Geekclock

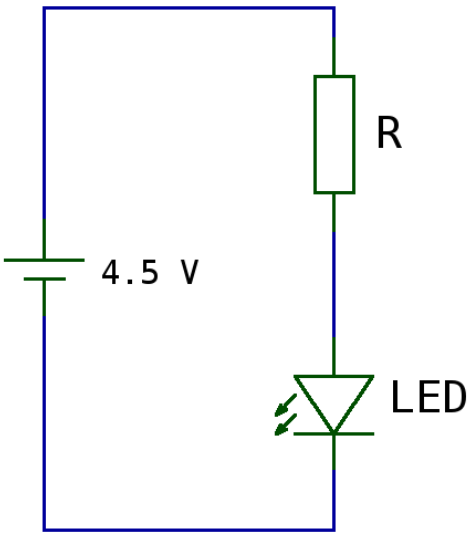
Code and Hardware Walkthrough

Andreas Müller

Chaos Singularity 2007

1. Elektronik
   - Stromkreis
   - Gesetz von Ohm
   - Knotenpunkt- und Maschenregel
   - Komponenten
2. Using Microcontrollers
   - What is an MCU?
   - ATMega8 features
   - Differences from coding on a PC
3. Geekclock Hardware
   - Hardware Overview
   - Circuit diagram
4. Geekclock Software
   - Software concept
   - Structure Overview
   - Code Walkthrough

Stromkreis



- Strom fliesst nur im geschlossenen Kreis
- hier sind LED und Widerstand in Serie
  - gleicher Strom fliesst durch beide Elemente
  - Spannungen über den Elementen sind unterschiedlich
  - bei Parallelschaltung wäre es umgekehrt

Ohm'sches Gesetz

Proportionalität zwischen Widerstand R, Strom I durch R, und Spannung U über R:

$$U = R * I$$

Beispiel von der letzten Folie – benötigter Widerstand?

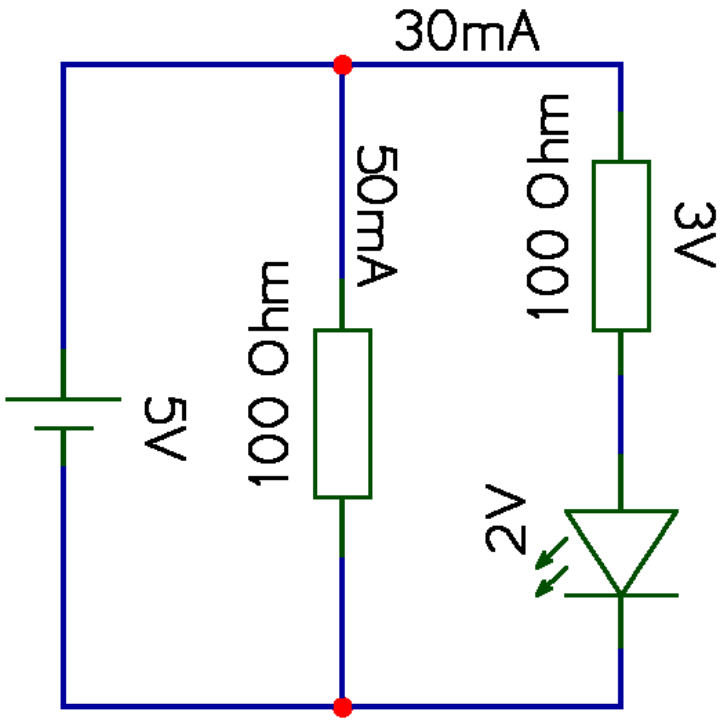
- Spannung über Widerstand (Batteriespannung - LED bias): $U = 4.5V - 2V = 2.5V$
- Strom: LED soll ca 10mA haben
- Widerstand:

$$R = \frac{U}{I} = \frac{2.5V}{10mA} = \frac{2.5V}{0.01A} = 250\Omega$$

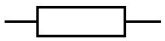
Knotenpunkt- und Maschenregel

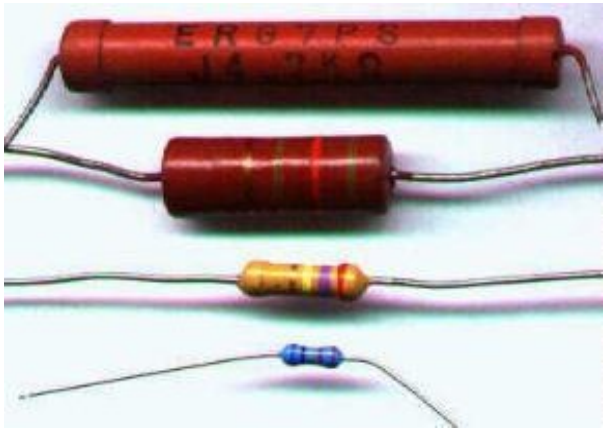
Kirchhoff'sche Gesetze:

- Knotenregel: Summe aller Ströme in einem Knoten ist Null ($\rightarrow$ es gehen keine Elektronen verloren)
- Maschenregel: Summe aller Spannungen in einer Masche ist Null ($\rightarrow$ es fällt über einer idealen Leitung keine Spannung ab)

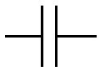
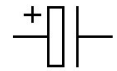


Widerstand

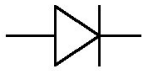
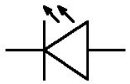
- Symbol: R
- Schaltzeichen: 
- Kenngrösse: Widerstand mit Einheit Ohm (Ω)
- Spannung über Widerstand ist proportional zu Strom
- Farbcodierung gibt Widerstandswert an



Kondensator

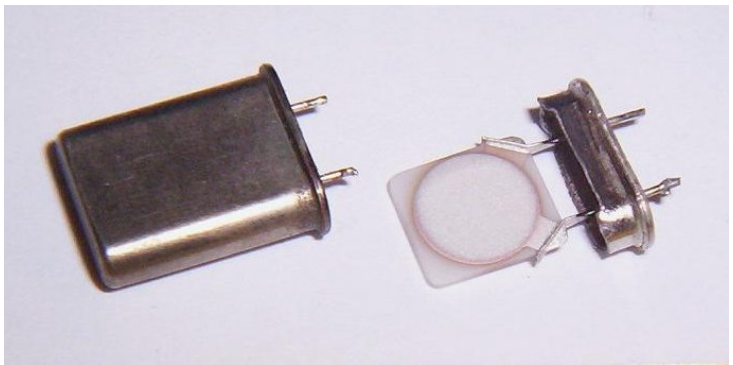
- Symbol: C
- Schaltzeichen: 
- Kenngrösse: Kapazität mit Einheit Farad (F)
- Schaltzeichen für Elektrolytkondensatoren: 
- speichert Strom / stabilisiert Spannung
- Werte sind meist direkt aufgedruckt

Diode und LED

- Symbol: *D*
- Schaltzeichen:  bzw.  (LED)
- Dioden lassen Strom nur in eine Richtung durch → Einbaurichtung (Polarität) beachten
- in der Geekclock als Anzeige (LED) und Verpolungsschutz
- LED: Light Emitting Diode

Quarz

- Symbol: Q
- Schaltzeichen: ⊣▯⊢
- liefert sehr stabile Referenzfrequenz
- Funktionsweise: Quarzplättchen mit angelegten Elektroden:
  - Quarz verbiegt sich beim Anlegen einer Spannung
  - Spannung weg → Deformation umgekehrt → Spannung wird produziert
  - positive Rückkoppelung nur bei Resonanzfrequenz und Harmonischen



What is a microcontroller?

- Wikipedia: A microcontroller (or MCU) is a computer-on-a-chip. It is a type of microprocessor emphasizing self-sufficiency and cost-effectiveness, in contrast to a general-purpose microprocessor (the kind used in a PC).
- RAM, ROM, memory and a CPU are, along with various peripherals, all contained on a single chip, which can be programmed to fulfill a specific task.

ATMega8

- RISC Microcontroller, max 16MHz
- 23 I/O lines
- lots of integrated peripherals
  - timers
  - AD converters
  - PWM
  - internal or external oscillator possible
- sleep mode support
- In-System Programmable Flash memory

MCU coding peculiarities

- less powerful hardware
- in our case
  - 32kHz core frequency (up to 16MHz would be possible)
  - 1KB SRAM
  - 8KB Flash memory
    ... ought to be enough for everyone
- no FPU

MCU coding peculiarities (continued)

