

Praktikum Systemprogrammierung

Versuch 1

Mikrocontroller & AD/DA-Wandlung

Lehrstuhl für Informatik 11 - RWTH Aachen

12. Oktober 2012

Inhaltsverzeichnis

1	Mikrocontroller & AD/DA-Wandlung	3
1.1	Versuchsinhalte	3
1.2	Lernziel	3
1.3	Grundlagen	4
1.3.1	Einführung in die Verwendung des ATmega 644	4
1.3.2	AD/DA-Wandlung	6
1.4	Hausaufgaben	12
1.4.1	Allgemeines	12
1.4.2	Aufgaben zur AD/DA-Wandlung	12
1.4.3	Aufgaben zum ATmega 644	15
1.4.4	Übersicht der Mikrocontroller-Aufgaben	22
1.5	Nutzung des Testpools	24
1.6	Präsenzaufgaben	24
1.6.1	Versuche zur AD/DA-Wandlung	24
1.6.2	Versuche zum ATmega 644	27
1.7	Pinbelegung AD/DA-Wandlung	30
1.8	Pinbelegung ADC Menü	31

Dieses Dokument ist Teil der begleitenden Unterlagen zum *Praktikum Systemprogrammierung*. Alle zu diesem Praktikum benötigten Unterlagen stehen im L²P-Lernraum unter <http://www.elearning.rwth-aachen.de> zum Download bereit. Folgende Emailadresse ist für Kritik, Anregungen oder Verbesserungsvorschläge verfügbar:

psp@embedded.rwth-aachen.de

1 Mikrocontroller & AD/DA-Wandlung

Mikrocontroller sind Halbleiterchips, welche neben dem Prozessor weitere Peripheriefunktionen auf einem Chip vereinen. Ihr geringer Preis, Größe und Leistungsaufnahme ermöglicht den ökonomischen Einsatz in einer Vielzahl von Anwendungsszenarien für die digitale Kontrolle von Prozessen oder Geräten. Mikrocontroller realisieren eingebettete Systeme die in einer Vielzahl von alltäglichen technischen Gebrauchsgegenständen eingesetzt werden. Die Programmierung von Mikrocontrollern stellt im Gegensatz zu der Programmierung von herkömmlichen Computersystemen jedoch andere Anforderungen an die Entwickler. Neben dem Umgang mit Ressourcenknappheit wird für die Programmierung von Mikrocontrollern mit einer geringen Rechenleistung im allgemeinen auf elementare hardwarenahe Programmiersprachen wie Assembler oder C zurückgegriffen. In diesem Versuch werden zum Einstieg in die Mikrocontrollerprogrammierung mehrere einfache Applikationen für Mikrocontroller implementiert werden.

1.1 Versuchsinhalte

Im ersten Teil dieses Versuchs wird das Evaluationsboard und der in diesem eingebettete Mikrocontroller ATmega 644 vorgestellt. Es wird eine für Mikrocontroller typische Anwendung, die Analog-Digital- und Digital-Analog-Wandlung (AD/DA-Wandlung) betrachtet. Diese Aufgabe stellt eine Einführung in die Programmierung eines Mikrocontrollers unter Verwendung der Programmiersprache C dar.

Im zweiten Teil dieses Versuchs werden einige Module des ATmega 644 vorgestellt und relevante Konzepte der Mikrocontrollerprogrammierung vertieft. Dazu sollen mehrere einfache Applikationen für den Mikrocontroller implementiert werden. Hierzu zählen beispielsweise eine Binäruhr sowie die Ausgabe von verschiedenen Informationen auf einem LED-Bargraph.

Spätere Versuche werden ebenfalls auf dem ATmega 644 basieren. Zum Einstieg in die Softwareentwicklung auf dem ATmega 644 muss das *Begleitende Dokument zum Praktikum Systemprogrammierung* gelesen werden. In diesem werden grundlegende Programmierkonzepte 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 AVR Studio

- Grundlagen der Programmiersprache C
- Programmierung eines Mikrocontrollers
- AD- und DA-Wandlung
- Verwendung von *Interrupts*
- Verwendung von *Zeigern*

1.3 Grundlagen

Dieser Abschnitt stellt grundlegende Informationen bereit, welche nötig sind, um die Haus- und Präsenzaufgaben erfolgreich bearbeiten zu können.

1.3.1 Einführung in die Verwendung des ATmega 644

In diesem Teil der Grundlagen wird zunächst das Evaluationsboard, auf welchem der Mikrocontroller ATmega 644 platziert ist, vorgestellt.

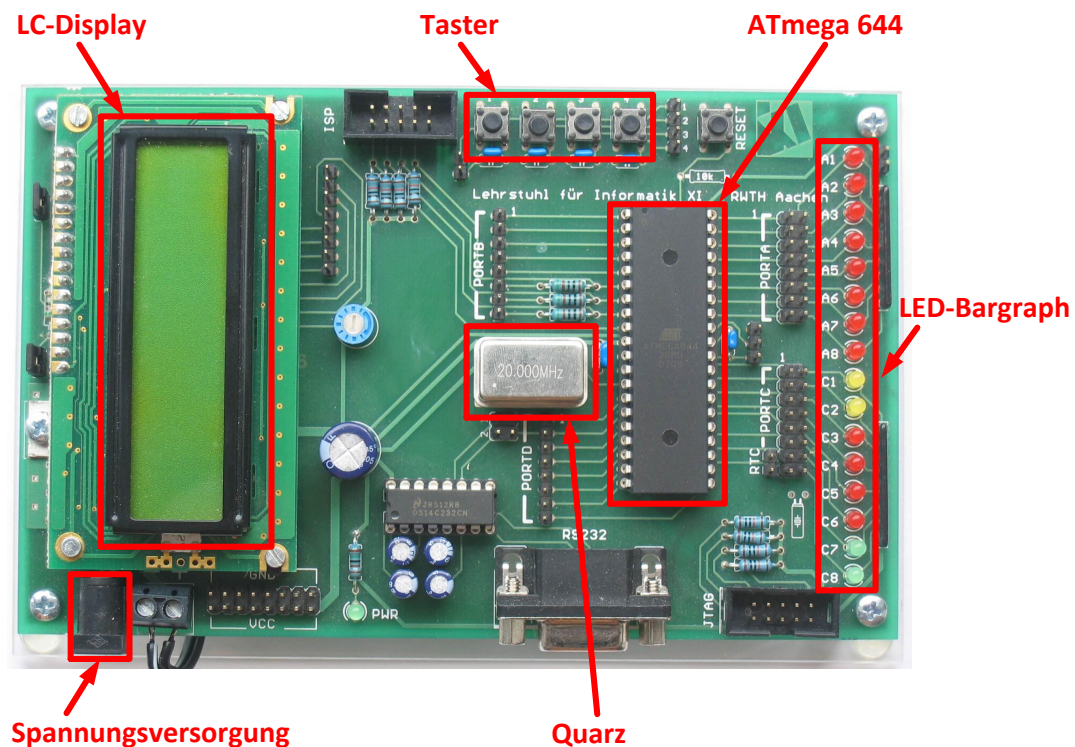


Abbildung 1.1: Das Evaluationsboard des ATmega 644

Informationen zum Evaluationsboard des ATmega 644

Das Evaluationsboard des Lehrstuhls für Informatik 11 der RWTH-Aachen (Abbildung 1.1) bietet eine komfortable Arbeitsumgebung für den Mikrocontroller ATmega 644 der Firma Atmel. Es stellt über Pinleisten anschließbare Eingabe- und Ausgabegeräte, beispielsweise ein LC-Display, Taster und Leuchtdioden bereit.

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.

