Mikrocontroller AG

Termin 2

Grundlagen

Wer sind wir

Michael Gehring
Studiert: Elektrotechnik Master 1 Semester
E-Mail: michael.gehring@rwth-aachen.de

Steffen Robertz
Studiert: Elektrotechnik Master 2 Semester
E-Mail: steffen.robertz@rwth-aachen.de

Dirk Braun
Studiert: Elektrotechnik Bachelor 6 Semester
E-Mail: dirk.joachim.braun@rwth-aachen.de

Organisatorisches

- Termine:
  - Montags 18:30 im AHII
  - Mehrere Termine zum Löten der Baustäzte
- Skript, Tutorials, Beispielcode etc. im L²P
- Bei Fragen:
  - michael.gehring@rwth-aachen.de
  - steffen.robertz@rwth-aachen.de
  - dirk.joachim.braun@rwth-aachen.de
- Gute Studienstartler müssen ihren Zettel mitbringen

Zahlendarstellung

Zahlendarstellung I Dezimal

- Zahlenstellen werden mit Zehnerpotenzen gewertet, bekannt
- Jede Dezimalzahl lässt sich in eine Summe gewichteter Potenzen der Basis 10 darstellen
- 10 ist für Menschen “schön” (10 Finger ?)

Beispiel:

1328 = 1*1000 + 3*100 + 2*10 +8*1
= 1*10^3 + 3*10^2 + 2*10^1 + 8*10^0

Formal: Wert*Basis^#Stelle

Zahlendarstellung II Binär

- Kleinste Einheit das Bit
- 2 Zustände in der Digitaltechnik an,True,1 aus,False, 0
- Zahlendarstellung bleibt im Prinzip gleich
- Basis = 2 da wir 2 Zustände pro Stelle bzw. Bit haben

Beispiel:

$$0b1010 = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0$$
$$= 1*8 + 0*4 + 1*2 + 0*1$$
$$= 10$$

Zahlendarstellung III Binär

Das Umrechnen von Dezimal in Binär und umgekehrt ist nervig und fehleranfällig. Deshalb entweder Taschenrechner oder nur binär!!

Diesen Spaß habt ihr in GGI2 :P

Zahlendarstellung IV Hex

- Hexadezimal Zahlen sind eine Vereinfachung zur Darstellung in Binär
- Neue Basis B=16
- Warum? 2^4 = 16
- 4 Bit könne zusammengefasst werden
- Zahlen gehen von 0-9 A-F mit A=10 und F=15
- Merkhilfe C = Zwölf D = Dreizehn

Zahlendarstellung V Hex

Beispiel:

0x7AF = 7*16^2 + A*16^1 + F*16^0

= 7*256 + A*16 + F*1

= 7*256 + 10*16 + 15*1

= 1967

Oder einfach dec(0x7AF) im Taschenrechner

Zahlendarstellung VI Hex

Beispiel:

1) 0b1101011110110101 = ???

2) 0b 0110 1011 1011 0101 Zerlegen der Quartette

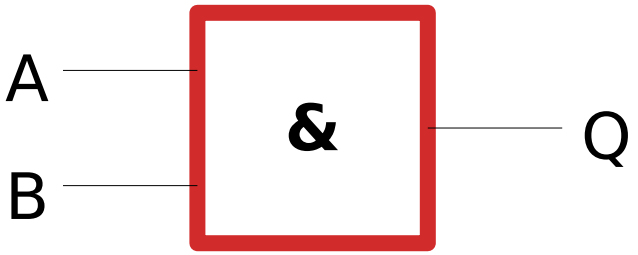
3) (6 11 11 5) Dezimal der Quartette

4) 0x 6 B B 5

5) 0x6BB5 Kompakte Darstellung

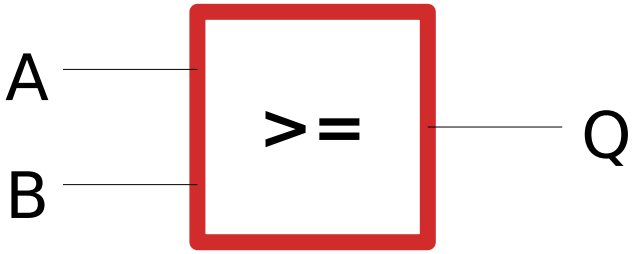
Binäre Operatoren

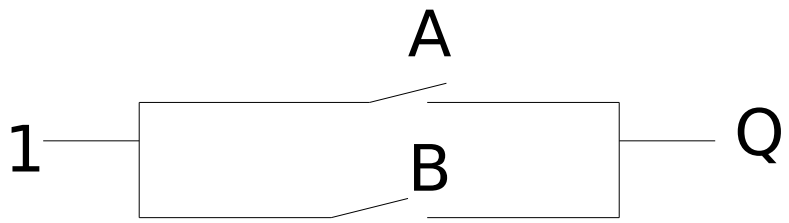
Operatoren I Das Und



A	B	A&B=Q
0	0	0
0	1	0
1	0	0
1	1	1

Operatoren II Das Oder

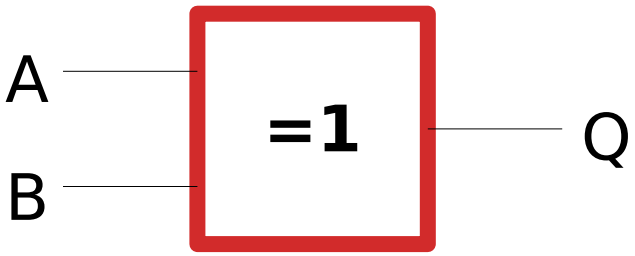




A	B	A|B=Q
0	0	0
0	1	1
1	0	1
1	1	1

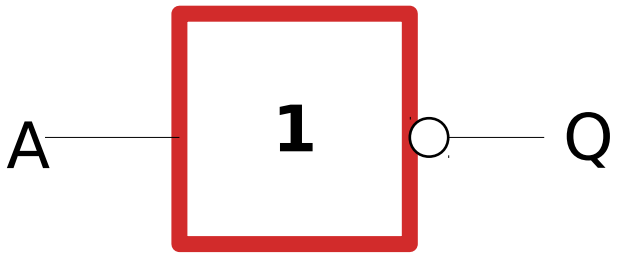
Operatoren III Das XOR

eXclusive OR “Entweder oder”, Antivalenz



A	B	A^B=Q
0	0	0
0	1	1
1	0	1
1	1	0

Operatoren IV Das NOT



A	~A=Q
0	1
1	0

Operatoren V Beispiele

Operatoren VI Linksshift

- Der Linksshift ist eine sehr nützliche Operation auf Binärzahlen
- Alle Bits werden 1 Stelle nach Links verschoben
- Nullen ziehen von Links nach
- Das entspricht einer Multiplikation um 2

Beispiel:

Binär:		Dezimal:	
0b00001010	<<	10	<<
0b00010100	<<	20	<<
0b00101000		40	

Operatoren VII Rechtssshift

- Der Rechtsshift ist eine ganz nützliche Operation auf Binärzahlen
- Alle Bits werden 1 Stelle nach Rechts verschoben
- Nullen ziehen von Links nach
- Das entspricht einer Division um 2

Beispiel:

Binär:		Dezimal:	
0b00001010	>>	10	>>
0b00000101	>>	5	>>
0b00000010		2	

Operatoren VIII Beispiele

Was ist ein Mikrocontroller

Was ist ein Mikrocontroller?

- Eine Art integrierter PC
- Integration wesentlicher Bestandteile auf einen Chip
- Besondere Perhipherie für spezielle Anwendungen
- Wir verwenden den Atmega16 der Firma Atmel

Der ATmega16

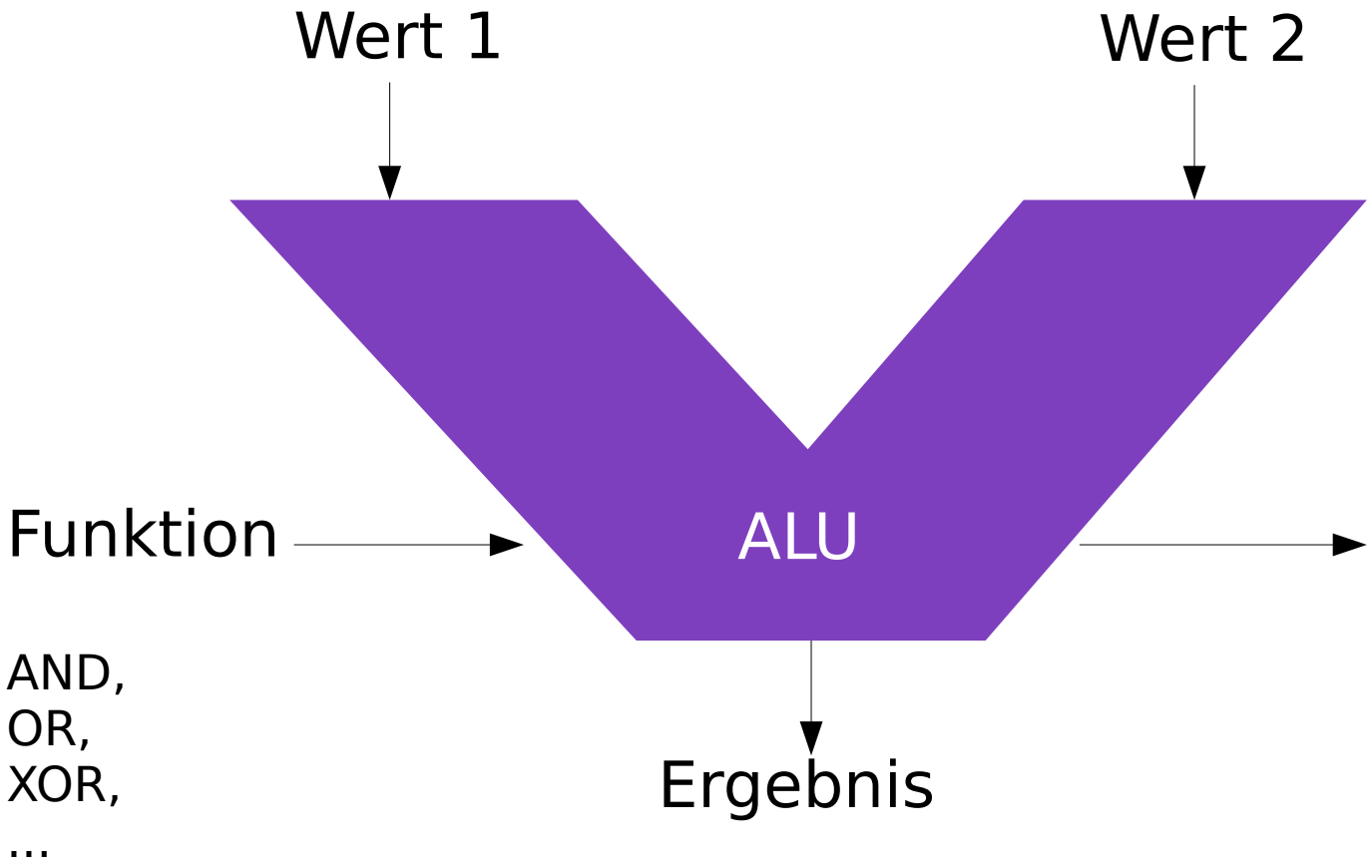
- 131 Instruktionen
- Bis zu 16 MHz
- 16 Kbyte Programmspeicher
- 1 Kbyte RAM
- 512 Bytes EEPROM
- Diverse Peripherie

