www.informatics4kids.de

Informatik in der Oberstufe

Messen, Steuern und Regeln
mit Linux, Java und dem IO-Warrior24

BAND IV

SPI-Projekte

Torsten Röhl

Informatik in der Oberstufe

Messen, Steuern und Regeln mit Linux, Java und dem IO-Warrior24
Band IV: SPI-Projekte

v.1.0 - Dezember 2013

Openbook

Vorwort

Dies ist der vierte Band der Reihe
INFORMATIK IN DER OBERSTUFE
Dieser Band setzt den Inhalt des ersten Bandes (Grundlagen) voraus. Behandelt werden Projekte, die die Verwendung der von Motorola entwickelten und weit verbreiteten **SPI-Schnittstelle** gemeinsam haben.
Alle Projekte verwenden das Gandalf-Board (Bauanleitung: www.informatics4kids.de) und ein Steckbrett.
Viel Spaß beim Experimentieren!

MSR MIT LINUX, JAVA UND DEM IO-WARRIOR24

BAND I	Grundlagen
BAND II	IO-Projekte
BAND III	IIC-Projekte
BAND IV	SPI-Projekte ✓

HINWEIS

Der Autor übernimmt keine Haftung, falls durch falsche Programmierung oder Handhabung der elektronischen Bauteile Schäden entstehen.

Inhaltsverzeichnis

Teil I

Theorie

1

Die SPI-Schnittstelle

SPI steht für **S**erial **P**eripheral **I**nterface[1] und stellt einen Standard für einen *synchronen seriellen Datenbus* dar. Seriell heißt, die Daten (Bits) werden einzeln nacheinander übertragen. Synchron bezieht sich darauf, dass ein Baustein ein Taktsignal vorgibt, nach dem sich die gesamte Kommunikation richtet. Den Baustein, der das Taktsignal vorgibt, nennt man Master, den anderen Baustein, mit dem der Datenaustausch erfolgt, bezeichnet man als Slave. Die SPI-Schnittstelle ist lizenzfrei, ein wesentlicher Grund für ihre weite Verbreitung. Es existieren hunderte von IC für die verschiedensten Zwecke. Eine kurze Übersicht bietet die nachfolgende Tabelle. Die SPI-Schnittstelle ist eine schnelle Schnittstelle, d. h. aber auch, dass die Verbindung zwischen MASTER und SLAVE kurz ($<$ 1 m) sein muss. Verringert man die Taktrate, kann der Abstand zwischen MASTER und SLAVE auf Kosten der Geschwindigkeit aber auch größer sein.

MERKE

Das SPI-Protokoll schreibt vor, dass immer nur ein MASTER existieren kann. Die Rolle des MASTERS übernimmt der IO-Warrior. Damit sind alle Bauteile, die wir an den Mikrocontroller anschließen, automatisch SLAVES.

Obwohl ein Master durchaus mit mehreren Slaves kommunizieren kann, beschreiben wir im Folgenden lediglich die Kommunikation mit jeweils einem MASTER und einem SLAVE.

1.1 Aufbau und Funktion der SPI-Schnittstelle

Die SPI-Schnittstelle benötigt vier Leitungen. Die Kommunikation wird immer vom Master aus initiiert.

[1] Manchmal auch als Microwire bezeichnet. Obwohl dies nicht ganz richtig ist, lassen sich in der Regel alle Bausteine mit Microwire-Schnittstelle auch mit SPI ansteuern.

MASTER	→ Peripherietypen, *mögliche Slaves*
IO-Warrior24	Wandler (ADC, DAC)
	Speicherbausteine (FLASH, EEPROM)
	Uhrenbausteine (Real Time Clocks)
	Sensoren (Druck, Temperatur)
	Potentiometer, LCD-Controller u.s.w. ...

Tabelle 1.1. IC's mit SPI-Schnittstelle. Es existieren eine Vielzahl von möglichen SPI-Bauteilen, die am häufigsten verwendeten Typen listet die Tabelle auf.

Abbildung 1.1. Vier Leitungen werden für die Kommunikation benötigt. Hiervon sind zwei Leitungen für den eigentlichen Datenaustausch notwendig. Eine Leitung (SCLK) gibt ein Taktsignal vor und eine andere (SS) ist für die Bausteinauswahl zuständig.

- **SCLK** (Serial Clock): Sie gibt das Taktsignal vor, nach dem sich der Slave richten muss.
- **MOSI** steht für Master-Out Slave-In. Es entspricht dem Datenausgang, d. h., es sind Daten, die vom Master zum Slave gesendet werden.
- **MISO** steht für Master-In Slave-Out. Es entspricht dem Dateneingang, d. h., es sind Daten, die vom Slave zum Master gesendet werden. MISO und MOSI sind nicht unabhängig voneinander. Für jedes Bit, das geschrieben wird, muss auch ein Bit gelesen werden.
- **SS** (Slave Select auch CS für Chip Select): Diese Leitung ist für die Auswahl des angeschlossenen Bausteins nötig. Sie wird bedeutender, wenn mehrere Bausteine am Master angesteuert werden sollen.

