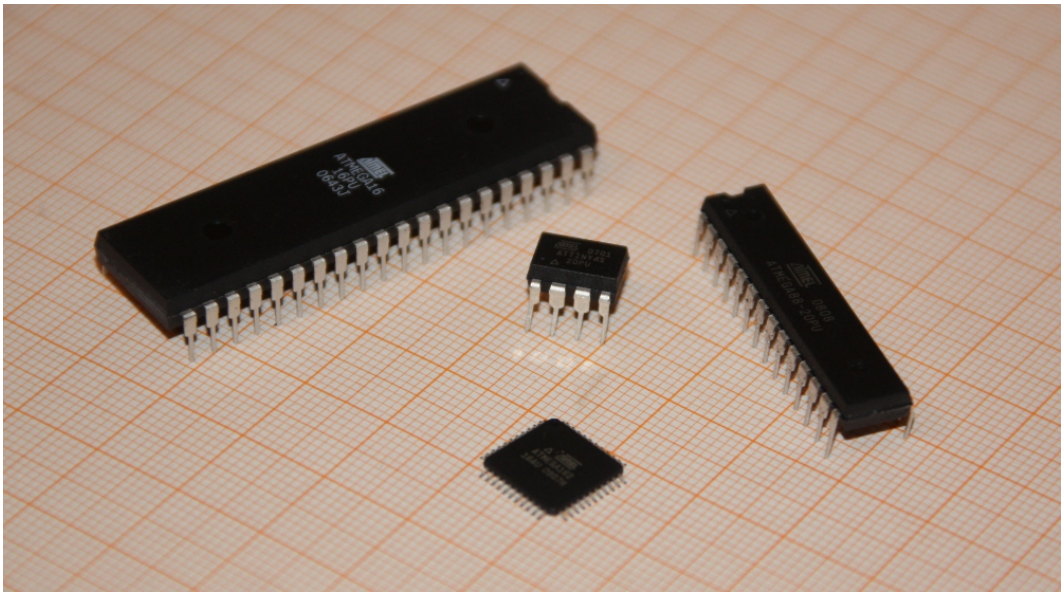
AVR-Mikrocontroller mit dem GCC programmieren

Mario Haustein

Chemnitzer Linux User Group

10. Februar 2012

Die Hardware



CPU + RAM + Flash + EEPROM + Peripherie
in einem Gehäuse

Typenbezeichnung

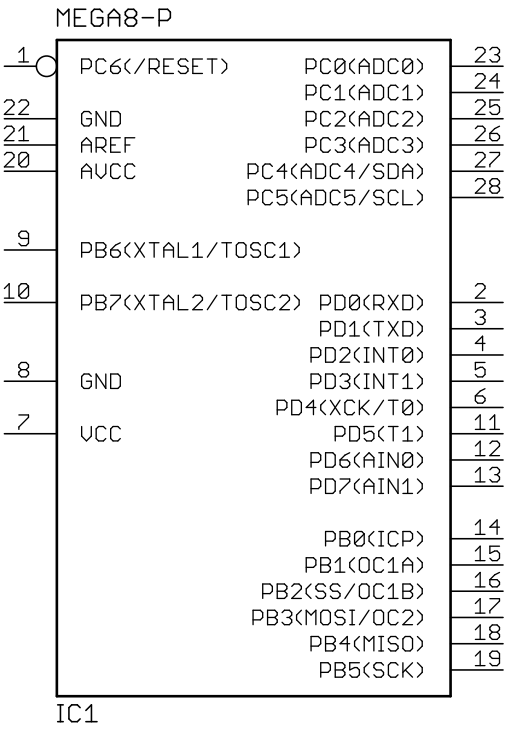
- Serie

ATtiny	wenig Speicher, kleine Bauform, wenig I/O
ATmega	**mehr Speicher, große Bauformen, viel I/O**
ATXMega	viel Speicher, nur SMD-Gehäuse, DMA, HW-Crypto, ...
AVR UC3	FPU, Speicherschutz, hohe Taktraten

- Typnummern (für ATmega):
  - Flash-Größe (Zweierpotenz)
  - I/O-Komponenten
  - Max. Taktrate
  - Revision, Gehäuse
- Bsp.:
  - ATmega16-16PU
  - ATmega162-16AU
  - ATmega88-20PU
  - ATmega328PA-20AU

Ein Beispiel: ATmega8 – der kleinste ATmega

- $\geq$ 20 ansteuerbare Pins
- Davon 5 am A/D-Wandler
- Einzeln als Ein- und/oder Ausgang konfigurierbar.
- Alternativ: Aufschaltung einer Sonderfunktion.
  - Interrupt
  - USART
  - I^2C
  - SPI
  - Timer
  - ...



