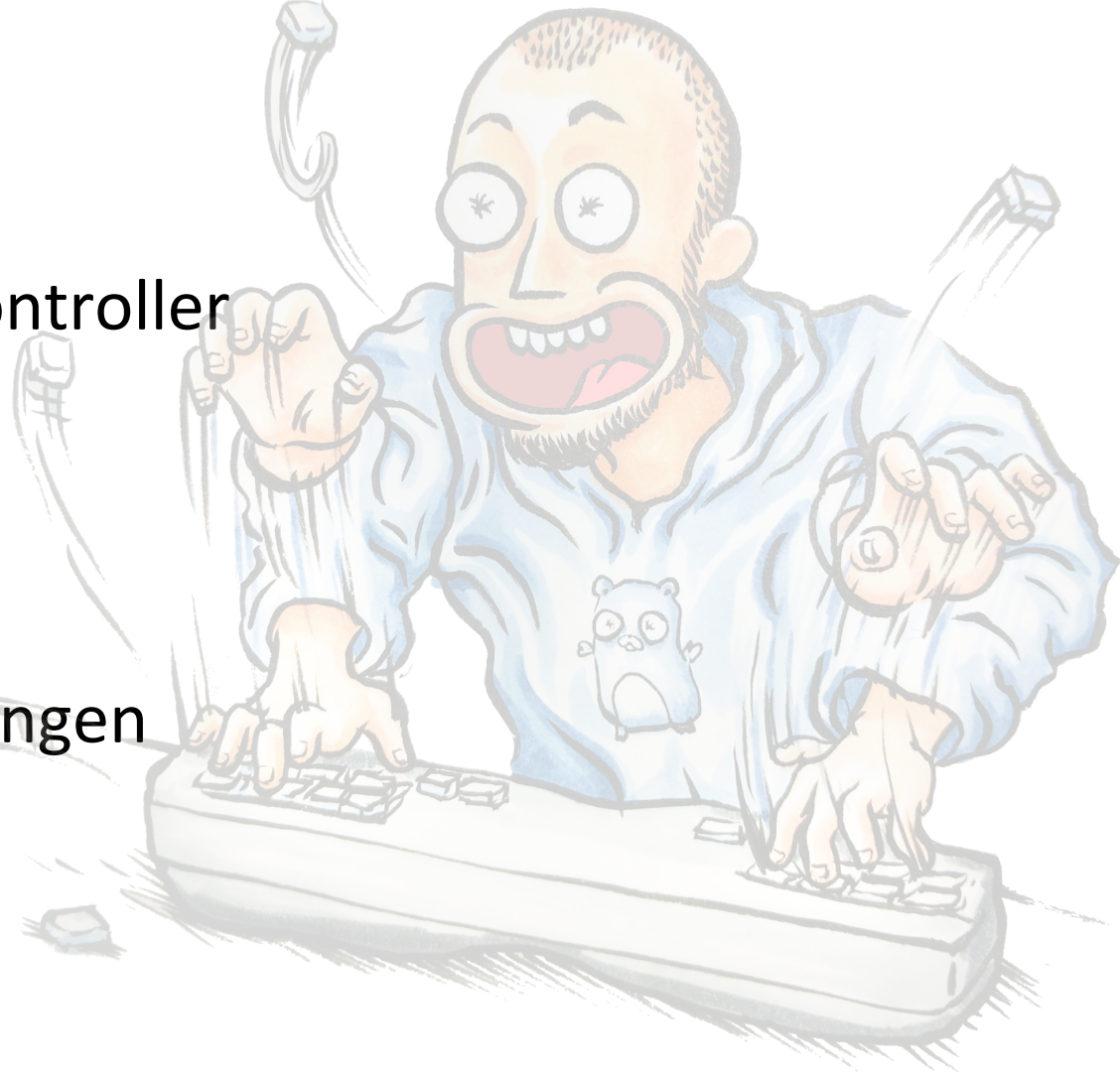
Was haben wir gelernt?