Das Taktsignal

In der Abbildung 1.2 ist ein Taktsignal, wie es der Master vorgibt, gezeigt. Dieses Signal ist allerdings nicht eindeutig, denn es ist z. B. offen, ob ein Bit geschrieben werden soll, wenn das Signal eine hohe Polarität besitzt oder eine geringe (LOW) besitzt. Außerdem müssen sich MASTER und SLAVE auch darüber einig sein, ob der Datenaustausch bei steigender oder fallender Flanke erfolgen soll. Da es insgesamt jeweils zwei Möglichkeiten für die Polarität und Phase gibt, sind vier verschiedene Kombinationen, die man als Modi bezeichnet, bekannt. Welchem Modus ein Bau-

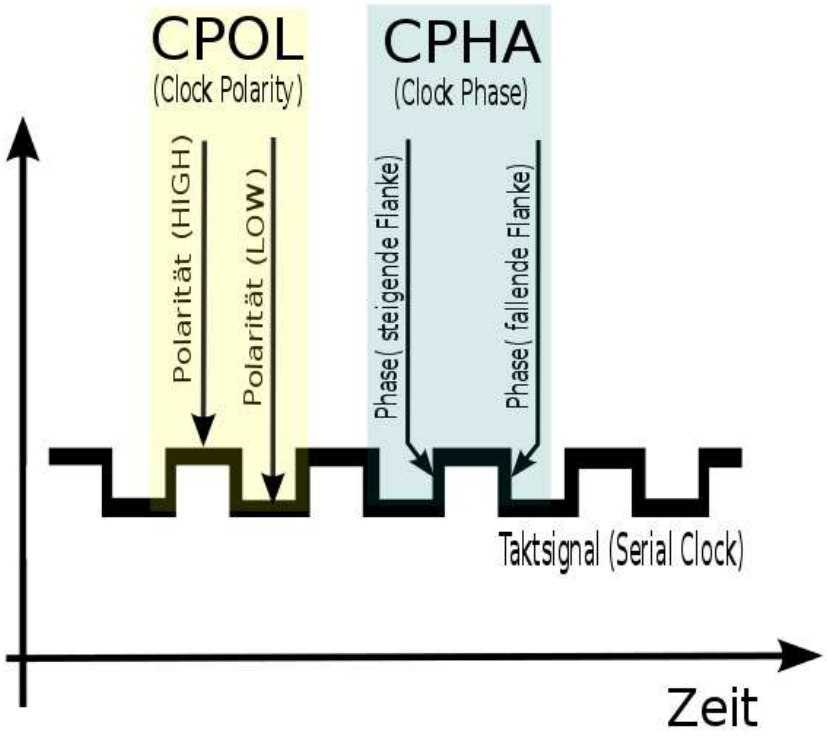


Abbildung 1.2. Der Master gibt das Taktsignal vor. Da das Signal aber nicht eindeutig ist, muss man erst dem Datenblatt des jeweils angeschlossenen Bausteins (Slave) entnehmen, welche Werte für Polarität (CPOS) und Phase (CPHA) zu wählen sind. Die höchste Frequenz ("Geschwindigkeit"), mit der der Master den Slave ansteuern kann, kann man dem Datenblatt entnehmen.

SPI-Modus	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Tabelle 1.2.

stein zur Verfügung steht, muss zusammen mit der Übertragungsrate dem Datenblatt des jeweiligen Bausteins entnommen werden. Der Master kann im Allgemeinen den Slave auch mit geringer Übertragungsrate ansteuern. Ist die vom Master vorgegebene Taktrate allerdings zu hoch, dann ist die Übertragung gestört. Der nächste Abschnitt beschreibt, wie man sich prinzipiell die Datenübertragung vorstellen kann.

Datenübertragung

Die Datenübertragung verläuft seriell, d. h., es wird jeweils ein Bit nach dem anderen übertragen. Typischerweise werden dabei jeweils 8 Bit (1 Byte) pro Schreib-/Lesezyklus übertragen. Falls es Abweichungen gibt, z. B., wenn 12 Bit pro Schreib-/Lesezyklus übertragen werden, so muss man diese Informationen dem jeweiligen Datenblatt des angeschlossenen Bausteins entnehmen. In der Grafik (Abbildung 1.3) wird der Datenaustausch zwischen Master und Slave anhand zweier Tische beschrieben, auf denen Bücher stehen. Der linke Tisch (orange) ist der “Mastertisch” und der rechte Tisch (gelb) der “Slavetisch”. Jedes Buch soll ein Bit repräsentieren. Übersichthalber wird die Übertragung nur anhand von zwei Bits demonstriert, auch wenn es theoretisch acht Bücher (1 Byte) sein müssten. Dieses Übertragungsprinzip nennt man Schieberegister.
Diese Übertragung erfordert damit zwangsweise, dass Daten immer nur gemeinsam geschrieben und gelesen werden können. Falls man also Bausteine am Master betreibt, von denen nur gelesen werden soll, müssen trotzdem Daten (Dummy-Daten) geschrieben werden. Verwendet man umgekehrt Bausteine, auf denen nur geschrieben werden soll, müssen trotzdem immer auch die anfallenden Bits gelesen werden, auch wenn man sie anschließend wieder verwerfen kann.

Zusammenfassung

Für eine erfolgreiche Datenübertragung mittels SPI müssen demnach folgende Dinge des anzusteuernden Bausteins in Erfahrung gebracht werden.

- Der SPI-Modus, d. h, die Werte für Polarität (CPOL) und Phase (CPHA) , die der Baustein unterstützt.
- Die Übertragungsgeschwindigkeit, d. h., die vom Slave unterstützten Taktfrequenzen.
- Die Anzahl der Bits die pro Schreib-/Lesezyklus übertragen werden.

Anschließend müssen z. B. aus den gelesenen Bits die erforderlichen Informationen extrahiert werden, d. h., man muss die Bedeutung jedes einzeln übertragenden Bits kennen, um die Daten sinnvoll auswerten zu können. Wie man die SPI-Schnittstelle mit dem IO-Warrior24 verwendet, wird im nächsten Abschnitt beschrieben.