Bestandteile eins Mikrocontrollers

Der Kern: Die ALU I

- Steht für Arithmetisch-logische Einheit
- “Rechnerkern”
- Komplexer und vielschichtiger Aufbau
- Führt logische Operationen auf Binärwerte aus
- Mögliche Operationen AND, OR, XOR

Der Kern: Die ALU II



Demo

Der Kern: Register I

Problem: Werte können nicht einfach aus dem Nichts kommen !!!

- Wir benötigen eine Möglichkeit die Werte für die Verarbeitung zu speichern

- Register sind die Lösung

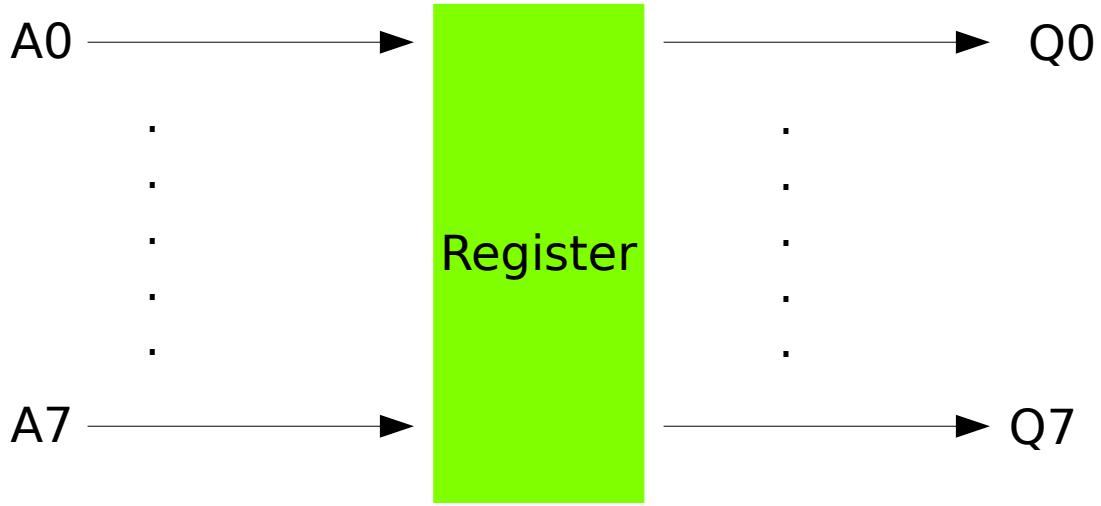
Def.: Ein Register ist eine einzelne Speicherstelle für ***n*** Bits

Für uns git n=8 → 8Bit Mikrocontroller

Der Kern: Register II

- ALU, RAM und alles Andere passt sich diesen 8 Bit an

- Warum?



- Register sind groß im Chip

- Brauchen viel Strom

Kleine Register sind günstiger

Demo

Der Kern: Register III

- So wenig wie möglich Register wünschenswert
- →Informationen werden gepackt

In einem Register können manche Bits als Zahlen und andere als einzelne Bits interpretiert werden

Der Kern: Register IV

Wir wollen Geschwindigkeit [0-15], Drehrichtung und Start Stop eines Motors speichern

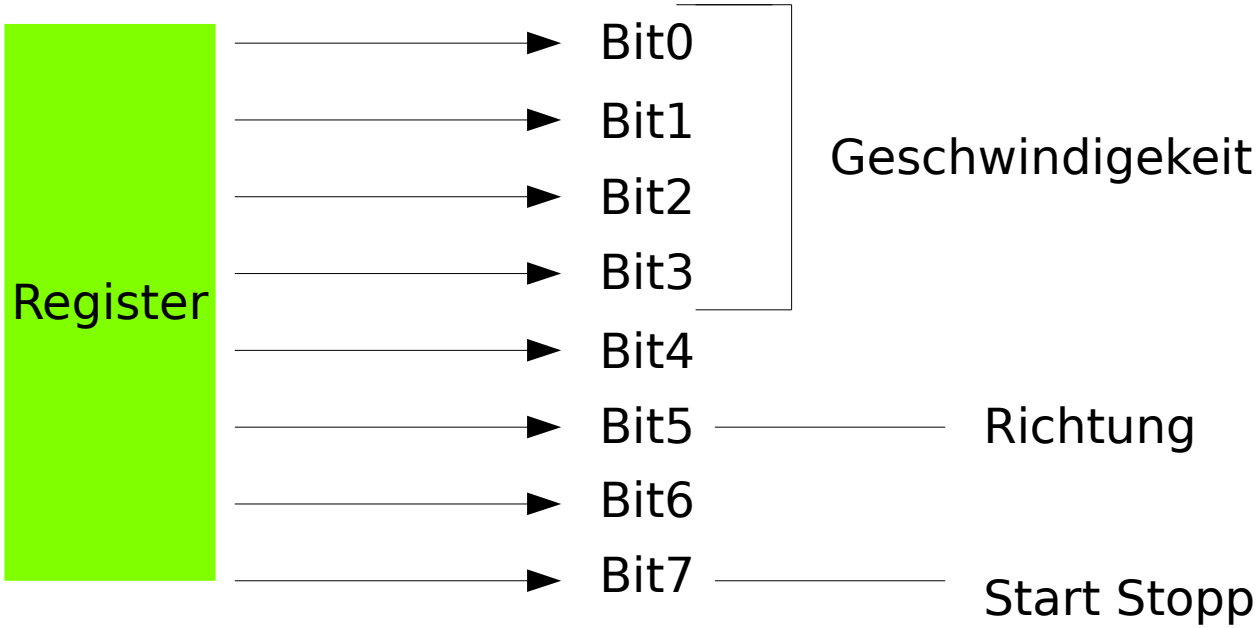
- Drehgeschwindigkeit [0-15] → Braucht 4 Bit
- Drehrichtung → Braucht 1 Bit (0=r, 1=l)
- Start, Stopp → Braucht 1 Bit (0=Stopp, 1=Start)

Naiver Ansatz

Register 1 Geschwindigkeit	Register 2 Drehrichtung	Register 3 Start/Stopp

Der Kern: Register V

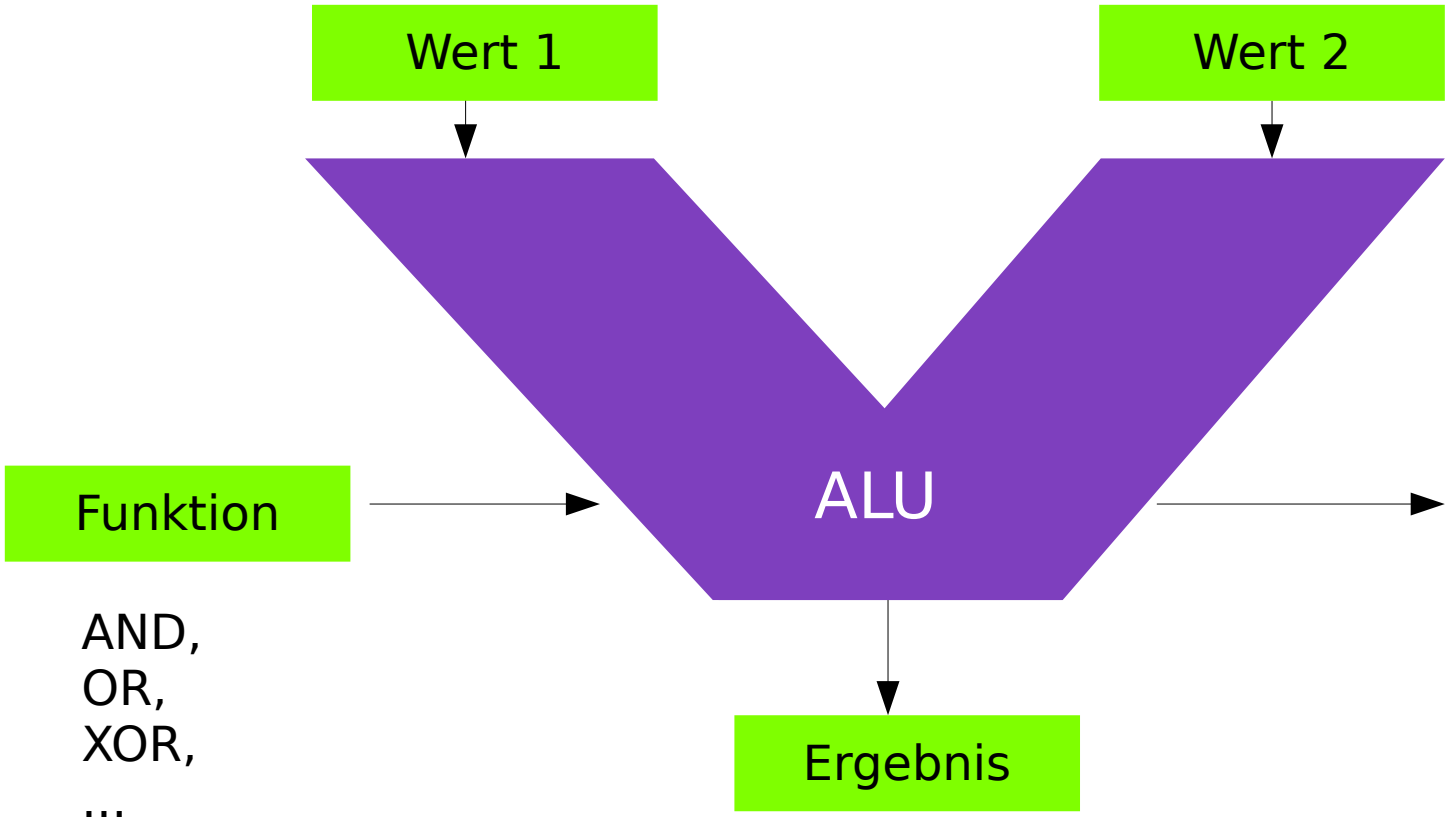
- Besser alles in 1 Register



So sieht ein Großteil des Datenblatt aus

- Wie starte ich den Motor?
- Wie setzte ich die Geschwindigkeit?