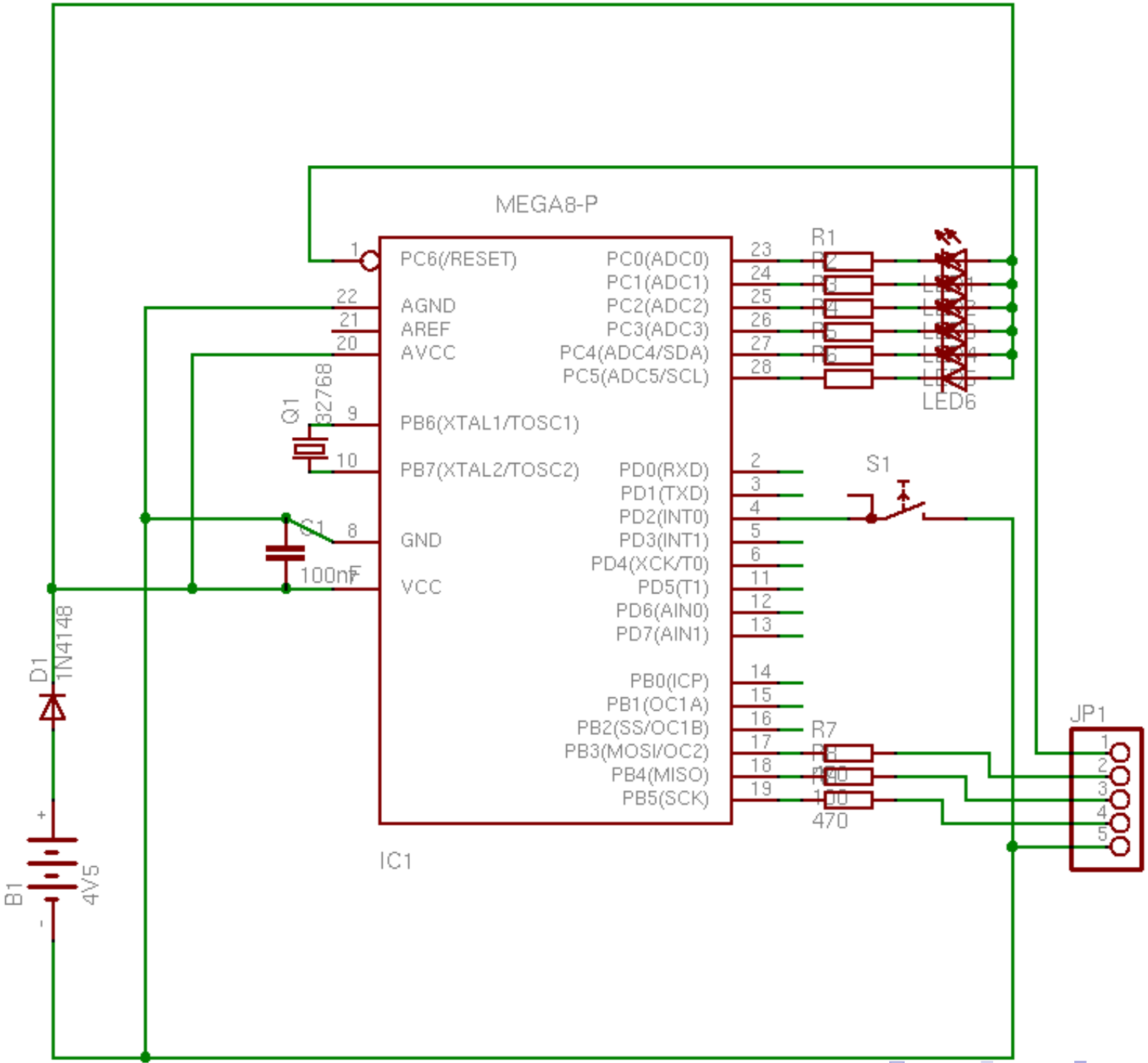
- no OS
  - only one process
  - no virtual memory, etc
  - hard real time is possible
  - avr-libc provides some functions
- no printf
- no easy way to tell if an error is in software or in hardware
- programs are usually designed to never reach an end

some advice for efficient coding

- use `gcc` with `-Os` (`-O2` and optimize for size)
- don't use 32bit integers, when you only need 8bit ($\rightarrow$ use `uint8_t` or `int8_t`) [demo]
- condition checks are preferable to expressions with modulo operations or multiplications
- avoid floating point variables and functions (`sin()`, `sqrt()`, `..`)
- keep variable count low (even if the SRAM is big enough – if you have only a few variables, they can always stay in the registers)
- there is usually no need to code in assembler
- don't worry ... 32kHz is more than it might seem

Geekclock Hardware Overview

- core: ATMega8 MCU
- clock from 32kHz crystal (low frequency to save power)
- 6 LEDs to show time in binary
- button to control clock
- diode to protect MCU from wrong polarity
- interface for programming via LPT cable or USB programmer


MEGA8-P
PC6(/RESET)
AGND
AREF
AVCC
PB6(XTAL1/TOSC1)
PB7(XTAL2/TOSC2)
GND
VCC
PC0(ADC0)
PC1(ADC1)
PC2(ADC2)
PC3(ADC3)
PC4(ADC4/SDA)
PC5(ADC5/SCL)
PD0(RXD)
PD1(TXD)
PD2(INT0)
PD3(INT1)
PD4(XCK/T0)
PD5(T1)
PD6(AIN0)
PD7(AIN1)
PB0(ICP)
PB1(OC1A)
PB2(SS/OC1B)
PB3(MOSI/OC2)
PB4(MISO)
PB5(SCK)
IC1
R1
R7
470
LED6
S1
JP1
Q1
32768
C1
100nF
D1
1N4148
B1
4V5

Software concept

- hardware timer generates interrupt each second
- time is updated in interrupt routine
- button generates interrupt
- time is shown in main routine after button was pressed

Structure overview

- `geekclock.c`: interrupts, main-routine
- `lowlevel.c`: initialisation (Timer, Ports), lowlevel functions
- `datetime.c`: calendar functions, time functions
- `led.c`: LED control, effects

Code

(Code)

Questions

Questions?