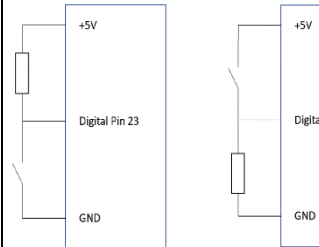
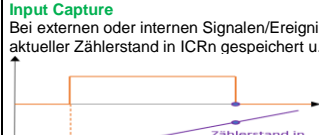
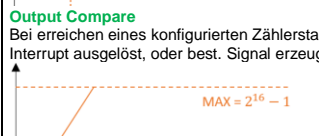
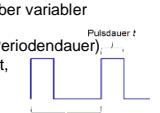

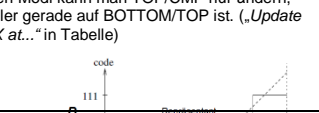
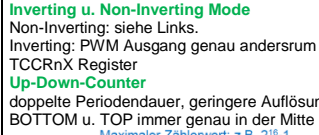
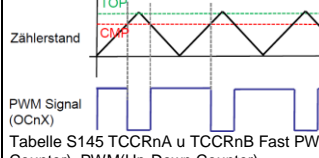
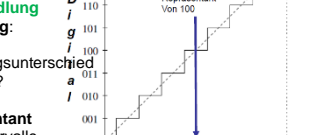


