fablab Karlsruhe

Henne oder Ei?

„Bare Metal“ Programmierung von Mikrocontrollern

Kleine Technikgeschichte in Stichworten I

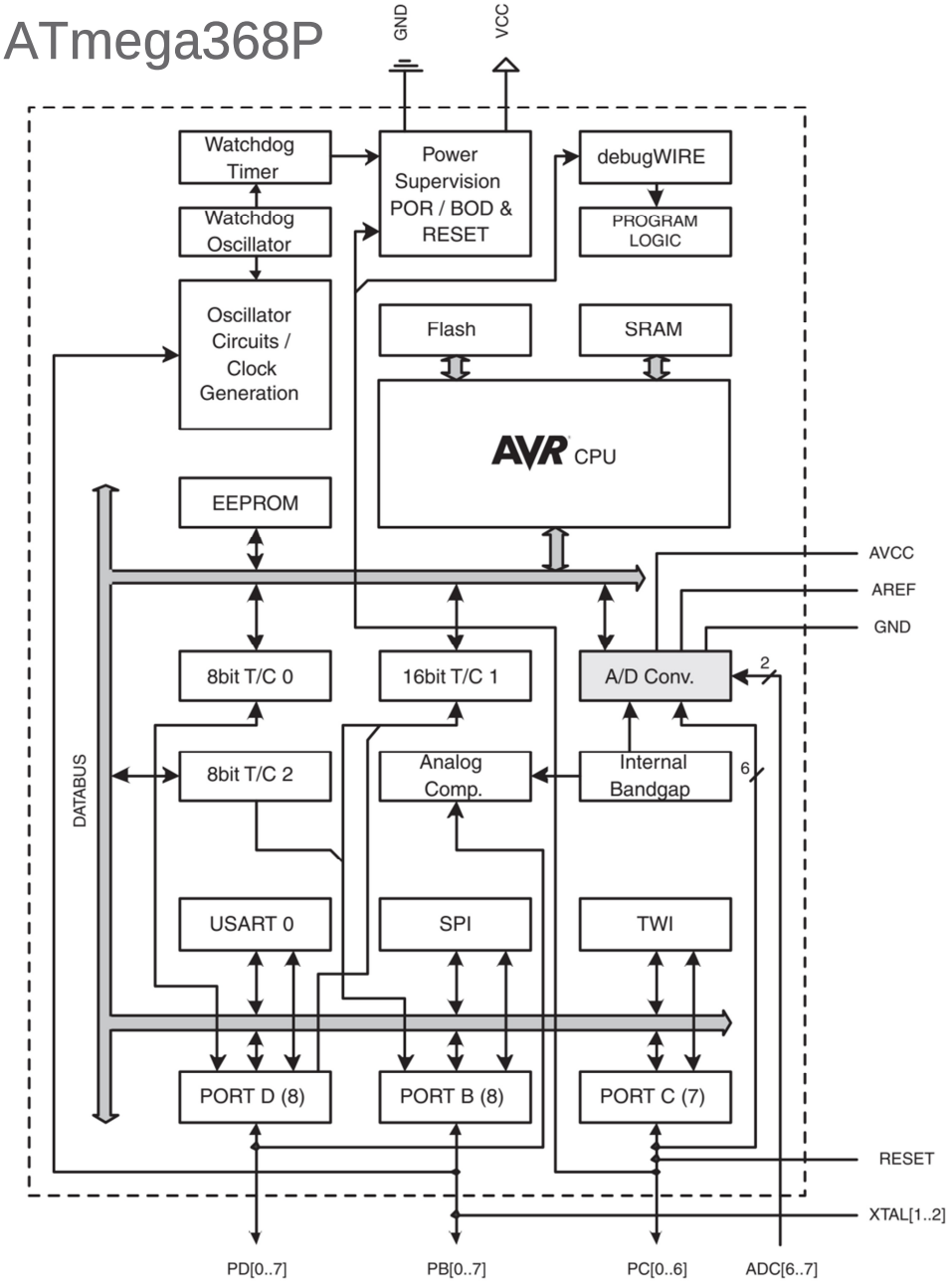
- Erfindung Anfang der 1970er Jahre; Texas Instruments (TI), Zilog und Intel
- zweite Hälfte der 70er Jahre: elektronische Taschenrechner werden Massenartikel
- zunächst als 4 bit und 8 bit Recheneinheiten, wenig später 16 bit und 32 bit aktuell 64 bit
- Spezialisierung:
  - Recheneinheit plus Kommunikations-/Steuermodule: Mikrocontroller
  - reine Recheneinheit: Mikroprozessor
  - immer mehr fließende Grenzen: MC, CPU, GPU, SoC
- Programmierung zunächst sehr geschlossen
  - Assembler, C mit herstellerabhängigen Erweiterungen und Anpassungen
  - teure, individuelle Programmiergeräte (EPROM, EEPROM), herstellereigene Standards
  - Entwicklungsumgebungen Teil des Geschäftsmodells, teils mit NDA

