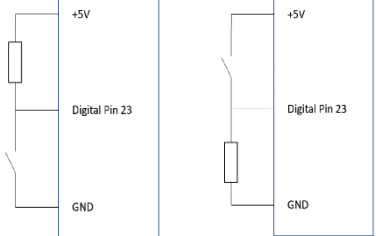

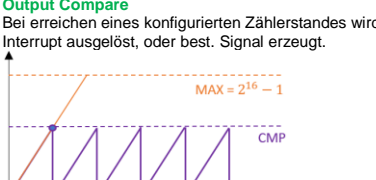
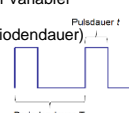
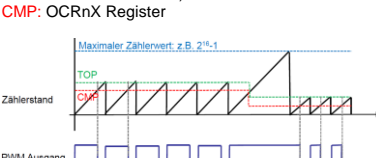
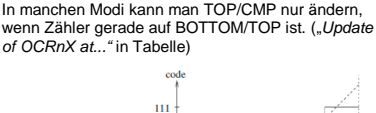
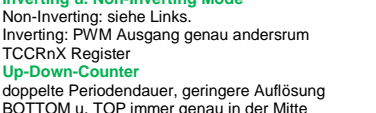
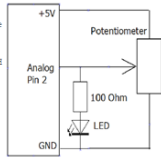