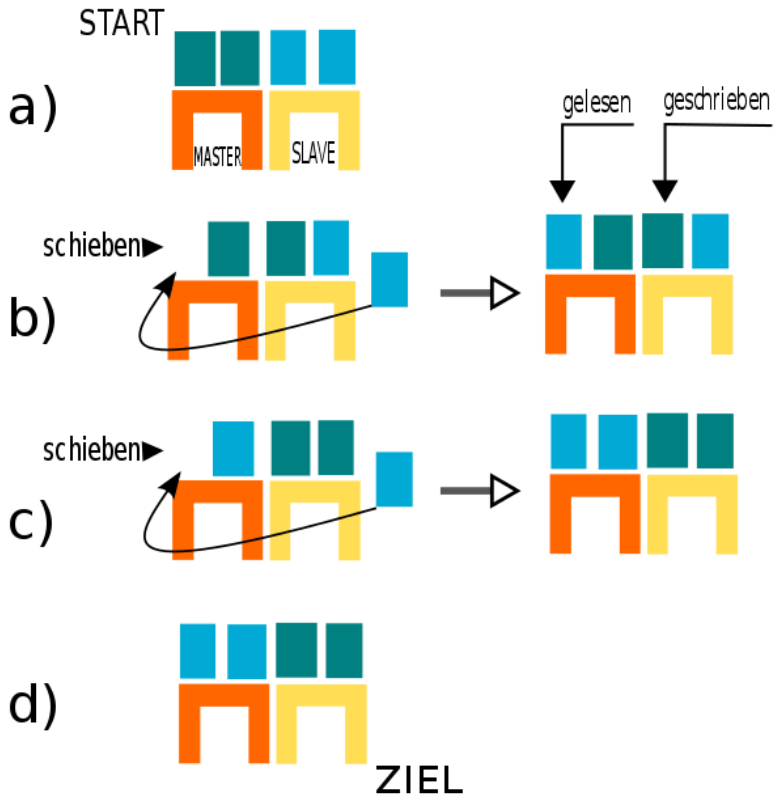


Abbildung 1.3. a) In der Ausgangskonfiguration sollen zwei Bücher (Bits) vom “Mastertisch” (Master) zum “Slavetisch” (Slave) geschoben (geschrieben) werden. Dabei sollen gleichzeitig zwei Bücher (Bits) gelesen werden. **b)** Schieben wir die Bücher (Schieberegister) nach rechts, dann gelangt ein Buch vom “Mastertisch” auf den “Slavetisch”. Das heruntergefallene Buch wird vorne an die frei gewordene Stelle des “Mastertisches“ gestellt. Auf diese Weise wird immer für jedes geschriebene Bit auch eines vom Slave gelesen. **c)** Der Vorgang wiederholt sich. Durch erneutes Schieben gelangt ein weiteres Buch vom Mastertisch zum Slavetisch. Anschließend wird das heruntergefallene Buch wieder vorne an die frei gewordene Stelle gestellt. **d)** Da wir hier nur zwei Bücher (Bits) haben, ist jetzt ein Schreib-/Lesezyklus beendet. Es wurden dabei 2 Bits vom Master und den Slave geschrieben und gleichzeitig 2 Bits (Bücher) vom Slave gelesen. Vom Master aus betrachtet, befinden sich jetzt an den Plätzen (Speicher-Register), wo sich anfangs die zu schreibenden Bits (Bücher) befanden, die gelesenen Bits (Bücher).

1.2 Verwenden der SPI-Schnittstelle mit dem IO-Warrior24

Um die SPI-Schnittstelle am IO-Warrior zu nutzen, muss zuerst SPI aktiviert werden. Insgesamt stehen zwei SPI-Spezial-Funktionen zur Verfügung. Ein Report mit

ReportID	Bedeutung
0x08	SPI aktivieren oder deaktivieren
0x09	Datentransfer (vorbereiten, lesen/schreiben)

Tabelle 1.3. SPI Spezial-Funktionen

der ID 0x08 aktiviert (bzw. deaktiviert) den SPI-Modus. Reports mit der ID 0x09 werden für die eigentliche Datenübertragung verwendet. Wenn der SPI-Modus aktiviert ist, dann übernehmen folgende IO-Ports, SPI Funktionen und stehen damit (bis zur Deaktivierung) nicht mehr für IO-Operationen zur Verfügung.

Pin-Port.Pin	SPI
Pin-0.3	DRDY (DataReady - Handshake nur mit ReportID 0x09)
Pin-0.4	SS
Pin-0.5	MOSI
Pin-0.6	MISO
Pin-0.7	SCK

Tabelle 1.4. SPI Spezial-Funktionen

Aktivierung (Deaktivierung) und Konfiguration

Ein Report mit der ID=0x08 steuert über das 2. Byte außerdem die Taktrate, Phase und Polarität.
Um den Wert für das 2. Byte (mode) zu bestimmen, müssen die ersten 4 Bits des

ReportID	1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6.Byte	7. Byte	
0x08	1	mode	0	0	0	0	0	SPI aktivieren
0x08	0	mode	0	0	0	0	0	SPI deaktivieren

Tabelle 1.5. Die Konfiguration der SPI-Schnittstelle erfolgt über das 2. Byte. Das 1. Byte aktiviert bzw. deaktiviert die Schnittstelle.

zweiten Bytes gesetzt werden. Dabei bestimmen Bit 0 und Bit 1 die Taktrate (Geschwindigkeit der Datenübertragung) während Bit 2 die Phase und Bit 3 die Polarität festlegen. Es gilt:

	0. Bit	1. Bit	2. Bit	3. Bit	4. Bit	5. Bit	6.Bit	7. Bit	Taktrate
mode =	0	0	CPHA	CPOL	0	0	0	0	2MBit/sec
mode =	0	1	CPHA	CPOL	0	0	0	0	1MBit/sec
mode =	1	0	CPHA	CPOL	0	0	0	0	0.5MBit/sec
mode =	1	1	CPHA	CPOL	0	0	0	0	0.0625MBit/sec

Tabelle 1.6. Die Konfiguration der SPI-Schnittstelle erfolgt über das 2. Byte. Es stehen vier verschiedene Taktfrequenzen (Taktraten) zur Verfügung. Unabhängig davon kann CPHA und CPOL gewählt werden.