Zwei Typen im Vergleich

ATmega8	**ATmega88**
8KByte Flash, 1KByte SRAM, 512Byte EEPROM	
16MHz	20MHz
2 8Bit-Zähler	2 8Bit-Zähler
1 16Bit-Zähler	1 16Bit-Zähler
3 PWM-Kanäle	6 PWM-Kanäle
USART	USART
SPI	SPI
I2C	I2C
8-Kanal-ADC	8-Kanal-ADC
Komparator	Komparator
2 ext. Interrupts	2 ext. Interrupts
	Pin-Change-Interrupts
Stromsparmodi	Stromsparmodi

CPU, Peripherie, Programmierung

- RISC-Befehlssatz
  - Zwischen den Typen kompatibel. Reduzierter Befehlssatz für ATtiny.
  - Auf die Programmiersprache C optimiert.
- Getrennter Programm- und Datenspeicher (Harvard-Architektur)
- I/O-Komponenten sind in den Datenspeicher eingeblendet.
  - Bsp.: ATmega8

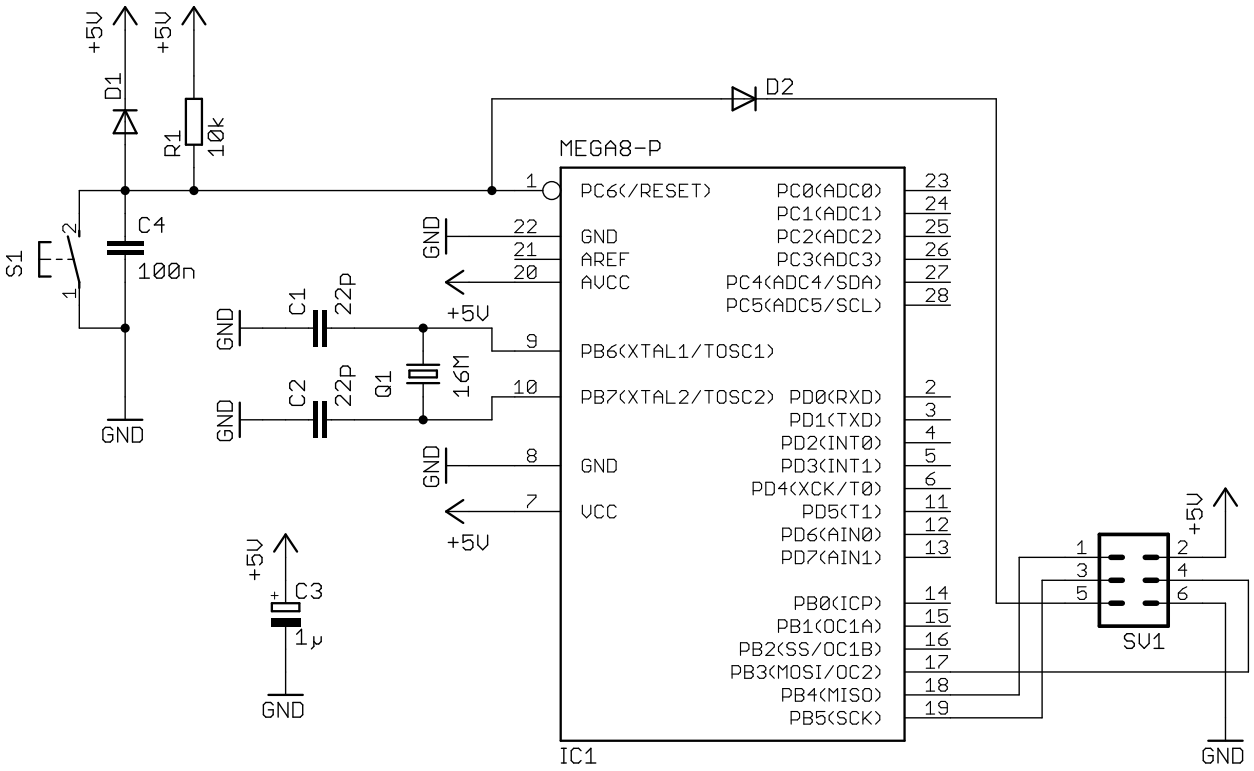
    0x000 – 0x01F Registerbank
    0x020 – 0x05F Peripherie
    0x060 – 0x45F SRAM
- In-System-Programming möglich
  - Der Flash kann auch noch in seiner Zielumgebung neu programmiert werden.
  - Programmer-Hardware: z.B. `http://www.usbprog.org/`
  - Programmer-Software: z.B. `http://www.nongnu.org/avrdude/`

