Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Praktikum Systemprogrammierung

Versuch 1

Mikrocontroller & AD/DA-Wandlung

Lehrstuhl Informatik 11 - RWTH Aachen

6. April 2023

Commit: b3175525

Inhaltsverzeichnis

Dieses Dokument ist Teil der begleitenden Unterlagen zum *Praktikum Systemprogrammierung*. Alle zu diesem Praktikum benötigten Unterlagen stehen im Moodle-Lernraum unter https://moodle.rwth-aachen.de zum Download bereit. Folgende E-Mail-Adresse ist für Kritik, Anregungen oder Verbesserungsvorschläge verfügbar:

support.psp@embedded.rwth-aachen.de

1 Mikrocontroller & AD/DA-Wandlung

Mikrocontroller sind Halbleiterchips, die neben dem Prozessor weitere Peripheriefunktionen auf einem Chip vereinen. Dadurch kommen Projekte mit Mikrocontrollern oft mit wenigen Bauteilen aus. Sie sind häufig in alltäglichen Gegenständen zu finden und werden dann eingebettete Systeme genannt. Sie finden insbesondere in der digitalen Kontrolle und Steuerung von Geräten und Prozessen Anwendung. Ihre Leistung und Ausstattung werden für ihren jeweilige Einsatzzweck angepasst, wodurch die Kosten, das Format und die Leistungsaufnahme optimiert werden können. Das Programmieren von Mikrocontrollern stellt, verglichen mit dem herkömmlicher Computersysteme, zusätzliche Anforderungen an die Entwickler. Dazu zählt, dass hardwarenahe Programmiersprachen wie C oder Assembler benutzt werden, sowie der Umgang mit Ressourcenknappheit.

In diesem Versuch werden zum Einstieg in die Mikrocontrollerprogrammierung mehrere elementare Applikationen implementiert.

Hinweis

Die Versuchsdokumente des Praktikums bestehen jeweils aus einem Theorie- und einem Aufgabenteil. Der Theorieteil dient der Einführung in die Thematik, damit die Details der gestellten Aufgaben nicht noch in der Aufgabenbeschreibung erläutert werden müssen. Es empfiehlt sich dringend während des gesamten Praktikums immer erst das gesamte Dokument komplett durchzulesen, bevor man mit der Implementierung beginnt.

1.1 Versuchsinhalte

Dieser Versuch beginnt mit der Vorstellung des im Praktikum verwendeten Evaluationsboards und des darauf befindlichen Mikrocontrollers ATmega 644. Anschließend gliedert sich der Versuch in zwei Teile. Der erste Teil stellt eine Einführung in die Mikrocontrollerprogrammierung dar.
Zuerst wird eine für Mikrocontroller typische Anwendung, die Analog-Digital- und Digital-Analog-Wandlung (AD/DA-Wandlung) implementiert. Dabei wird neben dem Evaluationsboard eine weitere Platine (siehe Abbildung 1.2) verwendet, welche die für einen AD/DA Wandler benötige Hardware bereitstellt.
Im zweiten Teil des Versuchs werden weitere Komponenten des ATmega 644 vorgestellt und relevante Konzepte der Mikrocontrollerprogrammierung vertieft. Dazu wird eine Applikation mit mehreren einfachen Unterprogrammen implementiert. Hierzu zählt beispielsweise eine Binäruhr und die Ausgabe von verschiedenen Informationen auf einem LED-Bargraph.
Für einen sanften Einstieg in die Softwareentwicklung auf dem ATmega 644 wird empfohlen, das *Begleitende Dokument zum Praktikum Systemprogrammierung* zu lesen. In diesem Dokument werden grundlegende Programmierkonzepte der Sprache C sowie eine Vielzahl von Besonderheiten in der Mikrocontrollerprogrammierung vorgestellt.

1.2 Lernziel

Das Lernziel dieses Versuchs ist das Verständnis der folgenden Zusammenhänge:

- Umgang mit der Entwicklungsumgebung Microchip Studio 7
- Grundlagen der Programmiersprache C
- Programmierung eines Mikrocontrollers
- AD/DA-Wandlung
- Verwendung von *Interrupts*

1.3 Grundlagen

Dieser Abschnitt stellt grundlegendes Wissen dar, das nötig ist, um die Haus- und Präsenzaufgaben erfolgreich bearbeiten zu können.

1.3.1 Einführung in die Verwendung des ATmega 644

Im Folgenden werden zunächst grundlegende Informationen zur Softwareentwicklung für eingebettete Systeme vorgestellt. Anschließend wird das Evaluationsboard und der darauf platzierte 8-Bit Mikrocontroller ATmega 644 eingeführt.

Softwareentwicklung für Atmel-Mikroprozessoren

Software für Mikroprozessoren der Firma Atmel kann in verschiedenen Programmiersprachen entwickelt werden. Im Rahmen des Praktikums Systemprogrammierung werden ausschließlich die Sprachen C und Assembler verwendet.
Ein C-Programm für einen Mikrocontroller unterscheidet sich nicht grundlegend von einem Programm für ein übliches Computersystem. Es wird sequenziell abgearbeitet und verfügt über eine sogenannte `main`-Methode, welche zu Beginn der Ausführung automatisch aufgerufen wird und den Programmeinstiegspunkt darstellt.
Um ein Programm auf den Mikrocontroller zu übertragen und zu debuggen, wird im Rahmen des Praktikums ein *JTAG Programmierer* verwendet. Als Entwicklungsumgebung stellt Microchip das Microchip Studio 7 kostenlos zur Verfügung. Einen entsprechenden Installer finden Sie im Moodle-Lernraum des Praktikums. Für weitere Informationen zu Microchip Studio 7 und C sei auf die Kapitel *3: Einführung in Microchip Studio 7* und Kapitel *5: Einführung in die C Programmierung* im *Begleitenden Dokument zum Praktikum Systemprogrammierung* verwiesen.

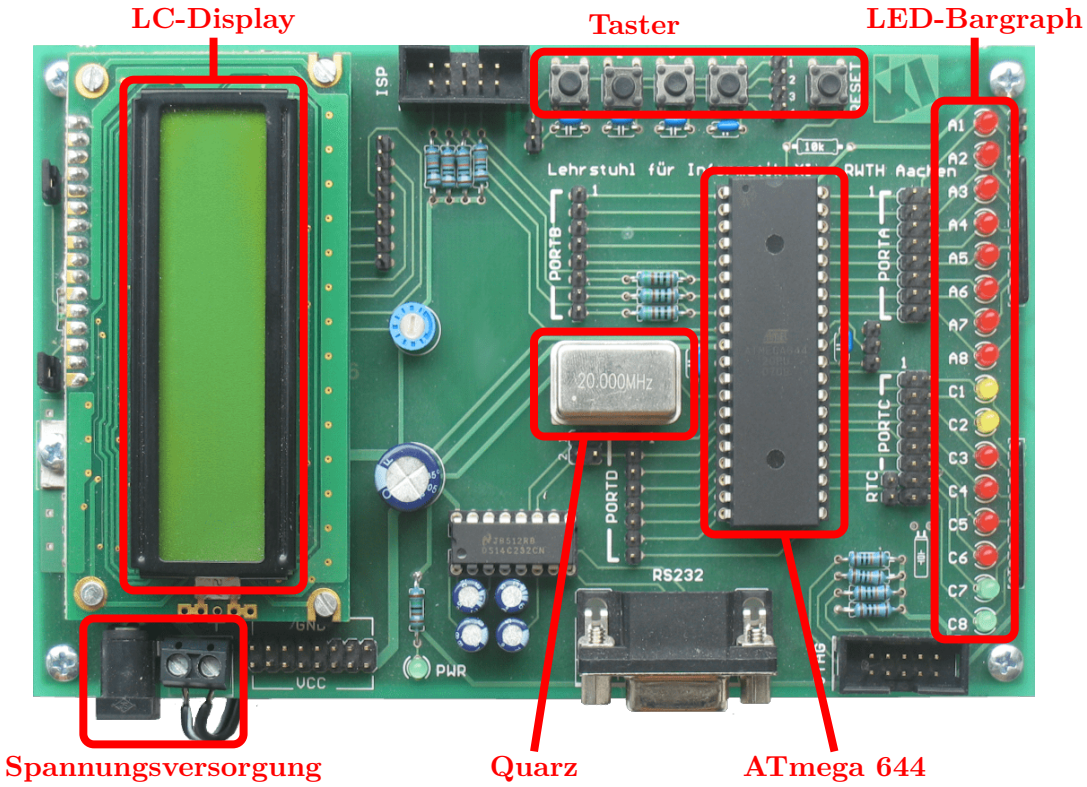

Abbildung 1.1: Das Evaluationsboard des ATmega 644

Informationen zum Evaluationsboard des ATmega 644

Im Praktikum Systemprogrammierung wird ein 8-Bit Mikrocontroller der Firma Atmel verwendet – der ATmega 644. Dieser Mikrokontroller ist auf dem Evaluationsboard des Praktikums mit 20MHz getaktet und besitzt verschiedene interne Komponenten, wie zum Beispiel EEPROM-Speicher, AD/DA-Wandler, vier I/O-Ports, eine USART Schnittstelle und drei Timer. Darüber hinaus verfügt er über einen Interrupt-Controller. Das heißt, dass unabhängig vom aktuell ausgeführten Code Ereignisse erkannt und auf diese reagiert werden kann (d.h. ein *Interrupt* eintritt), indem die Ausführung des Hauptprogramms unterbrochen wird. Diese Ereignisse können extern auftreten, beispielsweise in Form einer Pegeländerung an einem Pin, oder intern, beispielsweise wenn ein spezielles Register einen bestimmten Wert erreicht hat.
Ein Evaluationsboard ist eine Platine, die es erlaubt, einen Mikrocontroller bequem programmieren und debuggen zu können, ohne zuerst eine Schaltung für diesen zu entwickeln. Das Evaluationsboard des Lehrstuhls für Informatik 11 der RWTH Aachen (Abbildung 1.1) bietet eine komfortable Arbeitsumgebung für den Umgang mit dem Mikrocontroller und stellt Pinleisten bereit, über die dieser leicht mit Eingabe- und Ausgabegeräten (LCD, Buttons, LEDs etc.) verbunden werden kann.

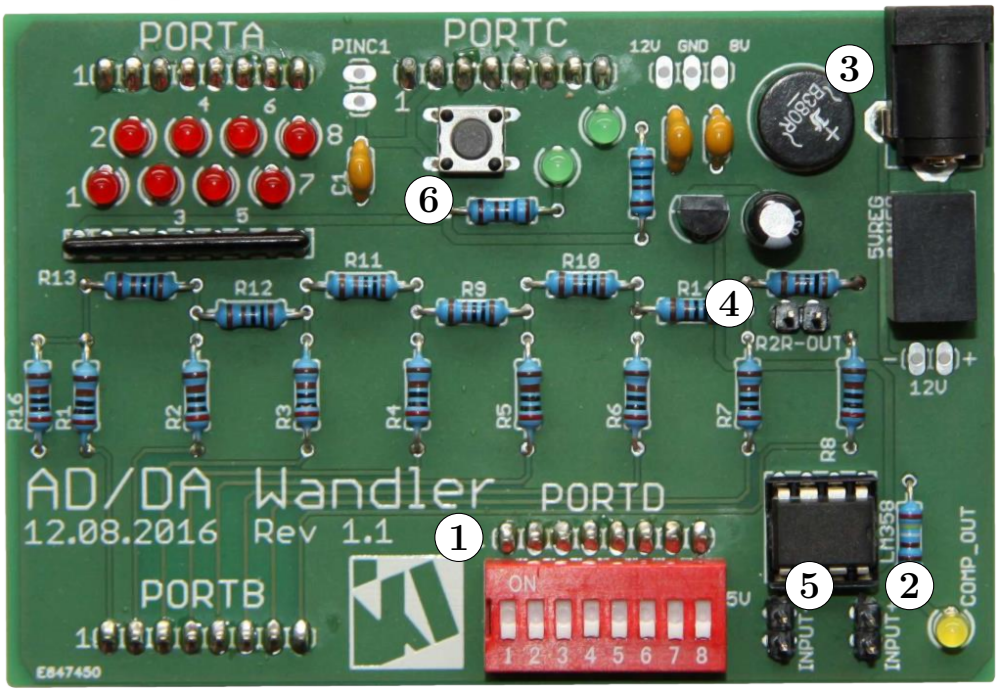

Abbildung 1.2: Platine mit Hardware zur AD/DA-Wandlung

Hinweis

Eine ausführlichere Beschreibung zu den Komponenten und deren Verwendung kann Kapitel *2.1: Der Mikrocontroller ATmega 644* im *Begleitenden Dokument zum Praktikum Systemprogrammierung* entnommen werden.

