Technische Hochschule Rosenheim
Technical University of Applied Sciences

Embedded Systems
Kapitel 2: Digitale Ein- und Ausgabe, GPIO

Prof. Dr. Wolfgang Mühlbauer
Fakultät für Informatik
wolfgang.muehlbauer@th-rosenheim.de

Sommersemester 2020

Inhalt

- **Technischer Hintergrund**

- Digitale Ein- und Ausgabe beim ATmega2560
  - General Purpose Input / Output (GPIO)
  - Arduino Library and AVR-Libc
  - Bitoperationen, Read-Modify-Write

- Digitale Eingabe: Taster
  - Entprellung
  - Pull-Up / Pull-Down Widerstand

Motivation: Interaktion mit Umgebung

- ***Eingabe***
  - Mikrocontroller ***liest*** Spannung (HIGH/LOW) an Eingangspin.
  - Arduino-Methode: `digitalRead(.)`
  - Beispiele
    - Taster, Schalter
    - Sensoren
    - …

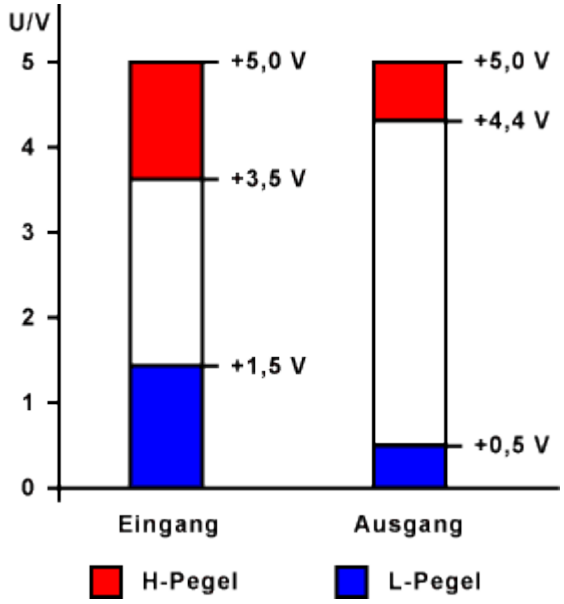
- ***Ausgabe:***
  - Mikrocontroller ***setzt*** Spannung (HIGH/LOW) an Ausgangspin.
  - Arduino-Methode: `digitalWrite(.)`
  - Beispiele
    - LED
    - Motor
    - LCD Displays
    - …

Digitale Ein- und Ausgabe

- ***Digitale Interpretation***
  - Spannungswerte sind prinzipiell "analog".
  - Mikrocontroller unterscheidet nur 2 Spannungsbereiche: ***HIGH*** und ***LOW***!

- ATmega2560 verwendet **CMOS Logik**
  - LOW: ≤ 1,5V, HIGH: ≥ 3,5V

- ***High- vs. Low-Active***
  - ***High-Aktiv***: Signalzustand HIGH bedeutet Vorhandensein des Zustands
    - Beispiel: Pinname "*WR"* ist Hinweis, dass bei HIGH geschrieben wird.
  - ***Low-Aktiv:*** Signalzustand nLOW bedeutet Vorhandensein des Zustands
    - Kennzeichnung durch "Überstreichung"
    - Beispiel: Pinname ($\overline{WR}$) oder */WR* ist Hinweis, dass Schreiben bei LOW erfolgt.

