www.informatics4kids.de

Informatik in der Oberstufe

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

BAND I

Grundlagen

Torsten Röhl

Informatik in der Oberstufe

Messen, Steuern und Regeln mit Linux, Java und dem IO-Warrior24
Band I: Grundlagen

v.1.0.3 - Januar 2014

Openbook

Vorwort

Dies ist der erste Band der Reihe
INFORMATIK IN DER OBERSTUFE

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

1

Einführung

1.1 Messen, Steuern und Regeln

MESSLABOR VON GALILEO GALILEI (1564-1642)

DAS MESSLABOR von Galileo Galilei (Nachbildung - Deutsches Museum München). Die mit einem Heimcomputer und der in diesem Buch vorgestellten Hardware erzielten Messergebnisse übertreffen um ein Vielfaches die Genauigkeit, die Galilei einst erzielen konnte.

Dieses Buch ist eine praktische Einführung in das Gebiet “Messen, Steuern und Regeln” (MSR) mit dem Computer. Als Programmiersprache wird Java und als Betriebssystem wird Linux verwendet. Aufgrund der Plattformunabhängigkeit der Programmiersprache Java und der verwendeten Hardware (IOWarrior24) sind die Projekte aber auch für Benutzer anderer Plattformen (Windows, Mac OS) nutzbar. Dieses Buch verwendet ausschließlich den USB-Port zum Messen, Steuern und Regeln. Das Buch liefert einen elementaren Zugang zu der erwähnten Thematik, der neugierig machen soll auf mehr.
Es ist für das Selbststudium, sowie für den Einsatz in der Ausbildung gleichermaßen geeignet. Nachfolgend eine Liste der Leitideen dieses Buches:

- Konsequente Verwendung der Programmiersprache Java zum Messen, Steuern und Regeln.
- Verwendung des USB-Ports für alle Aufgaben. Die “klassischen alten Schnittstellen” werden dabei nicht mehr thematisiert.
- Verwendung nur eines Bausteins, des IO-Warrior24, zur Ansteuerung des USB-Ports. Wichtig war dabei eine gute Verfügbarkeit des Bausteins sowie ein gutes Preis/Leistungsverhältnis. Ziel ist es, mit diesem Baustein eine Reihe von Projekten zu bearbeiten und sich so tatsächlich mit dem Thema MSR auseinanderzusetzen. Der IO-Warrior24 von der Firma Code Mercenaries ist ein Baustein, der speziell für MSR Zwecke vertrieben wird.
- Elementare Darstellung der Thematik. Wohl wissend, das dies ein Balanceakt ist, soll der Versuch unternommen werden, unterschiedliche MSR-Projekte auch für den Anfänger zugänglich zu machen.
- Für Selbststudium und Ausbildung geeignet
- Für die MSR-Projekte verwendete Hardware soll möglichst günstig sein, so dass die einzelnen Projekte für jeden erschwinglich sind.

1.2 Die Beschaffung der Hardware

Früher oder später wird Hardware benötigt, um die Projekte umzusetzen. Allen Projekten liegt der Hardwarebaustein IO-Warrior24 der Firma Code Mercenaries zugrunde. Ohne diesen Baustein kann nicht sinnvoll mit diesem Buch gearbeitet werden. Diesen Baustein muss man sich aus dem Internet bestellen (siehe auch auf der Homepage zu diesem Buch).
Der genaue Aufbau der Grundschaltung, die allen MSR-Projekten zu Grunde liegt, wird im Kapitel “Inbetriebnahme des IO-Warriors24” im Abschnitt: Aufbau der IO-Warrior24 Grundschaltung im Detail erklärt. Die für die einzelnen Projekte nötige Hardware wird im jeweiligen Projekt aufgelistet. Diese Hardware kostet in der Regel nur wenige Euro, wobei die Bezugsquellen auf der Internetseite zu diesem Buch aufgelistet sind.

- Code Mecenaries Hard- und Software GmbH http://www.codemercs.com

Abbildung 1.1. Das fertige Starterkit von Code Mercenaries. Der Bausatz kostet ca 60 Euro. Es ist alles dabei, um sofort loslegen zu können.

Abbildung 1.2. Der IO-Warrior24 als einzelnen Baustein. Die aktuelle Version ist 1.0.3. (Stand 2011)

Dabei kann man sich für eine fertige Entwickler-Platine für ca. 60 Euro entscheiden, oder für einen einzelnen Baustein für ca. 12 Euro. Dieses Buch verwendet den einzelnen Baustein, hauptsächlich weil dies die günstigere Variante ist. Um den Baustein

in Betrieb nehmen zu können, werden noch 3 weitere Bauteile im Wert von ungefähr 1 Euro benötigt.

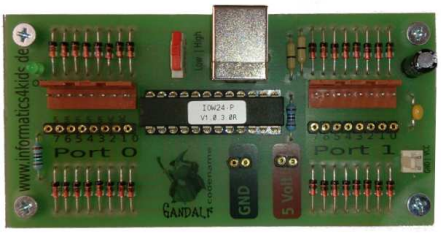


Abbildung 1.3. Das kleine Gandalf-Board kostet weniger als 1/3 des Starterkits und ist hervorragend geeignet, um mit einem Steckbrett zu Experimentieren.

Auf der Homepage dieses Buches *www.informatics4kids.de* gibt es eine komplette Anleitung, um sich ein kostengünstiges Board (Gandalf haben wir es genannt) zusammenzubauen.

1.3 Zielgruppe des Buches

Das eigentliche Ziel dieses Buches darin besteht, dem Leser beizubringen, wie man mit Hilfe des Computers und der Programmiersprache Java Messen, Steuern und Regeln kann. Zur Zielgruppe gehören demnach all diejenigen, die bereits Java Grundlagen haben, aber keinerlei Erfahrung im Bereich Messen, Steuern und Regeln aufweisen können, und dennoch Interesse am diesem Gebiet haben. Aber auch der umgekehrte Fall, d.h. diejenigen, die bereits über Erfahrungen in MSR verfügen und mit dem Entwickeln eigener Java-Software starten wollen, können von diesem Buch profitieren, obwohl der zuerst erwähnte Fall der typische sein dürfte. Dieses Buch soll einen möglichst elementaren Einstieg in die Welt des Messens, Steuerns und Regelns (MSR) liefern. Dadurch unterscheidet es sich von anderen Büchern, die sich diesem Thema widmen und gewöhnlich auf mehr Fachwissen voraussetzen. Dem Autor ist kein Buch bekannt, dass sich diesem Thema so elementar nähert. Bei Ansätzen dieser Art besteht immer die Gefahr, einerseits ins triviale abzurutschen und damit das eigentliche Thema, nämlich MSR gar nicht ausführlich zu behandeln. Oder auf der anderen Seite zu viel vorauszusetzen, so dass wiederum nur erfahrene Anwender einen Nutzen aus dem Buch ziehen können. Ein einfacher Einstieg beinhaltet zwar, dass wenig Grundwissen vorausgesetzt wird und dementsprechend ein genügend großer Raum geboten werden muss, dieses Grundwissen zu erwerben. Es bedeutet aber nicht, dass alles leicht und sofort verstanden werden kann. Ein Anfänger in MSR sollte die Bereitschaft mitbringen, sich intensiv mit der Materie auseinanderzusetzen. Tatsächlich sind die behandelten Projekte typische Projekte im Bereich des MSR und die Ergebnisse und Aussagen unterscheiden sich nicht wesentlich von der zur Verfügung stehenden (anspruchsvolleren) Literatur. “Elementar“ bezieht sich in

diesem Buch aber auch auf die Verwendung der Hardware, deren Funktionsweise noch einmal explizit erklärt wird, um das Wesentliche zu verstehen. Dies ist einerseits von Vorteil, wenn das Thema MSR in der Ausbildung (z.B. Schule) behandelt wird, andererseits ermöglicht es dieser Ansatz auch, dass sich Anfänger, ohne stundenlange Internetrecherchen machen zu müssen, die Thematik im Selbststudium aneignen können. Ich hoffe, dass dieses Buch ein nützlicher Ausgangspunkt für all diejenigen ist, die den eigenen Computer zum Messen, Steuern und Regeln verwenden wollen, aber nicht wissen, wo sie anfangen sollen. Entweder weil die gängige Literatur zu hoch ansetzt, oder weil das Hardwareangebot den Anfänger zu erschlagen droht. Das Buch soll dem Leser in die Lage versetzen, einerseits seine eigenen Ideen umsetzen zu können. Andererseits eine Anregung und Motivation sein, eigene Projekte und Ideen zu entwickeln und diese eigenständig umzusetzen. Abschließend soll noch einmal betont werden, dass sich dieser elementare Einstieg in die Welt des MSR vor allem an dem vorhandenen Vorwissen und nicht so sehr am Alter des an MSR interessierten Lesers orientiert.

1.4 Aufbau des Buches

Das Buch behandelt das Wissen, welches benötigt wird, um mit dem Computer etwas zu Messen oder Dinge Steuern und Regeln zu können, daher ist es naturgemäß etwas theoretischer als die nachfolgenden Bände, in denen es um konkrete Projekte geht. Behandelt werden folgende Themen:

- Java zum Messen, Steuern und Regeln
- USB
- Installation und Inbetriebname des Mikrokontrollers IO-Warrior24

Zuerst wird das nötige Javawissen vermittelt, dass für MSR wichtig ist. Die Schwerpunkte bilden Arrays und Bitoperatoren. Diejenigen, die bereits wissen, wie man in Java mit Arrays arbeitet, können diesen Abschnitt überspringen. Auch wenn sie bereits mit den Java-Bitoperatoren vertraut sind, wird empfohlen, diesen Abschnitt kurz zu überfliegen, da dieser Abschnitt einige Methoden bereitstellt, die im zweiten Teil häufig Verwendung finden werden. Nach dem Java-Kapitel wird die Installation der IO-Warrior Software (SDK) und die Inbetriebnahme des Bausteins unter Linux detailliert erklärt. Leser, die andere Plattformen einsetzen, können diesen Abschnitt überspringen. Anschließend werden die Grundlagen des USB-Ports behandelt. Es wird dabei ein praxisorientiertes Minimalwissen bereitgestellt. Als Interface (deutsch: Schnittstelle) wird in diesem Buch ausschließlich der USB-Port verwendet, da die klassischen Schnittstellen (leider) nicht mehr zeitgemäß und häufig gar nicht mehr vorhanden sind. Der letzte Teil des Buches beschäftigt sich ausführlich mit dem IO-Warrior24. Dieser Abschnitt sollte ausführlich studiert werden. Die Bände II-IV widmen sich dann ganz dem Thema Messen, Steuern und Regeln. Um mit diesem Buch arbeiten zu können, benötigen Sie natürlich auch die Hardware. Wie schon erwähnt, macht dieses Buch Gebrauch von spezifischer Hardware und zwar dem IO-Warrior24, der nur ein Baustein einer ganzen Serie bildet, die von Code Mercenaries

angeboten wird. Die anderen Bausteine aus dieser Serie werden nicht behandelt. Hier gilt das Motto: Weniger ist mehr. Nach dem Durcharbeiten des Buches sollte es aber ohne Schwierigkeiten möglich sein, auch die anderen Bausteine der Serie in Betrieb zu nehmen. Der IO-Warrior24 ist die wichtigste Hardwarekomponente, die in diesem Buch behandelt wird. Er wird als „Bindeglied“ benötigt, um all die andere behandelte Hardware (Motoren, Sensoren, ...) ansprechen zu können. Fast alle Kapitel enthalten einen Quellcode, der auf der Internetseite heruntergeladen werden kann. Auch wenn der Leser beim ersten Durcharbeiten der Projekte nicht gleich alles versteht, möchte ich ihn ermutigen, durchzuhalten. Lesen Sie dieses Kapitel zu Ende und versuchen Sie, die Quellcodebeispiele zum Laufen zu bringen. MSR ist naturgemäß ein eher schwieriges Thema, denn es gibt viele Fehlerquellen (auf der Hardwareseite und der Softwareseite). Lassen Sie sich davon nicht entmutigen.

1.5 Hintergrundwissen: Soft- und Hardware

Um dieses Buch verstehen zu können, muss sich der Leser ein wenig in der Java-Programmierung auskennen, idealerweise in Verbindung mit der frei verfügbaren Entwicklungsumgebung “Eclipse”. Schön wäre es, wenn Sie bereits ein paar Javaprogramme geschrieben haben, so dass Datentypen, Schleifen, Bedingungen, Klassen und Methoden keine Fremdwörter mehr sind. Java-Kenntnisse, die für Messen, Steuern und Regeln besonders wichtig sind, werden im Java-Teil dieses Buches noch einmal detailliert behandelt. Im Wesentlichen handelt es sich dabei um das Arbeiten mit Arrays und den Bitoperatoren. Eclipse ist nicht Voraussetzung. Jede andere Java Entwicklungsumgebung (IDE) oder das Arbeiten lediglich auf der Kommandozeile und mit Editor ist ebenso gut möglich. Ein wenig Linux-Grundwissen ist ebenfalls nötig, sofern Sie vorhaben, Linux einzusetzen. Der Autor nutzt ein Linux-OpenSuse System. Das Kapitel “Inbetriebnahme der IO-Warriors” erklärt, wie die IO-Warrior-Software (SDK) auf solch einem System installiert wird. Diese Informationen sind natürlich auch für andere Linux-Systeme notwendig. Alle die vorhaben, mit Linux zu arbeiten, sollten unbedingt einen Blick in dieses Kapitel werfen, denn die von Code Mercenaries herausgegeben Informationen sind nicht mehr auf dem neuesten Stand, bzw. unvollständig und ermöglichen es damit nicht immer, dass SDK richtig aufzusetzen, um mit der Programmierung beginnen zu können. Der eigentliche Hauptteil des Buches ist im Grunde plattformunabhängig, so dass es sich dabei ebenso gut auch um ein Windowssystem handeln könnte. Auf der Hardwareebene sind keine Vorkenntnisse notwendig. Alles Nötige wird genau erklärt.

1.6 Informationsquellen

Es gibt viele gute und interessante Bücher zum Thema “Messen, Steuern und Regeln”, die jedoch im allgemeinen deutlich mehr Fachwissen voraussetzen. Auf der Internetseite zu diesem Buch finden sie eine Reihe von Links. Da die “Halbwertszeit”

der Internetquellen enorm kurz ist, wurde auf eine detaillierte Auflistung der Internetquellen im Buch verzichtet, da auf diese Weise auf Änderungen viel schneller reagiert wird, so dass Sie dort immer eine Reihe aktueller Quellangaben zur Verfügung haben. Dennoch sollen hier drei für dieses Buch wichtige Links genannt werden:

- Informatics4kids.de `http://www.informatics4kids.de`
  - Dies ist die Internetseite zum Buch. Sie enthält u.a. den Quellcode für die Beispielprogramme dieses Buches zum Downloaden, sowie sämtliche Datenblätter zu der verwendeten Hardware. Wichtig für das Arbeiten mit diesem Buch ist, dass auf dieser Seite auch das modifizierte SDK von Code Mecenaries zu finden ist, das für die Installation auf Linux-Systemen optimiert wurde.
- Eclipse `http://www.eclipse.org/downloads/`
  - Dies ist die Internetseite, um die Entwicklungsumgebung Eclipse herunterzuladen. Im Anhang befinden sich Tipps für den Download. Bei Eclipse beschränkt sich die Installation darauf, das heruntergeladene Programm lediglich entpacken zu müssen. Danach kann sofort losgelegt werden.
- Code Mecenaries Hard- und Software GmbH `http://www.codemercs.com`
  - Hier können Sie den IO-Warrior24-Baustein beziehen. Dabei ist es sowohl möglich, ein Starterkit zu erwerben, als auch den Baustein einzeln zu kaufen. Die Seite listet auch eine Reihe anderer Bezugsquellen auf. Mehr dazu im Kapitel “Inbetriebnahme des IO-Warrior”.

2

Messen, Steuern und Regeln mit Java

Java zum Messen, Steuern und Regeln

Die Verwendung von Java zum Messen, Steuern und Regeln bietet eine Reihe von Vorteilen, von denen wir einige auflisten:

- Java ist eine objektorientierte Programmiersprache, die einfacher zu erlernen ist als C/C++.
- Java ist plattformunabhängig, d.h., Java läuft ausgereift auf vielen Betriebssystemen (Windows, Linux, OS X).
- Eclipse stellt für die tägliche Arbeit mit Java eine kostenlose und professionelle Entwicklungsumgebung (IDE) zur Verfügung.

JAVA-SYMBOL

Häufig wird eine Kaffeetasse als Java-Symbol verwendet. Viele Javaprogrammierer lieben Kaffee. Es ist aber auch möglich, ein guter Javaprogrammierer zu werden, wenn man Kaffee nicht leiden kann. Übrigens, Java ist auch eine Insel im Indischen Ozean.