Datentransfer

Der Datentransfer wird über einen Report mit der ID `0x09` gestartet. Das erste Byte beinhaltet dabei die Anzahl der zu sendenden Bytes. Die Bytes 2-7 enthalten dann, die an den Baustein (Slave) zu schreibenden Daten. Nach der Date-

ReportID	1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte	7. Byte
`0x09`	flags	data	data	data	data	data	data

Tabelle 1.7. Nach der Aktivierung der Schnittstelle wird mit einem Report (ID=0x09) der Datentransfer eingeleitet. Die zu sendenen Daten liegen an Byte 2 bis Byte 7 an. Das 1. Byte flags muss hierfür konfiguriert werden.

	0. Bit	1. Bit	2. Bit	3. Bit	4. Bit	5. Bit	6. Bit	7. Bit
flags =	Anzahl	Anzahl	Anzahl	0	0	0	0	0

Tabelle 1.8. Wenn die Datenübertragung nicht mehr als 6 Bytes umfasst und keine Handshake-Kommandos benötigt werden, bestimmen Bit 0 - bis Bit 3. Die Anzahl der zu sendenden Daten. Falls ein einzelner Transfer mehr als 6 Byte umfasst und Handshakes benötigt werden kann man genaueres dem IO-Warror Datenblatt entnehmen.

ninitialisierung können (müssen) die Daten gelesen werden. Hierfür wird ebenfalls die ReportID=`0x09` verwendet. Diese Darstellung geht davon aus, dass ein Datentransfer nicht mehr als 6. Bytes umfasst und dass außerdem keine Handshake-Kommandos benötigt werden. (Näheres siehe IO-Warrior-Datenblatt, Abschnitt: SPI Special mode function).

Zusammenfassung

Eine SPI Kommunikation verläuft mit dem IO-Warrior24 nach dem folgenden Schema:

ReportID	1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6.Byte	7. Byte
0x09	Anzahl	Daten	Daten	Daten	Daten	Daten	Daten

Tabelle 1.9. Daten werden vom Slave durch einen Report (ID=0x09) gelesen. Dabei enthält das 2. Byte die Anzahl der gültigen Bytes. Es können maximal 6 Bytes gelesen werden. Diese Daten sind im Byte 2 bis Byte 7 gespeichert.

1. Ein Report mit der ID=0x08, der u.a. eine 1 im ersten Byte stehen hat, wird zur Aktivierung der SPI-Schnittstelle gesendet.
2. Ein Report mit der ID=0x09 wird zur Initialisierung und zum Schreiben gesendet.
3. Die zu lesenden Daten werden mit der ReportID=0x09 empfangen. Es können pro Datentransfer maximal 6 Bytes geschrieben/gelesen werden. Die Anzahl der zu lesenden Daten befindet sich im 1. Byte. Die eigentlichen Daten in Byte 2 bis Byte 7.
4. Anschließend wird ein Report mit der ID=0x08, der eine 0 im ersten Byte enthält, zur Deaktivierung der SPI-Schnittstelle versendet.

Teil II

SPI-Projekte

2

MPC3008 - Ansteuern eines 10-Bit Analog-Digital-Wandlers

Der Baustein MCP3008 ist ein 10-Bit-AD-Wandler mit SPI-Schnittstelle. Er besitzt 8 Messkanäle, die entweder einzeln (single-ended) oder in vier Paaren (differential input pairs) aufgeteilt werden können. Die Spannungsversorgung des Baustein übernimmt der IO-Warrior (also das Gandalf-Board). Es werden keine weiteren Bauteile benötigt, um den Baustein in Betrieb zu nehmen.

2.1 Hardware-Steckbrief

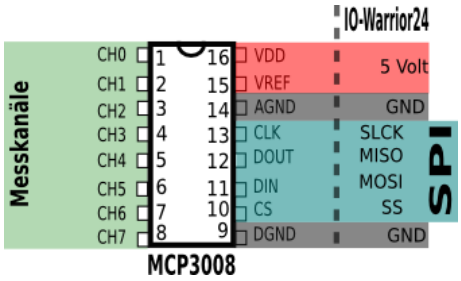


Abbildung 2.1. Der MCP3008 ist ein 10-Bit 8Kanal AD-Wandler. Alle Messkanäle befinden sich auf einer Seite (Pin1-Pin8) des Baustein.

- VDD (Spannungsversorgung: hier 5 Volt) Die Versorgungsspannung muss im Bereich +2,7 V bis +5,5 V liegen.
- VREF (Referenzspannung: wir benutzen die 5 Volt-Versorgungsspannung). Die Referenzspannung muss zwischen 0,25 V und VDD liegen.
- AGND (Analoge Masse)
- SCLK (SPI-Taktsignal)
- DOUT (Data-Out vom MCP3008 aus betrachtet)

- DIN (Data-In vom MCP3008 aus betrachtet)
- SS (CS, Chip Selekt, Standard Aktiv-LOW)
- DGND (Digitale Masse)
- CH0-CH7 Die 8 Eingangskanäle messen im Bereich AGND-VREF, d. h., in unserem Fall können Spannungen von 0-5 Volt gelesen werden. Falls größere Spannungen verarbeitet werden sollen, müssen diese zuvor auf diesen Messbereich abgebildet werden, da ansonsten der Baustein beschädigt wird.