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

Im Rahmen dieses Versuches wird die `lcd_writeVoltage`-Funktion verwendet um einen diskreten Spannungswert in einen Spannungswert in Gleitkommadarstellung umzuwandeln. Die Funktion kann folgendermaßen verwendet werden: Der Spannungswert wird als diskreter Wert im ersten Parameter übergeben, während der zweite Parameter den maximal erreichbaren Wert des ersten Parameters angibt. Der dritte Parameter gibt die obere Schranke der Spannung an. Die Funktion gibt die Spannung in Gleitkommadarstellung auf dem Display aus. `lcd_writeVoltage(113, 255, 5)` erzeugt beispielsweise die Ausgabe $\frac{113}{255} \cdot 5V = 2.215V$.

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. Als geeignete Entwicklungsumgebung stellt Atmel das AVR Studio kostenlos zur Verfügung. In diesem Praktikum wird die Version 4.19 verwendet. Für weitere Informationen zu AVR Studio und C sei auf die Kapitel 3: *Einführung in AVR Studio* und Kapitel 4: *Einführung in die C Programmierung* im *Begleitende Dokument zum Praktikum Systemprogrammierung* verwiesen.

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?

1.3.2 AD/DA-Wandlung

Dieser Teil der Grundlagen gibt einführende Informationen zum Thema der AD/DA-Wandlung. Zunächst wird hierzu der Operationsverstärker in der Verschaltungsmöglichkeit als Komparator vorgestellt, da dieser essentiell für die AD/DA-Wandlung ist. Die Funktionsweise eines Operationsverstärkers sollte aus der Veranstaltung *Einführung in die Technische Informatik* bereits bekannt sein. Im Anschluss daran werden verschiedene Methoden vorgestellt, um eine AD/DA-Wandlung durchzuführen.

Komparator

Ein Komparator wird mit Hilfe eines Operationsverstärkers realisiert. Durch die in Abbildung 1.2 gezeigte Verschaltung ergibt sich das für einen Komparator typische Verhalten. 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, *nicht-invertierenden*, Eingang höher als die Spannung am negativen, *invertierenden*, 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 in einem gut zu verarbeitenden Signal ausgedrückt werden.

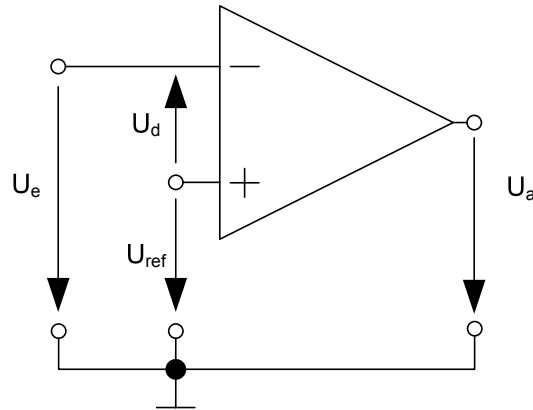


Abbildung 1.2: Komparatorschaltung

AD/DA-Wandler

Ein Digital-Analog-Wandler (DA-Wandler, engl. Digital-to-Analog Converter (DAC)) setzt quantisierte digitale Signale in analoge Signale um. Die einfachste Methode einer DA-Wandlung ist ein R-2R-Netzwerk. Dies ist das einzige in diesem Praktikum genutzte Verfahren für die Umwandlung von digitalen Signalen in analoge Spannungen.

Ein Analog-Digital Wandler (AD-Wandler, engl. Analog-to-Digital-Converter (ADC)) setzt mit unterschiedlichen Methoden analoge Eingangssignale in digitale Daten um. 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 gewisse 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 des Wandlers zu niedrig und sind somit die Abtastzeitpunkte zu weit voneinander entfernt, kann der ursprüngliche Signalverlauf nicht mehr aus dem digitalen Signal rekonstruiert werden (Alias-Effekt). Um dies zu verhindern ist nach dem Abtasttheorem von Nyquist und Shannon ein Wandler mit einer Umsetzungsgeschwindigkeit nötig, die 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 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 durch eine digitale,

zeitdiskrete Folge von Zahlen repräsentiert.

Beispiel Bei einem Abstand von 20 mV zwischen zwei Quantisierungsstufen und einer oberen Schranke des Wertebereichs von 5V ist die Binärzahl 0b1111011 (dezimal 123) die Codierung einer analogen Spannung von rund 2,46 V = $123 \cdot 20 \text{ mV}$.

Die wichtigsten Kenngrößen eines AD/DA-Wandlers sind seine Auflösung und die Umsetzungsgeschwindigkeit. Die Umsetzungsgeschwindigkeit gibt an, wie viele Messwerte pro Zeiteinheit abgetastet werden können. Die Auflösung wird in Bit angegeben und bestimmt, wie viele Quantisierungsstufen zur Verfügung stehen. Eine Auflösung von 8 Bit bedeutet beispielsweise, dass $2^8 = 256$ lineare Quantisierungsstufen zur Darstellung existieren. Dies entspricht einer Genauigkeitsgrenze von 0,39 % ($1/256$) bezogen auf den Messbereich.

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.

Im Folgenden werden das R-2R-Netzwerk als DA-Wandler, sowie der Tracking-Wandler und der Sukzessive-Approximation-Wandler als Beispiele für die Realisierung von AD-Wandlern vorgestellt. Es existieren auch weitere Arten von AD/DA-Wandlern (Delta-Sigma, Flash, ...), welche im Rahmen dieses Versuchs jedoch nicht betrachtet werden.

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?

R-2R-Netzwerk

Das R-2R-Netzwerk ist ein einfacher Digital-Analog-Wandler, bestehend aus nur zwei verschiedenen Arten von Widerständen der Größen R und $2R$. Abbildung 1.3 zeigt die Verschaltung eines n-Bit R-2R-Netzwerks. Durch Reihen- oder Parallelschaltungen wäre der Aufbau sogar mit identischen Widerständen möglich.

An den Eingängen b_0, \dots, b_{n-1} liegt zur Darstellung einer digitalen Eins die Betriebsspannung an, zur Darstellung einer digitalen Null werden die Eingänge auf Masse gelegt.

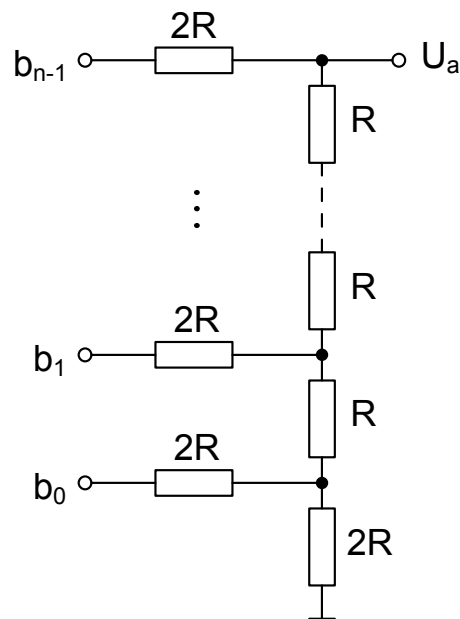


Abbildung 1.3: R-2R Widerstandsnetzwerk

Durch unterschiedliche Beschaltung der Eingänge kann die analoge Ausgangsspannung U_a variiert werden.

LERNERFOLGSFRAGEN

- Wie groß sind die kleinsten einstellbaren Spannungsschritte?
- Durch welche Größen ist der Ausgangswertebereich definiert?