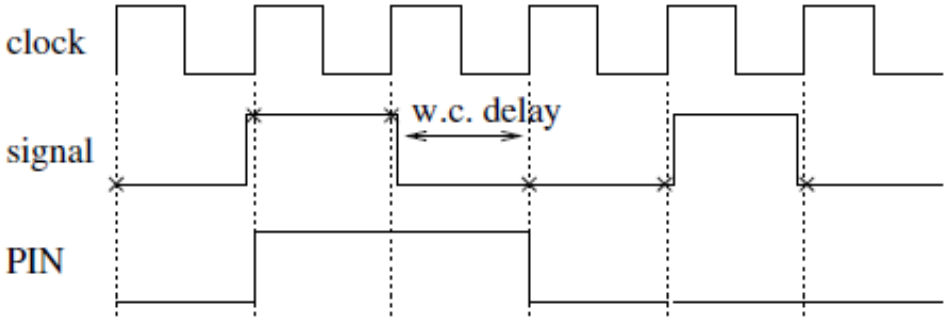


CMOS Logikpegel

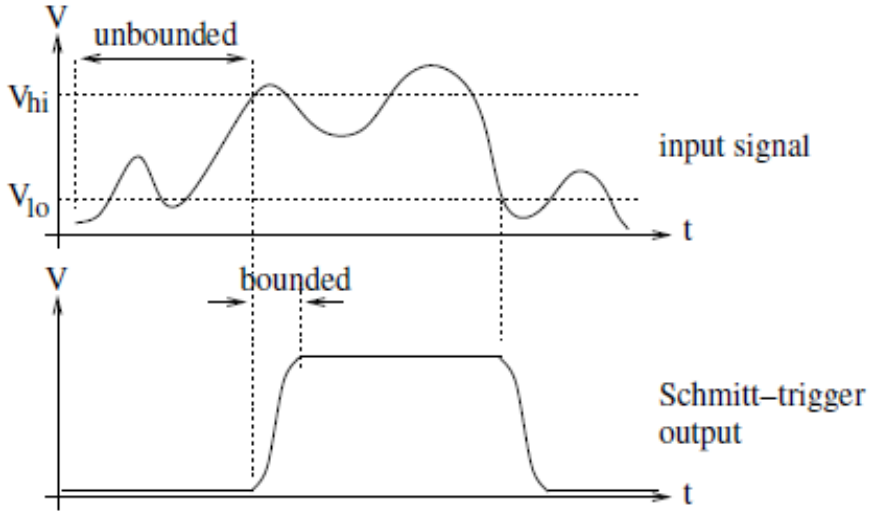
Quelle: [7]

Digitale Eingabe

- **Anforderungen**
  - *Abtastung:* Anliegende Spannung muss periodisch abgefragt werden.
  - Kurzfristige Spannungsschwankungen und *ungültige* Spannungen sollen ignoriert werden.
- Abtastung: ***Taktgesteuerte Flipflops*** lesen den Wert am Eingang z.B. nur bei steigender Taktflanke aus.
- ***Schmitt-Trigger*** eliminieren undefinierte Spannungswerte bei flachen Einfangsflanken und reduzieren Oszillationen.
  - ***Hysterese:*** Separate Ein- und Ausschalteschwelle



Wg. Abtastung werde Änderungen später erkannt oder gar übersehen. [1]



Funktionsweise: Schmitt-Trigger [1]

Inhalt

- Technischer Hintergrund
- **Digitale Ein- und Ausgabe beim ATmega2560**
  - General Purpose Input / Output (GPIO)
  - Arduino Library and avr-libc
  - Bitoperationen, Read-Modify-Write
- Digitale Eingabe: Taster
  - Entprellung
  - Pull-Up / Pull-Down Widerstand

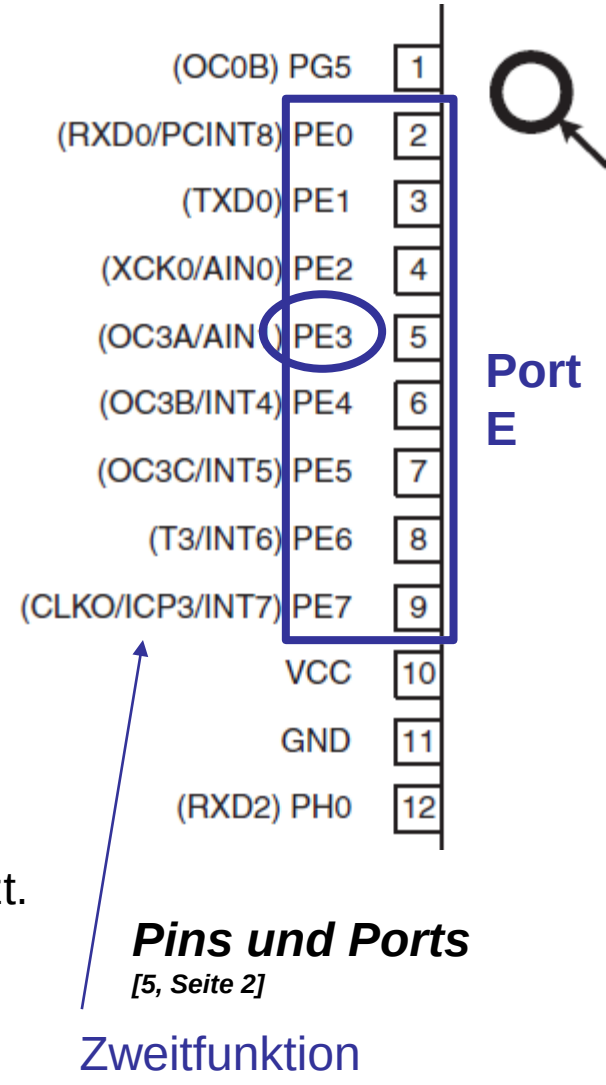
General Purpose Input / Output (GPIO)