Die grundliegende Idee, die Java von anderen Programmiersprachen unterscheidet, wird in dem Slogan: *Compile once run everywhere* zusammengefasst. Das bedeutet, dass man ein in Java geschriebenes Programm nur einmal übersetzen (kompilieren) muss und es dann überall lauffähig ist. Dieser Grundsatz gilt natürlich nur dann, wenn wir davon absehen, Hardwarenahe (architektur-spezifische) Programmierung anzuwenden. Wie wird die Plattformunabhängigkeit in Java umgesetzt?
Jedes Java-Programm muss übersetzt (kompiliert) werden. Dabei entsteht aus dem Quellcode (Sourcecode), der immer auf `.java` endet, nicht ein Maschinencode (wie z.B. bei C/C++), der sofort ausgeführt werden kann, sondern ein plattformunabhängiger Bytecode (siehe Abbildung 2.1). Den Bytecode erkennt man an der Endung `.class`. Um nun das Java Programm ausführen zu können, wird zusätzlich eine Java-Laufzeitumgebung benötigt (JRE = Java Runtime Environment). Die Laufzeitumgebung übersetzt das Programm in den notwendigen Maschinencode und führt es aus (Abbildung 2.2). Die Laufzeitumgebung gibt es für die verschiedenen Betriebssysteme kostenlos zum Downloaden. Das Übersetzen des Bytecodes in den Maschinencode geschieht dabei im Hintergrund. Der Programmierer oder Anwender bekommt davon nichts mit. Der Anwendungsbereich von Java ist vielfältig. Wir können mit Java ein Konsolenprogramm oder ein Programm mit grafischer Oberfläche für ein Betriebssystem erstellen. Ein Applet ermöglicht es, dass Java-Programme, die z.B. auf einer Homepage installiert sind, von einem Browser aufge-

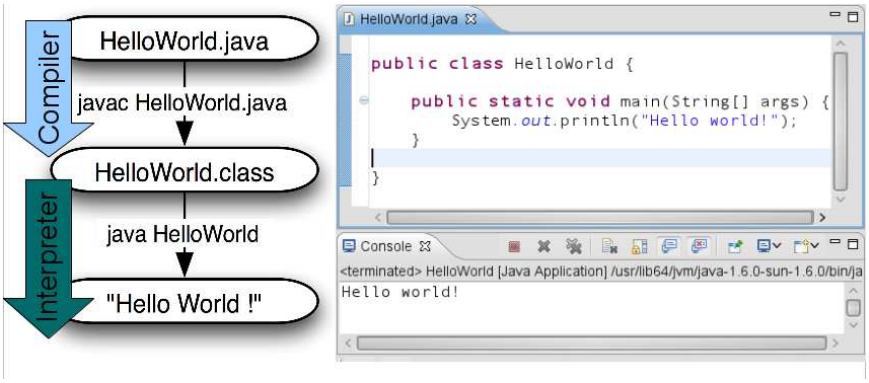


Abbildung 2.1. Das berühmte “Hello World”-Programm. Ein “Hello World”-Programm macht nichts anderes, als die Zeichenfolge “Hello World” auszugeben. Der Zweck des Programms liegt darin, zu zeigen, welche Bestandteile ein Minimalprogramm in einer bestimmten Programmiersprache besitzt. Außerdem kann man damit testen, ob die Programmierumgebung (Bibliotheken und die Entwicklungsumgebung) richtig installiert sind und laufen. Bei Wikipedia sind “Hello World” Programme für mehr als 100 verschiedene Programmiersprachen gelistet.

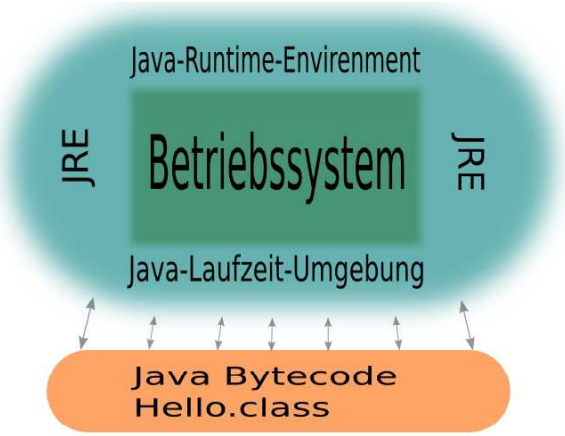


Abbildung 2.2. Jedes Java Programm benötigt eine Laufzeitumgebung (JRE). Die Laufzeitumgebung führt das Programm aus. Laufzeitumgebungen stehen für alle Betriebssysteme kostenlos zur Verfügung.

rufen werden können. Von vielen anderen Java-Technologien, die Java ermöglicht, einmal abgesehen. Für unsere Zwecke am interessantesten ist eine Technologie, die man als Java Native Interface (JNI) bezeichnet. Mit Hilfe von JNI ist es möglich, von Java aus auf C/C++ Code zuzugreifen. JNI ermöglicht auch den umgekehrten Weg, also die Nutzung von Java von C/C++ aus, der für unsere Zwecke aber nicht von Bedeutung ist. In anderen Worten mit dem als JNI bezeichneten Mechanismus ist es auch mit Java möglich, hardwarenah zu arbeiten, denn Treiber für Hardware stehen in der Regel in C/C++ zur Verfügung. Damit man also auch mit Java messen, steuern und regeln kann, muss lediglich die Funktionalität des Treibers, der in C/C++ vorliegt, mit JNI in einer entsprechenden Java-Bibliothek umgesetzt werden. Für den Java-Programmierer bleibt davon das Meiste verborgen. Im Hintergrund bewirkt jetzt aber ein Aufruf einer Java-Methode den Aufruf der entsprechenden C/C++ Methode und damit die Kommunikation mit der anzussteuernden Hardware. Der ursprüngliche Ansatz **Compile once run everywhere** von Programmen, die Gebrauch von JNI machen, gilt dann allerdings nicht mehr. Trotzdem kann diese Vorgehensweise von Vorteil sein. Wir kreieren einen neuen Slogan: **Learn once use anyware**. In vielen Situationen kann man davon profitieren, nur einmal eine Programmiersprache lernen zu müssen, um sich dann mit ihrer Hilfe die grundlegenden Prinzipien der Informationstechnologie zu vergegenwärtigen. Im Ausbildungsbereich wäre es geradezu unsinnig, gerade gelernte Prinzipien einer Programmiersprache, kaum ausführlich angewandt, wieder aufgeben zu müssen, um mit der nächsten Programmiersprache starten zu müssen, um z.B. die Grundlagen des Messens mit dem PC erlernen zu können. Mit der einmal gelernten Programmiersprache Java kann man sehr viel lernen auch in Sachen Messen, Steuern und Regeln. Und diejenigen, die es ganz genau wissen wollen, können das hier in Java erworbene Wissen gewinnbringend verwenden, wenn sie besonders tief eintauchen wollen in die Mess-, Steuer-, und Regelungstechnik.

Die nächsten Abschnitte wiederholen diejenigen Gebiete der Java-Programmierung, die für MSR besonders wichtig sind. Es wird davon ausgegangen, dass bereits Grundwissen in Java und im Umgang mit der Programmierumgebung (IDE) Eclipse vorhanden ist.

Im Einzelnen behandeln wir:

- Zahlensysteme
  Der Umgang mit dem dezimalen, binren und hexadezimalen Zahlensystem mit Java wird erläutert. Die Darstellung sollte ausführlich genug sein, um Vergessenes aufzufrischen.
- Arrays
  Sie werden benötigt, um Daten an Geräte zu senden bzw. zu empfangen
- Logik-Operationen
  Obwohl wir sie als bekannt voraussetzen, werden sie kurz wiederholt, denn sie bereiten den nächsten Abschnitt (Bits und Bytes) vor.
- Bits und Bytes
  Sie werden z.B. immer dann benötigt,wenn man die Zustände der I/O Ports des IO-Warriors erfahren oder verändern will.

2.1 Zahlensysteme in Java

Das uns vertraute 10er Zahlensystem (Dezimalsystem) reicht für MSR nicht aus. Befehle, die wir an die Hardware schicken und Adressangaben werden üblicherweise im Hexadezimalsystem (16er-System) angegeben. Daten, die wir an ein Gerät senden oder von ihm empfangen wollen, müssen zuvor binär (2er-System oder Dualsystem) verschlüsselt bzw. entschlüsselt werden. Wie das mit Java gemacht wird, zeigt dieser Abschnitt.
Das folgende Programm gibt die (dezimale) Zahl 1212 zusätzlich im Hexadezimal- und Binärsystem aus.

Listing 2.1. Beispiel21.java

```
1  public class Beispiel21{
2
3          public static void main(String[] args) {
4
5           int zahl = 1212;
6           System.out.println("Dezimal:\t"     + zahl);
7           System.out.println("Hexadezimal:\t" + Integer.
                toHexString(zahl));
8           System.out.println("Bin rzahl:\t"   + Integer.
                toBinaryString(zahl));
9          }
10 }
```

Das Programm liefert die folgende Ausgabe:

```
Dezimal:  1212
Hexadezimal: 4bc
Binärzahl:  10010111100
```

Mit den bereits in Java vorhandenen Methoden `Integer.toHexString(int zahl)` und `Integer.toBinaryString(int zahl)` wird die hexadezimale und binäre Ausgabe erzeugt.
Das funktioniert auch mit einer im hexadezimalen System gegebenen Zahl, wobei wir in Java lediglich die beiden Zeichen `0x` einer Zahl voranstellen müssen, wenn wir im hexadezimalen System arbeiten wollen.

Listing 2.2. Beispiel21a.java

```
1  public class Beispiel21a {
2
3    public static void main(String[] args) {
4   // hexadezimalen Zahlen muss ein 0x vorangestellt werden!
5    int zahl = 0x4bc;
6    System.out.println("Dezimal:\t"     + zahl);
7    System.out.println("Hexadezimal:\t" + Integer.toHexString(
         zahl));
8    System.out.println("Binaerzahl:\t"   + Integer.
         toBinaryString(zahl));
```

```
9     }
10 }
```

Ausgabe von Beispiel21a:

```
Dezimal:  1212
Hexadezimal: 4bc
Binärzahl:  10010111100
```

Das Beispiel21a liefert die selbe Ausgabe wie das Beispiel21. Das erste Beispiel benutzt aber die Dezimalzahl `1212`, während das Zweite mit der Hexadezimalzahl `0x4bc` rechnet.

MERKE

In Java haben Hexadezimalzahlen das Präfix `0x`.

Umrechnen der Zahlensysteme

Für unsere Zwecke genügt es, hexadezimale oder binäre Zahlen ins Dezimalsystem umrechnen zu können. Die Anzahl der Ziffern, die ein Zahlensystem besitzt, nennt man die Basis des Zahlensystems. Die Basis ist der Schlüssel, um Zahlensysteme ineinander umrechnen zu können. Die erlaubten Ziffern für das jeweilige System zeigt die Tabelle. Durch die Angabe der Ziffern ist das Zahlensystem festgelegt. Dezimalzahlen werden meist ohne Index verwendet. Falls es Verwechslungsmöglichkeiten zwischen den Binärzahlen mit den Dezimalzahlen geben könnte, verwenden wir für das erstere den Index 2. So ist z.B. 1012 die Binärzahl und nicht die Dezimalzahl 101. Für Hexadezimalzahlen sind eine Reihe von Schreibweisen in Gebrauch, z.B. `ab116, ab1hex, ab1h, ab1H` oder `ab1`. Wir verwenden `0x` wie es für Java notwendig ist. Im Datenblatt des IO-Warriors wird die Schreibweise mit `$` benutzt.

Zahlensystem	Basis	Ziffern	Beispiel
Dezimalsystem	10	0,1,2,3,4,5,6,7,8,9	345
Binärsystem	2	0,1	1010101
Hexadezimalsystem	2	0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f	`0xab`

Tabelle 2.1. Die Anzahl der Ziffern, die ein Zahlensystem bilden, nennt man die Basis des Zahlensystems.

Das Umrechnen einer Binärzahl oder einer Hexadezimalzahl ins Dezimalsystem ist besonders wichtig. Die Zahl 2320 lässt sich folgendermaßen in ihre “Bestandteile” (Einer, Zehner, usw.) zerlegen.

Basis 10	**Tausender**	**Hunderter**	**Zehner**	**Einer**	
	10^3=1000	10^2=100	10^1=10	10^0=1	
2320=	2	3	2	0	=2·1000+3·100 + 2· 10 + 0

Tabelle 2.2.

Das gleiche Prinzip gilt für jedes andere Zahlensystem auch. Für das Binärsystem ist die Basis 2 und für das Hexadezimalsystem wird als Basis 16 verwendet. Für die Binärzahl 10112 erhalten wir somit den Wert 11 im Dezimalsystem. Für Hexa-

Basis 2	**4. Stelle**	**3. Stelle**	**2. Stelle**	**1. Stelle**	
	2^3=8	2^2=4	2^1=2	2^0=1	
1011_2 =	1	0	1	1	=1·8+0·4 + 1·2 + 1·1 = 11

Tabelle 2.3.

dezimalzahlen benötigt man noch die Zuordnung der Ziffern a, b, c, d, e, f zu den Dezimalzahlen. Es gilt: a=10, b=11, c=12, d=13, e=14, f=15. Damit ergibt sich für die Hexadezimalzahl 0xab03 ein Wert von 43779 im Dezimalsystem. Schauen Sie

Basis 16	**4. Stelle**	**3. Stelle**	**2. Stelle**	**1. Stelle**	
	16^3=4096	16^2=256	16^1=16	16^0=1	
0xab03 =	a	b	0	3	=a·4096+b· 256 + 0· 16 + 3·1
					=10·4096+11· 256 + 0· 16 + 3·1
					=43779

Tabelle 2.4.

sich die drei letzten Beispiele noch einmal an und versuchen Sie das allgemeine Schema, das hinter den Umrechnungen steckt herauszufinden. In der nachfolgenden

Basis b	**u.s.w.**	**3. Stelle**	**2. Stelle**	**1. Stelle**	
		b^2	b^1	b^0=1	
Ziffer3 Ziffer2 Ziffer1 =	...	Ziffer3	Ziffer2	Ziffer1	=Ziffer3· b^2+Ziffer2· b+Ziffer1

Tabelle 2.5.

Tabelle sind die behandelten Zahlensysteme bis 15 aufgetragen. Eine Hexadezimalziffer kann dabei vier binäre Bits repräsentieren. Das ist der Grund, weswegen sie in

der Literatur häufig Verwendung finden. In Java spielt es keine Rolle, ob die Ziffern a-f groß oder klein geschrieben sind, d.h., es gilt z.B. 0xabcd = 0xABCD.

Dezimal	Hex	Binär
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	a	1010
11	b	1011
12	c	1100
13	d	1101
14	e	1110
15	f	1111

Tabelle 2.6. Dezimale, hexadezimale und binäre Zahlen. In Java können die hexadezimalen Zahlen (A, B, C, D, E, F) groß oder klein geschrieben werden.

Das folgende Programm gibt eine im String `dualzahl` angegebene Dualzahl im Dezimalsystem aus.

Listing 2.3. Beispiel21b.java

```
1  public class Beispiel21b {
2   public static void main(String[] args) {
3          String dualzahl = "1011011";
4          int faktor = 1;
5          int ergebnis = 0;
6          for (int i = dualzahl.length() - 1; i >= 0; i--) {
7           int zahl = dualzahl.charAt(i) == '1' ? 1 : 0;
8           ergebnis += faktor * zahl;
9           faktor *= 2;
10         }
11  System.out.println("(Dualzahl) "+ dualzahl+" -> "+ergebnis+ "
        (Dezimalzahl)");
12
13  }
14 }
```

Wie erhalten folgende Ausgabe:

```
(Dualzahl) 1011011 -> 91 (Dezimalzahl)
```

EXKURS: BEDINGUNGSOPERATOR ? :

Das Beispiel21b verwendet den Bedingungsoperator: '?:' Diesen Operator kann man immer durch eine if-Anweisung ersetzen, d.h. anstatt:

```
int zahl = dualzahl.charAt(i) == '1' ? 1 : 0;
```

kann auch geschrieben werden:

```
int zahl = 0;
if( dualzahl.charAt(i) == '1' )
   zahl = 1;
```

Damit hat zahl den Wert 0, es sei denn der String hat an der i-ten Stelle eine 1, dann wird zahl mit 1 überschrieben.

```
Bedingung ? Ausdruck1 :  Ausdruck2
```

Der Bedingungsoperator prüft den logischen Ausdruck (die Bedingung), der vor dem ? steht. Ergibt die Auswertung dieser Bedingung `true`, dann wird der Ausdruck `1` angenommen. Ergibt die Auswertung `false`, dann wird der Ausdruck `2` angenommen. Das ist alles.

Als abschließendes Beispiel wird der Operator benutzt, um zu entscheiden, ob die Zahl i gerade oder ungerade ist.

```
boolean gerade= (i%2 == 0) ? true : false;
if( gerade )
   System.out.println( i + "ist gerade");
   else
    System.out.println( i + " ist ungerade");
```

2.1.1 Aufgaben

Aufgabe 1

Welchen Wert hat die Dualzahl 1000110 im Dezimalsystem?

Aufgabe 2

Welchen Wert hat die Hexadezimalzahl 0xcf7 im Dezimalsystem?

Aufgabe 3

Welchen Wert hat die Zahl 12023 im Dezimalsystem?

Aufgabe 4

Schreiben Sie ein Programm, das als Ausgabe die Tabelle 2.6 erzeugt.

Aufgabe 5

Schreibe aufbauend auf dem Beispiel21b ein Programm, das anstelle eines Dualstrings einen hexadezimalen String erwartet, z.B. “abc3” und den entsprechenden Wert im Dezimalsystem ausgibt.

2.2 Arrays

Bei der Behandlung von Arrays (auch als Reihungen oder Felder bezeichnet) beschränken wir uns auf die Behandlung eindimensionaler Arrays. Sie werden für die Kommunikation mit dem IO-Warrior später häufig Verwendung finden.

2.2.1 Array erstellen, lesen und schreiben

Ein Array kann man sich als eine „Liste“ oder „Tabelle“ veranschaulichen, die viele Werte (Elemente) aufnehmen kann. Alle Elemente des Arrays müssen dabei vom selben Typ sein, wobei Arrays aber nicht auf primitive Datentypen (boolean, int, double usw) beschränkt sind. Arrays können beliebige Objekte speichern.
Eine Tabelle, die nur eine Zeile mit mehreren Spalten besitzt, wäre dann z.B. ein eindimensionales Array. Die folgende Tabelle könnte ein Array repräsentieren.

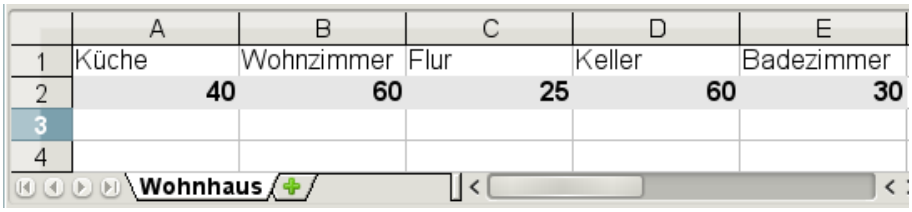


Abbildung 2.3. Eine Zeile einer Tabellenkalkulation könnte ein Array repräsentieren.