Der Kern: Die ALU + Register



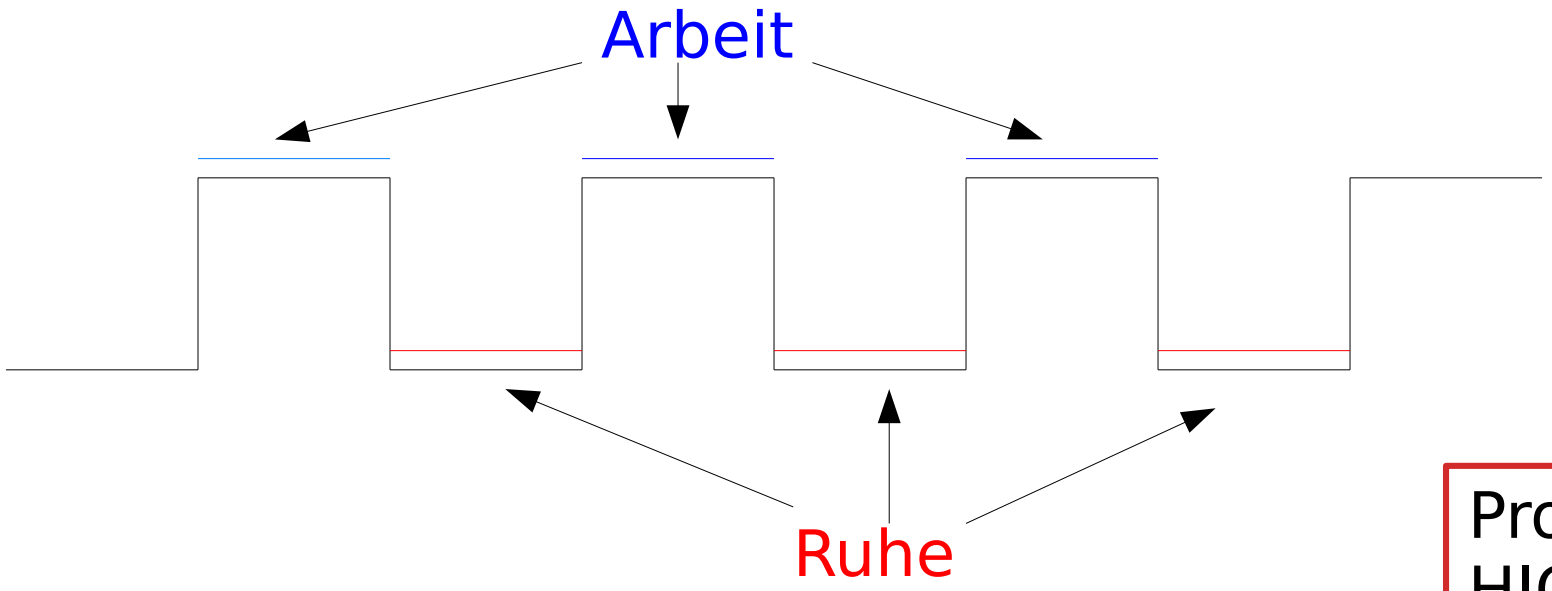
AND,
OR,
XOR,
...

z.B. Ergebnis zu groß,
Ergebnis negativ,
Ergebnis ist 0

Demo

Der Kern: Zeitbegriff I

- Komponenten müssen geziehlt zusammen arbeiten
- Brauchen eine Möglichkeit um Aktionen zu synchronisieren

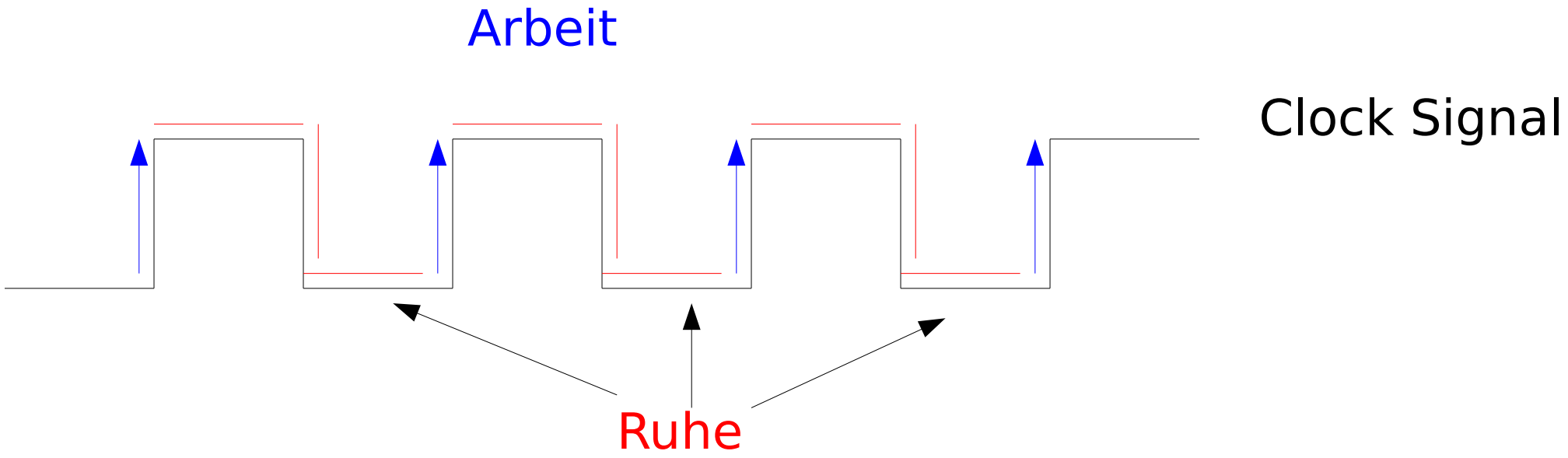


Clock Signal

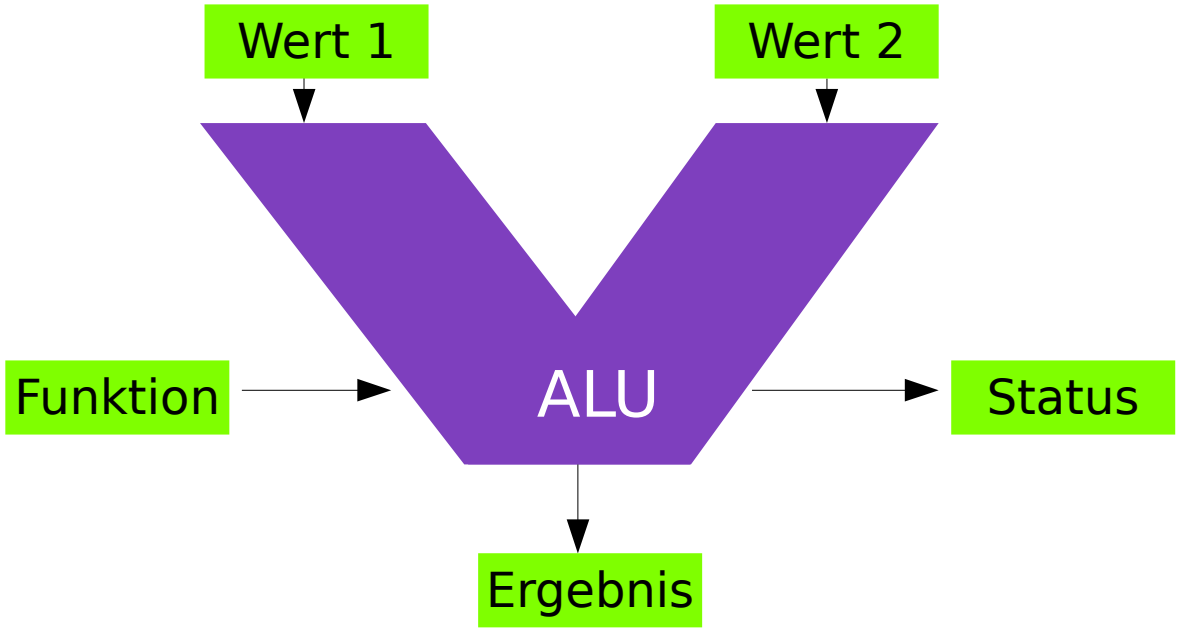
Problem:
HIGH ist lange!!