Kleine Technikgeschichte in Stichworten II

- Öffnung der Entwicklungsmöglichkeiten:
  - Internet ist Katalysator für schnelle Entwicklungen
  - freie Compiler und Assembler werden an zahlreiche MC angepasst
  - Entwicklung von Abstraktionsbibliotheken
  - beim ARM-Geschäftsmodell: Befehlssatz kein Unterscheidungskriterium
  - Ablauf zahlreicher Patente
  - mehrstufige Verwertungskette: z.B. 8-bit Technologie 40+ Jahre alt
- Die Rolle von Arduino
  - kombiniert geschickt zahlreiche Open Source Werkzeuge
  - verbirgt zahlreiche Schwierigkeiten herkömmlicher MC Programmierung
  - erreicht damit breite Massen
  - Vorbild für weitere Projekte, auch durch entstehende Polarisierung

Bestandteile eines Microcontrollers

am Beispiel ATmega368P



AVR Mikrocontroller-Programmierung

- Individualisierung über Fuses
- integriertes Flash
- integriertes EEPROM
- ISP/HVP/Debugwire
- Software-Bootloader
  - seriell
  - USB per USB-Seriell Adapter
  - USB Simulation per V-USB (Micronucleus)
- Datenblatt ist wichtige Informationsquelle:
  https://www.microchip.com/wwwproducts/en/ATtiny85
  https://www.microchip.com/wwwproducts/en/ATmega328p

Bootloader pro/contra

- Nur USB-Kabel bzw. USB-Seriell Adapter notwendig
- Kein Zugriff auf Fuses/kein Aussperren möglich
- Weniger freier Programmspeicher
- defekter Bootloader durch Powerglitches etc. macht MC unbrauchbar
- Programmierschnittstelle gleichzeitig auch Monitoringmöglichkeit
- V-USB: zwei Pins mit Zenerdiode, einer davon mit Pull-Up Widerstand
- V-USB: Möglichkeit HID Geräte zu simulieren
- V-USB: höherer Speicherbedarf
- V-USB/Micronucleus: weitere Programmierbarkeit bei deaktiviertem RESET
- manche MC haben Hardware-Bootloader (z.B. AT-SAM3X8E in Arduino Due)

Arduino IDE vorbereiten

Arduino Datei->Voreinstellungen

- Ausführliche Ausgabe während Kompilierung und Hochladen
- Alle Compiler Warnungen
- Zusätzliche Boardverwalter URLs
  - http://digistump.com/package_digistump_index.json
  - http://drazzy.com/package_drazzy.com_index.json
- Im Boardverwalter die folgenden Boards installieren:
  - Digistump AVR Boards installieren
  - Attiny Core

Software vorbereiten I

LIB-USB Treiber installieren (Windows Only)

- https://zadig.akeo.ie/
- VID/PID: 16D0/0753 mit libusb-win32 assoziieren

Micronucleus *command line tool* und *upgrade-t85_default.hex* herunterladen

- https://github.com/micronucleus/micronucleus

Littlewire Hexfile herunterladen (USBtinyISP Firmware)

- http://littlewire.github.io/resources/littlewire_v13.hex

Treiber für USBtinyISP installieren

- https://learn.adafruit.com/usbtinyisp/drivers

Software vorbereiten II

Version des vorhandenen Micronucleus feststellen

- micronucleus ?