<p><b>DDR<sub>x</sub></b> (Data Direction Register): Entsprechendes Bit auf 1 für Ausgang, oder 0 für Eingang</p> <p><b>PORT<sub>x</sub></b> (Port Register): Wenn <b>Pin auf Ausgang</b>, dann 1 = 5V und 0 = 0V</p> <p><b>PIN<sub>x</sub></b> (Port Input Register): Wenn <b>Pin auf Eingang</b>, dann 1 = 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 &lt;&lt; DDB4);   //pinMode(pin, OUTPUT);   DDRA  = ~(1 &lt;&lt; DDA0);   //pinMode(sw, INPUT);   PORTA  = ~(1 &lt;&lt; PA4);   //digitalWrite(pin, LOW); }  void loop() {   if (PINA &amp; (1 &lt;&lt; PINA0)) {     //if (digitalRead(sw)) {     PORTA  = (1 &lt;&lt; PA4);     delay(2000);     PORTA  = ~(1 &lt;&lt; PA4);     //digitalWrite(pin, LOW);   } }</pre> <p><b>Mikrocontroller Bestandteile:</b> MicroProzessor, Timer, Schnittstellen, Speicher, AD-Wandler</p> <p><b>Entwicklerboard:</b> Arduino Mega, Mikroprozessor: Atmega2560</p> <p><b>Cross-Compilation:</b> Programm wird nicht auf Zielplattform (Mikrocontroller), sondern auf anderer Plattform übersetzt</p> <p><b>Flashen:</b> hex-Datei von PC an Entwicklerboard senden</p> <p><b>Harvard-Architektur:</b> Daten- und Instruktionsspeicher getrennt</p> <p><b>Instruktionsspeicher:</b> Nicht flüchtiger Flash Speicher</p> <p><b>Daten:</b> in flüchtigem SRAM</p> <p><b>SRAM und DRAM sind flüchtig.</b> Rest nicht flüchtig.</p>	<p><b>Nichtflüchtige Speicher:</b></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 &lt;&lt; Bitnummer);</p> <p><u>Einzelne Bits löschen:</u> x &amp;= ~(1 &lt;&lt; Bitnummer)   (1 &lt;&lt; Bitnummer2));</p> <p><u>Testen ob Bit auf 1:</u> if (DDRC &amp; (1 &lt;&lt; Bitnummer)) {}</p> <p><u>Testen ob Bit auf 0:</u> if (!(DDRC &amp; (1 &lt;&lt; Bitnummer))) {}</p> <p><u>Alle Bits umdrehen:</u> x = 0xFF ^ x</p> <p><u>LED togglen:</u> PINA ^= (1 &lt;&lt; PINA2);</p> <p>Vorwiderstand berechnen: <math display="block">R_v = \frac{U_{ges} - U_F}{I_F}</math></p>	 <p><b>Pull-Up / Active Low</b> Bei offenem Taster wird Spannung am Pin auf HIGH gezogen.</p> <p><b>Pull-Down / Active High</b> Bei offenem Taster wird Spannung am Eingang Auf LOW gezogen.</p> <p><b>Entprellung</b> Einmaliges betätigen eines Schalters führt evtl zu mechanischen Vibrationen. SW-Lsg: Künstliche Wartezeit nach Zustandswechsel – Bis Schalter eingeschwungen.</p>
<p><b>Interrupts</b> sei(): Interrupts global aktivieren (oder: SREG  = 128) cli(): Interrupts global deaktivieren</p> <p><b>SREG</b> (AVR Status Register): Bit 7 auf 1 = sei();</p> <p><b>EIMSK</b> (External Interrupt Mask Register): Speziellen Interrupt de-/aktivieren</p> <p><b>EICRA</b> (External Interrupt Control Register A) <b>EICRB</b> (External Interrupt Control Register B): ISCn0 und ISCn1. Falling oder Rising Edge. n ist die Interrupt Nummer.</p> <p><b>EIFR</b> (External Interrupt Flag Register): Wenn Interrupt ausgelöst: Bit ist 1</p> <p><b>ISR</b> (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 &lt;&lt; INT2); //pin 19 EICRA  = (1 &lt;&lt; ISC20)   (1 &lt;&lt; ISC21); // rising edge }  void loop() {   Serial.println("Zählerstand: ");   Serial.println(counter);   delay(3000); }  ISR (INT2_vect) {   if (millis() - time &gt;= 250) {     time = millis(); //Entprellung     counter++;   } }</pre>	<p><b>Busy Waiting:</b> while (DDRC &amp; (1 &lt;&lt; DDC3));</p> <p><b>Polling:</b> periodisches Abfragen, ob Ereignis eingetreten</p> <p><b>Interrupt:</b> 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><b>Trap:</b> Art von Interrupt, die aber synchron und reproduzierbar ist. z.B. System Call, Div durch 0 ...</p> <p><b>Interrupt Request:</b> Interruptereignis – [InterruptController] – über IRQ Eingang Unterbrechungsanforderung an CPU – CPU unterbricht Programm und startet Unterbrechungsroutine</p> <p><b>Interrupt Vector Table:</b> Welches Interruptereignis gehört zu welcher ISR? Jede Vectornummer hat eine zugehörige Programmadresse. ISR ist selbst nicht unterbrechbar (1 Bit SREG)</p>	<p><b>Externe Interrupts</b> Controller tastet zu Beginn jedes Taktzyklus ab. Falls Interrupt aktiviert, Aufruf der ISR. Probleme: Leichte Verzögerung, „Prrellung“</p> <p><b>Interne Interrupts</b> Timer, A/D-Wandler Bei Auslauf eines Timers unterbricht HW Ausführung der normalen Software</p> <p><b>Volatile</b> Variable wird vor jedem Lesen aus SRAM gelesen und nach jedem Schreiben in SRAM geschrieben <b>!!Globale Variablen die in ISR vorkommen immer volatile!!!</b></p>
<p><b>Timer</b> n: Timer 1-5</p> <p><b>TCCRnA</b> (Timer/Counter n Control Register A): PWM</p> <p><b>TCCRB</b> (Timer/Counter n Control Register B): Prescaler; Starten des Timers; Input Capture, CTC Beide TCCRn erst auf 0x00 setzen.</p> <p><b>Auch wenn keinen Prescaler will, muss man setzen</b></p> <p><b>TCNTn</b> (Timer Counter n, 16 Bit): Aktueller Zählerstand. Anfangs auf 0 setzen.</p> <p><b>OCRnA, OCRnB, OCRnC</b> (Output Compare Register, 16 Bit): Wert gegen den Zählerstand verglichen werden kann</p> <p><b>ICRn</b> (Input Capture Register): Bei Input Capture erfasster Wert wird gespeichert</p> <p><b>TIMSKn:</b> Aktivieren/Deaktivieren der Timer Interrupts</p> <p><b>TIFRn:</b> Timer bezogene Interrupt Flags</p> <p><b>CTC Beispiel:</b> TCCR4B  = (1 &lt;&lt; WGM42); ISR(TIMER4_OVF_vect) {}; Interrupt bei Timer 4 Overflow ISR(TIMER4_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 &lt;&lt; DDB2);   // initialize timer4   TCCR4A = 0x00;   TCCR4B = 0x00;   TCNT4 = 0x00;   // activate clock, but don't use a prescaler p157   TCCR4B  = (1 &lt;&lt; CS40);   // enable timer overflow interrupt   TIMSK4  = (1 &lt;&lt; TOIF4);   sei(); // enable all interrupts }  // ISR, called when timer overflow occurs ISR(TIMER4_OVF_vect) {   // 16 MHz / 2^16 = 244,1 -&gt;   // during 1 second approx. 