- Ein-/Ausgänge ***(„Beinchen“, „Pins“)*** des µControllers sind bidirektional

- Viele µC fassen 8 **Pins** logisch zu 1 **Port** zusammen.
  - Vorteil: 1 Byte genügt um 8 Pins zu konfigurieren.
  - Beispiel: `<"Port"> = 0xA3;`

- ATmega2560
  - 11 Ports: Port A, Port B, …, Port H, Port J, Port K, Port L
  - Jeder Port hat 8 Pins, z.B. PE0, PE1, … PE7
  - Beispiel: **PE3** ist logisch gesehen der **4. Pin** des Ports **E,** also **„Beinchen“ Nummer 3**.

- ***Alternate Function:*** Zweitfunktion von Pins
  - Aus Platzgründen werden Pins für mehrere Zwecke eingesetzt.
  - Bsp: Interrupts, Timer, SPI Bus, usw. [5, S. 72]
  - Muss explizit aktiviert werden, siehe spätere Kapitel.
  - Zweitfunktion der Pins PD2 und PF1?



Zugriff auf GPIO-Pins

- Ports / Pins entsprechen Register.

- Jeder Port (= 8Pins) verfügt über 3 Register
  - ***Data Direction Register DDRx:*** *Datenrichtungsregister*
    - Für jeden Pin, der als Ausgang (Eingang) verwendet werden soll, muss das entsprechende Bit auf 1 (0) gesetzt werden.
    - Beispiel: `DDRB = 0x01` setzt **Pin 0** von **Port B** als Ausgabe.
  - ***Port Register PORTx:*** *Datenregister für Ausgabe*
    - Entsprechende Pins müssen auf Ausgang geschaltet sein.
    - Bit 1 bedeutet +5V, 0 bedeutet 0 V.
  - **Port Input Register PINx:** *Register für Eingabe*
    - Entsprechende Pins müssen auf Eingang geschaltet sein.
    - Bit 1 bedeutet es liegt HIGH an, Bit 0 bedeutet LOW.

- `x` im Namen entspricht `A, B, C,` usw. und gibt den Port an.

Übung:
Wie gibt man auf PB0 +5V aus?

```
DDRB = 0x01;
PORTB = 0x01;
```

DDRB

0	0	0	0	0	0	0	1

PB7 PB0

0	0	0	0	0	0	0	1

Port B

Erhöhung der Lesbarkeit durch Makros

- Bsp.: Kommando `PORTB = 0x10;` bzw. `PORTB = 16;`
  - Setzt Pin #4 des Ports B.
  - Nachteil: Schlecht lesbar.

- Alternative: `PORTB = (1 << PB4)`
  - 4x die 1 nach links verschieben ergibt 16!

