Rechnerarchitektur, Foliensatz 3
Schnittstellen und Zusatzwerke

G. Kemnitz

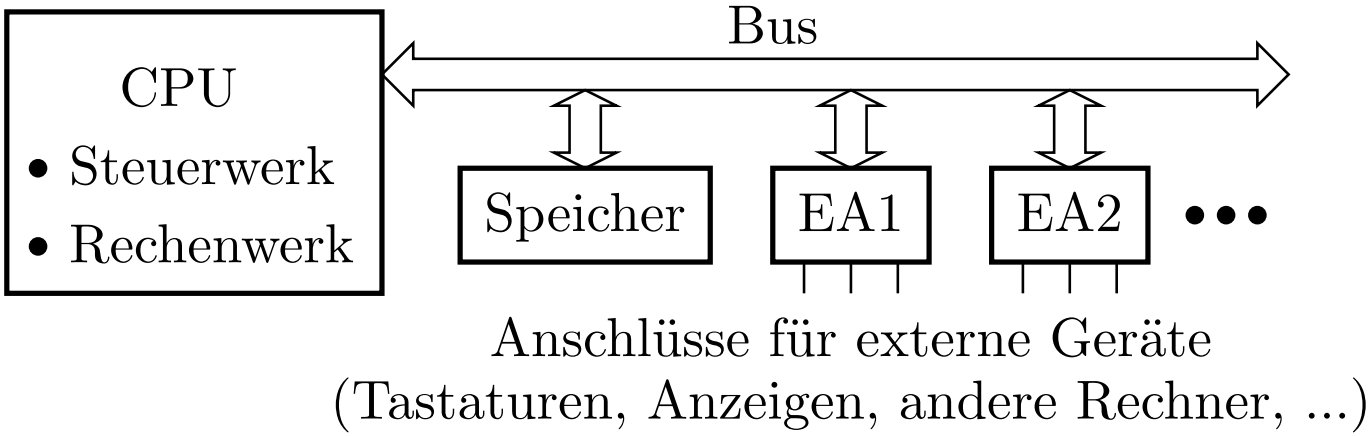
Institut für Informatik, TU Clausthal (RA-F3)

21. Januar 2016

Inhalt RA-F3: Ein- und Ausgabe

Prinzip

Prinzip der Ein- und Ausgabe



Ein Prozessor kommuniziert mit seiner Umgebung

- getrennten Werken (Timer, Watchdog, ...),
- Benutzer, Sensoren, Aktoren, anderen Rechnern, ...

über EA-Register.

Im einfachsten Fall, z.B. beim Lesen von Schalterwerten und der Ausgabe auf LEDs legt das Programm die Lese- und Schreibzeitpunkte fest.

Polling und Interrupt

Zur Abstimmung der Ein- und Ausgabezeitpunkte muss das EA-Gerät warten, bis der Rechner und der Rechner bis das EA-Gerät bereit ist. Dafür gibt es zwei Prinzipien:

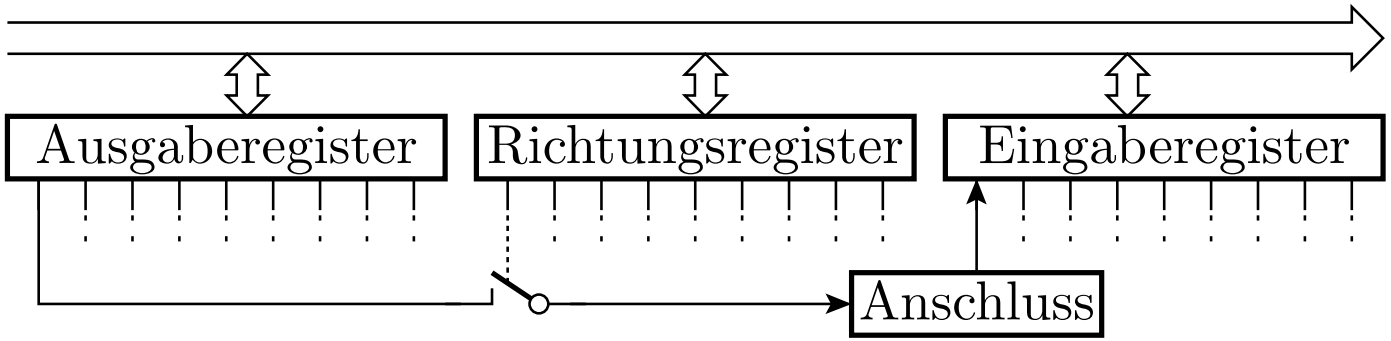
- Polling: Zyklische Abfrage aller EA-Geräte durch den Rechner, ob Datenübergabe angefordert oder zur Übernahme bereit. Wenn ja, Verzweigung zum Programmbaustein für den Datenaustausch.
- Interrupt: Gerät fordert Datenaustausch an. Rechner ruft, sobald er dafür bereit ist, eine Interrupt-Routine auf.

Unterbrechungen sind im Programm global und lokal (für jede Interrupt-Quelle einzeln) freizugeben.

Nach Freigabe kann das externe Gerät das Rechnerprogramm nach jedem Maschinenbefehl unterbrechen.

Ports

Ports (Parallele Schnittstellen)



Ports (Parallel Schnittstellen) sind EA-Register mit direkt anschließbaren Leitungen z.B. für Schalterein- LED-Ausgaben.

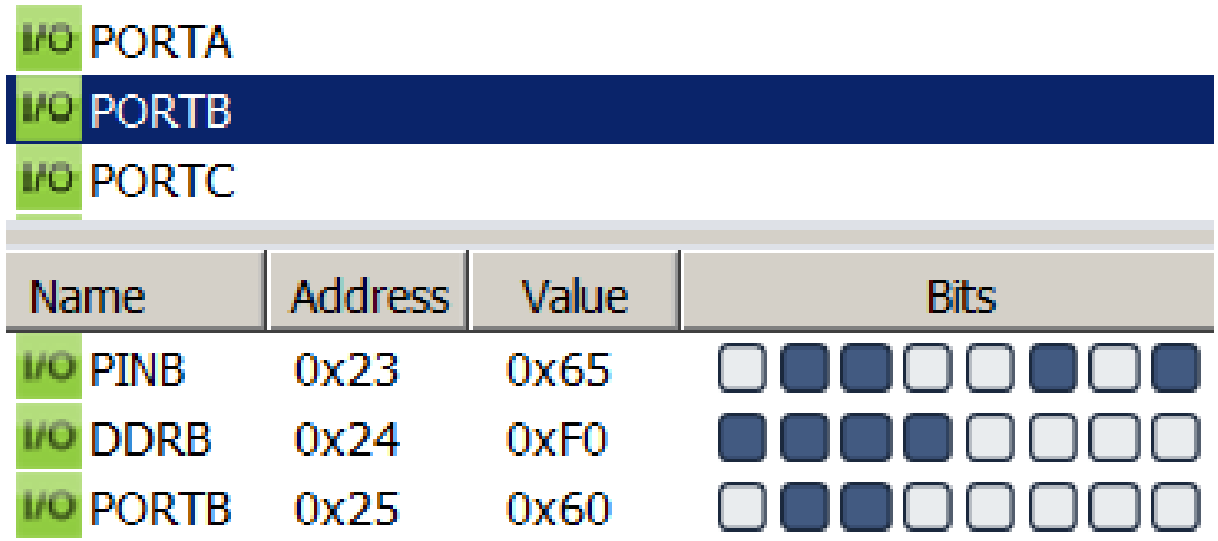
- Datenbreite nach Verarbeitungsbreite des Prozessors 8, 16 oder 32. Überflüssige Port-Anschlüsse bleiben ungenutzt.
- Die Port-Anschlüsse von Mikrorechnern sind in der Regel bitweise als Ein- und Ausgänge oder andere Funktionen (z.B. den seriellern Datenaustausch) konfigurierbar.
- Über Ports können externe Geräte ohne oder mit Synchronisation über Polling oder Interrupt kommunizieren.

Ports des ATmega 2560

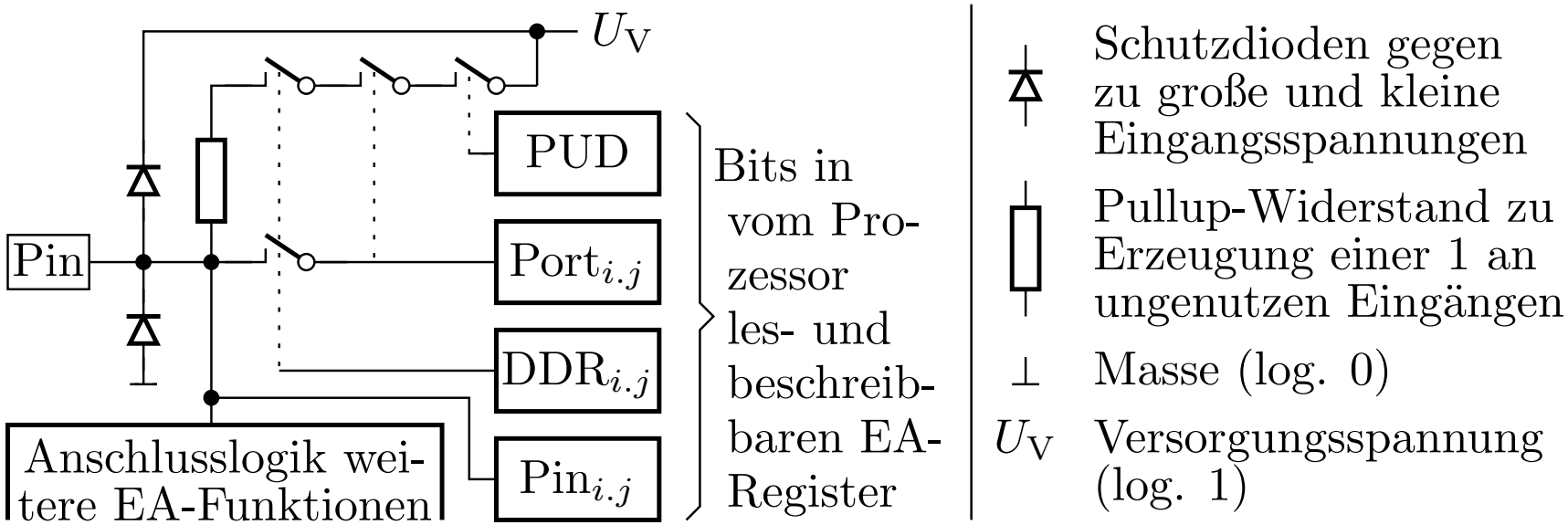
12 Ports mit Richtungsregister DDRx, Ausgaberegister PORTx und Eingaberegister PINx ($x \in \{\mathrm{A}, \mathrm{B}, ..., \mathrm{L}\}$). Adressen:

	A	B	C	D	E	•
PIN	0 / 0x20	3 / 0x23	6 / 0x26	9 / 0x29	0xC / 0x2C	•
DDR	1 / 0x21	4 / 0x24	7 / 0x27	0xA / 0x2A	0xD / 0x2D	•
PORT	2 / 0x22	5 / 0x25	8 / 0x28	0xB / 0x2B	0xE / 0x2E	•

Debug-Ansicht:

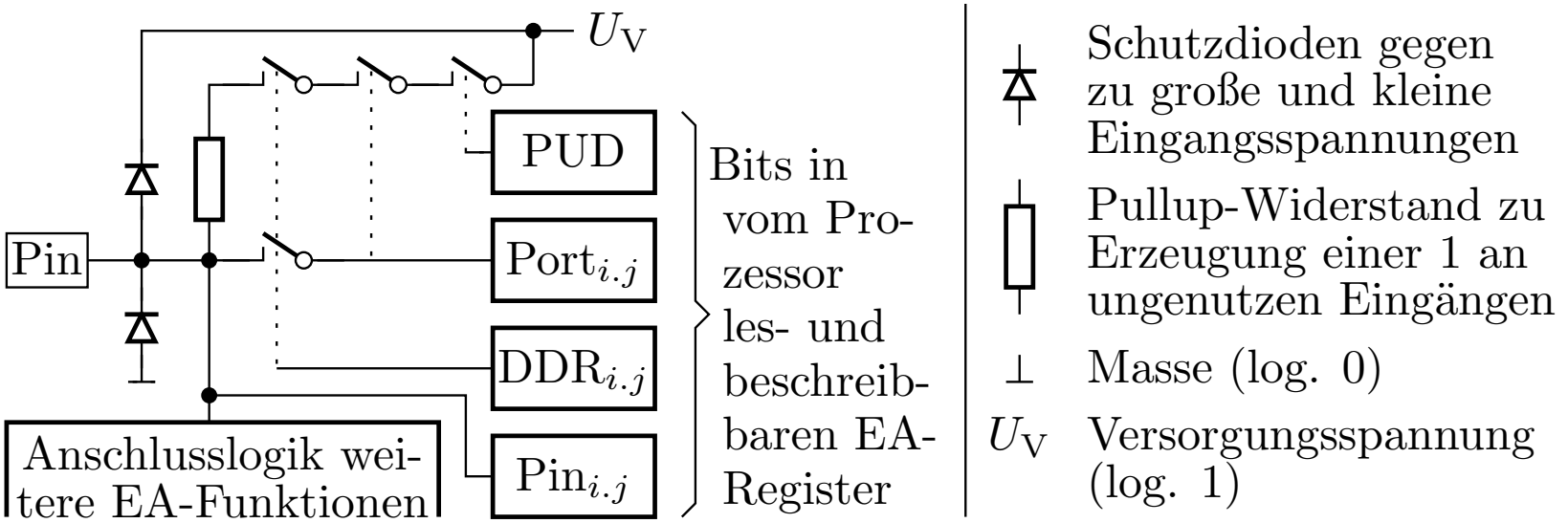


Anschlussbeschaltung im Prozessor



Nutzung als Ausgang:

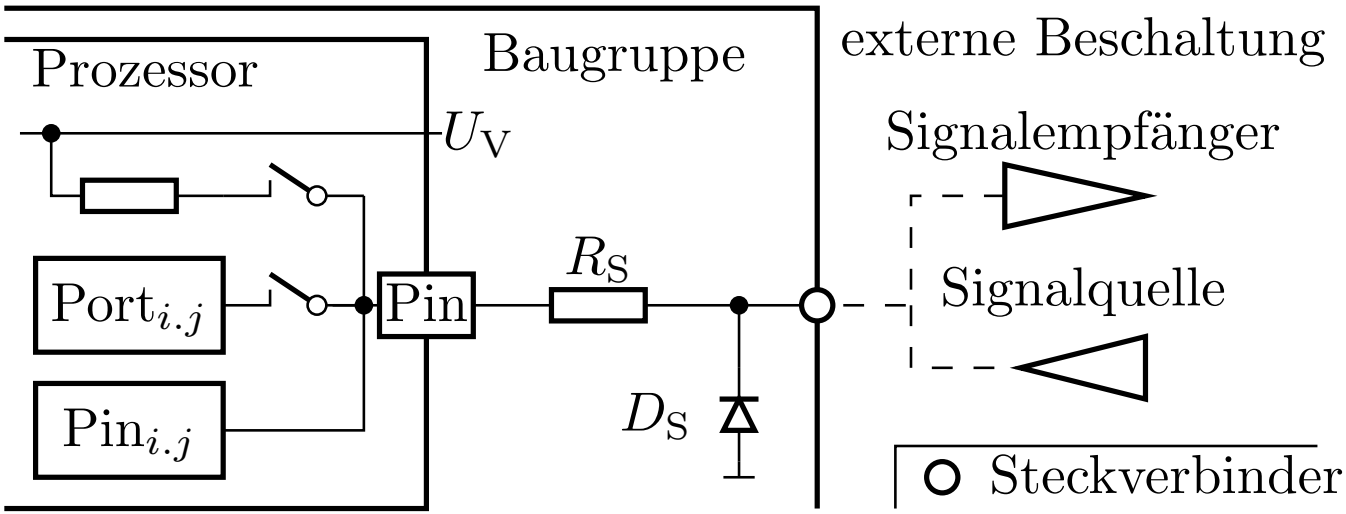
- Richtungsbit eins setzen: DDR$_{i.j} \leftarrow 1$:
- Ausgabe: PORT$_{i.j} \leftarrow$Rd (Rd – Prozessorregister)
- Rücklesen des Ausgabewertes: Rd$\leftarrow$PORT$_{i.j}$
- Eingabe: Rd$\leftarrow$PIN$_{i.j}$. PIN$_{i.j} \neq$PORT$_{i.j}$ ist möglich und deutet auf Programmier- oder Schaltungsfehler.