244 overflow events   if (overflow_counter == 244) {     overflow_counter = 0;     PINA  = (1 &lt;&lt; PINA2); // toggle LED   }   overflow_counter++; }</pre>	<p>Atmega2560 Systemtakt = 16Mhz</p> <p>16Bit Timer =&gt; Timer läuft nach <math>\frac{2^{16}-1}{16 \text{ MHz}} = 4 \text{ ms}</math> über</p> <p><b>Prescaler</b> Fallende Flanke des Prescalers an Bit Qn triggert Counter.</p> <p><b>Vorteil</b> großer Prescaler: Messen langer Zeiten möglich.</p> <p>kleinstes messbares Zeitintervall ohne Prescaler: <math>\frac{1}{16 \text{ MHz}} = 62 \text{ ns}</math></p> <p>mit f/1024 Prescaler: <math>\frac{1}{16 \text{ MHz} / 1024} = 64 \mu \text{s}</math> (1Tick)</p> <p><b>Nachteil:</b> Schlechtere Auflösung. =&gt; immer kleinstmögliche 16 Bit Timer? Takt: 1 MHz (Bei 16MHz bis 16*3*1000000) <math>\frac{3000000}{x} = 2^{16} - 1 \Rightarrow x = 45,8 \Rightarrow 64 \text{ Prescaler}</math></p>	<p><b>Input Capture</b> Bei externen oder internen Signalen/Ereignissen wird aktueller Zählerstand in ICRn gespeichert u. Flag ges</p>  <p><b>Output Compare</b> Bei Erreichen eines konfigurierten Zählerstandes wird Interrupt ausgelöst, oder best. Signal erzeugt.</p>  <p><b>CTC Mode (Clear Timer on Compare Match)</b> <b>TOP</b> Wert in OCRnA oder ICRn konfiguriert. Zähler bei Erreichen des Zählerstandes automatisch auf 0.</p>
<p><b>Pulsweitenmodulation</b> n: Nummer des Timers</p> <p><b>TCCRnA</b> Compare Output Mode Fast PWM usw</p> <p><b>TCCRnB</b> Fast PWM; Prescaler</p> <p><b>OCnA, OCnB, OCnC</b> (Output Compare Pins): PWM-Ausgang Inverting oder non-Inverting Mode Output Compare Pins müssen als <b>Ausgang im DDR</b> Register konfiguriert sein!</p> <p><b>OCRnX</b> (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 &lt;&lt; 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 &lt;&lt; WGM41); // WGM bits: "1101", manual p145   TCCR4B  = (1 &lt;&lt; WGM43)   (1 &lt;&lt; WGM42);    // non-inverting mode: clear on compare match: manual p155, Table 17-4   TCCR4A  = (1 &lt;&lt; COM4A1);    // good choice: use clk/8 prescaler -&gt; 16 MHz / 8 = 2 MHz   // counter counts up from 0 to 39999 within 20 ms   TCCR4B  = (1 &lt;&lt; CS41);    ICR4 = 40000; // configure period of PWM, i.e. TOP value in ICR register    // initial pulse width / duty cycle 1,25 ms   // -&gt; (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><b>Duty Cycle:</b> <math>t/T</math> (= Pulsdauer / Periodendauer).</p> <p>Nur Duty Cycle wird ausgewertet, nicht Periodendauer.</p>  <p><b>TOP:</b> ICRn Register (oder andere siehe S145 Tabelle)</p> <p><b>CMP:</b> OCRnX Register</p>  <p>In manchen Modi kann man TOP/CMP nur ändern, wenn Zähler gerade auf BOTTOM/TOP ist. („Update of OCRnX at...“ in Tabelle)</p> 	<p><b>Inverting u. Non-Inverting Mode</b> Non-Inverting: siehe Links. Inverting: PWM Ausgang genau andersrum</p> <p>TCCRnX Register</p> <p><b>Up-Down-Counter</b> doppelte Periodendauer, geringere Auflösung</p> <p>BOTTOM u. TOP immer genau in der Mitte</p> <p>Maximaler Zählerwert: z.B. <math>2^{16}-1</math></p>  <p>PWM Signal (OCnX)</p> <p>Tabelle S145 TCCRnA u TCCRnB Fast PWM(Up-Counter), PWM(Up-Down Counter)</p>
<p><b>Analoge Ein-/Ausgabe</b></p> <p><b>ADMUX</b> Referenzspannung wählen Analoge Eingangspins für A/D Umsetzung wählen</p> <p><b>ADCSRA</b> Aktivieren und Starten der A/D Umsetzung (ADSC für manuelles triggern, wenn fertig wieder 0) (ADATE für auto trigger z.B. bei Free running u. timer ovf (z.B. beim Sensor) im Free Running Mode nur einmal AD-UMSC triggern)</p> <p><b>Prescaler (ADPS1! 50-200kHz bei int. AD-Wandler)</b> Interrupts</p> <p><b>ADCSRB</b> Analoge Eingangspins für A/D Umsetzung wählen Single Ended oder Differential Conversion Free Running Mode oder manuelles Triggern</p> <p><b>ADCL, ADCH</b> 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   ADCSRA  = (1 &lt;&lt; ADEN);    // use /128 prescaler (ADC requires 50 kHz to 200 kHz, see manual p271,   // but system clock is 16 MHz)   ADCSRA  = (1 &lt;&lt; ADPS2)   (1 &lt;&lt; ADPS1)   (1 &lt;&lt; ADPS0);    // select ADC2 as input pin (there is one AD converter for the 16 AD:   ADMUX  = (1 &lt;&lt; MUX1);    // use reference voltage 5V (Note: AVCC is AREF), manual, p281   ADMUX  = (1 &lt;&lt; REFS0); }  void loop() {   // trigger ADC conversion   ADCSRA  = (1 &lt;&lt; ADSC);    // wait until conversion is finished, see   while (ADCSRA &amp; (1 &lt;&lt; ADSC));    // read analog value, first LOW then HIGH   unsigned int read = ADCL + 256 * ADCH;    double result = 5.0 * read / 1024.0;   Serial.print(result);   Serial.println(" Volt"); }</pre> 	<p><b>A/D Wandlung</b></p> <p><b>Auflösung:</b> Wie viel Spannungsunterschied pro Stufe? <math>V_{ref} / 2^r</math></p> <p><b>Repräsentant</b> eines Intervalls liegt in Intervallmitte um Quantisierungsfehler zu vermeiden</p> <p>Erstes Intervall: Repr. Von 000 Stufenbreite 1/2 LSB Letztes Intervall: Stufenbreite 1 1/2 LSB <math>r = 10</math> bei ATmega</p> <p><b>Fehlerquellen</b> Quantisierungsrauschen Umsetzungszeit Änderung des Eingangs während der Umsetzung</p> <p><b>Ersatzwiderstand</b> von zwei parallelen Widerständen <math display="block">R_{1,2} = \frac{R_1 \cdot R_2}{R_1 + R_2}</math></p>	<p><b>Ansätze zur A/D Wandlung</b> Komparator Parallelverfahren, Zählverfahren, <b>Wägeverfahren:</b> <math>r=3, V_{in}=(1011)V \rightarrow 1000 - 1100 - 1010 - 1011</math></p> <p><b>A/D Umsetzung beim Atmega</b> 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><b>Trigger:</b> Manuelles Auslösen: durch Codeanweisung Free Running Mode: Endlosschleife Auto Trigger: angestoßen durch Timeroverflow, Komparatorausgang etc</p> <p><b>Erkennen, dass A/D Umsetzung beendet wurde:</b> Auswerten eines speziellen Flags, oder durch speziellen Interrupt</p> <p><b>Wertebereich:</b> Single-Ended Conversion: <math>[0, V_{ref}]</math> Differential Conversion: <math>[-V_{ref}/2, V_{ref}/2]</math></p>



