

Systemnahe Programmierung

Technische Informatik II

Roman Wetenkamp

7. Dezember 2021



Inhaltsverzeichnis

1	Einführung	4
2	Hardware	5
3	Grundlegendes zur Programmierung	7

Vorwort

Systemnah Programmieren - Klingt komplizierter als es ist. Hier notieren wir alles Relevante aus der Vorlesung, ergänzen es um ein paar Aufgaben und Anmerkungen und bereiten uns auf die Klausur vor. Dieses Skript basiert auf der Vorlesung von Joachim Wagner an der DHBW Mannheim im Studiengang Informatik – Cyber Security.

Viel Erfolg!

Roman Wetenkamp
Mannheim, den 7. Dezember 2021

Warnung Das Studium an einer Dualen Hochschule unterscheidet sich von dem Studium an Universitäten oder regulären Fachhochschulen insbesondere dadurch, dass aufgrund der Dualität von Theorie und Praxis meist nur die Hälfte der Zeit zur Vermittlung des Stoffes zur Verfügung steht (wenn dann auch intensiver). Daher gehen Sie bitte nicht davon aus, dass Sie dieses Skript ausreichend auf Klausuren in regulären Vollzeitstudiengängen vorbereitet!

Hinweis Dieses Dokument ist kein Vorlesungsmaterial, hat nicht den Anspruch auf Vollständigkeit und enthält mit Sicherheit Fehler. Desweiteren ist es noch lange nicht vollendet (es ist infrage zu stellen, ob es das je sein wird), und doch möchte ich Sie ermutigen, beizutragen! Jegliche Fehler, Probleme oder Anmerkungen können Sie mir gerne über das dazugehörige GitHub-Repository unter der URL <https://github.com/RWetenkamp/kommmnet> zukommen lassen. Danke!

1 Einführung

Systemnah zu programmieren bedeutet zuallererst, viele Ebenen der Abstraktion, die Betriebssysteme oder Virtualisierungen hinter sich zu lassen und zurückzukehren zu dem grundlegendsten Konzept der Computertechnik, dem Bit.

Definition 1. *Ein Bit ist die kleinste Speichereinheit eines Computersystems. Der Wert eines Bits ist entweder 0 für **aus** oder 1 für **ein**. Sprechen wir von einem 8-Bit-Register, so bezeichnen wir damit acht einzeln adressierbare Werte, die jeweils auf 0 oder 1 gesetzt werden können.*

$$8 \text{ Bit} \hat{=} 1 \text{ Byte}$$

Sie werden in Ihren Programmen verschiedene Bitoperationen implementieren müssen, da die zugrundeliegende Hardware zu großen Teilen aus Registern besteht, die sie auf diese Weise ansprechen können.

Bezeichnung	Symbol	C-Operatorsymbol	<i>a</i>	<i>b</i>	Ergebnis
Konjunktion	\wedge	$\&$	0	0	0
			0	1	0
			1	0	0
			1	1	1
Disjunktion	\vee	$ $	0	0	0
			0	1	1
			1	0	1
			1	1	1
XOR	\oplus	\wedge	0	0	0
			0	1	1
			1	0	1
			1	1	0

Tabelle 1: Übersicht über binäre Bitoperationen

Bezeichnung	Symbol	C-Operatorsymbol	<i>a</i>	Ergebnis
Negation	\neg		0	1
			1	0
Linksverschiebung		\ll	0000	1 \ll 1: 0001, 1 \ll n: 00...1...0
Rechtsverschiebung		\gg	1000	1 \gg 1: 0100, 1 \ll n: 00...1...0

Tabelle 2: Übersicht über unäre Bitoperationen

Im folgenden Kapitel betrachten wir die verwendete Hardware und beginnen damit, die einzelnen Bestandteile und ihre Aufgaben zu erläutern. Wir werden jeweils anhand von Beispielprogrammen zeigen, wie die einzelnen Komponenten angesprochen und programmiert werden können. Die Programmiersprache hierfür ist C, in der Vorlesung wurde das „Microchip Studio“ als IDE verwendet. Sie können es hier beziehen:

<https://www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for>

2 Hardware

Grundlegend ist Ihnen der Aufbau eines Computersystems sicherlich vertraut.

