Einführung in Mikrocontrollertechnik

Am Beispiel des Atmel tiny84

Prof. Dr. Manfred Schimmler
M.Sc. Jan Christian Kässens

23. April 2014

Inhalt

Inhalt

Inhalt

Kapitelübersicht

Was ist ein Mikrocontroller?

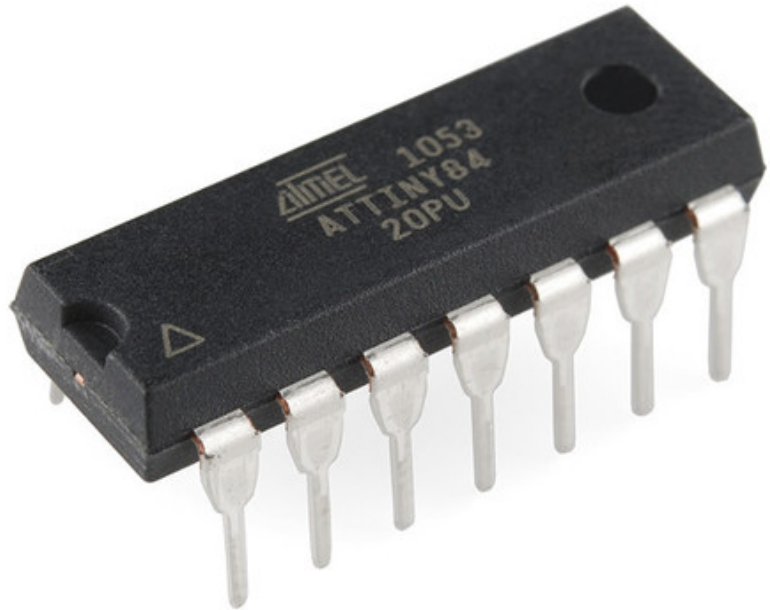
Ein Mikrocontroller ist ein Ein-Chip-Computersystem.

– Wikipedia

Was ist ein Mikrocontroller?

Vergleich mit Mikroprozessoren

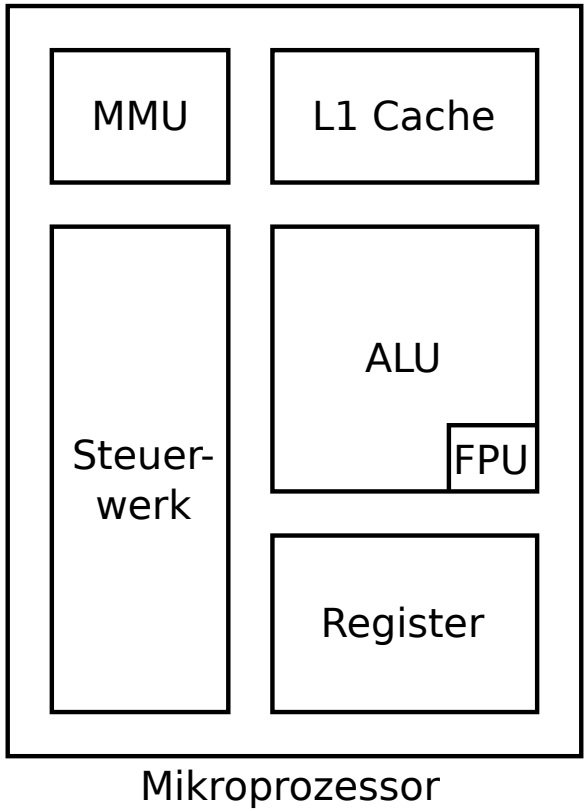
Intel Core 7 Extreme 975



Atmel ATtiny84

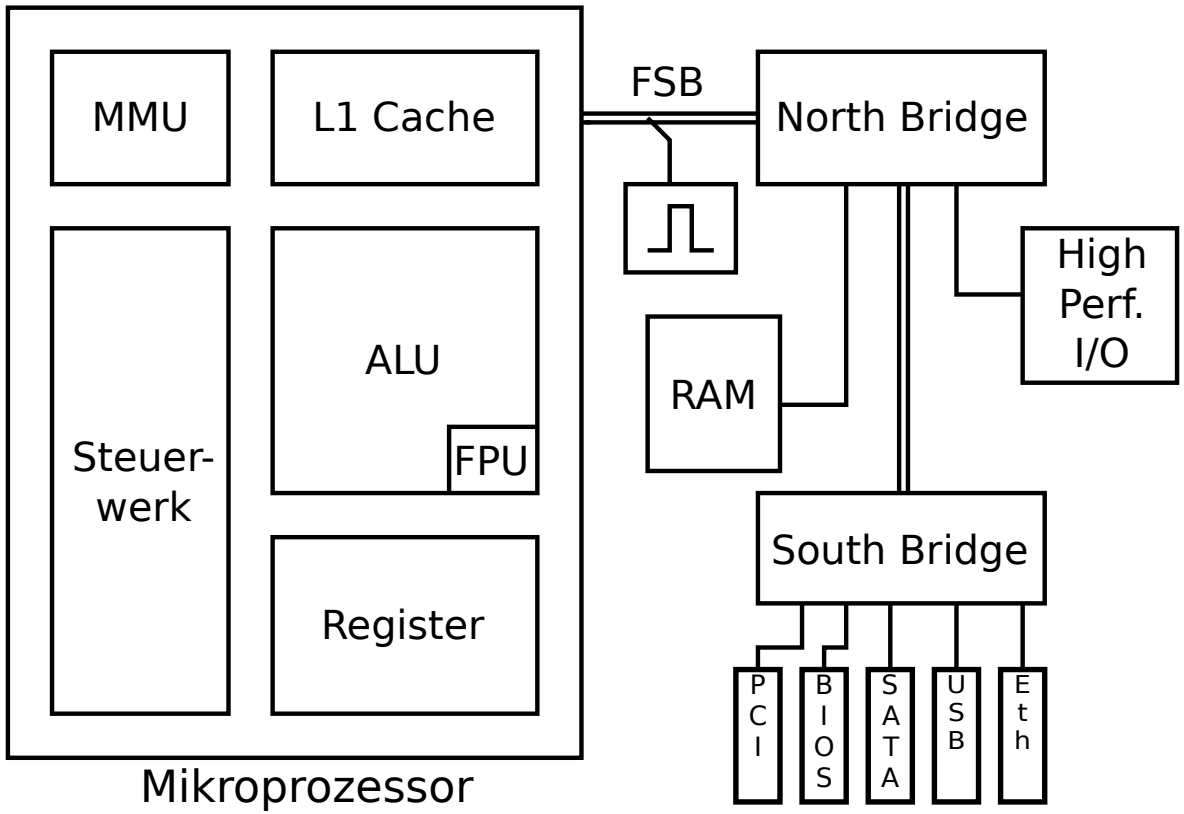
Was ist ein Mikrocontroller?

Vergleich mit Mikroprozessoren – Architektur



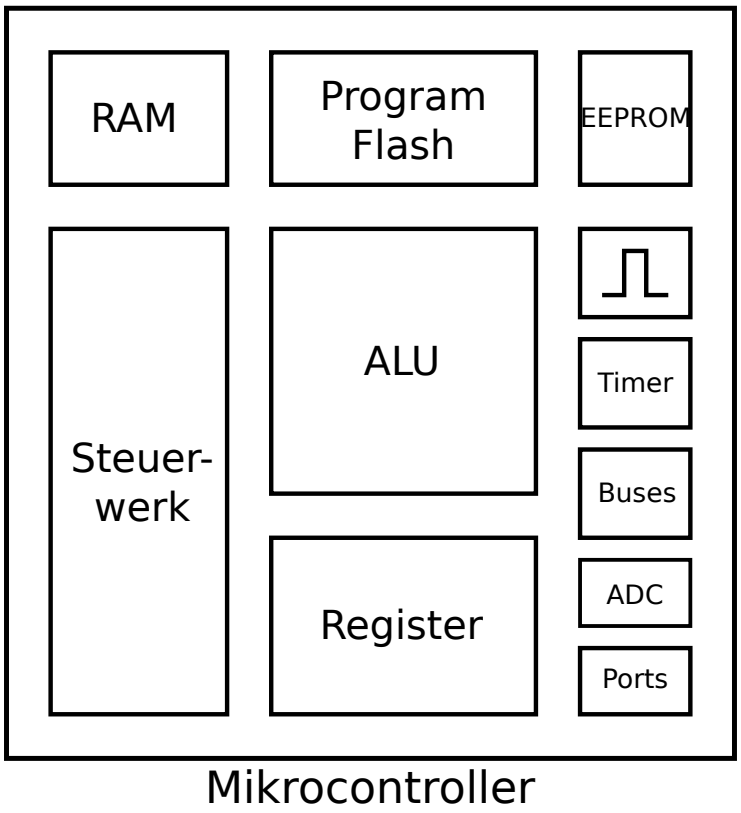
Was ist ein Mikrocontroller?

Vergleich mit Mikroprozessoren – Architektur



Was ist ein Mikrocontroller?

Vergleich mit Mikroprozessoren – Architektur



Was ist ein Mikrocontroller?