- NXP µc Doku eher spärlich
  bei ARM µc wohl leider normal :-/

- statt mit Teensy mit devboard anfangen (FRDM-K66F)
  → JTAG debugging
  → Hersteller-Beispiele funktionieren direkt

Inhalt

- eigener keyboard controller
- **eigener USB hub**
- input latency Messungen

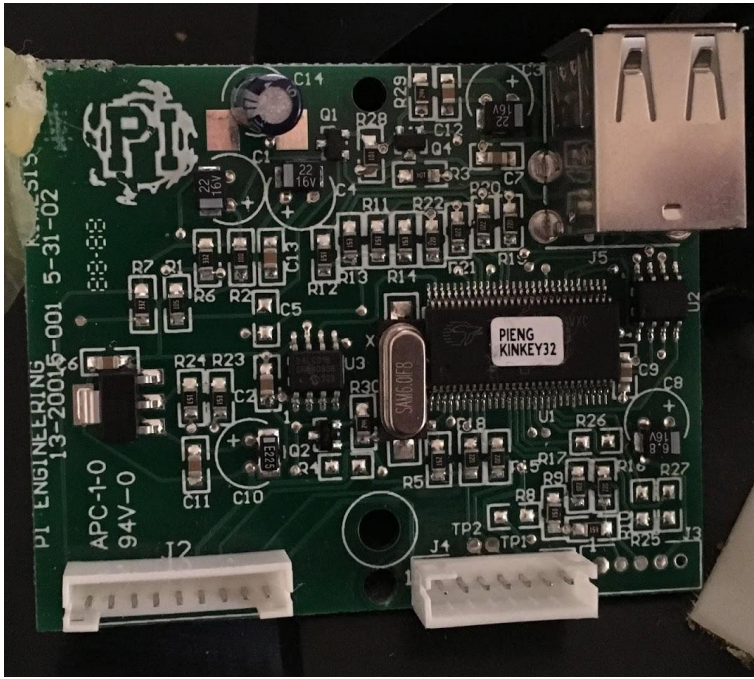


Eigener hub: Motivation

- der controller braucht 2 Kabel

- hub nutzt proprietäre Stecker
  konvertiert USB nach PS/2!

- hub nutzt nur USB 1.1
  und manche Geräte gehen nicht?