Nutzung als Eingang:

- Richtungsbit null setzen: $DDR_{i.j} \leftarrow 0$:
- Werte zwischen 0 und 1, z.B. bei ungenutzten Anschlüssen verursachen erhöhte Stromaufnahme.
- Ausgabewert eins ($PORT_{i.j} \leftarrow 1$) und SFR-Bit »PUD« nicht gesetzt, zieht ungenutzte Eingänge über einen Widerstand auf eins. Zu empfehlen für alle ungenutzten Eingänge.
- Bei externer Signalquelle und vor allem für analoge Eingänge Pullup-Widerstand mit ($PORT_{i.j} \leftarrow 0$) deaktivieren.

Port-Beschaltung auf der Versuchsbaugruppe



- R_S (220 Ω) Schutzwiderstand zur Strombegrenzung bei falscher externer Beschaltung oder falsch gesetztem $DDR_{i.j}$.
- D_S Schutzdiode zur Ableitung neg. Eingangsspannungen.
- Für ungenutzte Ports ist die Beibehaltung der Initialwerte $DDR_{i.j} = 0$ und $Port_{i.j} = 0$ wie in den bisherigen Übungen nicht empfehlenswert.

Polling und Interrupts für EA-Ports

Ohne Polling oder Interrupt-Freigabe: Das Programm bestimmt allein, wann Daten ein- oder ausgegeben werden.

Zur Synchronisation bedient das externe Gerät ein Ereignisbit:

- Polling: Prozessor fragt das Ereignisbit zyklisch ab und verzeigt, wenn aktiv[1] zur EA-Operation.
- Interrupt: Eingänge, z.B. Port B als Ereignis- (Interrupt-) Bits nutzbar[2]. Wenn im Programm Interrupts global und für das genutzte Ereignisbit freigeschaltet sind, startet der Prozessor die Interrupt-Routine auf der Adresse, die dem Ereignisbit zugeordnet ist.

[1] »Aktiv« steht für den Wert, bei dem das Gerät bereit ist. Kann bei Polling auch ein beliebiger Wert von mehreren Bits sein.

[2] Mit Interrupt-Controller verbunden. Interrupt-auslösender (aktiver) Bitwert/Flanke programmierbar.

Polling

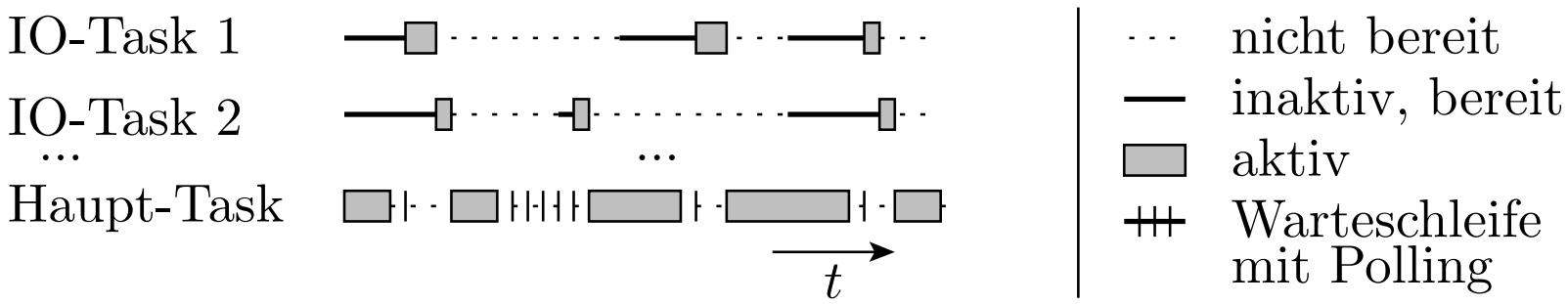
Bedienung von EA-Anforderungen mit Polling

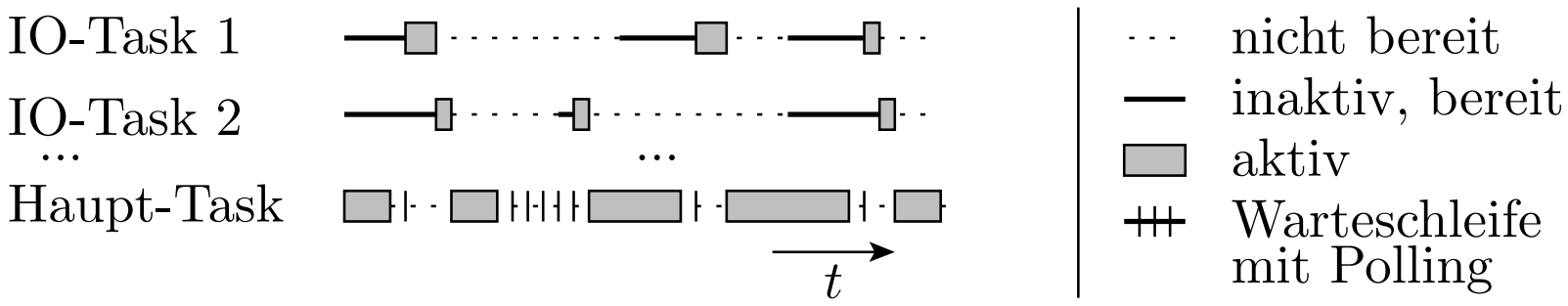
Jede EA-Funktion hat

- i.allg. ein Ereignisbit zur Signalisierung der Bereitschaft,
- eine Obergrenze für die Reaktionszeit vom Setzen des Ereignisbits bis zum Datenaustausch,

in der der Prozessor hundete bis tausende Befehle abarbeiten kann. Programme nutzen meist die Dienste mehrere EA-Einheiten.

Software-Ablauf mit einem Haupt-Task, mehreren EA-Funktionen und Polling:





- Der Haupt-Task arbeitet einige tausend Befehle ab und kehrt zur Haupt- (Ereignisabfrage-) Schleife zurück, z.B. wenn auf EA-Operationen gewartet werden muss.
- Die Hauptschleife fragt zyklisch alle Ereignis- (Bereitschafts-) Bits der aktivierten IO-Tasks und des Haupttasks ab.
- Wenn einer bereit, Verzweigung zur Operationsausführung.
- Am Ende der Operationsausführung löscht der Task das eigene und setzt die Ereignisbit der Tasks, die auf diesen gewartet haben + Rückkehr zur Hauptschleife.

Polling funktioniert gut für EA-Operationen mit großen zulässigen Reaktionszeiten.

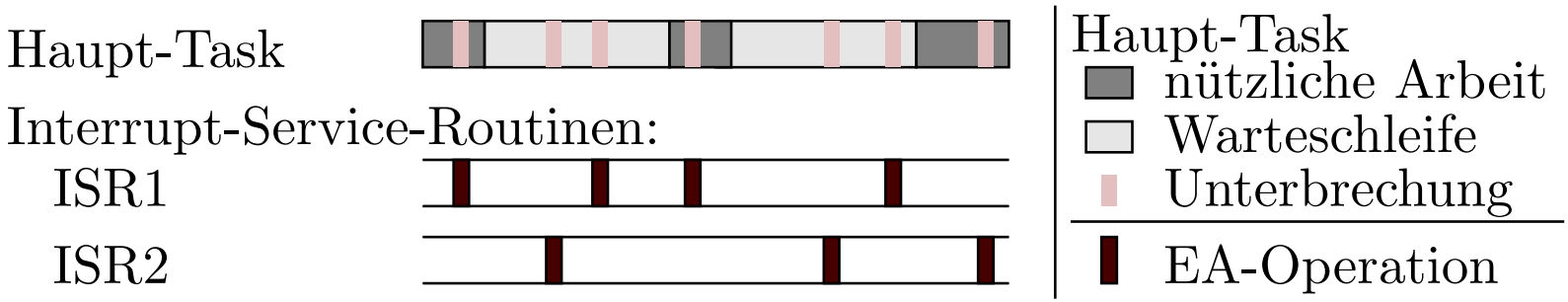
Programmstruktur

```
int main(){
  <Initialisierung, Variablen, ...>
  while (1){              // Endlosschleife
   if (<Haupt-Task bereit>)
     {<Haupt-Task weiter abarbeiten>}
   if (<IO-Task 1 bereit>)
     {<IO-Task 1 abarbeiten>}
   if (<IO-Task 2 bereit>)
     {<IO-Task 2 abarbeiten>}
   ...
  }
}
```

- Der Haupt-Task muss sich nach hinreichend kurzer Zeit für mindesten einen IO-Abfragezyklus unterbrechen.
- IO-Tasks max. wenige hundert abzuarbeitende Befehle.
- Keine Warteschleifen außer der Endlosschleife.

Interrupt

Interrupts



Haupt-Task:

- Ohne zyklische Abfrage von Ereignisbits.
- Lokale (individuelle) und globale Interrupt-Freigabe.

Interrupt-Behandlung:

- Unterprogrammaufruf der ISR (feste Hardware-Adresse).
- Interrupt-Freigabe global aus. Inhalte genutzter Register incl. Statusregister, Frame-Pointer, ... retten.
- EA-Operation ausführen
- gerettete Registerinhalte wiederherstellen und Rücksprung.

Interrupt-Freigabe, Ereignisbits, ... ATmega 2560

Globale Interrupt-Freigabe: Bit »I« (SREG.7, EA-Adresse 0xFE):

Bitnummer:	7	6	5	4	3	2	1	0
Bitname:	I	T	H	S	V	N	Z	C

C-Anweisungen für des Setzen und löschen von »I«:

```
sei();          // Interrupts global ein
cli();          // Interrupts global aus
```

Adresse und Konfigurationsbits externer Interrupts an Port B:

Interrupt	Adresse*1	Ereignisbit	Freigabebit	weiter Einstellung*2
INT0 (PB0)	0x0002	EIFR.0	EIMSK.0	EICRA[1:0]
INT1 (PB1)	0x0004	EIFR.1	EIMSK.1	EICRA[3:2]
	...	...	...	...
INT7 (PB7)	0x0010	EIFR.7	EIMSK.7	EICRB[7:6]

*1 Startadresse der Interrupt-Service-Routine (ISR)

*2 Interrupt bei null, steigender und/oder fallender Flanke.

Interrupt-Quellen des ATmega 2560

Ingesamt 57 (ATmega2560.pdf[3], Abschn. 14.1 Interrupts):

- 8×für Port B Anschlüsse. Bei Port B als Eingang auch als Software-Interrupts nutzbar.
- 3 ×Port-Change-Interrupts für programmierbare Bitänderungen auch an anderen Ports.
- 1×Watchdog (Timeout).
- 26× für Timer-Funktionen.
- 4× 3 für die 4 universellen seriellen Schnittstellen (USARTs).
- 1× SPI (serielle Übertragung abgeschlossen).
- 1×Analogkomparator.
- 1×ADC (Digital-Analog-Wandlung abgeschlossen).
- 1×TWI (2-wire Serial Interface).
- 1×EEPROM (Schreiboperation abgeschlossen), ...

[3] http://techwww.in.tu-clausthal.de/site/Lehre/Rechnerarchitektur...

Vor- und Nachteile von Interrupts

Vorteile:

- Schnellere Reaktion auf externe Ereignisse.
- Der Haupt-Task muss nicht aller paar tausend Befehle zur Hauptschleife zurückkehren, darf Warteschleifen enthalten.

Nachteile:

- Haupt-Task wird an zufälligen Programmstellen unterbrochen,
- dadurch kein deterministischer Ablauf.
- Viele zusätzliche Fehlermöglichkeiten.
- Erschwerter Test, erschwerte Fehlersuche.

Einige Regeln für Interrupt-Routinen:

- kurze, vorhersagbare Abarbeitungszeit, keine Warteschleifen.
- keine Änderung von Daten und Registerinhalten, die das unterbrochene Programm möglicherweise gerade bearbeitet.

Interrupt-Sperren

Ein unterbrechbares Programm muss die ISR immer sperren, während es Übergabedaten bearbeitet:

```
...
uint8_t tmp=<Interrupt-Freigaberegister>;
<Interrupt-Freigabebit löschen>
<Bearbeitung der Übergabedaten>
<Interrupt-Freigaberegister> = tmp;
```

Beispiel einer ISR:

```
#include <interrupts.h> // Header für Int.-Nutzung
#include <avr/io.h>
ISR(INT0_vect){       // Int. bei Tastendruck an PB0
  PORTA^=1;           // PA.0 (LED 1) invertieren
}
```

INT0_vect – Startadresse der Interrupt-Service-Routine (ISR).

Beispiel für ein Hauptprogramm hierzu:

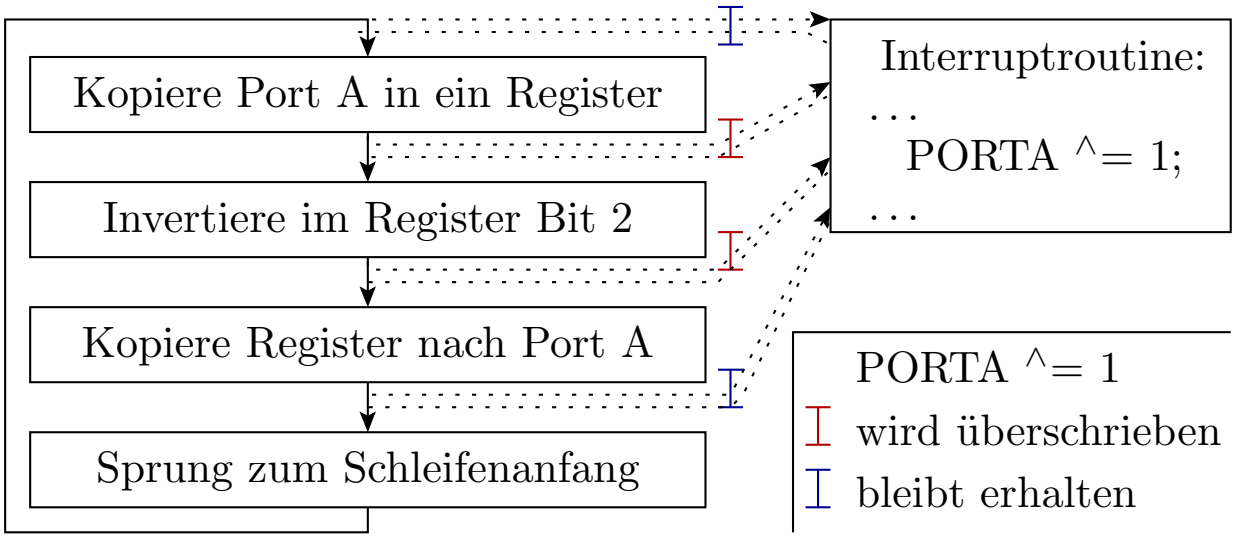
```
int main(){
  DDRB  = 1;      // PB0 Eingang
  DDRA  = 0;      // alle Bits Port A Ausgänge
  EIMSK =1<<INT0;// Interrupt-Freigabe PB0
  EICRA = 0b11;  // Interrupt bei 01-Flanke
  sei();          // globale Interrupt-Freigabe
  while (1)       // Endlosschleife
   uint8_t tmp = EIMSK; // Int.-Freigabe speichern
   EIMSK &= ~1;  // INT0 an PB.0 sperren
   PORTA^=2;     // PA.0 (LED 2) invertieren
   EIMSK=tmp;    // Int-Freigabe wiederherstellen
  }
}
```

Wenn INT0 nicht während »PORTA^=2« gesperrt wird, wird die Hälfte der Anweisungen »PORTA^=1« in der ISR bei Tastendruck nicht wirksam. Warum?