Vergleich mit Mikroprozessoren – Features

Eigenschaft	**µP**	**µC**
Taktfrequenz	2–4 GHz	1–20 MHz
RAM	2–8 GB (extern)	128–2 kB
Wortbreite	32–64 Bit	4–32 Bit
Architektur	Von Neumann	Harvard
Befehlssatz	CISC	„enhanced RISC“
Strukturgröße	65–22 nm	500–350 nm
Leistung	60–175 W	0,2 µW–10 mW
Preis	60–1 500 €	0,20–10 €

Tabelle: Typische Eigenschaften

Anwendungsgebiete

- Eingebettete Systeme

Unterhaltung:	DVD-Player, Fernbedienungen, Radios, Spielekonsolen, Mobiltelefone
Büro:	Tastaturen, Drucker, Scanner, Monitore, Telefone
Fahrzeuge:	in Steuergeräten für Motor, ABS, Airbags
Chipkarten:	EC-Karten, elektronische Personalausweise
Computer:	BIOS, Power Management, Temperaturfühler, Festplatten, Netzteile, Lademanager für Akkus
Werkstatt:	Multimeter, Lötkolben, Akkuschrauber
	...

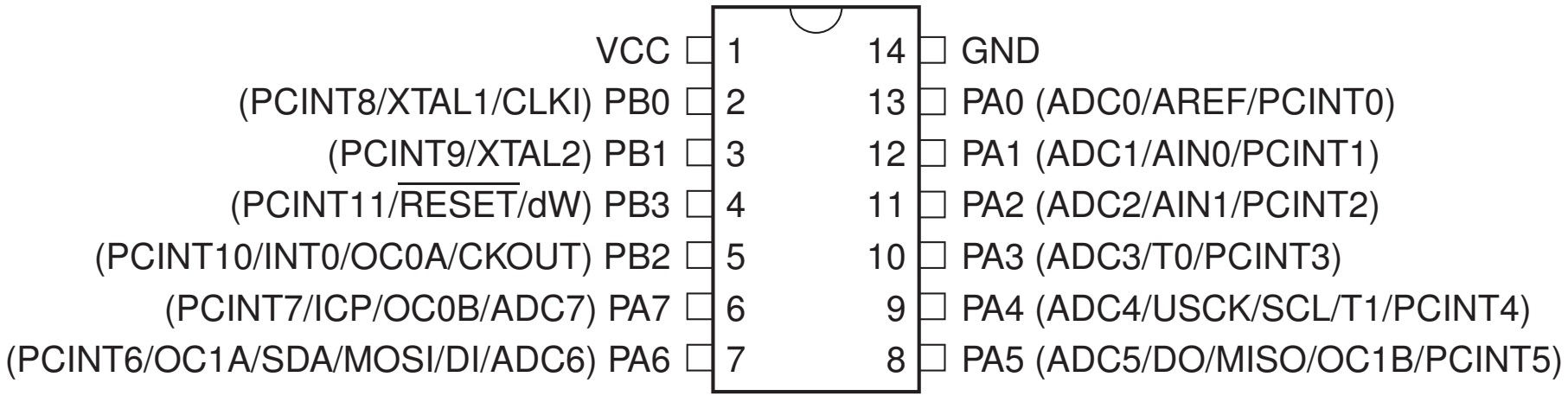
Kapitelübersicht

Interna

Eckdaten eines Atmel ATtiny84

Versorgungsspannung:	1,8–5.5 V
Taktfrequenz:	0–20 MHz
Speicher:	8 kBytes Programmspeicher
	512 Bytes RAM
	512 Bytes EEPROM (nicht-flüchtig)

PDIP/SOIC



Interna

Kerneigenschaften

- Klassischer RISC
  - Harvard-Architektur (Daten und Programm getrennt)
  - Speicherzugriff nur mit Load/Store
  - 32 Arbeitsregister à 8 Bit
    - 3 Paare nutzbar als 16-Bit-Register (X, Y, Z)
- 2-stufige Pipeline (FETCH und EXECUTE)
  - ALU-Ops in einem Takt ausführbar
- umfangreiche Peripherie
  - interner Oszillator als Taktgeber (bis 8 MHz)
  - diverse Stromsparmodi: idle, ADCNR, stand-by, power-down
  - Kommunikationsbausteine: USART, SPI, I^2C
  - 12 Mehrzweckpins, wahlweise Eingang/Ausgang, Pull-Up/offen

Speicher

Speicherarten im Überblick

Flash

- 8 kB, organisiert in 4096x16 Bit
- intern (Register) und extern (MISO/MOSI) beschreibbar
  - selbst-modifizierender Code möglich
- beliebt für Look-Up-Tables in sonst nicht genutzten Bereichen
- $>$ 10 000 Schreibzyklen

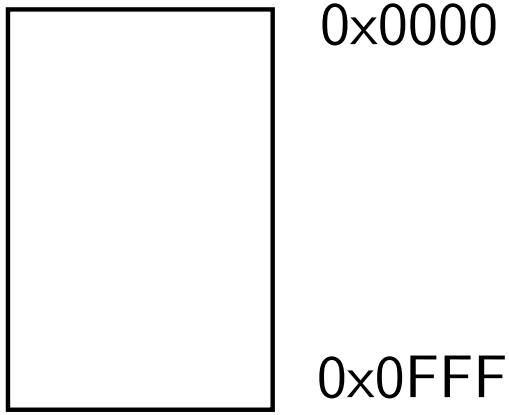


Abbildung: Programmspeicher

Speicher

Speicherarten im Überblick

SRAM

- 512 Bytes, einzeln adressierbar:
  - 5 Adressmodi, u.A. direkt, indirekt, indirekt mit Displacement
  - u.A. nutzbar für Stack Frames und Heap
- in größeren Speicherbereich eingebettet:

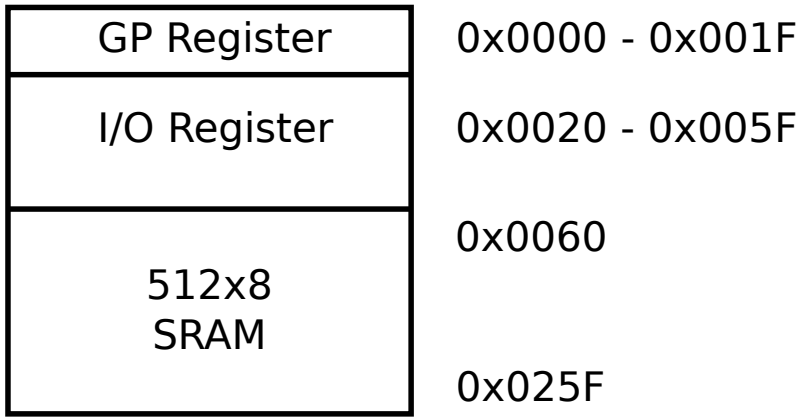


Abbildung: Datenspeicher

Speicher

Speicherarten im Überblick

EEPROM

- 512 Bytes, einzeln adressierbar
  - Zugriff über spez. I/O Register
- langsam: 3,4 ms pro Byte
  - ca. 27 000 Takte pro Byte $\hat{=}$ ca. 300 Byte/s
- nicht flüchtig
- $>$ 100 000 Schreibzyklen

Speicher

Speicherarten im Überblick

Fuse-Bits

- kleiner Konfigurationsspeicher (3 Bytes)
- einstellbare Optionen
  - Taktquelle:
    - Oszillator (intern oder extern)
    - externer Takt
    - externer Quartz-Kristall
  - Taktteiler
  - *Brown-Out Detector*: Grenzspannung
  - Watchdog und DebugWire
  - SPI-Programmierung, Selbstprogrammierung

Registerstruktur

Ansteuerung der Peripherie

- die gesamte Peripherie wird über Register gesteuert, die in den Datenspeicher eingeblendet wird

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	Page 8
0x3E (0x5E)	SPH	–	–	–	–	–	–	SP9	SP8	Page 11
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	Page 11
0x3C (0x5C)	OCR0B	Timer/Counter0 – Output Compare Register B								Page 85
0x3B (0x5B)	GIMSK	–	INT0	PCIE1	PCIE0	–	–	–	–	Page 51
0x3A (0x5A	GIFR	–	INTF0	PCIF1	PCIF0	–	–	–	–	Page 52
0x39 (0x59)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	Page 85
0x38 (0x58)	TIFR0		–	–	–	–	OCF0B	OCF0A	TOV0	Page 85
0x37 (0x57)	SPMCSR	–	–	RSIG	CTPB	RFLB	PGWRT	PGERS	SPMEN	Page 157
0x36 (0x56)	OCR0A	Timer/Counter0 – Output Compare Register A								Page 84
0x35 (0x55)	MCUCR	BODS	PUD	SE	SM1	SM0	BODSE	ISC01	ISC00	Pages 36, 51, and 67
0x34 (0x54)	MCUSR									