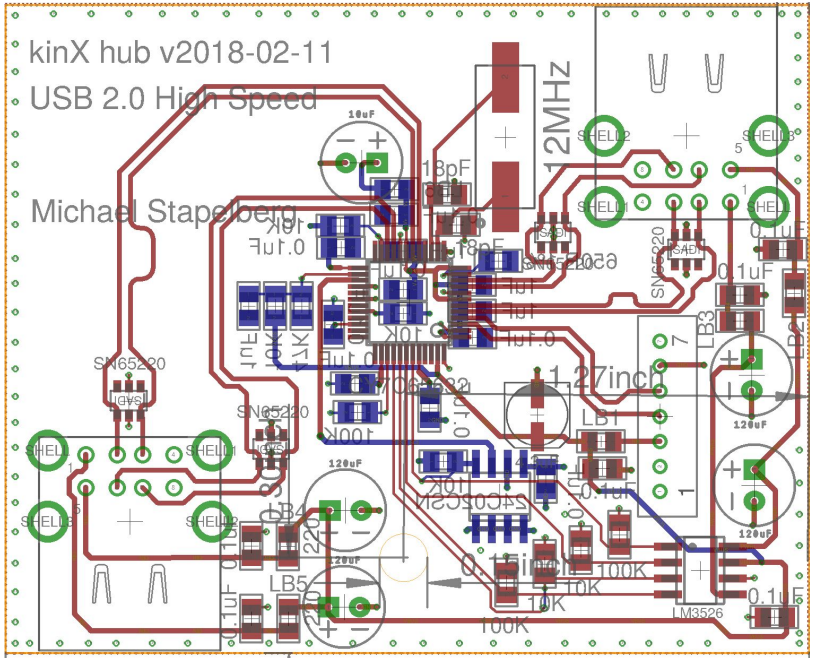


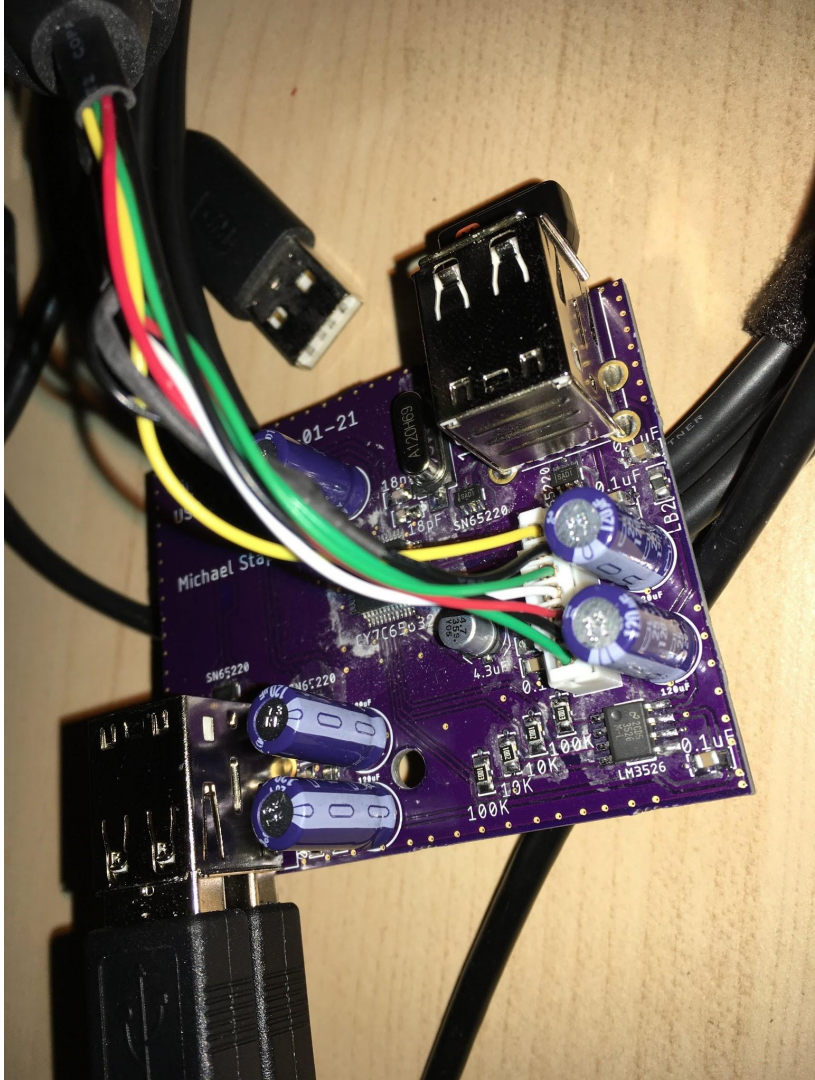
Bauen wir einen USB hub!

- kein freies USB hub Design im Netz gefunden

- USB hub chips von 5 Herstellern verglichen
  → USB2 statt USB3 hat höhere Erfolgschancen
  → Cypress HX2VL hat die beste Doku

Bauen wir einen USB hub!





USB hub: Stromversorgung

- erste revision: bus-powered (eigentlich korrekt!)

- geht mit logitech receiver, yubikey
  Festplatte und USB-Stick weigern sich:
  `kernel: usb 1-14.4.4: rejected 1 configuration due to insufficient available bus power`

USB hub: Stromversorgung (2)

- aber die Geräte gehen an anderen USB hubs?!

- alle gekauften Hubs behaupten, self-powered zu sein!
  → umgeht power calculation im Kernel
  → self-powered lässt 500mA über alle Ports zu
  (bus-powered: 100mA pro Port)

USB hub: EEPROM

- zum Konfigurieren von MaxPower (unnötig)
  (und manufacturer, product und serial)

- programmierbar durch „blaster“-Programm
  nur für Windows, schwer zu installieren
  Linux-Version für Cypress USB3, nicht die älteren USB2

USB hub: EEPROM (2)

