|  |  |  |  |
| --- | --- | --- | --- |
| Speicher, Digitale Ein- und Ausgabe DDRx (Data Direction Register):  Entsprechendes Bit auf 1 für Ausgang, oder 0 für Eingang  PORTx (Port Register):  Wenn Pin auf Ausgang, dann 1 = 5V und 0 = 0V  PINx (Port Input Register):  Wenn Pin auf Eingang, dann 1 = HIGH liegt an und 0 = LOW liegt an  pinMode(13, OUTPUT);  digitalWrite(13, HIGH);  digitalRead(13); … if (digitalRead(13) == HIGH)…  Serial.begin(9600);  Serial.println("Eingabe ist: " + Serial.readString());  Serial.available() :  Anz. Bytes, die zum Lesen verfügbar sind. Evtl. in while Schleife. |  | Nichtflüchtige Speicher:  ROM  OTPROM  EEPROM: Begrenzte Anzahl an Schreib/Lesezyklen.  Konfigurationsdaten, Kalibrierungsdaten  Flash: Programm- Daten/Code  SRAM(flüchtig): Arbeitsspeicher, Register, Stack usw  *Einzelne Bits setzen:* x |= (1 << Bitnummer);  *Einzelne Bits löschen:*  x &= ~((1 << Bitnummer) | (1 << Bitnummer2));  *Testen ob Bit auf 1:*  If (DDRC & (1 << Bitnummer)) { }  *Testen ob Bit auf 0:*  If ( ! (DDRC & (1 << Bitnummer))) { }  *Alle Bits umdrehen:*  x = 0xFF ^ x  *LED togglen:*  PINA |= (1 << PINA2);  Vorwiderstand berechnen: | **Pull-Up** / Active Low **Pull-Down** / Active High  Bei offenem Taster wird Bei offenem Taster wird  Spannung am Pin auf Spannung am Eingang  HIGH gezogen. Auf LOW gezogen.  Entprellung  Einmaliges betätigen eines Schalters führt evtl zu mechanischen Vibrationen.  SW-Lsg: Künstliche Wartezeit nach Zustandswechsel – Bis Schalter eingeschwungen. |
| **Mikrocontroller Bestandteile**: MicroProzessor, Timer, Schnittstellen, Speicher, AD-Wandler  **Entwicklerboard**: Arduino Mega,  **Mikroprozessor**: Atmega2560  **Cross-Compilation**: Programm wird nicht auf Zielplattform (Mikrocontroller), sondern auf anderer Plattform übersetzt  **Flashen**: hex-File von PC an Entwicklerboard senden  **Harvard-Architektur**: Daten- und Instruktionsspeicher getrennt  Instruktionsspeicher: Nicht flüchtiger Flash Speicher  Daten: in flüchtigem SRAM  SRAM und DRAM sind flüchtig. Rest nicht flüchtig. |
| Interrupts sei(): Interrupts global aktivieren (oder: SREG |= 128)  cli(): Interrupts global deaktivieren  *SREG* (AVR Status Register):  Bit 7 auf 1 = sei();  *EIMSK* (External Interrupt Mask Register):  Speziellen Interrupt de-/aktivieren  *EICRA* (External Interrupt Control Register A)  *EICRB* (External Interrupt Control Register B):  ISCn0 und ISCn1. Falling oder Rising Edge.  n ist die Interrupt Nummer.  *EIFR* (External Interrupt Flag Register):  Wenn Interrupt ausgelöst: Bit ist 1  *ISR* (INT0\_vect) { }  attachInterrupt(digitalPinToInterrupt(21), count, RISING); | //Entprellung | **Busy Waiting:** while (DDRC & (1 << DDC3));  **Polling**: periodisches Abfragen, ob Ereignis eingetreten  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.  **Trap**: Art von Interrupt, die aber synchron und reproduzierbar ist. z.B. System Call, Div durch 0 …  Interrupt Request:  Interruptereignis – [InterruptController] – über IRQ Eingang Unterbrechungsanforderung an CPU – CPU unterbricht Programm und startet Unterbrechungsroutine  Interrupt Vector Table:  Welches Interruptereignis gehört zu welcher ISR? Jede Vectornummer hat eine zugehörige Programmadresse.  ISR ist selbst nicht unterbrechbar (I Bit SREG) | Externe Interrupts  Controller tastet zu Beginn jedes Taktzyklus ab. Falls Interrupt aktiviert, Aufruf der ISR.  Probleme: Leichte Verzögerung, „Prellung“  Interne Interrupts  Timer, A/D-Wandler  Bei Auslauf eines Timers unterbricht HW Ausführung der normalen Software  Volatile  Variable wird vor jedem Lesen aus SRAM gelesen und nach jedem Schreiben in SRAM geschrieben  !!Globale Variablen die in ISR vorkommen **immer volatile**!!! |
| Timer **n**: Timer 1-5  *TCCRnA* (Timer/Counter n Control Register A):  PWM  *TCCRnB* (Timer/Counter n Control Register B):  Prescaler; Starten des Timers; Input Capture, CTC  Beide TCCRn erst auf 0x00 setzen.  Auch wenn keinen Prescaler will, muss man setzen  *TCNTn* (Timer Counter n, 16 Bit):  Aktueller Zählerstand. Anfangs auf 0 setzen.  *OCRnA, OCRnB, OCRnC* (Output Compare Register, 16 Bit):  Wert gegen den Zählerstand verglichen werden kann  *ICRn* (Input Capture Register):  Bei Input Capture erfasster Wert wird gespeichert  *TIMSKn*:  Aktivieren/Deaktivieren der Timer Interrupts  *TIFRn*:  Timer bezogene Interrupt Flags  **CTC Beispiel**: TCCR4B |= (1 << WGM42);  ISR(TIMER4\_OVF\_vect) {}:  Interrupt bei Timer 4 Overflow  ISR(TIMER4\_COMPA\_vect) {}: Timer 4 compare A |  | Atmega2560 Systemtakt = 16Mhz  16Bit Timer => Timer läuft nach über  Prescaler  Fallende Flanke des Prescalers an Bit Qn triggert Counter.  Vorteil großer Prescaler: Messen langer Zeiten möglich.  kleinstes messbares Zeitintervall ohne Prescaler:  mit f/1024 Prescaler: (1Tick)  Nachteil: Schlechtere Auflösung.   * immer kleinstmöglichen Prescaler!   Welchen Prescaler für 3s Intervalle mit 16 Bit Timer? Takt: 1 MHz (Bei 16MHz bis 16\*3\*1000000) | Input Capture  Bei externen oder internen Signalen/Ereignissen wird aktueller Zählerstand in ICRn gespeichert u. Flag ges    Output Compare  Bei erreichen eines konfigurierten Zählerstandes wird Interrupt ausgelöst, oder best. Signal erzeugt.  hier mit CTC  CTC Mode (Clear Timer on Compare Match)  TOP Wert in OCRnA oder ICRn konfiguriert.  Zähler bei erreichen des Zählerstandes automatisch auf 0. |