Abbildung: Registertabellenauszug

Registerstruktur

Ansteuerung der Peripherie

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	Page 8
0x3E (0x5E)	SPH	–	–	–	–	–	–	SP9	SP8	Page 11
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	Page 11
0x3C (0x5C)	OCR0B	Timer/Counter0 – Output Compare Register B								Page 85
0x3B (0x5B)	GIMSK	–	INT0	PCIE1	PCIE0	–	–	–	–	Page 51
0x3A (0x5A	GIFR	–	INTF0	PCIF1	PCIF0	–	–	–	–	Page 52
0x39 (0x59)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	Page 85
0x38 (0x58)	TIFR0		–	–	–	–	OCF0B	OCF0A	TOV0	Page 85
0x37 (0x57)	SPMCSR	–	–	RSIG	CTPB	RFLB	PGWRT	PGERS	SPMEN	Page 157
0x36 (0x56)	OCR0A	Timer/Counter0 – Output Compare Register A								Page 84
0x35 (0x55) 0x34 (0x54)	MCUCR MCUSR	BODS	PUD	SE	SM1	SM0	BODSE	ISC01	ISC00	Pages 36, 51, and 67

```
#include <avr/io.h>
void main(void) {
  /* Configure SLEEP instruction to enter
  stand-by mode */
  MCUCR |= (1 << SM1);
  MCUCR &= ~(1 << SM0);
  ...
}
```

Peripherie

General Purpose I/O

- „Mehrzweckanschlüsse“, direkt steuerbar
- als Eingabe oder Ausgabe konfigurierbar
- optionaler, interner Pull-Up-Widerstand (20–50 kΩ)
- Treiberstärke typ. 40 mA
- Register:
  - `DDR`*p* (*data direction register*): Eingang/Ausgang
  - `PORT`*p* als Eingang: interner Pull-Up aktiv/inaktiv, sonst: Wert
  - `PIN`*p* als Eingang: Wert, sonst: Wert toggeln
- Anwendungsbeispiele:
  - als Eingabe: Taster, Schalter, dig. Sensoren
  - als Ausgabe: LEDs, manuelle Takte, Relais, Aktoren
  - beides: nicht direkt unterstützte Bussysteme

Peripherie

General Purpose I/O – Eingänge

- „Mehrzweckanschlüsse“, direkt steuerbar

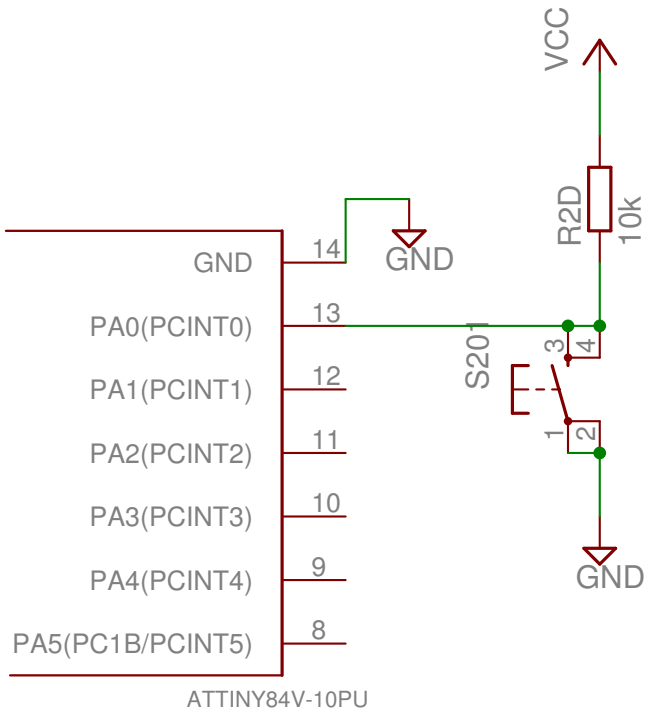


Abbildung: Taster am GPIO (Minimalbeschaltung)

Exkurs: Taster

Prellen

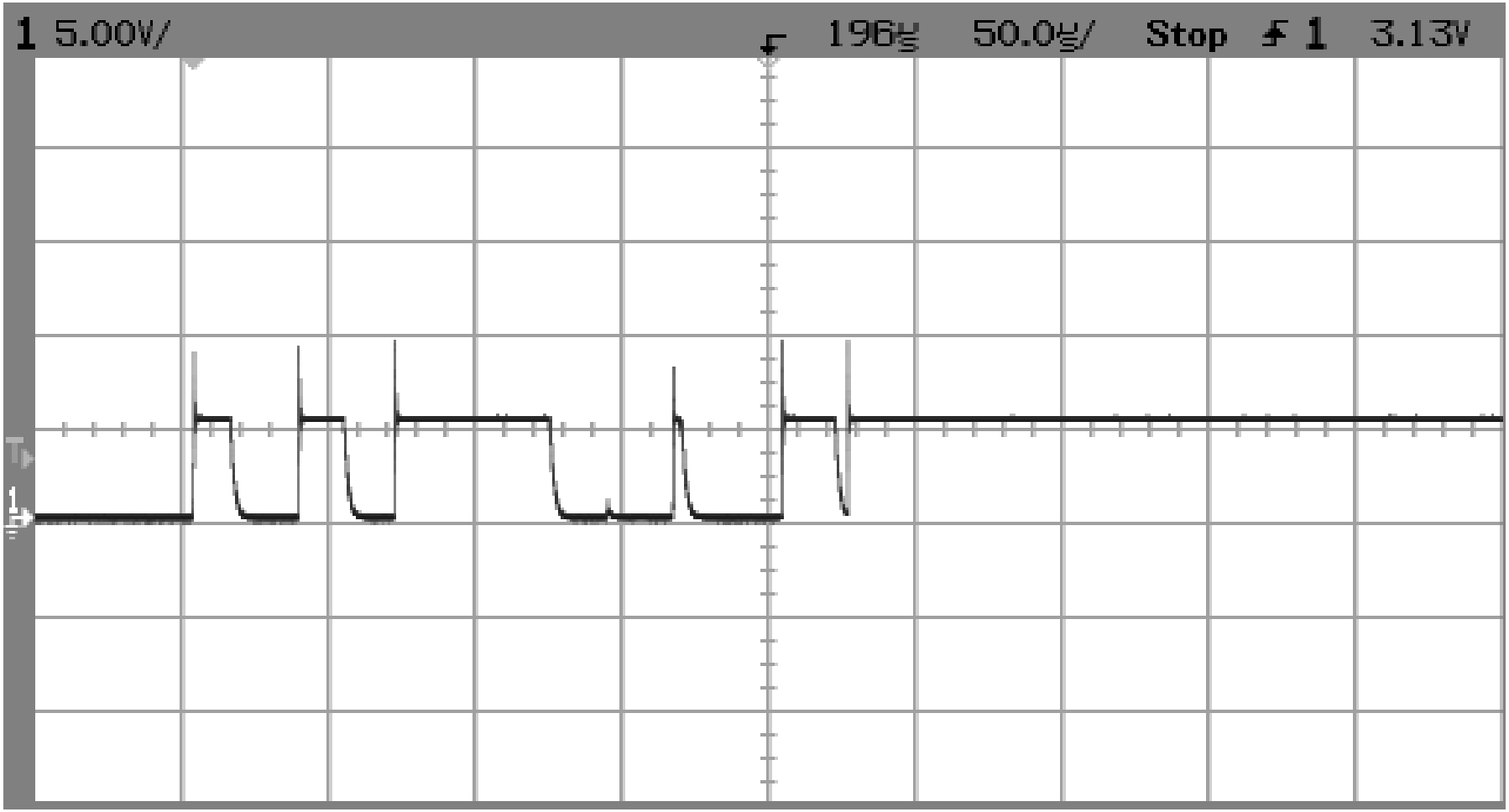


Abbildung: Prellen beim Schließen, Prellzeit etwa 250 µs

Exkurs: Taster

Entprellen

Hardware

RC-Glied als Tiefpass:

1. offen: C ist geladen
2. schließend: C entlädt über R und glättet hochfrequente Anteile
3. geschlossen: C vollständig entladen
4. öffnend: C lädt über R und glättet HF-Anteile
5. offen

Software

Entprellroutine mit Timer:
bei der ersten steigenden Flanke (Schließvorgang) wird ein Timer gestartet und bis zum Timer-Event alle folgenden Flanken ignoriert. Der nun anliegende Wert wird akzeptiert.