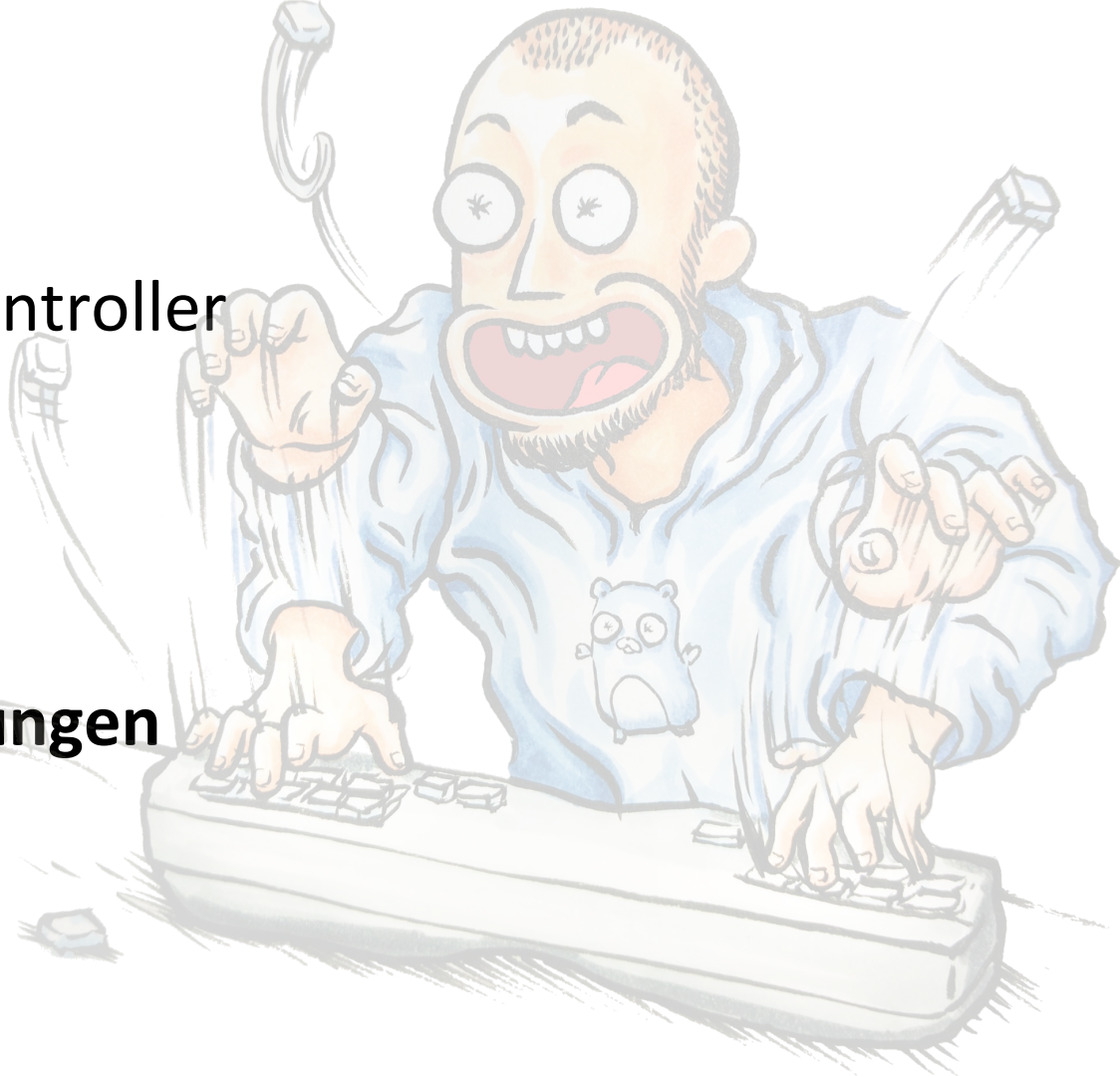
```
dev, _ := usb.OpenDeviceWithVIDPID(0x04b4, 0x6570)

eepromRequest := 14

wIndex := 0 // [0, 63] for 128 bytes of EEPROM

dev.Control(gousb.RequestTypeVendor|0x80,
  eepromRequest, 0, wIndex, make([]byte, 2))
```

USB hub: Stromversorgung (3)

- vorgeben, dass wir self-powered sind
  → USB-Stick funktioniert zuverlässig
  → HDD geht (wenn sie erstes/einziges Gerät ist)