<p>DDRx (Data Direction Register): Entsprechendes Bit auf 1 für Ausgang, oder 0 für Eingang</p> <p>PORTx (Port Register): Wenn Pin auf Ausgang, dann 1 = 5V und 0 = 0V</p> <p>PINx (Port Input Register): Wenn Pin auf Eingang, dann 1 = HIGH liegt an und 0 = LOW liegt an</p> <p>pinMode(13, OUTPUT); digitalWrite(13, HIGH); digitalRead(13); ... if (digitalRead(13) == HIGH)...</p> <p>Serial.begin(9600);</p> <p>Serial.println("Eingabe ist: " + Serial.readString());</p> <p>Serial.available() : Anz. Bytes, die zum Lesen verfügbar sind. Evtl. in while Schleife.</p>	<pre>void setup() { DDRB = (1 << DDB4); //pinMode(pin, OUTPUT); DDRA = ~(1 << DDA0); //pinMode(sw, INPUT); PORTA = ~(1 << PA4); //digitalWrite(pin, LOW); } void loop() { if (PINA & (1 << PINA0)) { //if (digitalRead(sw)) { PORTA = (1 << PA4); //digitalWrite(pin, HIGH); delay(2000); PORTA = ~(1 << PA4); //digitalWrite(pin, LOW); } }</pre> <p>Mikrocontroller Bestandteile: MicroProzessor, Timer, Schnittstellen, Speicher, AD-Wandler</p> <p>Entwicklerboard: Arduino Mega, Mikroprozessor: Atmega2560</p> <p>Cross-Compilation: Programm wird nicht auf Zielformat (Mikrocontroller), sondern auf anderer Plattform übersetzt</p> <p>Flashen: hex-Datei von PC an Entwicklerboard senden</p> <p>Harvard-Architektur: Daten- und Instruktionsspeicher getrennt</p> <p>Instruktionsspeicher: Nicht flüchtiger Flash Speicher</p> <p>Daten: in flüchtigem SRAM</p> <p>SRAM und DRAM sind flüchtig. Rest nicht flüchtig.</p>	<p>Nichtflüchtige Speicher:</p> <p>ROM</p> <p>OTPROM</p> <p>EEPROM: Begrenzte Anzahl an Schreib/Lesezyklen.</p> <p>Konfigurationsdaten, Kalibrierungsdaten</p> <p>Flash: Programm- Daten/Code</p> <p>SRAM(flüchtig): Arbeitsspeicher, Register, Stack usw</p> <p><u>Einzelne Bits setzen:</u> x = (1 << Bitnummer);</p> <p><u>Einzelne Bits löschen:</u> x &= ~(1 << Bitnummer); (1 << Bitnummer2));</p> <p><u>Testen ob Bit auf 1:</u> if (DDRC & (1 << Bitnummer)) {}</p> <p><u>Testen ob Bit auf 0:</u> if (!(DDRC & (1 << Bitnummer))) {}</p> <p><u>Alle Bits umdrehen:</u> x = 0xFF ^ x</p> <p><u>LED togglen:</u> PINA = (1 << PINA2);</p> <p>Vorwiderstand berechnen: $R_v = \frac{U_{ges} - U_F}{I_F}$</p>	 <p>Pull-Up / Active Low Bei offenem Taster wird Spannung am Pin auf HIGH gezogen.</p> <p>Pull-Down / Active High Bei offenem Taster wird Spannung am Eingang auf LOW gezogen.</p> <p>Entprellung Einmaliges betätigen eines Schalters führt evtl zu mechanischen Vibrationen. SW-Lsg: Künstliche Wartezeit nach Zustandswechsel – Bis Schalter eingeschwungen.</p>
<p>Interrupts sei(): Interrupts global aktivieren (oder: SREG = 128) cli(): Interrupts global deaktivieren</p> <p>SREG (AVR Status Register): Bit 7 auf 1 = sei();</p> <p>EIMSK (External Interrupt Mask Register): Speziellen Interrupt de-/aktivieren</p> <p>EICRA (External Interrupt Control Register A) EICRB (External Interrupt Control Register B): ISCn0 und ISCn1. Falling oder Rising Edge. n ist die Interrupt Nummer.