Der Kern: Zeitbegriff II

- Verschärfung der Bedingung
- Nur noch Zustandswechsel darf gearbeitet werden



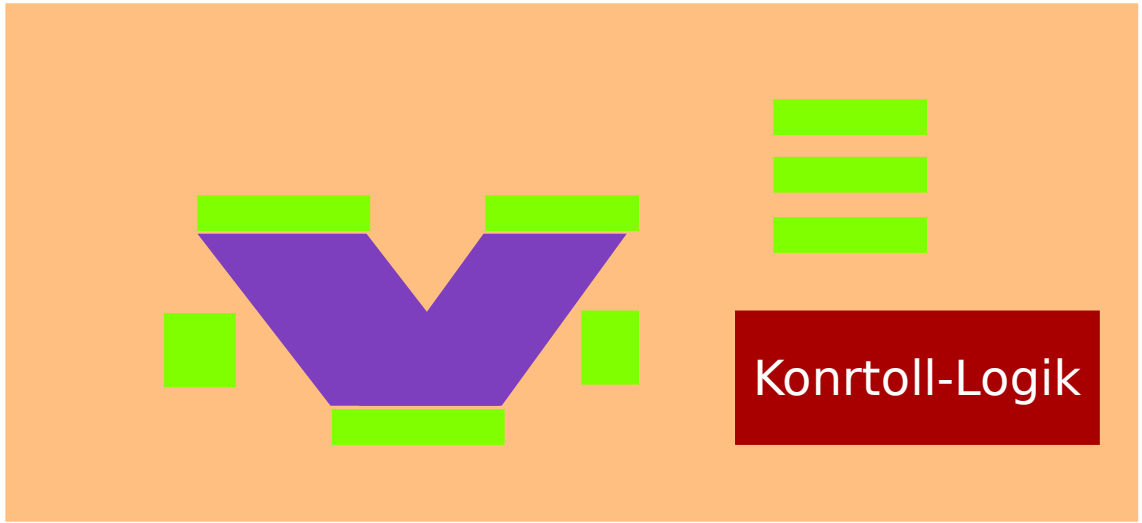
Der Kern: Die ALU + Register



# Takt	Aktion
1	Wert 1 speichern
2	Wert 2 speichern
3	Rechnen
4	Ergebnis speichern

Def. Kern

- Core AVR Familie
- ALU + Register + Kontroll-Logik



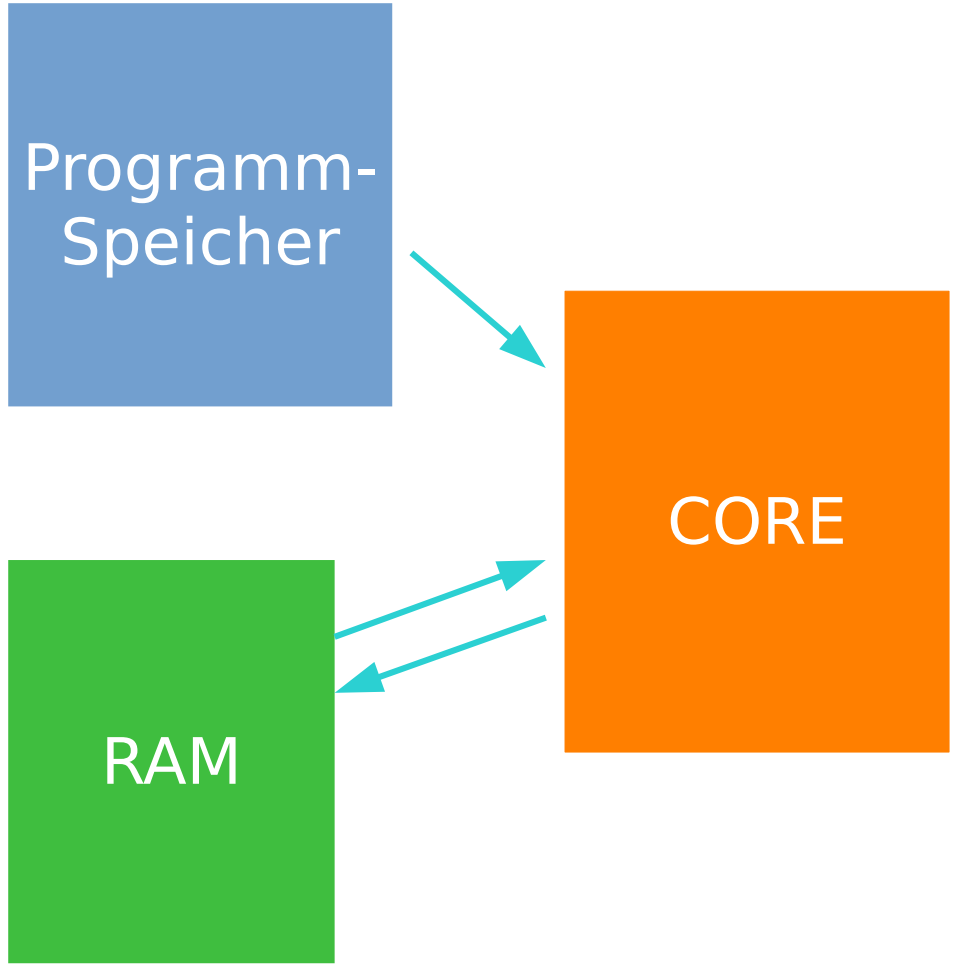
Clock

Core

Speicher

- Enthält das Progamm
- Kann vom MC nur gelesen werden
- Deshalb der Progammer
- nicht flüchtig

- Äquivalent zum RAM im PC
- Beinhaltet Variablen und andere Daten
- flüchtig



Was macht MCs besonders

- Bis jetzt kann der Controller nur rechnen
- Wir wollen Interaktion mit der Außenwelt
- Controller bekommt Peripherie

Def.: Peripherie

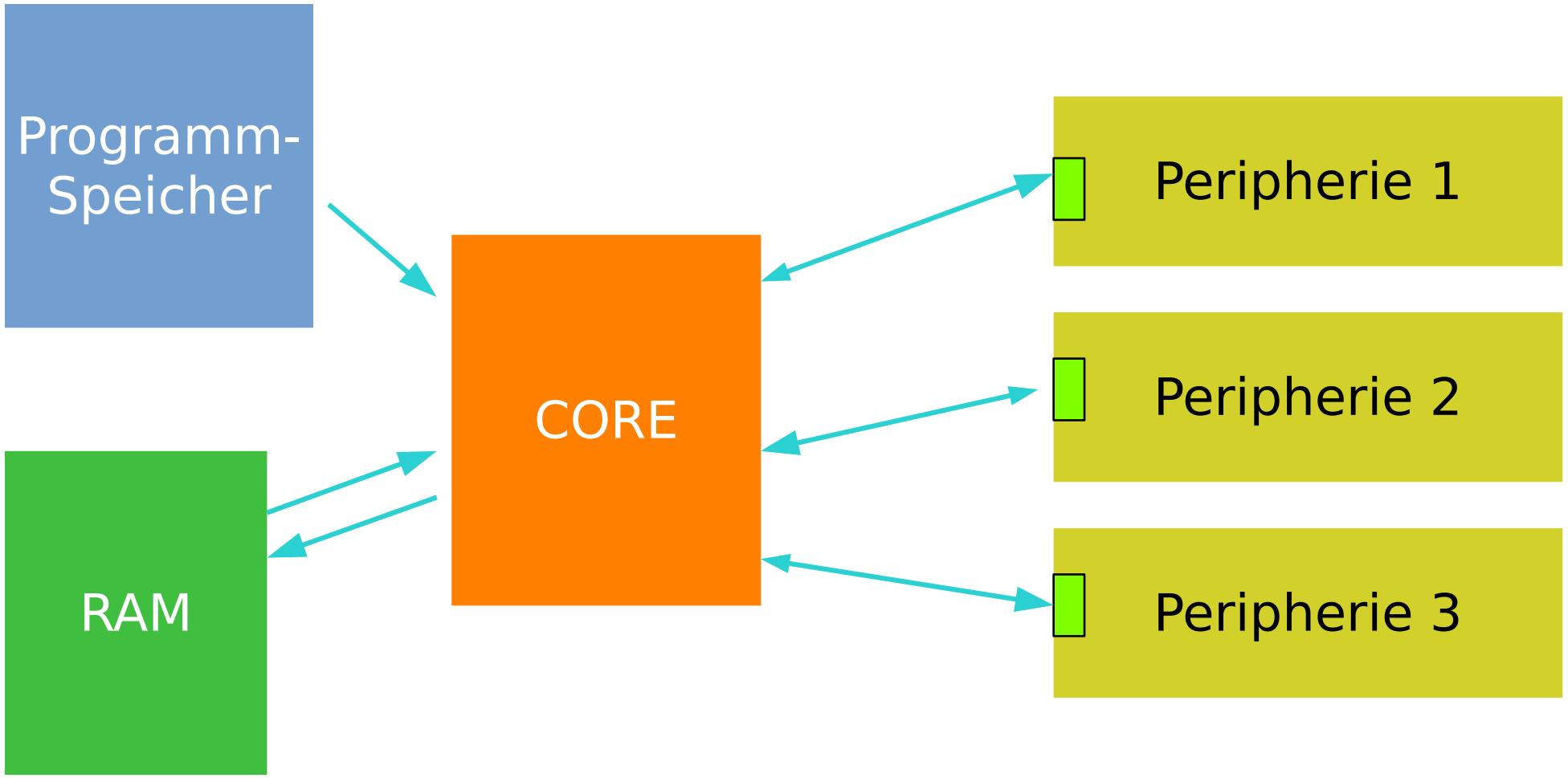
Eigene Funktionseinheiten die eine Aufgabe erfüllen und vom Core gesteuert werden

Peripherie Ausstattung ist der größte Unterschied zwischen verschiedenen Controllern

Wie kommuniziert der Core mit Peripherie I

- Core und Peripherie teilen sich Register
- Beide können darauf lesen und teilweise schreiben
- Durch spezielle Bits und Bit-Felder werden Funktionen gesteuert