Inhalt

- eigener keyboard controller
- eigener USB hub
- **input latency Messungen**



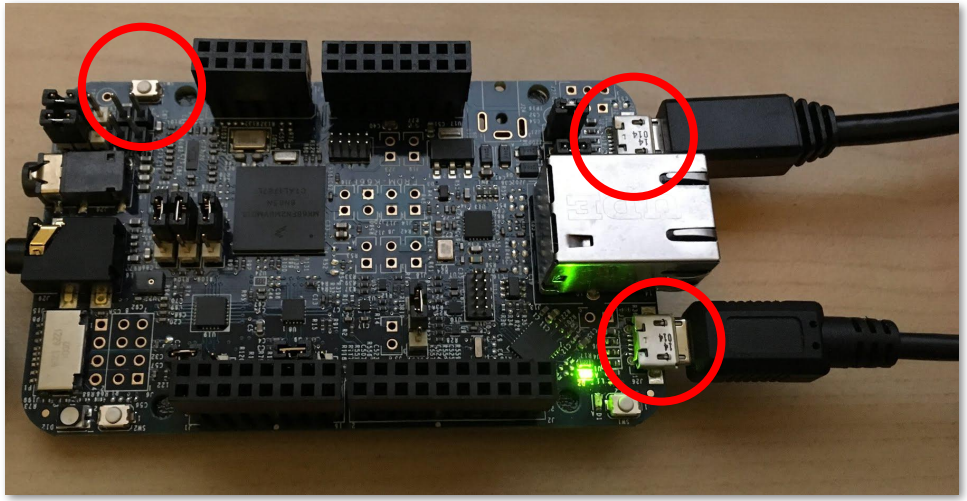
Messgerät

- FRDM-K66F eval board
  (≈ 60 USD)

- Mess-Firmware
  (fast wie keyboard FW)

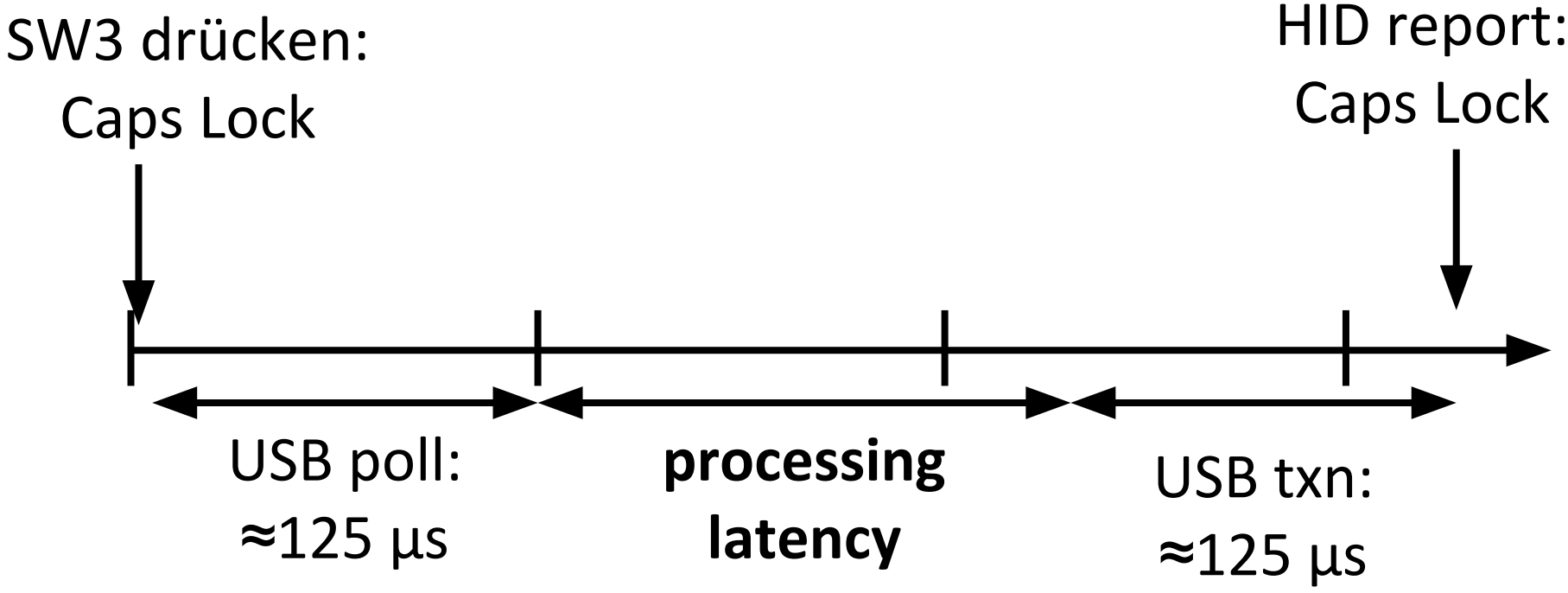
SW3

USB
High Speed



Debug

Messmethode (Gerät)



Messmethode (host)

- baseline messen:

  USB host controller

  Linux kernel (usb, input)

  input event API (evdev)