Falls Version < 2.3, updaten:

- micronucleus path/to/upgrade-t85_default.hex

boards.txt anpassen für Digispark anpassen:

- digispark-tiny.upload.maximum_size=6522 (bei micronucleus 2.3)

Littlewire ISP Software installieren:

- micronucleus.exe littlewire_v13.hex

Kontrolle im Gerätemanager:

- libusb-win32-Gerät USBtinyISP (VID/PID: 1781/0104)

Attiny85 Fuses

Logik der Fuses ist invertiert, da gelöschte Flashbits auf „1“ stehen.
Achtung: einige Fuse-Rechner im Internet stellen die Werte invertiert dar!

Extendend Byte		
Bit		Bedeutung
7	1	
6	1	
5	1	
4	1	
3	1	
2	1	
1	1	
0	1	self programming

High Byte		
Bit		Bedeutung
7	1	reset disabled
6	1	debugwire enabled
5	0	SPI prog. enabled
4	1	watchdog enabled
3	1	EESAVE
2	1	BOD Level 2
1	1	BOD Level 1
0	1	BOD Level 0

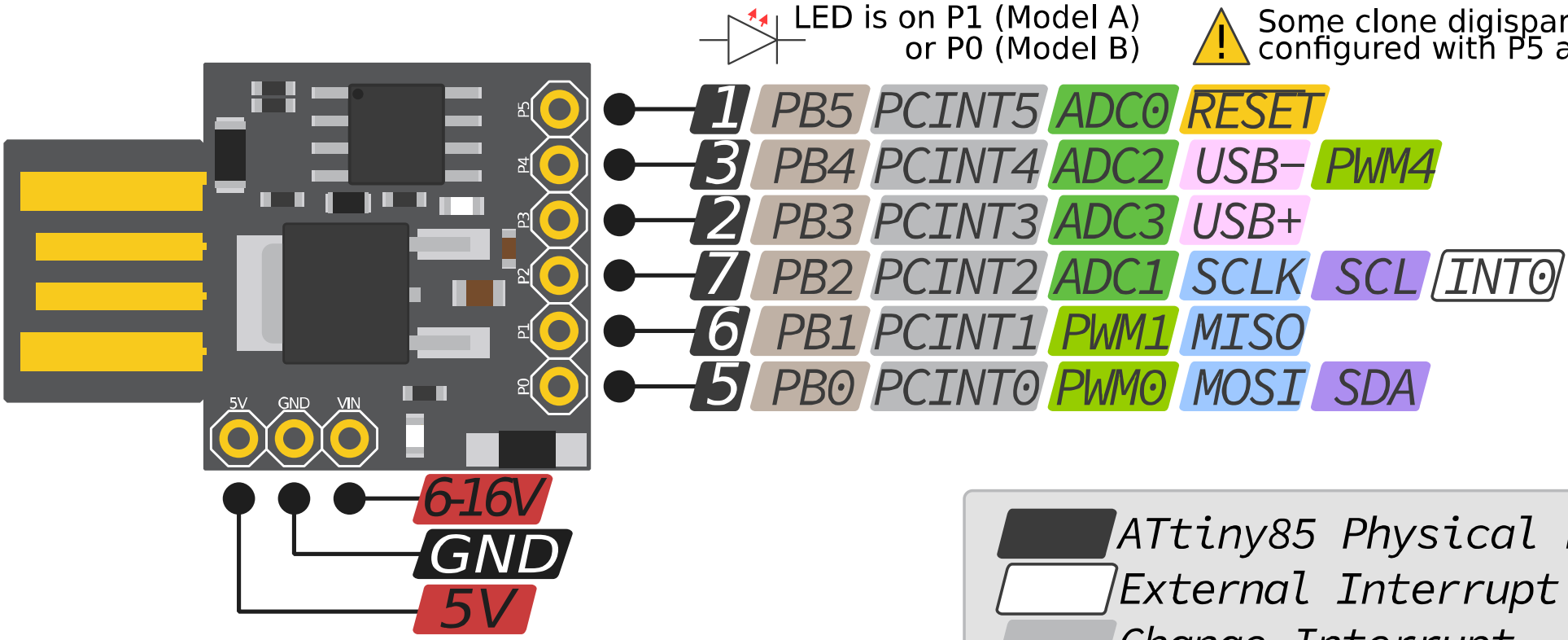
Low Byte		
Bit		Bedeutung
7	0	CLK DIV 8
6	1	Clock Out ENA
5	1	StartUp time 1
4	0	StartUp time 0
3	0	CKSEL 3
2	0	CKSEL 2
1	1	CKSEL 1
0	0	CKSEL 0

ROT: Controller kann nur noch per HVP programmiert werden, wenn geändert!