- Die **CPU** (Central Processing Unit), also der Prozessor, ist die zentrale Komponente, die die Befehle ausführt.
- Während der Ausführung speichert die CPU Programmdaten im Arbeitsspeicher, dem **RAM** (Random Accessible Memory). Dieser Speicher wird nur zur Laufzeit genutzt und ist daher nicht persistent.
- Ebenso gibt es einen Massenspeicher, eine Festplatte, auf der Daten persistent gespeichert werden und bei Bedarf gelesen/geschrieben werden.

Diese Grundstruktur gleicht im Wesentlichen dem VON-NEUMANN-PRINZIP und ist sowohl auf klassische PCs anwendbar als auch auf Microcontroller, denen wir uns im Rahmen dieser Vorlesung widmen wollen.

Bemerkung 1. *Ein **Microcontroller** ist ein Minicomputer auf einem Chip. Er enthält einen Microprozessor und Speicher.*

Ein solcher Microcontroller findet sich häufig auf Entwicklungsboards, wie z. B. der Arduino-Produktfamilie, die bereits aufgelötete Ports für weitere Komponenten enthalten. Derartige Boards gibt es zuhauf im Markt. Häufig sind die Unterschiede gering, da Arduino-Boards unter einer Open-Source-Lizenz stehen und somit von anderen ohne Weiteres kopiert werden dürfen.

Im Rahmen dieser Vorlesung arbeiten wir mit Arduino-Microcontrollern oder zu Arduino kompatiblen Äquivalenten. Die von uns verwendeten Boards enthalten allesamt folgenden Microprozessor:

ATMEL ATmega 328P

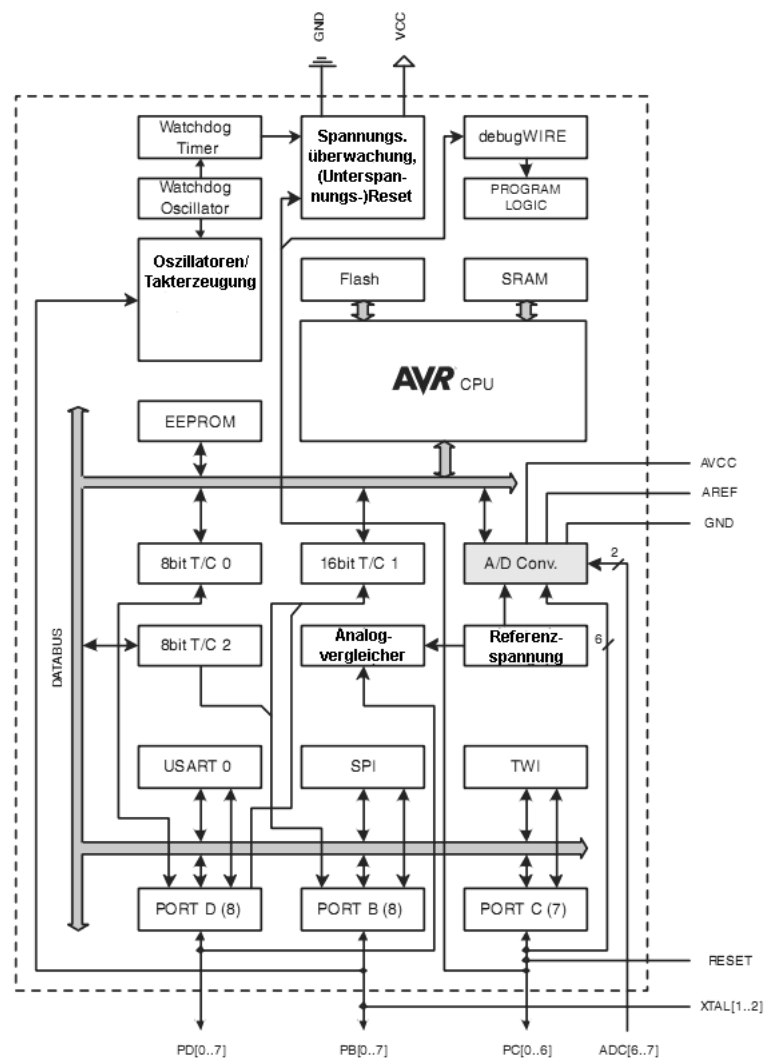
Die Kenntnis des Microcontrollers ist unabdingbar für den weiteren Verlauf der Vorlesung. Die Microcontroller Arduino Uno und Arduino Nano enthalten genannten Prozessor. Wir werden als Referenz jeweils den Arduino Uno verwenden.