Dokumentation, Toolchain

- Datenblatt
  - Dokumentation zu Speicherlayout, Peripheri-Ansteuerung, elektrischem Verhalten, ISP-Protokoll, ...
  - Frei unter `http://www.atmel.com/products/avr/` erhältlich.
- Toolchain
  - Binutils & GCC können AVR-Code erzeugen.
  - „Reduzierte“ libc: `http://www.nongnu.org/avr-libc/`
  - Programmer: avrdude
  - Mit GDB und Debug-Adapter ist sogar Debugging möglich.

Grundlegende Beschaltung

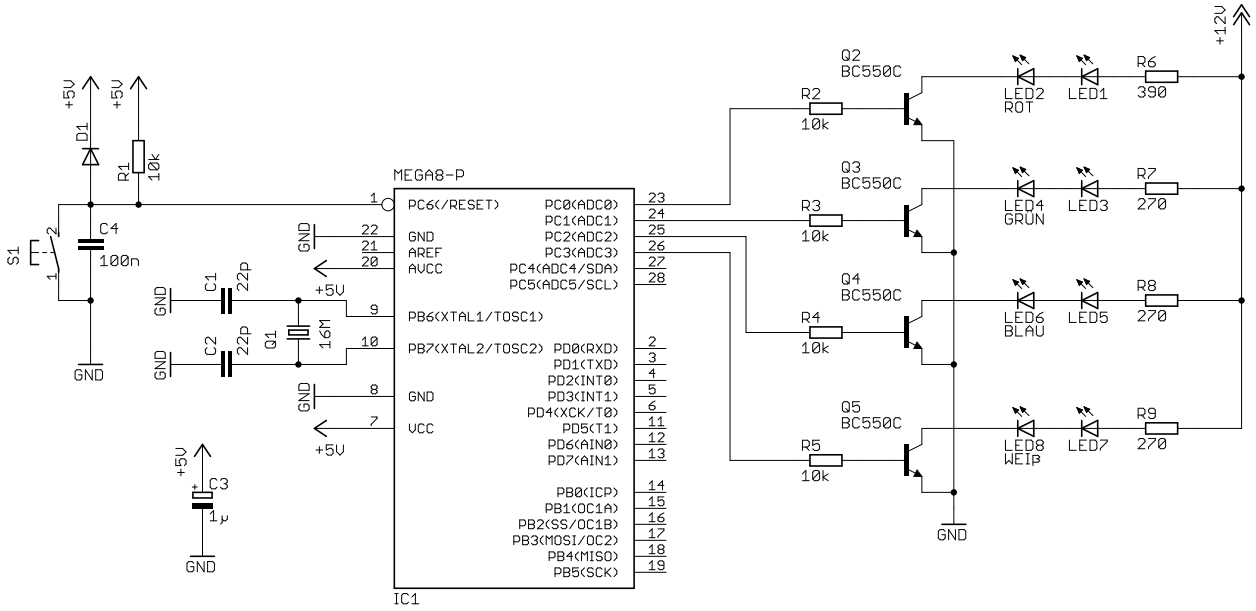
Takt, Reset-Logik, Stützkondensator, ISP-Schnittstelle



Fuse-Register programmieren

```
$ avrdude -c avrisp2 -p m8 -b 10 -P usb -t

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100%
     0.00s

avrdude: Device signature = 0x1e9307
avrdude> write lfuse 0 0xef
>>> write lfuse 0 0xef

avrdude> write hfuse 0 0xd9
>>> write hfuse 0 0xd9

avrdude> quit
>>> quit

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

Beispielschaltung



LED's an PORTC blinken lassen