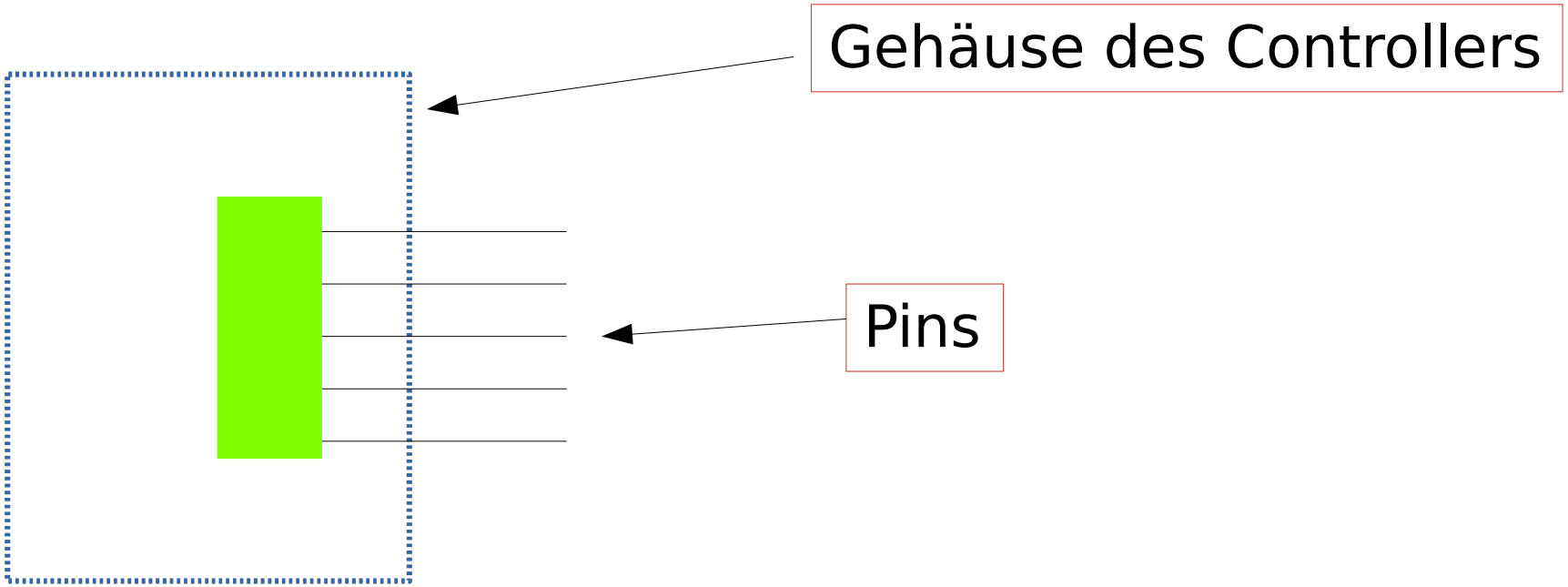
Wie kommuniziert der Core mit Peripherie II



Vorstellung der Peripherie

Digitale IO

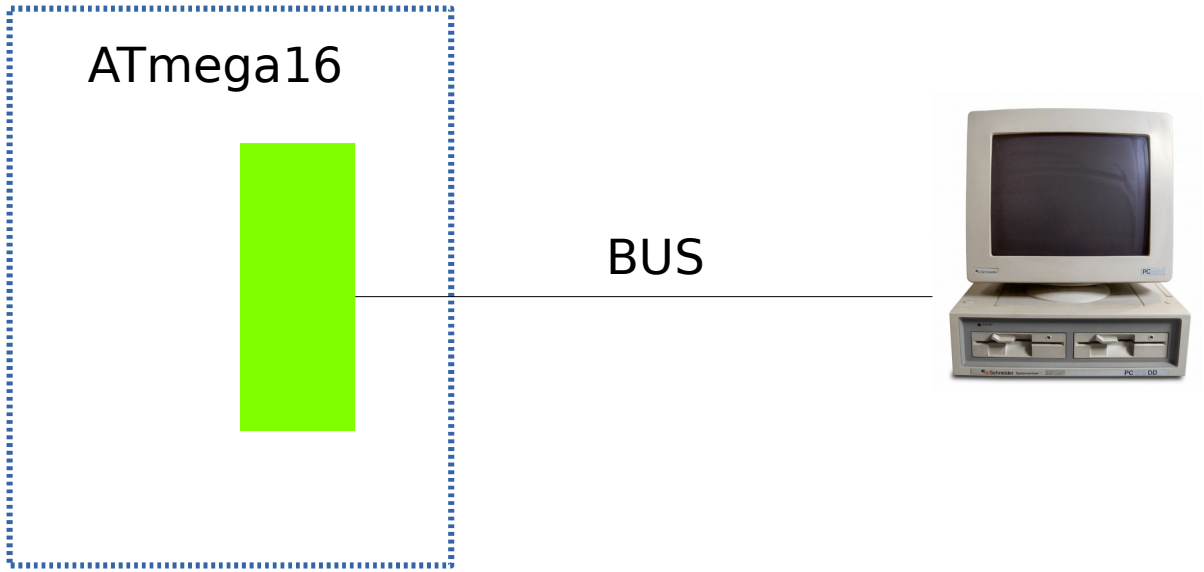
- Simpelste Peripherie
- Schreibt den Zustand eines Register nach außen
- Liest Zustand von Außen ein



Bussysteme

- UART, SPI, I²C

- Ermöglichen Kommunikation mit anderen Geräte z.B. Controller, PCs, Drucker etc.



ADC

- Analog Digital Converter
- Wandelt Analoge Spannung in Zahl um
- Bildet 0-5V auf 0-1023 ab
- Wie viele Bits brauche ich für 1023?
- Sehr nützlich für Sensoren, NTC, Photodiode

Timer/Counter

- Ermöglichen sehr präzises Zählen
- Zeitmessung
- Erzeugen von Waveforms

Case Study

Case Study I

Beispiel Auftrag:

Wir sind Ingenieur bei der RWTH Aachen und sollen einen Controller für eine Türschließanlage bauen.

Vorgaben:

- Muss mit Z80 realisiert werden
- 8 Taster um Türen zu entsperren
- 8 LEDs für die Tür Anzeige
- 1 Schnittstelle zu einem Verwaltungs-PC

Case Study II

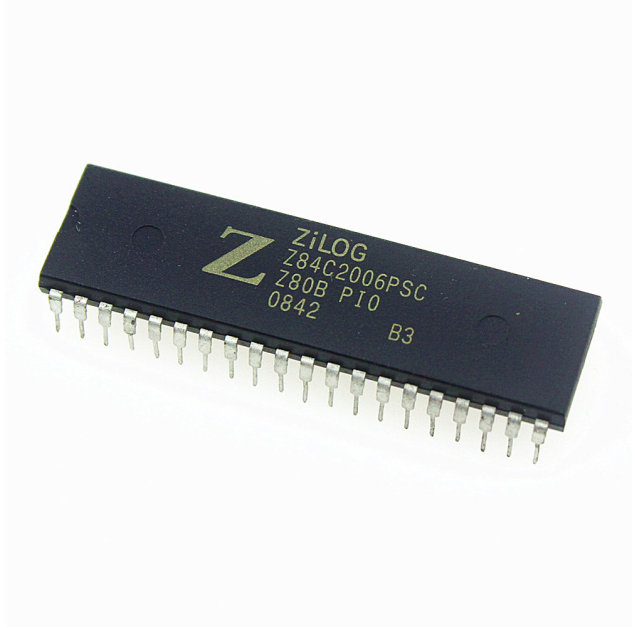
- Zilog Z80 Beispiel für 8-Bit CPU wie Intel 8080 oder Motorola 6800
- Konnte durch Peripherie um Funktionen erweitert werden
- IO / Bus / Timer etc.
- Viele Chips + Beschaltung → große Platinen → teuer

Case Study III

Unsere Bauteile

- CPU
- Programmspeicher
- Arbeitsspeicher
- Input Ouput Baustein
- Baustein für Schnittstelle
- Umschaltung