</p> <p>EIFR (External Interrupt Flag Register): Wenn Interrupt ausgelöst: Bit ist 1</p> <p>ISR (INT0_vect) {}</p> <p>attachInterrupt(digitalPinToInterrupt(21), count, RISING);</p>	<pre>volatile int counter = 0; volatile unsigned long time = 0; void setup() { Serial.begin(9600); } sei(); EIMSK = (1 << INT2); //pin 19 EICRA = (1 << ISC20) (1 << ISC21); // rising edge } void loop() { Serial.println("Zählerstand: "); Serial.println(counter); delay(3000); } ISR (INT2_vect) { if (millis() - time >= 250) { time = millis(); //Entprellung counter++; } }</pre>	<p>Busy Waiting: while (DDRC & (1 << DDC3));</p> <p>Polling: periodisches Abfragen, ob Ereignis eingetreten</p> <p>Interrupt: Kurze Unterbrechung des laufenden Programms um einen anderen zeitkritischen, kurzen Vorgang zu bearbeiten. Hardware prüft dauernd parallel, ob Ereignis eingetreten ist. Wenn auf ein seltenes Ereignis schnell reagiert werden muss.</p> <p>Trap: Art von Interrupt, die aber synchron und reproduzierbar ist. z.B. System Call, Div durch 0 ...</p> <p>Interrupt Request: Interruptereignis – [InterruptController] – über IRQ Eingang Unterbrechungsanforderung an CPU – CPU unterbricht Programm und startet Unterbrechungsroutine</p> <p>Interrupt Vector Table: Welches Interruptereignis gehört zu welcher ISR? Jede Vectornummer hat eine zugehörige Programmadresse. ISR ist selbst nicht unterbrechbar (1 Bit SREG)</p>	<p>Externe Interrupts Controller tastet zu Beginn jedes Taktzyklus ab. Falls Interrupt aktiviert, Aufruf der ISR. Probleme: Leichte Verzögerung, „Prrellung“</p> <p>Interne Interrupts Timer, A/D-Wandler Bei Auslauf eines Timers unterbricht HW Ausführung der normalen Software</p> <p>Volatile Variable wird vor jedem Lesen aus SRAM gelesen und nach jedem Schreiben in SRAM geschrieben !!Globale Variablen die in ISR vorkommen immer volatile!!</p>
<p>Timer n: Timer 1-5</p> <p>TCCRnA (Timer/Counter n Control Register A): PWM</p> <p>TCCRB (Timer/Counter n Control Register B): Prescaler; Starten des Timers; Input Capture, CTC Beide TCCRn erst auf 0x00 setzen.</p> <p>Auch wenn keinen Prescaler will, muss man setzen</p> <p>TCNTn (Timer Counter n, 16 Bit): Aktueller Zählerstand. Anfangs auf 0 setzen.</p> <p>OCRnA, OCRnB, OCRnC (Output Compare Register, 16 Bit): Wert gegen den Zählerstand verglichen werden kann</p> <p>ICRn (Input Capture Register): Bei Input Capture erfasster Wert wird gespeichert</p> <p>TIMSKn: Aktivieren/Deaktivieren der Timer Interrupts</p> <p>TIFRn: Timer bezogene Interrupt Flags</p> <p>CTC Beispiel: TCCR4B = (1 << WGM42); ISR(TIMERA_OVF_vect) {}; Interrupt bei Timer 4 Overflow ISR(TIMERA_COMPA_vect) {}: Timer 4 compare A</p>	<pre>// counts number of overflows since last second volatile unsigned int overflow_counter = 0; void setup() { DDRB = (1 << DDB2); // initialize timer4 TCCR4A = 0x00; TCCR4B = 0x00; TCNT4 = 0x00; // activate clock, but don't use a prescaler p157 TCCR4B = (1 << CS40); // enable timer overflow interrupt TIMSK4 = (1 << TOIE4); sei(); // enable all interrupts } // ISR, called when timer overflow occurs ISR(TIMERA_OVF_vect) { // 16 MHz / 2^16 = 244,1 -> // during 1 second approx. 