Ein Array besitzt einen eindeutigen Namen (hier Wohnhaus) und Typ (hier z.B. double) und mehrere Einträge (einzelne Felder oder Elemente), wobei Werte gespeichert, verändert oder gelesen werden können (hier die Raumgröße in m^2). Die Anzahl der Felder muss beim Erstellen des Arrays - das kann auch erst zur Laufzeit geschehen - festgelegt werden (hier 5 Räume) und kann dann nicht mehr geändert werden. Das gilt für eine Tabellenkalkulation natürlich nicht.
Die obige Tabelle könnte folgendermaßen in Java implementiert werden.

Listing 2.4. Beispiel221.java

```
1 public class Beispiel221
2   {
3    public static void main(String[] args)
4    {
5      double[] Wohnhaus = new double[5];
6        Wohnhaus[0] = 40;
7        Wohnhaus[1] = 60;
8        Wohnhaus[2] = 25;
```

[0] Der Begriff Feld wird doppeldeutig verwendet. Zum einen für das Array und zum anderen für die einzelnen Elemente des Arrays. Normalerweise ist der Kontext aber so eindeutig, dass es zu keiner Verwechslung kommen sollte.

```
9         Wohnhaus[3] = 60;
10        Wohnhaus[4] = 30;
11    }
12 }
```

In Zeile 5 wird das Array deklariert. Dazu werden eckige Klammern [] verwendet. Die Anzahl der Elemente, die das Array haben soll, wird auf 5 festgelegt und lässt sich nicht mehr ändern. Auf die einzelnen Elemente der Liste kann dann mit Hilfe eines Index zugegriffen werden. Der Index ist dabei die Feldnummer, die von 0 bis Anzahl der Elemente minus 1 reicht. Ein häufiger Fehler ist es, auf ein Feld zuzugreifen, das gar nicht existiert. Z.B. ist Wohnhaus[5] undefiniert. Zwar besitzt das Array 5 Elemente, das letzte Feld hat aber den Index 4 und nicht 5! Immer wenn das Programm mit der Fehlermeldung:
`java.lang.ArrayIndexOutOfBoundsException` abbricht, hat man das nicht berücksichtigt!

MERKE

Die Zählweise eines Arrays beginnt bei 0, d.h., bei einem Array mit n-Elementen läuft der Index von 0 bis n-1.

Man könnte prinzipiell immer auf Arrays verzichten und stattdessen für jeden Wert eine eigene Variable anlegen. Das ist allerdings nicht sehr elegant und wird schnell unübersichtlich. Außerdem werden Arrays natürlich in vielen Bibliotheken verwendet, sodass man um sie nicht herum kommt. Für die Deklaration eines Array kann neben der Schreibweise

```
double[] Wohnhaus = new double[5];
```

auch die folgende Schreibweise Verwendung finden:

```
double Wohnhaus[] = new double[5];
```

Es kommt also nicht auf die Reihenfolge der Klammern an. Die beiden obigen Beispiele sind eine Zusammenfassung von

```
double[] Wohnhaus;
Wohnhaus = new double[5];
```

Dies sind die Möglichkeiten ein Array in Java zu erstellen.
Das nächste Beispiel zeigt, warum es bequem ist, Arrays anstatt einzelnen Variablen zu verwenden. Außerdem wird gezeigt, wie auf die einzelnen Werte des Arrays zugegriffen wird.

Listing 2.5. Beispiel221a.java

```
1  public class Beispiel221a {
2     public static void main(String[] args)
3     {
4  // Ein Array mit 1000 Elementen erstellen
5       int[] daten = new int[1000];
```

```
6  // Das Array initialisieren mit daten[i] wird der i-te Wert
       gesetzt.
7  // i darf nur von 0 bis 999 (das sind 1000 Werte) laufen!!
8         for(int i = 0; i < 1000; i++ )
9               daten[i] = i;
10     // Ausgabe
11     for(int i = 0; i < 1000; i++ ){
12         int Wert = daten[i];
13         System.out.println("Wert: " + Wert );
14   }
15  }
16 }
```

Ausgabe Beispiel221a

```
Wert: 1
Wert: 2
..... (Ausgabe gekürzt)
Wert: 1000
```

Beispiel221a erstellt ein Array mit 1000 Werten (Zeile 5) und initialisiert es mit den ersten natürlichen Zahlen (Zeile 8 und 9). Anschließend wird der Inhalt des Arrays auf der Konsole ausgegeben. Das ist wesentlich einfacher als 1000 einzelne Variablen zu deklarieren. In Zeile 9 wird gezeigt, wie man einzelne Werte in ein Array schreibt und in Zeile 13, wie jeweils der i-te Wert gelesen und in der variablen 'Wert' gespeichert wird. Eine Zuweisung sieht demnach so aus:

```
// Der fünften Stelle des Arrays 'meinArray'
// wird der Wert 20 zugewiesen
meinArray[4] = 20;
```

Ein Wert wird folgendermaßen aus einem Array gelesen:

```
// Der Wert der sich an der fünften Stelle des Array
// 'meinArray' befindet,
// wird in der Variablen value gespeichert.
int value = meinArray[4];
```

Es soll nicht unerwähnt bleiben, dass Beispiel221a einen Schönheitsfehler hat. Wenn man die Anzahl der Elemente später verkleinern will z.B. von 1000 auf 500, indem man Zeile 5 entsprechend abändert, dann wird das Programm abstürzen (bitte ausprobieren). Denn in beiden Schleifen wird auf Feldelemente zugegriffen (Index $>$499), die dann nicht mehr existieren. Eine Möglichkeit, dieses zu umgehen, ist die Verwendung einer Variablen, die die Feldgröße definiert. Zum Beispiel

```
int arraySize = 10;
int[] daten = new int[ arraySize ];
```

und diese Variable dann auch in den for-Schleifen zu verwenden. Die Anzahl der Elemente kann auch mit dem für Arrays zur Verfügung stehenden Attribut length ermittelt werden. Die for-Schleife würde dann, z.B., folgendermaßen aussehen:

```
for(int i = 0; i < daten.length; i++ )
    daten[i] = i;
```

Auch diese Methode ist sicher gegenüber Änderungen der Arraylänge.

2.2.2 Arrays initialisieren, kopieren und ausgeben

Die vorherigen Beispiele zeigen eine klassische Methode auf, um Arrays zu initialisieren und zwar durch eine for-Schleife. Das nächste Programm legt ein Array 'lsDaten' der Länge 10 an und initialisiert dann in der darauf folgenden for-Schleife alle Elemente mit 0.

Listing 2.6. Beispiel222.java

```
1 public class Beispiel222
2   {
3    public static void main(String[] args)
4    {
5      double[] lsDaten = new double[10];
6        for(int i = 0; i < 10; i++ )
7              lsDaten[i] = 0;
8   }
9  }
```

Grundsätzlich sollte ein Array immer initialisiert werden. Neben der Verwendung einer for-Schleife kann man auch die fill-Methode der Klasse Arrays benutzen, wie das nächste Beispiel zeigt.

Listing 2.7. Beispiel222a.java

```
1 public class Beispiel222a
2   {
3    public static void main(String[] args)
4    {
5      double[] lsDaten = new double[10];
6       Arrays.fill( lsDaten,0.0);
7    }
8   }
```

Die Methode Arrays.fill(Name, Wert) (Zeile 6) erwartet als ersten Parameter den Namen des Arrays und als zweiten Parameter den zu initialisierenden Wert. Die Methode steht für alle Datentypen zur Verfügung. Manchmal, wenn das Array nur wenige Elemente hat, initialisiert man ein Array auch gleich beim erstellen, indem man die einzelnen Elemente in geschweiften Klammern schreibt.

```
int[] wuerfel = {1,2,3,4,5,6};
boolean[] glueck = {true,false};
```

Das Ausgeben eines Arrays erfolgt ebenfalls meist durch eine for-Schleife, die die einzelnen Werte des Arrays durchläuft und ausgibt. Ferner stellt die Klasse Arrays auch hier eine Methode Arrays.toString(Name) zur Verfügung, die sich zu Prüfzwecken (Debugging) sehr gut eignet.

Listing 2.8. Beispiel222b.java

```
1  public class Beispiel222b
2   {
3    public static void main(String[] args)
4    {
5      int[] lsDaten = new int[10];
6       Arrays.fill( lsDaten,22);
7       System.out.println(Arrays.toString( lsDaten ));
8    }
9   }
```

Ausgabe Beispiel 9. Mit der Arrays.toString-Methode kann man auf einfache Weise den Inhalt auf der Konsole ausgeben.

```
[22, 22, 22, 22, 22, 22, 22, 22, 22, 22]
```

Früher oder später könnte es vorkommen, dass man ein Array kopieren möchte. Das nächste Beispiel zeigt, wie man es **nicht** machen sollte.

Listing 2.9. Beispiel222c.java

```
1  import java.util.Arrays;
2  public class Beispiel222c {
3    public static void main(String[] args) {
4          int[] lsDaten = new int[10];
5          Arrays.fill(lsDaten, 22);
6          int[] lsKopie = lsDaten;
7          System.out.println("Array: lsDaten: " + Arrays.
               toString(lsDaten));
8          System.out.println("Array: lsKopie: " + Arrays.
               toString(lsDaten));
9          lsKopie[4]=1000;
10         System.out.println("Array: lsDaten: " + Arrays.
               toString(lsDaten));
11         System.out.println("Array: lsKopie: " + Arrays.
               toString(lsDaten));
12   }
13 }
```

Ausgabe:

```
Array: lsDaten:[22,22,22,22, 22, 22, 22, 22, 22, 22]
Array: lsKopie:[22,22,22,22, 22, 22, 22, 22, 22, 22]
Array: lsDaten:[22,22,22,22, 1000, 22, 22, 22, 22, 22]
Array: lsKopie:[22,22,22,22, 1000, 22, 22, 22, 22, 22]
```

Die ersten beiden Zeilen der Ausgabe zeigen das “Orginal Array” `lsDaten` und die Kopie. Alles scheint in Ordnung. Anschließend wird mit `lsKopie[4]=1000;` der 5. Wert des Arrays von 22 auf 1000 geändert. Nach erneuerte Ausgabe beider Arrays zeigt sich das Problem. Obwohl wir einen Wert im Array lsKopie geändert haben, hat sich auch das Array lsDaten geändert. Das ist nicht das, was man von

einer Kopie erwartet. Ein Versuch mit der Methode `Arrays.clone()` führt zum selben Ergebnis und ist damit auch nicht geeignet, ein Array zu kopieren. Auf diese Weise haben wir lediglich eine Referenz auf das “orginale” Array hergestellt aber keine Kopie.
Im Folgenden stellen wir zwei Methoden vor, mit deren Hilfe man eine “echte” Kopie eines Arrays erstellen kann.

Listing 2.10. Beispiel222d.java

```
1  import java.util.Arrays;
2
3  public class Beispiel222d {
4
5   public static void main(String[] args) {
6
7          int[] lsDaten = new int[10];
8          Arrays.fill(lsDaten, 22);
9          // Methode 1: copy mit for-Schleife
10         int[] lsKopie1 = new int[lsDaten.length];
11         for(int i = 0; i < lsKopie1.length; i++)
12           lsKopie1[i] = lsDaten[i];
13         //Test 1
14         lsKopie1[4]=1000;
15         System.out.println("Methode1\nOrginal:" + Arrays.
               toString(lsDaten));
16         System.out.println("Veraenderte Kopie: " + Arrays.
               toString(lsKopie1));
17
18         //Methode 2: copy mit hilfe der Arrays Klasse
19         int[] lsKopie2 = Arrays.copyOf(lsDaten,lsDaten.length)
               ;
20         // Test 2
21         lsKopie2[4]=1000;
22         System.out.println("Methode2\nOriginal: " + Arrays.
               toString(lsDaten));
23         System.out.println("Veraenderte Kopie: " + Arrays.
               toString(lsKopie2));
24
25  }
26
27 }
```

Die erste Methode erzeugt eine Kopie mit Hilfe einer for-Schleife. Dabei wird der Inhalt jedes Feldes in das neue Array gespeichert. Die zweite Methode benutzt die Methode `Arrays.copyof( Name, Länge)`. Die Methode erwartet als ersten Parameter den Namen des Arrays der als Kopiervorlage dient und als zweiten Parameter die Anzahl der zu kopierenden Elemente. Die Methode liefert dann ein Array zurück (hier lsKopie2).

2.2.3 Arrays und Methoden

Arrays können auch als Parameter an eine Methode übergeben werden oder Rückgabewert einer Methode sein. Z.B. erwartet die write-Methode des IO-Warriors ein Integer-Array als Parameter und die read-Methode liefert ein Integer-Array zurück. Die folgende Methode berechnet die Summe des übergebenen Arrays.

```
double arraySum( int[] array ){
      double sum = 0;
      for(int i = 0; i< array.length; i++)
            sum += array[i];
      return sum;
      }
```

Damit ein Array ausgegeben wird, erzeugen wir es im Methodenrumpf. Die folgende Methode erwartet einen Integer von 1 bis 10 und liefert eine kleine Multiplikationstabelle zurück.

```
int[]  arraySum( int zahl ){
      int[] ergebnis = new int[10];

      if(zahl < 1 || zahl > 10 )
          return null;

      for(int i = 1; i < 11; i++)
            ergebnis[i-1] = i*zahl;
       return ergebnis;
       }
```

MERKE

Um ein Array zurückzugeben, wird es im Methodenrumpf erstellt. In der return-Anweisung wird der Name des Arrays ohne Klammern angegeben.

2.2.4 Aufgaben

Aufgabe 1

Entferne die Schönheitsfehler aus Beispiel221a

Aufgabe 2

Schreibe eine Methode, die als Parameter ein Integer-Array erwartet und den Mittelwert zurück gibt. Benutze diese Methode, um aus den Wohnhaus-Array die durchschnittliche Größe eines Raumes zu berechnen.

Aufgabe 3

Im folgenden Programm sind zwei Fehler versteckt. Finde und korrigiere sie.

```
public class Aufgabe3
  {
   public static void main(String[] args)
   {
     double[] Einkommen = new double[5];
     Wohnhaus[1] = 5;
     Wohnhaus[2] = 10;
     Wohnhaus[3] = 15;
     Wohnhaus[4] = 20;
     Wohnhaus[5] = 25;

  for(int i = 0; i <= Einkommen.length; i++)
    System.out.println( "Einkommen: " + Einkommen[i] );
  }
}
```

2.3 Logische Operatoren

Die logischen Operatoren (UND, ODER und NICHT) gehören zu den wichtigsten Sprachelementen in Java. Beispiele mit logischen Operatoren sind:

Operator	Bezeichnung	Erklärung
&&	UND	Ist nur dann wahr, wenn beide Ausrdrücke wahr sind
||	ODER	Ist nur dann falsch, wenn beide Ausrdrücke falsch sind
!	NICHT	Invertiert die Aussage. Wenn **a** richtig ist (true), dann ist **!a** falsch (false). Und wenn **a** falsch (false) ist, dann ist **!a** wahr (true).

Tabelle 2.7.

```
boolean a = false;
boolean b = false;
boolean c = true;

 if( a || b )                    // ergibt false
 if( !a || b)                    // ergibt true
 if( ( !a && c ) || b )          // ergibt true
 if( a || b || c )               // ergibt true
 if( (!b && c ) && ( c || a ) ) // ergibt true
```

Mit einem kleinen Java-Programm kann die Wahrheitstafel (engl. truetable) für die logischen Operatoren && bzw. || erstellt werden. Dabei wird ein Array vom Typ boolean verwendet.

Listing 2.11. TrueTable23.java

```
1  public class TrueTable23 {
2
3    public static void main(String[] args) {
4                 boolean values[] = {true,false};
5   System.out.println("a" + "\t" + "b" + "\t" + "(a&&b)" + "\t"
        + "(a||b)");
6
7   for (int i = 0; i < values.length; i++) {
8     boolean a = values[i];
9     for (int j = 0; j < values.length; j++) {
10     boolean b = values[j];
11     System.out.println(a + "\t" + b + "\t" + (a&&b) + "\t" + (
           a||b));
12    }
13   }
14  }
15 }
```

Das Programm erzeugt die Ausgabe:

```
a      b      (a&&b)  (a||b)
true   true   true    true
true   false  false   true
false  true   false   true
false  false  false   false
```

Vergleichen Sie die Ausgabe mit den Erklärungen in der Tabelle.
Bei logischen Ausdrücken fehlen häufig die Vergleichsoperatoren auf der rechten Seite. Java wertet jeden Ausdruck aus. Wenn der Vergleichsoperator fehlt, wird automatisch auf true geprüft. Damit ergibt sich, dass die folgenden Ausdrücke gleichwertig sind.

die verkürzte Schreibweise bei Vergleichsoperatoren		
`if(a)`	ist gleichwertig mit	`if(a == true)`
`if(!b)`		`if(b == false)`
`while(c)`		`while(c == true)`

Tabelle 2.8. Beispiele Vergleichsoperatoren

2.4 Bits und Bytes

Motivation

Das Arbeiten mit Bits und Bytes ist für MSR fundamental. Für “normales” Programmieren wird es nicht so häufig verwendet, weswegen es auch vielen Java-Programmierern nicht immer so geläufig ist. Für das Lesen und Schreiben jedes einzelnen Datenpaketes des IO-Warrior ist ein gutes Verständnis dieses Abschnitts aber notwendig.
Warum ist das so?
Wenn wir die an Pin 1 angeschlossene Leuchtdiode[1] zum Leuchten bringen wollen (siehe Abbildung), dann muss der Pin 1 auf logisch 1 (true, high, 5 Volt) gesetzt werden. Wollen wir die Leuchtdiode wieder ausschalten, schreiben wir eine logische 0 (false, low, 0 Volt). Nur leider lassen sich einzelne Pins nicht auf diese Weise ansteuern. Zwar kann man das gewünschte Verhalten softwareseitig nachbilden, aber eben nur dann, wenn man weiß, wie man mit den Bits umzugehen hat. Den digitalen Wert des Pins, der die Zustände 0 oder 1 annehmen kann, bezeichnet man als Bit (binary digit).
Ein Bit kann immer genau zwischen zwei Zuständen unterscheiden, so wie es die

[1] Im konkreten Fall muss aber noch ein Widerstand dazwischengeschaltet werden, damit die LED kein schaden nimmt (BAND II).