Case Study IV



CPU

Programmspeicher

PC-Schnittstelle

Arbeitsspeicher

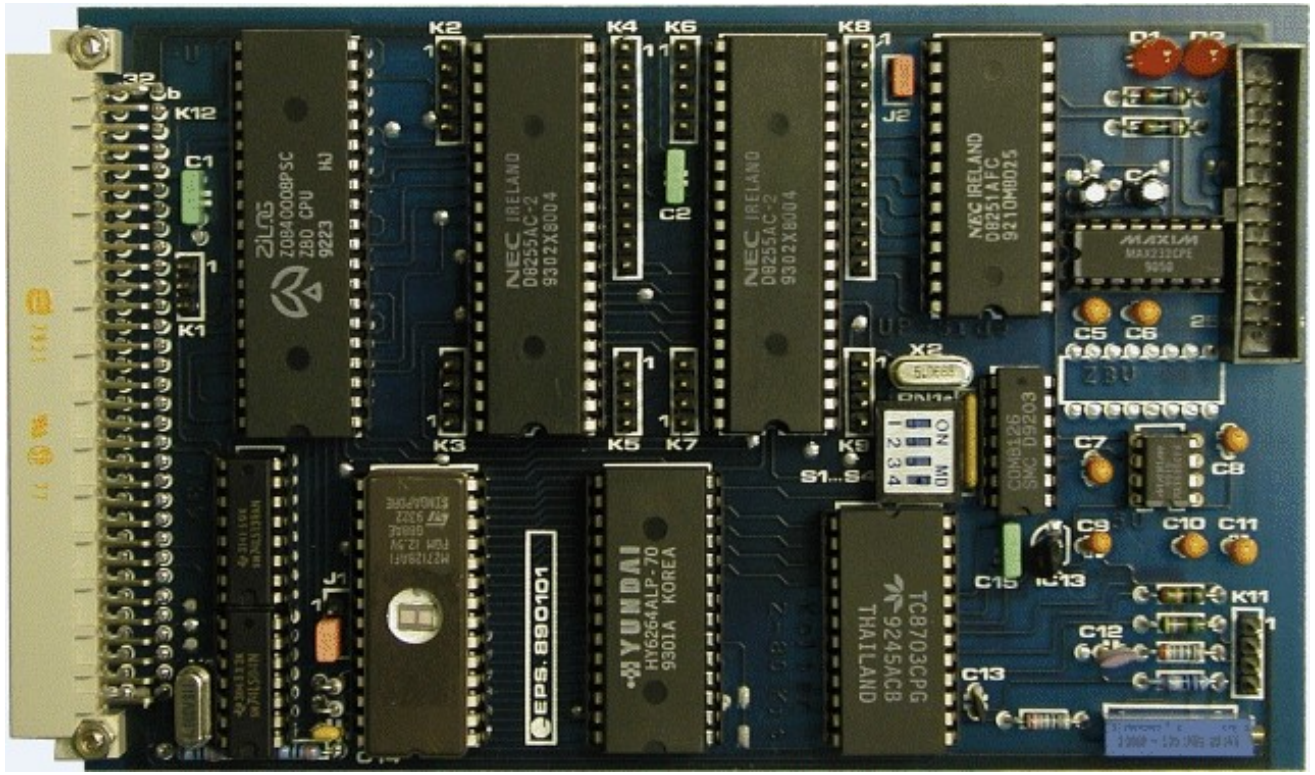
Tür-Schnittstelle

Case Study V

Unser Ergebnis

- groß
- teuer
- Viele Chips sind schwierig zu bekommen

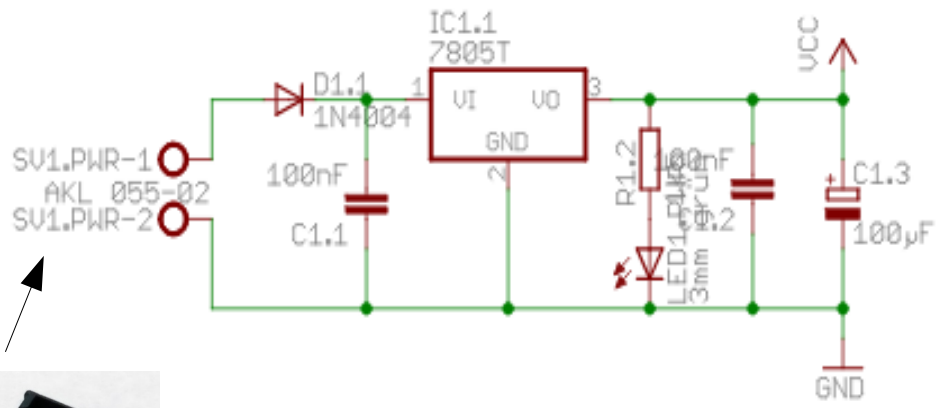
Aus dieser Not wurden MCs geboren



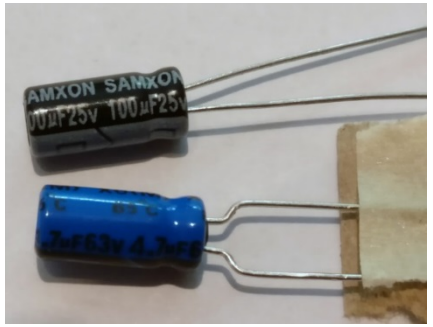
Unser Bausatz

Unser Bausatz I: Power

Regelt Spannung auf 5V



Glätten der Spannung



Anschluss für 9V Batterie Block

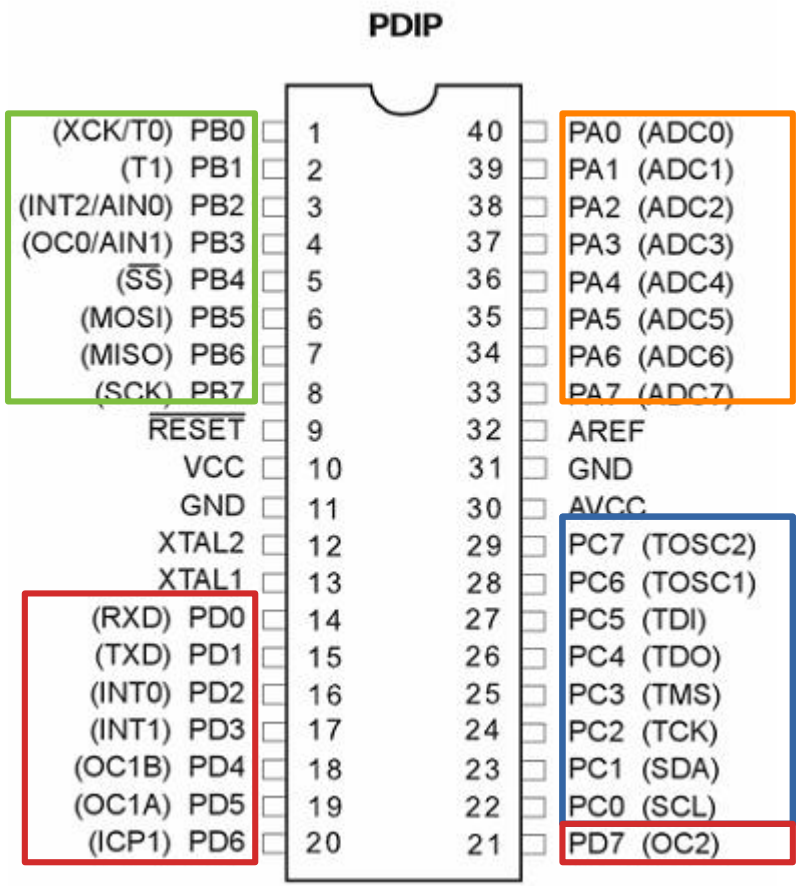
Unser Bausatz II: Pinout

Def: Ein Pin ist eine Schnittstelle des Controllers zur Außenwelt

- Pins können mehrere Funktionen besitzen

Def: Ein Port ist die Einheit aus 8 Pins

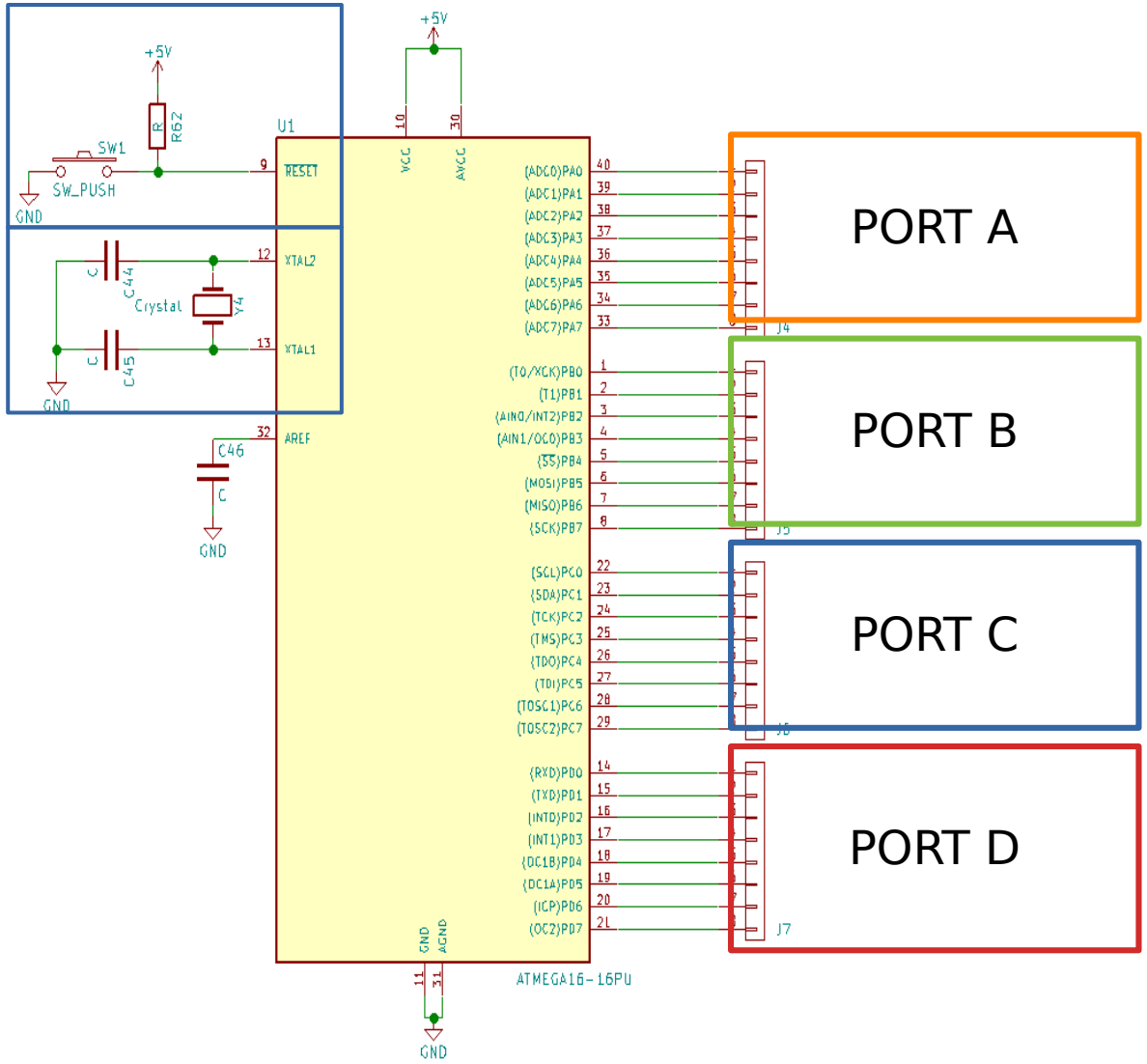
- PORTA, PORTB, PORTC, PORTD
- Pins heißen P[A-D][0-7]