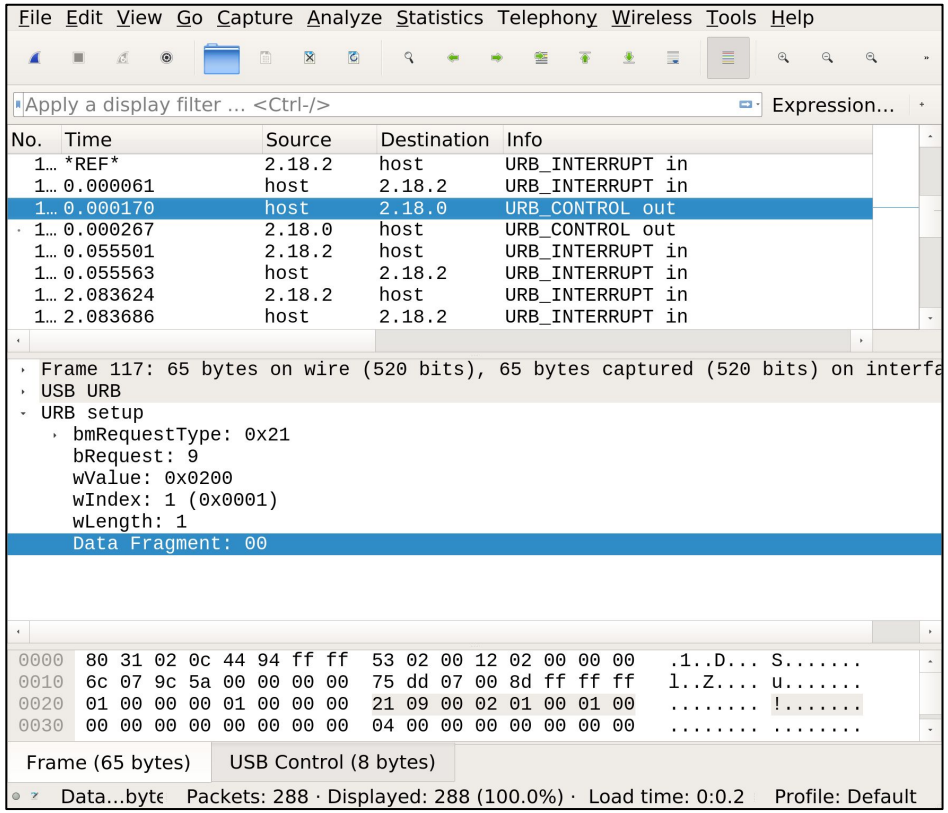
- ohne X11/Wayland

```c
#include <linux/input.h>

int main(int argc, char *argv[]) {
  if (getuid() != 0) {
    errx(EXIT_FAILURE, "run as root");
  }
  if (argc != 2) {
    errx(EXIT_FAILURE, "syntax: %s /dev/input/eventN", argv[0]);
  }
  int fd = open(argv[1], O_RDWR);
  if (fd < 0) {
    err(EXIT_FAILURE, "open(%s)", argv[1]);
  }

  if (ioctl(fd, EVIOCGRAB, 1) < 0) {
    err(EXIT_FAILURE, "Grabbing evdev keyboard device failed");
  }

  struct input_event ev;
  ssize_t size;
  struct input_event on = {
    .type = EV_LED,
    .value = MSC_PULSELED,
    .code = LED_CAPSL,
  };
  struct input_event off = {
    .type = EV_LED,
    .value = !MSC_PULSELED,
    .code = LED_CAPSL,
  };
  while (true) {
    size = read(fd, &ev, sizeof(struct input_event));
    if (size < sizeof(struct input_event)) {
      errx(EXIT_FAILURE, "short read: got %" PRIi32 ", want %" PRIu32, (int32_t)size, (ui
nt32_t)sizeof(struct input_event));
    }
    if (ev.type != EV_KEY) {
      continue; // not EV_KEY
    }
    if (ev.code != 58 /* MSG_CAPSLOCK */) {
      continue; // not caps lock
    }
    if (ev.value != 1) {
      continue; // depressed
    }
    // turn on LED when caps lock gets pressed
    write(fd, &on, sizeof(struct input_event));
    printf("MSG_CAPSLOCK processed\n");
    write(fd, &off, sizeof(struct input_event));
  }
}
```

```
-:---  measure-evdev.c   Bot (16,0)     (C/l EditorConfig Abbrev)
```

processing latency

- wireshark usbmon (ohne USB poll/txn)
- processing latency: im Mittel ≈ 152 µs (Ubuntu 17.10)