244 overflow events if (overflow_counter == 244) { overflow_counter = 0; PINA = (1 << PINA2); // toggle LED } overflow_counter++; }</pre>	<p>Atmega2560 Systemtakt = 16Mhz</p> <p>16Bit Timer => Timer läuft nach $\frac{2^{16}-1}{16 \text{ MHz}} = 4 \text{ ms}$ über</p> <p>Prescaler Fallende Flanke des Prescalers an Bit Qn triggert Counter.</p> <p>Vorteil großer Prescaler: Messen langer Zeiten möglich.</p> <p>kleinstes messbares Zeitintervall ohne Prescaler: $\frac{1}{16 \text{ MHz}} = 62 \text{ ns}$</p> <p>mit f/1024 Prescaler: $\frac{1}{16 \text{ MHz} / 1024} = 64 \mu \text{s}$ (1Tick)</p> <p>Nachteil: Schlechtere Auflösung => immer kleinstmögliche mit 16 Bit Timer? Takt: 1 MHz (Bei 16MHz bis 16*3*1000000) $\frac{3000000}{x} = 2^{16} - 1 \Rightarrow x = 45,8 \Rightarrow 64 \text{ Prescaler}$</p>	<p>Input Capture Bei externen oder internen Signalen/Ereignissen wird aktueller Zählerstand in ICRn gespeichert u. Flag ges</p>  <p>Output Compare Bei Erreichen eines konfigurierten Zählerstandes wird Interrupt ausgelöst, oder best. Signal erzeugt.</p>  <p>CTC Mode (Clear Timer on Compare Match) TOP Wert in OCRnA oder ICRn konfiguriert. Zähler bei Erreichen des Zählerstandes automatisch auf 0.</p>
<p>Pulsweitenmodulation n: Nummer des Timers</p> <p>TCCRnA Compare Output Mode Fast PWM usw</p> <p>TCCRnB Fast PWM; Prescaler</p> <p>OCnA, OCnB, OCnC (Output Compare Pins): PWM-Ausgang Inverting oder non-Inverting Mode Output Compare Pins müssen als Ausgang im DDR Register konfiguriert sein!</p> <p>OCRnX (Output Compare Register): Vergleichswert (Schwellwert) muss gesetzt werden, der jeweils PWM-Ausgang OCnX beeinflusst.</p>	<pre>void setup() { // set pin PB5 (pin5 of port B) to output: this is the PWM pin // (alternative function: alternative function of PB5: OC4C) DDRB = (1 << DDB5); // to be safe: initialize counter control registers to zero TCCR4A = 0x00; TCCR4B = 0x00; // Fast PWM mode, counter TOP value taken from ICR, TCCR4A = (1 << WGM41); // WGM bits: "1101", manual p145 TCCR4B = (1 << WGM43) (1 << WGM42); // non-inverting mode: clear on compare match: manual p155, Table 17-4 TCCR4A = (1 << COM4A1); // good choice: use clk/8 prescaler -> 16 MHz / 8 = 2 MHz // counter counts up from 0 to 39999 within 20 ms TCCR4B = (1 << CS41); ICR4 = 40000; // configure period of PWM, i.e. TOP value in ICR register // initial pulse width / duty cycle 1,25 ms // -> (1,25 ms / 20 ms) * 40000 OCR4C = 2500; void loop() { // Max Wert: 2,4 us: 4800 // Min-Wert: 544 us: 1088 OCR4C = 1088; // duty cycle: min value delay(3000); OCR4C = 4800; // duty cycle: max value delay(3000); } }</pre>	<p>Signal mit konstanter Periode, aber variabler Pulsdauer wird erzeugt.</p> <p>Duty Cycle: t/T (= Pulsdauer / Periodendauer).</p> <p>Nur Duty Cycle wird ausgewertet, nicht Periodendauer.</p>  <p>TOP: ICRn Register (oder andere siehe S145 Tabelle)</p> <p>CMP: OCRnX Register</p>  <p>In manchen Modi kann man TOP/CMP nur ändern, wenn Zähler gerade auf BOTTOM/TOP ist. („Update of OCRnX auf...“ in Tabelle)</p> 	<p>Inverting u. Non-Inverting Mode Non-Inverting: siehe Links. Inverting: PWM Ausgang genau andersrum</p> <p>TCCRnX Register</p> <p>Up-Down-Counter doppelte Periodendauer, geringere Auflösung</p> <p>TOP immer genau in der Mitte</p> <p>Maximaler Zählerwert: z.B. $2^{16}-1$</p>  <p>Zählerstand</p>  <p>PWM Signal (OCnX)</p> <p>Tabelle S145 TCCRnA u TCCRnB Fast PWM (Up-Counter), PWM (Up-Down Counter)</p>