Peripherie

General Purpose I/O – Eingänge

- „Mehrzweckanschlüsse“, direkt steuerbar
- als Eingabe oder Ausgabe konfigurierbar

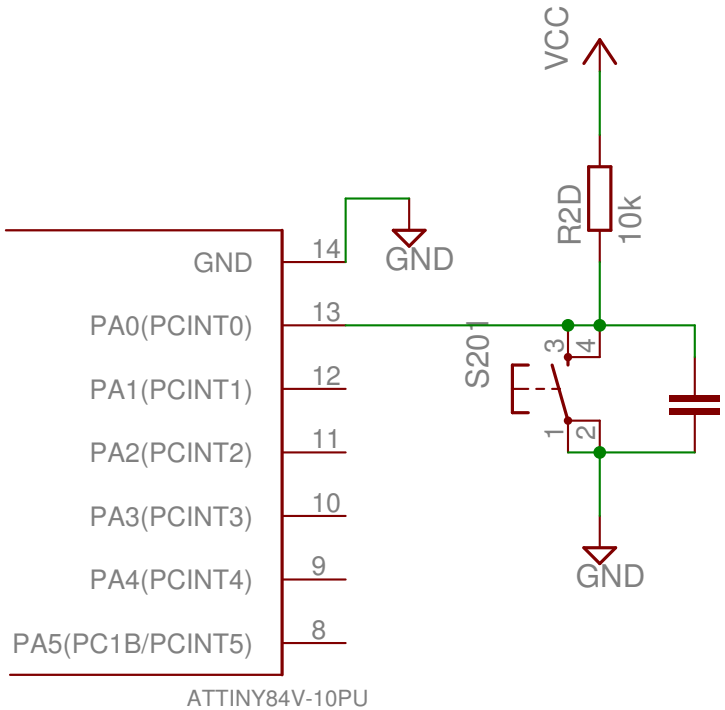


Abbildung: Taster am GPIO (entprellt)

Peripherie

General Purpose I/O – Ausgänge

- „Mehrzweckanschlüsse“, direkt steuerbar
- Treiberstärke typ. 40 mA

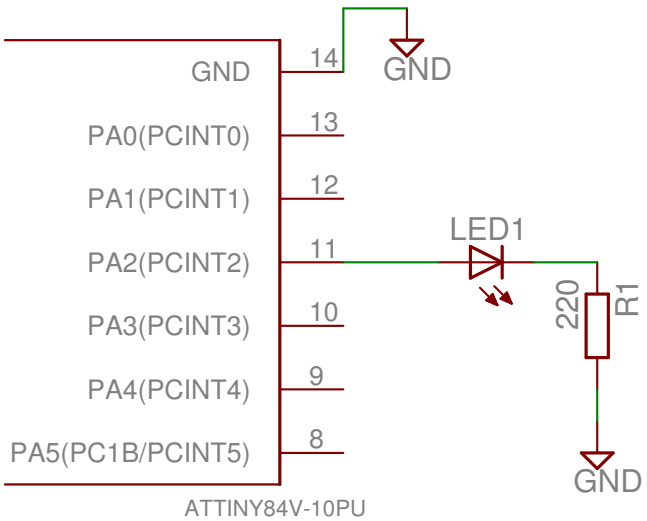


Abbildung: LED am GPIO

Peripherie

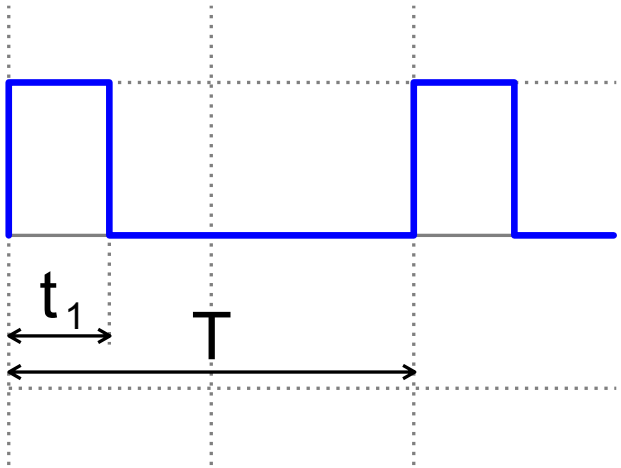
Analog/Digital-Wandler (ADC)

- Messungen analoger Spannungen von 0 – V_{CC}
- Genauigkeit: 8 oder 10 Bit
- Abtastrate: $\leq$200 kHz (bei 8 Bit: $\leq$1 MHz)
- Verstärkung: 1x bis 20x
- Anwendungsgebiete:
  - Audio (z.B. als Mikrofoneingang)
  - Sensoren (z.B. Thermowiderstände, Fotowiderstände)
  - „stufenloses Einstellen“ mit Potentiometern

Peripherie

Timer mit Pulsweitenmodulator (PWM)

- Erzeugung von Rechtecksignalen
  - einstellbare Periodendauer T
  - einstellbare Pulsbreite t_1
- Anwendungen:
  - Dimmung von LEDs
  - Drehzahlsteuerung von Motoren (z.B. beim CPU-Lüfter)
  - analoge Audioausgabe via *poor man's DAC*



Peripherie

Kommunikation – Universal Asynchronous Reciever/Transmitter (UART)

- serielle Punkt-zu-Punkt-Verbindung, vollduplex
- standardisiertes Protokoll:
  - normalerweise 1 Startbit, 8 Datenbits, 1 Stoppbit (kurz: 8N1)
  - Paritätsbit optional
  - auch exotischere Kombinationen möglich: 5O1.5
    - 5 Datenbits, ungerade Parität, 1,5 Stoppbits
  - typ. Geschwindigkeiten: 9 600, 57 600, 115 200 Baud
- Spannungspegel: CMOS, RS-232, EIA-485
- Emulation mit Bit Banging möglich (und beim ATtiny84 nötig)

Peripherie

Kommunikation – Universal Asynchronous Reciever/Transmitter (UART)

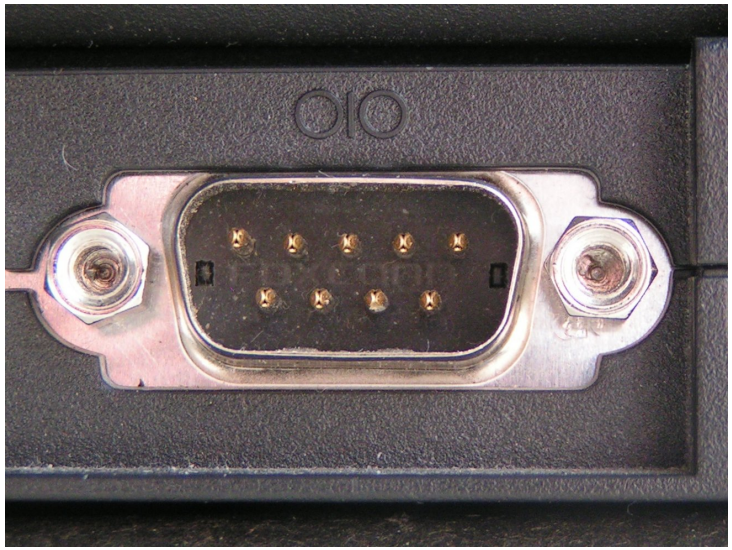


Abbildung: RS-232 über Sub D-9 Buchse

RS-232-Anwendungen:

- Terminals für Firmware-Upgrades, Debugging
- antike Computer-Peripherie, z.B. Modems, Drucker

Peripherie

Kommunikation – Universal Asynchronous Reciever/Transmitter (UART)

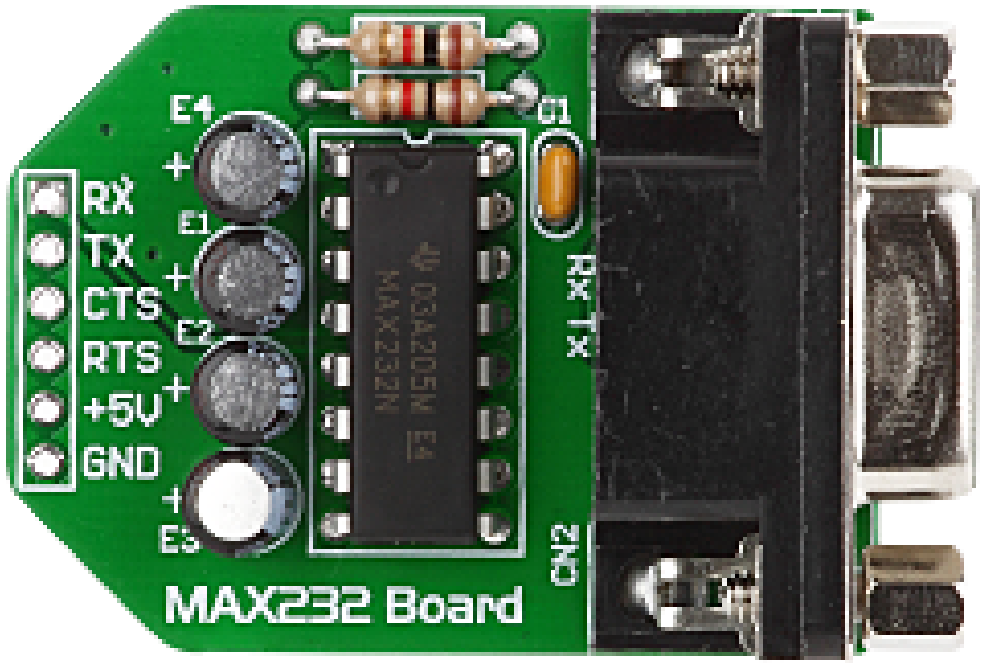


Abbildung: RS-232 mit Maxim MAX232N

Peripherie

Kommunikation – Universal Asynchronous Reciever/Transmitter (UART)