$\Rightarrow$ Bearbeitung derselben Daten (PORTA)

Ohne die drei Anweisungen zur Interrupt-Sperre wird der Schleifenkörper in die Schrittfolge übersetzt:

- Port A lesen,
- Wert bearbeiten,
- Wert schreiben und
- Sprung zum Schleifenbeginn:



An 50% der Unterbrechungsmöglichkeiten wird die Invertierung von PA.0 in der ISR vom Rückschreibwert für die Invertierung von PA.1 überschrieben, d.h. Reaktion nur auf 50% Tastendrucke.

Timer

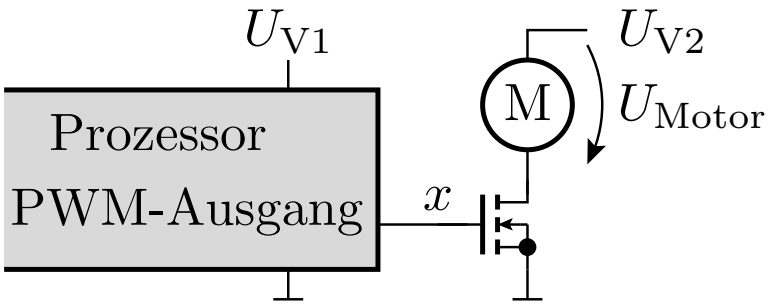
Timer

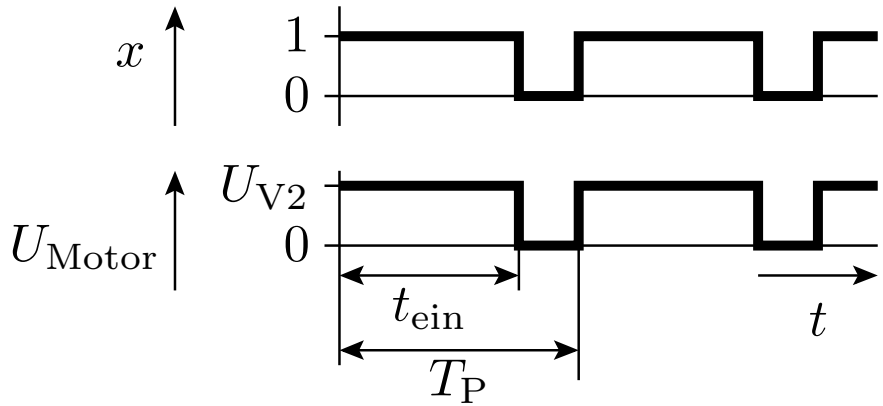
Ein Timer ist eine Hardware-Einheit aus Zähl-, Vergleichs-, Konfigurationsregistern, ... zur

- Erzeugung von Wartezeiten,
- zeitgesteuerten Ereignisabarbeitung,
- Erzeugung pulsweitenmodulierter (PWM-) Signale und
- Pulsweitenmessung.

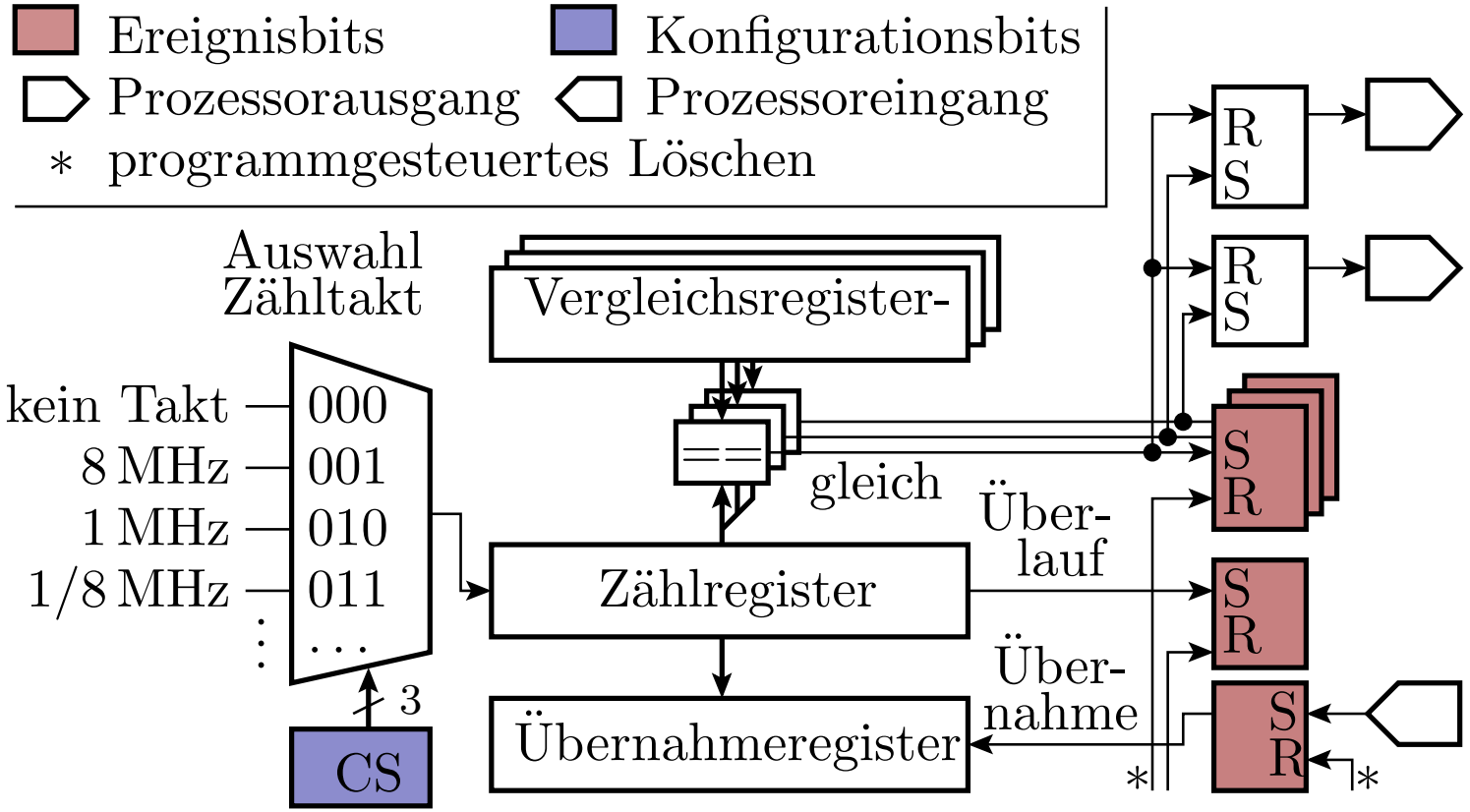
PWM-Signale dienen

- zur Informationsübertragung z.B. an Modellbauservos und
- zur stufenlosen Leistungssteuerung, z.B. von Elektromotoren.





Aufbau und Funktionsweise eines Timers



- Kern eines Timers ist ein Zählregister mit einem vom Programm zuschaltbaren programmierbaren Takt.
- Die Ereignisbits (Überlauf, Gleichheit, externe Flanke) sind vom Programm les- und löschbar.

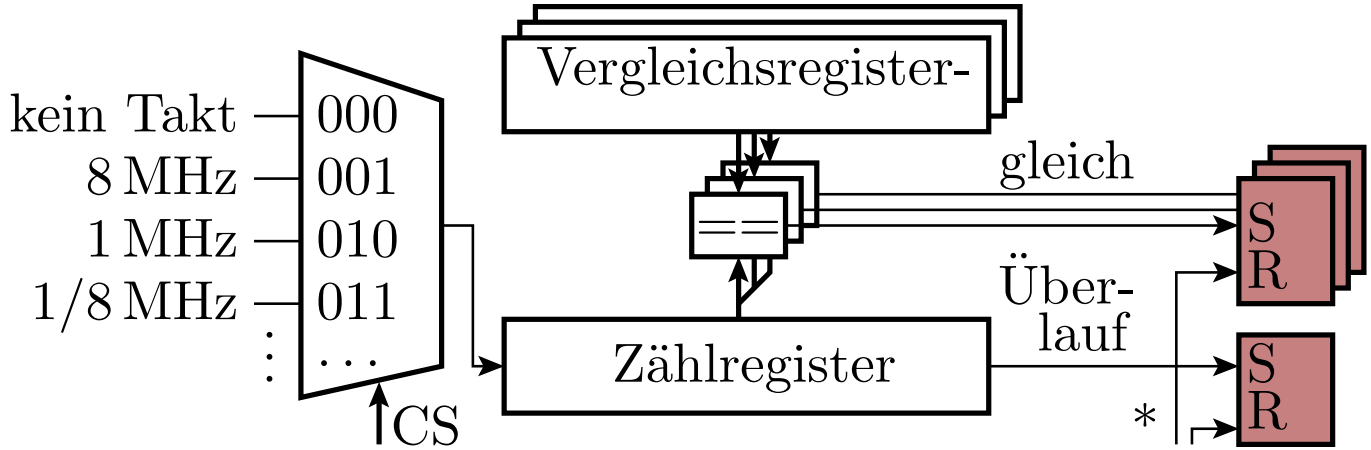
Timer des ATMega 2560

- Zwei 8-Bit Timer (Tmr0 und Tmr2).
- Vier 16-Bit-Timer (Tmr1, Tmr3, Tmr4 und Tmr5).

Die Bit-Anzahl beschreibt die Größe der Zähl- und Vergleichsregister.

Normalmodus

Normalmodus

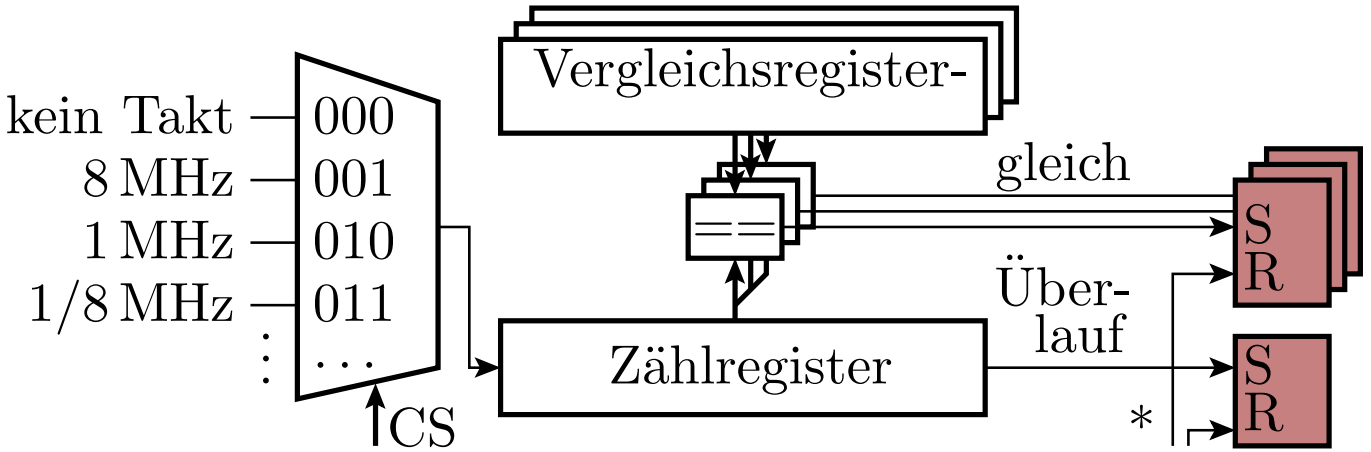


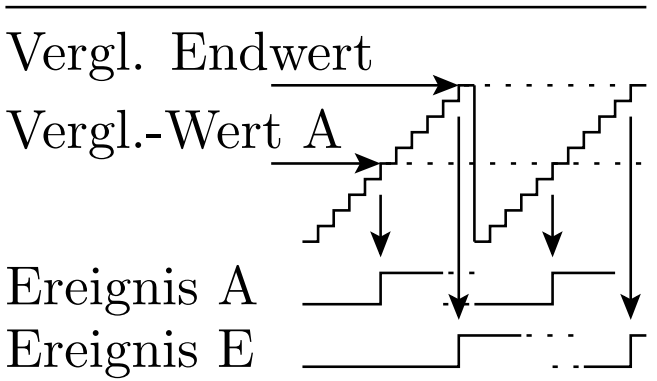
- Zählregister zählt zyklisch bis zum Überlauf.
- Beim Überlauf wir eine Überlaufbit und bei Gleichheit mit einem Vergleichsregister ein Vergleichsbit gesetzt.
- Beispiel Wartefunktion:

```
void wait(int32_t tw){
 <berechne und setze Taktauswahl und Vergleichswert>
 <Lösche Zähler und Vergleichsereignisbit>
 <warte bis Ereignisbit wieder gesetzt ist>
 <schalte Zähltakt aus> }
```