Konfiguration von Ports

Der ATmega 644 verfügt über 40 Pins, von denen 32 als Ein-/Ausgabe Pins verwendet werden können. Ist ein Pin als Ausgang konfiguriert, kann der Mikrocontroller die Spannung 5V (logisch 1) oder 0V (logisch 0) ausgeben. Ist ein Pin als Eingang konfiguriert, kann der Mikrocontroller feststellen, ob an diesem Pin ein Spannungspegel von 5V oder 0V angelegt wurde.
Die Pins sind auf vier Organisationseinheiten aufgeteilt: Die Ports A, B, C und D (siehe Abbildung 1.1). Die zu einem Port gehörenden Pins sind von 0 bis 7 durchnummeriert. Jeder dieser Pins kann über spezielle Prozessorregister konfiguriert werden. Für jeden Port existieren sogenannte *Data Direction*-, *Port Output*- und *Port Input*-Register (`DDR`, `PORT` und `PIN`), welche zur Konfiguration der Pins dienen. Diese Register können analog

zu Variablen ausgelesen und verändert werden. Sie sind jeweils 8 Bit breit, da jedes Bit einen Pin des jeweiligen Ports steuert.
Für die Konfiguration dieser Register ist die Manipulation einzelner Bits nötig. Diesbezüglich können weitere Informationen in Kapitel *5.1.16: Registerzugriff, Bitshifting & Bitmasken* im *Begleitenden Dokument zum Praktikum Systemprogrammierung* nachgelesen werden. In Listing 1.1 wird veranschaulicht, wie der Spannungspegel (5V oder 0V) des Pin C1 ausgelesen werden kann. Weitere Informationen zu der Bedeutung der Register können auch dem einführenden Abschnitt des Versuch 0-Dokuments entnommen werden.

```
1 // 1. Pin C1 als Eingang konfigurieren (Bit 1 von DDRC auf 0
     setzen)
2 DDRC  &= 0b11111101;
3
4 // 2. Pullup-Widerstand an Pin C1 aktivieren (Bit 1 von PORTC
     auf 1 setzen)
5 PORTC |= 0b00000010;
6
7 // 3. Pin C1 auslesen (Bit extrahieren und nach rechts
     verschieben)
8 uint8_t pinState = (PINC & 0b00000010) >> 1;
```

Listing 1.1: Zustand von Pin C1 wird abgefragt

Das nächste Beispiel zeigt die Verwendung von Pin `C1` als Ausgang. Nach Ausführung des Codes liegt an Pin `C1` eine Spannung von 5V an.

```
1 // 1. Pin C1 als Ausgang konfigurieren (Bit 1 von DDRC auf 1
     setzen)
2 DDRC  |= 0b00000010;
3
4 // 2. Logische 1 an Pin C1 ausgeben (Bit 1 von PORTC auf 1
     setzen)
5 PORTC |= 0b00000010;
```

Listing 1.2: Augabe einer logischen 1 an Pin C1

Buttons

In diesem Praktikum werden Buttons verwendet, um Eingaben an ein auf dem Mikrocontroller laufendem Programm tätigen zu können. Das Evaluationsboard verfügt über vier Buttons, die beliebig mit Pins verkabelt werden können. Des Weiteren kommt in diesem Versuch eine spezielle Platine zum Einsatz, die für die Realisierung des manuellen AD/DA-Wandlers nötig ist (siehe Abbildung 1.2). Diese verfügt ebenfalls über einen Button, welcher fest mit Pin `C1` verbunden ist.
Um den Zustand eines Buttons auslesen zu können, muss zunächst der entsprechende Pin

als Eingang konfiguriert werden. Bei Buttons ist es zudem notwendig, sogenannte *Pullup*-Widerstände für die jeweils genutzten Pins zu aktivieren, um zwischen einem gedrückten und nicht-gedrückten Zustand unterscheiden zu können. Im Anschluss daran kann der Zustand des Buttons aus dem zugehörigen `PIN`-Register ausgelesen werden. Hierbei ist zu beachten, dass das entsprechende Bit im `PIN`-Register im gedrückten Zustand den Wert 0 und im nicht-gedrückten Zustand den Wert 1 hat. Weitere Informationen hierzu können dem Kapitel *2.1.1: Input Pins im Begleitenden Dokument zum Praktikum Systemprogrammierung* entnommen werden.

Interrupts

Ein Interrupt ist eine automatische, ereignisgesteuerte Unterbrechung des Programmablaufs. Der ATmega 644 verfügt über eine Vielzahl von konfigurierbaren Interrupts. Eine Möglichkeit, Interrupts intern auszulösen, stellen Timer dar. Sie können beispielsweise so konfiguriert werden können, dass sie in äquidistanten Zeitabständen einen Interrupt auslösen.
Um auf Interrupts zu reagieren, müssen sogenannte *Interrupt Service Routinen* (ISRs) implementiert werden, welche als Behandlungsfunktionen für diese Ereignisse fungieren. Nach Abarbeitung einer ISR wird der normale Programmablauf an der Stelle fortgesetzt, an der dieser durch den Aufruf der ISR unterbrochen wurde.

LC-Display

Zusammen mit den Versuchsunterlagen wird ein Treiber für das LC-Display des Evaluationsboards zur Verfügung gestellt. Folgende Tabelle zeigt einige Schnittstellenfunktionen des LCD-Treibers:

Befehl	**Bedeutung**	**Beispiel**
`lcd_init(void)`	Initialisierung	`lcd_init();`
`lcd_clear(void)`	Löscht das Display. Das nächste Zeichen wird in der ersten Spalte und Zeile ausgegeben.	`lcd_clear();`
`lcd_writeChar(char)`	Gibt ein einzelnes Zeichen aus.	`lcd_writeChar('!');`
`lcd_writeDec(uint16_t)`	Gibt eine Ganzzahl ohne führende Nullen aus.	`lcd_writeDec(471);`
`lcd_writeString(const char*)`	Gibt eine Zeichenkette aus.	`lcd_writeString("xy");`
`lcd_writeVoltage( uint16_t, uint16_t, uint8_t)`	Gibt eine Spannung in Gleitkommadarstellung aus	`lcd_writeVoltage(113, 255, 5);`

In diesem Versuch wird die `lcd_writeVoltage`-Funktion benötigt, um einen diskreten Spannungswert in einen Spannungswert in Gleitkommadarstellung umzuwandeln. Die Funktion kann folgendermaßen verwendet werden: Der auszugebene Spannungswert wird als diskreter Wert im ersten Parameter übergeben, während der zweite Parameter

die obere Schranke des ersten Parameters angibt. Der dritte Parameter gibt die obere Schranke der Spannung an. `lcd_writeVoltage(113, 255, 5)` erzeugt beispielsweise die Ausgabe $\frac{113}{255} \cdot 5\text{V} = \texttt{2.215V}$.

Lernerfolgsfragen

- Was muss vor der ersten Verwendung des Displays sichergestellt werden?
- Wie kann das Display gelöscht werden? Wie wird es mit Buchstaben, Zahlen oder Zeichenketten beschrieben?

Debugging

Es wird empfohlen, das Kapitel *6.2: Debugging-Methoden* im *Begleitenden Dokument zum Praktikum Systemprogrammierung* zu lesen. In diesem werden Methoden vorgestellt, welche bei einer möglichen Fehlersuche hilfreich sind. Dazu gehören beispielsweise *Breakpoints* oder das Auslesen des Speichers und der aktuellen Konfiguration der Prozessorregister.

Verkabelung des Evaluationsboards

Auf den letzten Seiten dieses und folgender Versuchsdokumente ist jeweils eine Übersicht über die Pinbelegung des aktuellen Versuchs zu finden. Zudem ist zu beachten, dass die Mikrocontroller im Testpool, falls nicht ausdrücklich anders angegeben, stets gemäß dieser Tabelle verkabelt sind.

1.3.2 AD/DA-Wandlung

In diesem Abschnitt werden zunächst grundlegende Informationen zum Thema Analog-Digital- bzw. Digital-Analog-Wandlung (AD/DA-Wandlung) vorgestellt. Anschließend werden Grundlagen zu Komparatoren und R-2R-Netzwerken erläutert. Beides sind elektrische Schaltungen, die für die in diesem Versuch vorgestellte AD/DA-Wandlung benötigt werden. Abschließend werden Methoden vorgestellt, um eine AD/DA-Wandlung durchzuführen.

AD/DA-Wandler

Ein Digital-Analog-Wandler (DA-Wandler, engl. Digital-to-Analog Converter (DAC)) konvertiert quantisierte digitale Signale in analoge Signale. Ein Analog-Digital Wandler (AD-Wandler, engl. Analog-to-Digital-Converter (ADC)) hingegen, setzt analoge Eingangssignale in digitale Daten um. AD/DA-Wandler werden unter anderem in der Messtechnik oder Audio-Video-Verarbeitung eingesetzt. In diesen Anwendungen liegen die

Eingangsgrößen analog bzw. digital vor und sollen in das jeweils andere Format konvertiert werden. Ein Beispiel hierfür ist die Aufnahme und Wiedergabe von Musik auf einem Computer.

Um ein zeit- und wertdiskretes digitales Signal aus einem kontinuierlichen analogen Signal zu erzeugen, wird das Ursprungssignal zunächst an festen Zeitpunkten abgetastet. Die Abtastung einer analogen Spannung benötigt eine bestimmte Zeitspanne, da eine AD-Wandlung in den meisten Fällen durch eine iterative Approximation der zu messenden Spannung realisiert wird. Je kürzer diese Zeitspanne ist, desto größer ist die sogenannte *Umsetzungsgeschwindigkeit* des Wandlers.
Ist die Umsetzungsgeschwindigkeit eines Wandlers zu niedrig und damit die Abtastzeitpunkte zu weit voneinander entfernt, kann der ursprüngliche Signalverlauf nicht aus dem digitalen Signal rekonstruiert werden, was als *Alias-Effekt* bezeichnet wird. Um dies zu verhindern ist nach dem Abtasttheorem von Nyquist und Shannon ein Wandler mit einer Umsetzungsgeschwindigkeit nötig, welche mindestens doppelt so hoch ist wie die maximale Frequenz des abzutastenden Signals.
Im Anschluss an die Abtastung müssen die abgetasteten analogen Spannungswerte auf diskrete Werte gerundet werden. Dieser Vorgang heißt *Quantisierung*. Die dabei entstehenden diskreten Werte heißen Quantisierungsstufen. Je mehr Quantisierungsstufen zur Verfügung stehen, desto geringer ist der auftretende Rundungsfehler und desto genauer wird das ursprüngliche Signal repräsentiert. Die Quantisierungsstufen werden üblicherweise in dem erwarteten Wertebereich des analogen Signals gleichverteilt. Nach der Abtastung und Quantisierung können die Signalwerte codiert werden, indem jeder Quantisierungsstufe eine Zahl zugeordnet wird. Das analoge, zeitkontinuierliche Ursprungssignal wird somit durch eine digitale, zeitdiskrete Folge von Zahlen repräsentiert.

Beispiel Bei einer Quantisierung von 5V auf 256 Werte erhält man eine Quantisierungsstufe von $5\,\mathrm{V}/256 \approx 20\,\mathrm{mV}$. Die Codierung der Binärzahl `0b01111011` (dezimal 123) in eine analoge Spannung ist demnach rund $2{,}46\,\mathrm{V} = 123 \cdot 20\,\mathrm{mV}$.