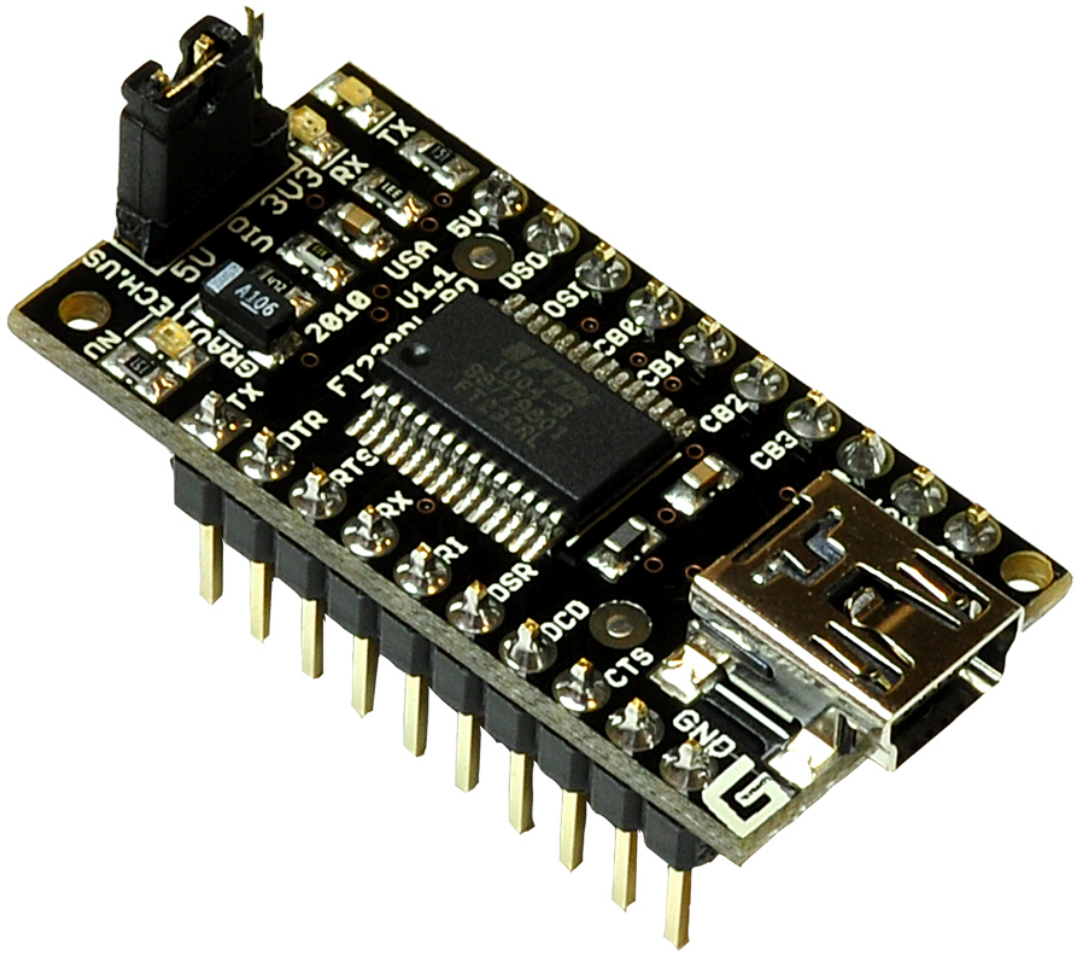


Abbildung: USB mit FTDI FT232RL

Peripherie

Kommunikation – Universal Asynchronous Reciever/Transmitter (UART)

Abbildung: Ethernet mit Microchip ENC28J60

Peripherie

Kommunikation – Serial Peripheral Interface (SPI)

- serielle Schnittstelle für mehrere Teilnehmer (Bus-Architektur)
- synchroner Takt $\longrightarrow$ keine Synchronisation nötig
- höherer Takt möglich als bei UART (einige MHz)
- Master-Slave-Prinzip
- Signale:
  - `SCLK`: slave clock
  - `MOSI`: Master Out/Slave In (Datenleitung)
  - `MISO`: Master In/Slave Out (Datenleitung)
- optional: Slave Select (`SS`, `CS`, `STE`)

Peripherie

Kommunikation – Serial Peripheral Interface (SPI)

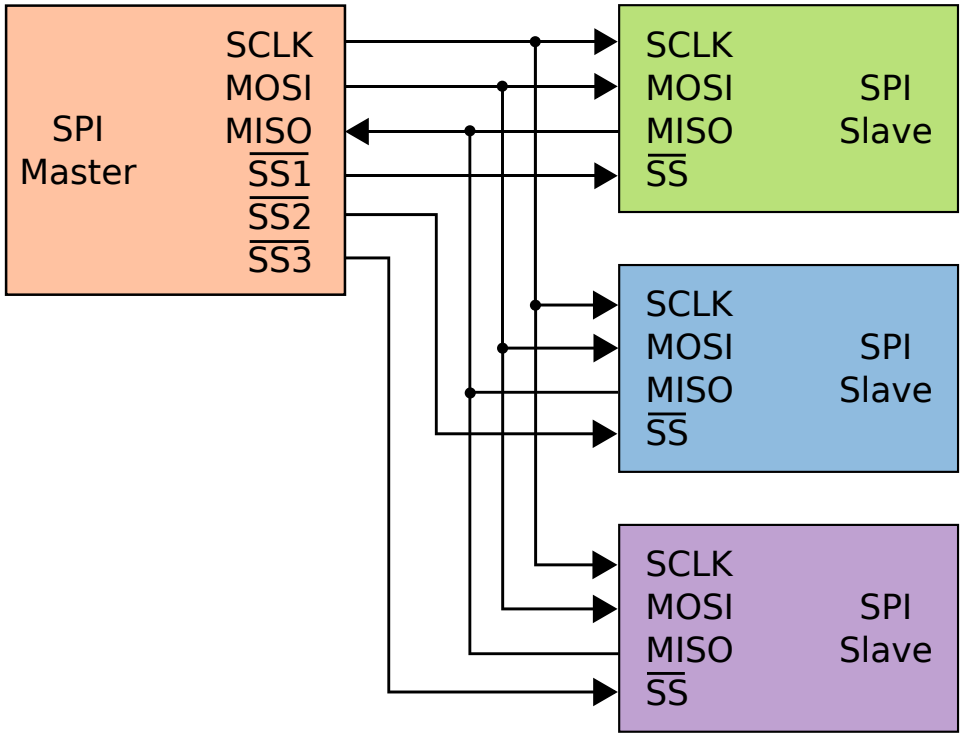


Abbildung: SPI-Sternverbindung

Bild: Wikimedia Commons, User Cburnett, GFDL

Peripherie

Kommunikation – Serial Peripheral Interface (SPI)

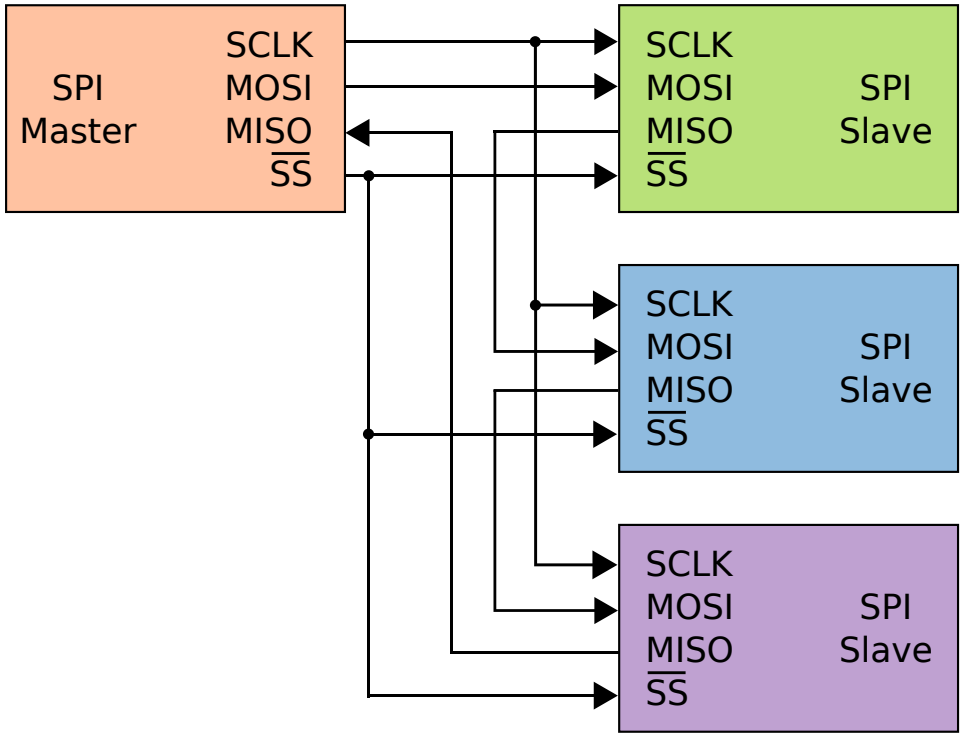


Abbildung: SPI-Busverbindung durch Kaskadierung der Slaves

Bild: Wikimedia Commons, User Cburnett, GFDL

Peripherie

Weitere interessante Bausteine

I^2C:	„Inter-Integrated Circuit“, synchroner Multi-Master-Bus mit 10-Bit-Adressierung, bis zu 1 Mbit/s, nur zwei Leitungen (`SCK`, `SDA`)
Debug Wire:	Proprietärer in-circuit Debug-Mechanismus, nutzbar mit GDB
Watchdog:	automatischer Reset bei Absturz

Interrupts

Ereignisgesteuerte Ausführung

Interrupts sind Programmunterbrechungen (*Interrupt Service Request*, IRQ) die durch externe, asynchrone Ereignisse ausgelöst werden und die Ausführung einer Unterbrechungsroutine (*Interrupt Service Routine*, ISR) ermöglichen.