Tracking-Wandler

Der Tracking-Wandler ist ein Analog-Digital-Wandler, der zum Spannungsvergleich auf eine von einem DA-Wandler erzeugte Referenzspannung zurückgreift. Der interne Aufbau eines Tracking-Wandlers ist in Abbildung 1.4 dargestellt. Der momentane digitale Referenzspannungswert wird mit der zu messenden Spannung durch einen Komparator verglichen. Ist die Messspannung höher als die Referenzspannung, inkrementiert ein Up-Down-Counter eine n-Bit-Variable, andernfalls wird die Variable dekrementiert. Die Wandlung gilt als abgeschlossen, wenn die Referenzspannung die Messspannung über- bzw. unterschritten hat. Die n-Bit-Variable wird vom DA-Wandler in einen analogen

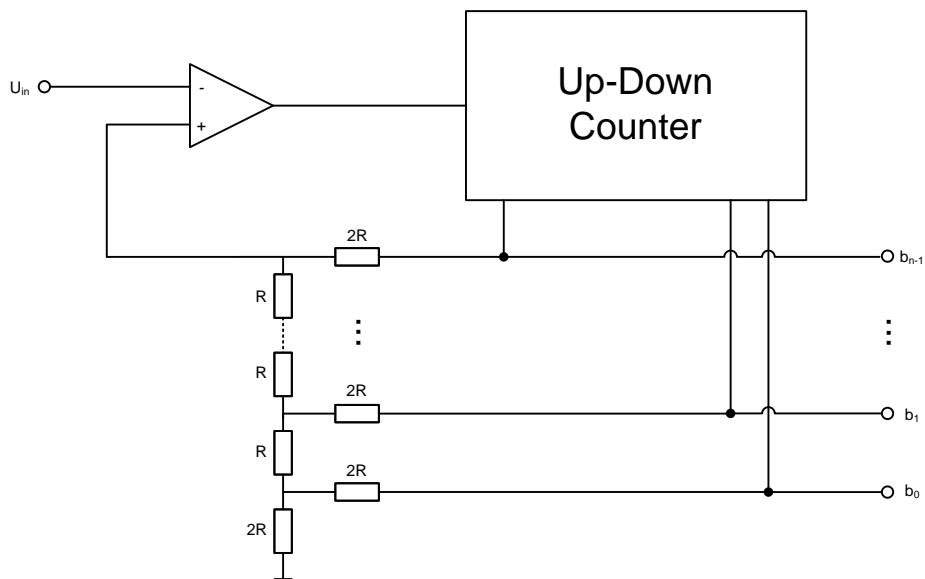


Abbildung 1.4: Tracking-Wandler

Spannungswert umgewandelt und dient als neue Referenzspannung. Da der Tracking-Wandler im schlechtesten Fall 2^n Schritte benötigt, eignet er sich nicht zur Messung von schnellen Spannungsänderungen. Spannungsverläufe, die einer nur geringen Veränderungsrate unterliegen, können jedoch effizient mit dem Tracking-Wandler digitalisiert werden.

Sukzessive-Approximation-Wandler

Eine weitere Möglichkeit zur Realisierung einer Analog-Digital-Wandlung ist die sukzessive Approximation. Dabei wird ein Sukzessive-Approximation-Register (SAR, Abbildung 1.5) verwendet. Es werden stets n Schritte für eine komplette Wandlung benötigt. Die Funktionsweise des SAR zeigt Abbildung 1.6 für einen 3 Bit-Wandler. Im ersten Schritt wird testweise das höchste Bit gesetzt. 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 folgenden, niedrigerwertigen Bit verfahren, bis alle Bits überprüft wurden.

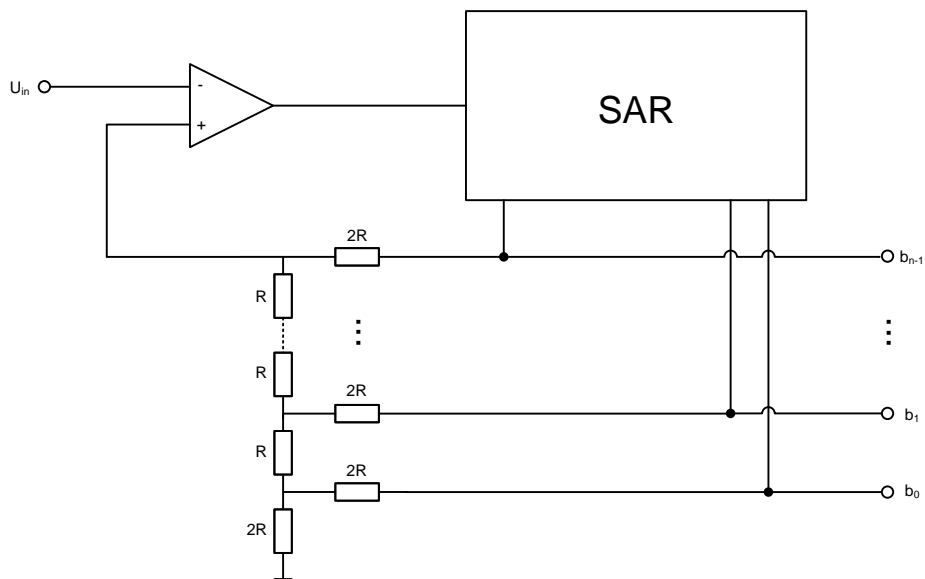


Abbildung 1.5: Sukzessive-Approximation-Wandler

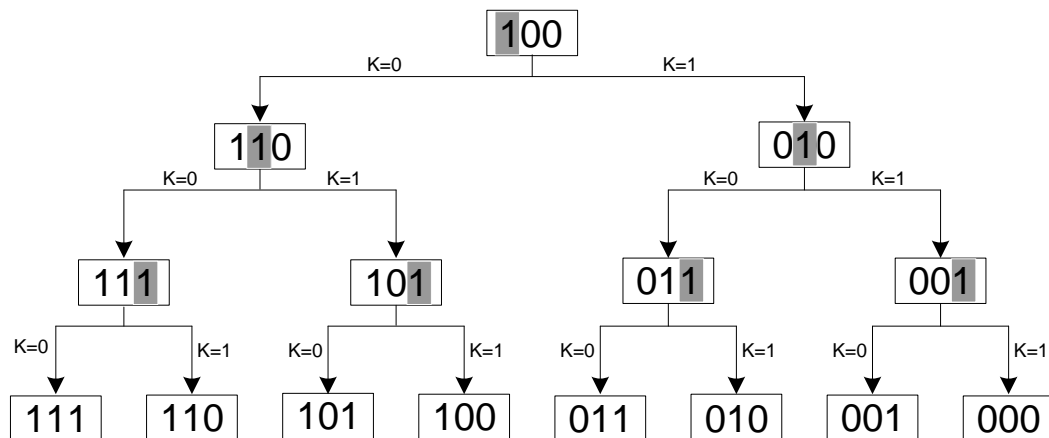


Abbildung 1.6: SAR Entscheidungsbaum für $n = 3$ (K ist das Ergebnis des Komparators)

LERNERFOLGSFRAGEN

- Warum wird zuerst das höchstwertige Bit testweise gesetzt?
- Benötigt ein Sukzessive-Approximation-Wandler immer weniger Schritte als ein Tracking-Wandler?

1.4 Hausaufgaben

Bearbeiten Sie die im folgenden Abschnitt beschriebenen Aufgaben zur AD/DA-Wandlung und zum ATmega 644. Beachten Sie dabei gegebene Hinweise und halten Sie sich bei den Programmieraufgaben an die verwendeten Namen und Bezeichnungen für Variablen und Funktionen.

HINWEIS

In diesem wie in allen folgenden Versuchen ist das Dokument die einzig relevante Vorgabe für Ihre Hausaufgaben. Lesen Sie deshalb die Hausaufgaben vollständig durch und befolgen Sie die Instruktionen des Dokuments. Sollten Unklarheiten auftreten, kontaktieren Sie Betreuer über den Gemeinsamen Bereich im L²P, per E-Mail oder durch den Besuch einer Sprechstunde.