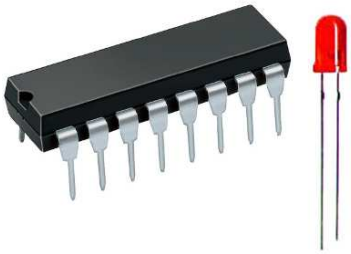


Abbildung 2.4. Die einzelnen “Beinchen” (Pins) des Bausteins können mit einer Leuchtdiode verbunden werden. Wenn man den Pin des Bausteins auf 5V (1) setzt, leuchtet die Diode. Setzt man den Pin auf 0V (0), dann leuchtet sie nicht.

Eingabe/Ausgabe-Pins auch können. Eine Leuchtdiode kann entweder leuchten oder nicht (repräsentiert also zwei Zustände) und dient damit als anschauliches Beispiel für den Begriff Bit.

Pin 1	Bedeutung
1	Leuchtdiode an, High oder Bit = 1
0	Leuchtdiode aus, Low oder Bit = 0

Tabelle 2.9. 1 Pin repräsentiert 2 mögliche Zustände

MERKE

Ein Bit repräsentiert die kleinste Informationseinheit und kann nur zwischen zwei Zuständen unterscheiden.

Da man die zwei Zustände beim Namen nennen muss, nennt man sie gewöhnlich 1 und 0. Der Tabelle kann man weitere Bezeichnungen entnehmen. Zurück zur unse-

Bit	Umgangssprachliche Bezeichnungen
1	an (on), wahr (true), ja (yes), high, 5V, ...
0	aus (off), falsch (false), nein (no), low, 0V, ...

Tabelle 2.10. Das Bit als kleinste Informationseinheit hat viele Namen.

rem Leuchtdiodenbeispiel. Nehmen wir den zweiten Pin dazu, ergibt sich folgende Tabelle:

Pin 2	Pin 1	Bedeutung
0	0	keine Lampe an
1	1	beide Lampe an
1	0	zweite an, erste aus
0	1	zweite aus, erste an

Tabelle 2.11. Mit zwei Pins kann es vier verschiedene Zustände geben.

Mit zwei Pins können demnach vier verschiedene Kombinationen gebildet werden. Für den IO-Warrior gilt, dass jeweils acht Pins (oder 8 Bits) eine Gruppe bilden. Acht Bit nennt man ein Byte. Z.B. könnte 01011100 das Byte darstellen, bei dem die dritte, vierte, fünfte und siebente Leuchtdiode an ist.

MERKE

Eine Abfolge von 8 Bits bezeichnet man als Byte.

Beim IO-Warrior24 können immer nur alle 8 Bits (1 Byte) gleichzeitig gelesen oder geschrieben werden.
Um beim Beispiel 01011100 zu bleiben, müssen wir als erstes diese Dualzahl in eine Dezimalzahl umwandeln. Diese Dezimalzahl können wir dann an den IO-Warrior schicken, der die I/O-Pins entsprechend der binären Werte einzeln auf 0 (Low) oder 1 (High) setzt. Da $01011100_2 = 92_{10}$ ist, bedeutet das, dass wir den Integerwert 92 an das Gerät senden müssen.
Betrachten wir nun den umgekehrten Fall, d. h., wir möchten nun wissen, ob der Pin 4 auf 1 oder 0 gesetzt ist. In diesem Fall müssen Daten vom Gerät gelesen werden. Diese Daten werden in Form einer Integerzahl geliefert. Dieser Integerwert beinhaltet den Zustand aller 8 Bits (1 Byte). Es ist deshalb wichtig die einzelnen Bits eines Integer-Wertes auch wieder extrahieren zu können, um etwas über den Zustand der Pins in Erfahrung zu bringen.
Mit anderen Worten, wenn Sie mit den IO-Warrior24 arbeiten wollen, müssen Sie wissen, wie man in Java mit Bitoperatoren arbeitet.

Technische Anmerkung

Daten an den IO-Warrior werden immer byteweise gesendet oder empfangen. Wir erfahren also immer gleichzeitig den Zustand von 8 I/O-Pins. Allerdings besitzt der I/O-Warrior24 16 I/O Pins, also 2·8(Bits) = 2 Byte. Alle 16 I/O-Pins werden immer gleichzeitig beschrieben (oder gelesen). Mehr darüber im Kapitel IO-Warrior.
Die Java-Bibliothek von Code Mercenaries ermöglicht das Senden/Empfangen von Daten mit Hilfe von Integer-Werten. Wir wissen bereits, dass es wichtig ist, zu wissen, wie sich die einzelnen Bits eines Integer-Wertes manipulieren lassen. Überlegen wir zuerst in welchen Zahlenbereich die zu sendenden/empfangenden Daten liegen.
Dazu machen wir folgende Überlegung:
1 Bit bietet 2 Möglichkeiten

2 Bits bieten $(2 \cdot 2 = 2^2)$ 4 Möglichkeiten
3 Bits bieten $(2 \cdot 2 \cdot 2 = 2^3)$ 8 Möglichkeiten
... u.s.w.
8 Bits bieten $(2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 2^8)$ 256 Möglichkeiten.

Das heißt ein Byte kann 256 verschiedene Zustände speichern. Da die Zählweise bei 0 beginnt erhalten wir einen Zahlenbereich von 0 - 255 für ein Byte.

MERKE

Jeder Integerwert, den wir an das Gerät senden (oder von ihm empfangen) liegt zwischen 0 und 255.

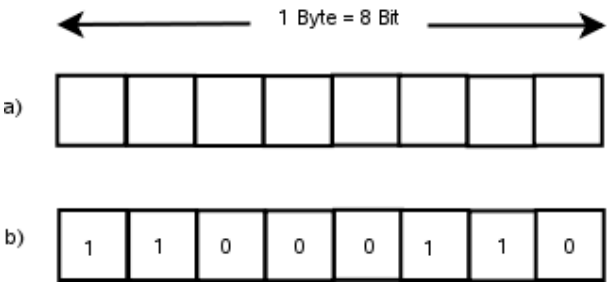


Abbildung 2.5. Jeweils 8 Bits bilden ein Byte, auch die oben abgebildeten 8 Kästen (a) repräsentieren damit 1 Byte. In jedem der oben abgebildeten Kästchen kann eine 1 oder eine 0 stehen. Insgesamt gibt es dann 256 verschiedene Bytes. Ein Beispiel zeigt Bild b).

Methode	Bedeutung
`boolean isBit(int pos, int value)`	Prüft, ob das Bit mit der Position pos gesetzt ist. Falls ja ist der Rückgabewert true, ansonsten false.
`void clearBit(int pos, int value)`	Setzt das Bit mit der Position pos auf 0.
`void setBit(int pos, int value)`	Setzt das Bit mit der Position pos auf 1.
`int flipBit(int pos, int value)`	Setzt das Bit auf 0, wenn es zuvor auf 1 war und umgekehrt.

Tabelle 2.12. nützliche Bits und Bytes Methoden

Im nächsten Abschnitt werden die in Java vorhandenen Bit-Operatoren behandelt, mit deren Hilfe wir folgende Methoden für die Bit-Manipulation programmieren können. Diese Methoden sind recht nützlich, wenn es darum geht, etwas über die Zustände der Pins zu erfahren. Allen Methoden ist gemein, dass der erste Parame-

ter die Position angibt, deren Bit wir verändern (oder lesen) möchten. Der zweite Parameter value ist der Integerwert, der geschrieben/gelesen werden soll.

2.4.1 Bit-Operatoren

In Java stehen die folgenden vier Bitoperatoren zur Verfügung:

Operator	Bezeichnung	Bedeutung
&	bitweises UND (AND)	a & b verknüpft jedes Bit einzeln mit UND
|	bitweises ODER (OR)	a | b verknüpft jedes Bit einzeln mit ODER
∼	bitweises NICHT (NOT, Komplement)	∼ invertiert jedes Bit einzeln mit NICHT
∧	bitweises exklusives ODER (Xor)	a ∧ b verknüpft jedes Bit einzeln mit Xor

Tabelle 2.13. Java Bit-Operatoren

Diese Operatoren arbeiten immer mit den einzelnen Bits einer Zahl. Der übliche Weg die Arbeitsweise solcher Operatoren zu veranschaulichen ist, die zu verknüpfenden Zahlen im Binärsystem darzustellen, um die Umrechnung dann bitweise nachvollziehen zu können.

2.4.2 Bitweise (&) UND

Die Wahrheitstafel der & (UND)-Verknüpfung lautet:

Bit-1	Bit-2	Bit-1 & Bit-2
0	0	0
0	1	0
1	0	0
1	1	1

Tabelle 2.14. Wahrheitstafel UND-Verknüpfung

HINWEIS

Die UND-Verknüpfung ist nur dann wahr, wenn beide Bits auf 1 gesetzt sind.

```
int a = 11;
int b = 7;
int c = a & b;
```

```
System.out.println("a & b = " + c +
                   "\t(Binär): " +Integer.toBinaryString(c));
```

Die folgende Ausgabe wird erzeugt:

```
a & b = 3  (Binär): 11
```

Dieses Ergebnis ist erst einmal unverständlich. Stellen wir nun aber die gegebenen Zahlen als Dualzahlen dar und wenden für jede Stelle (einzelnes Bit) die &-Operation an, dann wird das Ergebnis verständlich. Für jedes einzelne Bit wurde

	0 0 0 0 1 0 1 1	$= 11_{10} = a$
&	0 0 0 0 0 1 1 1	$= 7_{10} = b$
	0 0 0 0 0 0 1 1	$= 3_{10} = c$

eine UND-Verknüpfung vollzogen. Das Ergebnis ist die Binärzahl 11_2 dessen Dezimalwert 3 beträgt.

BEISPIEL

Was ergibt 21 & 14?

LÖSUNG

In der Binärdarstellung kann man das Ergebnis ablesen.

	0 0 0 1 0 1 0 1	$= 21_{10} = a$
&	0 0 0 0 1 1 1 0	$= 14_{10} = b$
	0 0 0 0 0 1 0 0	$= 4_{10} = c$

Das Ergebnis muss wieder ins Dezimalsystem zurückgerechnet werden, es ergibt sich 21 & 14 = 4.

Betrachten wir die UND-Verknüpfung einer Zahl mit sich selbst, z.B. 8 & 8, dann ergibt sich wieder 8. Das gilt für alle Zahlen die mit sich selbst UND-verknüpft werden, d.h. a & a = a.
Betrachten wir hingegen eine Verknüpfung einer beliebigen Zahl mit 0, z.B. 12 & 0, dann muss sich zwangsweise 0 ergeben, da die UND-Verknüpfung nur dann 1 ist, wenn beide Bits 1 sind, bei der Zahl 0 aber alle Bits identisch 0 sind.
Das bitweise UND kann man nun benutzen, um herauszufinden, ob ein bestimmtes Bit einer Zahl gesetzt ist oder nicht. Betrachten wir dazu folgende Tabelle. Um

	0 1 0 1 1 1 0 1	$= 93_{10}$
&	0 0 0 0 **1** 0 0 0	Testsequenz, auch als **Bitmaske** bezeichnet
	0 0 0 0 1 0 0 0	$\neq 0$, also ist das Bit gesetzt

herauszufinden ob bei der Dezimalzahl 93 das 4. Bit gesetzt ist, wird eine Bitmaske

erstellt, die an der 4. Stelle eine 1 hat und ansonsten nur aus Nullen besteht. Diese Maske wird dann mit der zu untersuchenden Zahl verknüpft. Ist das Ergebnis ungleich null, dann ist das Bit gesetzt, ansonsten ist das Bit nicht gesetzt.
Diese Vorgehensweise nennt man bitweise Maskierung. Die Testsequenz bezeichnet man als Bitmaske.

HINWEIS

Mit der bitweisen Maskierung können wir herausfinden, ob ein Bit an einer bestimmten Stelle einer Zahl gesetzt ist oder nicht.

Um Bitmasken bequem zu generieren, benötigen wir noch bitweise Verschiebungen, die weiter unten im Kapitel folgen. Als nächstes wird die bitweise ODER-Verknüpfung betrachtet.

2.4.3 Bitweise (|) ODER

Die Wahrheitstafel der bitweisen ODER-Verknüpfung lautet:

Bit-1	Bit-2	Bit-1 & Bit-2
0	0	0
0	1	1
1	0	1
1	1	1

Tabelle 2.15. Wahrheitstafel bitweise ODER-Verknüpfung

HINWEIS

Die ODER-Verknüpfung ist nur dann falsch, wenn beide Bits auf 0 gesetzt sind.

Das folgende Codefragment ergibt für c einen Wert von 15.

```
int a = 11;
int b = 7;
int c = a | b;
System.out.println("a | b = " + c + "\t(Binär): "
          + Integer.toBinaryString(c));
```

Ausgabe

```
a | b = 15 (Binär): 1111
```

Dieser Wert ist erst einmal unverständlich. Stellen wir nun aber die gegebenen Zahlen als Dualzahlen dar und werten für jede Stelle (einzelnes Bit) die |-Operation

	0	0	0	0	1	0	1	1	$= 11_{10} = a$
|	0	0	0	0	0	1	1	1	$= 7_{10} = b$
	0	0	0	0	1	1	1	1	$= 15_{10} = c$

an, dann wird das Ergebnis verständlich. Für jedes einzelne Bit wurde eine ODER-Verknüpfung vollzogen. Das Ergebnis ist die Binärzahl 1111_2 dessen Dezimalwert 15 beträgt.

2.4.4 Bitweise ($\sim$) NICHT

Die Wahrheitstafel der bitweise NICHT-Verknüpfung lautet:

Bit-1	**$\sim$Bit-1**
0	1
1	0

Tabelle 2.16. Wahrheitstafel bitweise NICHT-Verknüpfung

HINWEIS
Die NICHT-Verknüpfung invertiert (vertauscht) jedes Bit.

Die unterstehende Tabelle zeigt, wie sich die einzelnen Bits einer Variablen 'a' bei der NICHT- Verknüpfung ändern. Für jedes einzelne Bit wurde eine Verknüpfung

	0	0	0	0	1	0	1	1	$= a$
$\sim$	1	1	1	1	0	1	0	0	$= \sim a$

vollzogen.

Bits löschen

Für uns interessant ist die Möglichkeit das bitweise NICHT mit einer UND-Maske zu kombinieren, denn dadurch ist es möglich, Bits an einer bestimmten Stelle zu löschen.
Angenommen, wir haben eine Dualzahl `1011011` und wollten das Bit an der 5. Stelle löschen (auf 0 setzen), sodass die Dualzahl `1001011` entsteht.
Das können wir erreichen, indem wir folgendes tun:

1. Erstellen einer UND-Maske, die an der 5. Stelle eine 1 und ansonsten nur aus Nullen besteht.

2. Invertieren dieser UND-Maske, d.h., die NICHT-Operation auf die UND-Maske anwenden.
3. Wende die UND-Verknüpfung auf die Zahl und die invertierte UND-Maske an. Das Ergebnis ist die ursprüngliche Zahl, die nun allerdings an der 5. Stelle eine 0 stehen hat.

	0	1	0	1	1	0	1	1 $= 91_{10} = a$
&	1	1	1	0	1	1	1	1 $= 239_{10} = b$
	0	1	0	0	1	0	1	1 $= 75_{10} = c$

BEISPIEL

Gegeben ist die Zahl 89. Löschen Sie das Bit an der Position 4?
LÖSUNG
Dazu erstellen wir die Bit-Maske 00001000 und invertieren diese: $\sim$(00001000) = 11110111=mask.
Jetzt bilden wir das Bitweise & von 89 und der Bitmaske: mask.

	0	1	0	1	1	0	0	1 $= 89_{10}$
&	1	1	1	1	0	1	1	1 $= 247_{10}$ =mask **(Bitmaske)**
	0	1	0	1	0	0	0	1 $= 81_{10}$

Damit haben wir von der ursprünglichen Zahl das 4. Bit gelöscht. Der Dezimalwert beträgt nun 81.
Das Codefragment, das die Bitfolge 1010001 in Java erzeugt lautet:

int c = 89 & 247;
System.out.println(Integer.toBinaryString(c));

In Java sind Integerzahlen 32 Bit (das sind 4 Byte) lang. Die Bitmaske 247 könnte man auch mit *int mask = $\sim$8;* erzeugen.

2.4.5 Bitweise ($\wedge$) EXCLUSIVE ODER

Die Wahrheitstafel der bitweise EXKLUSIV-ODER-Verknüpfung lautet:

HINWEIS
Die Xor-Verknüpfung ist nur dann wahr (1, true), wenn lediglich eines der beiden Bits auf 1 gesetzt ist.

Das folgende Codefragment ergibt für c einen Wert von 13.

Bit-1	Bit-2	Bit-1 ∧ Bit-2
0	0	0
0	1	1
1	0	1
1	1	0

Tabelle 2.17. Wahrheitstafel bitweise Xor-Verknüpfung

Listing 2.12. Main245.java

```
1  public class Main245 {
2
3   public static void main(String[] args) {
4
5    int a = 8;
6    int b = 5;
7
8    System.out.println("Dezimal: " + a + "\tBinaer: "
9                                    + Integer.toBinaryString(a));
10   System.out.println("Dezimal: " + b + "\tBinaer: "
11                                   + Integer.toBinaryString(b));
12
13   int c = a ^ b;
14   System.out.println("a ^ b = " + c + "\t(Binaer): "
15                                   + Integer.toBinaryString(c));
16
17         }
18 }
```

Ausgabe:

```
Dezimal: 8 Binaer: 1000
Dezimal: 5 Binaer: 101
a ^ b = 13 (Binaer): 1101
```

Auch dieses Ergebnis wird erst auf “binärer Ebene” verständlich. Das folgende Fragment erzeugt die Wahrheitstafel (Truetable) der Xor-Operation

```
System.out.println(1^0);
System.out.println(0^1);
System.out.println(0^0);
System.out.println(1^1);
```

Die Ausgabe ist:

```
1
1
0
1
```

HINWEIS

Das bitweise Xor ($\wedge$) können wir verwenden, wenn wir ein Bit invertieren wollen. D.h., ist das Bit auf 0 gesetzt, setzen wir es auf 1 und ist es auf 1 gesetzt, dann setzen wir es auf 0.