```
#include <stdint.h>


int main()
{
  unsigned long int i;

  *((volatile uint8_t*)0x34) = 0x0f;
  while(1)
  {
    *((volatile uint8_t*)0x35) |= 0x0f;
    for(i = 0; i < 100000; i++);
    *((volatile uint8_t*)0x35) &= ~0x0f;
    for(i = 0; i < 100000; i++);
  }

  return 0;
}
```

```
$ avr-gcc -mmcu=atmega8 -Wall -Wextra -o main.bin main.c
$ avr-size main.bin
$ avr-objcopy -R .eeprom main.bin -O ihex main.hex
$ avrdude -c avrisp2 -p m8 -b 10 -P usb -U flash:w:main.hex:i
```

Mit der avr-libc ist es übersichtlicher

```
#include <stdint.h>
#include <avr/io.h>


__attribute__((naked,noreturn)) void main()
{
  volatile unsigned long int i;

  DDRC = 0x0f;
  while(1)
  {
    PORTC |= 0x0f;
    for(i = 0; i < 100000; i++);
    PORTC &= ~0x0f;
    for(i = 0; i < 100000; i++);
  }
}
```

```
$ avr-gcc -mmcu=atmega8 -Wall -Wextra -Os -o main.bin main.c
$ avr-objcopy -R .eeprom main.bin -O ihex main.hex
$ avrdude -c avrisp2 -p m8 -b 10 -P usb -U flash:w:main.hex:i
```


Und noch etwas übersichtlicher . . .