1.4.1 Allgemeines

Lesen und verstehen Sie die folgenden Dokumente, welche im L²P des Praktikums verfügbar sind:

- Das *Grundlagenkapitel* dieses Dokuments
- Das Dokument *Bedienungsanleitungen und Sicherheitshinweise*. Dieses Dokument stellt Ihnen die sichere Bedienung des Multimeters und der Spannungsquelle vor.
- Das *Begleitende 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 dieser Aufgabe sollen Sie sich mit dem Erzeugen von Projekten im AVR Studio vertraut machen. Dazu sollen Sie ein Programm für den ATmega 644 entwickeln, welches folgende Funktionalitäten bereitstellt:

1. Eine manuelle Ansteuerung des R2R Netzwerks
2. Einen Sukzessive-Approximation-Wandler
3. Einen Tracking-Wandler

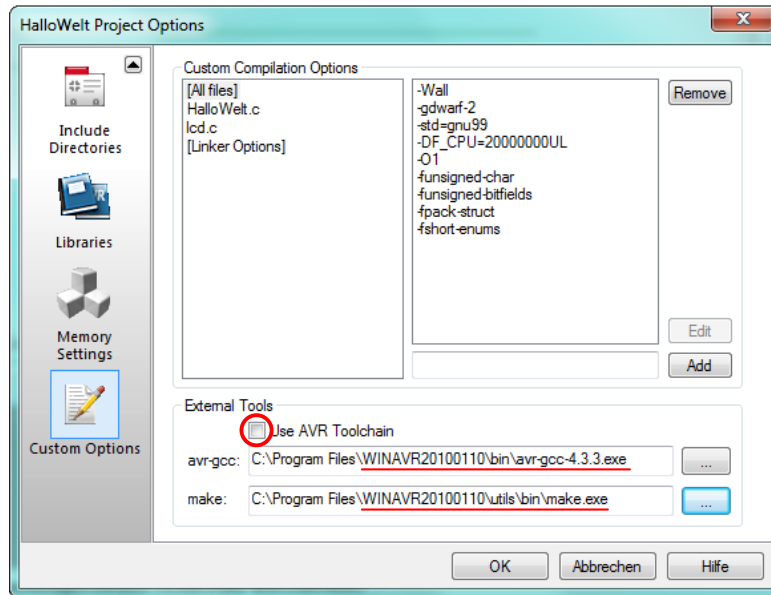


Abbildung 1.7: Custom Options in den Projekteinstellungen

Legen Sie dazu zunächst ein neues Projekt an, wie in Kapitel 3.3.1: *Anlegen eines neuen Projektes* im *Begleitenden Dokument für das Praktikum Systemprogrammierung* beschrieben. Achten Sie darauf, dass Sie in den Projekteinstellungen als Build-Tool die WINAVR-Toolchain, wie in Abbildung 1.7 dargestellt, eingestellt haben. Der Pfad kann je nach Installation abweichen. Weitere Informationen dazu finden Sie in Kapitel 3.3.4 *Die Toolchain WINAVR* im *Begleitenden Dokument für das Praktikum Systemprogrammierung*.

Fügen Sie für jede der oben aufgezählten Funktionalitäten eine Funktion hinzu. Die manuelle Ansteuerung kann beispielsweise als `void manuell(void)` realisiert werden. Fügen Sie unterhalb der drei zu implementierenden Funktionen eine `int main(void)`-Funktion hinzu. Der Mikrocontroller wird diese Funktion beim Start aufrufen. Aus dieser Funktion soll je nach Bedarf eines der Unterprogramme in einer Endlosschleife aufgerufen 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.

Beginnen Sie mit der Funktion zur manuellen Ansteuerung der AD/DA-Platine. In dieser Funktion soll der durch die Schiebeschalter (Markierung 4 in Abbildung 1.12) eingestellte Bit-Wert intern im Mikrocontroller auf das R-2R-Netzwerk umgeleitet werden. Legen Sie diesen Wert ebenfalls an die Leuchtdioden an, um zu überprüfen, welche Bitkombination aktiv ist.

Dazu muss Port D (Schiebeschalter) als Eingang konfiguriert werden. Port A (LEDs) und Port B (R-2R-Netzwerk) müssen als Ausgang konfiguriert werden. Lesen Sie den Wert an Port D ein und schreiben Sie ihn an Port A und Port B. Wie Sie die I/O Ports

des Mikrocontroller s als Ein- bzw. Ausgänge definieren und auf diese zugreifen können entnehmen Sie bitte Kapitel 2.1.1 und 4.1.16 im *Begleitenden Dokument zum Praktikum Systemprogrammierung*.

Die verwendeten Anschlüsse des ATmega 644 für diese Aufgabe sind am Ende der Versuchsunterlagen in Tabelle 1.7 aufgeführt. Das LC-Display aus der Aufgabe 1.4.3 wird für diese Aufgabe nicht verwendet. Achten Sie auf die korrekte Initialisierung aller verwendeten Portregister.

Im Anschluss an die Implementierung der manuellen Ansteuerung des R-2R-Netzwerks müssen zusätzlich die Funktionen für die Realsierung der SAR- und Tracking-Wandler implementiert werden. Achten Sie auch hier auf eine korrekte Initialisierung der verwendeten Register. Die Implementierung der Wandler orientiert sich an dem vorgestellten Verfahren im Grundlagenkapitel. Legen Sie Ihre Implementierung so aus, dass der jeweilige Wandlungsvorgang erst nach dem Drücken des an Pin C1 angeschlossenen Tasters begonnen wird und eine Iteration ausführt.