BEISPIEL I

Flippen (invertieren) Sie das Bit an der 3. Stelle von der Zahl 89.
LÖSUNG

	0	1	0	1	1	0	0	1	$= 89_{10}$
$\wedge$	0	0	0	0	0	1	0	0	= Bitmaske für die 3. Stelle
	0	1	0	1	1	**1**	0	1	Bit an der 3.Stelle ist von 0 auf 1 “geflippt”.

Das Bit wurde von ursprünglich 0 auf jetzt 1 geflippt.

BEISPIEL II

Flippen (invertieren) Sie das Bit an der 4. Stelle von der Zahl 89.
LÖSUNG

	0	1	0	1	1	0	0	1	$= 89_{10}$
$\wedge$	0	0	0	0	1	0	0	0	= Bitmaske für die 4. Stelle
	0	1	0	1	**0**	0	0	1	Bit an der 4.Stelle ist von 1 auf 0 “geflippt”.

Das Bit wurde von ursprünglich 1 auf jetzt 0 geflippt.

Als Anwendung könnte man sich z.B. vorstellen, dass an den IO-Warrior angeschlossene Leuchtdioden in bestimmten Abständen blinken sollen.

2.4.6 Bitweise Verschiebungen <<,>>,>>>

In Java gibt es drei Verschiebungsoperatoren:

$<<$ verschiebt nach links
$>>$ und $>>>$ verschieben nach rechts.

Falls sie sich fragen, was denn da verschoben wird. Nun ja, es sind die Bits.

Der << Operator (Left Shift Operator)

Angenommen wir haben folgende Dualzahlen.

```
Zahl1: 10111011
Zahl2: 11011000
```

Die Zahl2 kann man sich aus der Zahl1 entstanden denken, wenn man alle Bits um drei Stellen nach links schiebt (siehe Abbildung). Die links stehenden Bits gehen

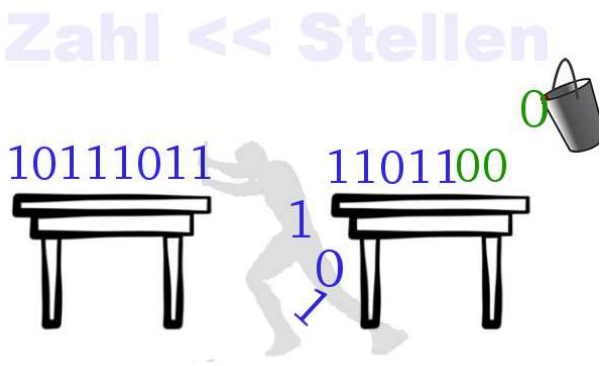


Abbildung 2.6. Bei der Linksverschiebung von Bits (<<), werden alle Bits um soviele Stellen nach links verschoben, wie hinter dem << Operator steht (hier 3). Die herausgeschobenen Bits gehen dabei verloren. Links wird immer mit Nullen aufgefüllt.

dabei verloren (fallen runter). Rechts wird immer mit “frischen” Nullen aufgefüllt.

Die Dualzahl Zahl1 10111011 lautet im Dezimalsystem 187.
Die Dualzahl Zahl2 11011000 lautet im Dezimalsystem 216.

Das folgende Java Programm bearbeitet obiges Problem:

Listing 2.13. LeftShift.java

```
1 public class LeftShift {
2
3  public static void main(String[] args) {
4   int i = 187 << 3;
5   System.out.println("Dezimal: " + i + " Binaer: "
6         + Integer.toBinaryString(i));
7
8   }
9 }
```

Programmausgabe:

```
Dezimal: 1496 Binaer: 10111011000
```

Um in Java Bits nach links zu verschieben, schreiben wir a << b.
Wobei a die Zahl ist, deren Bits verschoben werden sollen und b die Zahl angibt,

um wie viel Stellen das geschehen soll. In Zeile 4 ist `a = 187` und `b = 3`. Das Ergebnis dieser Operation wird in der Variablen `i` gespeichert.

HINWEIS

Mit Hilfe des << (Links-Shift) Operators kann man auf einfache Weise Bitmasken erstellen.

Das folgende Fragment erzeugt eine Bitmaske in der nur die 4. Stelle gesetzt ist.

```
// erzeugt die Bitmaske: 1000
int mask = 1 << 3;
```

Die 1 im Dezimalsystem hat im Dualsystem nur an der ersten Stelle eine 1. Eine Stelle plus drei weitere Stellen nach links, verschiebt die 1 auf die vierte Stelle (Die Nullen links fallen immer runter, aufgefüllt wird rechts immer mit Nullen).

Der >> Operator (Right Shift Operator)

Der >> Operator (auch Arithmetischer Right-Shift-Operator) verschiebt die Bits einer Zahl nach rechts. Rechts werden die Bits "herausgeschoben" und gehen dabei verloren. Die von links hinzukommenden Bits werden mit 0 oder 1 aufgefüllt, je nachdem ob das Bit an der linken Seite eine 0 oder 1 war. In Java bestimmt das Bit, das ganz links steht (den höchsten Wert repräsentiert, auch msb (s. Abschnitt 2.4.8)) das Vorzeichen der Zahl. Ist es 0, dann handelt es sich um eine positive Zahl. Ist das Bit 1, dann ist es eine negative Zahl.

Beispiel für positive Zahlen

Listing 2.14. RightPosShift.java

```
1  public class RightPosShift {
2
3   public static void main(String[] args) {
4
5    int a = 1234;
6    // alle Bits 2 Stellen nach Rechts (Vorzeichen beachten!)
7    int b = a >> 2;
8    System.out.println("a : " + a + " Binaer: " + Integer.
         toBinaryString(a));
9    System.out.println("b : " + b + " Binaer: "
10                                  + Integer.toBinaryString(b));
11
12         }
13
14 }
```

Das Programm erzeugt die Ausgabe.

```
a : 1234 Binär: 10011010010
b : 308 Binär: 100110100
```

Alle Bits wurden um 2 Stellen nach rechts verschoben mit a >> b, werden die Bits der Zahl a um b Stellen nach rechts verschoben.

Beispiel für negative Zahlen

Listing 2.15. RightNegShift0.java

```
1  public class RightNegShift0 {
2
3   public static void main(String[] args) {
4
5    int a = -1234;
6    // alle Bits 2 Stellen nach rechts (Vorzeichen beachten!)
7    int b = a >> 1;
8    System.out.println("a : " + a + " Binaer: " + Integer.
         toBinaryString(a));
9    System.out.println("b : " + b + " Binaer: " + Integer.
         toBinaryString(b));
10  }
11 }
```

Das Programm erzeugt die Ausgabe.

```
a : -1234 Binär: 11111111111111111111101100101110
b : - 617 Binär: 11111111111111111111110110010111
```

Alle Bits wurden 1 Stelle nach rechts verschoben (a>>1). Aufgefüllt wurde mit 1, denn a=-1234 hat ganz links das Bit 1 gesetzt, wie bei allen negativen Zahlen der Fall ist.

Der >>> Operator

Der (logische Right-Shift-Operator) >>> wird genau wie der Arithmetische Operator verwendet, d.h. $a >>> b$ verschiebt die Bits der Zahl a um b Stellen nach rechts. Allerdings wird unabhängig von der Zahl a immer mit 0 aufgefüllt.
Das Beispiel verschiebt die Bits der (negativen) Zahl -1234 um jeweils ein Bit nach rechts. Zuerst wird der >> Operator verwendet und dann der >>> Operator.

Listing 2.16. RightNegShift.java

```
1  public class RightNegShift {
2
3  public static void main(String[] args) {
4          int a = -1234;
5          int b = a >> 1;
6          int c = a >>> 1;
7          // Ausgabe von a als Binaerzahl
8          System.out.println("-1234\t\t\tBinaer: " + Integer.
               toBinaryString(a));
9          // Right-Shift aufgefuellt wird mit 1
```

```
10          System.out.println("-1234 >> 1 = " + b + "\tBinaer: "
11                          + Integer.toBinaryString(b));
12          // Right-Shift aufgefuellt wird mit 0 (diese wird
                nicht mit
13          // ausgegeben, daher ist der String um eine Stelle
                kuerzer
14          System.out.println("-1234 >>> 1 =" + c + "\tBinaer: "
15                          + Integer.toBinaryString(c));
16          }
17  }
```

Das Programm erzeugt die Ausgabe.

```
-1234 Binär:
11111111111111111111101100101110
-1234 >> 1 =-617 Binär:
11111111111111111111110110010111
-1234 >>> 1 =2147483031 Binär:
01111111111111111111110110010111
```

Beim $>>>$Operator wird mit 0 nachgefüllt.

2.4.7 Nützliche Bit- und Bytemethoden

Die folgende Klasse stellt die im Anfang des Kapitels angesprochenen Bit-Methoden als statische Methoden bereit. Mit etwas Überlegen sollten alle Methoden aus den vorherigen Abschnitten verstanden werden können.

Listing 2.17. BitHelper.java

```
1   public class BitHelper {
2
3           public static int setBit(int pos, int value) {
4                   return value | (1 << pos);
5           }
6
7           public static int clearBit(int pos, int value) {
8                   return value & ~(1 << pos);
9           }
10
11          public static int flipBit(int pos, int value) {
12                  return value ^ (1 << pos);
13          }
14
15          public static boolean isBit(int pos, int value) {
16                  int mask = 1 << pos;
17                  return (value & mask) == mask;
18          }
19
20  }
```

Das nächste Beispiel zeigt wie die Klasse BitHelper angewendet wird:

Listing 2.18. Beispiel2410.java

```
1  public class Beispiel2410 {
2
3   public static void main(String[] args) {
4
5    // 255 entspricht der Dualzahl: 11111111
6    int zahl = 255;
7    // Bin re Ausgabe
8          System.out.println("255 => \tBin r: " + Integer.
               toBinaryString(zahl));
9          // Bit 2 und 5 l schen
10         zahl = BitHelper.clearBit(2, zahl);
11         zahl = BitHelper.clearBit(5, zahl);
12         System.out.println("\tBin r: " + Integer.
               toBinaryString(zahl));
13         // Bit 2 und 4 umdrehen
14         zahl = BitHelper.flipBit(2, zahl);
15         zahl = BitHelper.flipBit(4, zahl);
16         System.out.println("\tBin r: " + Integer.
               toBinaryString(zahl));
17         System.out.println("\tBin r: " + Integer.
               toBinaryString(zahl));
18         // Bit 5 setzen
19         zahl = BitHelper.setBit(5, zahl);
20         System.out.println("\tBin r: " + Integer.
               toBinaryString(zahl));
21
22         }
23 }
```

Das Beispiel verwendet die Dualzahl l 11111111 und wendet dann die Methoden der Klasse an, um gezielt einige Bits dieser Zahl zu verändern. Die Ausgabe lautet:

```
255 =>  Binär: 11111111
Binär: 11011011
Binär: 11001111
Binär: 11001111
Binär: 11101111
```

Diese Klasse wird uns später noch von Nutzen sein.

2.4.8 MSB/LSB oder wie man Bits und Bytes auch noch nennt

Der letzte Abschnitt über Bits und Bytes dient der Klärung zweier Begriffe, die in Zusammenhang mit Bit und Bytes früher oder später auftauchen werden. Für jede Zahl existieren die folgenden Begriffe (die in der Literatur leider nicht immer einheitlich verwendet werden):

- MSB (engl. most significant byte) ist das höchstwertigste Byte (und befindet sich immer links)
- msb (engl. most significant bit) ist das höchstwertigste Bit (und befindet sich ebenfalls immer links)
- LSB (engl. least significant Byte) ist das geringstwertigste Byte. Das ist das Byte, das rechts steht.
- lsb (engl. least singnificant bit) ist das geringstwertigste Bit. Das ist das Bit an der 1. Position ganz rechts.

Zur Veranschaulichung betrachten wir die folgende 2 Byte lange Zahl:

MSB (dies ist das höchstwertigste Byte)								LSB (dies ist das geringstwertige Byte)							
15	2. Byte							1. Byte							0
1	0	1	0	0	1	1	1	0	1	0	1	0	1	1	0
msb	← Das ist das höchstwertigste Bit							Dies ist das geringtswertigste Bit →							lsb

Abschließend geben wir noch zwei Beispiele für eine Integer-Variable. Integer sind in Java vier Bytes (32 Bit) lang. Der kleinste Wert ist -2.147.483.648 und der größte Wert, der mit Hilfe eines Integers dargestellt werden kann, beträgt 2.147.483.647.

MSB		Integer in Java sind 4 Byte lang		LSB	
msb	4. Byte	3. Byte	2. Byte	1. Byte	lsb

Beispiel 1:

```
int a = 47;  // 00101111
```

Für 47 gilt: lsb = 1, msb = 0 und LSB = 00101111
Beispiel 2:

```
int b = -24;
```

Für -24 gilt: lsb = 0; msb = 1, LSB = 11101000 und MSB = 11111111.
Um Beispiel 2 zu verifizieren, kann das kleine Java-Programm im nächsten Listing verwendet werden. Führen Sie es aus und überprüfen Sie die Aussagen.

Listing 2.19. MSBTest.java

```
1 public class MSBTest {
2   public static void main(String[] args) {
3    System.out.println(Integer.toBinaryString(-24));
4   }
5 }
```

3

Inbetriebname des IO-Warriors

Die Installation und Konfiguration des SDK wird unter Linux erklärt. Während alle anderen Kapitel weitgehend unabhängig vom verwendeten Betriebssystem sind, beziehen wir uns hier auf Linux (i.a. auf OpenSuse). In diesem Kapitel arbeiten Sie das erste mal mit echter Hardware, wenn auch nur um die Verbindung zu testen. Am Ende dieses Kapitels steht dem Messen, Steuern und Regeln mit dem Computer nichts mehr im Weg.

3.1 Programmieren auf dem IO-Warrior

Um ein erstes Testprogramm zum Laufen bringen zu können, ist einige Vorarbeit nötig. Zuerst wird eine IO-Warrior-Grundschaltung aufgebaut (doppelt kontrollieren!). Dann überprüfen wir, ob der Treiber schon vorhanden ist. Danach wird das SDK installiert und ein Testprogramm ausgeführt. Wenn alles geklappt hat, Konfigurieren wir noch Eclipse, um bequemer mit dem IO-Warrior arbeiten zu können.

Abbildung 3.1. Das Grundschema um mit dem IO-Warrior24-Baustein messen, steuern und regeln zu können, bleibt immer gleich. Der Baustein wird über USB mit dem PC verbunden. Ein Programmierer, der Anwendungen für den IO-Warrior erstellen will, greift auf das SDK zu. Das SKD verwendet den Treiber, um mit dem Baustein zu kommunizieren, d.h. Daten zu lesen oder Kommandos zu senden/empfangen.

Wir listen den Fahrplan, um den IO-Warrior ansteuern zu können nochmal auf:

- Grundschaltung aufbauen (dieser Hardwareteil ist unabhängig von den anderen Schritten und könnte auch an das Ende gestellt werden)
- Überprüfen, ob der IO-Warrior-Treiber (iowarrior.ko) bereits vorhanden ist
- Installation des SDK und starten eines Testprogramms
- Konfiguration von Eclipse

3.2 Hallo Welt $\rightsquigarrow$ Hardware

Für erste Testzwecke wird eine Grundschaltung benötigt. Wenn Sie das Gandalf-Board benutzen sind bereits alle Vorraussetzungen erfüllt.

3.2.1 IO-Warrior24-Grundschaltung

Falls ein Gandalf-Board zur Verfügung steht kann dieser Abschnitt übersprungen werden. Falls kein Gandalf-Board oder Starterkit zur Verfügung steht oder wenn man wissen möchte welche Bausteile für den Betrieb des IO-Warrors unbedingt notwendig sind sollte man diesen Abschnitt lesen. Der IO-Warrior24 hat eine Kerbe

Nr.	Bauteil	Anmerkungen
1	IO-Warrior24	Einbaurichtung beachten und Pins genau abzählen!
1	Widerstand 1.3Ω	
1	Kondensator C1 mit 100 nF	
1	Kondensator C2 mit 10 uF	Bei diesem Kondensator muss die
		Einbaurichtung beachtet werden.
1	USB Kabel oder Buchse von Typ B	Bei den Kabeln die Farben beachten!
1	IC-Sockel	Nicht notwendig aber empfehlenswert.

Tabelle 3.1. Stückliste für die IO-Warrior24 Grundschaltung

(alle ICs besitzen eine Markierung). Diese muss nach oben gerichtet sein. Die Pins werden dann gegen den Uhrzeigersinn von links beginnend nummeriert.
Es gibt nun verschiedene Möglichkeiten diese Grundschaltung aufzubauen, davon abgesehen, dass man natürlich auch das Starterkit von *Code Mercenarus* verwenden kann.

1. Die Bauteile werden auf eine Loch- oder Streifenplatine gelötet.
2. Man stellt sich selbst (Ätzen, Fräsen, ...) eine Platine her.
3. Man verzichtet zunächst auf eine Platine und benutzt eine Steckplatine (Steckbrett).