- Bitnummern (z.B. PCx, PINCx und DDCx) sind in den io*.h Dateien der avr-libc definiert und dienen nur der besseren Lesbarkeit!
  - Beispiel 1 : `PCx, PINCx` und `DDCx` für PORT C
  - Beispiel 2 : `PDx, PINDx` und `DDDx` für PORT D

```
...
/* PORTC */
#define PC7      7
#define PC6      6
#define PC5      5
#define PC4      4
#define PC3      3
#define PC2      2
#define PC1      1
#define PC0      0

/* DDRC */
#define DDC7     7
#define DDC6     6
#define DDC5     5
#define DDC4     4
#define DDC3     3
#define DDC2     2
#define DDC1     1
#define DDC0     0

/* PINC */
#define PINC7    7
#define PINC6    6
#define PINC5    5
#define PINC4    4
#define PINC3    3
#define PINC2    2
#define PINC1    1
#define PINC0    0
```

Setzen einzelner Bits

```
x |= (1 << Bitnummer);
```

- **Aufgabe:** Setzen der Bitnummer 0 und von 4 von DDRB

- **Variante 1**: Direkte Zuweisung
  - `DDRB = 0x11;`
  - Überschreibt ***alle*** anderen Bitnummern mit 0 → gefährlich!

- **Variante 2**: Binärschreibweise
  - `DDRB = 0b00010001;`
  - Überschreibt ***alle*** anderen Bitnummern mit 0 → gefährlich!
  - Kein ISO-C, wird aber von GNU-C unterstützt.

- **Variante 3:**
  - `DDRB |= (1 << DDB0) | (1 << DDB4);`
  - Idee: Logisch ODER mit den Werten
    - 1 = `0b00000001`
    - 16 = `0b00010000`
  - Zwar länger, aber gute Lesbarkeit!
  - Kein Überschreiben anderer Bitnummern.

Löschen einzelner Bits

```
x &= ~(1 << Bitnummer);
```

- **Aufgabe:** Löschen der Pins 5 und 7 von DDRB

- **Variante 1**: Direkte Zuweisung
  - `DDRB = 0x5F;`
  - Überschreibt ***alle*** anderen Bitnummern mit 0 → gefährlich!

- **Variante 2**: Binärschreibweise
  - `DDRB = 0b01011111;`
  - Überschreibt ***alle*** anderen Bitnummern mit 0 → gefährlich!
  - Kein ISO-C, wird aber von GNU-C unterstützt.

- **Variante 3:**
  - `DDRB &=  ~ ((1 << DDB5) | (1 << DDB7));`
  - Idee: Logisch UND mit negierten Werten.
    - 32 = `0b00100000` → negiert `~` : `0b11011111`
    - 128 = `0b10000000` → negiert `~` : `0b01111111`
  - Zwar länger, aber gute Lesbarkeit.
  - Kein Überschreiben.

Übung

- „Programmieren“ Sie die folgenden Anweisungen
  - Löschen Sie das Bit 3 des Registers `DDRC`.
  - Testen Sie mit einer if-Bedingung ob das Bit 3 des Registers `DDRC` auf 1 gesetzt ist.
- Was macht die folgende Anweisung?
  - `DDRC = 0xFF ^ DDRC`

Arduino Library vs. AVR Libc

- ***AVR Libc***
  - 3 Register pro Port: `DDRx, PORTx, PINx`
  - Registeradressen und Bitnummern sind über Makros vordefiniert
    - → Erhöht Lesbarkeit!
  - Besonderheiten beim ATmega2560:
    - `DDRx` ist Ausgang: Schreiben von 1 in PIN invertiert Bit.
    - `DDRx` ist Eingang: Schreiben in `PORT` (de)-aktiviert Pull-Up Widerstände, siehe später.

- **Arduino Library:** Bietet einfachere Kommandos
  - `pinMode`: Konfiguriert das DDR Registers.
  - `digitalWrite`: Schreibt das PORT Registers.
  - `digitalRead`: Liest das PIN Registers.