Da in diesem und kommenden Versuchen häufig mit Tastern gearbeitet wird, bietet es sich an, ein universell einsetzbares Modul für die Verwaltung dieser zu implementieren. Im zur Verfügung gestellten Codegerüst für diese Aufgabe sind die Dateien `os_input.c` und `os_input.h` enthalten, welche die Struktur dieses Moduls beinhalten. Fügen Sie diese Dateien Ihrem Projekt hinzu und implementieren Sie die Buttonabfrage in den bereitgestellten Methodenrümpfen. Listing 1.1 zeigt vereinfacht, wie der Zustand des Buttons an Pin C1 abgefragt werden kann, ohne andere Eingangs-/Ausgangsports zu beeinflussen. Die Schritte 1 und 2 müssen einmalig als Initialisierung ausgeführt werden und bleiben solange bestehen, bis diese überschrieben werden. Weitere Informationen über die Verwendung der Eingangs- bzw. Ausgangsports können dem *Begleitenden Dokument zum Praktikum Systemprogrammierung* in Kapitel 2.1.1: *I/O-Ports* entnommen werden.

```
1 // 1. Pin C1 als Eingang konfigurieren
2 DDRC  &= 0b11111101;
3
4 // 2. Pullup-Widerstand an Pin C1 aktivieren
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: Auslesen eines Buttons an Pin C1

Weiterhin sollten Sie die Funktionen `os_waitForInput` und `os_waitForNoInput` in Ihrem Projekt implementieren und in Ihrer Lösung dort einsetzen, wo auf das Drücken bzw. Loslassen des Tasters gewartet werden soll.

HINWEIS

Es ist notwendig, nach jedem Setzen von Port B (R-2R-Netzwerk) eine gewisse Zeit zu warten, da der Komparator langsamer ist als die Ausführungsgeschwindigkeit des Mikrocontrollers.

Wählen Sie zunächst eine Wartezeit von 50 Millisekunden.

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 4.1.2 entnommen werden. Nachdem diese Headerdatei eingebunden wurde, können folgende Funktionen genutzt werden:

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

Für die beiden Wandler soll der Komparator, dessen Ausgang an Pin C0 liegt, benutzt werden. Für diesen gilt:

- $U_{REF} < U_{MESS} \implies \text{Pin C0} = 0$
- $U_{REF} > U_{MESS} \implies \text{Pin C0} = 1$

Geben Sie außerdem die während der Wandlung benutzen Ansteuerungssignale des R-2R-Netzwerks auf den an Port A angeschlossenen LEDs aus, damit der Wandlungsvorgang visuell nachvollzogen werden kann.

1.4.3 Aufgaben zum ATmega 644

In dieser Teilaufgabe sollen weiterführende Funktionen für den ATmega 644 implementiert werden. Es steht ein Codegerüst für diese Aufgabe zur Verfügung, welches Sie im Rahmen dieses Versuchsteils erweitern müssen. Die bereits vorhandene Implementierung sowie die in diesem Abschnitt erläuterten Aufgaben dienen dazu, fundamentale Konzepte von Mikrocontrollern zu verstehen und mit deren Besonderheiten umzugehen. In dem vorgegebenen Projekt ist eine einfache Menüführung implementiert, über welche mehrere Unterprogramme ausgeführt werden können. Diese müssen von Ihnen implementiert werden.

In der `main`-Funktion des Projekts wird das Tastermodul und ein Treiber für das LC-Display, welches in dieser Aufgabe verwendet wird, initialisiert. Nach der Initialisierung wird das Menü auf dem LC-Display angezeigt. Die Implementierung des Menüs selbst, befindet sich in der Datei `menu.c`. Machen Sie sich die Arbeitsweise des Menüs anhand

der Funktion `showMenu` klar. Das Verständnis dieser Funktion wird Ihnen bei der Implementierung der Programmieraufgaben helfen.

Die erste Aufgabe besteht darin, das Tastermodul zu implementieren, welches innerhalb des Menüs verwendet wird. Aufbauend auf dem Tastermodul der vorigen Aufgabe wird dieses Tastermodul vier statt einem Taster verwalten müssen. An den Pins C0, C1, C6 und C7 sind nun Taster angeschlossen, welche als Enter, Up, Down und ESC bezeichnet werden. Tabelle 1.8 zeigt die Verschaltung der Taster für diese Aufgabe. Machen Sie sich mit dem Verfahren des *Bitshifting* vertraut, um die Zustände der Taster effizient in das hier standardisierte Format umzuformen. Informationen dazu sind im *Begleitenden Dokument für das Praktikum Systemprogrammierung* enthalten.

Das Starten eines Unterprogramms durch das Menü wird durch Benutzung der Funktion `start(programIndex)` ermöglicht. Diese Funktion verwaltet die drei verfügbaren Unterprogramme und initialisiert die genutzten Module vor jedem Start. Die Unterprogramme sind so konzipiert, dass ein Zurückkehren in das Hauptmenü ermöglicht wird.

Programm 1: HelloWorld

Als erstes Unterprogramm soll der Inhalt des Funktionsrumpfs `helloWorld()` implementiert werden. Diese Funktion soll Sie in die Verwendung des LC-Displays einführen. Informationen über die Verwendung des LCD können dem Grundlagenkapitel entnommen werden. Darüberhinaus sind im Codegerüst dieser Aufgabe Anwendungsbeispiele in Form des Benutzermenüs (siehe Funktion `showMenu`) vorhanden. Der Ablauf des Programms soll wie folgt sein:

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

Programm 2: Binäruhr

Das zweite Unterprogramm, welches im Hauptmenü dieser Aufgabe ausgewählt werden kann, soll eine Binäruhr realisieren. Binäruhren zeigen mithilfe von LEDs eine im Binärformat kodierte Uhrzeit an. Hierbei werden Stunden, Minuten und Sekunden separat angezeigt. Jede LED ist mit einem Wert assoziiert, sodass die Summe der leuchtenden LEDs den Wert für die Stunden, Minuten bzw. Sekunden bildet.

Zur Realisierung einer Binäruhr dienen neben der Anzeigefunktion `displayClock` die Module `bin_clock` und `led`. Die Funktionalität der Binäruhr soll hierbei im `bin_clock`-Modul implementiert werden, welches über Zugriffsfunktionen die aktuelle Uhrzeit zur Verfügung stellt. Da zur Ausgabe der Binäruhr nicht mehr als 16 LEDs zur Verfügung stehen, wird die Uhrzeit im 12 Stunden Format verwaltet. Es werden jeweils sechs LEDs für den Wert der Sekunden und Minuten sowie vier LEDs für den Wert der Stunden verwendet.