Die wichtigsten Kenngrößen eines AD/DA-Wandlers sind seine Auflösung und die Umsetzungsgeschwindigkeit. Die Auflösung gibt an, wie viele Bits zur Codierung der Quantisierungsstufen zur Verfügung stehen. Eine Auflösung von 8 Bit bedeutet beispielsweise, dass $2^8 = 256$ Quantisierungsstufen zur Darstellung existieren. Dies entspricht einer Genauigkeit von 0,39 % ($1/256$) bezogen auf den Messbereich.

Lernerfolgsfragen

- Wozu dienen AD/DA-Wandler?
- Was passiert bei der Abtastung eines Signals?
- Was passiert bei der Quantisierung eines Signals?
- Welche sind die wichtigsten Kenngrößen von AD/DA-Wandlern?

Komparator

Durch die in Abbildung 1.3 gegebene Verschaltung kann ein Komparator mit Hilfe eines Operationsverstärkers konstruiert werden. Am Ausgang des Komparators steht ein Signal zur Verfügung, das anzeigt, welche der beiden Eingangsspannungen U_e bzw. U_{ref} höher ist. Ist die Spannung am positiven Eingang höher als die Spannung am negativen Eingang, so entspricht die Ausgangsspannung annähernd der positiven Versorgungsspannung. Im gegenteiligen Fall ($U_{ref} < U_e$) entspricht die Ausgangsspannung der negativen Versorgungsspannung. Somit können selbst minimale Unterschiede der beiden Eingangsspannungen erkannt werden. Um die Ausgangsspannung des Komparators einfach ablesen zu können befindet sich auf dem AD/DA-Board eine mit COMP_OUT beschriftete LED, an der die Ausgangsspannung anliegt.

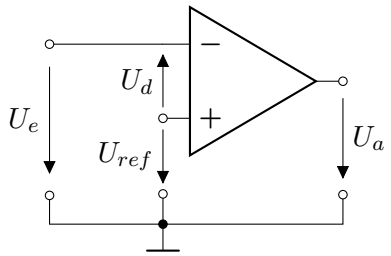

Abbildung 1.3: Komparatorschaltung

Im Folgenden werden das R-2R-Netzwerk als DA-Wandler, und der Tracking-Wandler und Sukzessive-Approximation-Wandler als AD-Wandler vorgestellt.

DA-Wandler: R-2R-Netzwerk

Die einfachste Methode einer DA-Wandlung ist ein R-2R-Netzwerk. Dies ist das einzige in diesem Versuch genutzte Verfahren für die Umwandlung von digitalen Signalen in analoge Spannungen. Ein R-2R-Netzwerk verwendet nur Widerstände der Größen R und $2R$. Abbildung 1.4 zeigt die Verschaltung eines n-Bit R-2R-Netzwerks.

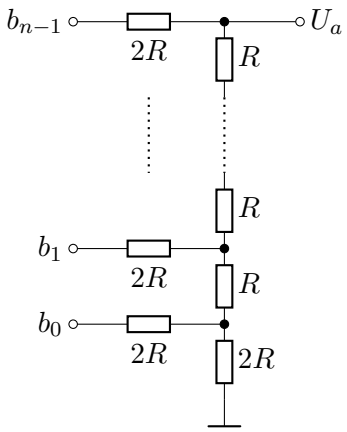

Abbildung 1.4: R-2R Widerstandsnetzwerk

An den Eingängen $b_0, ..., b_{n-1}$ liegt jeweils zur Darstellung einer digitalen Eins die Betriebsspannung an, zur Darstellung einer digitalen Null werden die Eingänge auf Masse gelegt. Die Ausgangsspannung U_a kann wie folgt berechnet werden:

$$U_a = V_{cc} \cdot (\frac{b_{n-1}}{2^1} + \frac{b_{n-2}}{2^2} + \frac{b_{n-3}}{2^3} + ... + \frac{b_0}{2^n})$$

Hierbei stellt b_i jeweils den logischen Wert des Eingangs i dar.
Durch unterschiedliche Beschaltung der Eingänge $b_0, ..., b_{n-1}$ kann die analoge Ausgangsspannung U_a variiert werden.

AD-Wandler: Tracking-Wandler

Der Tracking-Wandler ist ein Analog-Digital-Wandler, der auf eine von einem DA-Wandler erzeugte Referenzspannung zurückgreift, um die zu messende Spannung zu ermitteln. Die quantisierte Referenzspannung wird dabei in einer n-Bit Variable ref festgehalten, welche zu Beginn mit einem beliebigen Wert initialisiert werden kann. Der interne Aufbau eines Tracking-Wandlers ist in Abbildung 1.5 dargestellt und wird im Folgenden erläutert. Die generierte Referenzspannung wird mit der zu messenden Spannung mit Hilfe eines Komparators verglichen. Ist die Messspannung höher (Komparator K = 0) als die Referenzspannung, inkrementiert ein Up-Down-Counter die Variable ref, andernfalls wird ref dekrementiert. Nach Inkrementierung bzw. Dekrementierung wird der neue Wert von ref als neue Referenzspannung ausgegeben und das Ergebnis des Komparators neu ausgewertet. Die Wandlung gilt als abgeschlossen, wenn die Referenzspannung die Messspannung über- oder unterschritten hat. In diesem Fall ist sichergestellt, dass sich die zu messende Spannung zwischen den letzten beiden Referenzspannungen befindet. Da der Tracking-Wandler im schlechtesten Fall 2^n Schritte benötigt, eignet sich dieser nicht zur Messung von großen Spannungsänderungen. Spannungsverläufe, die einer nur geringen Veränderungsrate unterliegen, können hingegen effizient mit dem Tracking-Wandler digitalisiert werden.

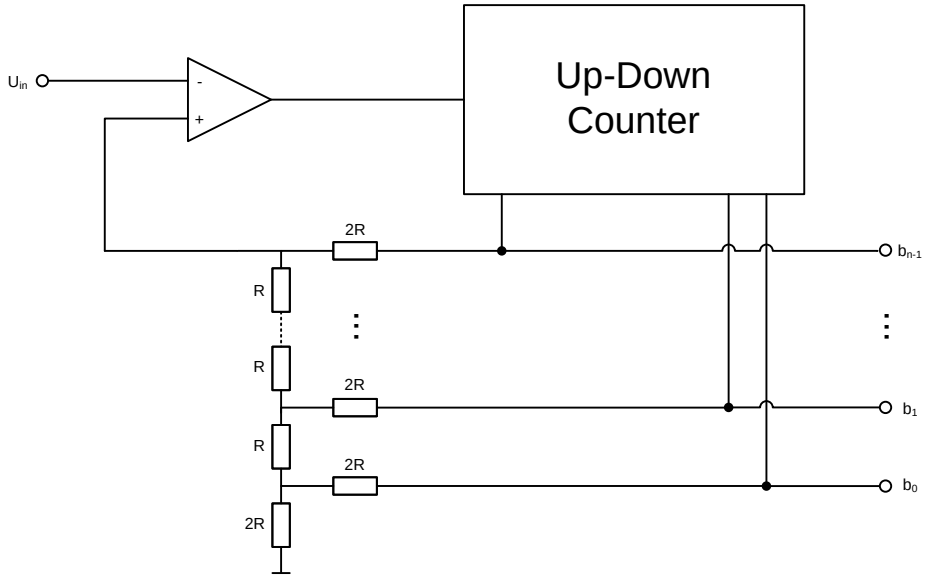

Abbildung 1.5: Tracking-Wandler

AD-Wandler: Sukzessive-Approximation-Wandler

Eine weitere Möglichkeit zur Realisierung einer Analog-Digital-Wandlung ist die sukzessive Approximation. Dieses Verfahren arbeitet nach dem Prinzip der Binärsuche. Dabei wird ein Sukzessive-Approximation-Register (SAR, Abbildung 1.6) verwendet. Die Funktionsweise des SAR zeigt Abbildung 1.7 für einen 3 Bit-Wandler. Die Referenzspannung wird, wie beim Tracking-Wandler, in einer n-Bit Variable ref festgehalten. Zu Beginn einer Wandlung wird die Referenzspannung auf 0 gesetzt. Im ersten Schritt wird testweise das „Most Significant Bit“ (MSB) der Variable ref auf 1 gesetzt. Anschließend wird der Wert von ref als Referenzspannung mit Hilfe des DA-Wandlers ausgegeben. Wie beim Tracking-Wandler vergleicht ein Komparator die Mess- und Referenzspannung. Ist die Messspannung größer (Komparator K = 0) als die Referenzspannung, wird die Eins beibehalten, andernfalls (K = 1) wird das Bit auf 0 zurückgesetzt. Danach wird analog mit dem nächstniedrigerwertigem Bit verfahren. Die Wandlung ist beendet, wenn alle Bits überprüft wurden. Der nach Überprüfung aller Bits in ref gespeicherte Wert ist das Ergebnis der Wandlung. Für eine komplette Wandlung werden somit stets n Schritte benötigt.

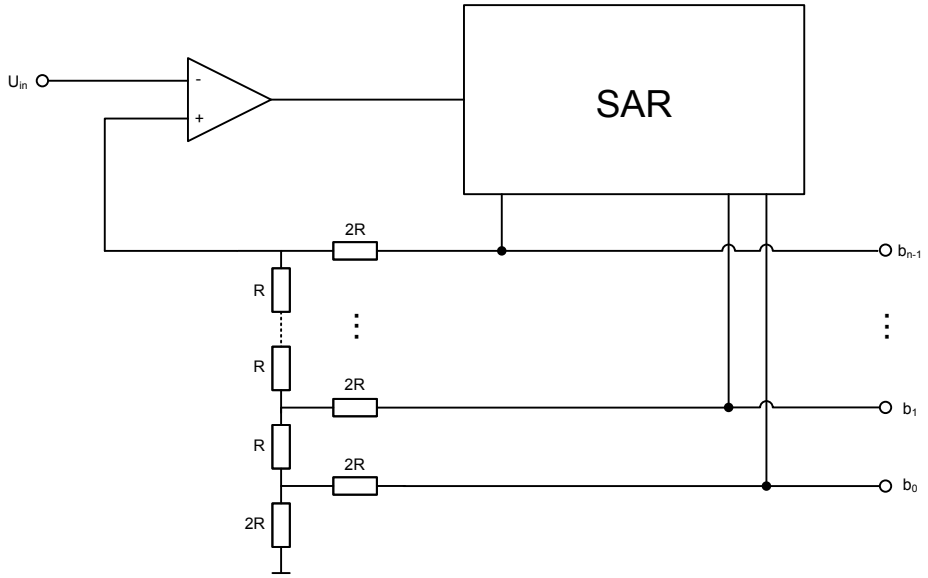

Abbildung 1.6: Sukzessive-Approximation-Wandler

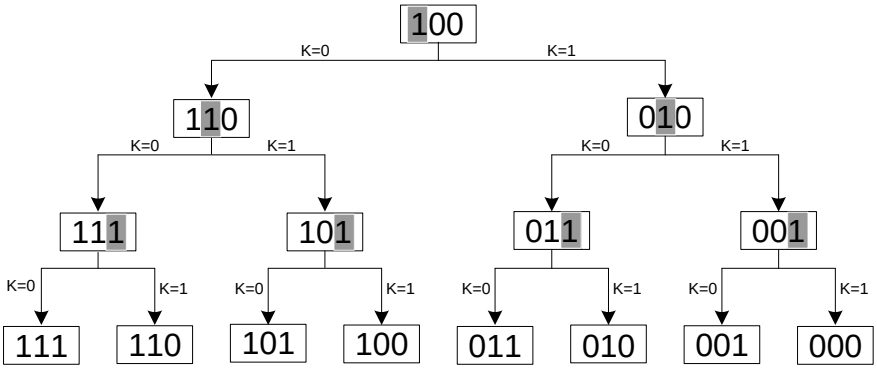

Abbildung 1.7: SAR Entscheidungsbaum für n = 3 (K ist das Ergebnis des Komparators)

Lernerfolgsfragen

- Warum wird bei der sukzessiven Approximation das MSB zuerst testweise gesetzt?
- Benötigt ein Sukzessive-Approximation-Wandler immer weniger Schritte als ein Tracking-Wandler?