Auflösung

Ein 10-Bit AD-Wandler zerlegt den Messbereich (hier 0-5 Volt) in $2^{10} = 1024$ Abschnitte. In unserem Fall beträgt die Auflösung $5000/1024 = 4,88$ mV. Je genauer die Auflösung ist, desto genauer kann die zu messende Größe digital abgebildet werden.

$$\text{Auflösung} = \frac{V_{REF}}{1024}$$

Würde man beispielsweise den Messbereich auf (0-2,5V) einschränken, indem man an VREF=2.5 Volt anlegt, dann könnte der Wandler bereits $2500/1024 = 2,44$ mV auflösen.

AUFGABE

Der AD-Wandler gibt einen Wert von 322 aus ($V_{REF} = 5V$). Welche Spannung wurde eingelesen?

LÖSUNG

Die eingelesene Spannung ergibt sich aus dem digital gewandelten Wert und der benutzten Auflösung.

$$\begin{aligned}\text{Spannung} &= \text{Auflösung} \cdot \texttt{digital gewandelter Messwert} \\ &= 4,88 \cdot 322 = 1,573\,\texttt{Volt}\end{aligned}$$

Es wurde ein Spannung von 1,573 V eingelesen.

Alle gewandelten Werte liegen im Bereich 0-1023.

SPI

Der Baustein unterstützt den SPI-Mode 3. Er kann bis 200 ksps[1] übertragen. Wir steuern ihn mit 0,0625 MBit/S an. Ein Schreib-/Lesezyklus entspricht dem Auslesen eines zuvor gewählten Kanals. Hierzu müssen 3 Byte gesendet werden und entsprechend auch 3 Byte empfangen werden. Aber beim Senden[2] muss der zu lesende

[1] ksps: Kilosamples per second. Das sind 1000 Samples in der Sekunde. Ein Sample entspricht hier einem gewandelten Wert.

[2] Wir beschränken uns hier auf die Einzelkanalmessung (single-ended). Der Baustein unterstützt auch Differential-Messung (Kanal gegen Kanal) (siehe Datenblatt).

SPI-Modus	CPOL	CPHA
3	1	1

Tabelle 2.1.

Kanal (Channel 0-7) angegeben werden. Dieser Kanal wird gegenüber Masse gemessen. Die Bits $D_2 D_1 D_0$ bestimmen dabei den Messkanal. Die gelesenen Daten

1. Byte								2. Byte								3. Byte								
0	0	0	0	0	0	0	1	1	D_2	D_1	D_0	0	0	0	0	0	0	0	0	0	0	0	0	schreiben
Schreib-/Lesezyklus																								↻
1. Byte								2. Byte (High-Byte)								3. Byte (Low-Byte)								lesen
?	?	?	?	?	?	?	?	?	?	?	?	?	0	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	

Tabelle 2.2. Beim Schreib-/Lesezyklus werden sowohl 3 Byte geschrieben als auch gelesen. Beim Schreiben bestimmen die Bits $D_2 D_1 D_0$ den Messkanal, während der gewandelte Wert in den Bits $B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9$ zur Verfügung steht.

enthalten im zweiten und dritten Byte den Messwert in den Bits B_0 bis B_9. D. h., dass das erste Byte verworfen werden kann. Damit befinden sich die beiden höchstwertigsten Bits im 2. Byte. Die anderen 8 Bits befinden sich im 3. Byte (Low-Byte). Beim IO-Warrior24 dauert ein Schreib-/Lesezyklus ca. 10 ms, sodass ohne Probleme 100 Messwerte in der Sekunde gelesen werden können.

2.2 Bauteileliste

Zum Testen des Bausteins verwenden wir eine normale 1,5 Volt Batterie. Der Wandler wird am Steckbrett benutzt. Weitere Bauteile sind nicht nötig.

2.3 Hardware - Schaltplan

Der MCP3008 wird auf ein Steckbrett gesteckt und mit dem IO-Warrior24 nach dem Schaltplan verbunden. Anschließend kann, z. B. über CH0, die Spannung einer Batterie gemessen werden, um die korrekte Funktionsweise des Wandlers zu überprüfen. Die Zuordnung der MCP3008-Pins zu den IO-Warror24 Pins zeigt die folgende Tabelle:

2.4 Software - Ansteuerung

Das Beispielprogramm verwendet die Klasse MCP3008. Diese Klasse enthält die Methode `open`, die die Verbindung mit dem IO-Warrior24 herstellt und den SPI

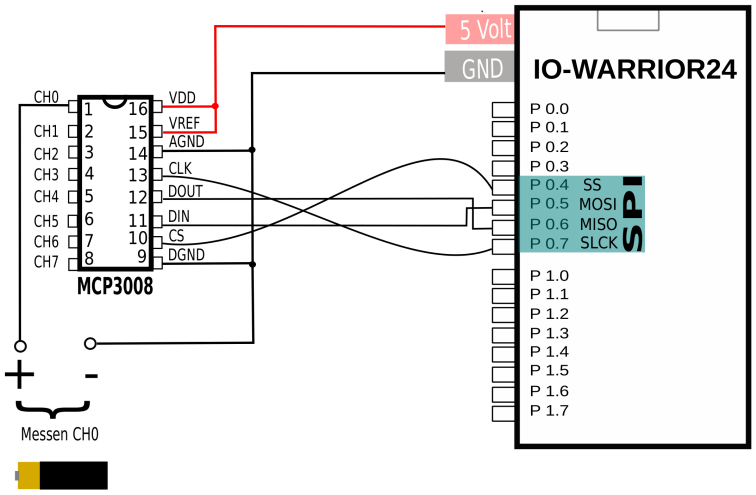


Abbildung 2.2. Schaltplan: Über Kanal 1 (CH0) soll die Spannung der Batterie gemessen werden.