<p>Analoge Ein-/Ausgabe</p> <p>ADMUX Referenzspannung wählen Analoge Eingangspins für A/D Umsetzung wählen</p> <p>ADCSRA Aktivieren und Starten der A/D Umsetzung (ADSC für manuelles triggern, wenn fertig wieder 0) (ADSC für auto trigger z.B. bei Free running u. timer ovr (z.B. beim Sensor) im Free Running Mode nur einmal ADSC triggern)</p> <p>Prescaler (ADPS1! 50-200kHz bei int. AD-Wandler) Interrupts</p> <p>ADCSRB Analoge Eingangspins für A/D Umsetzung wählen Single Ended oder Differential Conversion Free Running Mode oder manuelles Triggern</p> <p>ADCL, ADCH Speichert Ergebnis der A/D Umsetzung Erst ADCL, dann ADCH lesen (atomarer Zugriff) ADCL + 256 * ADCH</p>	<pre>void setup() { // activate serial console Serial.begin(9600); // enable ADC functionality ADSCRA = (1 << ADEN); // use /128 prescaler (ADC requires 50 kHz to 200 kHz, see manual p271, // but system clock is 16 MHz) ADSCRA = (1 << ADPS2) (1 << ADPS1) (1 << ADPS0); // select ADC2 as input pin (there is one AD converter for the 16 AD: ADMUX = (1 << MUX1); // use reference voltage 5V (Note: ADC is AREF), manual, p281 ADMUX = (1 << REFS0); } void loop() { // trigger ADC conversion ADSCRA = (1 << ADSC); // wait until conversion is finished, see manual p286 while (ADSCRA & (1 << ADSC)); // read analog value, first LOW then HIGH register unsigned int read = ADCL + 256 * ADCH; double result = 5.0 * read / 1024.0; Serial.println(result); Serial.println(" Volt"); }</pre>	<p>A/D Wandlung</p> <p>Auflösung: Wie viel Spannungsunterschied pro Stufe? $V_{ref} / 2^n$</p> <p>Repräsentant eines Intervalls liegt in Intervallmitte um Quantisierungsfehler zu vermeiden</p> <p>Erstes Intervall: Repr. von 000 Stufenbreite 1/2 LSB Letztes Intervall: Stufenbreite 1/2 LSB $r = 10$ bei ATmega</p>  <p>Fehlerquellen Quantisierungsrauschen Umsetzungszeit Änderung des Eingangs während der Umsetzung</p> <p>Ersatzwiderstand von zwei parallelen Widerständen $R_{1,2} = \frac{R_1 \cdot R_2}{R_1 + R_2}$</p>	<p>Ansätze zur A/D Wandlung Komparator Parallelverfahren, Zählverfahren, Wägeverfahren: $r=3, V_{in}=(1011)V \rightarrow 1000 - 1100 - 1010 - 1011$</p> <p>A/D Umsetzung beim Atmega Nur 1 interner A/D Umsetzer, aber 16 analoge Eingangspins können an A/D Umwandler weitergeleitet werden. Konfigurierbar, welcher Eingang an A/D Um. Weitergeleitet wird</p> <p>Trigger: Manuelles Auslösen: durch Codeanweisung Free Running Mode: Endlosschleife Auto Trigger: angestoßen durch Timeroverflow, Komparatorausgang etc</p> <p>Erkennen, dass A/D Umsetzung beendet wurde: Auswerten eines speziellen Flags, oder durch speziellen Interrupt</p> <p>Wertebereich: Single-Ended Conversion: $[0, V_{ref}]$ Differential Conversion: $[-V_{ref}/2, V_{ref}/2]$</p>