CTC-Modus

CTC- (Clear on Compare) Modus



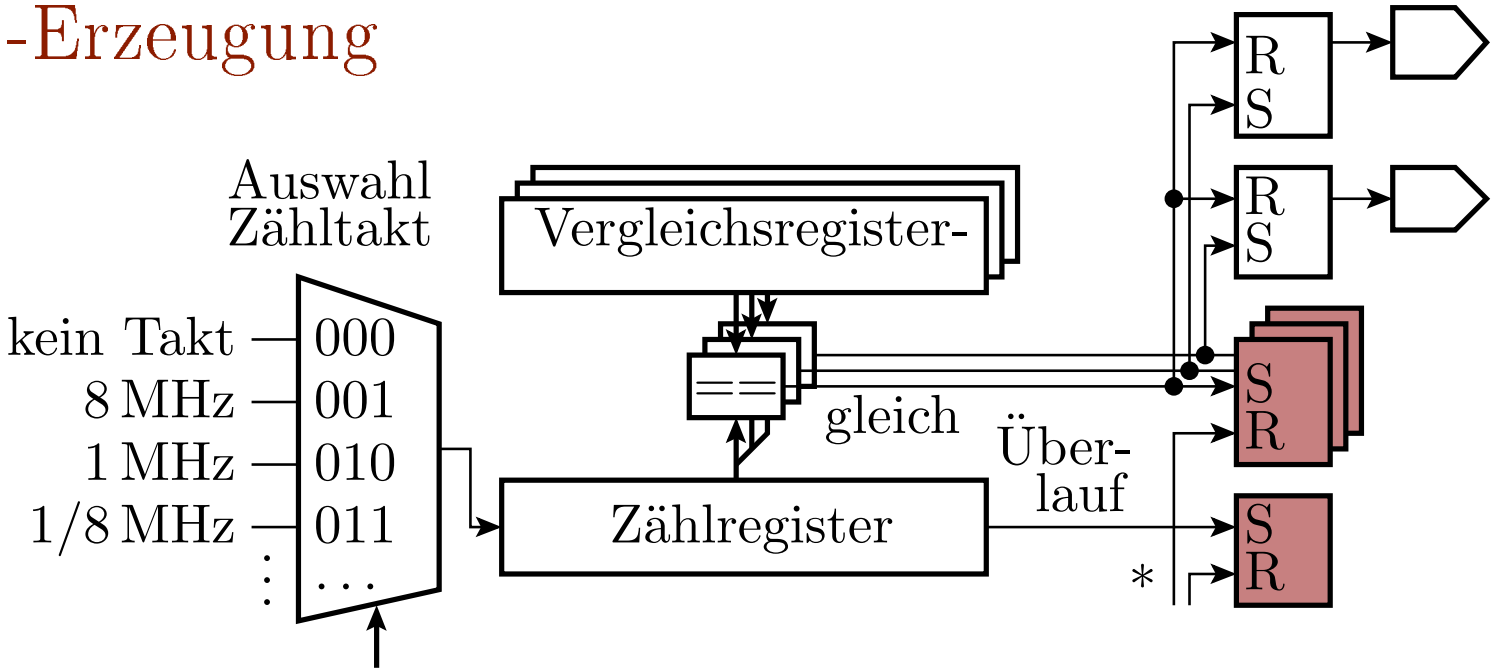


- Zähler wird bei Gleichheit mit einem der Vergleichsregister rückgesetzt.
- Auslösung zyklischer Ereignisse, z.B. Uhrenprozess:

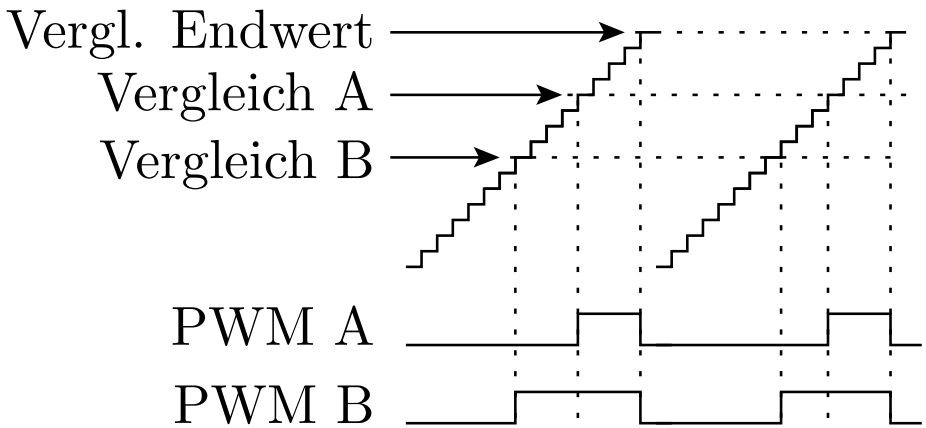
```
void Schrittfunktion Uhr(){
  if (<Vergleichs-Rücksetz-Ereignis>)
   <lösche Ereignisbit(s) und schalte Uhr weiter>
}
```

PWM-Erzeugung

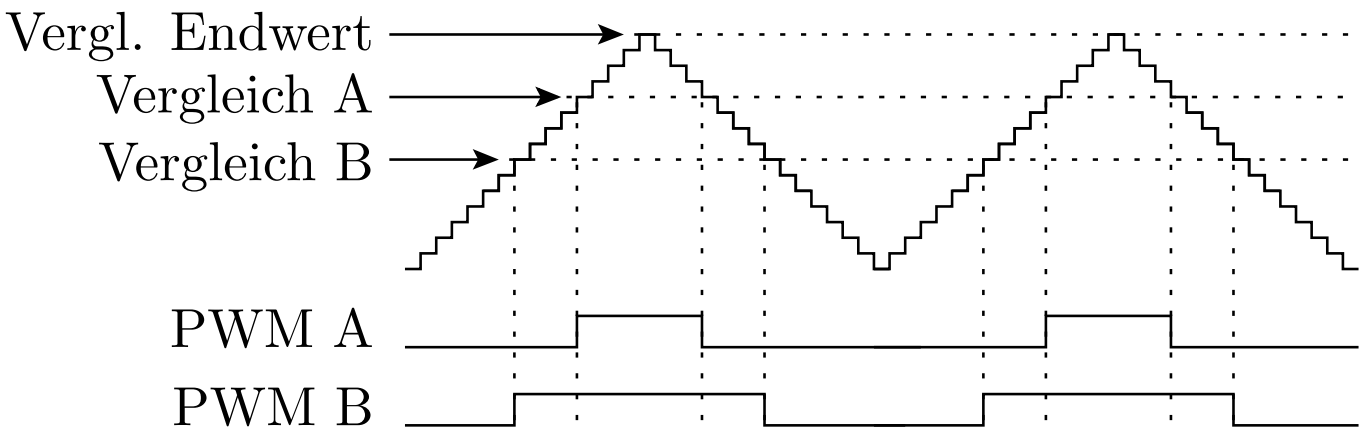
PWM-Erzeugung



- Vergleichsereignis setzt Zählerrücksetzereignis und löscht Ausgabe.
- Pulsgenerierung z.B. zur Motoransteuerung ohne Schrittfunktion.



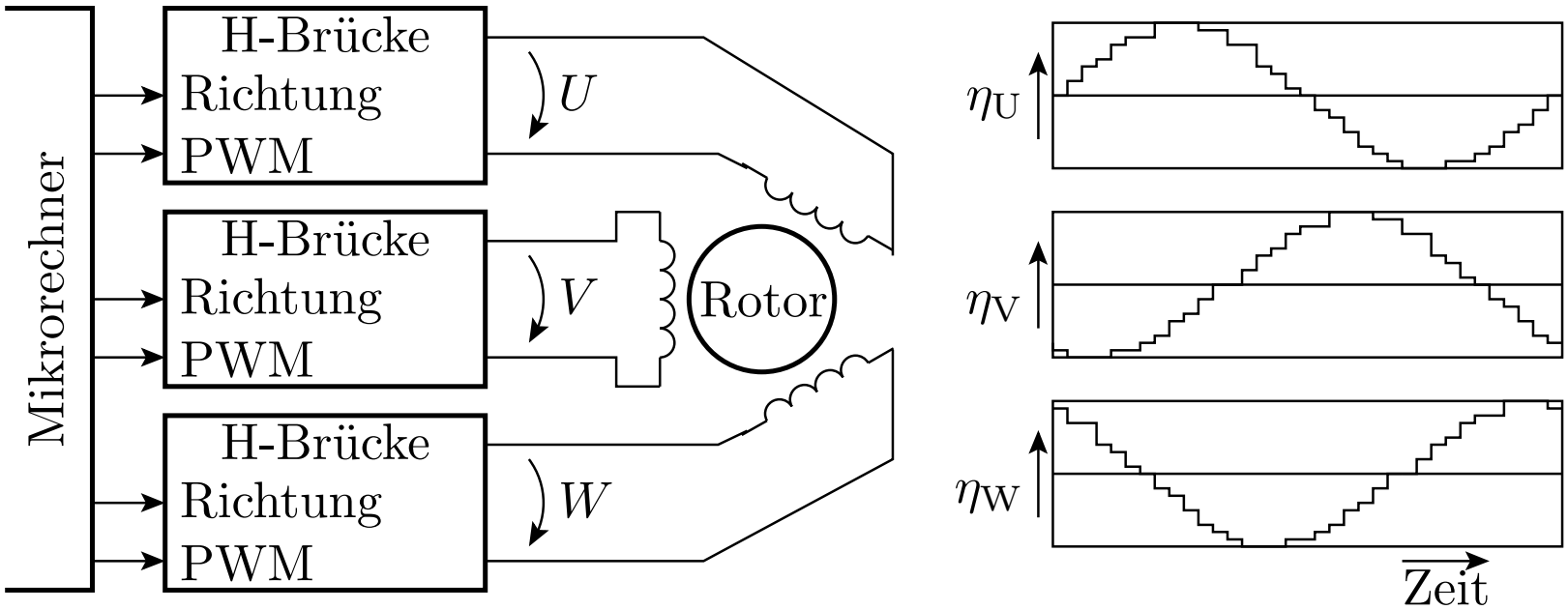
Symmetrische PWM[4]



- Endvergleichswert schaltet die Zählrichtung um.
- Bei Gleichheit und Hochzählen wird die Ausgabe ein- und bei Gleichheit und Abwärtszählen ausgeschaltet.
- Bei dieser und der vorherigen PWM kann auch eine invertiert Ausgabe programmiert werden, so dass der Vergleichswert statt der Ausschalt-, die Einschaltzeit festlegt.

[4] Im Datenblatt unseres Prozessors ist das die phasenrichtige und die vorhergehende normale PWM die schnelle (Fast-) PWM.

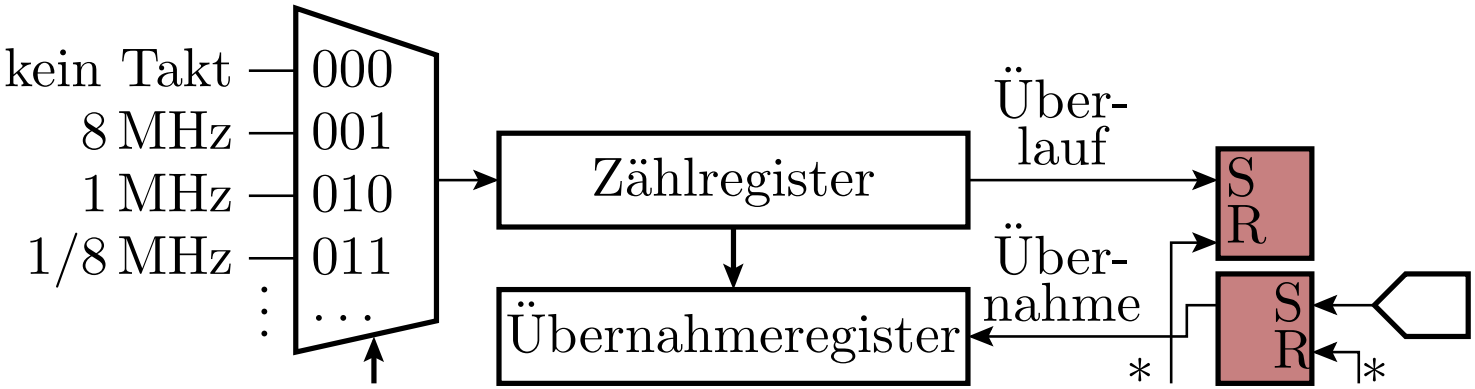
Typische Motoransteuerung



- Die Erzeugung von 3 sinusförmigen Mittelwertverläufen erfordert eine PWM-Einheit mit drei Vergleichsregistern.
- Ansteuerung über H-Brücken.
- Stufenlose Positions-, Geschwindigkeits- und Drehmomentsteuerung für Typen von Elektromotoren.

Pulsweitenmessung

Pulsweitenmessung



- Externes Ereignis (Schaltflanke) bewirkt Übernahme des Zählwerts in das Übernahmeregister.
- Programmgesteuerte Differenzbildung der Übernahmewerte zwischen den Übernahmeereignissen.

Watchdog

Watchdog-Timer (WDT)

Jedes größere Programm enthält statisch gesehen Fehler, die unter anderem auch dazu führen, dass das Programm abstürzt. Der WDT begrenzt die Dauer der Nichtverfügbarkeit durch Abstürze auf wenige ms bis s.

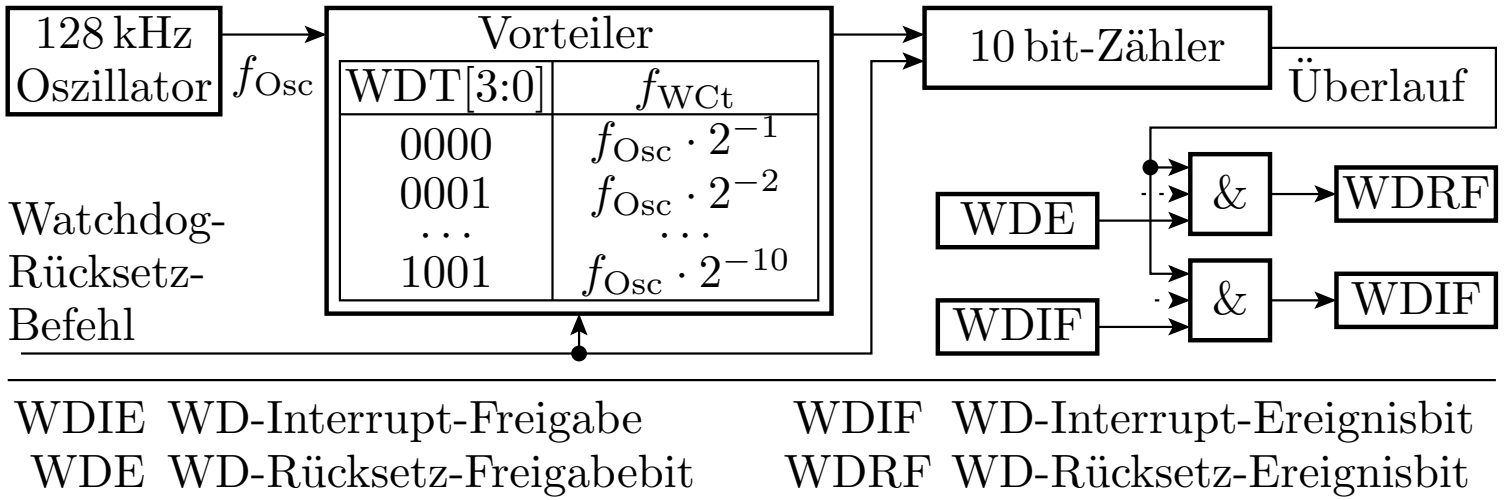
Funktionsprinzip:

- Zeitzähler, der Zeitimpulse zählt und bei Überlauf das System neuinitialisiert (und/oder Interrupt auslöst).
- Um Überläufe (Neuinitialisierungen) zu verhindern, muss das System in einer vorprogrammierten Mindestzeit Rücksetzbefehle für den Watchdog ausführen.

Programmierbar sind:

- An-/Aus-Zeit bis zum Überlauf und
- Reaktion bei Überlauf (Interrupt und/oder Neustart).