## WDTCSR, Energiesparmodus, Reset

### WDTCSR

Watchdog Modul Konfiguration  
WDP's: Prescaler für Watchdog Zeit  
**Achtung!!: einmal nur "="**

**Spezielles vorgehen zum Beschreiben des Registers! (Damit nicht ausversehen)**

### MCUSR

Informationen über Ursache des Resets (nach Neustart abrufbar)

`wdt_reset()` (in C) (Assembler: WDT)  
Watchdog Timer zurücksetzen

### SMCR

Energiesparmodus wählen

### sleep\_mode()

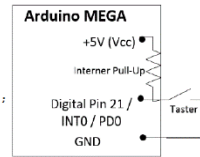
(Assembler: SE-Bit in SMCR setzen, dann SLEEP-Instruktion) `sleep_mode()` macht das automatisch!!  
Energiesparmodus aktivieren

```
void setup() {
  Serial.begin(9600);
  Serial.println("System restart");
  // start watchdog
  cli();
  wdt_reset();
  // preparation for configuration: write logic
  // one to WDCE and WDE, manual p61
  WDTCSR |= (1<<WDCE) | (1<<WDE);
  // immediately afterwards (within 4 clock cycles):
  // set timeout to 4 seconds and start watchdog
  // Hint: WDCE bit must be cleared
  WDTCSR = (1<<WDFR) | (1<<WDE);

  // configure interrupt (alternative: use Arduino commands)
  DDRC &= ~(1<<DDDR0); // configure PD0 as input
  PORTD |= (1<<PORTD0); // pull up (manual p68)
  EIMSK |= (1<<INT0); // turn on INT0
  PCRB = (1<<ISC01); // set INT0 to trigger on falling edge
  sei();
}

void loop() {}

ISR (INT0_vect) {
  wdt_reset();
  Serial.println("ResetWDT");
}
```



## Kommunikationsschnittstellen

### USART Register (n: welches der 4 UART Module):

`UDRn` (char a = `UDR2`, oder `UDR2 = a`)

Zu sendendes, oder empfangenes Byte

### UCSRnA

Übertragungsinfos, z.B. Übertragung erfolgt beendet?