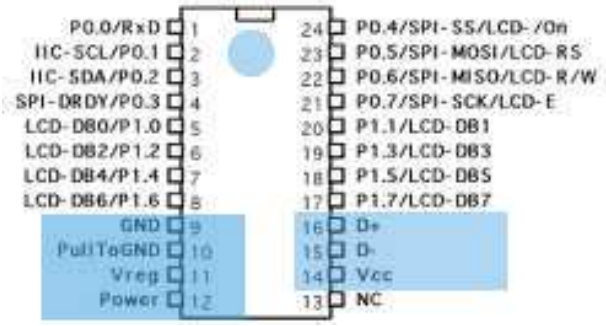


Abbildung 3.2. Pinbelegung IO-Warrior24. Die Pins für die Grundschaltung sind blau hinterlegt.

Pin	Funktion
Pin 9	GND (Masse)
Pin 10	PullToGND
Pin 11	Vreg (3V über R1 an Pin 15)
Pin 12	Power
Pin 14	Vcc
Pin 15	D- (weiß)
Pin 16	D+ (grün)

Tabelle 3.2. Benötigte Pins für die Grundschaltung.

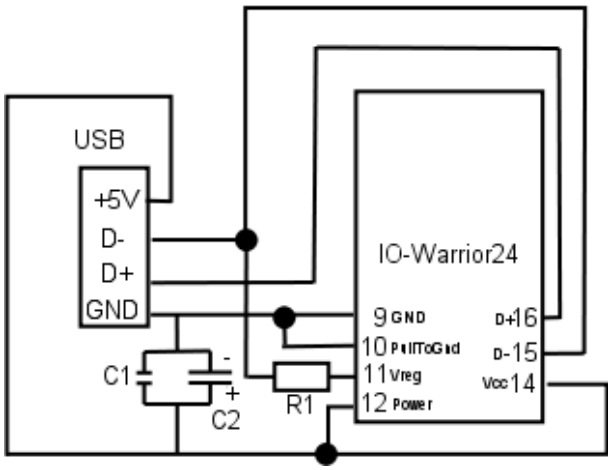


Abbildung 3.3. Schaltplan für die IO-Warrior24-Grundschaltung

Der Widerstand R1 soll undefinierte Zustände der Datenleitungen verhindern. Dazu wird die Datenleitung (D-, weiß, Pin 15) über R1 mit Vreg (Pin 11), der eine Spannung von 3V liefert, verbunden.
Die beiden Kondensatoren C1 und C2 dienen dem Ausgleich von Spannungsschwankungen der Spannungsversorgung und sorgen damit dafür, dass der IO-Warrior24 immer mit konstant 5V versorgt wird. Spannungsschwankungen können z.B. durch Schaltvorgänge verursacht werden. Kondensatoren sind Speicher für elektrische Ladungen und damit in der Lage ,im Falle eines kurzfristigen Spannungsabfalls Ladungen zu liefern, um den Spannungsabfall zu kompensieren. Steht dann wieder genügend Spannung zur Verfügung, laden sich die Kondensatoren wieder auf.

3.2.2 Gandalf-Board

Das Gandalf-Board ist eine erweiterte Grundschaltung. Es enthält Schutzdioden für alle Pins und Steckerleisen um bequem mit dem Steckbrett zu experimentieren. Damit sind alle Hardwarevorraussetzungen erfüllt und man kann sich dem nächsten Schritt (Überprüfen des Treibers) zuwenden.

3.3 Überprüfen des IO-Warrior-Treibers

In allen neueren Linux-Systemen ist der IO-Warrior-Treiber (iowarror.ko) bereits vorhanden. Das bedeutet, dass das Erstellen und Einbinden des Treibers nicht mehr notwendig ist. Deswegen wird es hier nicht weiter besprochen. Sollte der Treiber nicht vorhanden sein, dann muss man das von Codemerics zur Verfügung gestellte SDK herunterladen. Dort wird beschrieben, wie er erstellt wird.
Um zu überprüfen, ob der Treiber vorhanden ist, wird eine Konsole geöffnet und mit dem Befehl su in den Root-Modus gewechselt.

```
anaximenes:/home/torsten # modprobe iowarrior
```

Der Befehlt `modprobe` lädt den Kerneltreiber. In unseren Fall also den IO-Warrior. Wenn keine Fehlermeldung erscheint, ist das ein gutes Zeichen. Wir vergewissern uns noch ob der Treiber in der Liste aller gerade geladenen Treiber aufgeführt wird. Dazu dient der Befehl `lsmod`. Wir sollten etwa folgendes sehen:

```
anaximenes:/home/torsten # lsmod
Module                  Size  Used by
iowarrior              11790  0
snd_pcm_oss            53701  0
snd_mixer_oss          19447  1 snd_pcm_oss
snd_seq                68169  0
snd_seq_device          7834  1 snd_seq
...
```

Wichtig ist die Zeile in der das Wort **iowarrior** auftaucht, denn sie zeigt uns an, dass der iowarrior Treiber geladen ist. Wenn das der Fall ist, ist alles in Ordnung und der Treiber kann mit dem Befehl `rmmod` iowarrior wieder entladen werden.

```
anaximenes:/home/torsten # rmmod iowarrior
```

Der Treiber wird von Linux automatisch geladen wenn er mit dem Computer verbunden ist. Auch nach dem Ziehen des USB-Kabels des IO-Warriors bemerkt der Kernel, dass kein Baustein mehr vorhanden ist und entlädt den nun nicht mehr benötigten Treiber.
Unter Linux wird Hardware durch Gerätedateien repräsentiert. Die meisten neueren Linuxsysteme benutzen das udev System, um zur Laufzeit die benötigten Gerätedateien zur Verfügung zu stellen. Für den IO-Warrior24 existieren zwei Gerätedateien (eine für jedes Interface).

3.3.1 Setzen der USB-Rechte

Wir sollten noch überprüfen, ob ein normaler Benutzer die nötigen Rechte hat, um den IO-Warrior anzusprechen, denn es ist nicht unbedingt ratsam, im Root-Modus programmieren zu müssen. Die Installations-Routine von Codemerics berücksichtigt diesen Sachverhalt leider nicht, sodass wir ein wenig nacharbeiten müssen.
Zuerst wird der Inhalt des Verzeichnises `/dev/usb` aufgelistet:

```
torsten@anaximenes:/dev/usb> ls -l
insgesamt 0
crw------- 1 root root  180, 96 15.Okt14:09 hiddev0
crw------- 1 root users 180,208 15.Okt18:18 iowarrior0
crw------- 1 root users 180,209 15.Okt18:18 iowarrior1
```

Dem `ls -l` Kommando kann entnommen werden, dass zur Zeit nur der Benutzer Root in der Lage ist, den IO-Warrior Baustein zu verwenden. Für Testzwecke könnte man manuell diese Rechte ändern (z.B. mit `chmod 666 iowarior0`), um auch anderen Usern zu erlauben, auf den Baustein zuzugreifen. Zu beachten ist aber, dass diese Dateien jedesmal neu angelegt werden, wenn der IO-Warrior-Baustein mit dem Computer verbunden wird. Eine dauerhafte Änderung erfolgt durch das Anlegen einer Datei im `/etc/udev/rules.d` Verzeichnis.
HINWEIS: Falls nach dem Anstecken des IO-Warriors die Datei /dev/usb/iowarror0 (/dev/usb/iowarror1) bereits die entsprechenden Rechte besitzen sollten, ist es nicht nötig, den folgenden Abschnitt zu bearbeiten.
Um auch, nicht-Root-Benutzern den Zugriff auf die USB-Schnittstellle zu erlauben, wird eine Datei `10-iowarrior.rules` im Verzeichnis `/etc/udev/rules.d` mit dem weiter unten stehenden Inhalt angelegt.
HINWEIS: Diese Datei befindet sich auch in den zur Verfügung gestellten Downloads auf der Internetseite zu diesem Buch und muss ins Verzeichnis `/etc/udev/rules.d` kopiert werden.
Falls bereits eine Datei mit der Startnummer 10-Dateiname existiert, wird einfach eine andere noch freie Nummer wählen. Dateien mit kleineren Nummern werden nach dem Systemstart zuerst abgearbeitet.
Den Inhalt der Datei zeigen wir mit dem cat-Befehl an.

```
demokrit:/etc/udev/rules.d # cat 10-iowarrior.rules
```

```
# udev rules for iowarrior device nodes
SUBSYSTEM=="usb",KERNEL=="iowarrior[0-9]*",NAME="usb/iowarrior%n",
GROUP="users",MODE="666",OPTIONS="last_rule"
```

Tabelle 3.3. Die Datei 10-iowarriors.rules im Verzeichnis /etc/udev/rules.

Um die neu angelegte Regel dem System bekannt zu machen, ist es notwendig, die Regeln neu zu laden.

```
demokrit:/etc/udev/rules.d # udevadm control --reload-rules
```

Damit sind die USB-Rechte gesetzt.

3.4 Installation der Bibliotheken

Um den IO-Warrior ansteuern zu können, muss das Software Development Kit (SDK deutsch: Software Entwicklungs-Kit) von Codemerics installiert werden. Erst danach ist es möglich, z.B. von Eclipse aus, bequem auf den Baustein zuzugreifen.
Hierzu gibt es drei Möglichkeiten:

- **Quick-Install**
  Man benutzt die Quick-Installation, die lediglich Eclipse die Codemerics Bibliotheken mitteilt. Diese Methode ist die empfohlene Methode.
  – `http://www.informatics4kids.de/index.php/list-download`
- **bereinigte SDK**
  Man benutzt das “bereinigte SDK” auf der Seite von *www.informatics4kids.de*.
- Erfahrene Linux-User können auch das Orginal-Codemerics SDK verwenden. Man kann es auf der Codemerics-Homepage downloaden.

Das “orginal-SDK” unterscheidet sich in den folgenden Punkten von dem “bereinigten SDK”.

1. Das Verzeichnis `iowkit.1.5` enthält ein Leerzeichen. Das macht beim Übersetzen (compilieren) der Software Probleme.
2. Alle Altlasten z.B. das alte SDK 1.4 und die mittlerweile überflüssigen Treiber wurden entfernt. Für diejenigen die sich für den Treiber interessieren, ist das SDK von Codemerics also ein Muss. Für diejenigen, die den IOWARRIOR für eigene Mess-, Steuer- oder Regelungsaufgaben verwenden wollen, ist er nicht notwendig.
3. Bei der Verwendung von Java auf 64-Bit-Systemen gibt es ein Problem mit dem JNI bei einer Umwandlung von Integer in Long in der Datei `iowkitjni.c`. Dies führt dazu, dass Java auf diesen Systemen ohne diese Korrektur nicht verwendet werden kann.
4. Bereitstellen der `10-iowarrior.rules` Datei, um die USB-Rechte richtig zu setzen.

5. Im Verzeichnis `doc` wurde eine `codemerics.jar` mit Quellcode (Sourcecode) hinzugefügt. Dies dient lediglich dazu, um von Eclipse aus bequem durch den Sourcecode zu navigieren.

3.4.1 Quick-Install

Laden Sie von der *www.informatics4kids.de* Seite unter Download die folgende Datei herunter.

```
i4k_iowarrior_quick_install.zip
```

Entpacken Sie die Datei in Ihr Home-Verzeichnis. Das war **alles**.
Sie können nun mit der Konfiguration von Eclipse fortfahren. Durch das Entpacken ist folgende Verzeichnisstruktur entstanden.

```
i4k___doc
    |_install
    |_lib32
    |_lib64
    |_readme.txt
```

- **doc** Enthält die Javadoc-Dokumentation.
- **install** Enthält eine Datei um die USB-Rechte zu setzen und ein Beispiel als Testprogramm.
- **lib64** Die Bibliotheken wenn Sie ein 64-Bit System verwenden.
- **lib32** Die Bibliotheken wenn Sie ein 32-Bit System verwenden.

Um zu erfahren, ob Sie ein 32-Bit oder 64-Bit System verwenden geben Sie den Befehl `arch` in eine Konsole ein.

```
torsten@anaximenes:~> arch
i686
```

↷ Ist die Ausgabe für ein 32-Bit-System

und

```
torsten@anaximenes:~> arch
x86_64
```

↷ Deutet auf ein 64-Bit-System hin.

Diese Informationen werden bei der Konfiguration von Eclipse benötigt.

3.4.2 Installation des bereinigten SDK's

Wenn Sie nicht von der Quick-Installation gebrauch machen wollen, können Sie alternativ von der Internetseite dieses Buches das bereinigte LinuxSDK herunterladen.

```
linux_sdk_iowarrior_1.5_informatics4kids_v1.0.zip
```

Speichern Sie die Datei
`linux_sdk_iowarrior_1.5_informatics4kids_v1.0.zip` in Ihrem home-Verzeichnis.
Es folgen jetzt noch vier Schritte, die je nach der Geschwindigkeit des PCs ein wenig Zeit benötigen. Wenn Sie wieder die Eingabeauforderung an der Konsole sehen, können Sie den nächsten Schritt starten. Falls Sie Fehlermeldungen erhalten, müssen Sie nochmal zurück zu dem Abschnitt Voraussetzungen überprüfen.

1. Wechseln sie zuerst in die Root-Konsole (mit su) und tippen sie Anschließend `./configure` ein:

```
torsten@anaximenes:~> su
Passwort:
torsten@anaximenes:/home/torsten # ./configure
```

2. Geben Sie make ein:

```
torsten@anaximenes:/home/torsten #  make
```

3. Geben Sie make install ein:

```
torsten@anaximenes:/home/torsten # make install
```

4. Geben Sie make java-install ein:

```
torsten@anaximenes:/home/torsten # make java-install
```

Das war es. Wenn alles geklappt hat, ist das SDK nun installiert und einsatzbereit.

3.5 Konfiguration von Eclipse

Die folgenden Dateien werden fortan für alle IO-Warrior Projekte benötigt:

- `codemerics.jar` Java Bibliothek von Codemerics.
- `libiowkit.so` zugehörigte (native) Library (so=shared objects).
- `libiowkit.so.1` zugehörigte (native) Library (so=shared objects).
- `libiowkit.so.1.0.5` zugehörigte (native) Library (so=shared objects).

3.5.1 Konfiguration

Die Konfiguration erfolgt in drei Schritten. Dabei wird die `codemercs.jar` sowie die zugehörigte Nativelibrary `iowkit.so` dem Projekt hinzugefügt. Immer wenn ein neues IO-Warrior Projekt benötigt wird, sind die Schritte 2. und 3. durchzuführen.

1. Erstellen Sie ein neues Java Projekt mit Eclipse und nennen Sie es z.B. “Test”. Kopieren Sie die Klasse `Main` die sich im `i4k\install\` Verzeichnis befindet in das `src` Verzeichnis des Projekts und drücken sie `F5` zum Aktualisieren.

2. Jetzt wird die `codemercs.jar` dem Projekt hinzugefügt. Klicken Sie auf das Projekt mit der rechten Maustaste (Kontextmenü) und wählen Sie "**Properties**". In dem erscheinenden Dialogfeld wählen Sie "**Java Build Path**". Gehen Sie auf den Reiter "**Libraries**" und klicken Sie auf "**Add External JARs...**". Navigieren Sie zum Verzeichnis wo sich die `codemercs.jar` befindet. Damit ist die Java-Bibliothek dem Projekt hinzugefügt worden.
3. Da die Bibliothek aber von `*.so`-Datein abhängt, muss diese Nativelibrary das Verzeichnis der `*.so`-Dateien kennen. Dazu "klappen" Sie das ▷-`codemercs.jar` auf und geben das Verzeichnis `lib32` (oder `lib64` je nach System) an. Anschließend sollten Sie noch unter Javadoc den Pfad angeben, denn dann steht Ihnen in Eclispe die Hilfe zur Verfügung.

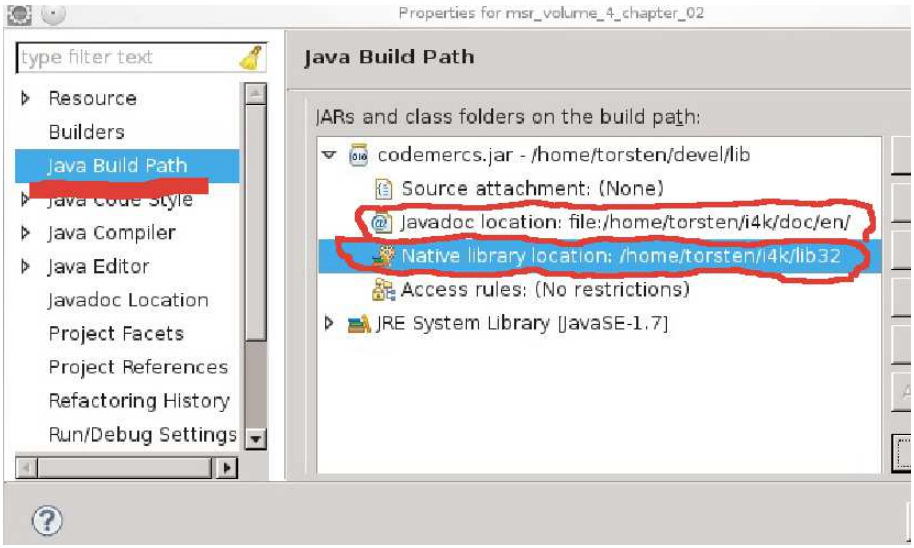


Abbildung 3.4. Nachdem die *codemercs.jar* hinzugefügt wurde muss noch die "Native library location" angegeben werden. Um bequem arbeiten zu können, empfiehlt es sich auch die "Javadoc location", das ist das Verzeichnis *i4k\doc\en*, einzutragen.

3.5.2 Testen der Konfiguration

Führen Sie nun das Java Programm aus. Das Programm gibt die Produkt und Seriennummer des Bausteins aus.

Listing 3.1. Main.java

```
1 import com.codemercs.iow.IowKit;
2
3 public class Main{
```

```
4
5  public static void main(String[] args) {
6
7    long h = IowKit.openDevice();
8    System.out.println("Product = "+ Long.toHexString(IowKit.
         getProductId(h)));
9    System.out.println(" Serial = "+ IowKit.getSerialNumber(h));
10   IowKit.closeDevice(h);
11   }
12 }
```

Das Programm erzeugt folgende Ausgabe, die zeigt das der Baustein gefunden wurde und somit alle Bibliotheken richtig eingebunden sind.

```
Product = 1501
Serial = 000012A6
```

Wenn Sie diese Ausgabe erhalten ist alles richtig eingerichtet.