PIN MCP3008		PIN IO-Warror24
9	↔	**GND**
10	↔	**P0.4**
11	↔	**P0.5**
12	↔	**P0.6**
13	↔	**P0.7**
14	↔	**GND**
15	↔	**5 Volt**
16	↔	**5 Volt**

Tabelle 2.3. Pinzuordnung

Modus aktiviert. Entsprechend schließt die Methode `close()` die Verbindung und deaktiviert zuvor die SPI-Funktionen. In einer For-Schleife wird dann wiederholt die Methode `read(int channel, int vref)` aufgerufen. Der erste Parameter ist der zu messende Kanal und der zweite Parameter die angelegte Referenzspannung. Diese Methode liefert den Spannungswert als double zurück, der anschließend auf der Konsole ausgegeben wird.

Listing 2.1. Main.java

```
1 public class Main {
2
3         public static void main(String[] args) {
4
5                 MCP3008 wandler = new MCP3008();
6
```

```
7                   /* Verbindung oeffnen, SPI Aktivieren */
8                   wandler.open();
9
10                  for (int i = 0; i < 1000; i++) {
11                          double value = wandler.read(0 /*
                                channel */, 5000 /* vRef/mV */);
12                          System.out.println("gelesene Spannung:
                                 " + value);
13                  }
14
15                  /* SPI deaktivieren, Verbindung beenden */
16                  wandler.close();
17
18          }
19  }
```

Im Weiteren wird die Funktionsweise der Klasse MCP3008 besprochen.

Die Klasse MCP3008

Die Klasse MCP3008 bestitzt die folgenden Methoden:

- **MCP3008()** Im Konstruktor der Klasse wird eine Verbindung zu dem angeschlossenen Gerät hergestellt. Der Gerätezugriff (h) wird in allen anderen Methoden benötigt. SPI-Reports sind 8 Byte lang. Da beim Lesen aber nur der Messkanal variabel ist, werden alle anderen Bytes bereits initialisiert.
- **open()** Über die Report-ID=`0x08` wird die SPI Funktionalität hergestellt.
- **read(int channel)** Dies ist die wichtigste Methode der Klasse. Über das vierte Report-Byte wird der Kanal ausgewählt und anschließend 3 Byte an den Baustein geschrieben. Danach wird ein Report gelesen und das Ergebnis in Array `lsReceive` gespeichert. Das High-Byte wird mit 3 maskiert und verschoben. Eine Addition des highByte mit dem lowByte ergibt jetzt den Messwert als Zahl zwischen 0 und 1023.
- **read(int channel, double vref)** berechnet unter Aufruf von `read(int channel)` den gewandelten Wert in einen Spannungswert (mV) um.
- **close()** Die SPI-Funktionalität wird abgeschaltet und die Verbindung wird geschlossen.

Listing 2.2. MCP3008.java

```
1   import com.codemercs.iow.IowKit;
2
3   public class MCP3008 {
4
5           private long h; /* Geraetezugriff */
6           private int pipe = 1; /* SPI benutzt immer Pipe 1 */
7           private int[] lsSend = new int[8];
8           private int[] lsReceive = new int[8];
```

```
9
10          public MCP3008() {
11
12                  h = IowKit.openDevice();
13
14                  lsSend[0] = 0x9; /* ReportID */
15                  lsSend[1] = 3; /* write 3 Bytes */
16                  lsSend[2] = 1; // Start Bit
17                  lsSend[3] = 128; // -> default Channel 0
18                  lsSend[4] = 0; // do not care byte
19          }
20
21          public void open() {
22
23                  /* enable SPI */
24                  int[] report = new int[8];
25                  report[0] = 0x8; /* ReportID */
26                  report[1] = 1; /* SPI on */
27                  report[2] = 15; /* 0.0625 MBit/S CPHA=1,CPOL=1
                        */
28                  /* lsSend[3-7] sind default auf 0 */
29                  long result = IowKit.write(h, pipe, report);
30                  if (result != report.length)
31                          System.out.println("ERROR: MCP3008.
                            open()");
32
33          }
34
35          public int read(int channel) {
36
37                  /* step 1: Report schreiben, um den
                        Datentransfer einzuleiten */
38                  // lsSend[0] = 0x9; /* ReportID */
39                  // lsSend[1] = 3; /* write 3 Bytes */
40                  // lsSend[2] = 1; // Start Bit
41                  lsSend[3] = 128 + (channel << 4); // Single +
                        Channel
42                  // lsSend[4] = 0 - do not care byte - already
                        set
43                  IowKit.write(h, pipe, lsSend);
44                  /* step 2: Report lesen und Daten auswerten */
45                  lsReceive = IowKit.read(h, pipe, 8);
46
47                  int highByte = lsReceive[3];
48                  int lowByte = lsReceive[4];
49
50                  /* maskieren mit 11 */
51                  highByte = highByte & 3;
52                  /* verschieben */
```

```
53                 highByte = highByte << 8;
54                 int result = highByte + lowByte;
55                 return result;
56 
57         }
58 
59         public double read(int channel, double vref /* In mV
               */) {
60                 return read(channel) * vref / 1024.0;
61         }
62 
63         public void close() {
64 
65                 int[] report = new int[8];
66                 report[0] = 0x8; /* ReportID */
67                 report[1] = 0; /* SPI off */
68                 long result = IowKit.write(h, pipe, report);
69 
70                 if (result != report.length)
71                         System.out.println("ERROR: MCP3008.
                            close()");
72 
73                 IowKit.closeDevice(h);
74 
75         }
76 
77 
78 
79         /**
80          * Method to measure elapsed time since start time
                until now
81          *
82          * Example: long startup = System.nanoTime();
83          * System.out.println("This took " + stopWatch(startup
                ) + " seconds.");
84          *
85          * @param startTime
86          *            Time of start in Nanoseconds
87          * @return Elapsed time in milliseconds as double
88          */
89         public static double stopWatch(long startTime) {
90                 double elapsedTime = 0.0;
91                 elapsedTime = System.nanoTime() - startTime;
92 
93                 return (double) elapsedTime / 1000000.0;
94         }
95 
96 }
```