```
#include <stdint.h>
#include <avr/io.h>
#include <util/delay.h>


__attribute__((naked,noreturn)) void main()
{
  DDRC = 0x0f;
  while(1)
  {
    PORTC |= 0x0f;
    _delay_ms(500);
    PORTC &= ~0x0f;
    _delay_ms(500);
  }
}
```

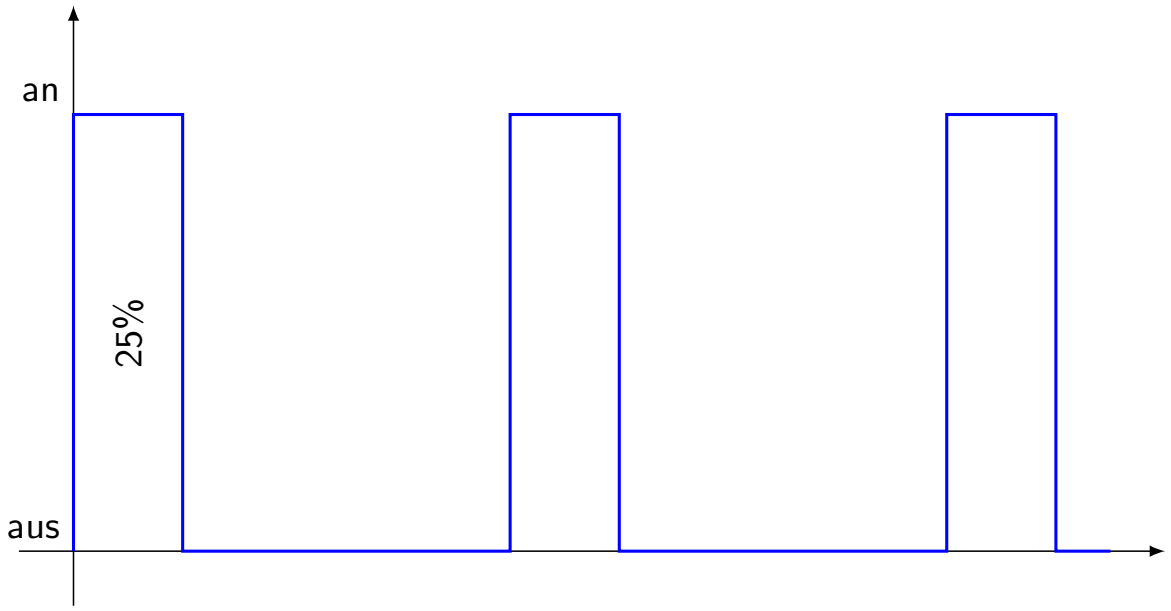
```
$ avr-gcc -mmcu=atmega8 -Wall -Wextra -Os -DF_CPU=16000000 -o main.
    bin main.c
$ avr-objcopy -R .eeprom main.bin -O ihex main.hex
$ avrdude -c avrisp2 -p m8 -b 10 -P usb -U flash:w:main.hex:i
```

Ein komplexeres Beispiel

- Helligkeitssteuerung von 4-LED-Kanälen
  - Pulsweitenmodulation