1.4 Hausaufgaben

Implementieren Sie die in den nächsten Abschnitten beschriebenen Funktionalitäten. Halten Sie sich an die hier verwendeten Namen und Bezeichnungen für Variablen, Funktionen und Definitionen.
Lösen Sie alle hier vorgestellten Aufgaben zu Hause mithilfe von Microchip Studio 7 und schicken Sie die dabei erstellte und funktionsfähige Implementierung über Moodle ein. Ihre Abgabe soll dabei die `.atsln`-Datei, das Makefile, sowie den Unterordner mit den `.c/.h`-Dateien inklusive der `.xml/.cproj`-Dateien enthalten. Beachten Sie bei der Bearbeitung der Aufgaben die angegebenen Hinweise zur Implementierung! Ihr Code muss ohne Fehler und ohne Warnungen kompilieren sowie die Testtasks mit aktivierten Compileroptimierungen bestehen. Wie Sie die Optimierungen einschalten ist dem begleitenden Dokument in Abschnitt *6.3.2 Probleme bei der Speicherüberwachung* zu entnehmen.

Achtung

Verwenden Sie zur Prüfung auf Warnungen den Befehl „Rebuild Solution“ im „Build“-Menü des Microchip Studio 7. Die übrigen in der grafischen Oberfläche angezeigten Buttons führen nur ein inkrementelles Kompilieren aus, d.h. es werden nur geänderte Dateien neu kompiliert. Warnungen und Fehlermeldungen in unveränderten Dateien werden dabei nicht ausgegeben.

1.4.1 Allgemeines

Folgende Dokumente, welche im Moodle-Lernraum des Praktikums verfügbar sind, dienen dem grundlegenden Verständnis:

- *Bedienungsanweisung für das Praktikum Systemprogrammierung* - Dieses Dokument stellt Ihnen die sichere Bedienung des Multimeters und der Spannungsquelle vor, die Sie für die Präsenzaufgaben benötigen.
- *Begleitendes Dokument für das Praktikum Systemprogrammierung* - Dieses Dokument enthält alle Informationen, die Sie für den Einstieg in die C-Programmierung eines Mikrocontrollers benötigen. Es dient als grundlegendes Nachschlagewerk und beinhaltet eine Beschreibung aller für dieses Praktikum benötigten Informationen.

1.4.2 Aufgaben zur AD/DA-Wandlung

In diesem Teil des Versuchs sollen Sie die im Grundlagenkapitel vorgestellten Wandler implementiert. Beachten Sie, dass für diesen Teil des Versuchs eine AD/DA-Platine auf das Evaluationsboard aufgesetzt wird (siehe Abbildung 1.2). Die Pin-Belegung für diesen Aufgabenteil können Sie Tabelle 1.7 am Ende des Versuchsdokuments entnehmen. Das LC-Display wird für diesen Teil des Versuchs nicht verwendet.
Installieren Sie die im *Begleitenden Dokument* vorgeschriebene Version von Microchip Studio 7. Öffnen Sie anschließend das bereitgestellte Codegerüst für den AD/DA-Teil des Versuchs.

Entwickeln Sie für den ATmega 644 ein Programm, welches folgende Funktionalitäten enthält:

1. Eine manuelle Ansteuerung des R-2R Netzwerks
2. Einen Sukzessive-Approximation-Wandler
3. Einen Tracking-Wandler

Fügen Sie für jede der oben aufgezählten Funktionalitäten eine Funktion hinzu. Die manuelle Ansteuerung kann beispielsweise als `void manuell(void)` realisiert werden.
Fügen Sie neben den drei zu implementierenden Funktionen eine `int main(void)`-Funktion hinzu. In dieser soll je nach Bedarf eines der drei Unterprogramme ausgewählt werden. Eine `switch-`Case Anweisung eignet sich zur Selektierung des gewünschten Unterprogramms, welche eine zuvor deklarierte Variable auswertet und entsprechend ihres Wertes eines der Unterprogramme startet. Es genügt, das zu startende Unterprogramm im Sourcecode festlegen zu können. Selbstverständlich steht es Ihnen frei, beliebige weitere Hilfsprogramme anzulegen.

DA-Wandlung: Ansteuerung des R-2R-Netzwerks

Beginnen Sie mit der Funktion zur manuellen Ansteuerung der AD/DA-Platine. In dieser Funktion soll der durch den DIP-Schalter (Markierung 1 in Abbildung 1.2) eingestellte Bit-Wert intern im Mikrocontroller auf das R-2R-Netzwerk abgebildet werden. Legen Sie diesen Wert ebenfalls an die LEDs an, um zu überprüfen, welche Bitkombination aktiv ist. Hierzu konfigurieren Sie zunächst Port D (DIP-Schalter) als Eingang. Entsprechend müssen Port A (LEDs) und Port B (R-2R-Netzwerk) als Ausgang konfiguriert werden. Lesen Sie anschließend den Wert an `PIND` ein und legen Sie diesen geeignet an `PORTA` und `PORTB`. Beachten Sie, dass im entsprechenden Bit in `PIND` eine 0 steht, wenn der zugehörige Dip auf `ON` geschaltet ist.
Achten Sie auf die korrekte Initialisierung aller relevanten Register. Beachten Sie, dass Änderungen am DIP-Schalter zur Laufzeit direkt umgesetzt werden sollen. Der Mikrocontroller soll den neu eingestellten Wert kontinuierlich am R-2R-Netzwerk und an die LEDs ausgeben, ohne dass ein Neustart des Mikrocontrollers nötig ist.

AD-Wandlung: Tracking-Wandler und SAR

Implementieren Sie als Nächstes den Tracking-Wandler und den SA-Wandler. Beide wurden im Grundlagenkapitel vorgestellt und sollen gemäß dieser Beschreibung programmiert werden. Verwenden Sie zur Erzeugung der Referenzspannung das R-2R Netzwerk, dessen Ausgang mit dem Komparator verbunden ist. Die zu messende Spannung wird von einer externen Spannungsquelle erzeugt, welche an der AD/DA-Platine angeschlossen wird. Für den auf der AD/DA-Platine befindlichen Komparator, gilt folgendes Verhalten:

- $U_{REF} < U_{MESS} \Longrightarrow$ Pin `C0 = 0`
- $U_{REF} > U_{MESS} \Longrightarrow$ Pin `C0 = 1`

Hinweis

Aufgrund der hohen Ausführungsgeschwindigkeit des Mikrocontrollers ist es notwendig, nach jeder Änderung am R-2R-Netzwerk eine Mindestzeit zu warten, bis das Ergebnis des Komparators ausgelesen wird. Wählen Sie eine Wartezeit von 50 ms.

Benutzen Sie zum Warten die Delay-Funktionen aus der Bibliothek `util/delay.h`, welche mit einem entsprechenden `include`-Befehl in die Sourcecodedatei eingebunden werden muss. Informationen über das Einbinden von Bibliotheken können dem *Begleitenden Dokument zum Praktikum Systemprogrammierung* in Kapitel *5.1.2* entnommen werden.

Nachdem diese sogenannte *Headerdatei* eingebunden wurde, können folgende Funktionen genutzt werden:

- `void _delay_us(double __us)`
- `void _delay_ms(double __ms)`

Geben Sie während der Wandlung stets die aktuelle Referenzspannung auf die an Port A angeschlossenen LEDs aus, damit der Wandlungsvorgang visuell nachvollzogen werden kann.

Implementieren Sie beide Wandler so, dass eine vollständige Wandlung der aktuell angelegten Spannung erst dann erfolgt, wenn der Button an Pin `C1` gedrückt wurde. Der Button befindet sich auf der AD/DA Platine. Nachdem eine komplette Wandlung durchgeführt wurde, soll das Programm erst bei erneutem Tastendruck die nächste Wandlung durchführen.
Da in diesem und kommenden Versuchen häufig mit Buttons gearbeitet wird, bietet es sich an, ein universell einsetzbares Modul zur Benutzung zu implementieren. Ihnen werden für diese Aufgabe die Dateien `os_input.c` und `os_input.h` vorgegeben, welche die Struktur dieses Moduls beinhalten. Die Funktion `os_initInput` dient zur Konfiguration des entsprechenden Ports, mit dem der Button verbunden werden soll. Verwenden Sie die Funktion `os_getInput` zur Zustandsabfrage der Buttons. Achten Sie hierbei besonders auf die Erklärung im Funktionskommentar, der verschiedene Szenarien für den Rückgabewert beschreibt. Implementieren Sie die Funktionen `os_waitForInput` und `os_waitForNoInput`. Verwenden Sie diese in Ihrem Projekt dort, wo auf das Drücken bzw. Loslassen des Buttons gewartet werden soll.

1.4.3 Aufgaben zum ATmega 644

In diesem Hausaufgabenteil sollen weiterführende Funktionen für den ATmega 644 implementiert werden. Hierzu wird ein separates Projekt verwendet, welches Ihnen ebenfalls als eigenes Codegerüst zur Verfügung gestellt wird. Im Unterschied zum ersten Teil des Versuchs arbeiten Sie hier nur mit dem Evaluationsboard ohne die aufgesetzte AD/DA-Platine. In Tabelle 1.8 finden Sie die für diesen Versuchsteil geltende Pin-Belegung. Die in diesem Abschnitt erläuterten Aufgaben dienen dazu, weitere fundamentale Konzepte der Mikrocontrollerprogrammierung zu verstehen und mit deren Besonderheiten umzugehen. Im vorgegebenen Codegerüst ist bereits eine einfache Menüführung implementiert, über welche mehrere Unterprogramme ausgeführt werden können. Diese Unterprogramme sind jedoch noch nicht vollständig implementiert und müssen von Ihnen ergänzt werden.

Der Aufbau des vorgegebenen Codegerüsts ist im Folgenden beschrieben. In der `main`-Funktion des Projekts wird zuerst das Buttonmodul und der Treiber für das LC-Display, welches in dieser Aufgabe verwendet wird, initialisiert. Nach der Initialisierung wird das Menü auf dem LCD angezeigt. Die Implementierung des Menüs befindet sich in der Datei

`menu.c` und wird durch die Funktion `showMenu` realisiert. Nach der Auswahl eines Unterprogramms im Menü wird die Funktion `start(programIndex)` verwendet, um in die entsprechenden Unterprogramme zu verzweigen. Darüber hinaus initialisiert diese Funktion die für das entsprechende Unterprogramm nötigen Module. Die Unterprogramme sind so konzipiert, dass ein Zurückkehren in das Hauptmenü möglich ist.

Das Buttonmodul

Die erste Aufgabe besteht darin, das Buttonmodul zu implementieren, das zur Bedienung des Menüs verwendet wird. An den Pins `C0`, `C1`, `C6` und `C7` sind nun Buttons angeschlossen, welche als *Enter*, *Down*, *Up* und *ESC* bezeichnet werden. Vervollständigen Sie in der Datei `os_input.c` die Funktionen `os_initInput`, `os_getInput`, `os_waitForInput` und `os_waitForNoInput`. Es steht Ihnen frei, die aus dem ersten Versuchsteil vorhandene Implementierung dieser Funktionen zu übernehmen und anzupassen. Beachten Sie jedoch die für diesen Versuchsteil veränderte Anzahl und Verkabelung der Buttons. Die unteren vier Bits des Rückgabewertes der Funktion `os_getInput` sollen, entsprechend dem Kommentar im Codegerüst, den Status der Buttons repräsentieren.

Programm 1: HelloWorld

Als erstes Unterprogramm soll die Funktion `helloWorld` vervollständigt werden. Dieser Teil des Versuchs stellt eine Einführung in die Verwendung des LCD dar. Der LCD-Treiber wird bereits an mehreren Stellen im vorgegeben Codegerüst verwendet (siehe Funktion `showMenu`). Sie können diese als zusätzliche Anwendungsbeispiele heranziehen. Der Ablauf der Funktion `helloWorld` soll wie folgt sein:

1. Mit Hilfe des LCD-Treibers die Zeichenkette „Hallo Welt!“ auf dem LCD ausgeben
2. 500 ms warten
3. Die Zeichenkette wieder von dem LCD entfernen
4. Weitere 500 ms warten
5. Die Schritte 1. bis 4. solange wiederholen, bis der ESC-Button (Button 4) gedrückt wird

Sobald der ESC-Button gedrückt und losgelassen wurde, soll ins Hauptmenü zurückgekehrt werden. Verwenden Sie dazu Ihr Buttonmodul. Die Verwendung eines Interrupts ist nicht erforderlich.