| Pulsweitenmodulation **n**: Nummer des Timers  TCCRnA  Compare Output Mode  Fast PWM usw  TCCRnB  Fast PWM; Prescaler  OCnA, OCnB, OCnC (Output Compare Pins):  PWM-Ausgang  Inverting oder non-Inverting Mode  Output Compare Pins müssen **als Ausgang im DDR** Register konfiguriert sein!  OCRnX (Output Compare Register):  Vergleichswert (Schwellwert) muss gesetzt werden, der jeweils PWM-Ausgang OCnX beeinflusst. |  | Signal mit konstanter Periode, aber variabler Pulsdauer wird erzeugt.  **Duty Cycle**: t/T (= Pulsdauer / Periodendauer)  Nur Duty Cycle wird ausgewertet,  nicht Periodendauer.  TOP: ICRn Register (oder andere  siehe S145 Tabelle)  CMP: OCRnX Register    In manchen Modi kann man TOP/CMP nur ändern, wenn Zähler gerade auf BOTTOM/TOP ist. („*Update of OCRnX at...“* in Tabelle) | Inverting u. Non-Inverting Mode  Non-Inverting: siehe Links.  Inverting: PWM Ausgang genau andersrum  TCCRnX Register  Up-Down-Counter  doppelte Periodendauer, geringere Auflösung  BOTTOM u. TOP immer genau in der Mitte    Tabelle S145 TCCRnA u TCCRnB Fast PWM(Up-Counter), PWM(Up-Down Counter) |
| Analoge Ein-/Ausgabe ADMUX  Referenzspannung wählen  Analoge Eingangspins für A/D Umsetzung wählen  ADCSRA  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 ADSC triggern)  **Prescaler** (ADPS!! 50-200kHz bei int. AD-Wandler)  Interrupts  ADCSRB  Analoge Eingangspins für A/D Umsetzung wählen  Single Ended oder Differential Conversion  Free Running Mode oder manuelles Triggern  ADCL, ADCH  Speichert Ergebnis der A/D Umsetzung  Erst ADCL, dann ADCH lesen (atomarer Zugriff)  ADCL + 256 \* ADCH |  | A/D Wandlung  **Auflösung**:  Wie viel  Spannungsunterschied  pro Stufe?  Vref / 2r  **Repräsentant**  eines Intervalls  liegt in Intervallmitte um  Quantisierungsfehler zu vermeiden.  Erstes Intervall: Repr. Von 000 Stufenbreite ½ LSB  Letztes Intervall: Stufenbreite 1 ½ LSB  r = 10 bei ATmega  Fehlerquellen  Quantisierungsrauschen  Umsetzungszeit  Änderung des Eingangs während der Umsetzung  **Ersatzwiederstand** von zwei parallelen Widerständen | Ansätze zur A/D Wandlung  Komparator  Parallelverfahren, Zählverfahren, **Wägeverfahren**:  r=3, Vin=(1011)V -> 1000 – 1100 – 1010 – 1011  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  **Trigger**:  Manuelles Auslösen: durch Codeanweisung  Free Running Mode: Endlosschleife  Auto Trigger: angestoßen durch Timeroverflow, Komparatorausgang etc  **Erkennen, dass A/D Umsetzung beendet wurde:**  Auswerten eines speziellen Flags, oder durch speziellen Interrupt  **Wertebereich:**  Single-Ended Conversion: [0,Vref]  Differential Conversion: [-Vref/2, Vref/2] |