3

MPC3008 - 10-Bit AD-Wandler - Entladekurve eines Kondensators

Die Entladung eines Kondensators soll visualisiert werden. Die mit dem A/D-Wandler gewonnen Daten können z. B. mit LibreOffice Calc eingelesen und weiterverarbeitet werden.

KONDENSATOR
$Q = C \cdot U$

$Q = C \cdot U$ beschreibt den Zusammenhang von Ladung (Q) und Spannung (U) am Kondensator. Die Ladung ist dabei der Spannung proportional. Die Proportionalitätskonstante (C) nennt man Kapazität. Sie besitzt die Einheit Farad (F).

3.1 Kondensator

Kondensatoren speichern Energie, die sie natürlich auch wieder abgeben können. Ein einfacher Versuch soll zeigen, wie die Energieabgabe von der Zeit abhängt. Dazu soll ein Programm "Kondensator" erstellt werden, dass jede Sekunde einen Messwert der noch vorhandenen Spannung liefert. Die Messwerte werden in einer Datei gespeichert, um sie grafisch darstellen zu können. Die grafische Darstellung kann mit einer der folgenden Methoden erfolgen:

- Einlesen der Daten und grafische Darstellung mit *LibreOffice Calc*.
- Einlesen der Daten und grafische Darstellung mit *Gnuplot* oder *XmGrace*.
- Einlesen der Daten und grafische Darstellung mit *Mathematica*.

Der Versuch verwendet Elektrolytkondensatoren (Elko's). Diese haben im Gegensatz zu anderen Kondensatoren eine Polarität, d. h., sie haben einen positiven und einen negativen Pol. Deshalb ist es bei ihnen wichtig, die Einbaurichtung zu beachten. Die Batterie sollte natürlich auch richtig herum angeschlossen werden.

3.2 Bauteileliste

Als AD-Wandler wird der Baustein MCP3008 (10-Bit AD-Wandler) des vorherigen Kapitels verwendet. Neben einer handelsüblichen 1,5 Volt Batterie werden dann noch ein Elektrolytkondensator und ein Widerstand benötigt. Für diesen Versuch wurde ein Kondensator mit 100 μF und ein Widerstand mit 100 $k\Omega$ verwendet. Es kommt aber nicht auf die genauen Werte dieser Bauteile an (siehe Tabelle).

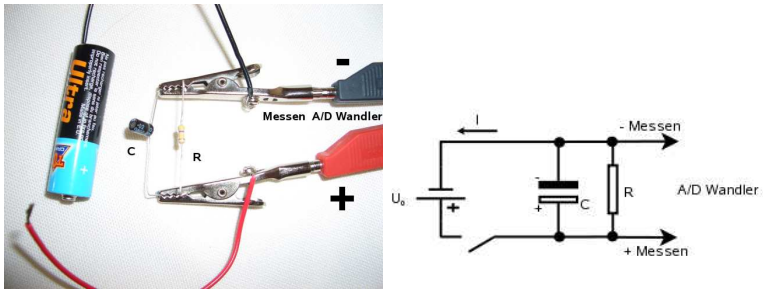


Tabelle 3.1. Versuchsaufbau und Schaltplan zur Aufnahme der Kondensatorentladekurve.

Menge	Bezeichnung	
1	C	Kondensator z. B. 22 μF Elko ($10\mu F - 100\mu F$)
1	R	Widerstand z. B. 100 $k\Omega$ ($10k\Omega - 1000k\Omega$)
1		Batterie 1,5 Volt (max 5 Volt)
1	MCP3008	10-Bit Analog-Digial Wandler
		Kabel

Tabelle 3.2. Kondensator und Widerstand kosten ca. 40 Cent.

3.3 Software

Das Programm “Kondensator” verwendet die Klasse MCP3008 des vorherigen Kapitels. Es ist mit dem Programm zur Messung der Batterie-Spannung fast identisch, lediglich `Thread.sleep(1000)` ist eingefügt, um die Messwerte in Abständen von einer Sekunde zu erzeugen. Startet man dieses Programm von der Konsole, liefert es jede Sekunde einen Messwert.

Listing 3.1. Kondensator.java

```
1
2  public class Kondensator {
3
4          public static void main(String[] args) throws
               InterruptedException {
5
6                  MCP3008 wandler = new MCP3008();
7                  int delay = 1000; // eine Sekunde warten
8
9                  /* Verbindung oeffnen, SPI Aktivieren */
10                 wandler.open();
11
```

```
12                  for (int i = 0; i < 300; i++) {
13                          double value = wandler.read(0 /*
                                channel */, 5000 /* vRef/mV */);
14                          System.out.println(value);
15                          Thread.sleep(delay);
16                  }
17
18                  /* SPI deaktivieren, Verbindung beenden */
19                  wandler.close();
20
21          }
22
23
24  }
```

Für die grafische Auswertung sollten die Messwerte in einer Datei vorliegen. Deshalb wird der Pipe-Mechanismus benutzt und eine Datei `daten.txt` erzeugt.