  - Nur 3 Hardware-PWM-Kanäle $\Longrightarrow$ Software-PWM
- Implementierung von Übergangsmustern
- Entkopplung von PWM und Ansteuerung über Interrupts
- Zugriff auf den Programmspeicher
- Ausblick: Hardware-PWM

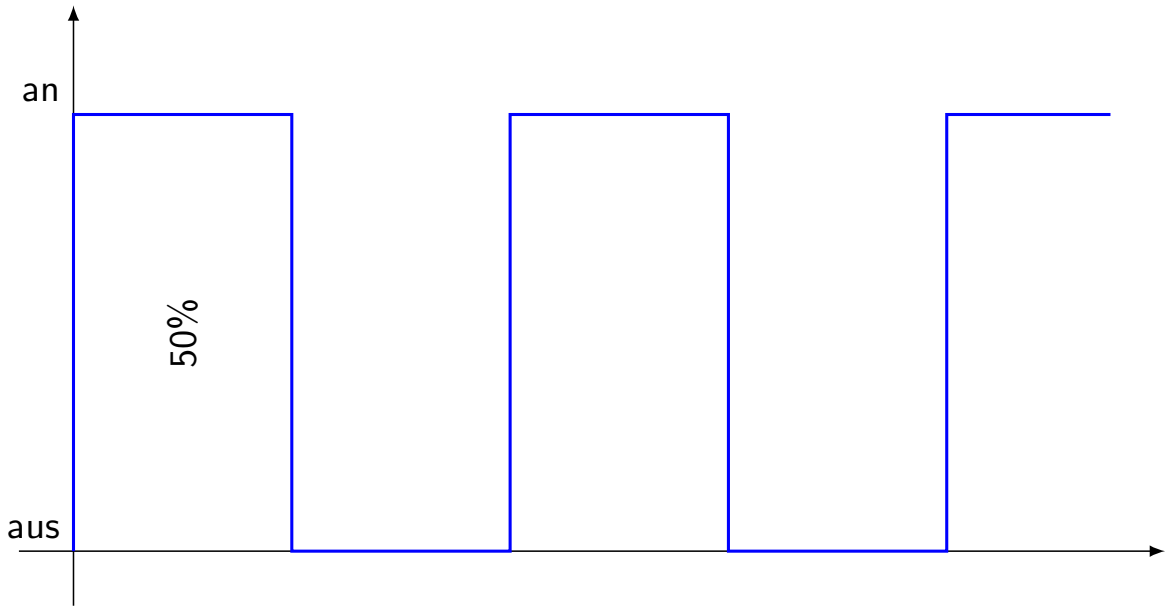
Was ist PWM?

- LED's haben zwei Zustände: an und aus.
- Helligkeitsregelung durch Verhältnis zwischen An-Zeit und Aus-Zeit.
- Bedingung: $f \geq 100\,\mathrm{Hz}$



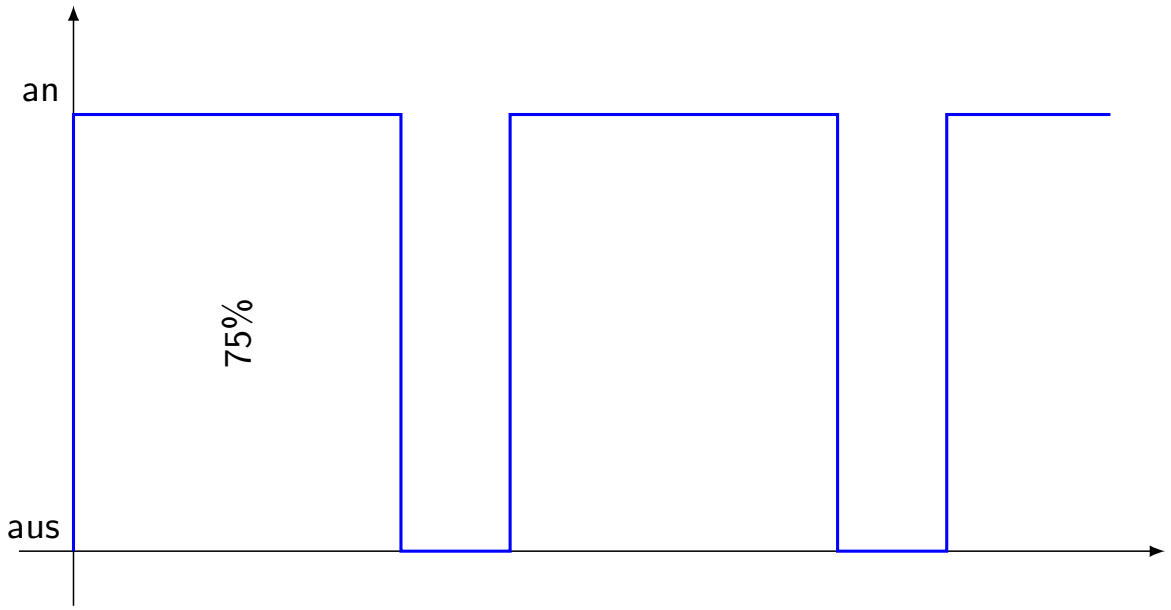
Was ist PWM?

- LED's haben zwei Zustände: an und aus.
- Helligkeitsregelung durch Verhältnis zwischen An-Zeit und Aus-Zeit.
- Bedingung: $f \geq 100\,\mathrm{Hz}$