HINWEIS

Die `codemerics.jar` muss in Eclipse eingebunden werden. Da es sich um eine Native-Library handelt muss das Bibliotheksverzeichnis zusätzlich angegeben werden. Falls das Testprogramm den Warrior nicht erkennt und lediglich `0000` zurückgibt, sind die USB-Rechte nicht richtig gesetzt (siehe Abschnitt USB-Rechte setzen).

3.6 Kurzübersicht SDK

Abschließend eine Auflistung der Methoden des IO-Warrior LinuxSDK. Beispiele wie einzelne Methoden anzuwenden sind, befinden sich in den Bänden II-IV. Funktionen, die für Linux nicht relevant sind, sind nicht mit aufgeführt. Die Bibliothek wurde von Thomas Wagner und Robert Marquardt entwickelt.

	METHOD SUMMARY CLASS IOWKIT VERSION: 1.5
static boolean	**cancelIo**(long `devHandle`, long `numPipe`)
	Bricht einen Lese/Schreibvorgang ab.
static void	**closeDevice**(long `devHandle`)
	Schießt alle geöffneten Geräte.
static long	**getDeviceHandle**(long `numDevice`)
	Liefert ein Handle (Gerätezugriff) zurück.
static long	**getNumDevs**()
	Ermittelt die Anzahl der angeschlossenen IO-Warrior's.
static long	**getProductID**(long `devHandle`)
	Liefert unter Angabe des Handle die Produkt ID zurück.
static long	**getRevision**(long `devHandle`)
	Liefert die Revisionsnummer der Firmware unter Angabe des Handels zurück.
static String	**getSerialNumber**(long `devHandle`)
	Liefert die Seriennummer unter Angabe des Handels zurück.
static long	**openDevice**()
	Öffnet alle IO-Warrior-Geräte. Ein Handle auf das erste Gerät wird zurückgeliefert.
static int[]	**read**(long `devHandle`, long `numPipe`, long `numPipe`)
	Liest Daten vom IO-Warrior. Diese Methode wartet (blockiert) solange bis Daten anliegen.
static int[]	**readImmediate**(long `devHandle`)
	Liefert den aktuellen Status der I/O Pins des IO-Warrior zurück.
static int[]	**readNonBlocking**(long `devHandle`, long `numPipe`, long `numPipe`)
	List Daten vom IO-Warrior ohne zu blockieren.
static boolean	**setTimeout**(long `devHandle`, long `timeout`)
	Setzt den Lese I/O Timeout in Millisekunden.
static boolean	**setWriteTimeout**(long `devHandle`, long `timeout`)
	Setzt den Timeout der I/O in Millisekunden.
static String	**version**()
	Liefert die Version als Zeichenkette zurück.
static long	**write**(long `devHandle`, long `numPipe`, int[] `buffer`)
	Schreibt das Array buffer über die Pipe mit der Nummer num-Pipe an den IO-Warrior.

4

Der IO-Warrior24

Dieses Kapitel beschreibt den Mikrocontroller IO-Warrior24, der die Grundlage für alle MSR-Projekte bildet. Dieses Kapitel soll Sie mit der grundlegenden Funktionsweise des Bausteins vertraut machen. Auf eine ausführliche Behandlung der Spezialfunktion wird an dieser Stelle verzichtet und auf die entsprechenden Kapitel in den Bänden II-IV verwiesen. Die erste Informationsquelle für Bausteine dieser Art ist immer das Datenblatt.

Was ist ein Mikrocontroller?

Ein Mikrocontroller (auch MC oder μC) integriert mehrere Komponenten (Prozessor, Speicher,...) auf einem einzigen Baustein (Chip). Dadurch benötigt der Baustein einerseits nur wenige externe Zusatzkomponenten, um in Betrieb genommen zu werden, und kann andererseits gleichzeitig eine Reihe von Funktionen bereitstellen. Deswegen nennt man Mikrocontroller auch Einchip-Computer (Single-Computer). Mikrocontroller sind weit verbreitet, z.B. in ihren Smartphone ihre Uhr, der Waschmaschine oder Autos. Überall sind diese Chips im Einsatz und der Markt wächst ständig weiter.
Ein Mikrocontroller besteht aus einem Kern und Peripheriekomponenten.

- Der Kern entspricht in etwa, was man bei einen PC als CPU bezeichnet. Er enthält also u.a. ein Rechenwerk, Steuerwerk und Register.
- Die Peripheriekomponenten des Controllers sind das Fenster zur Außenwelt. Durch Bereitstellung einer Reihe von Standardschnittstellen wird der Controller seiner eigentlichen Aufgabe gerecht. Nämlich Steuerungs-, Kontroll- und Kommunikationsaufgaben zu ermöglichen.

Die Merkmale und Peripheriekomponenten des IO-Warrior24 sind:

- 24 PIN Mikrocontroller mit Firmware (aktuell V. 1.0.3) von Code Mercenarias
- erhältlich im 24-poligen Gehäuse DIL24
- USB Schnittstelle V1.1/2.0
- einsetzbar im Low Power Mode (100 mA) oder High Power Mode (500 mA)

- 16 I/O Pins (Input/Output) typ. 125 Hz
- IIC Master Funktion, 100kHz, Durchsatz typ. 750 Bytes/sec
- SPI Master Schnittstelle, bis zu 2MHz, Durchsatz typ. 750 Bytes/sec
- Ansteuerung von HD44780 kompatiblen Displaymodulen und einigen Grafikmodulen
- Ansteuerung einer bis zu 8x32 großen LED Matrix
- Zwei 24 Bit (Capture) Timer mit einer Auflösung von 4 μsec.
- Empfang von RC5 kompatiblen Infrarot-Fernsteuerungen

Über die Pins kann der Mikrocontroller mit seiner Umwelt kommunizieren. Deswegen ist es wichtig, die Funktionalität dieser Pins genau zu kennen. Bevor wir im nächsten Abschnitt die Funktion der einzelnen Pins im Detail vorstellen, geben wir einen Überblick über die Grundfunktionalitäten die der IO-Warrior24 (bzw. ein Mikrocontroller) bereitstellt. Ein- / Ausgabeschnittstellen (Input/Output-Ports)

Ein- / Ausgabeschnittstellen (Input/Output-Ports)

Input/Output-Ports (IO-Ports) ermöglichen es, dass der Controller digitale Signale mit seiner Umwelt austauschen kann. Dies ist eine der wichtigsten Funktionalitäten für Mikrocontroller. Jeder Mikrocontroller verfügt daher über IO-Ports, wobei 8 Ports häufig eine Gruppe bilden.

Zusatzfunktionen (Spezialfunktionen)

Da die Anzahl der Pins begrenzt ist, und damit der Chip gleichzeitig möglichst viele Funktionen erfüllen kann, können einige die Pins neben ihrer I/O Fähigkeit häufig alternative Funktionalitäten repräsentieren. Über die Software kann dann die gewünschte Funktionalität ausgewählt werden.

Zähler (Timer)

Die meisten Controller besitzen Zähler. Ein Capture-Timer (engl. für Auffangen) ist ein Zähler, bei der der momentane Wert des Zählers im Mikrocontroller gespeichert wird, ohne das der Zähler dabei anhält. Der Zähler kann über die Software gestartet und ausgelesen werden.

4.1 Pinbelegung des IO-Warrior

I/O Ports

I/O Port steht für Input/Output Port also Eingabe/Ausgabe Port. Der Chip stellt 16 Pins zur Verfügung. Die Wahlweise als digitale Ein- oder Ausgänge benutzt werden können. Je 8 Pins bilden einen Port. Bei jedem Lese/Schreibvorgang an den Baustein

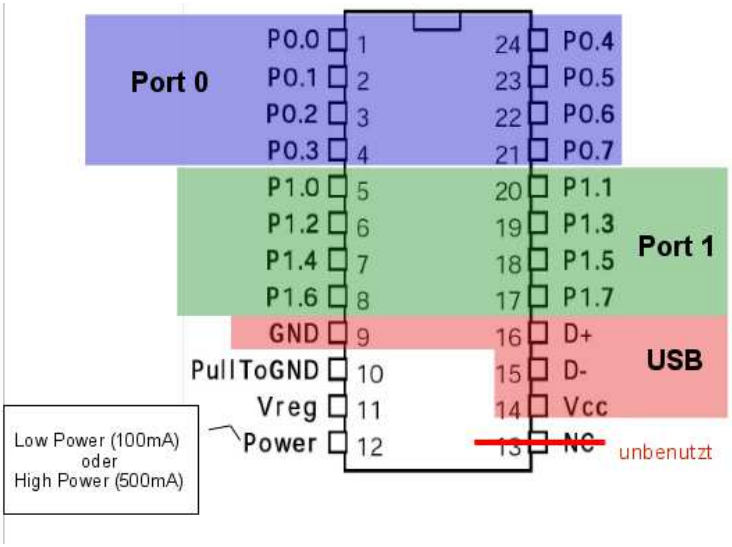


Abbildung 4.1. Draufsicht auf den IO-Warrior24 (Beachte die obere Markierung). Die Bedeutung der einzelnen "Beinchen" (Pins) wird im nächsten Abschnitt erklärt.

werden immer beide Ports (Port0 u. Port1) gelesen/geschrieben, d.h. man erfährt immer gleich den Zustand aller 16 I/O Pins. Mit diesen digitalen Ein- und Ausgabeports lassen sich unzählige Aufgaben erledigen. Wenn ein Ausgangspin auf 1 (High) gesetzt ist, hat er gegenüber Masse einen Wert von 5V. Ist er dagegen auf 0 (Low) gesetzt, nimmt er, gegenüber Masse einen Wert von 0 V an. Typische Beispiele die Eingabeports betreffen, sind das Einlesen von Schalterzuständen oder Ereignisse die von Lichtschranken ausgelöst worden sind. Klassische Anwendungen für Ausgabeports sind das Ansteuern von Leuchtdioden (LED) oder Motoren.

HINWEIS

Die Ausgangspins dürfen maximal mit 2 mA belastet werden. Schließen sie keine Verbraucher (auch keine Leuchtdioden) direkt an die Ausgangspins an.

USB-Anschluss

Die vier Leitungen des USB Anschlusses können direkt mit dem IO-Warrior24 verbunden werden. Lediglich bei Pin 15 (D-) darf nicht vergessen werden, ihn über einen 1.3 kΩ Widerstand mit Pin 11 (Vreg) zu verbinden. Vreg wird verwendet um die Datenleitung (D-) auf einen definierten Zustand (hier 3V) zu setzen. So können Störungen in der Übertragung vermieden werden.

Pins	Name	Beschreibung
15, 16	D-, D+	USB Datenleitungen
1,2,3 4,24,23 22, 21	P0.0, P0.1, P0.2 P0.3, P0.4, P0.5 P0.6, P0.7	Erste I/O Port
5,20,6 19, 7, 18 8, 17	P1.0, P1.1, P1.2 P1.3, P1.4, P1.5 P1.6, P1.7	Zweiter I/O Port
12	Power	Wird benutzt, um den Baustein im Low Power (10 0mA) Mode oder im High Power Mode (500 mA) zu betreiben.
10	PullToGND	Wird für die Chipherstellung benutzt und hat jetzt keine Bedeutung mehr. Dieser Pin sollte auf dem kürzesten Weg mit GND (Pin 9) verbunden werden.
9	GND	Masse (Ground)
14	Vcc	5 V Spannungsversorgung
11	Vreg, 3V	Stabilisierungsspannung für D-. Wird über einen (Pullup) Widerstand mit Pin 15 (D-) verbunden. Diese Spannung sollte nicht für andere Zwecke missbraucht werden!
13	NC (Not Connect)	Ungenutzt - diesen Pin nicht verbinden

Tabelle 4.1.

Power

Pin 12 kann entweder mit Pin 14 (Vcc, 5V) verbunden werden, oder mit Pin 9 (GND, Masse). Wird der Pin 12 mit Vcc verbunden, meldet sich das USB-Gerät als HighPower-Gerät am USB-Host an und stellt damit 500 mA zur Verfügung. Verbindet man dagegen den Pin 12 mit GND, dann stehen lediglich 100mA zur Verfügung, da sich das Gerät als LowPower-Gerät anmeldet. Dieser Status wird beim Anstecken des IO-Warriors an dem Computer gesetzt. Um also vom High-Power Modus in den Low-Power-Modus zu wechseln, müssten sie den Warrior vom PC trennen, den Pin 12 mit Masse anstatt mit Vcc verbinden und wieder an den PC stecken. In der Praxis benutzt man häufig Steckverbindungen (Jumper), um bequem eine Startkonfiguration auswählen zu können. Während des laufenden Betriebes des Bausteins, darf man den Status nicht ändern.

4.2 IO-Warrior Grundschaltung

Die Grundschaltung wurde schon im Kapitel “Inbetriebnahme des IO-Warriors” vorgestellt, soll aber der Vollständigkeit wegen hier noch einmal aufgeführt werden.

HINWEIS

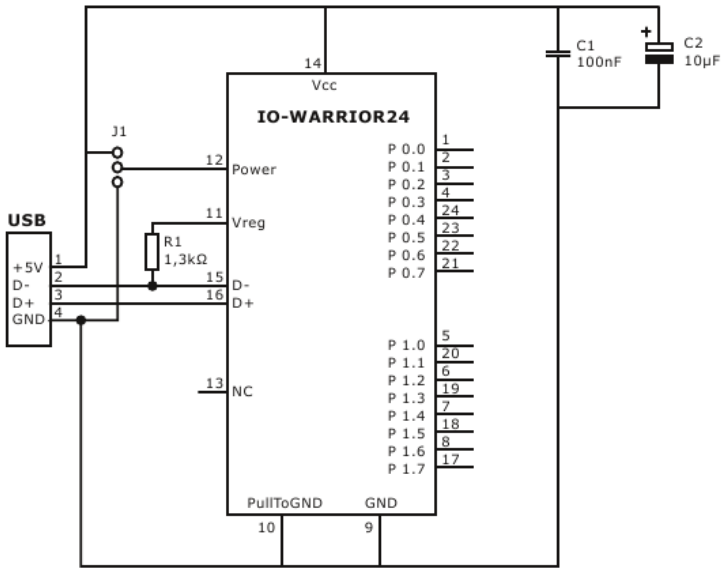


Abbildung 4.2. Grundschaltung: Wenn der Jumper J1 die Pins 12 und 14 (High Power Mode) verbindet, stehen 500 mA zur Verfügung. Wird Pin 12 auf GND (Masse) gezogen, dann werden 100 mA bereitgestellt. Beachten Sie den (Pullup)-Widerstand von D- auf Vreg. Mit dieser Schaltung ist der Mikrocontroller einsatzbereit und die 16 Pins stehen für allerlei Experimente bereit.

Bei dieser Schaltung stehen die Pins nicht an der richtigen Stelle. Diese Schaltung ist eher “schematisch”, d.h., man muss sich hier an der Pin-Nummerierung orientieren. Bei der Grundschaltung des vorherigen Kapitels stimmten die Positionen der Pins mit denen am echten Chip überein. Achten Sie bei zukünftigen Schaltungen darauf, ob es sich um einen “schematischen” oder “naturgetreuen” Schaltplan handelt.

4.3 Spezial-Funktionen

Der IO-Warrior24 verfügt neben den Digitalen Eingabe/Ausgabe Ports über eine Reihe von weiteren Möglichkeiten.

- I2C (sprich I-Quadrat-C, auch I^2C oder IIC, steht für Inter-Integrated Circuit), ist ein Industriestandard und wurde vor 20 Jahren von Philips entwickelt. Vom Handy bis zum Auto mit dieser Schnittstelle sind alle indirekt schon einmal in Kontakt getreten.
- SPI (Serial Peripheral Interface, von Motorola entwickelt) ist eine weitere häufig benutzte Schnittstelle für die es viele Bausteine gibt.

- Der IO-Warrior24 besitzt 2 Timer zur genauen Zeitmessung (bzw. Frequenzmessung oder Periodenbestimmung) mit einer Auflösung von 4 μs.
- Auch als RC5 Infrarot-Empfänger (z.B. für Fernsehsteuerungen) kann der IO-Warrior24 eingesetzt werden.
- Letztendlich können auch (LCD)-Displays angesprochen werden.

Wie die Spezialfunktionen im Detail benuzt werden, wird ausführlich im den Bänden III-IV gezeigt. Zunächst stellt sich die Frage, wie der Baustein diese zusätzlichen Funktionen zur Verfügung stellen kann.
Die Antwort ist, das einige Pins mit diesen zusätzlichen Funktionen belegt werden können.
Pinbelegung für die Spezial-Funktionen (Special mode functions):

- I^2C belegt P0.1 und P0.2
- SPI belegt die Pins P0.3, P0.4, P0.5, P0.6 und P0.7
- Die Timer belegen P0.0 und P0.1
- RC5 belegt P0.0
- LCD: Ein Display belegt P0.4, P0.5 P0.6, P0.7, P1.0, P1.2, P1.3, P1.4, P1.5, P1.6 und P1.7

Wenn ein Baustein eine Spezial-Funktion nutzt, dann stehen die jeweiligen Pins nicht mehr für I/O-Aufgaben zur Verfügung. Außerdem können die Pins jeweils nur eine Spezial-Funktion zur Zeit nutzen. Wenn also der I^2C - Bus genutzt wird, kann nicht gleichzeitig der Timer P0.1 benutzt werden, da der entsprechende Pin bereits belegt ist. Aus dem selben Grund kann auch SPI und LCD nicht gleichzeitig genutzt werden. Der Timer und RC5 können ebenfalls nur einzeln genutzt werden. Wenn also RC5 und ein Timer benötigt werden, könnte man einen zweiten Baustein am PC anschließen. Linux erlaubt, dass bis zu 8 Bausteine gleichzeitig betrieben werden können.
Grundsätzlich gilt, wenn Spezial-Funktionen genutzt werden sollen, wird durch die Software ein Kommando zu dem Baustein geschickt, der ihn veranlasst, diese Spezial-Funktion bereit zu stellen. Wenn die Spezial-Funktion später nicht mehr benötigt wird, wird ein weiteres Kommando an den Baustein geschickt, um den Modus wieder zu beenden. Damit stehen dann auch die I/O Pins wieder zur Verfügung.