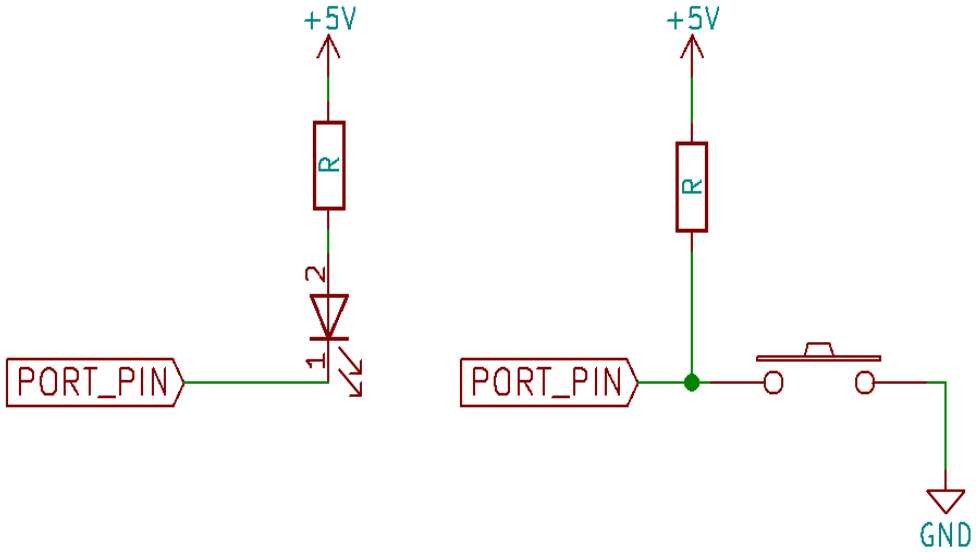
Unser Bausatz III: AVR Beschaltung

Reset Schaltung
Setzt Mikrocontroller zurück

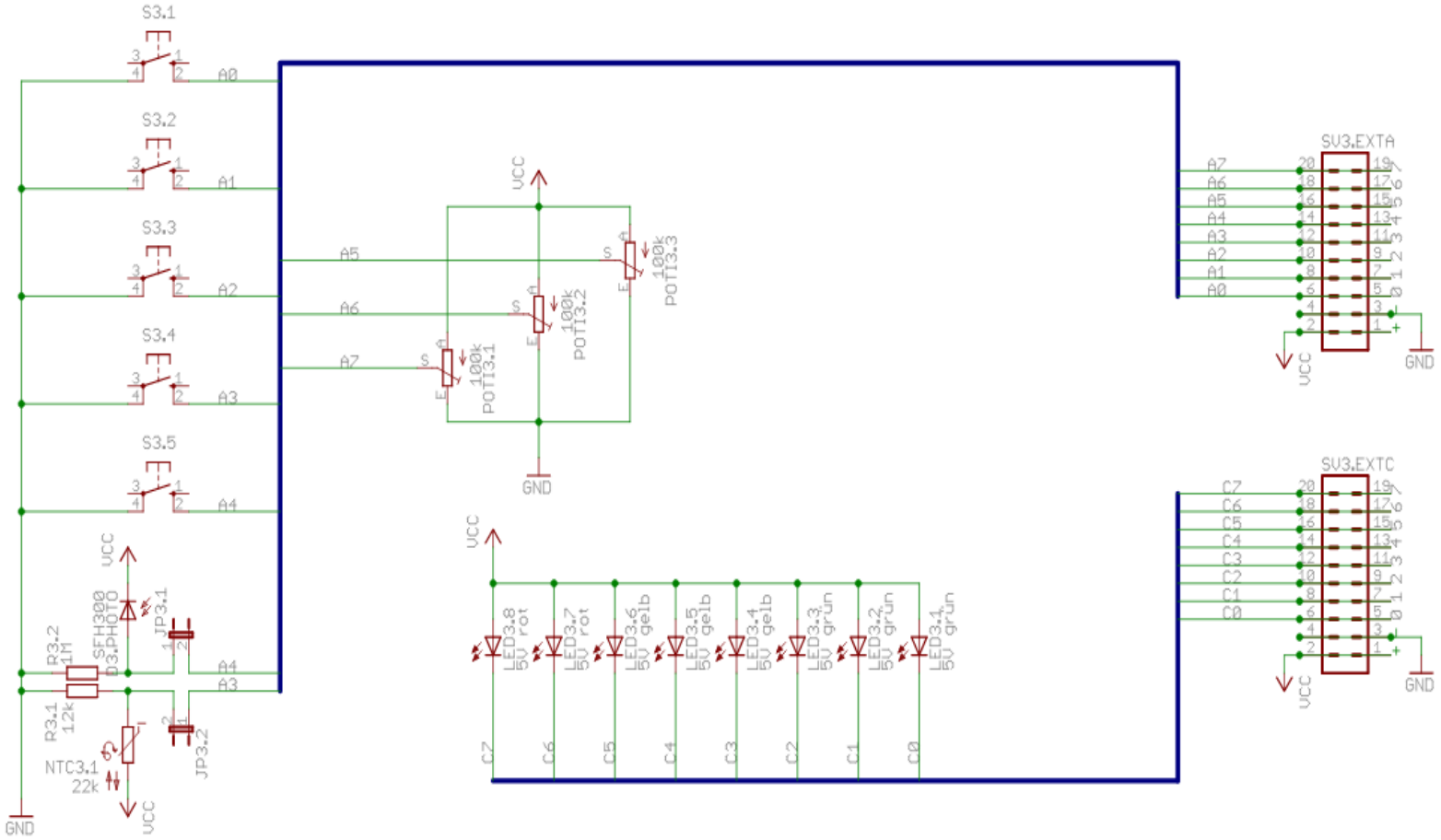
Schwingschaltung
Generiert den Takt



Unser Bausatz IV: Spielwiese

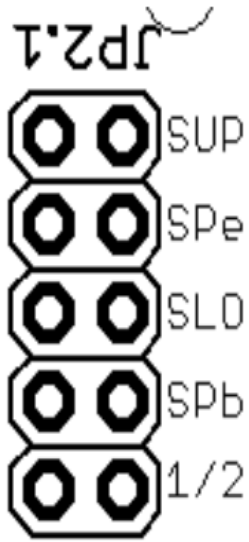


Unser Bausatz IV: Spielwiese



Unser Bausatz V: Jumper

Jumper können wie folgt gesetzt werden:



SUP (Supply)	Wenn dieser Jumper gesetzt ist, wird das an den Programmer angeschlossene Board mit 5V Spannung versorgt.
SPe (Self-Programm-Extern)	Muss gesetzt sein, wenn man den Programmer über einen anderen Programmer flashen möchte.
SLO (Slow)	Wenn dieser Jumper gesetzt ist, wird die Taktfrequenz beim flashen gedrosselt. Notwendig für einige kleine Mikrocontroller (z.B. Atmega8).
SPb (Self-Programm-Bootloader)	Muss gesetzt sein, wenn man den Programmer über den Bootloader (USB) flashen möchte. (Nicht implementiert)
Programmjumper	Leer für Program

C-Kurs

C-Kurs I: Standard Variablen

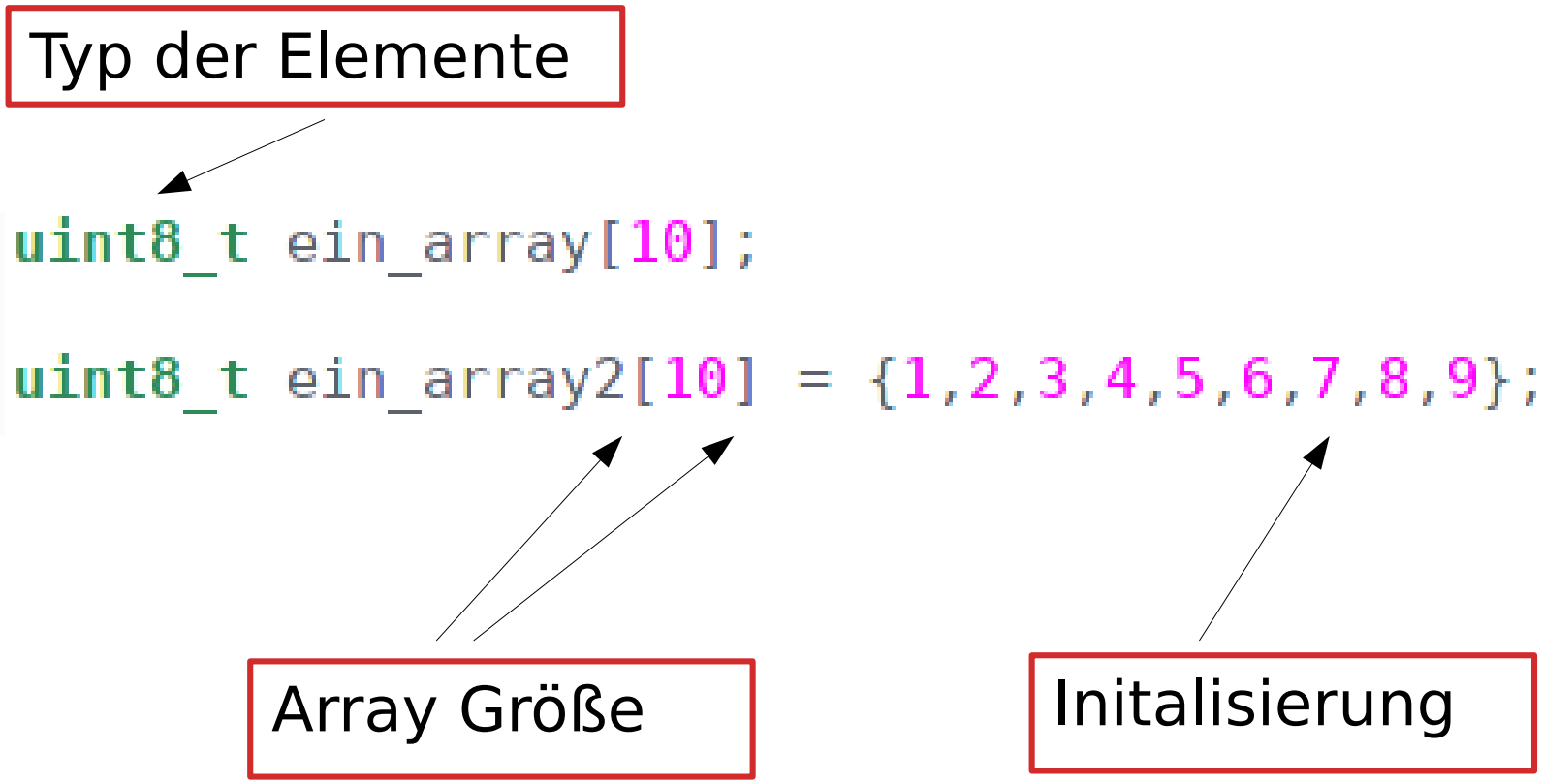
- char — Zeichen / Zahlen -128 bis 127
- unsigned char — Zeichen / Zahlen 0 bis 255
- int — Zahlen -32768 bis 32767
- unsigned int — Zahlen 0 bis 65535
- float — Komma-Zahlen 1.17*10^-38 bis 3.4*10^38
- double — Komma-Zahlen 2.22*10^-308 bis 1.79*10^308
- Kein String!! → Char Arrays “C-Strings”

C-Kurs II: Neue Variablen

- int8_t — Zeichen / Zahlen -128 bis 127
- uint8_t — Zeichen / Zahlen 0 bis 255
- int16_t — Zahlen -32768 bis 32767
- uint16_t — Zahlen 0 bis 65535
- float — Komma-Zahlen 1.17*10^-38 bis 3.4*10^38
- double — Komma-Zahlen 2.22*10^-308 bis 1.79*10^308

Wir werden die neuen Typen verwenden → Besser definiert
Int muss nicht immer 16-Bit sein!!!