- Interrupts können global oder einzeln ein- und ausgeschaltet werden

Interrupts

Interruptquellen und -arten

Hardware-Interrupts

Auslösung durch (CPU-)externe Peripherie:

- Pegeländerungen am Pin `INT0`, weitere Interrupt-Pins können via externem Interrupt-Controller angeschlossen werden
  - Auslöser konfigurierbar: high level, low level, rising edge, falling edge
- Versorgungsspannung zu niedrig / wiederhergestellt *Brown-Out Detection*
- Timerüberlauf
- Ergebnis der ADC-Messung verfügbar
- Watchdog-Zeitüberschreitung

Interrupts

Interruptquellen und -arten

Software-Interrupts

Auslösung durch (CPU-)interne Zustände:

- ALU-Ausnahmen, z.B. Division durch null, Überlauf
- MMU-Ausnahmen, z.B. Stack-Überläufe, Adressfehler, Instruktionsfehler (z.B. ungültige Opcodes)
- Künstlich ausgelöst durch spezielle Instruktionen

Software-Interrupts werden auch *traps* genannt.

```c
/* without interrupts */

void main(void) {
  initialize();

  while(1) {
    while(PINB & (1 << PB2))
      ;
    handle_event();
  }
}
```

```c
/* with interrupts */

volatile int have_event = 0;

ISR(INT0_vect) {
  have_event = 1;
}

void main(void) {
  initialize();

  while(1) {
    if(have_event) {
      have_event = 0;
      handle_event();
    }
    sleep();
  }
}
```

Interrupts

Anwendungsszenarien

Nebenläufigkeit:	oft können Hardware-Prozesse asynchron gestartet werden (z.B. ADC), sodass während des Prozesses die Anwendung andere Dinge erledigen kann
Echtzeit:	bestimmte Events müssen sofort bearbeitet werden, ein Polling-Mechanismus wäre zu träge
Energieeffizienz:	während auf ein bestimmtes Event gewartet wird, kann der µC in einem Energiesparmodus sein und vom Interrupt geweckt werden

Kapitelübersicht

Beschaltung

Pinout

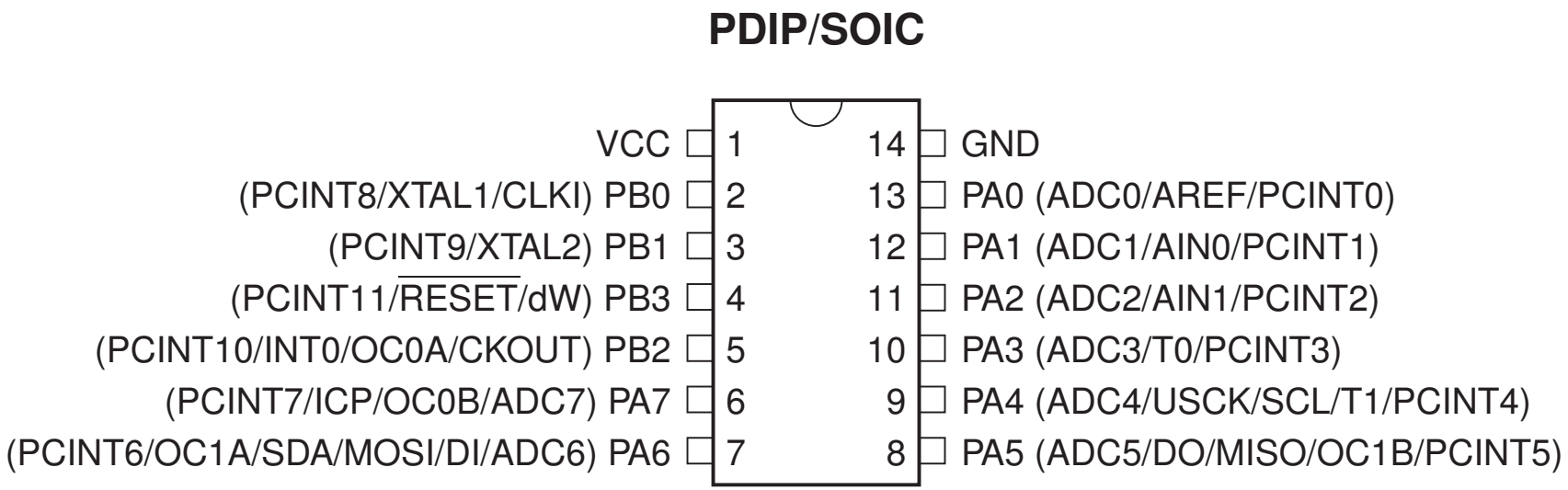
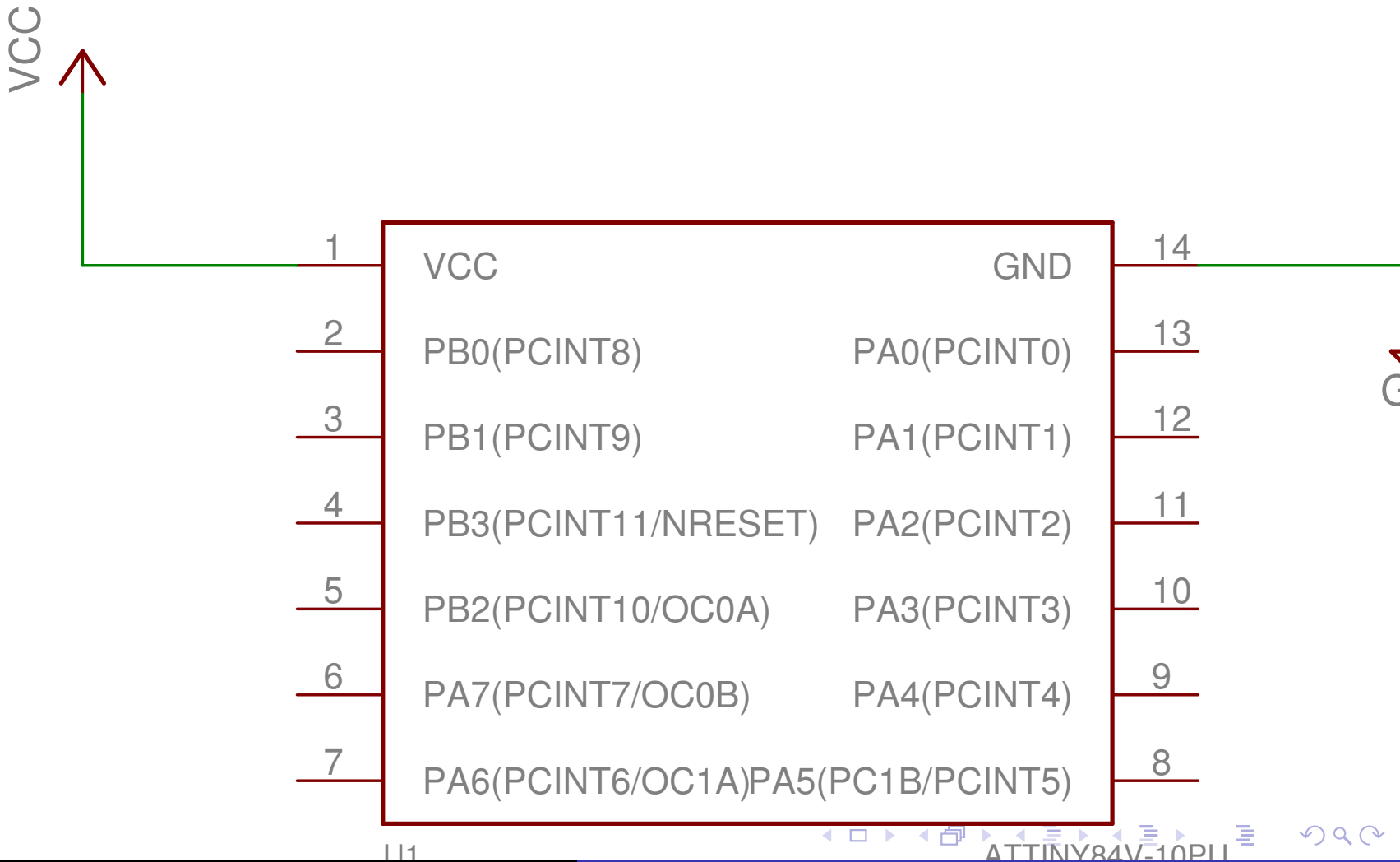