GRÜN: Programmieren über Bootloader nicht mehr möglich

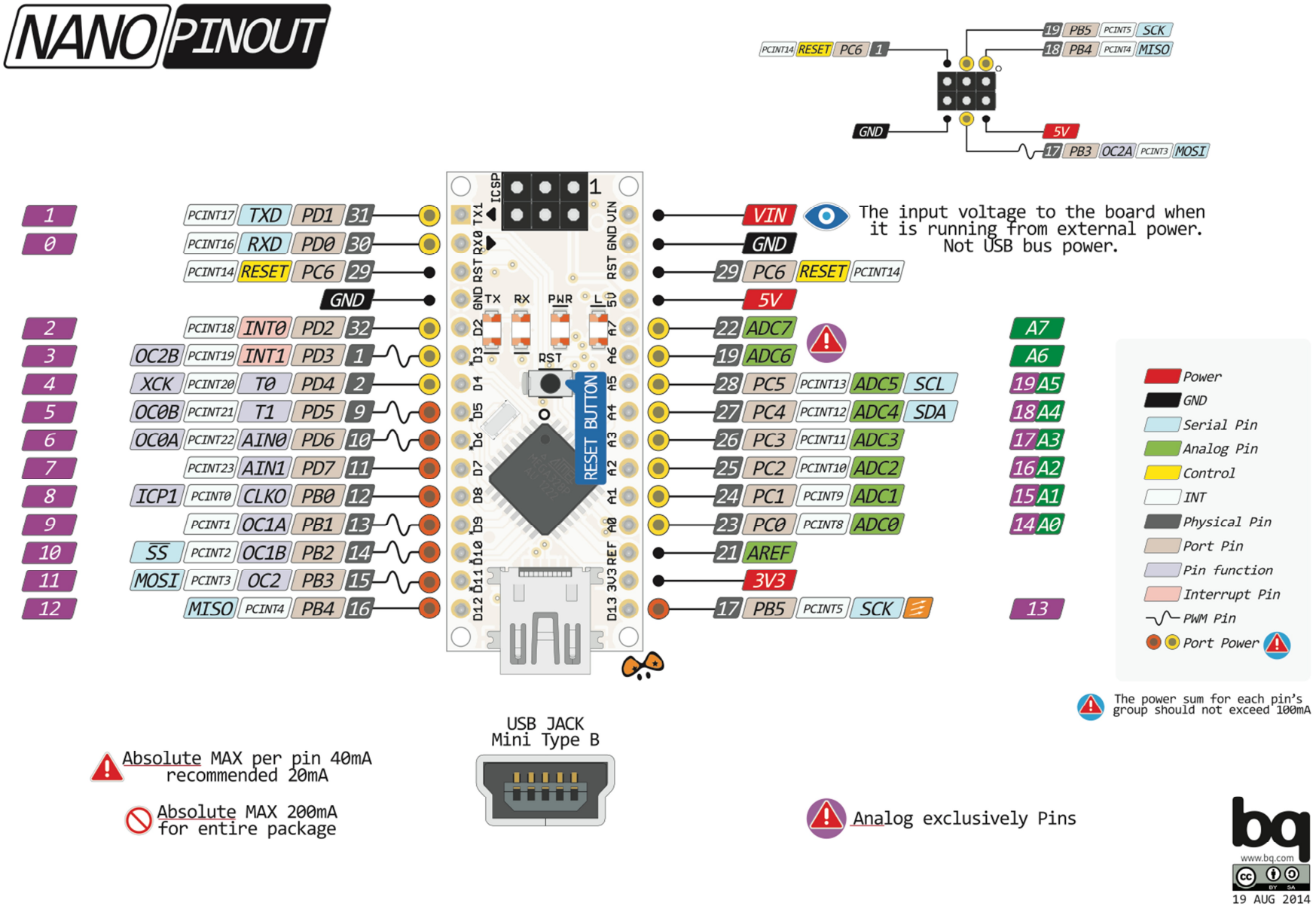
GELB: Controller läuft nicht an, wenn falsch gesetzt

Digispark PinOut



ATtiny85 Physical Pin
External Interrupt
Change Interrupt
Analog Read/Write
Port Pin
SPI
I^2C
V-USB; Zener-Diodes limit to 3.3V
P3 additional Pull-Up

Arduino Nano Pinout



ATMEGA328P Fuses

Logik der Fuses ist invertiert, da gelöschte Flashbits auf „1“ stehen.
Achtung: einige Fuse-Rechner im Internet stellen die Werte invertiert dar!