C-Kurs III: Arrays



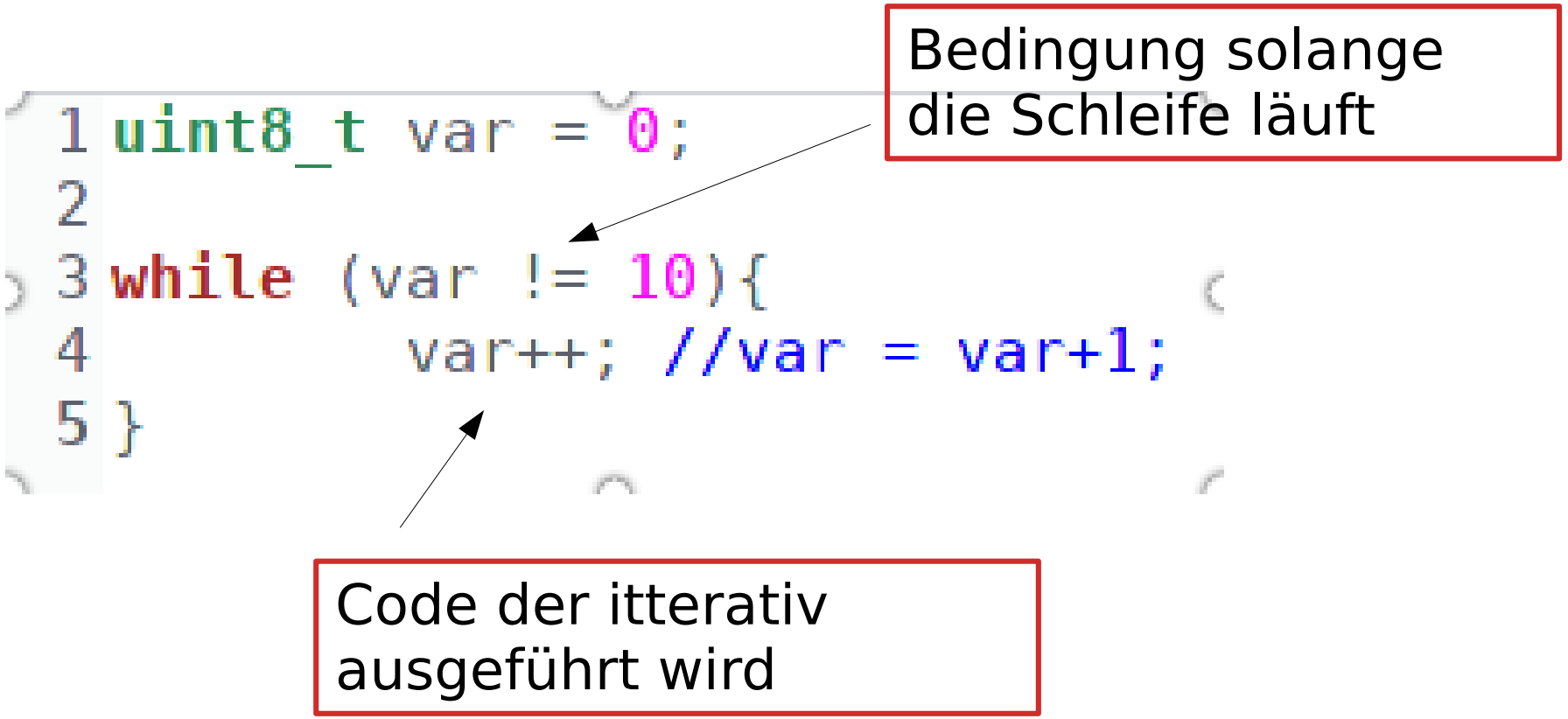
C-Kurs IV: Operatoren

- Arithmetische Operatoren +, -, *, /
- Binäre Operatoren
  - Und &
  - Oder |
  - XOR ^
- Shifts <<, >>
- Kombiniert mit =
  - +=, -=, ... <<=, >>= , &=, |=

C-Kurs V: if - else

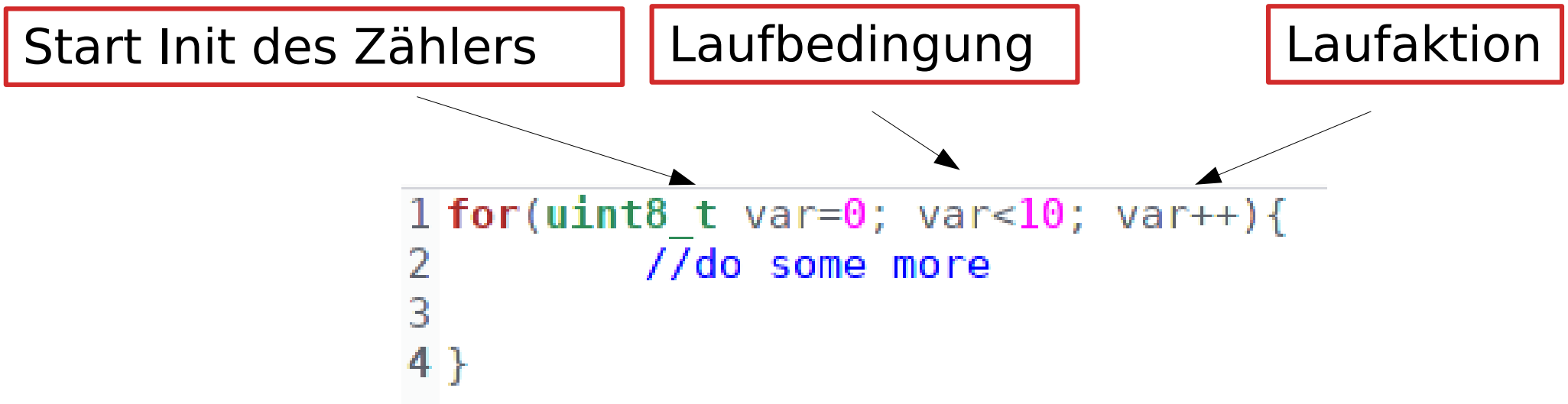
```c
uint8_t var = 1;

if(var == 1){
        //do something
}else{
        //do something else
}

Was passiert hier?
if(var){
        //do something
}
```

C-Kurs VI: while



C-Kurs VII: for

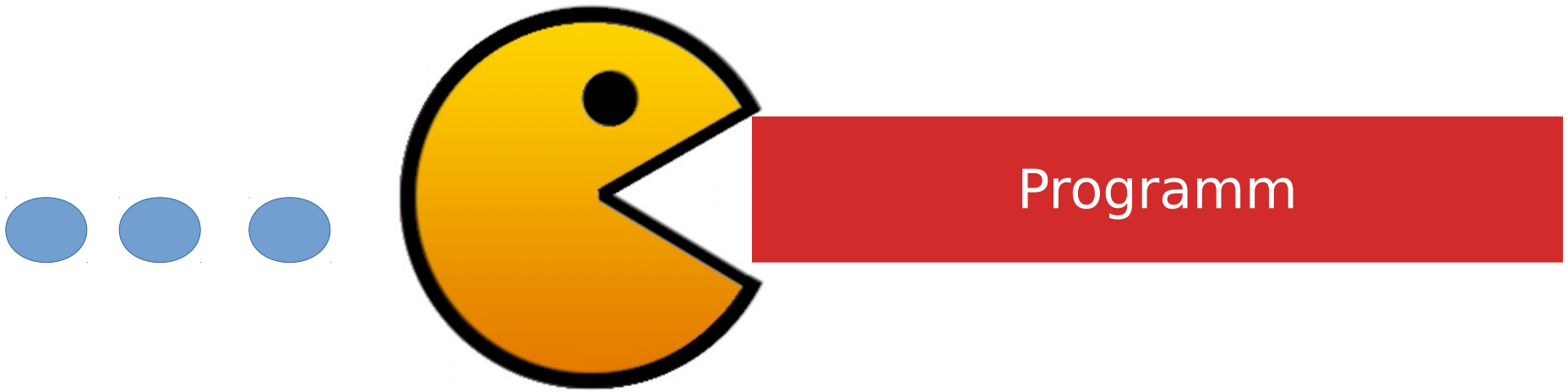
- Zählschleife → Wie While nur das immer gezählt wird in welchem Durchlauf wir sind



C-Kurs VIII: AVR Libc

- AVR Libc ist die Standard Lib für AVRs in C
- Liefert alle Definitionnen von Register #include <avr/io.h>
- Liefert Hilfsfunktionen wie Warteschleifen #include <util/delay.h>
  - _delay_ms(x) wartet x msecs
  - _delay_us() wartet x usecs
  - Beide werden zu Compilezeit aufgelöst

C-Kurs IX: Standard Programm



C-Kurs IX: Standard Programm

C-Kurs IX: Standard Programm

```
 3
 4   #include <avr/io.h>
 5   #include <util/delay.h>
 6
 7   #define F_CPU 14745600
 8
 9   int main(void)
10   {
11       //Initialisierung
12       while(1){
13           //Endlos-Schleife
14       }
15
16       return 0;
17   }
18
```