Abbildung: Pinbelegung des ATtiny84

Beschaltung

Minimalbeschaltung



Beschaltung

Spannungsversorgung

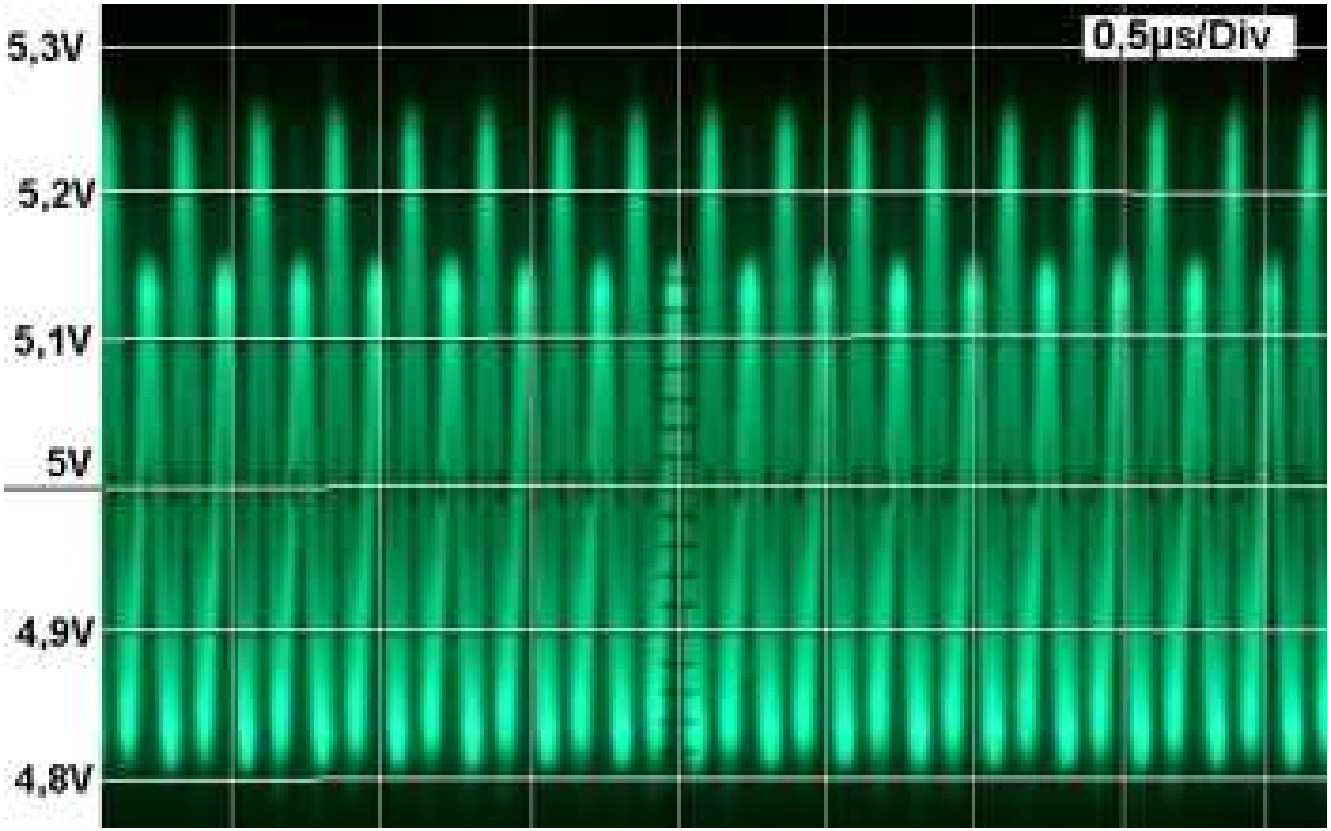


Abbildung: Versorgungsspannung, kurze Spannungsstöße im Takt des µC

Beschaltung

Spannungsversorgung

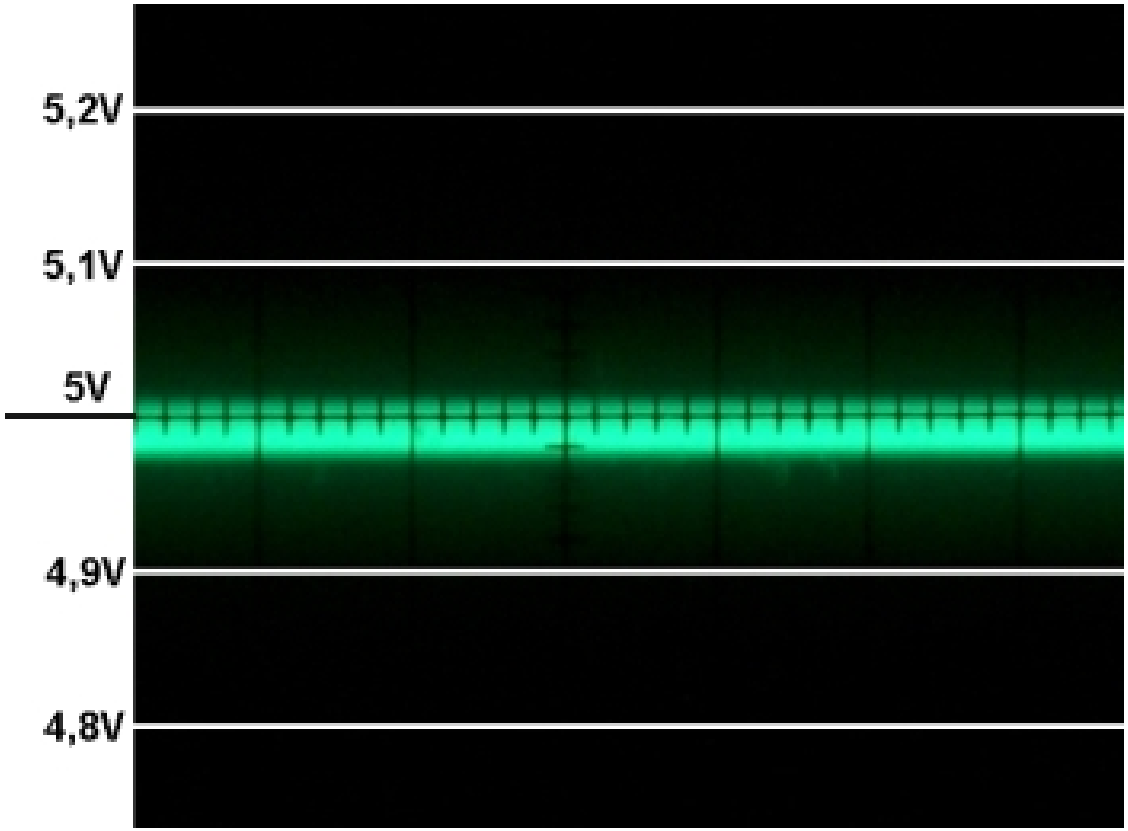


Abbildung: Versorgungsspannung mit 100 nF Abblockkondensator

Beschaltung

Spannungsversorgung

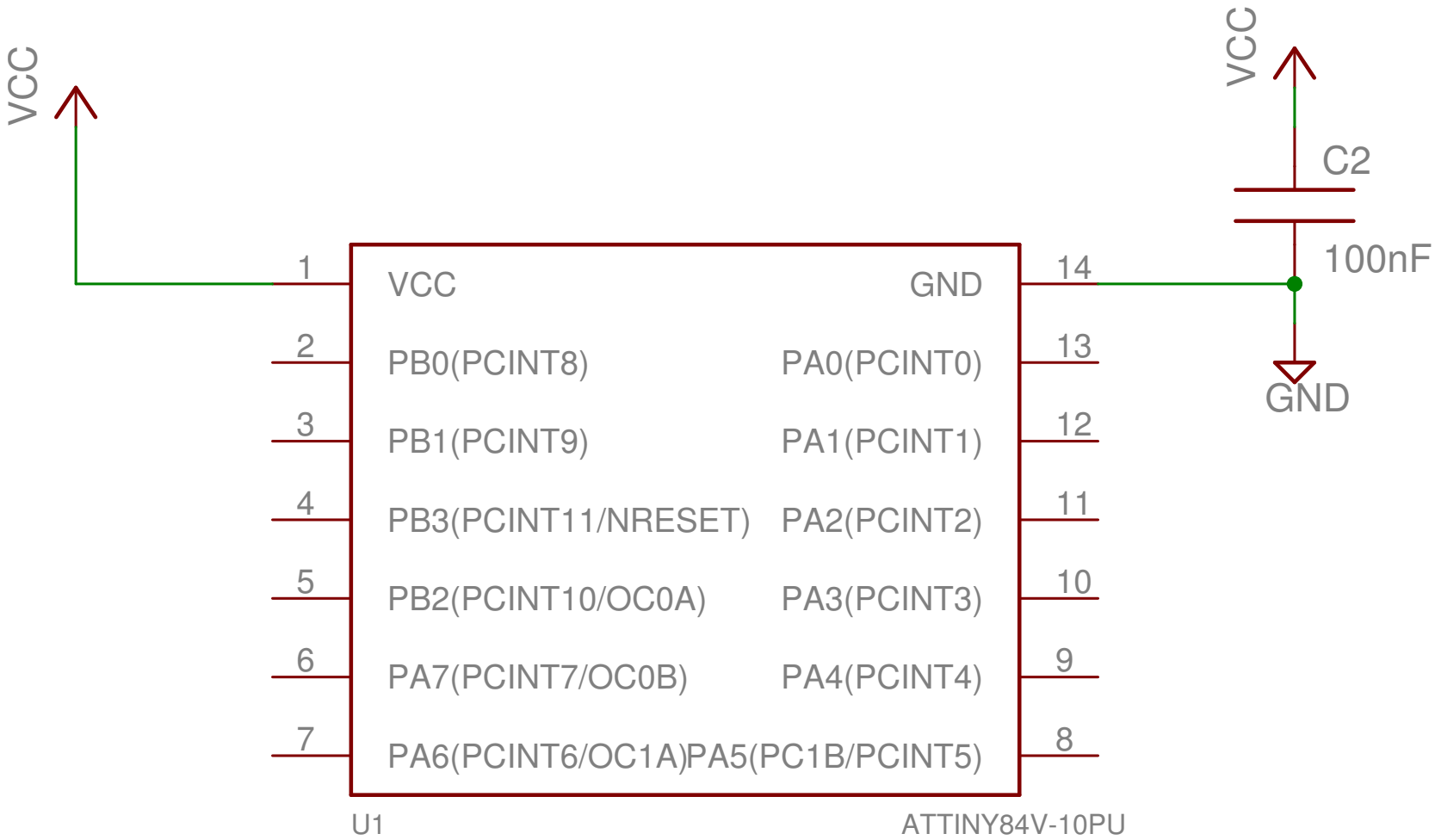


Abbildung: Minimalbeschaltung mit Abblockkondensator

Beschaltung

Immunisierung des RESET-Eingangs

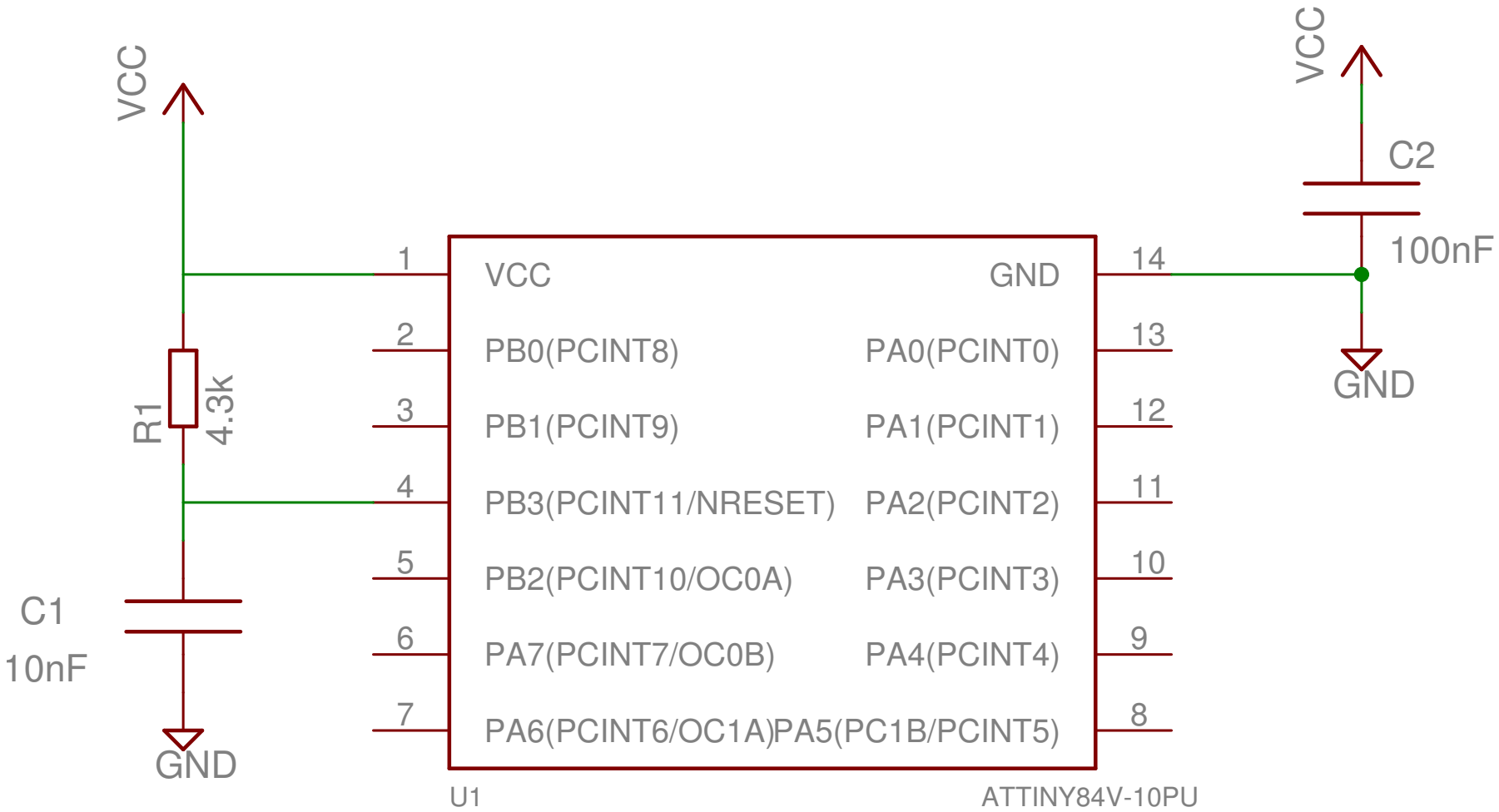


Abbildung: RESET-Stabilisierung nach Atmel Application note AVR042

Programmierung

Programmiermethoden

In-System Programming (ISP)

- Programmierung innerhalb einer Schaltung möglich (*in-target*)
- benötigt 6 Pins, 4 davon am µC
  - VCC, GND
  - MISO, MOSI, SCK, RESET
- Taktgenerator muss aktiv sein

Abbildung: AVR ISP mkII

Programmierung

Programmiermethoden

High Voltage Programming

- In-target Programmierung oft unmöglich:
  - RESET wird auf *very high* gelegt, 12 Volt
- kein Takt nötig
- Fuse-Bits werden ignoriert