Emacs processing latency

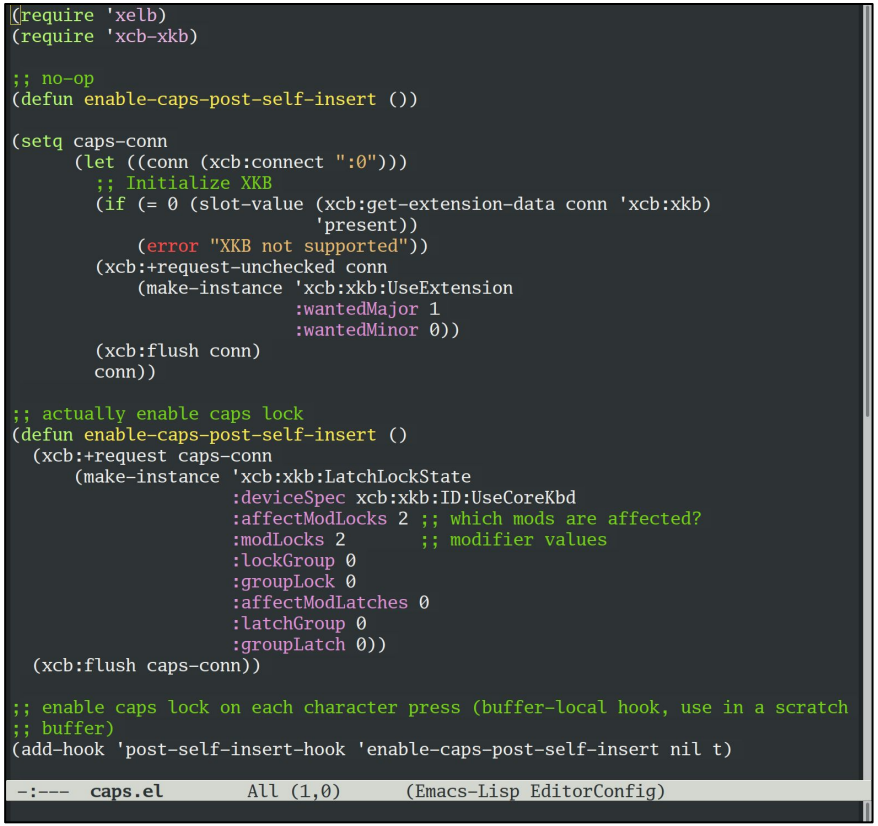
- Emacs latency:

  im Mittel ≈ 278 µs

  (Ubuntu 17.10)

  (Emacs 25.2.2)

  (excludes baseline)



```
(require 'xelb)
(require 'xcb-xkb)

;; no-op
(defun enable-caps-post-self-insert ())

(setq caps-conn
      (let ((conn (xcb:connect ":0")))
        ;; Initialize XKB
        (if (= 0 (slot-value (xcb:get-extension-data conn 'xcb:xkb)
                             'present))
            (error "XKB not supported"))
        (xcb:+request-unchecked conn
            (make-instance 'xcb:xkb:UseExtension
                           :wantedMajor 1
                           :wantedMinor 0))
        (xcb:flush conn)
        conn))

;; actually enable caps lock
(defun enable-caps-post-self-insert ()
  (xcb:+request caps-conn
      (make-instance 'xcb:xkb:LatchLockState
                     :deviceSpec xcb:xkb:ID:UseCoreKbd
                     :affectModLocks 2 ;; which mods are affected?
                     :modLocks 2       ;; modifier values
                     :lockGroup 0
                     :groupLock 0
                     :affectModLatches 0
                     :latchGroup 0
                     :groupLatch 0))
  (xcb:flush caps-conn))

;; enable caps lock on each character press (buffer-local hook, use in a scratch
;; buffer)
(add-hook 'post-self-insert-hook 'enable-caps-post-self-insert nil t)
```

```
-:---  caps.el        All (1,0)      (Emacs-Lisp EditorConfig)
```

Ende-zu-Ende input latency

Schritt	**latency**
matrix scan	≈ 100 µs
USB poll	≈ 125 µs
Linux	≈ 152 µs
Emacs	≈ 278 µs

→ total ≈ 655 µs

+ output latency ≥ [0, 16ms] (at 60 Hz)

Das latency-Spiel

- wie sehr nehmen Menschen input latency wahr?