AUSGABEUMLEITUNG
Pipes

```
java -jar Kondensator > daten.txt
```

Um den Inhalt dieser Datei grafisch darzustellen, kann man z. B.:

- Die Datei in eine Tabellenkalkulation einlesen. Zusätzlich müssen dann noch die Werte für die X-Achse (Abszisse, hier Zeitachse) erzeugt werden.
- Alternativ kann man auch das Programm Kondensator abändern, so dass Wertepaare ausgegeben werden, wie z. B.

  ```
  1 1.5
  2 1.5
  3 1.45
  ... u.s.w.
  ```

  Möchte man die Daten mit Gnuplot einlesen kann, der plot-Befehl verwendet werden.

  ```
  Terminal type set to 'qt'
  gnuplot> plot "daten.txt"
  gnuplot>
  ```

LINUX
bietet eine einfache Möglichkeit alle Ausgaben, die auf der Konsole erscheinen, in eine Datei umzuleiten (DAS NENNT MAN PIPEN). Dazu wird am Ende eines Befehls das > Zeichen gefolgt vom gewünschten Dateinamen angegeben.
ls > test.txt
Erzeugt die Datei: test.txt. Probieren Sie es aus!

Messvorgang

Wenn der A/D-Wandler läuft und wie vorgegeben aufgebaut ist, kann mit der Messung begonnen werden. Zunächst werden ein paar Messwerte mit geschlossenem Schalter (Batterie) aufgenommen, damit sich der Kondensator aufladen kann. Dann wird die Batterie entfernt und der Kondensator kann sich über den Widerstand R entladen. Bei den von mir verwendeten Werten für R und C dauert das etwa 20 Sekunden. Um den A/D-Wandler auszulesen, wurde das Java Konsolen-Programm “Kondensator“ benutzt, das jede Sekunde einen Messwert liefert.

Ergebnisse

Die Entladung des Kondensators folgt einer Exponentialfunktion. Die Spannung $U(t)$ als Funktion der Zeit besitzt folgende Form.

$$U(t) = U_0 \cdot e^{-\frac{t}{R \cdot C}}$$

- U_0, Spannung zum Zeitpunkt $t = 0$, d. h. die Ausgangsspannung der Batterie.
- $R \cdot C$, das Produkt aus Widerstand und Kondensator wird manchmal mit τ (Zeitkonstante) bezeichnet. Für jede Widerstand/Kondensatorkombination ist es eine Konstante.

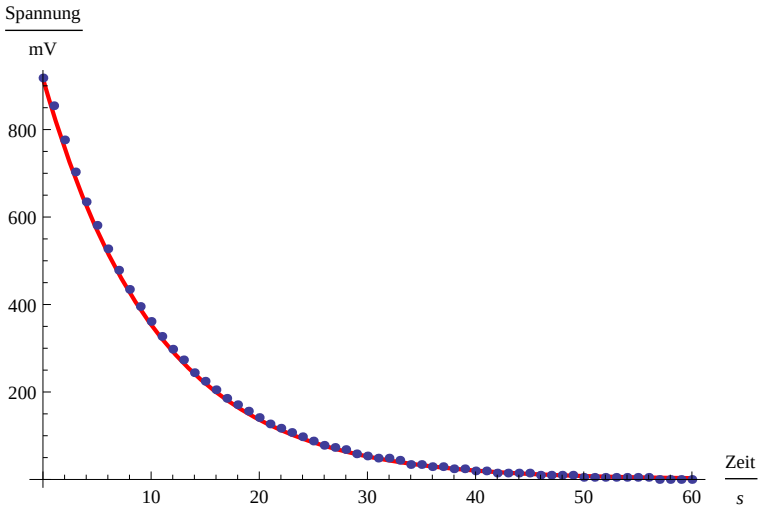


Abbildung 3.1. Die Punkte sind die aufgenommenen Messwerte. Die Kurve ist der theoretisch zu erwartende Verlauf. Wie man sieht, ist diese Messung ausgezeichnet verlaufen. Die Ausgangsspannung war $U_0 = 917$ mV. Es wurde ein Widerstand von 100 $k\Omega$ und ein Kondensator von 100 μF verwendet.

Quellen

Wer Schwierigkeiten mit der grafischen Darstellung der Messwerte hat, findet unter *www.physics4school.de* (Projekte, Kondensator) eine Anleitung, um Messdaten mit OpenOffice darzustellen.

Teil III

Anhang

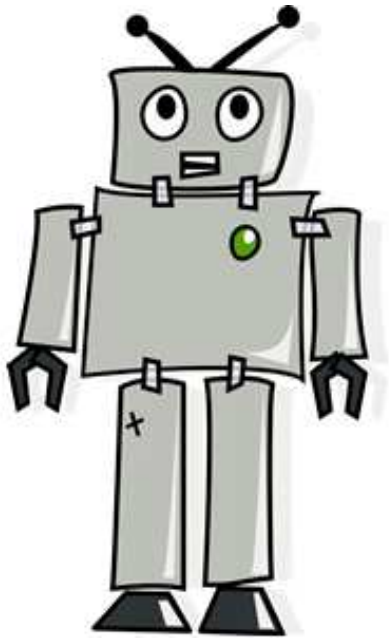
A

Quellcode

Der Quellcode zu den Projekten kann unter unter der folgenden URL heruntergeladen werden:
http://www.informatics4kids.de/index.php/list-download

Index

Openbook
Band IV



MSR mit Linux, Java und dem IO-Warrior24

Quellcode ↷ http://www.informatics4kids.de

Torsten Röhl