Inhalt

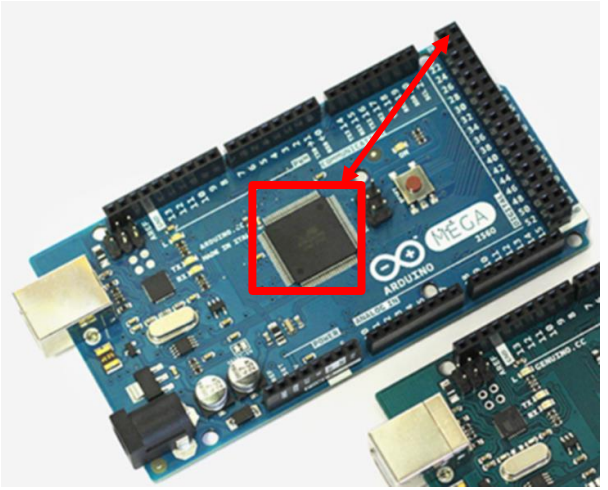
- Technischer Hintergrund
- Digitale Ein- und Ausgabe beim ATmega2560
  - General Purpose Input / Output (GPIO)
  - Arduino Library and avr-libc
  - Bitoperationen, Read-Modify-Write
- **Digitale Eingabe: Taster**
  - Entprellung
  - Pull-Up / Pull-Down Widerstand

Entwicklerboard vs. µController / Pin Mapping

- Wie greift man Pins des µControllers zu?
  - Mikrocontroller ist fest in Entwicklerboard verbaut!

- ***Tabelle / Mapping***
  - Beschreibt welcher Pin der Buchsenleiste mit welchem Pin des Mikrocontrollers leitend verbunden ist.
  - https://www.arduino.cc/en/Hacking/PinMapping2560
  - Diese Tabelle werden wir oft benötigen!

- ***Beispiel:***
  - Pin 1 (Name PG5) des ATmega2560 Mikrocontrollers ist mit Digital Pin 4 auf der Buchsenleiste des Entwicklerboards Arduino Mega verbunden.



Elektrisch leitende Verbindung zwischen Buchsenleiste und Pin des Mikrocontrollers

Pin Number	Pin Name	Mapped Pin Name
1	PG5 (OC0B)	Digital pin 4 (PWM)

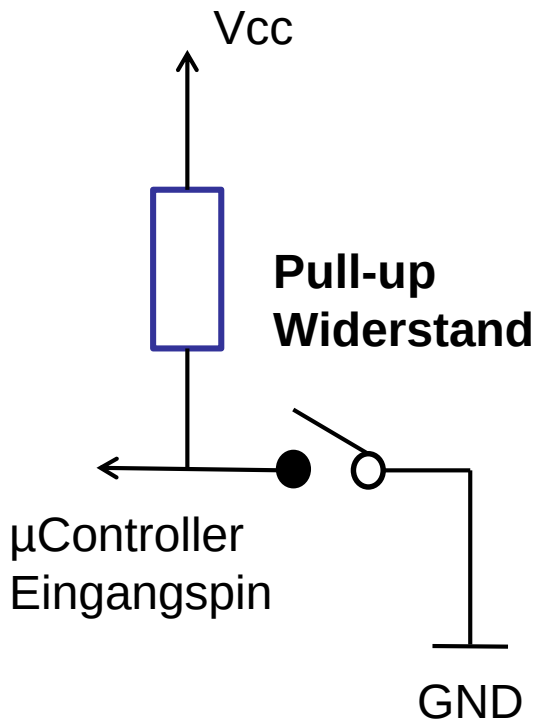
Pinnummer µController (siehe Handbuch)

Pinname des µControllers (siehe Handbuch)

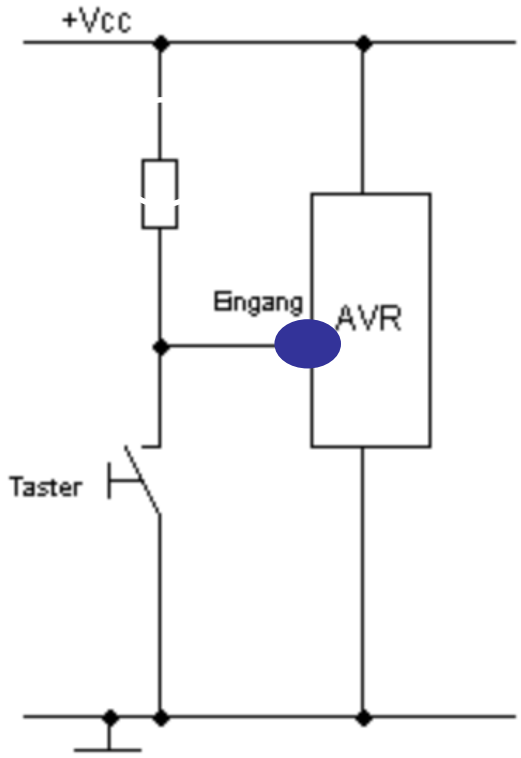
Pin auf Entwicklerboard

Digitale Eingabe: Einlesen eines Tasters (1)