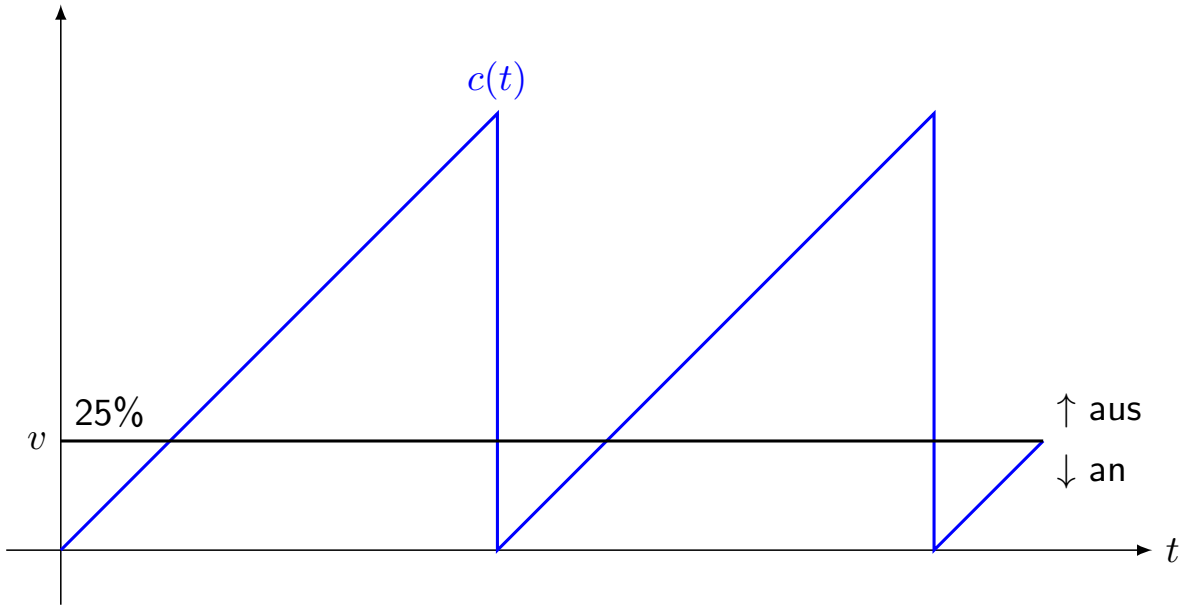
Was ist PWM?

- LED's haben zwei Zustände: an und aus.
- Helligkeitsregelung durch Verhältnis zwischen An-Zeit und Aus-Zeit.
- Bedingung: $f \geq 100\,\mathrm{Hz}$



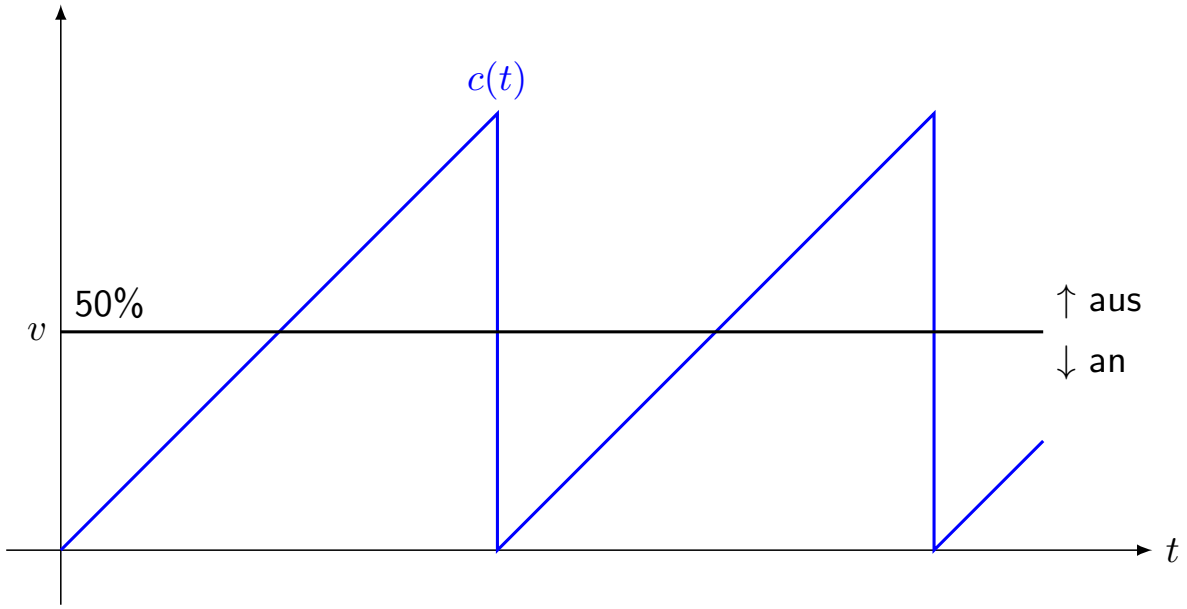
Implementierung

- Zähler c, Überlauf ca. alle $10\,\mathrm{ms}$.
- Schwellwert v



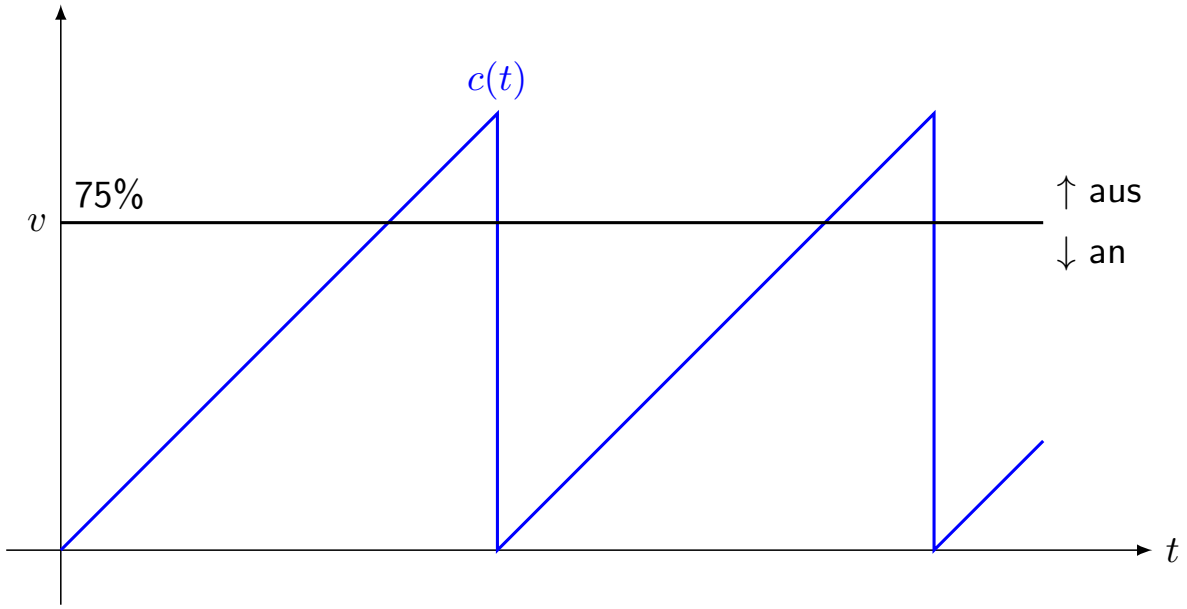
Implementierung

- Zähler c, Überlauf ca. alle $10\,\mathrm{ms}$.
- Schwellwert v



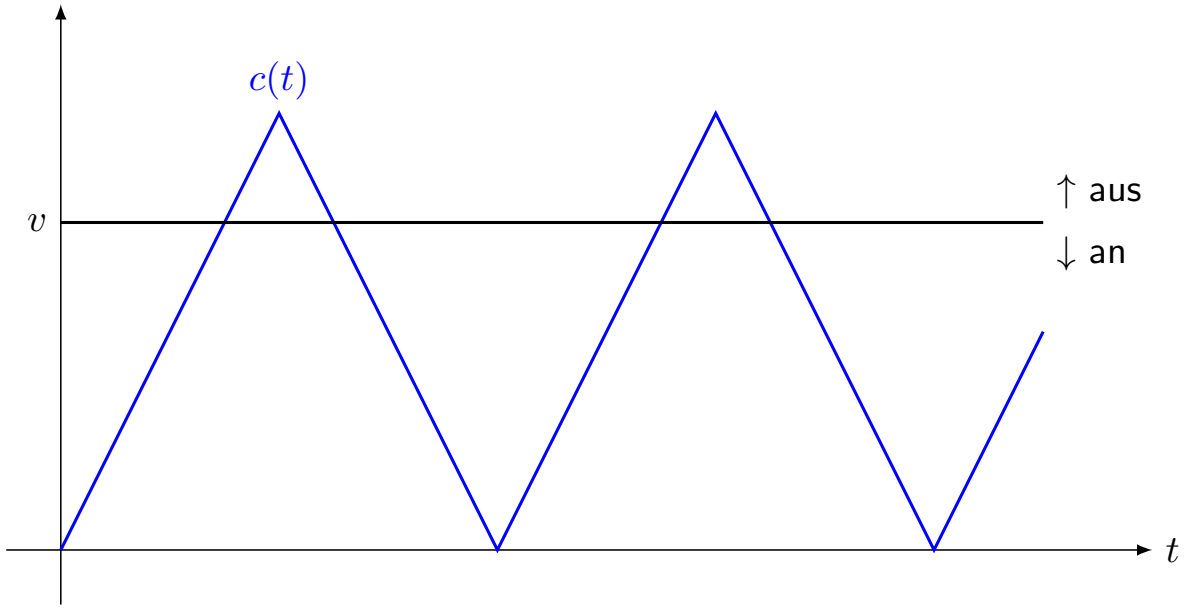
Implementierung

- Zähler c, Überlauf ca. alle $10\,\mathrm{ms}$.
- Schwellwert v



Demo

Implementierung



Hardware-PWM

Auf Zuhörerwunsch: Phasenkorrekte PWM

```
#include <stdint.h>
#include <avr/io.h>
#include <util/delay.h>


__attribute__((noreturn)) void main()
{
  TCCR2  = (1 << CS22) | (1 << CS21) | (1 << WGM20) | (1 << COM21);
  OCR2   = 0;
  DDRB   |= 1 << 3;

  while(1)
  {
    OCR2++;               /* PWM-Level in OCR2 */
    _delay_ms(10);
  }
}
```

Ausblick

Oder: Woran könnte man als AVR-Neuling noch versuchen?

- Serielle Schnittstelle
- I^2C-Bus, SPI-Bus
- A/D-Wandler
- Sensortasten
- LCD / 7-Segment-Anzeigen
- EEPROM
- Energiesparmodi