Aufbau Dieser Microcontroller enthält einige relevante Komponenten, die über einen Bus mit der CPU verbunden sind. Wir entnehmen diese dem folgenden Blockschaltbild.

- **Flash** – Der Flash-Speicher ist neben SRAM und EEPROM einer der drei Speichertypen. Der Flash-Speicher ist unveränderlich und kann lediglich von außen gebrannt werden. Hier wird das aktuelle Skript / Programm gespeichert.
- **SRAM** – Hierbei handelt es sich um einen winzig kleinen Arbeitsspeicher. Dieser Speicher ist nicht persistent und wird von der CPU während der Laufzeit verwaltet und verwendet.
- **EEPROM** – Dieser Speicher ist persistent und fungiert als „Festplatte“.
- **Watchdog** – Sollte der Chip nach einer gewissen Zeit kein Lebenssignal mehr senden, setzt der Watchdog den Microcontroller automatisch zurück.



Abbildung 1: Arduino Uno mit ATMEL ATmega328P



- **A/D Converter** – Die Prozessoren der ATmega-Reihe können analoge Signale in digitale umwandeln. Dafür ist dieser Baustein zuständig.

- **8/16bit T/C** – Diese Bauteile sind Timer/Counter, die zur Prozesssteuerung genutzt werden können. Wir widmen uns diesen Bauteilen später.
- **USART** – Eine serielle Schnittstelle für diversen Datenverkehr ist der USART-Baustein.
- **TWI** – Die TWI-Schnittstelle kann für die Verknüpfung mehrerer Microcontroller untereinander genutzt werden.

Im Allgemeinen wird die Kommunikation mit jeder Hardware/CPU über I/O-Ports abgewickelt.

Bemerkung 2. *Gelegentlich kann es zu Verwirrung kommen, da sowohl internen I/O-Ports des Microcontrollers als auch die Anschlusspins für elektronische Bauteile als **Ports** bezeichnet werden. Im Kontext dieses Kurses bezeichnen wir letztere als „Beinchen“.*

System Der hier gezeigte Controller folgt der **RISC**-Architektur. RISC steht für „Reduced Instruction Set Controller“. Es hat sich gezeigt, dass mit einem im Vergleich zu Assembler reduzierten Befehlssatz ähnlich effizient, jedoch viel leichter dekodierbar und in der Regel schneller, gearbeitet werden kann. Der Microcontroller enthält von sich aus kein Betriebssystem. Wollen wir ein Programm ausführen, so müssen wir dieses speziell für diesen Microprozessor in einer hardwarenahen Programmiersprache wie Java oder C implementieren.

3 Grundlegendes zur Programmierung

Wir werden in dieser Vorlesung – und damit auch in diesem Skript – die Programmiersprache C verwenden. Um die benötigten Bibliotheken nutzen zu können, müssen die passenden Treiber installiert werden. Am einfachsten lässt sich dies durch die Arduino IDE und das zuvor erwähnte Microchip Studio realisieren. Das **Flashen**, also das Aufspielen der Programme auf das Board, erfolgt über USB.

Programmaufbau Wie Sie es von C gewohnt sein werden, unterscheiden wir **Programmzeilen** und **Präprozessordirektiven**. Die Befehle, die ausgeführt werden sollen, müssen Bestandteil der `main()`-Funktion des Programms sein.

Bemerkung 3. *Jedes Programm, dass Sie flashen wollen, muss eine Endlosschleife enthalten! Andernfalls wird die Operation lediglich ein einziges Mal ausgeführt.*

Angewandt bedeutet dies:

```
#define F_CPU 16000000UL
#include <avr/io.h>
```

```
int main() {
    // Initialisierungen
    while(1) {
        // Hier koennen Sie Ihren Code platzieren
    }
}
```

Sie erkennen die Definition der Konstante `F_CPU` in der ersten Zeile. Hier legen wir fest, dass der Prozessor mit einem Takt von 16 MHz arbeitet.