<p>Watchdog, Energiesparmodus, Reset</p> <p>WDTCSR</p> <p>Watchdog Modul Konfiguration</p> <p>WDP's: Prescaler für Watchdog Zeit</p> <p>Achtung!!: einmal nur "="</p> <p>Spezielles vorgehen zum Beschreiben des Registers! (Damit nicht ausversehen)</p> <p>MCUSR</p> <p>Informationen über Ursache des Resets (nach Neustart abrufbar)</p> <p>wdt_reset() (in C) (Assembler: WDT)</p> <p>Watchdog Timer zurücksetzen</p> <p>SMCR</p> <p>Energiesparmodus wählen</p> <p>sleep_mode()</p> <p>(Assembler: SE-Bit in SMCR setzen, dann SLEEP-Instruktion) sleep_mode() macht das automatisch!!</p> <p>Energiesparmodus aktivieren</p>	<pre>void setup() { Serial.begin(9600); Serial.println("System restart"); // start watchdog c11(); wdt_reset(); // preparation for configuration: write logic // one to WDCE and WDE, manual p61 WDTCR = (1<<WDCE) (1<<WDE); // immediately afterwards (within 4 clock cycles): // set timeout to 4 seconds and start watchdog // Hint: WDCE bit must be cleared WDTCR = (1<<WDP3) (1<<WDE); // configure interrupt (alternative: use Arduino commands) DDRC &= ~(1 << DDRC); // configure P0 as input PORTD = (1 << PORTD0); // pull up (manual p68) EIMSK = (1 << INT0); // turn on INT0 EICRA = (1 << ISC01); // set INT0 to trigger on falling edge sei(); } void loop() {} ISR (INT0_vect){ wdt_reset(); Serial.println("ResetWDT"); }</pre>	<p>Watchdog</p> <p>Timer, der hoch oder runterzählt. Muss vor Überlauf zurückgesetzt werden. Sonst: Interrupt oder Reset.</p> <p>Aufgaben:</p> <p>Überprüfung: Codestellen in vorgegebener Zeit erreicht? SW noch aktiv und nicht abgestürzt?</p> <p>Bei Timeout: Überführen in wohldefinierten Zustand. Neustart oder Interrupt auslösen.</p> <p>Erkennt Probleme, löst sie aber nicht!</p> <p>Prescaler: Beeinflusst Zeit bis Watchdog Timeout</p> <p>Energiesparmodus</p> <p>Energieverbrauch verringern durch: Systakt verlangsamen, Betriebsspannung verringern, abschalten nicht benötigter Module (Energiesparmodi (ESM))</p> <p>ESM unterscheiden sich bzgl. Abgeschalteter Komponenten und ausweckender Ereignisse (Ext Interrupts, Watchdog Interrupt, Speicherzugriff beendet, Timer, Anlegen einer (leeren) ISR und Aktivieren des Interrupts genügt).</p> <p>Aufwachen kann verzögert passieren</p> <p>Energiesparmodi beim Atmega2560:</p> <p>Idle Mode, ADC Noise Reduction Mode, Power Save Mode, Power Down Mode, Standby Mode</p>	<p>Reset</p> <p>System von wohldefiniertem Zustand starten.</p> <p>Init. aller Register u. I/O Ports auf Default Werte, künstl. Delay, damit sich Spannungswerte stabilisieren, erste Instruktion an Adresse 0x0000ausführen, wo im Normalfall JMP zur Reset-Routine ist, Reset Routine Initialisiert stack pointer u. letzte Anweisung ist JMP in Main-Routine (setup)</p> <p>Sensordaten</p> <p>In bestimmten Bereich linearer Zusammenhang zw. Messgröße (z.B. °C) u. Ausgangsspannung.</p> <p>Beispiel TMP 36: -40°C – 125°C</p> <p>750mV bei 25°C. Output <i>Scale Factor</i> 10mV/°C</p> <p>Min Ausgangsspannung: 100 mV</p> <p>Max Ausgangsspannung: 1750 mV</p> <p>Max Ausgangsspannung sollte möglichst knapp unter Referenzspannung liegen.</p> <p>Binäre Zahl (bei $V_{ref} = 2,56V$):</p> <p>Max: $1,750V / 2,56V * 2^{10} = 700$</p> <p>Min: $0,100V / 2,56V * 2^{10} = 40$</p> <p>Binäre Zahl in Messgröße: $y=mx + t$</p> <p>y: Messgröße, x: binäre Zahl</p> <p>-40°C = $m*40 + t$ [°C] 125°C = $m*700 + t$ [°C]</p>																																				