Watchdog-Timer (WDT) des ATmega 2560



- Zeit bis zum Überlauf: programmierbar von 16 ms bis 8 s.
- Nur-Interrupt: »Wiederbelebung« per Software.
- Interrupt + Rücksetzen: Datenretten + Neustart.
- WDT-Reset mit Fuse-Bit »WDTON« auch dauerhaft aktivierbar. Dann keine Deaktivierung durch Software (-Fehler) möglich.

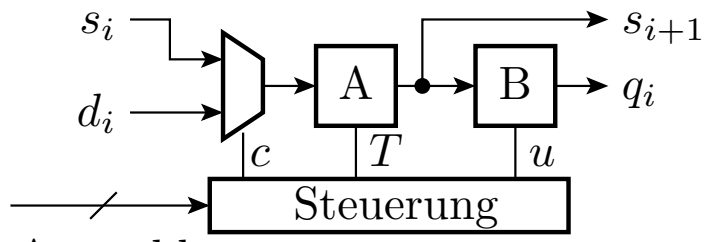
Serielle Schnittstellen

Serieller Datenaustausch

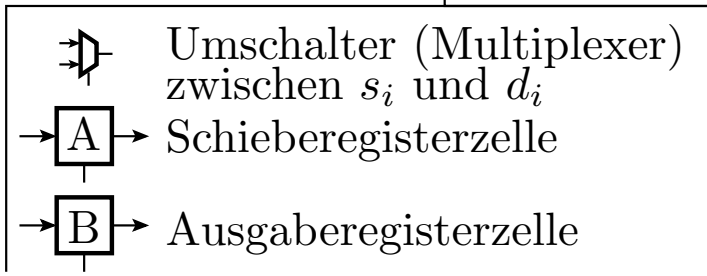
Der Datenaustausch zwischen Rechnern erfolgt in der Regel seriell[5]. Grundbaustein Schieberegister mit den Funktionen

- parallele Übernahme der zu übertragenden Daten,
- serielle Übertragung und
- parallele Übergabe.

Schaltung einer Schieberegisterzelle:



Steueraufgabe	c	T	u	Operation
Schieben	0	$\int$	0	$s_{i+1} \leftarrow s_i$
Übernahme	1	$\int$	0	$s_{i+1} \leftarrow d_i$
Übergabe	–	0	$\int$	$q_i \leftarrow s_{i+1}$
keine	–	0	0	——

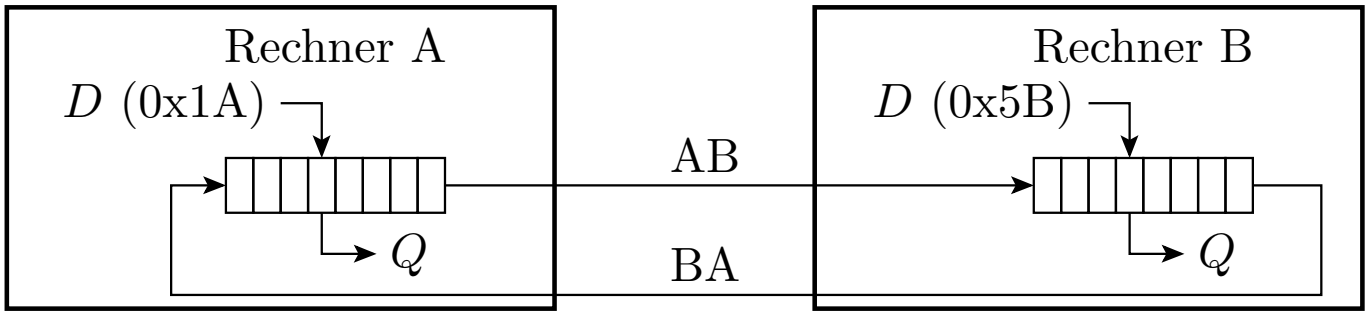


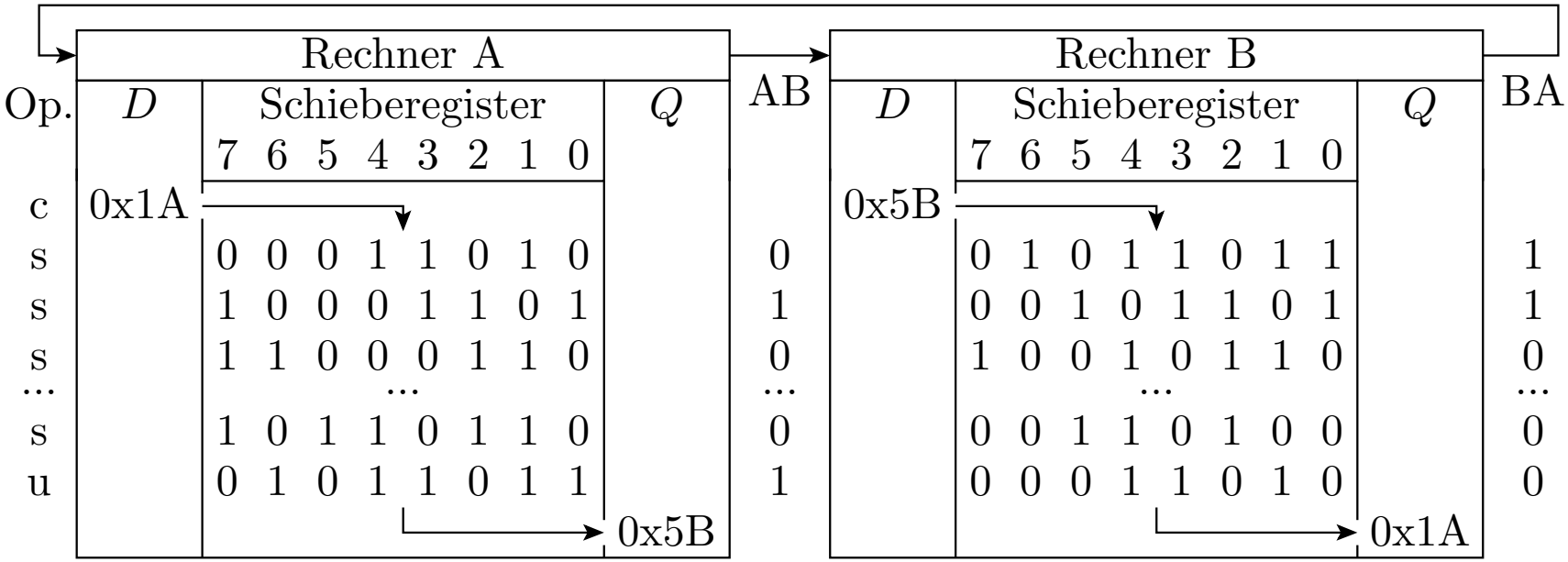
interne Takt- und Steuersignale:

u Update
c Capture
T Takt

[5] Seriell, d.h. hintereinander über eine, statt parallel über viele Leitungen.

Bidirektionale Kopplung zweier Rechner



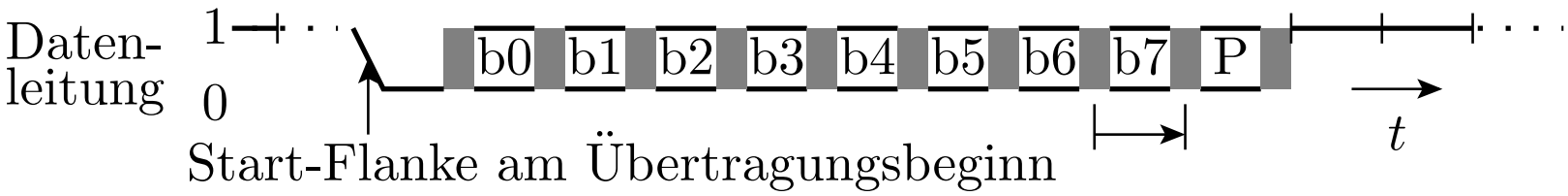


c Übernahme (Capture) s Schieben (Shift) u Übergabe (Update)

USART

UART (Universal Asynchronous Receiver Transmitter)

Übertragung ohne Takt und Steuersignale.



$b \in \{0,1\}$	Datenbits	$\cdots$	Übertragungspause
$P \in \{0,1\}$	Paritätsbit	$\longmapsto$	Bitzeit, z.B. $t_{\mathrm{Bit}} \approx 0{,}1\,\mathrm{ms}$
$\vdash\!\!\dashv$	Stoppbit (Wert 1)		Übertragungsdauer: $12 \cdot t_{\mathrm{Bit}}$

Der Empfänger erkennt den Übertragungsbeginn an der Stopp-/Start-Flanke und übernimmt die Werte nach 1,5, 2,5 etc. Bitzeiten. Voraussetzung: Gleich eingestellte Bitzeit, Bitanzahl, Stoppbitanzahl und Parität bei Sender und Empfänger. Die Baudrate b als Kehrwert der Bitzeit t_{Bit} wird mit einem Teiler aus dem Prozessortakt gebildet.

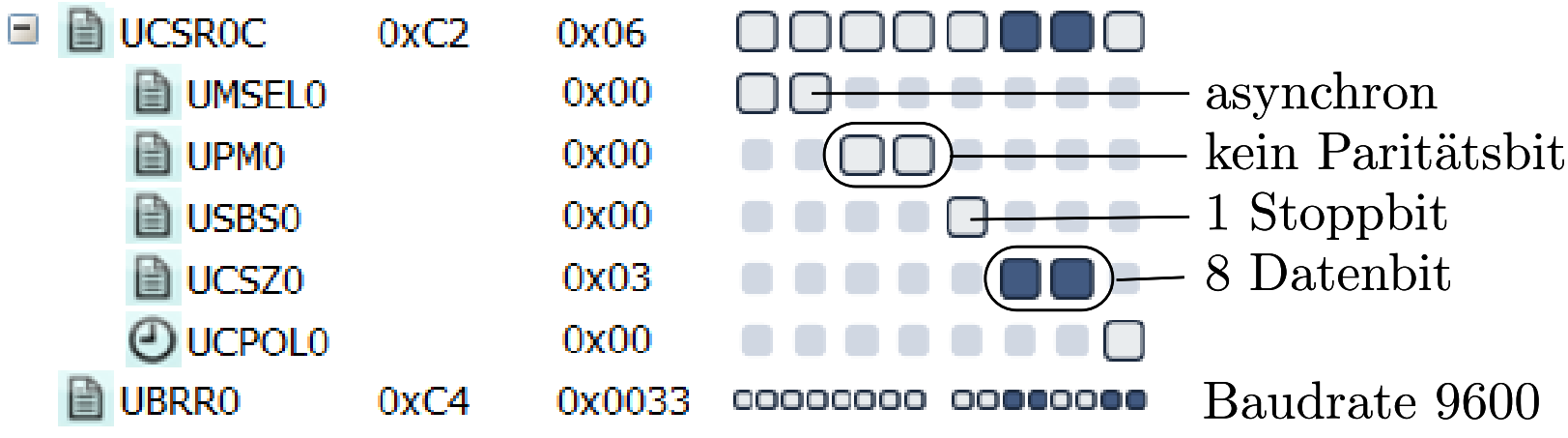
Baudrate b als Funktion der Prozessortaktfrequenz f_{Proz} und dem programmierbaren Teilerfaktor T:

$$b = \frac{f_{\text{Proz}}}{16 \cdot (T + 1)}$$

Teilerfaktor für $f_{\text{Proz}} = 8\,\text{MHz}$ und $b = 9600\,\text{Baud}$:

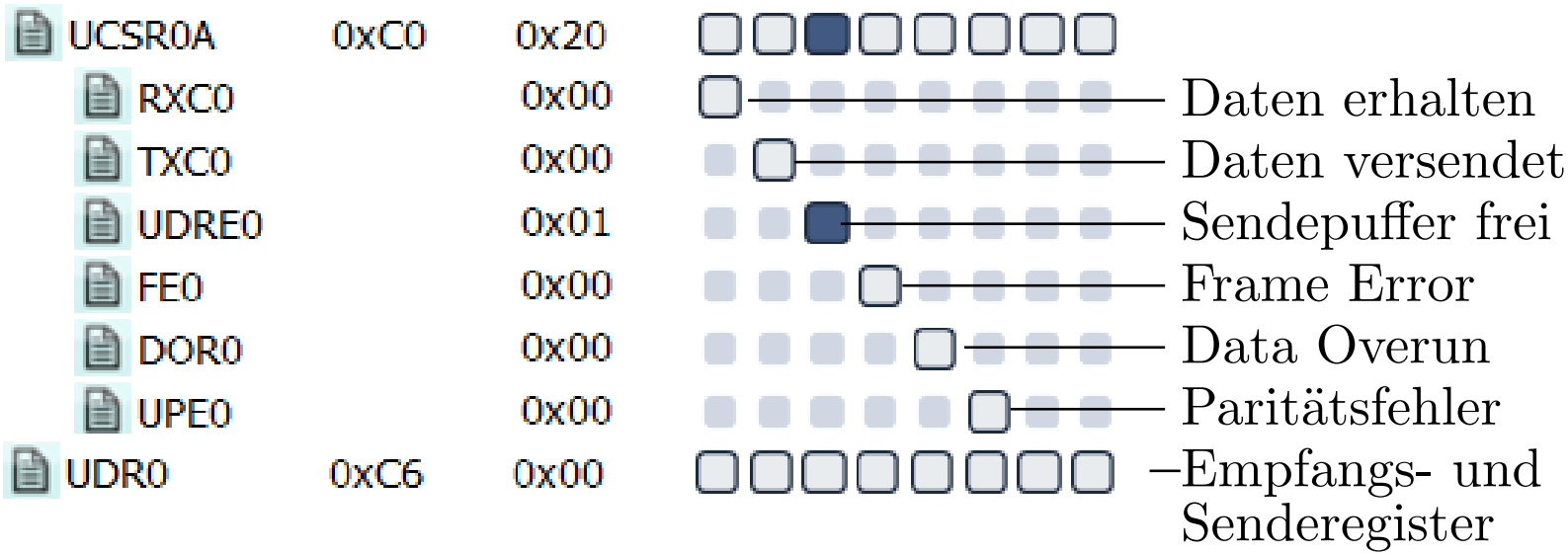
$$T = \frac{8\,\text{MHz}}{16 \cdot 9600\,\text{Hz}} - 1 = 51 = 0\text{x}33$$

Initialisierung: 9600 Baud, 8 Daten-, 1 Stoppbit, keine Parität:




```
UBRR0  = 0x33;          Teilerwert für 9600 baud
UCSR0B = 0b00011000;    Sender und Empfänger einschalten
UCSR0C = 0b00000110;    Datenformat 8n1
```

Sende- und Empfang



Funktionen für den Empfang und das Versenden eines Bytes

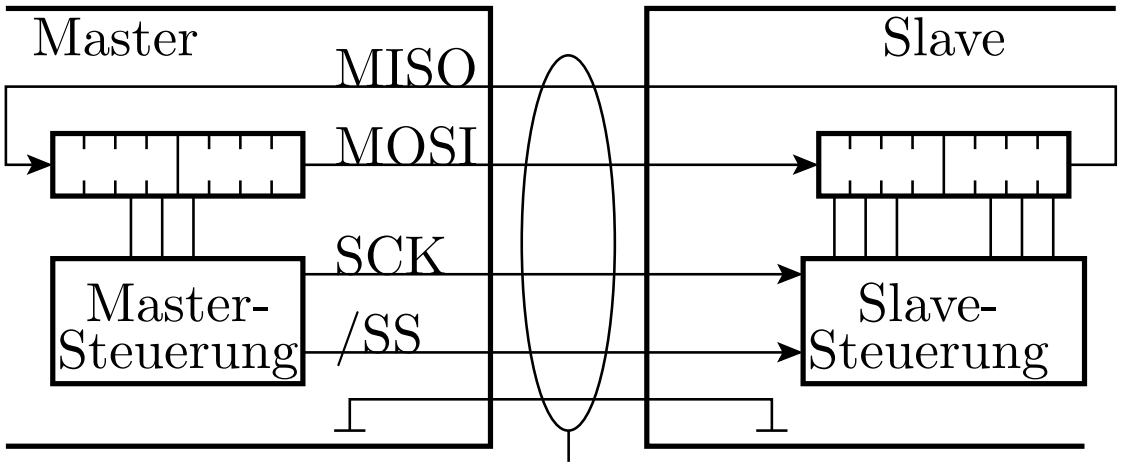
```
uint8_t getChar(){
  while (!(UCSR1A & (1<<RXC0)));   // Warte auf Empfang
  return UDR0;                     // Rückgabe Empfangsbyte
}

void putChar(uint8_t c){
  while (!(UCSR1A & (1<<UDRE0)));  // Warte bis Sendepuffer
  UDR0 = c;                        // frei. Byte versenden
}
```

SPI-Bus

SPI-Bus

Serieller Bus zur Vernetzung von Schaltkreisen.