### UCSRnB

USART-bezogene Interrupts, Empfänger / Receiver aktivieren

### UCSRnC

Modus wählen (synchron oder asynchron)

Datenformat(Stoppbit, Parität)

`UBRRnL` (8Bit) / `UBRRnH` (4 MS Bits)

Baudrate einstellen

### SPI Register:

Mstr: MOSI, SCK als Ausgang, Slv: MISO als Ausgng

### SPCR

Konfiguration: Aktivierung, Interrupts, Master/Slave?

Daten bei steigender oder Fallender Flanke lesen?

`SPSR` Infos, z.B. trat SPI Interrupt auf?

`SPDR` (`SPDR = 7` oder `char tmp = SPDR`)

Nach einer Taktperiode sind 8 Bits aus Register

gesendet worden u. 8 empfangene Bits stehen jetzt

im Register

```
//SLAVE
void setup() {
  Serial.begin(9600);
  // data direction: set MISO=DDRB3 to output, all other input
  DDRA = (1<<DDRA3);
  // Slave Init: Enable SPI,
  SPCR = (1<<SPE); //not setting MSTR means that it is a slave
}

// send and receive data
unsigned char spi_transceive(unsigned char data) {
  // start transmission by putting data into buffer
  SPDR = data;
  // wait until transmission completes
  while(! (SPSR & (1<<SPIF)));
  // return received data
  return (SPDR);
}

void loop() {
  char text[] = "Hallo Master!";
  for (int i = 0; i < sizeof(text); i++) {
    char received = spi_transceive(text[i]);
    Serial.print(received);
    delay(1000);
  }
}
```

```
//MASTER
void setup() {
  Serial.begin(9600);
  // data direction: set MOSI=DDRB2 and SCK=DDRB1 and SS=DDRB0 to output
  DDRA = (1<<DDRA2) | (1<<DDRA1) | (1<<DDRA0);
  // Master init: Enable SPI, set as master, set clock rate to fck/128
  SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR0) | (1<<SPR1);
}

// send and receive data
unsigned char spi_transceive(unsigned char data) {
  // set SS to low, activating slave, synchronization
  PORTB |= ~(1<<DDRB0);
  // start transmission by putting data into buffer
  SPDR = data;
  // wait until transmission completes
  while(! (SPSR & (1<<SPIF)));
  // return received data
  char result = SPDR;
  // set SS to high, deactivate slave
  PORTB |= (1<<DDRB0);
  return result;
}
```

## Debugging

**HW Breakpoint:** Spezielles HW-Modul überwacht Adressbus und wartet auf Holen einer Instruktion von einer bestimmten Adresse.

**SW Breakpoint:** Opcode am Ort des Breakpoints wird vorübergehend mit einer speziellen „Halte“-Instruktion ersetzt.

## Hilfsmethoden zum Debuggen:

LEDs, Taster und Schalter, UART (sout)

## Simulation

Target Controller wird auf Host System simuliert.

## HW-Debugging

Debugging direkt auf Ziel-Hardware – HW-Breakpoints auf Mikrocontroller

Schnittstelle: **JTAG**

Kommunikation von PC zu Mikrocontroller über Zusatz-HW / JTAG-Adapter  
Register die für Debugging wichtig sind, werden häufig in JTAG Chain eingebunden

## USART BEISPIEL

```
#define MYUBRR 103//calculated. table S.202
void setup() {
  // set BAUD RATE
  UBRR0L = (unsigned char) MYUBRR;
  UBRR0H = (unsigned char) (MYUBRR >> 8);
  UCSRB = (1<<TXEN0); // ENABLE transmitter
  // Set FRAME FORMAT: 8 data, 1 stop bit
  UCSRC = (1<<UCSZ01) | (1<<UCSZ00);
}

void uart_putchar(char c) {
  // Wait until data register empty.
  while (! (UCSR0A & (1<<UDRE0)));
  UDRO = c;
}

void loop() {
  char text[] = "Hallo!";
  int i = 0;
  for (i = 0; i < sizeof(text); i++) {
    uart_putchar(text[i]);
    delay(2000);
  }
}
```

```
// Function Set, 4-bit, 2 line mode, 5x8 dots
writeInstruction(0x01); // D7 state (DB3 = 1)
// Return Home, set cursor to beginning
writeInstruction(0x02);
// Entry Mode Set, increment cursor, no display shift
writeInstruction(0x06);
writeInstruction(0x01); // clear display: 0x01
delayMicroseconds(300);
// write an instruction, indicated by RS == LOW
void writeInstruction(char instr) {
  digitalWrite(RS, LOW); // when data > HIGH
  PORTC = instr;
  enablePulse(); // commit command, short pulse on E pin
  PORTC = (instr << 4);
  enablePulse();
}

void enablePulse() {
  digitalWrite(E, LOW);
  delayMicroseconds(3);
  digitalWrite(E, HIGH);
  delayMicroseconds(3); // enable pulse must be > 450 ns, see p49 of HD44780 manual
  digitalWrite(E, LOW);
  delayMicroseconds(200); // commands need > 37 us to settle
}

void loop() {
  // set cursor to beginning of first line: command 0x01
  writeInstruction(0x01); // (UDRAM address 0x00, see p11 of manual)
  char line1[] = "Tested System '1";
  for (int i = 0; i < strlen(line1); i++) {
    writeData(line1[i]);
  }

  // set cursor to beginning of 2nd line: command 0x02
  writeInstruction(0x02); // (UDRAM address 0x04, see p11 of manual)
  char line2[] = "Tacht Spas";
  for (int i = 0; i < strlen(line2); i++) {
    writeData(line2[i]);
  }
}
```