4.4 IO-Warrior Kommunikation

Es werden einige Informationen über den Baustein aufgelistet. Diese Informationen wurden mit den im Kapitel: Grundlagen von USB vorgestellten Befehlen erarbeitet.
Die folgenden Informationen wurden mit dem Kommando
`tail -f -n 20 /var/log/messages` erhalten:

```
New USB device found, idVendor=07c0, idProduct=1501
Product: IO-Warrior24
Manufacturer: Code Mercenaries
SerialNumber: 000012A6
```

```
IOWarrior product=0x1501, serial=000012A6
interface=0 now attached to iowarrior0
IOWarrior product=0x1501, serial=000012A6
interface=1 now attached to iowarrior1
```

Die `VendorID=0x07C0` und die `ProduktID=0x1501` des Bausteins werden nun benutzt, um mit dem `lsusb` Befehl genaueres in Erfahrung zu bringen:

```
 anaximenes:/home/torsten #  lsusb -v -d 07c0:1501
Bus 001 Device 004: ID 07c0:1501 Code Mercenaries Hard-
und Software GmbH IO-Warrior 24
Device Descriptor:
bLength                18
bDescriptorType         1
bcdUSB               1.10
bDeviceClass            0 (Defined at Interface level)
bDeviceSubClass         0
bDeviceProtocol         0
bMaxPacketSize0         8
idVendor  0x07c0 Code Mercenaries Hard- und Software GmbH
idProduct          0x1501 IO-Warrior 24
bcdDevice           10.30
iManufacturer           1 Code Mercenaries
iProduct                2 IO-Warrior24
iSerial                 3 000012A6
bNumConfigurations      1
Configuration Descriptor:
bLength                 9
bDescriptorType         2
wTotalLength           59
bNumInterfaces          2
bConfigurationValue     1
iConfiguration          4 Generic IO
bmAttributes            0xa0
(Bus Powered)
Remote Wakeup
MaxPower              500mA
Interface Descriptor:
bLength                 9
bDescriptorType         4
bInterfaceNumber        0
bAlternateSetting       0
bNumEndpoints           1
bInterfaceClass         3 Human Interface Device
bInterfaceSubClass      0 No Subclass
bInterfaceProtocol      0 None
```

```
  iInterface              5 Plain I/O
  HID Device Descriptor:
  bLength                 9
  bDescriptorType        33
  bcdHID               1.01
  bCountryCode            0 Not supported
  bNumDescriptors         1
  bDescriptorType        34 Report
  wDescriptorLength      33
  Report Descriptors:
  ** UNAVAILABLE **
  Endpoint Descriptor:
  bLength                 7
  bDescriptorType         5
  bEndpointAddress     0x81  EP 1 IN
  bmAttributes            3
  Transfer Type            Interrupt
  Synch Type               None
  Usage Type               Data
  wMaxPacketSize     0x0002  1x 2 bytes
  bInterval              10
  Interface Descriptor:
  bLength                 9
  bDescriptorType         4
  bInterfaceNumber        1
  bAlternateSetting       0
  bNumEndpoints           1
  bInterfaceClass         3 Human Interface Device
  bInterfaceSubClass      0 No Subclass
  bInterfaceProtocol      0 None
  iInterface              6 Complex Interfaces
  HID Device Descriptor:
  bLength                 9
  bDescriptorType        33
  bcdHID               1.01
  bCountryCode            0 Not supported
  bNumDescriptors         1
  bDescriptorType        34 Report
  wDescriptorLength     215
  Report Descriptors:
  ** UNAVAILABLE **
  Endpoint Descriptor:
  bLength                 7
  bDescriptorType         5
  bEndpointAddress     0x82  EP 2 IN
```

```
bmAttributes            3
Transfer Type            Interrupt
Synch Type               None
Usage Type               Data
wMaxPacketSize     0x0008  1x 8 bytes

Interval               10
Device Status:     0x0000
(Bus Powered)
torsten@anaximenes:/home/torsten #
```

Die mit dem `lsusb` Befehl ermittelten und in diesem Zusammenhang relevanten Informationen werden in der folgenden Grafik zusammengefasst. Der IO-Warrior24

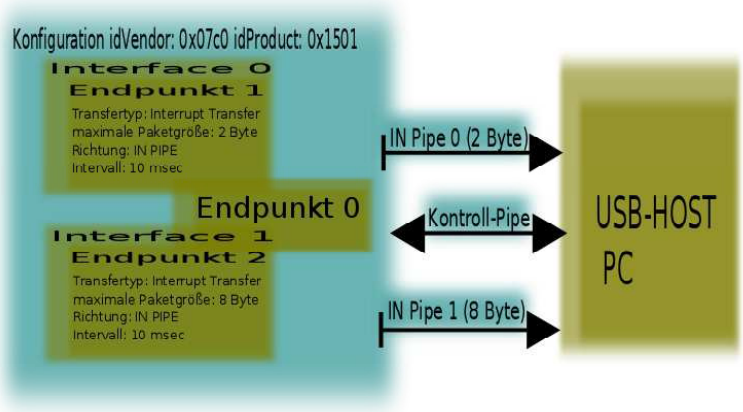


Abbildung 4.3. Logischer Aufbau des IO-Warrior24. Für den Endpunkt 0 gibt es keinen "Beschreiber" (Descriptor), jedoch muss jedes Gerät über diesen Endpunkt verfügen. Der Endpunkt 0 steht außerdem beiden Interfaces zur Verfügung.

besitzt eine Konfiguration und zwei Interfaces. Der Baustein hat neben dem Endpunkt 0 für jedes Interface einen weiteren Endpunkt (als LowSpeed Device darf es nur über 3 Endpunkte verfügen). Der IO-Warrrior24 hat die `VendorID: 0x07c0` und die `ProduktID: 0x1501`.
Interface 0 mit Endpunkt 1 steht für die Kommunikation mit den Eingangs-/Ausgabe-Pins. Interface 1 mit Endpunkt 2 ist für die Kommunikation der Spezial-Funktion. Der Endpunkt 0 kann sowohl mit Interface 0 als auch mit Interface 1 operieren.

Allgemeines Kommunikationsprinzip

- Über die Pipe 0 werden die I/O Pins gesteuert.
  - Ausgabe-Pins: Über die Kontrollpipe (Endpunkt 0, Interface 0) werden die Ausgabeports gesetzt.

  - Eingabe-Pins: Über Endpunkt 1 (Interface 0) kann der Zustand der Pins eingelesen werden.
- Über die Pipe 1 werden die Spezial-Funktionen gemanagt.
  - Kommandos: Über die Kontrollpipe (Endpunkt 0, Interface 1) werden Kommandos für die IO-Warrior24 Spezial-Funktionen gesetzt.
  - Daten: Über Endpunkt 2 (Interface 1) werden Antworten auf die Spezial-Kommandos bzw. Daten empfangen.

Das Senden/Empfangen von Kommandos/Daten erfolgt dabei immer in Form sogenannter Reports. Diese Reports werden im nächsten Abschnitt behandelt.

4.5 Alles läuft über Reports

Alle Datenpakete, die mit dem IO-Warrior24 ausgetauscht werden, werden in Form sogenannter Reports verschickt. Der IO-Warrior24 kann maximal 125 Reports pro Sekunde übertragen. Jeder Report repräsentiert ein komplettes Datenpaket. Für die I/O-Pins, z.B., enthält ein Report jeweils den kompletten Status der 16 Pins. Bezogen auf eine Spezialfunktion (IIC, SPI, LCD) repräsentiert ein Report ein Kommando (oder Datenpaket) von/zu den Spezial-Funktionen.

Zwei verschieden lange Reports

Reports von/zu den I/O Pins sind 3 Byte lang, während Reports von/zu den Spezialfunktionen eine Länge von 8 Byte haben. Um ein Report verschicken zu können, muss zusätzlich die Pipe gewählt werden. Reports die 3 Byte lang sind, laufen über Pipe 0, solche die 8 Byte lang sind über Pipe 1. In diesem Zusammenhang spielt die ReportID eine wichtige Rolle. Jeder Report-Typ besitzt eine eigene ReportID. Diese ID bildet jeweils das 0. Byte eines Reports. Die ReportIDs sind für alle Schreibvorgänge (oder Lesevorgänge) eindeutige Zahlen, die eine bestimmte Aktion einleiten (oder beenden). Die ReportIDs ermöglichen es, dass unterschiedliche Funktionen über einen Endpunkt genutzt werden können.

I/O Reports

Ein Report für die I/O Pins sieht demnach folgendermaßen aus. Das nullte Byte bildet die ReportID. Das erste Byte repräsentiert den Port 0 (P0.0-P0.7= 8 Pins) und das zweite Byte repräsentiert den Port 1 (P1.0-P1.7 = 8 Pins).

```
******** JAVA CODE ZUR ERSTELLUNG EINES I/O-REPORTS ******
 int report[] = new int [3];
 report[0] = 0 // ReportID = 0 für I/O Kommunikation
 // report[1] =  //EINE ZAHL 0 bis 255    für den Port 0
 // report[2] =  //EINE ZAHL 0 bis 255    für den Port 1
```

0. Byte	1. Byte	2. Byte
ReportID=0	Port0	Port1

Tabelle 4.2. Reports zu den I/O Pins haben eine Länge von drei Bytes und die ReportID 0x0. Jedes Byte wird durch ein Integer-Wert (0-255) repräsentiert.

0. Byte	1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte	7. Byte
ReportID							

Tabelle 4.3. Reports die über Spezialfunktionen laufen sind 8 Byte lang.

Spezialfunktion Reports

Ein Report für die Spezialfunktionen (Interface 1) besitzt folgende Struktur. Die jeweilige Bedeutung der Bytes 1-7 wird im in den Bänder III-IV behandelt und spielt zunächst keine Rolle.

4.6 Java: Grundlegende Kommunikation und Reports

Ein kleines Java-Programm, das einen Report erzeugt und an das Gerät sendet, wird vorgestellt. Die hier vorgestellten Prinzipien ändern sich nicht mehr und müssen zukünftig in allen Java Programmen wiederzufinden sein

1. Verbindung öffnen, um ein Handle zu erhalten.
2. Mit Hilfe des Handles lesend oder schreibend auf das Gerät zugreifen.
3. Verbindung schließen.

Der Begriff Handle bedeutet in etwa Zugriffsspezifizierer (am besten verzichtet man aber auf eine deutsche Übersetzung). Bei dem Öffnen der Verbindung laufen einige Dinge im Hintergrund ab. Das Resultat ist letztendlich ein gültiges Handle, mit dem fortan weitergearbeitet werden kann ,um z.B. Schreib- oder Leseoperationen durchführen zu können. Das Handle repräsentiert sozusagen das Device und mit Hilfe des Handles kann zwischen mehreren angeschlossenen Geräten unterschieden werden.
Das folgende Programm öffnet zunächst das Device. Anschließend wird ein Report erstellt und an das Gerät geschickt. Dabei werden alle Pins des Port 0 auf Low (0V) gesetzt. Die Pins von Port 1 werden auf High (5V) gesetzt. Danach wird das Device geschlossen.

Listing 4.1. Grundprinzip.java

```
1 import com.codemercs.iow.IowKit;
2
3 public class Grundprinzip {
4
```

ReportID	Bedeutung
0x0	Lesen oder Schreiben zu den I/O Pins
	IIC Spezial-Funktionen
0x01	IIC aktivieren oder deaktivieren (abhängig vom 1. Byte)
0x02	IIC Daten senden (write request) oder Bestätigung, dass Daten empfangen wurden
0x03	IIC zum Daten Lesen vorbereiten oder IIC Daten lesen
	LCD Spezial-Funktionen
0x04	LCD aktivieren oder deaktivieren (abhängig vom 1. Byte)
0x05	LCD Daten schreiben
0x06	LCD Daten lesen vorbereiten oder LCD Daten lesen
	SPI Spezial-Funktionen
0x08	SPI aktivieren oder deaktivieren (abhängig vom 1. Byte)
0x09	Datentransfer vorbereiten oder Daten lesen
	Aktueller Pin-Status
0xFF	Aktuellen Pin-Status erfragen (write request) oder aktuellen Pin-Status lesen
	Empfangen eines RC5 Infrarot-Codes
0x0C	RC5 aktivieren oder deaktivieren (abhängig vom 1. Byte) oder RC5 Daten lesen
	LED Spezial-Funktionen
0x14	LED aktivieren oder deaktivieren (abhängig vom 1. Byte)
0x15	LED-Daten anzeigen
	Timer
0x28	Timer aktivieren oder deaktivieren (abhängig vom 1. Byte)
0x29	Lesen Timer A (Pin 0.0)
0x2A	Lesen Timer B (Pin 0.1)

Tabelle 4.4. Liste der möglichen Reports (mit ID's) für den IO-Warrior24. Einige ReportIDs tauchen scheinbar zweimal auf (z.B. 0x0). Das liegt daran, dass die ID einmal im Report verwendet wird, der an das Gerät gesendet wird und anderseits auch Verwendung findet im Report, der vom Gerät gelesen werden kann.

```
5    public static void main(String[] args) {
6
7     long h = IowKit.openDevice();
8
9      // Report erstellen
10    int report[] = new int [3];
11    // ReportID = 0 fuer I/O Kommunikation
12    report[0] = 0;
13    // Port 0 LOW setzen
14    report[1] = 0;
15    // Port 1 HIGH setzen
16    report[2] = 255;
17
18    long result = IowKit.write(h, 0 /* Pipe 0*/ , report);
19       if(result != report.length)
20           System.out.println("ERROR: IowKit.write()");
```

```
21
22      IowKit.closeDevice(h);
23      }
24  }
```

Mit der folgenden Zeile

```
long h = IowKit.openDevice();
```

wird das Device geöffnet und ein Handle h zurückgeliefert. Dieses Handle wird in den beiden folgenden Methoden jeweils verwendet.
Ein Report ist in Java immer ein Integer-Array entweder der **Länge 3 (für I/O Operationen)** oder der **Länge 8 (für die Spezial-Funktionen)**. Die gültige ReportID kann aus der Tabelle entnommen werden. Sie ist für I/O immer 0. Der oben stehende

0. Byte	1. Byte	2. Byte
0x0	0	255

Tabelle 4.5. I/O Pins: Ein Report, der im Port 0 eine 0 und im Port 1 eine 255 setzt.

Report wird in Java folgendermaßen geschrieben.

```
int report[] = new int [3];
report[0] = 0;
report[1] = 0;   // Binär 00000000 alle Pins auf Low
report[2] = 255; // Binär 11111111 alle Pins auf High
```

Das vorgestellte Schema wird noch oft Anwendung finden. Auf die Tatsache, dass die Reports in Java mit Integer-Arrays realisiert werden, wird noch einmal eingegangen. Ein Integer hat in Java die Länge von 4 Bytes, die Methoden des IowKits berücksichtigen aber immer nur das erste Byte des Integer-Wertes, die anderen 3 Bytes werden ignoriert. D.h.,

```
report[2] = 256 ;//Binär 100000000
```

würde alle Pins auf Low setzen (die 1 würde nicht berücksichtigt werden). Zusammengefasst heißt das, die Java-Reports werden durch Integer-Arrays repräsentiert, deren Wertebereich zwischen 0-255 liegen müssen.
Die write()-Methode schreibt nun diesen Report über die Pipe 0 an das Device.

```
IowKit.write(h, 0 /* Pipe 0*/ , report);
```

Als ersten Parameter wird das Handle `h` übergeben. Der zweite Parameter ist die zu verwendende Pipe, hier 0 (grundsätzlich entweder 0 oder 1, andere Werte gibt es nicht). Der letzte Parameter sind die Daten (der Report), die geschrieben werden sollen. Anschließend wird das Device geschlossen. Dazu wird das Handle `h` als Parameter übergeben.

4.6.1 IO-Warrior „Hello World“ Testprogramm

Das im Abschnitt „Testen der Konfiguration“ vorgestellte Programmen können Sie als eine Art „Hello World“ Programm auffassen. Wenn es Fehlerfrei läuft, haben Sie erfolgreich eine Verbindung zu dem Mikrocontroller aufgebaut. Sie können nun mit dem Messen, Steuern und Regeln loslegen.

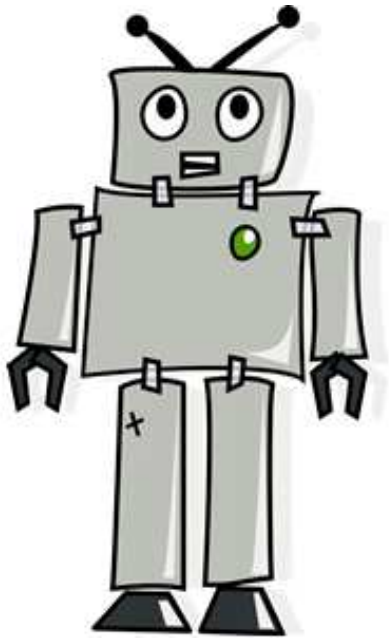
Listing 4.2. Main.java

```
1  import com.codemercs.iow.IowKit;
2
3  public class Main{
4
5   public static void main(String[] args) {
6
7    long h = IowKit.openDevice();
8    System.out.println("Product = "+ Long.toHexString(IowKit.
         getProductId(h)));
9    System.out.println(" Serial = "+ IowKit.getSerialNumber(h));
10   IowKit.closeDevice(h);
11   }
12 }
```

Wenn die ProduktID und die Seriennummer erkannt wurde (also nicht 0 sind) ist der Baustein einsatzbereit.

Index

Openbook
Band I



MSR mit Linux, Java und dem IO-Warrior24

Quellcode ↷ http://www.informatics4kids.de

Torsten Röhl