|  |  |  |  |
| --- | --- | --- | --- |
| Watchdog, 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 |  | 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 augweckender 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 | 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 Ausgansspannung: 100 mV  Max Ausgangsspannung: 1750 mV  Max Ausgangsspannung sollte möglichst knapp unter Referenzspannung liegen.  **Binäre Zahl** (bei Vref = 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°C = m\*40 + t [°C] 125°C = m\*700 + t [°C] |
| 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 erfolgr 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 |  | 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-toPeer** 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 | I2C | | Seriell | Ja | Ja | Ja | | Duplex | Ja | Ja | Nein | | Synchron | Nein  Baud konfig | Ja | Ja | | Bus | Nein | Jein | Ja | | Anz.Leitungen | 3 | 5 | 3 | | Differential |  | Nein | Nein | | Datenrate  ATmega2560 | BAUD = fosc / (16(UBRRn+1)) | fosc/128 – fosc/2 | Max 400 kbit/s | | UART (oder SCI) (Arduino hat 4 USARTs)  2 Datenleitungen: TxD und RxD  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, : Slave Select (aktiver Slave)  I2C, 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/: Master spezifiziert, ob Lese oder Schriebzugriff -> Slave: ACK -> Datentransfer  *Stoppbedingung*: (Steigende Fl. SDA)+(SCL==HIGH) |
| Peripherie |  | Mikrocontroller -> ASCII an DDRAM -> Steuereinheit schlägt Muster im CGRAM nach u. blendet es auf Display ein.  CGRAM (definiert Aussehen von Schriftzeichen)  ROM: Standardzeichen, a-z, A-Z, ...  RAM: Benutzerdefinierte Zeichen    DDRAM (welche Zeichen zeigt Display aktuell?)  Speicheradresse:  1 Zeilenmodus: 0x00 – 0x4F  2 Zeilenmodus:  0x00 – 0x27 (1. Zeile), 0x40 – 0x68 (2. Zeile) | Cursor  Display knn 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 Didplay sendet    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 |
| SW-Download / Debugging Jeder Atmel hat eindeutigen Signaturcode. Wird bei ISP überprüft.  Aktuell geladenes Programm kann aus Flash des uC’s als HEX-Datei auf PC geladen werden.  Bootloader reparieren: Bootloader (Hex-File) ins Flash laden. | JTAG (Boundary Scan, Debugging, In-System Programming)  Erreichbarkeit aller virtuellen Testpunkte über eine einzige, einheitliche Schnittstelle prüfen.  JTAG Chain: alle DR-Register werden in Chain eingebunden | Software Download (Flashing)  Standardformat: Intel Hex-Format.  Nicht nur Sequenz von Opcodes, sondern auch Checksumme, Info über Programmgröße und Zielspeicherort  In-System Programming  Mikrocontroller direkt im Einsatzsystem programmieren.  **Erste Möglichkeit**: Programmierung mit seriellen Schnittstellen: SPI, oder JTAG  Benötigt Zusatzhardware, die zuvor erstelltes Programm/Daten in internen nichtflüchtigen Speicher (EEPROM, Flash) schreibt.  uC erkennt Programmierung durch spezielle Signalfolgen, Timing usw. **zweite Möglichkeit**:  Bootloader  Programm für das laden von Programmen + USB Kommunikation  Lauscht nach Reset, ob neues Programm über USB hochgeladen werden soll.  Falls nein: Bereits vorhandenes Programm wird gestartet.  Separater Speicherbereich für Bootloader | 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 |  | Ereignisse  Ereignisse in der Praxis asynchron, da der Zustandsautomat nie blockieren darf.  delay kann dazu führen, dass andere Ereignisse ignoriert werden.  2 Ansätze:  Event-Driven: Ereignisse werden vorwiegend über HW-Interrupts erkannt. Zustandsübergang in ISR. Problem: Gleichzeitige Interrupts (priorisieren?)  Polling: Prüfe in jedem Schleifendurchlauf, ob neues Ereignis vorliegt. -> Zustandübergang  Arduino als ISP  Programm auf Arduino laden, damit er als ISP fungiert.  Bootloader brennen  Programm auf anderen uC laden |