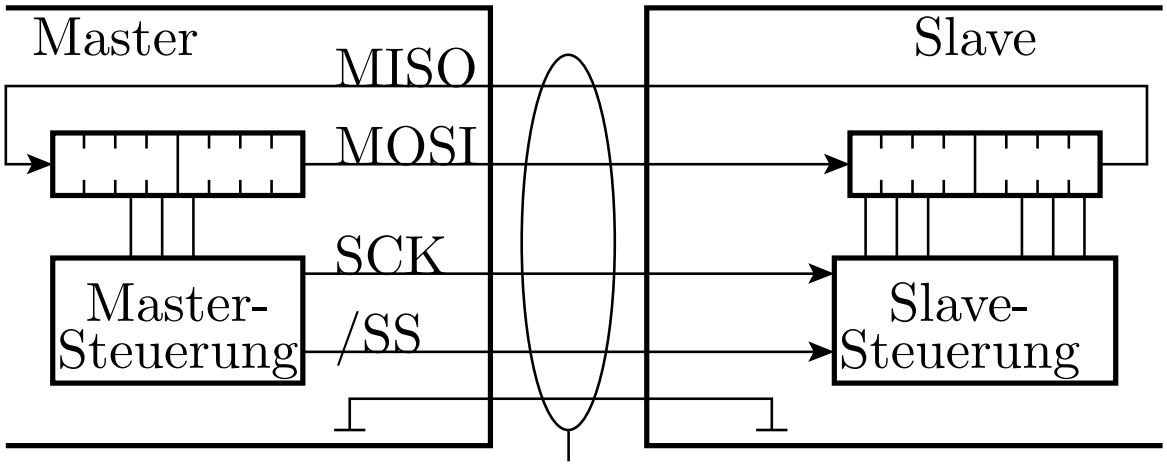


Leitungen zwischen den Schaltkreisen

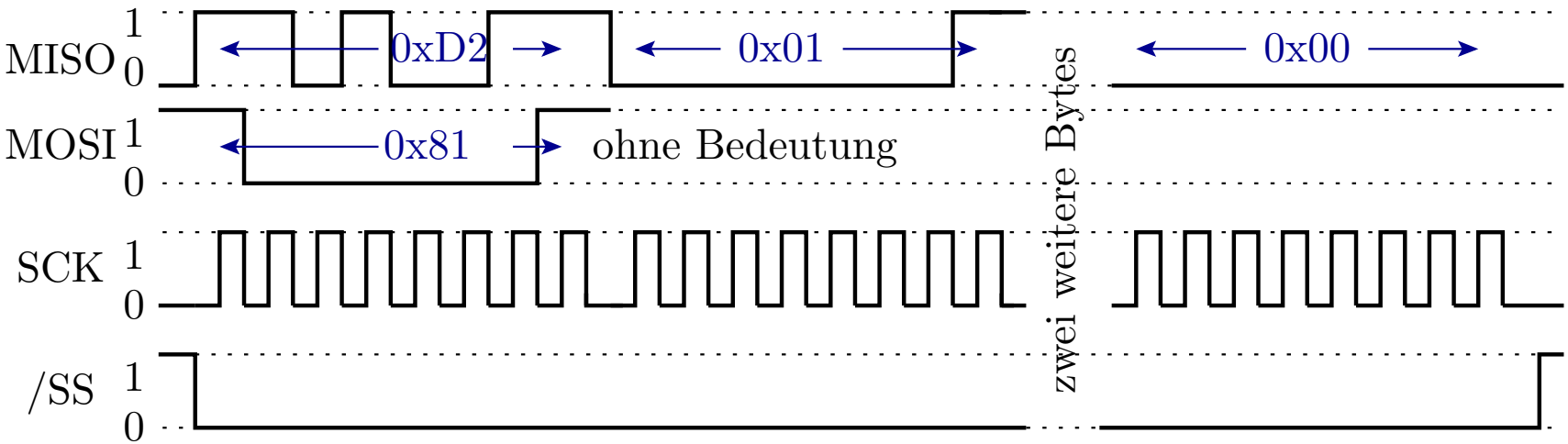
MOSI	Master Out Slave In
MISO	Master In Slave Out
SCK	Schiebetakt
/SS	Slave-Auswahl

Aktion	/SS	SCK
Übernahme	⊓	
Schieben	0	⊓
Ausgabe	⊓	
keine	sonst	

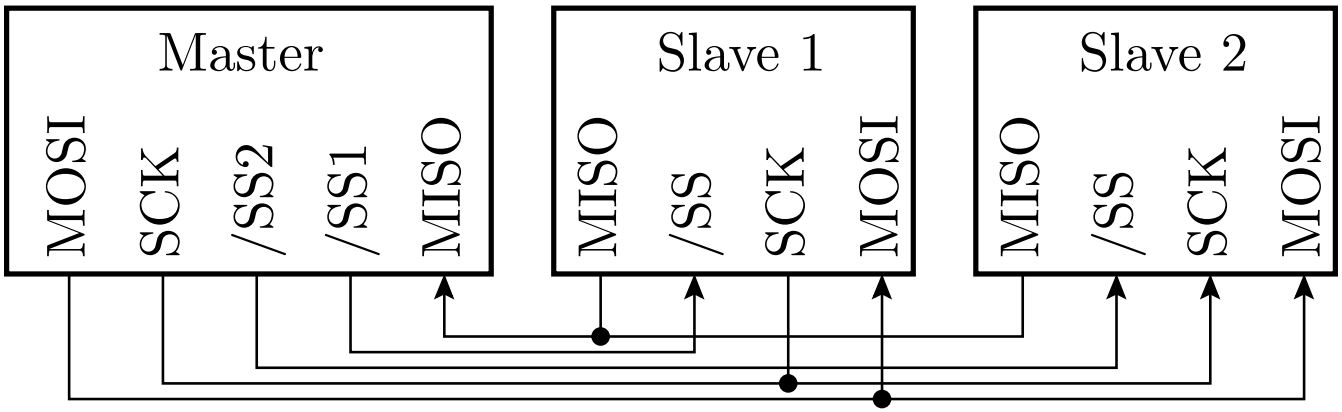
- Ein Schaltkreis ist der Master, der den Takt SCK und die Slave-Auswahlsignale erzeugt, die anderen sind Slaves, die diese Signale vom Master erhalten.



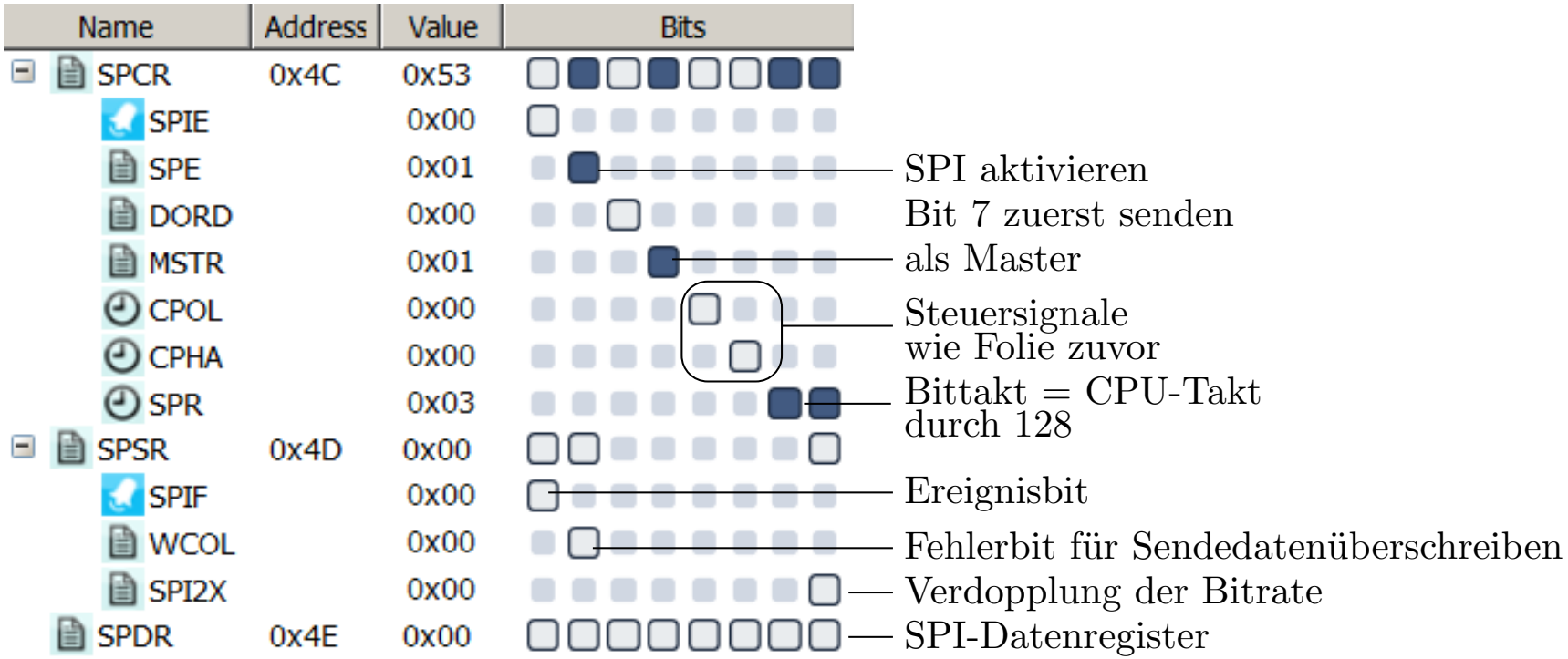
Beispiel für die Übertragung 0x81 vom Master zum Slave und von 0xD201...00 vom Slave zum Master:



- Eine Übertragung beginnt mit Aktivierung von /SS=0 (Übernahme), gefolgt von n Schiebetakten und endet mit Deaktivierung /SS=1.
- Das /SS-Signal des Masters wird über einen Ausgang einer parallelen Schnittstelle gesteuert.
- An einen Master können parallel mehrere Slaves mit unterschiedlichen /SS-Signalen angesteuert werden.



Konfiguration des SPI-Controllers



- Einschalten als Master oder Slave.
- Festlegen der Bitrate und Protokollparameter.
- Pins für /SS Signale konfigurieren, beim Master als Ausgänge mit Wert eins, beim Slave als Eingänge.

Algorithmus für den Datenaustausch

Name	Address	Value	Bits
SPSR	0x4D	0x00	
SPIF		0x00	
SPDR	0x4E	0x00	

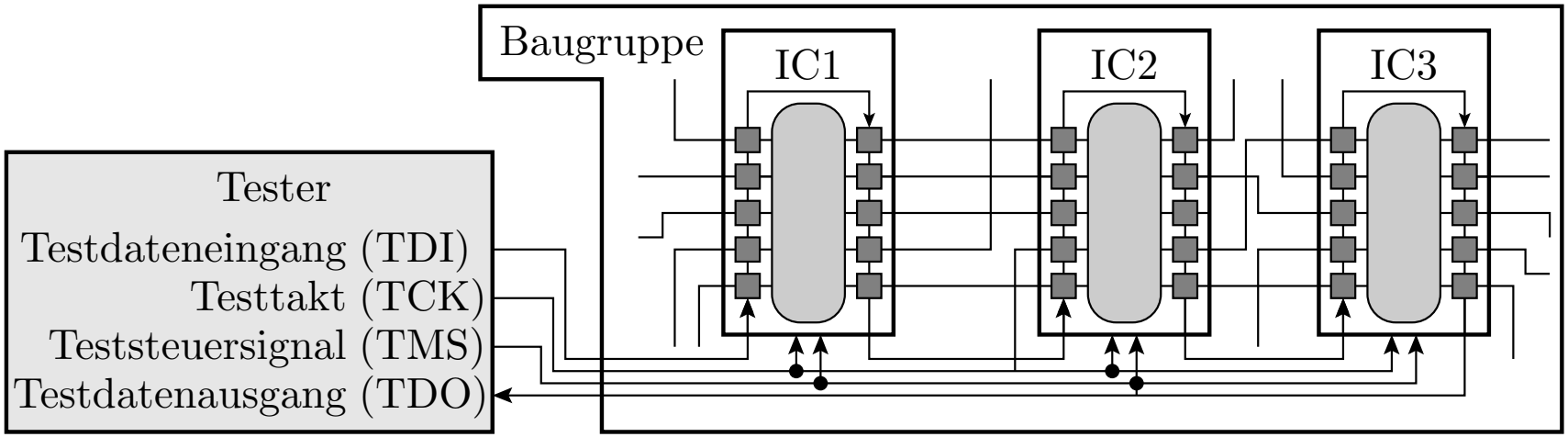
Für jede n-Byte-Übertragung

- Aktiviere das Slave-Auswahlsignal /SS (Master) bzw. warte auf /SS=0 (Slave).
- Für jedes Byte
  - Schreibe Sendewert in das Datenregister.
  - Warte bis Ereignisbit SPIF eins ist.
  - Lese empfangenes Byte aus und schreibe nächstes zu sendende Byte in das SPI-Datenregister SPDR.
- Deaktiviere das Slave-Auswahlsignal.