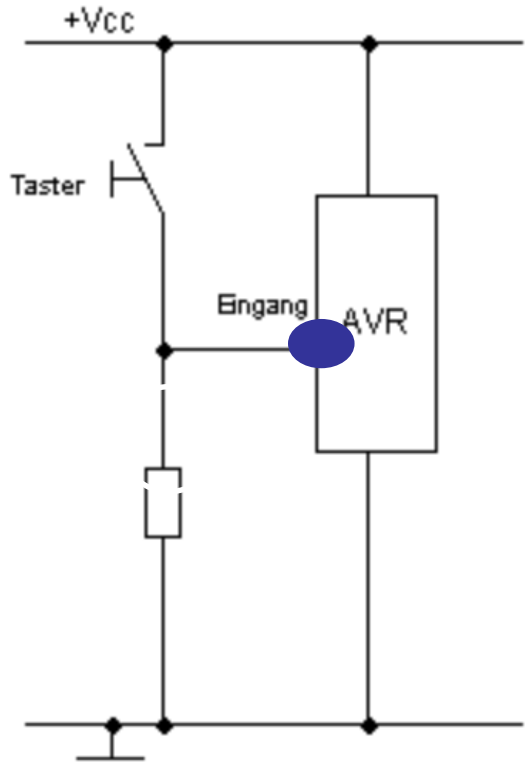
- Gewünschtes Verhalten in Abbildung
  - ***Taster geschlossen:*** *µC* liest 0V / LOW.
  - ***Taster offen:*** *µC* liest HIGH.
- **Problem**
  - Undefinierte Spannung bei ***offenem*** Taster
- ***Lösung***
  - Verbinden des Pins mit der Versorgungsspannung V_{cc}
  - Wichtig: Verwenden eines Widerstandes (***Pull-Up***), sonst Kurzschluss!



Digitale Eingabe: Einlesen eines Tasters (2)



Quelle [3]



- **Pull-Up / Active Low**
  - Bei offenem Taster wird Spannung am Eingang auf HIGH gezogen.
- **Pull-Down / Active High**
  - Bei offenem Taster wird Spannung am Eingang auf LOW gezogen.

Digitale Eingabe: Entprellung

- Vermeintliches, einmaliges Betätigen eines (mechanischen) Tasters:
  - Führt häufig zu mechanischen Vibrationen des Schaltkontaktes.
  - Mikrocontroller realisiert mehrere ungewollte Zustandsänderungen.

- ***Hardware-Lösung***
  - Prellfreie Schalter, Wechselschalter, RS-Flipflops, Kondensatoren, etc.
  - Zuverlässig, aber teuer

- ***Software-Lösung***
  - Künstliche Wartezeit nach Zustandswechsel, in der SW keinen weiteren Zustandswechsel akzeptiert/behandelt
  - Warte so lange bis Schalter eingeschwungen (siehe Übung)

Spannungsverlauf an Eingangspin, der an prellbehaftetem Taster angeschlossen ist.

Quelle [4]

Quellenverzeichnis

[1] G. Gridling und B. Weiss. *Introduction to Microcontrollers*, Version 1.4, 26. Februar 2007, verfügbar online: https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf (abgerufen am 08.03.2017)

[2] https://www.mikrocontroller.net/articles/AVR-Tutorial (abgerufen am 25.03.2019)

[3] Elektronik Kompendium https://www.elektronik-kompendium.de/ (abgerufen am 27.03.2017)

[4] Quelle: http://www.mikrocontroller.net/articles/Entprellung#Warteschleifen-Verfahren (04.04.2016)

[5] Datenblatt ATmega2560, http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf, (abgerufen am 19.03.2017)

[6] http://www.elektronik-kompendium.de/sites/com/0701281.html (abgerufen am 22.03.2018)

[7] https://www.elektronik-kompendium.de/sites/dig/0205171.html (abgerufen am 22.03.2018)