- Spiel stellt zusätzliche Latenz in der Tastatur ein, du sagst ob du Latenz wahrnimmst

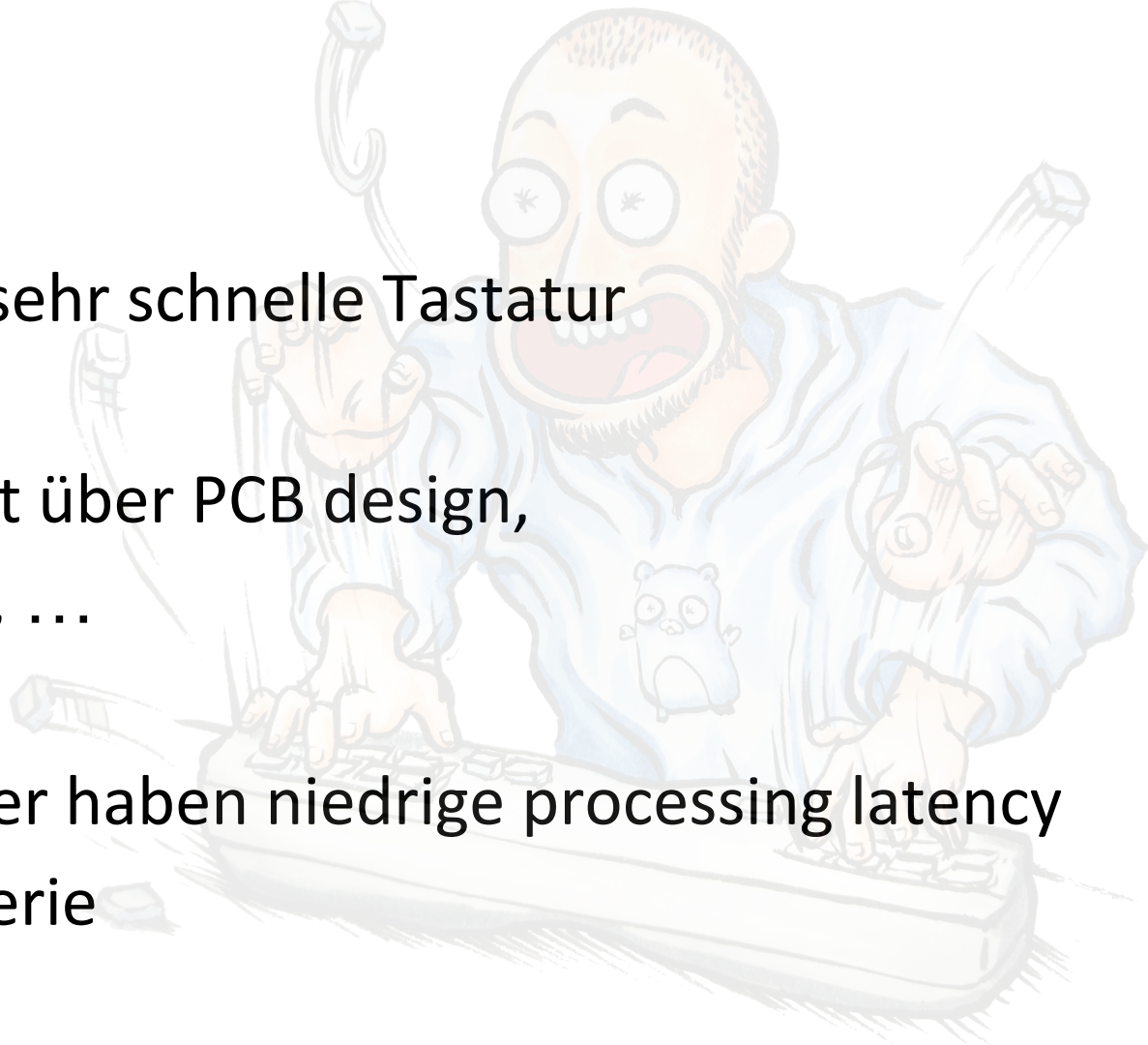
- höhere Levels haben niedrigere Zusatz-Latenz

Das latency-Spiel (2)

- 17 Spieler auf dem 34C3
- bester Spieler konnte 15ms zuverlässig unterscheiden
- einige Spieler konnten 75ms nicht unterscheiden

Fazit

- Ich hab jetzt eine sehr schnelle Tastatur
- Ich hab viel gelernt über PCB design,
  ARM µc, USB, HID, …
- Moderne Computer haben niedrige processing latency
  nutze gute Peripherie



Ende

Fragen?

Feedback geben