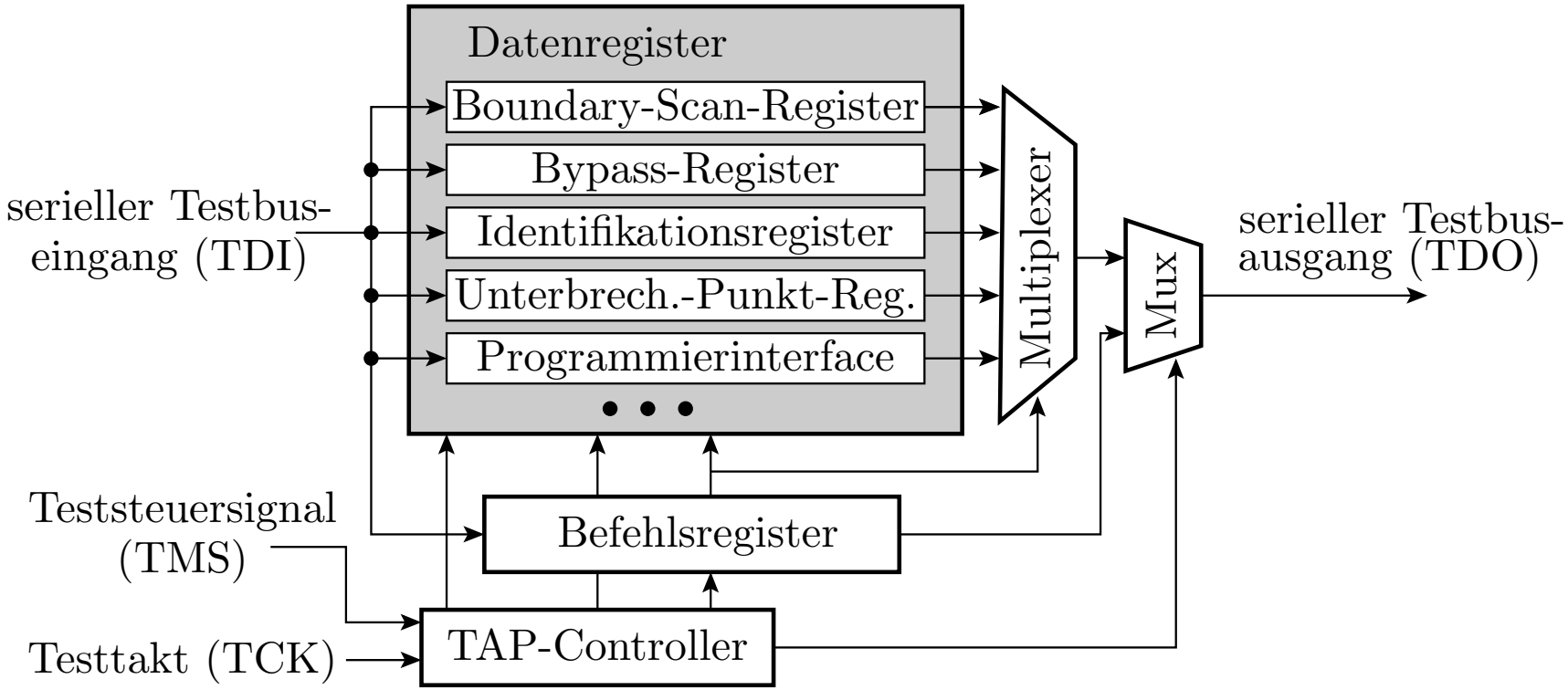
JTAG (Testbus)

JTAG

Test-, Diagnose-, Debug- und Programmierbus.

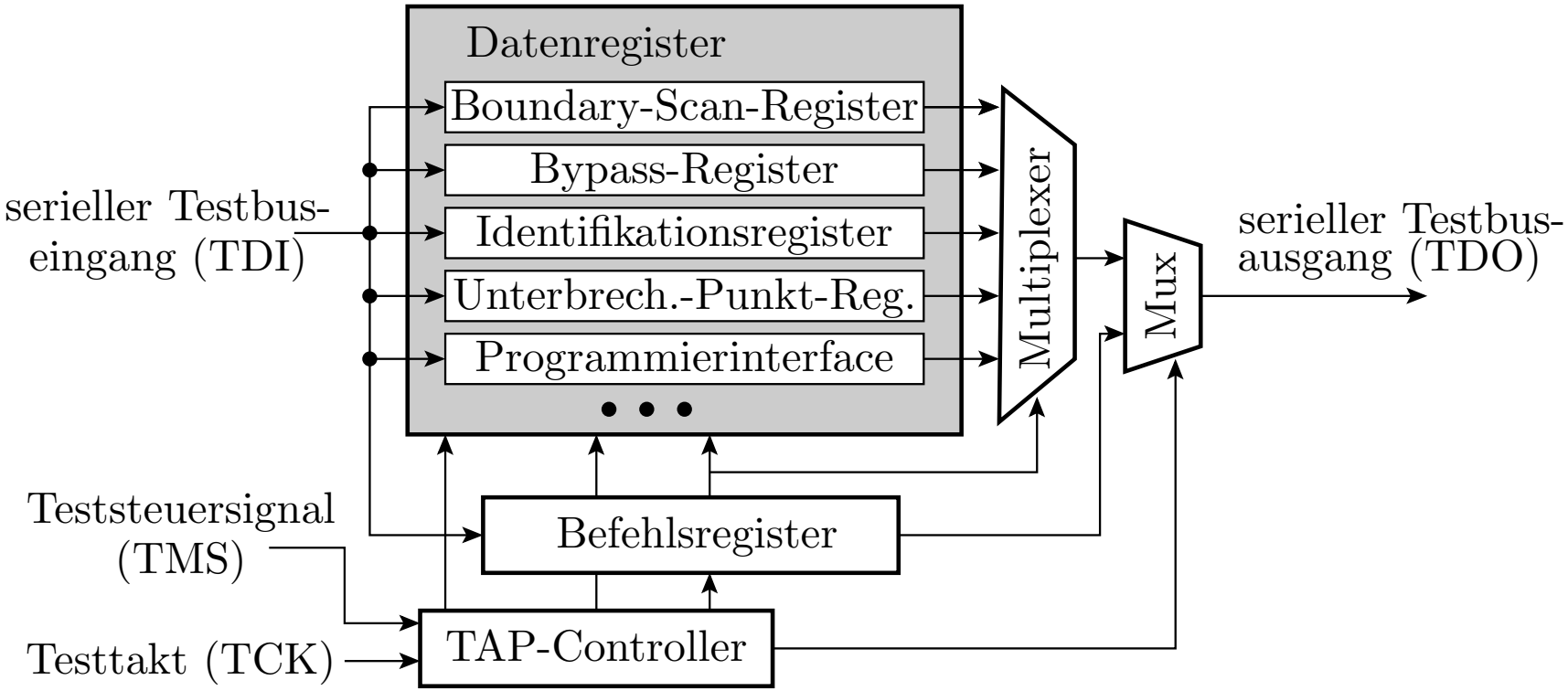


- Verschaltung aller Schaltkreise zu einer Kette.
- In der Ursprungsidee (Boundary-Scan) waren die Schieberegister mit den Funktionen Übernahme, Schieben und Übergabe am Schaltkreisrand angeordnet, um die Verbindungen zwischen den Schaltkreisen ohne mechanische Kontaktierung testen zu können.



Ein Schaltkreis mit JTAG-Bus hat mehrere über ein Befehlswort auswählbare Datenregister:

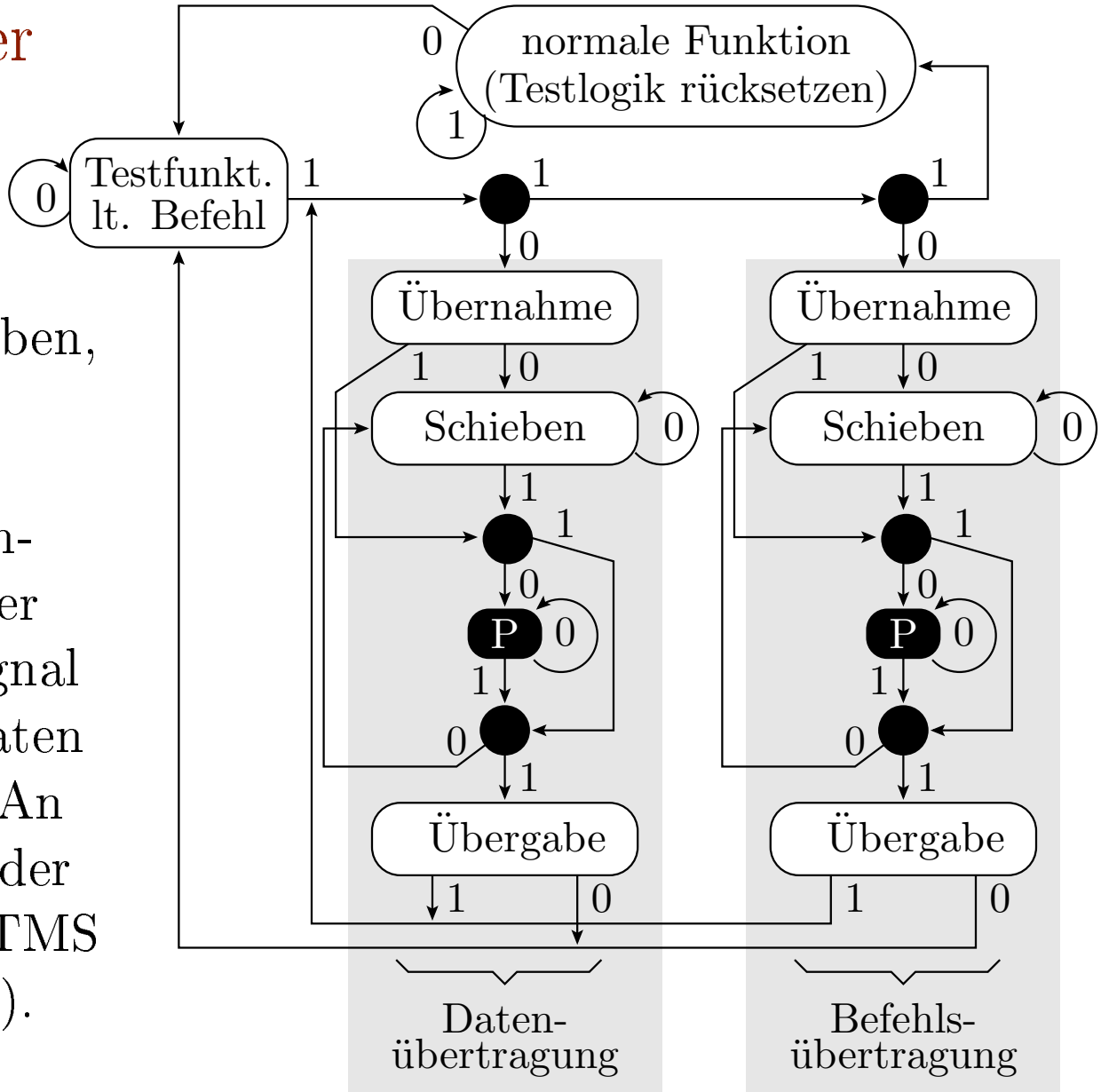
- Bypass-Register zur Verkürzung der Länge des Schieberegisters durch den Schaltkreis auf 1 Bit.
- Identifikationsregister mit Hersteller- und Bauteilnummer.



- Programmier-Interface: Schnittstelle zum Lesen und Schreiben des Befehls-Flashes, des Daten-EEPROMs und der Fuse-Register.
- Schnittstellenregister zum OCD (On-Chip Debugger), ...

TAP-Controller

Die Auswahl der 6 Busaktionen: Übernahme, Schieben, Übergabe für des Befehls- und das ausgewählte Datenregister erfolgt über ein 1-Bit-Steuersignal und einen Automaten mit 8 Zuständen. An den Kanten steht der Wert des Signals TMS (Test Mode Select).



Von der JTAG-Implementierung in unserem Prozessor sind nur die standardisierten Testfunktionen, die für den Bestückungstest von Baugruppen vorgesehen sind, öffentlich zugänglich dokumentiert. Die Befehle für die Programmierung und den OCD (On-Chip Debugger) sind in den Dokumentationen nicht beschrieben.

Speicher

Externer Speicher und EEPROM (ATmega 2560)

Externer Datenspeicher:

- Max. 64 kByte großer an den Prozessor angeschlosser RAM.
- Erweitert Adressbereich für interne Variablen von 2 kByte.
- Verlängerte Zugriffszeit.
- Nutzt 2,5 Ports.

Daten-EEPROM:

- Größe: 2 kByte
- Für Daten, die nach Abschalten der Versorgungsspannung erhalten bleiben sollen.
- Vom Programm byteweise les- und beschreibbar. Über Programmierschnittstellen seitenweise (8 Byte) programmierbar.

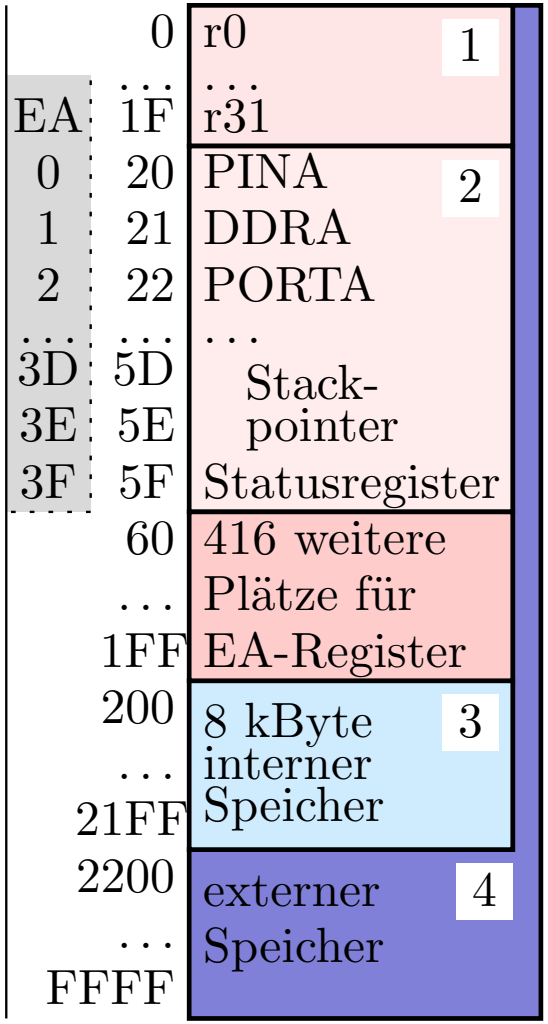
Programm-Flash (ATmega 2560)

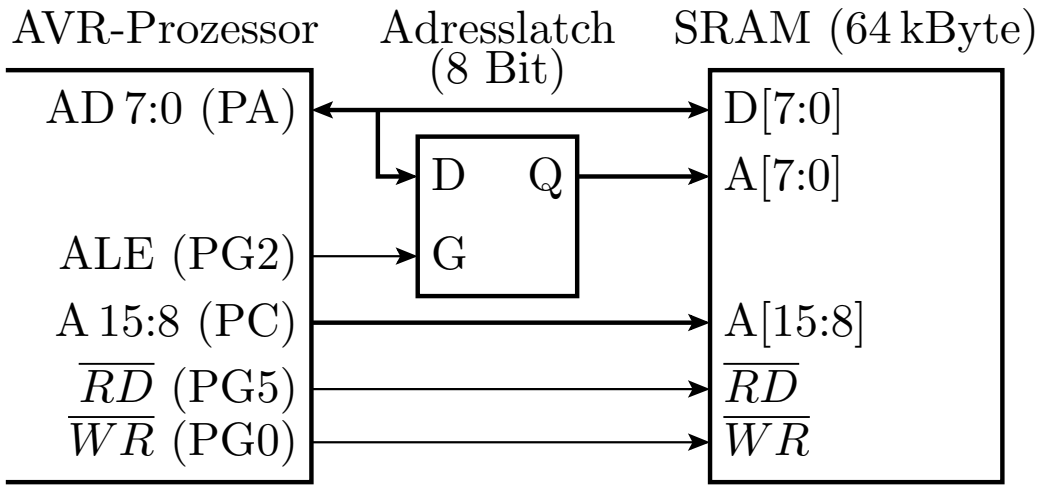
- Größe: 128 k × 2 Byte.
- Programmierung: JTAG, parallel, seriell oder mit Bootloader. Seitenweise. Seitengröße 128 × 2 Byte.
- Bootloader: Programm in einem reservierten oberen Adressbereich, der bei entsprechender Fuse-Einstellung nach Reset angesprungen wird. Von diesem Speicherbereich sind Flash-Seiten des unternen Speicherbereichs programmierbar.
- Lesezugriff aus dem Programm: LPM- oder ELPM-Befehl (Foliensatz 2, Abschn. 2.3 Konstanteninitialisierung).