Beachten Sie bei der Implementierung des `bin_clock`-Moduls, dass die Uhrzeiten 00:00 bis 00:59 sowie 12:00 bis 12:59 des 24 Stunden Formats auf 12:00 bis 12:59 im 12 Stunden Format abgebildet werden und somit die Stundenzählung bei 12 begonnen werden muss. Das Anzeigen der Uhrzeit wird durch die `displayClock`-Funktion bewerkstelligt, welche die Uhrzeit zum einen auf dem Display und zum anderen auf dem LED-Bargraph ausgibt. Als erster Schritt soll das `bin_clock`-Modul implementiert werden. Legen Sie zur internen Verwaltung der Uhrzeit in diesem Modul insgesamt vier Variablen für Stunden, Minuten, Sekunden und Millisekunden an. Beachten Sie eine korrekte Dimensionierung der Variablen. Lesen Sie im Kapitel 4.1.4 *Datentypen des Begleitenden Dokuments für das Praktikum Systemprogrammierung* nach, welche Datentypen für den Wertebereich der Variablen geeignet sind und wählen Sie diese geeignet aus.

Die Initialisierungsfunktion `initClock` dient dazu, einen *Timer* des Mikrocontrollers so zu initialisieren, dass dieser in einem Intervall von 10 ms aufgerufen wird. 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 $\frac{1}{20\,000\,000} \cdot 1024 \text{ Sek} = 51,2 \mu\text{s}$ 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 $51,2 \mu\text{s} \cdot 195 \approx 10 \text{ ms}$.

Nachdem der Timer konfiguriert und aktiviert wurde, wird im Takt von 10 ms die Interrupt-Service-Routine `ISR(TIMER0_COMP_vect)` aufgerufen. In dieser ist der Wert der Millisekunden-Variable entsprechend des Timer-Intervalls anzupassen. Ein darauf folgender Aufruf der zu implementierenden Funktion `updateClock` stellt sicher, dass die Werte der übrigen Zeitvariablen korrekt angepasst werden. Erreicht der Wert der Millisekunden-Variable beispielsweise den Wert 1000, so muss die Sekunden-Variable um Eins 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 ermöglicht 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 jedoch 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 \sim Operator, welcher das bitweise Invertieren einer Variable ermöglicht (siehe Kapitel 4.1.13: *Operatoren des Begleitenden Dokuments für das Praktikum Systemprogrammierung*).

Das niederwertigste Bit des Parameters soll auf Pin A0 und das höchstwertige Bit auf Pin D7 ausgegeben werden (siehe Abbildung 1.8). Der Parameter `activateLedMask` ist für dieses Unterprogramm nicht relevant und muss daher nicht berücksichtigt werden.

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

Abbildung 1.8: Ausgabeformat der Binäruhr auf dem LED-Bargraph.

Nach Fertigstellung des Binäruhr- und LED-Moduls können die zur Verfügung gestellten Funktionen in der `displayClock`-Funktion genutzt werden. 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. Die Ausgabe der Uhrzeit auf dem Display soll die Zeitvariablen Stunden bis Millisekunden ausgeben. Zur besseren Lesbarkeit sollen die ausgegebenen Zahlen auf 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.

Zur Ausgabe der Uhrzeit auf dem LED-Bargraphen müssen die einzelnen Zeitvariablen durch Bitshifting so kombiniert werden, dass diese durch 16 LEDs nach dem Prinzip einer Binäruhr angezeigt werden können. Abbildung 1.8 zeigt die Aufteilung der 16-Bit Variable für die Ausgabe auf dem LED-Bargraphen.

Programm 3: Interner AD-Wandler

Nachdem im ersten Teil des Versuchs die grundlegenden Techniken und die Umsetzung von AD-Wandlern behandelt wurden, wird in diesem Teil der in den ATmega 644 integrierte AD-Wandler eingesetzt. 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 kHz liegen. Die Wandlungsfrequenz des integrierten AD-Wandlers kann mithilfe des Registers `ADCSRA` angepasst werden, welches 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: $128 \approx 156 \text{ kHz}$. Weiterhin können verschiedene Pins als Kanal zur Messung einer analogen Spannung gewählt werden. Diese werden beim ATmega 644 als `ADC0` bis `ADC7` bezeichnet und sind mit Pin A0 bis Pin A7 verbunden. Für

diese Aufgabe wurde der Kanal ADC0 gewählt, welcher mit Pin A0 verbunden ist. Detaillierte Erläuterungen über die Konfigurationsmöglichkeiten des internen AD-Wandlers sind im Datenblatt des ATmega 644 in Kapitel 21.9 zu finden.

Die Initialisierungs- und Wandlungsfunktion sind bereits vorgegeben und können in der Funktion `displayAdc` verwendet werden. Diese Funktion soll folgende Aufgaben erfüllen:

- Anzeigen der Spannung in der ersten Zeile des Displays
- Ausgabe der Spannung auf dem LED-Bargraphen (Format siehe Abbildung 1.9)
- Ein Untermenü in der zweiten Zeile des Displays, welche das Speichern und Anzeigen von Spannungswerten ermöglicht

Nach jeder Iteration ist eine Verzögerung von 100ms einzubauen, um eine sowohl flüssige als auch flackerfreie Darstellung der Informationen zu gewährleisten.

Zum Anzeigen der Spannung auf dem Display eignet sich die Funktion `lcd_writeVoltage` des LCD-Treibers, welche selbstständig den diskreten Eingabewert auf eine entsprechend skalierte Gleitkommazahl abbildet und auf dem Display als Spannungswert ausgibt.

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

Abbildung 1.9: Ausgabe der digitalisierten Spannung auf dem LED-Bargraph.

Die Ausgabe des Spannungswertes auf dem LED-Bargraphen soll, wie in Abbildung 1.9 gezeigt, linear erfolgen. Es stehen 15 LEDs zur Verfügung und die obere Schranke des zumessenden diskreten Spannungswert liegt bei 1023. Somit beträgt die Auflösung der Ausgabe pro LED $0,3\text{ V}$ und bezogen auf die diskret hinterlegte Spannung 68. Der Grund dafür, dass bei diesem Unterprogramm nur 15 statt 16 LEDs zur Ausgabe auf dem LED-Bargraphen zur Verfügung stehen liegt darin, dass Pin A0 als Kanal für die Wandlung des analogen Spannungssignals verwendet wird und somit von LED 0 getrennt werden muss.

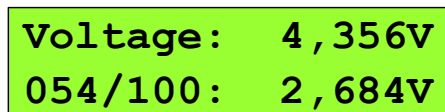
Da Berechnungen auf dem Mikrocontroller vereinfacht durchgeführt werden müssen, bietet sich für die Umformung der binär hinterlegten diskreten Spannung in eine Binärzahl zur Ansteuerung des LED-Bargraphen folgender Ablauf an:

1. Eine 16-Bit Variable `ledValue` mit 0 initialisieren
2. Folgende Schritte solange wiederholen, bis die diskrete Spannung `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 das Bit 0 den Wert 0 beibehält
 - c) Von der Variablen `adcResult` den Wert 68 subtrahieren