Programm 2: Binäruhr

Das zweite Unterprogramm soll eine Binäruhr im 12-Stunden-Format realisieren. Die Binäruhr zeigt eine im Binärformat codierte Uhrzeit mit Hilfe der auf dem Evaluationsboard befindlichen LEDs an. Zusätzlich wird die Uhrzeit auf dem LCD ausgegeben. In

dieser Implementierung werden Stunden, Minuten und Sekunden auf der Binäruhr angezeigt. Die vorhandenen LEDs werden entsprechend in drei Bereiche unterteilt, welche mit diesen drei Größen assoziiert werden. Wird nun für jede Größe der Leuchtzustand der entsprechenden LEDs als Binärzahl interpretiert, so lässt sich daraus die aktuelle Uhrzeit ableiten. Da zur Ausgabe der Binäruhr nicht mehr als 16 LEDs zur Verfügung stehen, wird die Uhrzeit im 12-Stunden Format verwaltet, wie auf einer analogen Uhr. Anders als im 24-Stunden Format gibt es keine Uhrzeiten 0:00 bis 0:59; stattdessen werden 12:00 bis 12:59 verwendet. Die Zeitzählung fängt dementsprechend bei 12:00 an. Um 12:59 springt die Uhr direkt auf 1:00 und wird dann wie gewohnt wieder bis 12:59 hochgezählt.
Es werden jeweils sechs LEDs für den Wert der Sekunden und Minuten sowie vier LEDs für den Wert der Stunden verwendet. In Abbildung 1.9 ist exemplarisch die Ausgabe der Uhrzeit 5:26:53 auf den LEDs gezeigt. Des Weiteren finden Sie in Abbildung 1.1 ein allgemeines Schema zur Semantik der LEDs.
Zusätzlich zur Ausgabe auf den LEDs soll die Uhrzeit in der üblichen Dezimaldarstellung auf dem LCD ausgegeben werden. Da auf dem LCD mehr Platz zur Verfügung steht, werden hier zusätzlich Millisekunden angezeigt. Der Ablauf dieses Unterprogramms kann Abbildung 1.8 entnommen werden.

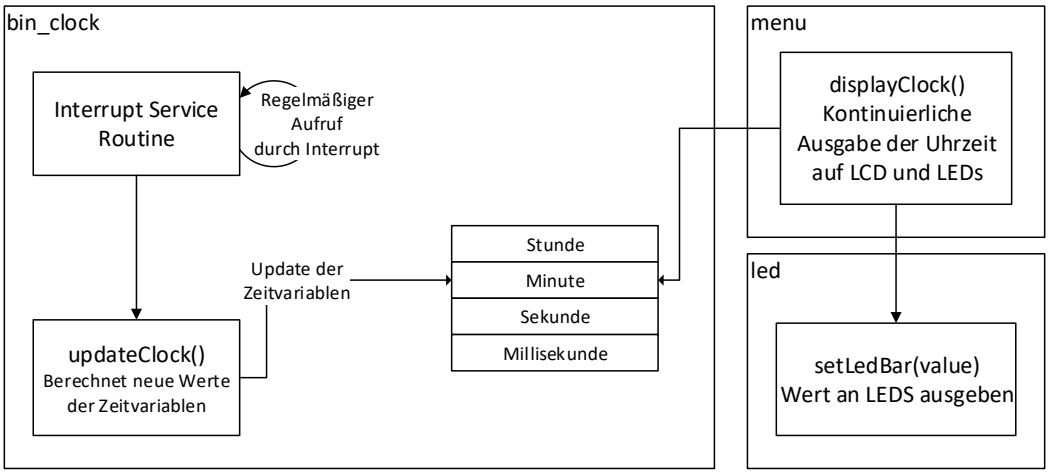

Abbildung 1.8: Der Aufbau der Binäruhr

Zur Realisierung der Binäruhr dienen neben der Anzeigefunktion `displayClock` und die Module `bin_clock` und `led`.

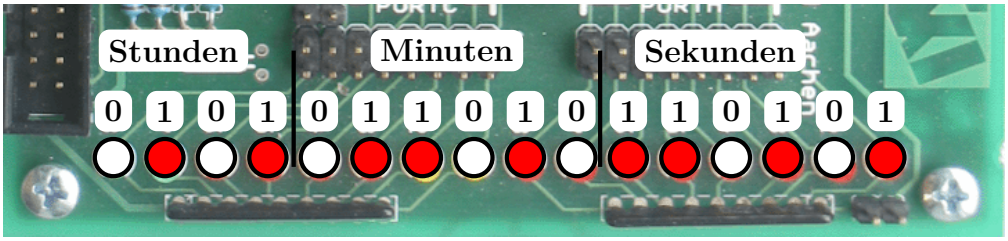

Abbildung 1.9: Die Uhrzeit 5:26:53 wird auf den LEDs ausgegeben.

Im Folgenden wird schrittweise die Vervollständigung des `bin_clock`-Modules erläutert. Legen Sie als Erstes vier Variablen für Stunden, Minuten, Sekunden und Millisekunden in diesem Modul an, welche mit 0 bzw. 12 initialisiert werden. Beachten Sie die korrekte Dimensionierung der Variablen. Lesen Sie im Kapitel *5.1.4 Datentypen* des *Begleitenden Dokuments für das Praktikum Systemprogrammierung* nach, welche Datentypen für den Wertebereich der Variablen geeignet sind.
Zur Implementierung der Uhr wird ein interner Timer des Mikrocontrollers verwendet. Der Timer aktiviert in äquidistanten Abständen einen Interrupt. Bei jedem Aufruf des Interrupts müssen die entsprechenden Zeitvariablen aktualisiert werden.
Die Ihnen vorgegebene Initialisierungsfunktion `initClock` dient dazu, einen *Timer* des Mikrocontrollers so zu initialisieren, dass dieser in einem Intervall von 10 ms einen Interrupt aktiviert. Hierzu wurde der Timer im *Clear Timer on Compare Match* (CTC) Modus konfiguriert. Der im `TCCR0B`-Register hinterlegte Prescaler von 1024 bewirkt, dass das Zählregister `TCNT0` bei jedem 1024. Takt des Oszillators um Eins inkrementiert wird. Bei einem Prozessortakt von 20 MHz wird das Zählregister somit alle $\frac{1}{20\,000\,000} \cdot 1024\,\mathrm{s} = 51{,}2\,\mu\mathrm{s}$ um Eins inkrementiert. Aufgrund des CTC-Modus löst der Timer genau dann einen Tick aus, wenn das Zählregister den im `OCR0A` Register hinterlegten Wert (hier 195) erreicht hat. Somit ergibt sich ein Intervall von $51{,}2\ \mu\mathrm{s} \cdot 195 \approx 10\,\mathrm{ms}$. Beachten Sie, dass ein Aufruf der `initClock`-Funktion im vorgegebenen Codegerüst bereits existiert. Nachdem der Timer konfiguriert und aktiviert wurde, wird im Takt von 10 ms die Interrupt-Service-Routine `ISR(TIMER0_COMPA_vect)` aufgerufen. In dieser ist der Wert der Millisekunden-Variable entsprechend des Timer-Intervalls anzupassen.
Rufen Sie nach dem Anpassen der Millisekunden-Variable die Funktion `updateClock` auf. Diese soll die Werte der übrigen Zeitvariablen korrekt anpassen. Erreicht der Wert der Millisekunden-Variable beispielsweise den Wert 1000, so muss die Sekunden-Variable inkrementiert und die Millisekunden-Variable auf den Wert Null zurückgesetzt werden.

Die zweite benötigte Komponente für die Binäruhr ist das `led`-Modul. In diesem soll eine Ansteuerung des LED-Bargraphen des Mikrocontrollers realisiert werden. Bedingt durch die Verschaltung der LEDs leuchten diese genau dann, wenn der Ausgang des Mikrocontrollers einen Low-Pegel (0V) aufweist. Die Funktion `setLedBar` soll den übergebenen 16-Bit Parameter so auf den LED-Bargraphen umsetzen, dass eine LED genau dann leuchtet, wenn das entsprechende Bit im übergebenen Parameter Eins ist. Für diese Transformation eignet sich der Operator $\sim$, der das bitweise Invertieren einer Variable

ermöglicht (siehe Kapitel *5.1.13: Operatoren* des *Begleitenden Dokuments für das Praktikum Systemprogrammierung*). Das „Least Significant Bit“ (LSB) des Parameters soll auf Pin `A0` und das MSB auf Pin `D7` ausgegeben werden (siehe Abbildung 1.1). Der Parameter `activateLedMask` ist für dieses Unterprogramm nicht relevant und muss daher nicht berücksichtigt werden. Beachten Sie hierbei, dass ein Aufruf der Funktion `initLedBar` bereits im Codegerüst vorhanden ist.

Wert	8	4	2	1	32	16	8	4	2	1	32	16	8	4	2	1
Format	H	H	H	H	M	M	M	M	M	M	S	S	S	S	S	S
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pin	D7	D6	D5	D4	D3	D2	D1	D0	A7	A6	A5	A4	A3	A2	A1	A0

Tabelle 1.1: Ausgabeformat der Binäruhr auf dem LED-Bargraph.

Nach Implementierung der Binäruhr- und LED-Funktionen können diese von Ihnen verwendet werden, um die vorgegebene `displayClock`-Funktion zu vervollständigen. Legen Sie eine temporäre Variable `clockVal` vom Typ `uint16_t` an und verwenden Sie Bitshift-Operationen, um die Zeitvariablen an die entsprechenden Positionen in diese Variable zu verschieben. Rufen Sie anschließend `setLedBar` mit dieser Variable als Parameter auf, um die Uhrzeit auf den LEDs auszugeben. Bei der Auswahl des entsprechenden Menüpunkts im vorgegebenen Menüdialog wird diese Funktion aufgerufen. Analog zu dem ersten Unterprogramm soll auch dieses Unterprogramm solange aktiv bleiben, bis der ESC-Button gedrückt wird. Solange dies nicht geschehen ist, soll die Uhrzeit auf dem Display des Mikrocontrollers und dem LED-Bargraphen ausgegeben werden. Verwenden Sie hierbei die Getter-Funktionen aus dem `bin_clock`-Modul. Auf dem Display und den LEDs sollen permanent die aktuellen Größen für Stunden, Minuten, Sekunden und Millisekunden angezeigt werden. Zur besseren Lesbarkeit der Uhrzeit auf dem Display sollen die ausgegebenen Zahlen in das Format `HH:MM:SS:mmm` transformiert werden. Hierzu muss der auszugebende Zahlenwert auf seine Größe analysiert werden, bevor er auf dem Display ausgegeben wird. Weist dieser zu wenige Stellen für das gewünschte Format auf, so müssen eine oder mehrere Nullen durch eine entsprechende Ausgabe auf dem Display vorangestellt werden. Beim Verlassen des Unterprogramms durch die ESC-Taste muss der Timer nicht ausgeschaltet werden.