Externer Speicher

Externer Speicher

- Zur Erweiterung des Datenspeichers kann ein max. 64 kByte großer externe Speicher an den Prozessor angeschlossen werden.
- Anschluss an die Ports A, C und G.
- Damit Lade- und Speicherbefehle auf den externen Speicher zugreifen, sind in SFR-Register XMCRA das Bit SRE (Bit 7) und gegebenenfalls weitere Bits in XMCRA und XMCRB zu setzen.





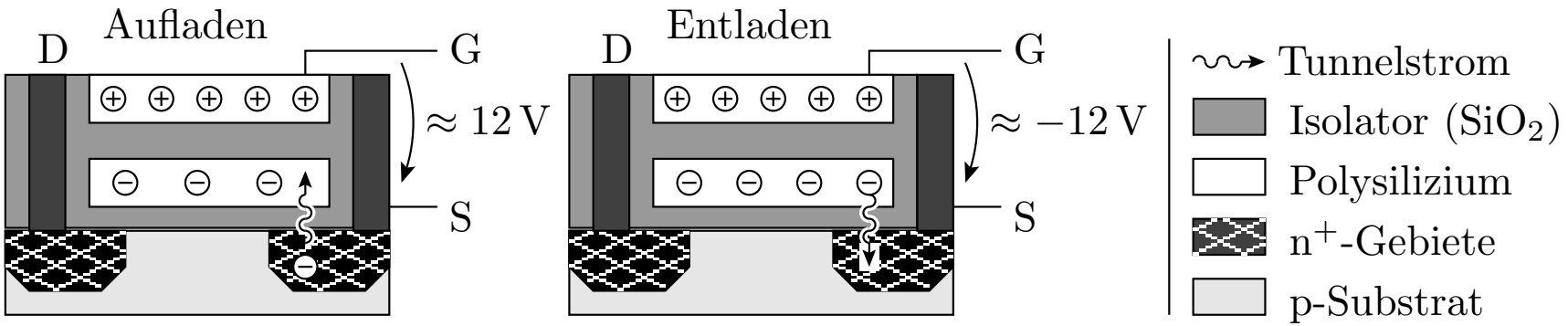
- Port A liefert das niederwertige Adressbyte und übernimmt einen Schritt später die Lesedaten bzw. gibt die Schreibdaten aus.
- Port C liefert das höherwertige Adressbyte. Ein Teil von Port G liefert die Steuersignale.
- Wenn der externe, auf der Versuchsbaugruppe vorhandene Speicher nicht genutzt wird (SRE=0), sind auf Port G5 und Port G0 jeweils eine »1« auszugeben[6]

[6] Durch falsche Ansteuerung sind bereits auf mehreren Versuchsbaugruppen im Übungsraum die Speicher-Chips kaputt gegangen.

EEPROM

EEPROM

- Der Datenspeicher verliert ohne Spannung seine Information.
- Die Werte für Konstanten stehen im Befehlsspeicher und werden vom Startup-Code in den RAM kopiert.
- Wo werden während des Betriebs anfallende Daten gespeichert, die nach Abschalten nicht verloren gehen dürfen? ⇒ EEPROM (Electrically Erasable and Programmable Read-Only Memory).
- Programmierung über Tunnelströme. Schreibzeit ca. 30.000 Prozessortakte.



EEPROM-Ansteuerung

EEPROM-Adressregister	–	–	–	–	Bit 11	Bit 10	Bit 9	Bit 8	EEARH (0x42)
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	EEARL (0x41)
Datenregister	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	EEDR (0x40)
Kontrollregister	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR (0x40)

löschen+schreiben	0	0
nur löschen	0	1
nur schreiben	1	0

Bits zur Aktivierung und Steueung der Programmierung

Ablauf zum Schreiben eines Bytes in den EEPROM:

1. Warte bis EEPE null ist[7].
2. Schreibe die Adresse in EEAR und die Daten in EEDR.
3. Schreibe EEMPE$\leftarrow$ 1 und EEPE$\leftarrow$ 0.
4. Innerhalb der nächsten 4 Zyklen schreibe auch EEPE$\leftarrow$ 1.

[7] Warte, bis die vorherige EEPROM-Schreiboperation abgeschlossen ist. Für Schreiboperation ca. 3.3 ms nach letztem Setzen des Bits EEPE. Bei möglichen Flash-Op. zusätzliche warten, bis SPMEN in SPMCSR null ist.

Schreiboperation als C-Funktion:

```
void EEPROM_write(uint16_t uiAddress, uint8_t ucData)
  // Warte auf Abschluss letzte Schreiboperation
  while(EECR & (1<<EEPE));
  EEAR = uiAddress;  // Adressübergabe
  EEDR = ucData;     // Datenübergabe
  EECR |= (1<<EEMPE);
  EECR |= (1<<EEPE); // Start der Schreiboperation
}
```

Die Leseoperation muss auf den Abschluss der Schreiboperation warten und ist einen Takt nach Start der Leseoperation fertig:

```
uint8_t EEPROM_read(uint16_t uiAddress) {
 // Warte auf Abschluss letzte Schreiboperation
  while(EECR & (1<<EEPE));
  EEAR = uiAddress;  // Adressübergabe
  EECR |= (1<<EERE); // Start der Leseoperation
  return EEDR;       // Ergebnisrückgabe
}
```

Analoge Eingaben

Messung und Überwachung von Analogeingabe

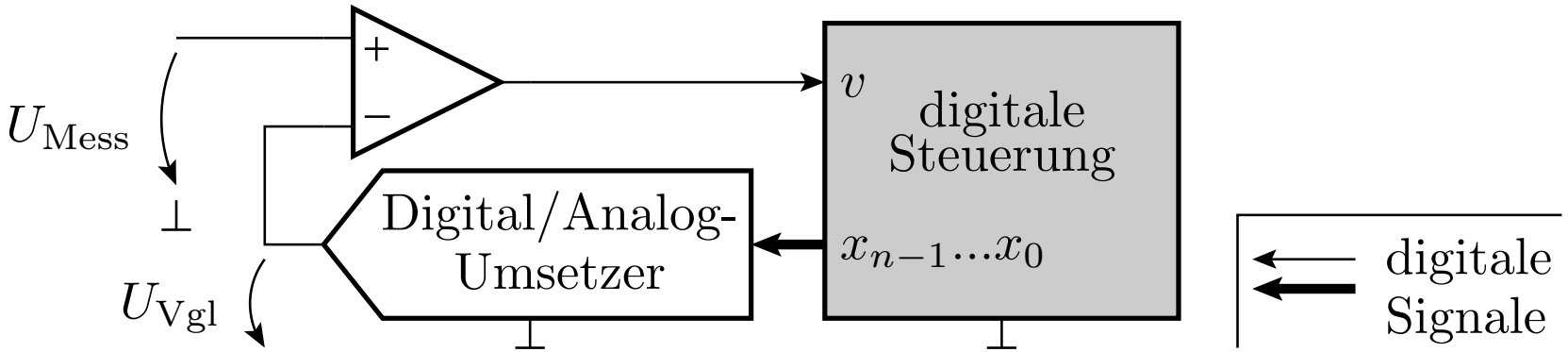
Analog-Digital-Wandler:

- Umwandlung einer analogen Eingangsspannung oder Eingangsspannungsdifferenz in einen Bitvektor (beim ATmega 2560 10 Bit).
- Wandlung verlangt 13 bis 25 Zeitschritte der Dauer $\geq 10\,\mu$s.
- Über einen programmierbaren Eingabemultiplexer kann zwischen mehreren Signalquellen ausgewählt werden.

Analog-Komparator:

- Vergleich zweier analoger Eingangsspannungen.
- Das 1-Bit-Vergleichergebnis kann programmgesteuert ausgewertet, Interrupts auslösen oder Zeitmessungen mit Timern steuern.

Prinzip eines seriellen Analog-Digital-Wandlers



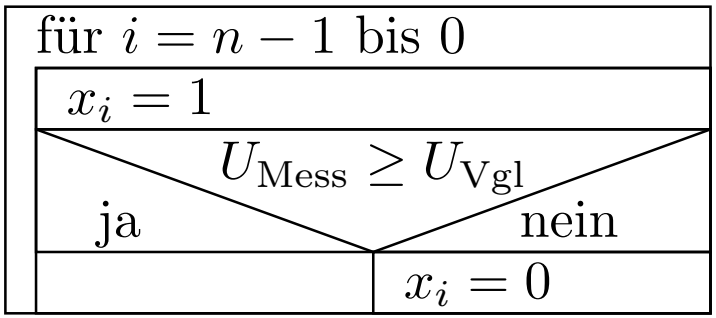
$$v = \begin{cases} 0 & \text{wenn } U_{\text{Mess}} < U_{\text{Vgl}} \\ 1 & \text{sonst} \end{cases}$$

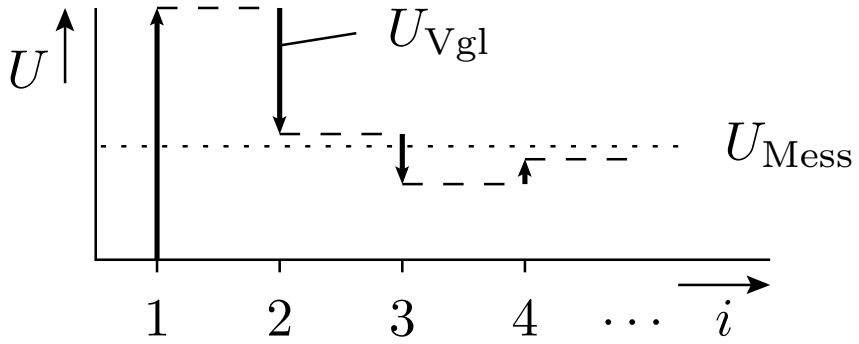
- Die Vergleichsspannung, die der DAU (Digital-Analog-Wandler) ausgibt:

$$U_{\text{Vgl}} = U_{\text{ref}} \cdot \frac{x}{2^n}$$

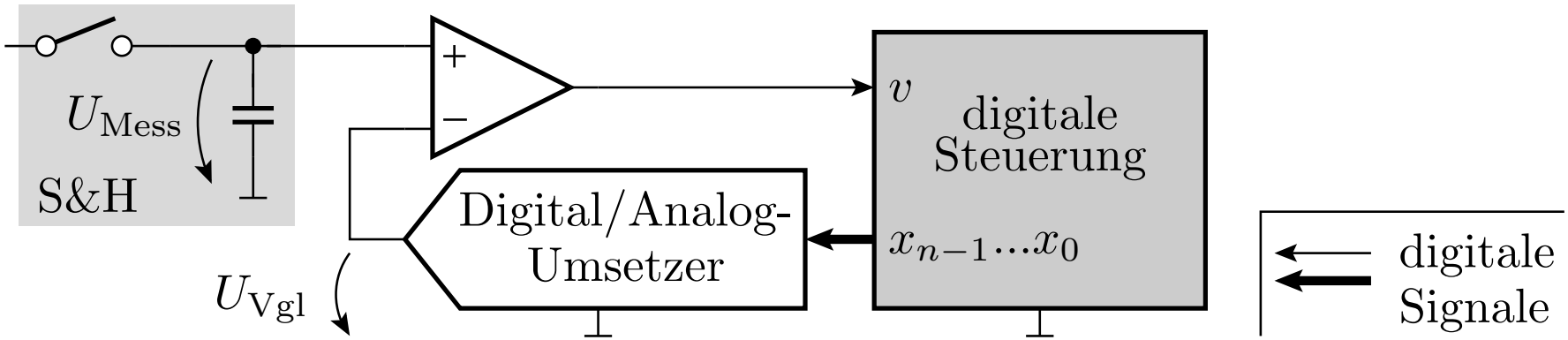
(U_{ref} – Referenzspannung; x – Ausgabewert (Bitvektor); n – Bitanzahl des Ausgabewerts).

Sukzessive Approximation





- Ein Vergleich je Bit. Schnellster serielle Wandlung.
- Der Messwert darf sich während der Wandlung nicht ändern.
  Erweiterung um eine Sample-and-Hold-Schaltung:



Analog-Digital-Wandler des ATmega 2560

- 10 Bit-Wandler.
- Eine normale Wandlung benötigt 13 und die erste mit Initialisierung der Analogschaltung 25 10µs-Schritte.
- Mögliche Messeingänge sind die Anschlüsse ADCi (PF, PK)
- Programmierbar sind einmalige und kontinuierliche Messungen. Max. $\approx 10^4$ Messungen pro s.
- Bei mehreren Messeingängen erfolgen die Messungen nacheinander.
- Zur Minderung der Messfehler kann der Prozessor während der Wandlung in den Sleep-Modus versetzt werden (Abschalten des Prozessortakts).
- ...

Die Konfiguration erfolgt über die SFR-Register ADMUX, ADCSRA, ADCSRB, ...

Beispiel zur ADC-Ansteuerung

ADC-Initialisierung:

```
11  void initADC(){
12    ADMUX = (3<<MUX0); // Kanal 3 auswählen
13    // Einschalten mit Wandlungstaktteiler 64
14    ADCSRA = (1<<ADEN)|(0b110<<ADPS0);
15    DDRE &= ~0x80;   // Sensoreingang als Eingang
16    PORTE &=~0x80;   // Ausgabewert 0 (hochohmig)
17  }
```

- Der Sensor ist an ADC3 (PF3) (Kanal 3 auswählen).
- Der Wandlertakt als CPU-Takt durch Teilerwert

$$f_{\mathrm{ADC}} = \frac{f_{\mathrm{CPU}}}{64} \approx 117\,\mathrm{kHz}$$

- Um den Analogwert nicht zur verfälschen, ist PF3 als Eingang mit Ausgabewert 0 (Pullup aus) zu konfigurieren.

Funktion zur Rückgabe eines Analogwerts

```
19 uint32_t getADC(){
20   uint16_t wert;
21   ADCSRA |= (1<<ADSC); //Wandlung starten
22   // Warten bis "fertig"-Bit gesetzt ist
23   while(!(ADCSRA & (1<<ADIF))){}
24   // "fertig"-Bit durch Schreiben einer
25   ADCSRA |= (1<<ADIF); // 1 löschen
26   wert = ADC;
27   return wert;
28 }
```

- Wandlungsstart durch Setzen von ADSC in ADCSRA.
- Bei Wandlungsabschluss setzt der Prozessor ADIF=1.
- ADIF wird durch Schreiben einer Eins gelöscht.