Im Anschluss an die Umformung kann die Variable `ledValue` auf dem LED-Bargraphen ausgegeben werden. Dieses Modul muss zuvor jedoch um die Funktion erweitert werden, nur bestimmte LEDs zur Ausgabe zu nutzen, da Pin A0 für die AD-Wandlung verwendet wird und nicht beeinflusst werden darf. Wird dennoch auf diesen Pin zugegriffen, so resultiert dies in einer fehlerhaften Spannungsmessung. Hierzu dient die Variable `activateLedMask` aus der Datei `led.c`, welche eine Bitmaske für aktivierte LEDs darstellt. Es sollen nur solche LEDs ansteuerbar sein, welche mit einer Eins in der Bitmaske gekennzeichnet sind. Passen Sie die Initialisierungsfunktion sowie die Setter-Funktion so an, dass nur diejenigen Bits der DDR- und PORT-Register verändert werden, welche in der `activateLedMask`-Variable mit einer Eins markiert sind. Das Mapping der Bits dieser Variable auf die Pins des Mikrocontrollers ist analog zu dem Parameter der Setter-Funktion (siehe auch Abbildung 1.8).

Neben der Anzeige der momentan gemessenen Spannung soll es nun ermöglicht werden, Messwerte aufzunehmen und diese zu einem späteren Zeitpunkt anzuzeigen. Die dazu benötigte Datenbank soll in der Datei `adc.c` implementiert werden, welche geeignete Zugriffsfunktionen zur Verfügung stellt. Weiterhin muss zur Benutzerinteraktion eine Menüführung in die Funktion `displayAdc` eingebaut werden.

Der Enter-Button soll zum Speichern des aktuellen Spannungswertes genutzt werden. Wird der Button gedrückt, so wird die Funktion `storeVoltage` des ADC-Moduls aufgerufen und der zuletzt gemessene Spannungswert dort gesichert. Weiterhin dienen die Buttons Up und Down dazu, einen internen Zählindex der Funktion `displayAdc` für den aktuell anzuzeigenden Wert zu inkrementieren bzw. zu dekrementieren. Der Wertebereich reicht von 0 bis zum Rückgabewert der Funktion `getBufferSize-1`. Das Ausgeben der Spannungswerte wird in einer separaten Funktion `displayVoltageBuffer` realisiert, die als Parameter den Index des anzuzeigenden Spannungswertes erhält. Die resultierende Ausgabe auf dem Display soll sich an dem Beispiel in Abbildung 1.10 orientieren.



The image shows a green rectangular display area with black text. It contains two lines of text. The first line reads 'Voltage: 4,356V' and the second line reads '054/100: 2,684V'. The text is in a monospaced font.

Abbildung 1.10: Beispielhafte Ausgabe des ADC-Menüs.

Im Folgenden wird die Verwaltung der Datenbank innerhalb des ADC-Moduls beschrieben:

Die Funktion `storeVoltage` dient gleichermaßen zum Speichern von Messwerten als auch zur Initialisierung der Datenstruktur. Diese soll in Form eines Puffers realisiert werden, auf den mit Zeigern zugegriffen werden soll. Ist beim Aufruf der Funktion die Puffergröße gleich 0, so muss zuerst Speicher für den Puffer alloziert werden. Dies wird mithilfe 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, so steht nicht genügend Speicherplatz zur Verfügung und der Puffer kann nicht angelegt werden.

Für diese Aufgabe soll ein Puffer für 100 Spannungswerte angelegt werden. Ein Spannungswert hat die Größe einer `uint16_t`-Variable, also 16 Bit bzw. 2 Byte. Der Aufruf `sizeof(uint16_t)` liefert die Speichergröße einer `uint16_t`-Variable in Byte. Diese Information ist sowohl bei der Speicherallozierung als auch beim Zugriff auf die Speicherbereiche von Bedeutung, da diese angibt, wie viele Speicherstellen der übergebene Datentyp benötigt.

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 1.2 veranschaulicht die Nutzung von Zeigern in diesem Kontext. Zu Beginn 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.

Der Typ `uint16_t*` gibt an, dass es sich um einen Zeiger auf eine 2-Byte-Speicherstelle handelt. Der Zugriff unter Punkt 3 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` stattdessen ein `uint8_t*`, so würde der Zugriff unter Punkt 3 auf die Speicherstelle 274 statt 278/279 erfolgen. Analog zum Schreibvorgang unter Punkt 2 und 3 wird der Lesevorgang unter Punkt 4 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 4.1.7 *Zeiger (Pointer)* entnommen werden. Neben der Speicherung von Spannungswerten muss die als `getStoredVoltage` bezeichnete Zugriffsfunktion implementiert werden, welche den Zugriff auf die interne Datenstruktur des ADC-Moduls ermöglicht. In dieser Funktion muss zunächst überprüft werden, ob der übergebene Index gültig ist, d.h. ob er unterhalb des aktuellen Pufferindizes liegt. Ist dies der Fall, so gibt die Funktion den gewünschten

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

Listing 1.2: Verwendung von Zeigern

Spannungswert zurück, anderenfalls den Wert 0.

1.4.4 Übersicht der Mikrocontroller-Aufgaben

Dieser Abschnitt bildet eine Zusammenfassung der genannten Hausaufgaben und stellt dar, welche Funktionen zu implementieren sind.

AD/DA-Wandlung:

- Erstellung eines AVR-Studio Projektes
- Implementierung des os_input-Moduls
- void manuell(void)
 - Ermöglicht das Ansteuern des R-2R-Netzwerks mit den Schieberegler
 - Zeigt den Status der Schieberegler auf den LEDs an
- void sar(void)
 - Implementierung eines AD/DA-Wandlers nach dem Prinzip eines SAR
 - 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 Trackingwandlers
 - 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 LC-Display
- *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 LC-Display
- *Programm 3: Interner AD-Wandler*
 - Anzeige der gemessenen Spannung auf dem LED-Bargraphen und dem LC-Display
 - Implementierung einer Datenstruktur zur Verwaltung von Messwerten
 - Umsetzung der Benutzerschnittstelle, welche die momentane Messspannung anzeigt sowie das Speichern und Anzeigen von Messdaten ermöglicht

ACHTUNG

Alle in den Aufgaben zur AD/DA-Wandlung und zum ATmega 644 erstellten Projekte für den Mikrocontroller müssen inklusive aller Dateien **24 Stunden vor Versuchsbeginn** mithilfe des Uploadformulars auf den Server des Lehrstuhls geladen werden. Dabei muss der Quelltext bei Verwendung des AVR Studio-Befehls „Rebuild All“ **ohne Fehler oder Warnungen** kompilieren. Der Code darf noch funktionale Fehler enthalten, jedoch muss erkennbar sein, dass **alle Aufgaben ausführlich bearbeitet** wurden. Die für den Versuch angesetzte Anwesenheitszeit wird nicht ausreichen, um alle Aufgaben zu programmieren.