Programm 3: Interner AD-Wandler

Sie haben bereits im ersten Teil des Versuchs grundlegende Techniken zur AD-Wandlung kennengelernt. Dabei haben Sie die AD/DA-Platine verwendet und die entsprechenden Wandler selbst implementiert. Der ATmega 644 verfügt jedoch auch über einen integrierten AD-Wandler. Dieser wendet das Prinzip eines SAR-Wandlers an und besitzt eine Auflösung von 10 Bit. Seine Geschwindigkeit wird abhängig vom Prozessortakt gesteuert und sollte in dem Frequenzband von $50 - 200\,\text{kHz}$ liegen. Die Wandlungsfrequenz des integrierten AD-Wandlers kann mit Hilfe des Registers `ADCSRA` angepasst werden,

der einen Prescaler zum Prozessortakt festlegt. Um innerhalb des gültigen Frequenzbandes zu bleiben, ist ein Prescaler von 128 gewählt worden. Die Frequenz des AD-Wandlers beträgt somit 20 MHz/128 ≈ 156 kHz. Als Eingang für die zu messende Spannung können verschiedene Pins ausgewählt werden. Diese werden beim ATmega 644 als `ADC0` bis `ADC7` bezeichnet und sind mit den Pins `A0` bis `A7` (Port A) verbunden. Für diese Aufgabe wurde der Kanal `ADC0` gewählt, welcher mit Pin `A0` verbunden ist. Beachten Sie daher, dass Ihnen der Pin `A0` nicht für die entsprechende LED zur Verfügung steht (siehe Tabelle 1.8).
Die Funktion `initAdc` dient der Initialisierung der soeben erläuterten Register und ist Ihnen vollständig vorgegeben. Beachten Sie, dass ein Aufruf dieser Funktion bereits im Codegerüst vorhanden ist. Das Ergebnis einer Wandlung wird von der ebenfalls vorgegebenen Funktion `getAdcValue` zurückgeliefert. Implementieren Sie die Funktion `displayAdc`, die folgende Aufgaben erfüllen soll:

- Anzeigen der Spannung in der ersten Zeile des Displays
- Ausgabe der Spannung auf dem LED-Bargraphen (Format siehe Abbildung 1.2)
- Ein Untermenü in der zweiten Zeile des Displays, die das Speichern und Anzeigen von Spannungswerten ermöglicht

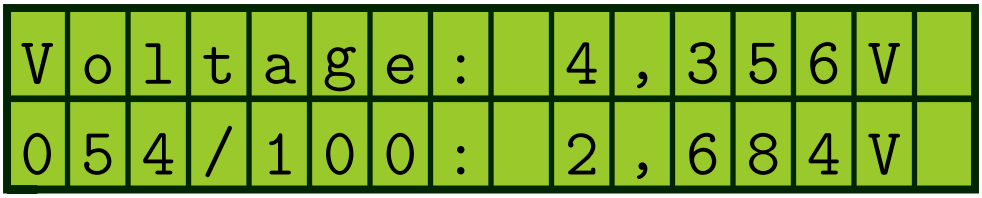

Abbildung 1.10: Beispielhafte Ausgabe des ADC-Menüs.

Anzeige der Spannung: Abbildung 1.10 zeigt das ADC Menü exemplarisch. Es soll kontinuierlich die zuletzt gemessene Spannung sowohl auf dem LCD als auch auf dem LED-Bargraphen angezeigt werden. Zum Anzeigen der Spannung auf dem Display eignet sich die Funktion `lcd_writeVoltage` des LCD-Treibers.
Es bietet sich an, das gesamte Programm in Form einer Endlosschleife aufzubauen, welche eine Verzögerung von 100 ms enthält und in jedem Schleifendurchlauf die Spannung misst und ausgibt. Somit wird eine sowohl flüssige als auch flackerfreie Darstellung der Informationen gewährleistet. Analog zu den Programmen 1 und 2 soll auch dieses durch das Drücken der ESC-Taste verlassen werden.
Die Ausgabe des Spannungswertes auf dem LED-Bargraphen soll, wie in Abbildung 1.2 gezeigt, linear erfolgen. Es stehen 15 LEDs zur Verfügung und die obere Schranke des zu messenden diskreten Spannungswertes liegt bei 1023. Somit beträgt die Auflösung der Ausgabe pro LED $\frac{5\text{V}}{15} = 0,\overline{3}\text{V}$ und bezogen auf die diskret hinterlegte Spannung

Spannung (V)	5,0	4,7	4,3	4,0	3,7	3,3	3,0	2,7	2,3	2,0	1,7	1,3	1,0	0,7	0,3	
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pin	D7	D6	D5	D4	D3	D2	D1	D0	A7	A6	A5	A4	A3	A2	A1	A0

Tabelle 1.2: Ausgabe der digitalisierten Spannung auf dem LED-Bargraph.

$\frac{1023}{15} = 68$. In Abbildung 1.11 sind beispielhaft drei verschiedene Spannungsausgaben auf dem LED-Bargraphen dargestellt.

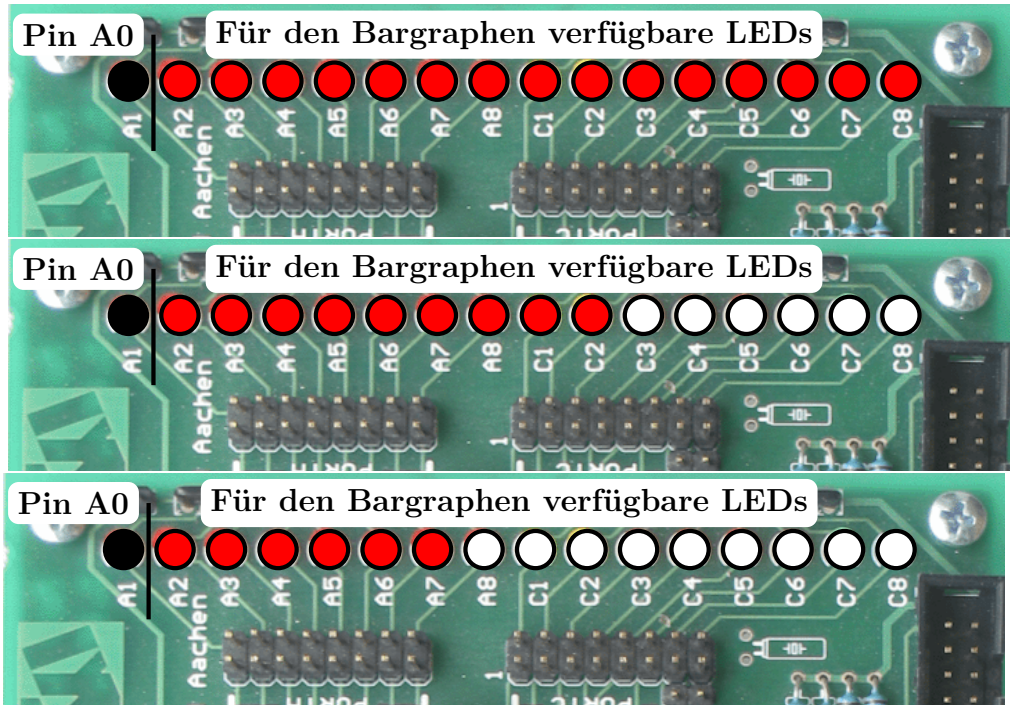

Abbildung 1.11: Bargraph bei Messung von 5V (oben), 3V (mitte) und 2V (unten)

Für die Umwandlung des Rückgabewertes der Funktion `getAdcValue` in eine Binärzahl, die zur Ausgabe auf dem LED-Bargraphen geeignet ist, bietet sich folgendes Vorgehen an:

1. Initialisierung einer 16-Bit Variable `ledValue` mit 0. Jedes Bit dieser Variable ist stellvertretend für eine LED auf den Bargraphen. Es stehen jedoch nur 15 LEDs zur Verfügung, da Pin A0 für die AD-Wandlung verwendet wird, weshalb die dazu zugehörige LED nicht verwendet werden kann.

2. Initialisierung einer 16-Bit Variable `adcResult` mit dem Rückgabewert der Funktion `getAdcValue`

3. Wie oben erläutert repräsentiert der Wert 68 der Variable `adcResult` eine Spannung von $0,\overline{3}$V auf den LED-Bargraphen. Damit der Bargraph die gesamte Spannung widerspiegelt, müssen die folgenden Schritte wiederholt werden, bis `adcResult` kleiner als 68 ist:

   a) `ledValue` um Eins nach links shiften (entspricht der Multiplikation mit Zwei)

   b) `ledValue` mit dem Wert 2 verodern (entspricht der Addition von Zwei). Dieses Vorgehen bewirkt, dass der Bargraph entsprechend der Spannung gefüllt wird wobei Pin A0 unberührt bleibt, da dieser nicht für den Bargraphen verwendet werden darf.

   c) Von der Variablen `adcResult` den Wert 68 subtrahieren

Im Anschluss an die Umformung kann die Variable `ledValue` auf dem LED-Bargraphen ausgegeben werden. Verwenden Sie hierfür das bereits angelegte und im vorhergehenden Aufgabenteil verwendete Modul `led`. Dieses Modul muss zuvor jedoch so erweitert werden, dass nur bestimmte LEDs zur Ausgabe genutzt werden, da Pin A0 für die AD-Wandlung verwendet wird. Wird dennoch auf diesen Pin zugegriffen, so resultiert dies in einer fehlerhaften Spannungsmessung. Hierzu dient der Parameter `activateLedMask` aus der Datei `led.c`, der eine Bitmaske für aktivierte LEDs darstellt. Es sollen nur diejenigen LEDs ansteuerbar sein, welche mit einer Eins in der Bitmaske gekennzeichnet sind. Passen Sie die Funktion `initLedBar` sowie `setLedBar` so an, dass nur diejenigen Bits der `DDR`- und `PORT`-Register verändert werden, die in der Variable `activateLedMask` mit einer Eins markiert sind. Die Zuordnung der Bits dieser Variable zu den Pins des Mikrocontrollers geschieht analog zu dem Parameter der Funktion `setLedBar` (siehe auch Abbildung 1.2). Beachten Sie hierbei, dass ein Aufruf der Funktion `initLedBar` und das Setzen der Variable `activateLedMask` mit einer geeigneten Bitmaske für jedes Unterprogramm bereits im Codegerüst vorhanden ist.

Speichern von Messwerten: Neben der Anzeige der momentan gemessenen Spannung soll es möglich sein, Messwerte zu speichern und diese zu einem späteren Zeitpunkt anzuzeigen. Zur Benutzerinteraktion muss die Funktion `displayAdc` um eine Menüführung erweitert werden.
Messwerte sollen abgespeichert werden, wenn der Enter-Button gedrückt wird. Um auch mehrere Messwerte nacheinander speichern zu können, soll nicht auf das Loslassen des Buttons gewartet werden. Integrieren Sie daher das Speichern von Messwerten in die Hauptschleife des Programms, um in einem Intervall von 100 ms Messwerte aufzeichnen zu können.
Mithilfe des Up- und Down-Buttons soll es zudem möglich sein, die Spannungswerte aus dem Puffer auszulesen und in der zweiten Zeile des LCD gemäß Abbildung 1.10 anzuzeigen. Verwenden Sie hierzu einen lokalen Zählindex in der Funktion `displayAdc`. Die Anzeige der Spannungswerte soll durch die separate Funktion `displayVoltageBuffer` realisiert werden, die als Parameter den Index des anzuzeigenden Spannungswertes er-

hält. Wurde einer der Buttons gedrückt, soll nicht auf das Loslassen dieses gewartet werden, sondern der Anzeigeindex im Intervall von 100 ms inkrementiert bzw. dekrementiert werden, was das *Blättern* durch die Messwerte ermöglicht.
Eine automatische Inkrementierung des Anzeigeindexes beim Speichern von neuen Werten ist nicht erforderlich. Wichtig ist hingegen, dass die angezeigten Indizes innerhalb der Grenzen des Puffers liegen. Achten Sie bei der Ausgabe der Indizes darauf, bei Bedarf führende Nullen einzufügen, um eine gute Lesbarkeit beim Blättern zu gewährleisten.

Implementieren Sie die Datenbank zur Speicherung der Messwerte in der Datei `adc.c`. Zur Initialisierung und Verwendung dieser Datenbank werden Ihnen Funktionsrümpfe bereitgestellt, die vervollständigt werden müssen. Die Datenbank soll als Puffer realisiert werden, auf den mit Zeigern zugegriffen werden kann. Soll ein neuer Messwert gespeichert werden, wird die Funktion `storeVoltage` des ADC-Moduls aufgerufen und der zuletzt gemessene Spannungswert unter dem ersten freien Index des Puffers gespeichert. Anschließend soll der Pufferindex der Datenbank unter Beachtung der Größe des Puffers um eine Position verschoben werden. Der Wertebereich von diesem Index reicht von 0 bis zum Rückgabewert der Funktion `getBufferSize` − 1. Ist die maximale Indexposition erreicht, sollen keine weiteren Messwerte gesichert und der Index nicht weiter inkrementiert werden.