<p>Kommunikationsschnittstellen</p> <p>USART Register:</p> <p>UDR</p> <p>UCSRnA</p> <p>UCSRnB</p> <p>UCSRnC</p> <p>UBRRnL</p> <p>SPI Register:</p> <p>SPCR</p> <p>SPSR</p> <p>SPDR</p>		<p>Klassifizierung</p> <p>Seriell vs parallel vs </p> <p>Synchron (meist eigener Takt für Datenleitung) vs asynchron (Empfänger muss Takt d. Senders kennen)</p> <p>Bus (Mehr als zwei Geräte verbunden, erfordert Adressierung) vs Point-to-Point</p> <p>Vollduplex (Datenübertragung in beide Richtungen gleichzeitig möglich, separate Leitungen für Senden u. Empfangen) vs halbduplex</p> <p>Peer-to-Peer vs Master-Slave (Nur Master darf Kommunikation starten)</p> <p>Differential (Spannungsunterschied zw. 2 Leitungen trägt Information) vs Single-Ended (Gemeinsame GND Leitung für alle Datenleitungen)</p> <table border="1"> <thead> <tr> <th></th> <th>UART</th> <th>SPI</th> <th>I²C</th> </tr> </thead> <tbody> <tr> <td>Seriell</td> <td>Ja</td> <td>Ja</td> <td>Ja</td> </tr> <tr> <td>Duplex</td> <td>Ja</td> <td>Ja</td> <td>Nein</td> </tr> <tr> <td>Synchron</td> <td>Nein</td> <td>Ja</td> <td>Ja</td> </tr> <tr> <td></td> <td>kein Takt</td> <td></td> <td></td> </tr> <tr> <td>Bus</td> <td>Nein</td> <td>Jein</td> <td>Ja</td> </tr> <tr> <td>Anz. Leitungen</td> <td>3</td> <td>5</td> <td>3</td> </tr> <tr> <td>Datenrate</td> <td>BAUD = $f_{osc} / (16(UBRRn+1))$</td> <td>$f_{osc} / 128$</td> <td>Max 400 kbit/s</td> </tr> <tr> <td>ATmega2560</td> <td></td> <td>$- f_{osc} / 2$</td> <td></td> </tr> </tbody> </table>		UART	SPI	I²C	Seriell	Ja	Ja	Ja	Duplex	Ja	Ja	Nein	Synchron	Nein	Ja	Ja		kein Takt			Bus	Nein	Jein	Ja	Anz. Leitungen	3	5	3	Datenrate	BAUD = $f_{osc} / (16(UBRRn+1))$	$f_{osc} / 128$	Max 400 kbit/s	ATmega2560		$- f_{osc} / 2$		<p>UART (oder SCI)</p> <p>2 Datenleitungen: TxD und RxD</p> <p>Sender u. Empfänger müssen Baudrate kennen</p> <p>Übertragung von UART-Frames D(E)O(N)S</p> <p>Beispiel: 8E1: 1 Startbit8 Datenbits, gerade Parität, 1 Stopbit</p> <p>SPI (hohe Geschwindigkeit)</p> <p>Master-Slave. 4 Datenleitungen:</p> <p>MOSI: Master Out, Slave IN (8 Bit Schieberegister)</p> <p>MISO: Master In, Slave Out (8 Bit Schieberegister)</p> <p>SCK: System Clock, \overline{SS}: Slave Select</p> <p>I²C, TWI (Viele Geräte)</p> <p>Bus mit 7 Bit Adressierung</p> <p>SCL: Serial Clock Line, SDA: Serial Data Line</p> <p>Startbedingung: Fallende Flanke: SDA+SCL == HIGH</p> <p>Adresse anlegen -> R/w: Master spezifiziert, ob Lese oder Schreibzugriff -> Datentransfer</p> <p>Stoppbedingung: Steigende Fl.: SDA+SCL == HIGH</p>
	UART	SPI	I²C																																				
Seriell	Ja	Ja	Ja																																				
Duplex	Ja	Ja	Nein																																				
Synchron	Nein	Ja	Ja																																				
	kein Takt																																						
Bus	Nein	Jein	Ja																																				
Anz. Leitungen	3	5	3																																				
Datenrate	BAUD = $f_{osc} / (16(UBRRn+1))$	$f_{osc} / 128$	Max 400 kbit/s																																				
ATmega2560		$- f_{osc} / 2$																																					
<p>Peripherie</p>		<p>CGRAM</p> <p>DDRAM</p> <p>Cursor</p> <p>4-Bit Modus</p> <p>Initialisierung Liquid Crystal</p>																																					
<p>SW-Download / Debugging</p>		<p>JTAG</p>																																					
<p>Automaten</p>																																							