## Peripherie

## Watchdog

Timer, der hoch oder runterzählt. Muss vor Überlauf zurückgesetzt werden. Sonst: Interrupt oder Reset.

### Aufgaben:

Überprüfung: Codestellen in vorgegebener Zeit erreicht? SW noch aktiv und nicht abgestürzt?  
Bei Timeout: Überführen in wohldefinierten Zustand.  
Neustart oder Interrupt auslösen.

Erkennt Probleme, löst sie aber nicht!

**Prescaler:** Beeinflusst Zeit bis Watchdog Timeout  
**Energiesparmodus**

Energieverbrauch verringern durch: Systakt verlangsamen, Betriebsspannung verringern, abschalten nicht benötigter Module (Energiesparmodi (ESM))

**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).  
Aufwachen kann verzögert passieren

**Energiesparmodi beim ATmega2560:**

Idle Mode, ADC Noise Reduction Mode, Power Save Mode, Power Down Mode, Standby Mode

## Klassifizierung

**Seriell vs parallel** | vs **||||**

**Synchron** (meist eigener Takt für Datenleitung) vs **asynchron** (Empfänger muss Takt d. Senders kennen)

**Bus** (Mehr als zwei Geräte verbunden, erfordert Adressierung) vs **Point-to-Point**

**Vollduplex** (Datenübertragung in beide Richtungen gleichzeitig möglich, separate Leitungen für Senden u. Empfangen) vs **Halbduplex**

**Peer-to-Peer** vs **Master-Slave** (Nur Master darf Kommunikation starten)

**Differential** (Spannungsunterschied zw. 2 Leitungen trägt Information) vs **Single-Ended** (Gemeinsame GND Leitung für alle Datenleitungen)

	UART	SPI	I <sup>2</sup> C
Seriell	Ja	Ja	Ja
Duplex	Ja	Ja	Nein
Synchron	Nein	Ja	Ja
Baud konfig	Nein	Ja	Ja
Bus	Nein	Ja	Ja
Anz. Leitungen	3	5	3
Differential	Nein	Nein	Nein
Datenrate	BAUD = fosc / (16*(UBRRn+1))	f <sub>osc</sub> /128	Max 400 kbit/s
ATmega2560			

Mikrocontroller -> ASCII an DDRAM -> Steuereinheit schlägt Muster im CGRAM nach u. blendet es auf Display ein.

## CGRAM (definiert Aussehen von Schriftzeichen)

RAM: Standardzeichen, a-z, A-Z, ...

ROM: Benutzerdefinierte Zeichen

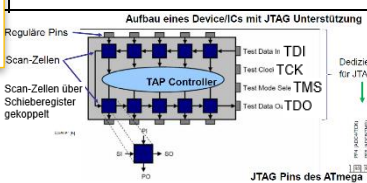
CGRAM-Adresse	16	8	4	2	1	Byte dezimal	Byte hexadezimal
0x00						16+1 = 17	0x11
0x01						16+1 = 17	0x11
0x02						16+2+1 = 19	0x13
0x03						16+4+1 = 21	0x15
0x04							
0x05						16+1 = 17	0x11
0x06						16+1 = 17	0x11
0x07						0	0x00

## DDRAM (welche Zeichen zeigt Display aktuell?)

Speicheradresse:

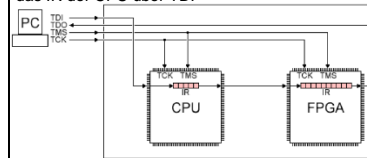
1 Zeilenmodus: 0x00 – 0x4F

2 Zeilenmodus: 0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)



## Beispiel Schreiben des IR-Register

Beide Controller müssen in SHIFT-IR Zustand versetzt werden (TMS: 01100). Dann senden der 10 Bits für IR des FPGAs über TDI und dann 5 Bits für das IR der CPU über TDO



## Debugging

**HW Breakpoint:** Spezielles HW-Modul überwacht Adressbus und wartet auf Holen einer Instruktion von einer bestimmten Adresse.