Implementierung der Datenbank: Die Funktion `storeVoltage` dient gleichermaßen zum Speichern von Messwerten, als auch zur Initialisierung der Datenbank. Ist beim Aufruf der Funktion die Puffergröße gleich 0, muss zuerst Speicher für den Puffer alloziert werden. Dies wird mit Hilfe der Funktion `malloc` durchgeführt, die durch die Standard C-Bibliothek zur Verfügung gestellt wird. Diese Bibliothek kann durch das Inkludieren der Headerdatei `stdlib.h` verwendet werden. Als Argument erhält die `malloc`-Funktion die gewünschte Größe des Speicherbereichs in Byte. Der Rückgabewert ist die Adresse des ersten Bytes des Speicherbereichs. Hat der Rückgabewert die Adresse 0, steht nicht genügend Speicherplatz zur Verfügung und der Puffer kann nicht angelegt werden. Die Verwendung der `malloc`-Funktion ist in Listing 1.3 veranschaulicht.
Für diese Aufgabe soll ein Puffer für 100 Spannungswerte angelegt werden. Für einen Spannungswert müssen im Puffer 2 Byte Speicher reserviert werden. Intern wird der nächste freie Index im Puffer mit `bufferIndex` bezeichnet. Ist dieser größer oder gleich der Puffergröße, so kann kein neuer Spannungswert gespeichert werden. Anderenfalls kann der zuletzt gemessene Spannungswert `lastCaptured` an diese Position gespeichert werden. Bedenken Sie, dass im Anschluss daran der `bufferIndex` inkrementiert werden muss.
Das Beispiel in Listing 1.3 veranschaulicht die Nutzung von Zeigern in diesem Kontext. In Zeile 2 wird ein Zeiger `position` vom Typ `uint16_t*` angelegt, welcher im weiteren Verlauf auf das erste Byte des Speicherbereichs zeigen soll. Als Rückgabewert der Funktion `malloc` wird in diesem Beispiel ein Zeiger auf die 2-Byte-Speicherstelle von Adresse 270 bis 271 angenommen.
Mit Hilfe des Operators `*` kann bei Zeigern der Inhalt an einer Adresse verändert werden.

```
1  // 1. Zeiger auf einen 2-Byte Speicherbereich an Adresse
       270/271 (entsprechend Rückgabewert von malloc):
2  uint16_t* position = malloc(10 * sizeof(uint16_t));
3  uint8_t offset = 4;
4
5  // 2. Beschreiben der Bytes an der Speicherstellen 270/271 mit
       dem Wert 155:
6  *position = 155;
7
8  // 3. Beschreiben der Bytes an der Speicherstellen 278/279 mit
       dem Wert 1000. Das Offset wird automatisch mit 2
       multipliziert, da position ein Zeiger auf einen 2-Byte
       Datentyp ist:
9  *(position + offset) = 1000;
10
11 // 4. Auslesen des Wertes der Speicherstellen 278/279:
12 uint16_t value = *(position + offset); // value = 1000
```

Listing 1.3: Verwendung von Zeigern

In Zeile 6 wird somit der Inhalt an Adresse 270/271 auf den Wert 155 verändert. In einer Zeigervariable vom Typ `uint16_t*` kann eine Adresse abgelegt werden, welche auf eine 2-Byte große Speicherstelle zeigt. Der Zugriff in Zeile 9 macht deutlich, dass die Wahl eines `uint16_t*` Zeigers bewirkt, dass das Offset nicht auf einzelne 1-Byte-Speicherstellen sondern auf 2-Byte-Speicherstellen bezogen wird. Wäre `position` vom Typ `uint8_t*`, so würde der Zugriff in Zeile 9 auf die Speicherstelle 274 statt 278/279 erfolgen. Analog zum Schreibvorgang in Zeile 6 und 9 wird der Lesevorgang in Zeile 12 durchgeführt. Das Voranstellen des Operators `*` signalisiert, dass der Wert der Summe von `position` und `offset` als Adresse aufgefasst werden und von den Speicherstellen an dieser Adresse gelesen werden soll.
Weitere Informationen über die Verwendung von Zeigern können dem *Begleitenden Dokument für das Praktikum Systemprogrammierung* in Kapitel *5.1.7 Zeiger (Pointer)* entnommen werden.
Neben der Speicherung von Spannungswerten muss die als `getStoredVoltage` bezeichnete Zugriffsfunktion implementiert werden, welche den Zugriff auf die interne Datenbank des ADC-Moduls ermöglicht. Diese Funktion hat die Aufgabe, den an dem übergebenen Index gespeicherten Messwert auszulesen. In dieser Funktion muss zunächst überprüft werden, ob der übergebene Index gültig ist, d.h. ob er unterhalb der aktuellen Position von `bufferIndex` liegt. Ist dies der Fall, so gibt die Funktion den gewünschten Spannungswert zurück, anderenfalls den Wert 0.

1.4.4 Übersicht der Hausaufgaben

Folgende Übersicht listet alle Typen, Funktionen und Aufgaben auf. Alle aufgelisteten Punkte müssen zur Teilnahme am Versuch bis zur Abgabefrist bearbeitet und hochgeladen werden. Diese Übersicht kann als Checkliste verwendet werden und ist daher mit Checkboxen versehen.

Aufgaben zum AD/DA-Wandlung:

- ☑ Implementierung des `os_input`-Moduls
- ☑ `void manuell(void)`
  - ☑ Ermöglicht das Ansteuern des R-2R-Netzwerks mit den Schiebereglern
  - ☑ Zeigt den Status der Schieberegler auf den LEDs an
- ☑ `void sar(void)`
  - ☑ Implementierung eines AD/DA-Wandlers nach dem Prinzip eines SAR-Wandlers
  - ☑ Die Wandlung soll erst nach dem Drücken des Buttons starten und zusätzlich auf den LEDs angezeigt werden
- ☑ `void tracking(void)`
  - ☑ Implementierung eines AD/DA-Wandlers nach dem Prinzip eines Tracking-wandlers
  - ☑ Die Wandlung soll erst nach dem Drücken des Buttons starten und zusätzlich auf den LEDs angezeigt werden

Aufgaben zum ATmega 644:

- ☑ Erweiterung des os_input-Moduls auf vier Taster
- ☑ *Programm 1: HelloWorld*
  - ☑ Verwendung des LCD-Treibers
  - ☑ Ausgabe des Textes „HalloWelt!“ auf dem LCD
- ☑ *Programm 2: Binäruhr*
  - ☑ Implementierung eines Moduls zur Verwaltung der Uhrzeit
  - ☑ Implementierung eines Treibers zur Ansteuerung des LED-Bargraphen
  - ☑ Anzeige einer Binäruhr auf dem LED-Bargraphen und dem LCD
- ☑ *Programm 3: Interner AD-Wandler*
  - ☑ Anzeige der gemessenen Spannung auf dem LED-Bargraphen und dem LCD
  - ☑ Implementierung einer Datenbank zur Verwaltung von Messwerten
  - ☑ Umsetzung der Benutzerschnittstelle, welche die momentane Messspannung anzeigt sowie das Speichern und Anzeigen von Messdaten ermöglicht

1.5 Nutzung des Testpools

Da dieser Versuch mit mehreren Hardwarekonfigurationen arbeitet, wurden die PCs im Testpool in die Kategorien *V1: AD/DA* und *V1: ADC Menue* unterteilt. Der Programmcode für die Aufgaben zur AD/DA-Wandlung kann auf PCs der ersten Kategorie getestet werden. Steuerelemente der *ATMega Remote*-Software ermöglichen das Erzeugen einer Messspannung sowie das Bedienen der DIP-Schalter.

Achtung

Beachten Sie diesbezüglich, dass im Versuch das Setzen der **Pullup-Widerstände** notwendig ist. Für das Testen im Testpool müssen diese jedoch nicht - können aber - gesetzt werden. Allerdings wird dringend empfohlen die Pullup-Widerstände **immer** zu setzen, damit es zu keinem unterschiedlichen Verhalten Ihrer Software im Testpool und Versuch kommt. Zudem stimmt die eingestellte Spannung der Software nicht exakt mit der gemessenen Spannung durch den Mikrocontroller überein. Es ist mit Abweichungen von 0,1 - 0,2 V zu rechnen. Ferner wird davon abgeraten, die Programmbibliothek **math.h** zu verwenden, weil diese zu komplexe Datentypen und Funktionen für den Mikrocontroller enthält.

Analog dazu kann Programmcode für die Aufgaben zum ATmega 644 auf PCs der zweiten Kategorie getestet werden. Während des Testens von *Unterprogramm 2: Binäruhr* ist zu beachten, dass die LED von Pin A0 nicht verbunden und somit nicht ansteuerbar ist. Der Grund hierfür liegt darin, dass Pin A0 mit der einstellbaren Messspannung verbunden ist, welche für das *Unterprogramm 3: Interner AD-Wandler* benötigt wird.

1.6 Präsenzaufgaben

Im folgenden Abschnitt werden die Aufgaben vorgestellt, die Sie während der Präsenzzeit des Praktikumsversuchs bearbeiten müssen. Lesen Sie sich diesen Abschnitt bereits vor dem Versuch durch, um sich im Vorfeld mit den Aufgaben vertraut zu machen.

1.6.1 Versuche zur AD/DA-Wandlung

Stellen Sie zunächst sicher, dass der *JTAG Programmierer* (Abb. 1.12) eingeschaltet und mit dem Evaluationsboard verbunden ist. Bei einem korrekt verbundenen JTAG Programmierer leuchten eine grüne und zwei rote LEDs. Ist der JTAG Programmierer eingeschaltet, aber nicht verbunden, leuchtet nur eine rote LED.
Die Versuchsplatine aus Abbildung 1.2 ermöglicht es, einen AD-Wandler wie beispielsweise den Tracking-Wandler oder den Sukzessive-Approximation-Wandler umzusetzen (vgl. Abbildungen 1.5 und 1.6). Beide Verfahren (Tracking und Sukzessive-Approximation) wurden als Hausaufgabe vorbereitet und werden nun auf der Hardware getestet.

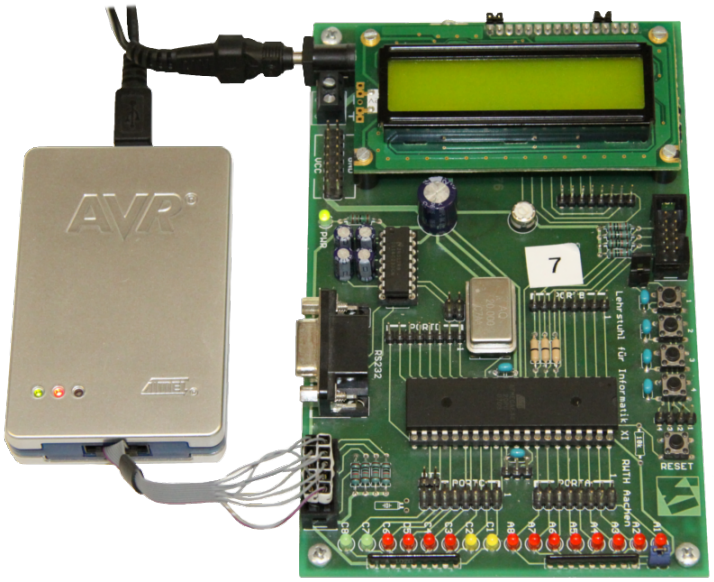

Abbildung 1.12: Korrekt verbundener JTAG Programmierer

Auf der AD/DA-Platine befindet sich ein 8-Bit DA-Wandler in Form eines R-2R-Netzwerks. Die Platine ist so konstruiert, dass sie auf das Evaluationsboard aufgesetzt werden kann und somit mit den Ein- und Ausgängen des Mikrocontrollers verbunden ist.
Der Pin C0 an Port C zeigt den Status des Ausgangs des Operationsverstärkers an und wird durch die `COMP_OUT-LED` (Markierung 2 in Abbildung 1.2) auf der Versuchsplatine wiedergegeben. Die `COMP_OUT-LED` leuchtet, wenn Pin C0 = 1 gilt. Die AD/DA-Platine benötigt eine Versorgungsspannung, die um mindestens 1,5 V größer ist, als die zu vergleichenden Spannungen, da sonst eine fehlerfreie Messung nicht möglich ist. Damit nur ein Steckernetzteil benötigt wird, geschieht die Spannungsversorgung des Mikrocontrollerboards über die Versuchsplatine. Dabei muss lediglich die AD/DA-Platine über das Steckernetzteil an die abgeschaltete Spannungsversorgung angeschlossen werden (Markierung 3 in Abbildung 1.2).