Register

**Digital IO:**

* DDRx (Data Direction Register):
  + Entsprechendes Bit auf 1 für Ausgang, oder 0 für Eingang
* PORTx (Port Register):
  + Wenn Pin auf Ausgang, dann 1 = 5V und 0 = 0V
* PINx (Port Input Register):
  + 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):
  + Prescaler
  + Starten des Timers
  + Input Capture
* TCNTn (Timer Counter n, 16 Bit):
  + Aktueller Zählerstand
* OCRnA (Output Compare Register A, 16 Bit):
  + Wert gegen den Zählerstand verglichen werden kann
* OCRnB (Output Compare Register B, 16 Bit):
  + Wert gegen den Zählerstand verglichen werden kann
* ICRn (Input Capture Register):
  + Bei Input Capture erfasster Wert wird gespeichert
* TIMSKn:
  + Aktivieren/Deaktivieren der Timer Interrupts
* TIFRn:
  + Timer bezogene Interrupt Flags

**Pulsweitenmodulation:**

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

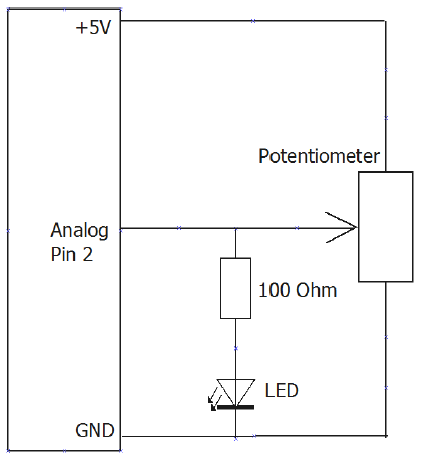
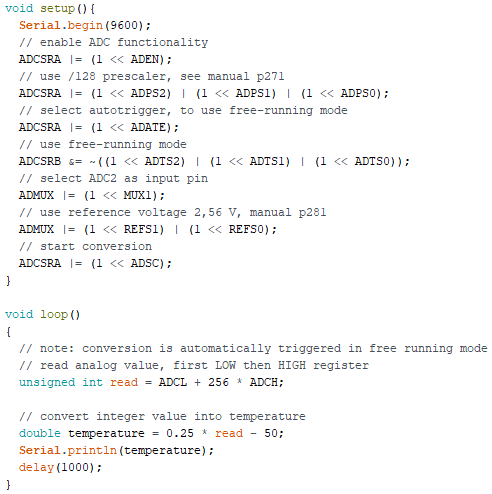
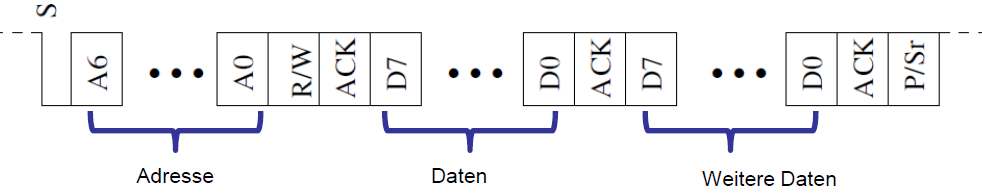
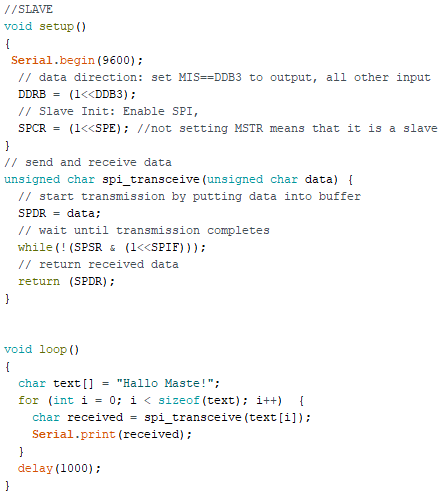
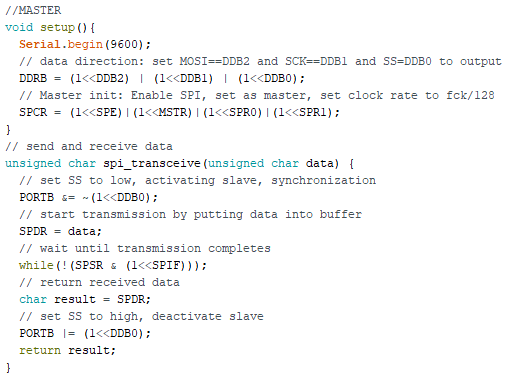
**Interrupts:**

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

**Analoge IO:**

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

TODO:

* Übung 7 vielleicht noch mehr Beispiele
* 
* Sensor Beispiel: 
* I^2C Datenübertragung: 
* Übung Master Slave: SLAVE: 
* Übung Master Slave: MASTER: 
* Bestandteile JTAG Kompatible ICs