Extendend Byte		
Bit		Bedeutung
7	1	
6	1	
5	1	
4	1	
3	1	
2	1	BOD Level 2
1	1	BOD Level 1
0	1	BOD Level 0

High Byte		
Bit		Bedeutung
7	1	reset disabled
6	1	debugwire enabled
5	0	SPI prog. enabled
4	1	watchdog enabled
3	1	EESAVE
2	1	BootSize 1
1	1	BootSize 0
0	1	Select Reset Vector

Low Byte		
Bit		Bedeutung
7	0	CLK DIV 8
6	1	Clock Out ENA
5	1	StartUp time 1
4	0	StartUp time 0
3	0	CKSEL 3
2	0	CKSEL 2
1	1	CKSEL 1
0	0	CKSEL 0

ROT: Controller kann nur noch per HVP programmiert werden, wenn geändert!

GELB: Controller läuft nicht an, wenn falsch gesetzt

BLAU: Beim 328P und nahen Verwandten bleibt Reset zum Start aktiv und wird erst nach der Initialisierung abgeschaltet.

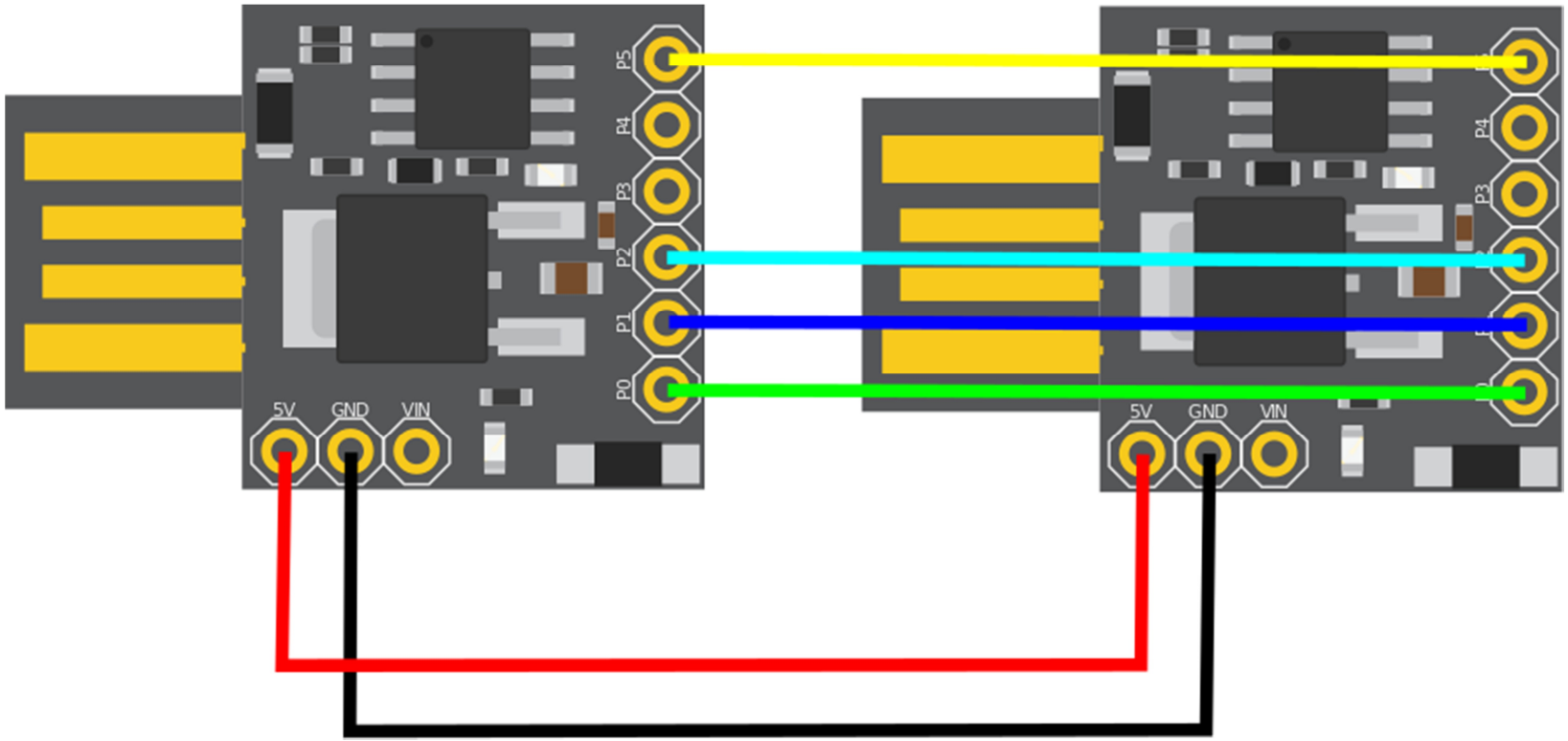
Digispark zu ISP-Adapter machen

Wir benötigen 4 Pins zum Programmieren des Ziel-Mikrocontrollers:
MISO, MOSI, SCK und RESET,
weitere 2 Pins für die USB-Kommunikation mit dem PC

Derzeit nur 5 Pins verfügbar, da die verwendeten Digisparks den Reset-Pin nicht deaktiviert haben, daher muss zunächst ein schon entsprechend programmierter Digispark verwendet werden:

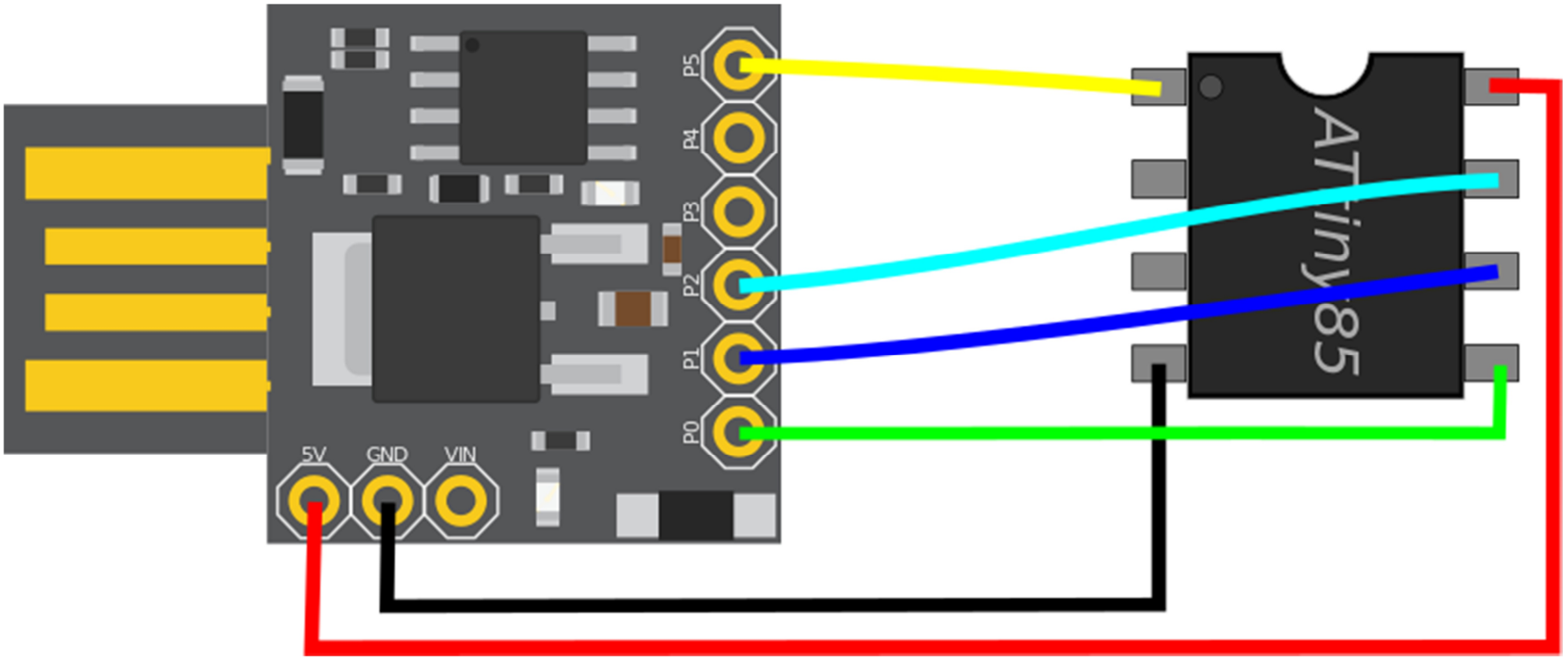
- Kontrolle, dass Kommunikation klappt und Zielcontroller erkannt wird:
  ```
  .\bin\avrdude -c usbtiny -C .\etc\avrdude.conf -p t85
  ```
- Fuses setzen, jedoch external RESET noch nicht deaktivieren
  ```
  .\bin\avrdude -c usbtiny -C .\etc\avrdude.conf -p t85 \
  -U lfuse:w:0xf1:m -U hfuse:w:0xdf:m -U efuse:w:0xfe:m
  ```
- Fuses setzen, RESET deaktivieren
  ```
  .\bin\avrdude -c usbtiny -C .\etc\avrdude.conf -p t85 \
  -U lfuse:w:0xf1:m -U hfuse:w:0x5f:m -U efuse:w:0xfe:m
  ```