Überprüfung des R-2R-Netzwerks

Eingabe (MSB ... LSB)	gemessene Spannung / V
00000000	
00011001	
01100101	
01111111	
11111111	

Tabelle 1.3: Schalterstellungen für die Überprüfung des R-2R-Netzwerks.

Zunächst soll die Funktionalität des R-2R-Netzwerks überprüft werden. Verwenden Sie dazu die vorbereitete, manuelle Ansteuerung des R2R. Messen Sie mit dem Multimeter die fünf in Tabelle 1.3 angegebenen Schalterstellungen und notieren Sie jeweils die Ausgangsspannung. Machen Sie sich jedoch vorher klar, wo sich das MSB und LSB auf der Platine befinden. Verwenden Sie dafür den Messpunkt R2R-OUT (Markierung 4 in Abbildung 1.2) und für die Masse den Messpunkt GND auf dem Evaluationsboard des ATmega 644. Wie genau ist die Wandlung? Wie groß ist die maximal bzw. minimal erzeugbare Spannung?

SAR-Wandler

Ändern Sie den Versuchsaufbau nun wie folgt:

- Begrenzen Sie den Strom des Netzgerätes auf 200 mA (siehe Dokument *Bedienungsanleitungen und Sicherheitshinweise*). Stellen Sie die Spannung des Netzgeräts nun auf 0 V ein und schalten Sie das Netzgerät wieder aus.
- Verbinden Sie "+" vom Ausgang des Netzgeräts mit "Input +" (Markierung 5 in Abbildung 1.2) auf der Platine.
- Verbinden Sie "-" vom Ausgang des Netzgeräts mit "Input -" (Markierung 5 in Abbildung 1.2) auf der Platine. Bei dieser Spannung handelt es sich um die zu messende Spannung (U_{MESS}).

Testen Sie den Sukzessive-Approximation-Wandler, den Sie programmiert haben. Überprüfen Sie ihn auf geeignete Art die Funktionsweise. (Markierung 6 in Abbildung 1.2)

Tracking-Wandler

Testen Sie den Tracking-Wandler, den Sie programmiert haben. Was fällt Ihnen im Vergleich zum Sukzessive-Approximation-Wandler auf?

1.6.2 Versuche zum ATmega 644

Trennen Sie die Spannungsversorgung der AD/DA-Platine sowie des Evaluationsboards. Nehmen Sie die Platine vorsichtig und gleichmäßig vom Evaluationsboard ab. Verkabeln Sie im Anschluss daran das Evaluationsboard entsprechend Tabelle 1.8 bzw. Abbildung 1.13.

Programm 1: HelloWorld

Übertragen Sie die Software des ADC Menüs auf den Mikrocontroller. Während des Navigierens im Menü muss die korrekte Funktionsweise des Input-Moduls geprüft werden: Das Blättern soll erst beim Loslassen der Taster ausgeführt werden und ein Programm erst dann gestartet werden, wenn der Enter-Button gedrückt und losgelassen wurde. Die Ausgabe dieses Programms ist ein gleichmäßiger blinkender Schriftzug „Hallo Welt!“.

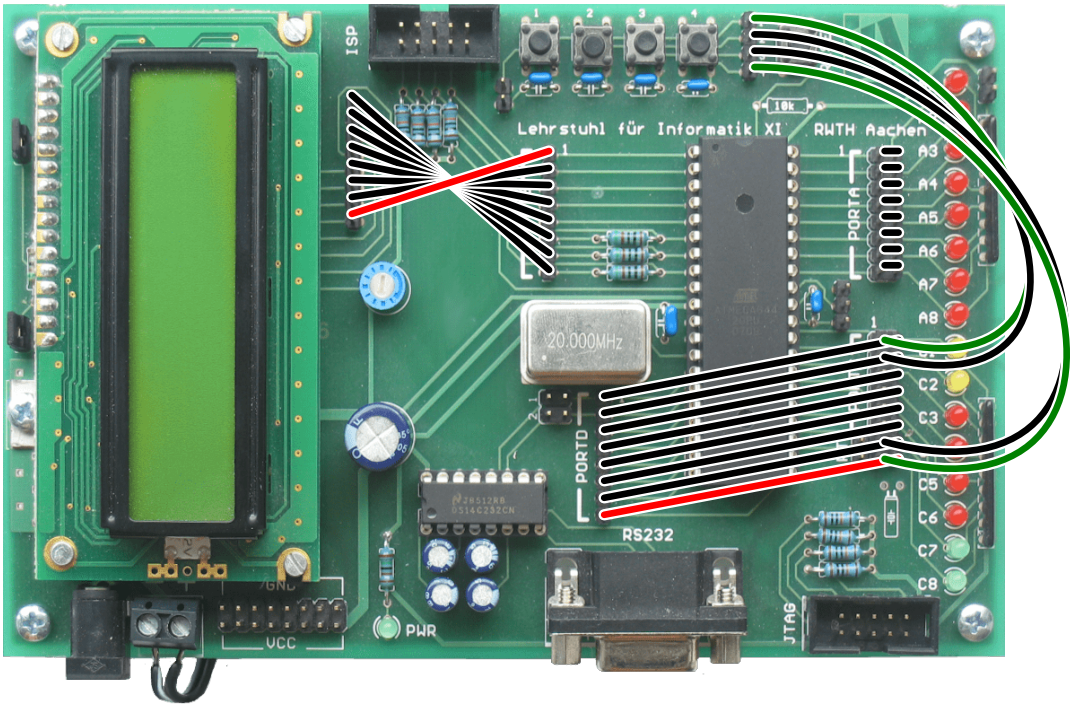

Abbildung 1.13: Verbindungen des Evaluationsboards für die Aufgaben zum ATmega 644.

Programm 2: Binäruhr

Für dieses Programm müssen die Kanäle Pin `A0` bis `A7` mit den nebenliegenden LEDs verbunden werden. Die auf dem Display ausgegebene Uhrzeit muss dem beschriebenen Format entsprechen und mit der binär auf dem LED-Bargraphen ausgegebenen Uhrzeit übereinstimmen.

Programm 3: Interner AD-Wandler

Für den internen AD-Wandler muss die Verbindung des Pins A0 zur nebenliegenden LED entfernt werden. Stattdessen wird Pin A0 mit der Spannungsquelle verbunden. Gehen Sie wie folgt vor:

- Begrenzen Sie den Strom des Netzgerätes auf 200 mA
- Stellen Sie den Ausgang A des Netzgeräts auf 0 V ein und schalten Sie das Netzgerät aus
- Verbinden Sie "+" von Output A mit Pin A0 des Evaluationsboards
- Verbinden Sie "-" von Output A mit GND (Stiftleiste neben VCC) auf dem Evaluationsboard

Achtung

Achten Sie darauf, dass die ausgegebene Spannung des Netzgerätes einen Spannungswert von 6V nicht überschreitet! Zu hohe Spannungen zerstören den Mikrocontroller.

Führen Sie einige Testmessungen durch und beobachten Sie das Verhalten der Spannungsanzeige bei gleichbleibender Spannung des Netzgerätes. Welches Verhalten ist zu beobachten?
Speichern Sie Messwerte durch Drücken der Enter-Taste und prüfen Sie, ob die Datenbank die Messwerte korrekt ablegt und wieder abrufen kann. Lassen Sie die korrekte Funktionsweise des internen AD-Wandlers und den zugehörigen Funktionen von einem Betreuer abnehmen.

Erweitern Sie Ihre Implementierung anschließend wie folgt: Um exaktere Spannungsmesswerte zu erhalten, bietet es sich an, mehrere aufeinanderfolgende AD-Wandlungen durchzuführen und den Mittelwert aus diesen zu bilden. Passen Sie hierzu die Funktion `getAdcValue` so an, dass 16 aufeinanderfolgende Messungen durchgeführt werden und die Messwerte lokal aufsummiert werden. Nach Durchführung der Messungen muss der Mittelwert gebildet, in der Variable `lastCaptured` abgelegt und zurückgegeben werden. Führen Sie erneut einige Testmessungen durch und beobachten Sie das Verhalten der Spannungsanzeige bei gleichbleibender Spannung des Netzgerätes. Welches Verhalten ist nun zu beobachten?

1.7 Pinbelegung AD/DA-Wandlung

Port	Pin	Belegung
Port A	A0	LED für Wandlungsergebnis (Bit 1)
	A1	LED für Wandlungsergebnis (Bit 2)
	A2	LED für Wandlungsergebnis (Bit 3)
	A3	LED für Wandlungsergebnis (Bit 4)
	A4	LED für Wandlungsergebnis (Bit 5)
	A5	LED für Wandlungsergebnis (Bit 6)
	A6	LED für Wandlungsergebnis (Bit 7)
	A7	LED für Wandlungsergebnis (Bit 8)
Port B	B0	R-2R-Netzwerk (Ausgabe Steuersignale)
	B1	R-2R-Netzwerk (Ausgabe Steuersignale)
	B2	R-2R-Netzwerk (Ausgabe Steuersignale)
	B3	R-2R-Netzwerk (Ausgabe Steuersignale)
	B4	R-2R-Netzwerk (Ausgabe Steuersignale)
	B5	R-2R-Netzwerk (Ausgabe Steuersignale)
	B6	R-2R-Netzwerk (Ausgabe Steuersignale)
	B7	R-2R-Netzwerk (Ausgabe Steuersignale)
Port C	C0	Eingang für das Ergebnis des Komparators
	C1	Button (Eingabe)
	C2	Reserviert für JTAG
	C3	Reserviert für JTAG
	C4	Reserviert für JTAG
	C5	Reserviert für JTAG
	C6	LED1 (LED zur freien Verfügung)
	C7	LED2 (LED zur freien Verfügung)
Port D	D0	DIP-Schalter 1 (Eingabe für manuelle Wandlung)
	D1	DIP-Schalter 2 (Eingabe für manuelle Wandlung)
	D2	DIP-Schalter 3 (Eingabe für manuelle Wandlung)
	D3	DIP-Schalter 4 (Eingabe für manuelle Wandlung)
	D4	DIP-Schalter 5 (Eingabe für manuelle Wandlung)
	D5	DIP-Schalter 6 (Eingabe für manuelle Wandlung)
	D6	DIP-Schalter 7 (Eingabe für manuelle Wandlung)
	D7	DIP-Schalter 8 (Eingabe für manuelle Wandlung)

Pinbelegung für die Aufgaben zur AD/DA-Wandlung.

1.8 Pinbelegung ADC Menü

Port	Pin	Belegung
Port A	A0	LED 1 / Messspannung (Interner AD-Wandler)
	A1	LED 2
	A2	LED 3
	A3	LED 4
	A4	LED 5
	A5	LED 6
	A6	LED 7
	A7	LED 8
Port B	B0	LCD Pin 1 (D4)
	B1	LCD Pin 2 (D5)
	B2	LCD Pin 3 (D6)
	B3	LCD Pin 4 (D7)
	B4	LCD Pin 5 (RS)
	B5	LCD Pin 6 (EN)
	B6	LCD Pin 7 (RW)
	B7	frei
Port C	C0	Button 1: Enter
	C1	Button 2: Down
	C2	Reserviert für JTAG
	C3	Reserviert für JTAG
	C4	Reserviert für JTAG
	C5	Reserviert für JTAG
	C6	Button 3: Up
	C7	Button 4: ESC
Port D	D0	LED 9
	D1	LED 10
	D2	LED 11
	D3	LED 12
	D4	LED 13
	D5	LED 14
	D6	LED 15
	D7	LED 16

Pinbelegung für die Aufgaben zum ATmega 644.