Register

Digital IO:

- DDRx (Data Direction Register):
 - o Entsprechendes Bit auf 1 für Ausgang, oder 0 für Eingang
- PORTx (Port Register):
 - o Wenn Pin auf Ausgang, dann 1 = 5V und 0 = 0V
- PINx (Port Input Register):
 - o Wenn Pin auf Eingang, dann 1 = HIGH liegt an und 0 = LOW liegt an

Timer:

- TCCRnA (Timer/Counter n Control Register A):
- TCCRnB (Timer/Counter n Control Register B):
 - o Prescaler
 - o Starten des Timers
 - o Input Capture
- TCNTn (Timer Counter n, 16 Bit):
 - o Aktueller Zählerstand
- OCRnA (Output Compare Register A, 16 Bit):
 - o Wert gegen den Zählerstand verglichen werden kann
- OCRnB (Output Compare Register B, 16 Bit):
 - o Wert gegen den Zählerstand verglichen werden kann
- ICRn (Input Capture Register):
 - o Bei Input Capture erfasster Wert wird gespeichert
- TIMSKn:
 - o Aktivieren/Deaktivieren der Timer Interrupts
- TIFRn:
 - o Timer bezogene Interrupt Flags

Pulsweitenmodulation:

- OCnA:
- OCnB:
- OCnC (Output Compare Pins):
 - o Inverting oder non-Inverting Mode
 - o Output Compare Pins müssen als Ausgang im DDR Register konfiguriert sein!
- OCRnX (Output Compare Register):
 - o Vergleichswert muss gesetzt werden

Interrupts:

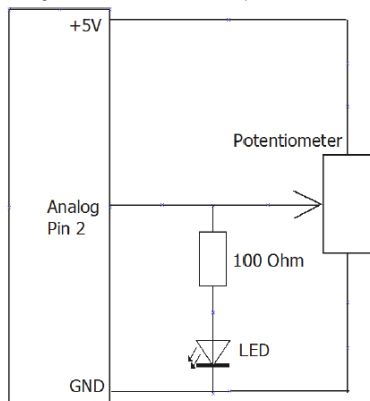
- sei() (Set Enable Interrupt):
 - o Interrupts global aktivieren
- SREG:
 - o 1 Bit hier setzen statt sei() möglich
- EIMSK:
 - o De/aktivieren von speziellen Interrupts
- EIFR:
 - o Interrupt Flags
- EICRA:
- EICRB:
 - o Steigende/fallende Flanke?

Analoge IO:

- ADMUX:
 - o Referenzspannung wählen
 - o Analoge Eingangspins für A/D Umsetzung wählen
- ADCSRB:
 - o Analoge Eingangspins für A/D Umsetzung wählen
 - o Single Ended oder Differential Conversion
 - o Free Running Mode oder manuelles Triggern
- ADCSRA:
 - o Aktivieren und Starten der A/D Umsetzung
 - o Prescaler
 - o Interrupts
- ADCL u. ADCH:
 - o Speichert Ergebnis der A/D Umsetzung
 - o Erst ADCL, dann ADCH lesen (atomarer Zugriff)

TODO:

- Übung 7 vielleicht noch mehr Beispiele



```

void setup(){
  Serial.begin(9600);
  // enable ADC functionality
  ADCSRA |= (1 << ADEN);
  // use /128 prescaler, see manual p271
  ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
  // select autotrigger, to use free-running mode
  ADCSRA |= (1 << ADSCF);
  // use free-running mode
  ADCSRA |= (1 << ADSCF);
  // select ADC2 as input pin
  ADMUX |= (1 << MUX1);
  // use reference voltage 2,56 V, manual p281
  ADMUX |= (1 << REFS1) | (1 << REFS0);
  // start conversion
  ADCSRA |= (1 << ADSC);
}

void loop()
{
  // note: conversion is automatically triggered in free running mode
  // read analog value, first LOW then HIGH register
  unsigned int read = ADCL + 256 * ADCH;

  // convert integer value into temperature
  double temperature = 0.25 * read - 50;
  Serial.println(temperature);
  delay(1000);
}

```

Sensor Beispiel: }