Verkabelung USBtinyISP -> Digispark (USBtinyISP)



MOSI und SCKL-Leitungen im Zweifel mit Widerstand 1kΩ versehen, falls auf dem zu programmierenden Controller (Ziel) diese Pins als Output gesetzt sind!

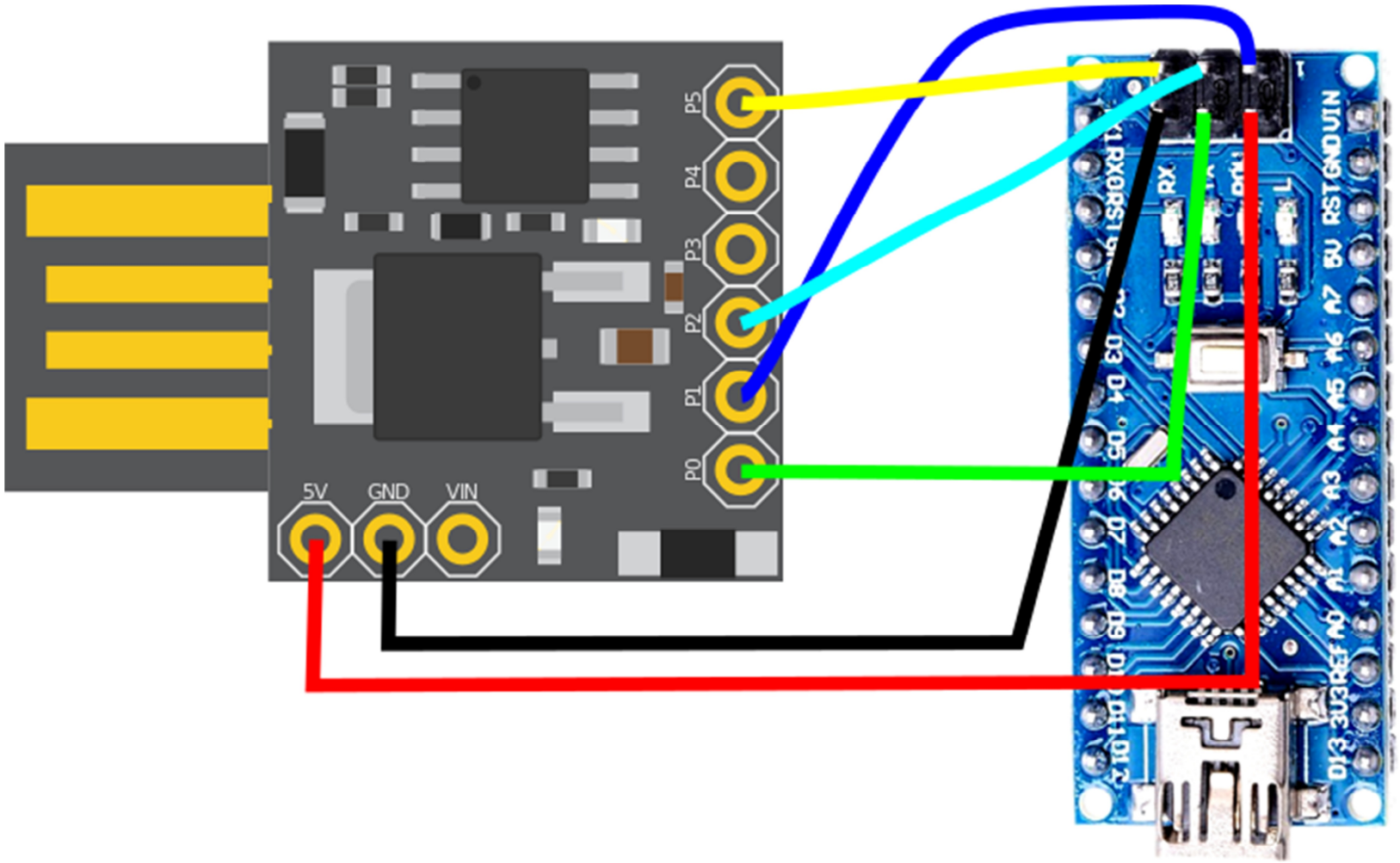
Verkabelung USBtinyISP -> Attiny85



MOSI und SCKL-Leitungen im Zweifel mit Widerstand 1kΩ versehen, falls auf dem zu programmierenden Controller (Ziel) diese Pins als Output gesetzt sind!

Nackte Zielcontroller an Spannungsversorgung mit Kondensator stützen!

Verkabelung USBtinyISP -> Arduino Nano



MOSI und SCKL-Leitungen im Zweifel mit Widerstand 1kΩ versehen, falls auf dem zu programmierenden Controller (Ziel) diese Pins als Output gesetzt sind!

Programmieren eines Attiny85 mit der Arduino-IDE

- Werkzeuge Board: Attiny25/45/85
- Clock: 8/16 MHz internal (8MHz für Spannungen < 4V)
- Chip: Attiny85
- B.O.D. Level: je nach Projekt
- Link Time Optimization (LTO): disabled
- Port: Einmal auf „serielle Ports“ clicken, auch wenn sich nichts tut! *)
- Programmer: USBtinyISP auswählen
- Bootloader brennen (brennt keinen Bootloader, aber setzt FUSES!)
- Sketch hochladen

*) Die Arduino-IDE wählt einen nicht vorhandenen COM-Port zum Hochladen, wenn dieser nicht deselektiert wird. Der Programmer wird nur dann gewählt, wenn kein COM-Port aktiv ist.

Arduino Nano: Flash um 1.5 kByte vergrößen I