**SW Breakpoint:** Opcode am Ort des Breakpoints wird vorübergehend mit einer speziellen „Halte“-Instruktion ersetzt.

## Hilfsmethoden zum Debuggen:

LEDs, Taster und Schalter, UART (sout)

## Simulation

Target Controller wird auf Host System simuliert.

## HW-Debugging

Debugging direkt auf Ziel-Hardware – HW-Breakpoints auf Mikrocontroller

Schnittstelle: **JTAG**

Kommunikation von PC zu Mikrocontroller über Zusatz-HW / JTAG-Adapter  
Register die für Debugging wichtig sind, werden häufig in JTAG Chain eingebunden

## Automaten

```
int microPin = 24;
int ledPin = 21;
// states
typedef enum {
  WAIT_FOR_CLAP_ONE,
  ONE_CLAP_DETECTED,
  WAIT_FOR_CLAP_TWO,
  TWO_CLAP_DETECTED,
  state_t;
} event_t;

// global variables
state_t state;
// timestamp for 100 ms timer, 0 means inactive
unsigned long timer100 = 0;
// timestamp for 300 ms timer, 0 means inactive
unsigned long timer300 = 0;

// transition functions
void state() {
  void enterOneClapDetected() {
    state = ONE_CLAP_DETECTED;
    timer100 = millis(); // start timer for 100 ms
  }
  void enterWaitForClapTwo() {
    state = WAIT_FOR_CLAP_TWO;
    timer300 = millis(); // start timer for 300 ms
  }
  void enterWaitForClapOneTimer300() {
    state = WAIT_FOR_CLAP_ONE;
  }
  void enterWaitForClapOneMicroPin() {
    state = WAIT_FOR_CLAP_ONE;
    digitalWrite(ledPin, !digitalRead(ledPin)); // toggle LED
  }
}
```

```
// transition table
state_t (*state_table[3][4]) (void) = {
  //
  // WAIT_FOR_CLAP_ONE, NO_EVENT, MICRO_PIN_HIGH
  // ONE_CLAP_DETECTED, (stay), enterOneClapDetected,
  // WAIT_FOR_CLAP_TWO, (stay), enterWaitForClapOneTimer300
};

void setup() {
  pinMode(microPin, INPUT);
  pinMode(ledPin, OUTPUT);
  state = WAIT_FOR_CLAP_ONE; // initial state
  Serial.begin(9600);
}
```

```
void loop() {
  // detect events
  event_t event = NO_EVENT;
  if (digitalRead(microPin) == HIGH) {
    event = MICRO_PIN_HIGH;
  } // timer valid and 100 ms expired
  else if (timer100 <= millis() - timer100 > 100) {
    event = TIMER100_EXPIRED;
    timer100 = 0; // reset timer;
  }
  else if (timer300 <= millis() - timer300 > 300) {
    event = TIMER300_EXPIRED;
    timer300 = 0;
  }
  // use transition table to switch state
  state_table[state][event]();
}
```

## Reset

System von wohldefiniertem Zustand starten.  
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)

## Sensordaten

In bestimmten Bereich linearer Zusammenhang zw. Messgröße (z.B. °C u. Ausgangsspannung).  
Beispiel TMP 36: -40°C – 125°C  
750mV bei 25°C. Output Scale Factor 10mV/°C  
Min Ausgangsspannung: 100 mV  
Max Ausgangsspannung: 1750 mV  
Max Ausgangsspannung sollte möglichst knapp unter Referenzspannung liegen.

**Binäre Zahl** (bei  $V_{ref} = 2,56V$ ):

Max:  $(1,750V / 2,56V) * 2^{10} = 700$

Min:  $(0,100V / 2,56V) * 2^{10} = 40$

**Binäre Zahl in Messgröße:  $y=mx + t$**

$y$ : Messgröße,  $x$ : binäre Zahl

$-40^{\circ}C = m \cdot 40 + t [^{\circ}C]$   $125^{\circ}C = m \cdot 700 + t [^{\circ}C]$

## UART (oder SCI) (Arduino hat 4 USARTs)

2 Datenleitungen: Tx und Rx  
Sender u. Empfänger müssen Baudrate kennen  
Übertragung von UART-Frames D[E]O[N]S  
Bsp: 8E1: 1Startbit, 8Datenbits, even parity, 1 Stopbit

**SPI (hohe Geschwindigkeit, kein Overhead)**

Master-Slave. 4 Datenleitungen:

**MOSI:** Master Out, Slave IN (8 Bit Schieberegister)

**MISO:** Master In, Slave Out (8 Bit Schieberegister)

**SCK:** System Clock, SS: Slave Select (aktiver Slave)

**I<sup>2</sup>C, TWI (Viele Geräte)**

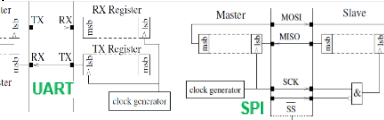
Bus mit 7 Bit Adressierung

**SCL:** Serial Clock Line, **SDA:** Serial Data Line

**Startbedingung:** (Fall. Flanke SDA) + (SCL == HIGH)

Adresse anlegen -> R/w: Master spezifiziert, ob Lese oder Schriebzugriff -> Slave: ACK -> Datentransfer

**Stoppbedingung:** (Steigende Fl. SDA)+(SCL==HIGH)



## Cursor

Display kann nicht alle Zeichen des DDRAM anzeigen  
Shift-Operationen zum Verschieben des sichtbaren Bereichs

Cursor zeigt auf Zeichen, das User verändert, wenn er WriteCommand zum LCD Display sendet

DDRAM



## 4-Bit Modus (4 statt 8 Datenleitungen)

D4-D7 statt D0-D7

Sende **erst höherwertige** Nibble, dann das niedrigwertige Nibble => 2 statt 1 Schreibzyklus

Zum Aktivieren des 4 Bit Modus muss eine spezielle Initialisierungssequenz durchlaufen werden.

0x18 in DDRAM: erst 0001, dann 1000

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile)

0x00 – 0x27 (1. Zeile), 0x40 – 0x

# 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:

TDI

TCK

TMS

TDO

- 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:

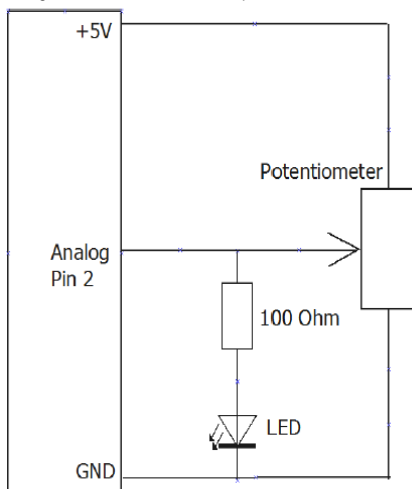
- sei() (Set Enable Interrupt):
  - o Interrupts global aktivieren
- SREG:
  - o 1 Bit hier setzen statt sei() möglich
- EIMSK:
  - o De/aktivieren von speziellen Interrupts
- EIFR:
  - o Interrupt Flags
- EICRA:
  - o Steigende/fallende Flanke?
- EICRB:

## 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



```

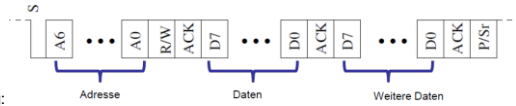
void setup() {
  Serial.begin(9600);
  // enable ADC functionality
  ADCSRA |= (1 << ADEN);
  // use /128 prescaler, see manual p271
  ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
  // select autotrigger, to use free-running mode
  ADCSRA |= (1 << ADFR);
  // use free-running mode
  ADCSRA |= (1 << ADFR);
  // read analog value, first LOW then HIGH register
  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: }



- I2C Datenübertragung:

- Bestandteile I2C Kompatible ICs

```

// transition table
state_t ('state_table[3][4]) (void) = {
  //
  /*WAIT_FOR_CLAP_ONE*/ (state_t, MICRO_PIN_HIGH, TIMER100_EXPIRED, TIMER300_EXPIRED)
  /*ONE_CLAP_DETECTED*/ (state_t, enterOneClapDetected, stay, stay),
  /*WAIT_FOR_CLAP_TWO*/ (state_t, enterWaitForClapTwo, stay, enterWaitForClapOneTimer300)
};

void setup() {
  pinMode(microPin, INPUT);
  pinMode(ledPin, OUTPUT);
  state = WAIT_FOR_CLAP_ONE; // initial state
  Serial.begin(9600);
}

void loop() {
  // detect events
  event_t event = NO_EVENT;
  if (digitalRead(microPin) == HIGH) {
    event = MICRO_PIN_HIGH;
  }
  else if (timer100 <= millis() - timer100 > 100) { // timer valid and 100 ms expired
    event = TIMER100_EXPIRED;
    timer100 = 0; // reset timer;
  }
  else if (timer300 <= millis() - timer300 > 300) {
    event = TIMER300_EXPIRED;
    timer300 = 0;
  }

  // use transition table to switch state
  state_table[state][event]();
}

```