  - sinnvoll bei kleinen µC: RESET abschalt- und als GPIO nutzbar
  - fehlerhafte Fuse-Bits korrigieren

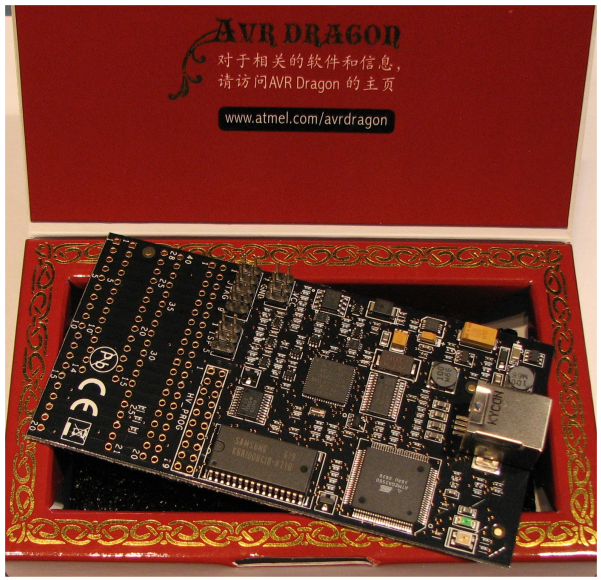


Abbildung: AVR Dragon

Programmierung

Programmiermethoden

debugWire

- proprietäres Protokoll
  - Einsatz nur mit Atmel Programmiergeräten möglich
- In-target Programmierung möglich, benötigt nur zwei Pins: GND, RESET
- Fuse-Bits nicht veränderbar
- verwendbar mit GDB

Kompilierung

Workflow

1. Kompilieren des Programms
   - GCC toolchain (C++, C und ASM) mit `libc-avr`
   - BASCOM-IDE für BASIC-Entwicklung
2. Erstellen der EEPROM-Belegung (optional)
3. Berechnen der Fuse-Bits
   - diverse Online-Rechner
   - Atmel AVR Studio
4. Programmieren von Fuses, EEPROM und Flash
   - Vielzahl an Programmiertools verfügbar, die wichtigsten sind `avrdude`, PonyProg2000 und Atmel AVR Studio

```
 $ avrdude -p REdS -w vorlesung.hex

avrdude: CAU device initialized and ready to accept instructions

Writing | ################################################## | 100% 90 mins

avrdude: Device signature = 0x23042014

avrdude: safemode: Fuses OK

avrdude done. Thank you.
```