Die neuesten Nanos haben einen Bootloader, der nur noch 512 Byte, statt knapp 2kByte Flash benötigt.
https://www.heise.de/make/artikel/Arduino-Nano-mit-neuem-Bootloader-4011641.html

Jedoch hat Arduino vergessen, die Fuses anzupassen, so dass das Speicherlimit nicht vergrößert wurde. Wir können dies jedoch einfach selber erledigen:
In der Datei
arduino-1.x.x\hardware\arduino\avr\boards.txt
einfach die folgende Rubrik bei den Nano-Definitionen eintragen:

```
## Arduino Nano w/ ATmega328P (optimized Bootloader)
## ------------------------
nano.menu.cpu.atmega328opti=ATmega328P (optimized Bootloader)
nano.menu.cpu.atmega328opti.upload.maximum_size=32256
nano.menu.cpu.atmega328opti.upload.maximum_data_size=2048
nano.menu.cpu.atmega328opti.upload.speed=115200
nano.menu.cpu.atmega328opti.bootloader.low_fuses=0xFF
nano.menu.cpu.atmega328opti.bootloader.high_fuses=0xDE
nano.menu.cpu.atmega328opti.bootloader.extended_fuses=0xFD
nano.menu.cpu.atmega328opti.bootloader.file=optiboot/optiboot_atmega328.hex
nano.menu.cpu.atmega328opti.build.mcu=atmega328p
```

Arduino Nano: Flash um 1.5 kByte vergrößen II

- Änderung bei geschlossener Arduino-IDE durchführen
- Nano an den USBtinyISP anschliessen
- IDE neu starten
  - Board: Arduino Nano
  - Prozessor: Atmega 328P (optimized Bootloader)
  - Programmer: USBtinyISP
- Bootloader brennen
- Nun kann dieser Nano mit 32256 Bytes anstelle mit 30720 Bytes geflasht werden!