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 in der ersten Kategorie getestet werden. Steuerelemente der *ATMega Remote*-Software ermöglichen das Erzeugen einer Messspannung sowie das Bedienen der Schiebeschalter. Beachten Sie bezüglich der Schiebeschalter, dass im Versuch das Setzen der Pullup-Widerstände notwendig ist. Für das Testen im Testpool müssen diese jedoch nicht gesetzt werden.

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.

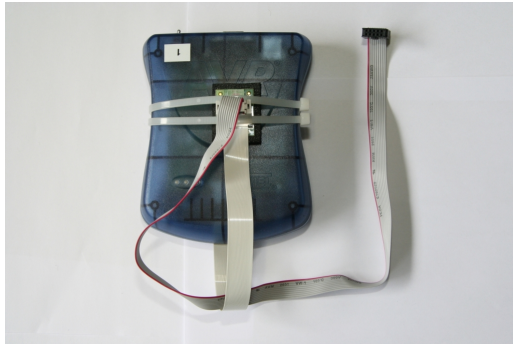
1.6.1 Versuche zur AD/DA-Wandlung

Stellen Sie zunächst sicher, dass der *JTAG Programmierer* (Abb. 1.11) eingeschaltet und mit dem Evaluationsboard verbunden ist.

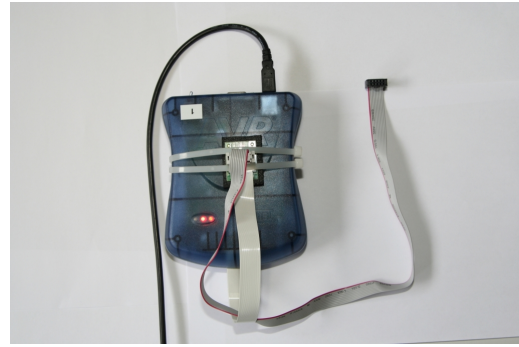
Die Versuchsplatine aus Abbildung 1.12 ermöglicht es, einen AD-Wandler wie beispielsweise den Tracking-Wandler oder den Sukzessive-Approximation-Wandler umzusetzen (vgl. Abbildungen 1.4 und 1.5). Beide Verfahren (Tracking und Sukzessive-Approximation) wurden als Hausaufgabe vorbereitet und werden nun auf der Hardware getestet. 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. Die Eingänge des R-2R-Netzwerks (DA-Wandler) sind über die Platine mit dem Mikrocontroller-Board an Port B verbunden (siehe Abbildung 1.12) und können darüber angesteuert werden. Die Spannungspegel, die der Mikrocontroller liefern kann, betragen 5 V (high) und 0 V (low).

Der Pin C0 an Port C zeigt den Status des Ausgangs des Operationsverstärkers an und wird durch die COMP_OUT-LED (Markierung 5 in Abbildung 1.12) 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.

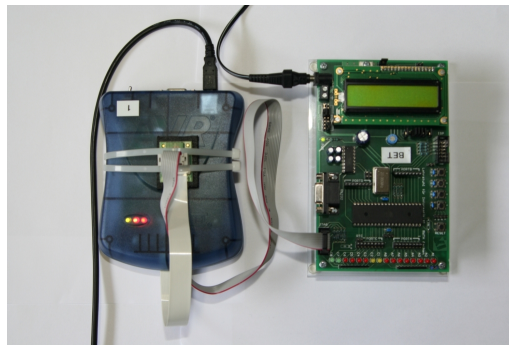
1 Mikrocontroller & AD/DA-Wandlung



(a) ausgeschalteter JTAG Programmierer, LEDs aus



(b) eingeschalteter JTAG Programmierer, 2 rote LEDs



(c) korrekt verbundener JTAG Programmierer, 1 grüne + 2 rote LEDs

Abbildung 1.11: JTAG Programmierer in drei Zuständen

ACHTUNG

Lassen Sie jede Schaltung von einem Betreuer abnehmen, bevor Sie die Spannung einschalten!

Gehen Sie wie folgt vor:

- Schließen Sie die AD/DA-Platine über das Steckernetzteil an die abgeschaltete Spannungsversorgung an (Markierung 1 in Abbildung 1.12).
- Schließen Sie den Buchsenstecker, der an der AD/DA-Platine befestigt ist, an das Evaluationsboard an (Markierung 2 in Abbildung 1.12).

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

Tabelle 1.1: Schalterstellungen für die Überprüfung des R-2R-Netzwerks.

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 1.1 angegebenen Schalterstellungen und notieren Sie jeweils die Ausgangsspannung. Verwenden Sie dafür Messpunkt R2R-OUT (Markierung 6 in Abbildung 1.12) und für die Masse den Messpunkt GND (Markierung 4). Wie genau ist die Wandlung? Wie groß ist die maximal bzw. minimal erzeugbare Spannung?

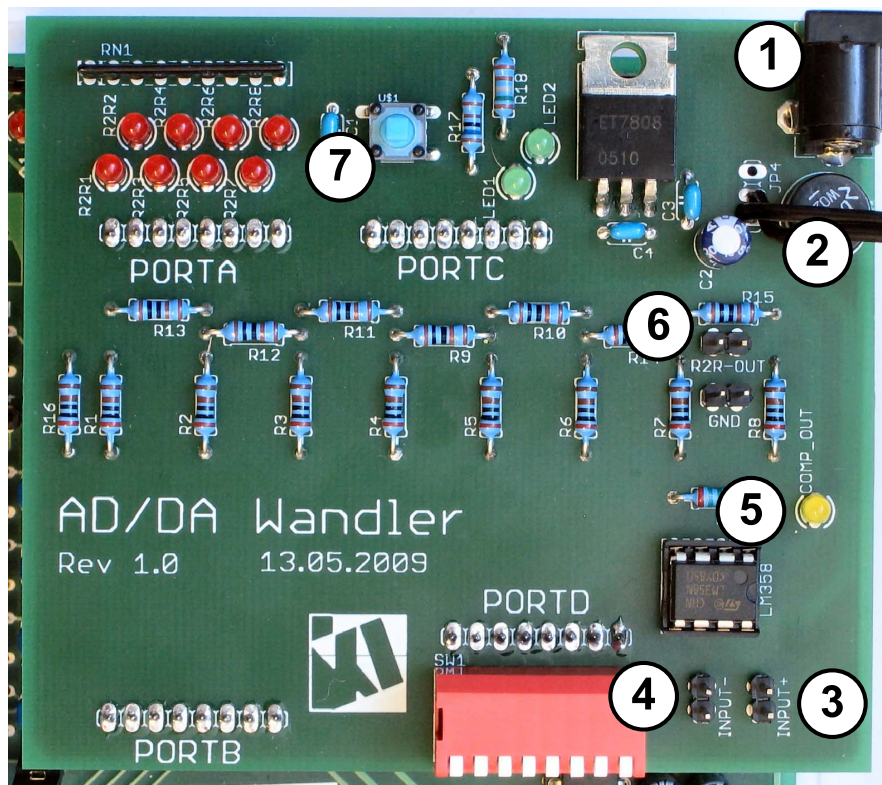


Abbildung 1.12: Auf ein Evaluationsboard aufgesteckte AD/DA-Platine

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 den Ausgang A des Netzgeräts nun auf 0 V ein und schalten Sie das Netzgerät wieder aus.
- Verbinden Sie “+” von Output A mit “Input +” (Markierung 3 in Abbildung 1.12) auf der Platine.
- Verbinden Sie “-” von Output A mit “Input -” (Markierung 4 in Abbildung 1.12) 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 auf geeignete Art die Funktionsweise. Die Wandlung soll erst nach Tastendruck ausgeführt werden und eine Iteration durchführen. Der Taster befindet sich an Pin C1 von Port C. (Markierung 7 in Abbildung 1.12)

Tracking-Wandler

Testen Sie den Tracking-Wandler, den Sie programmiert haben. Überprüfen Sie auf geeignete Art die Funktionsweise. Die Wandlung soll erst nach Tastendruck ausgeführt werden und stoppen, sobald die Messspannung digitalisiert wurde. 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 gleichmäßig blinkender Schriftzug „Hallo Welt!“.

Programm 2: Binäruhr

Für dieses Programm müssen die Kanäle Pin A0 bis A7 mit Jumpers 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.

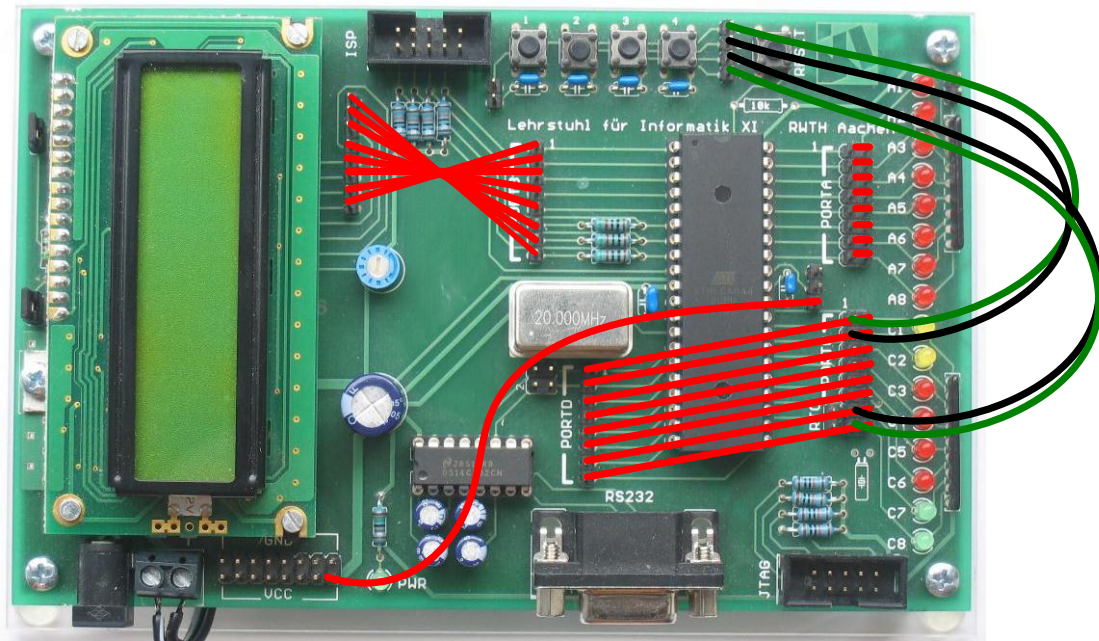


Abbildung 1.13: Verbindungen des Evaluationsboards für die Aufgaben zum ATmega 644.

Programm 3: Interner AD-Wandler

Für den internen AD-Wandler muss der Jumper von Pin A0 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 werden, 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
PORTA	1	LED für Wandlungsergebnis (Bit 1)
	2	LED für Wandlungsergebnis (Bit 2)
	3	LED für Wandlungsergebnis (Bit 3)
	4	LED für Wandlungsergebnis (Bit 4)
	5	LED für Wandlungsergebnis (Bit 5)
	6	LED für Wandlungsergebnis (Bit 6)
	7	LED für Wandlungsergebnis (Bit 7)
	8	LED für Wandlungsergebnis (Bit 8)
PORTB	1	R-2R-Netzwerk (Ausgabe Steuersignale)
	2	R-2R-Netzwerk (Ausgabe Steuersignale)
	3	R-2R-Netzwerk (Ausgabe Steuersignale)
	4	R-2R-Netzwerk (Ausgabe Steuersignale)
	5	R-2R-Netzwerk (Ausgabe Steuersignale)
	6	R-2R-Netzwerk (Ausgabe Steuersignale)
	7	R-2R-Netzwerk (Ausgabe Steuersignale)
	8	R-2R-Netzwerk (Ausgabe Steuersignale)
PORTC	1	Eingang für das Ergebnis des Komparators
	2	Button (Eingabe)
	3	Reserviert für JTAG
	4	Reserviert für JTAG
	5	Reserviert für JTAG
	6	Reserviert für JTAG
	7	LED1 (LED zur freien Verfügung)
	8	LED2 (LED zur freien Verfügung)
PORTD	1	Schiebeschalter 1 (Eingabe für manuelle Wandlung)
	2	Schiebeschalter 2 (Eingabe für manuelle Wandlung)
	3	Schiebeschalter 3 (Eingabe für manuelle Wandlung)
	4	Schiebeschalter 4 (Eingabe für manuelle Wandlung)
	5	Schiebeschalter 5 (Eingabe für manuelle Wandlung)
	6	Schiebeschalter 6 (Eingabe für manuelle Wandlung)
	7	Schiebeschalter 7 (Eingabe für manuelle Wandlung)
	8	Schiebeschalter 8 (Eingabe für manuelle Wandlung)

Pinbelegung für die Aufgaben zur AD/DA-Wandlung.

1.8 Pinbelegung ADC Menü

Port	Pin	Belegung
PORTA	1	LED 0 / Messspannung (Interner AD-Wandler)
	2	LED 1
	3	LED 2
	4	LED 3
	5	LED 4
	6	LED 5
	7	LED 6
	8	LED 7
PORTB	1	frei (schwer zugänglich)
	2	LCD Pin 6
	3	LCD Pin 5
	4	LCD Pin 4
	5	LCD Pin 3
	6	LCD Pin 2
	7	LCD Pin 1
	8	LCD Pin 0
PORTC	1	Button 1: Enter
	2	Button 2: Up
	3	Reserviert für JTAG
	4	Reserviert für JTAG
	5	Reserviert für JTAG
	6	Reserviert für JTAG
	7	Button 3: Down
	8	Button 4: ESC
PORTD	1	LED 8
	2	LED 9
	3	LED 10
	4	LED 11
	5	LED 12
	6	LED 13
	7	LED 14
	8	LED 15

Pinbelegung für die Aufgaben zum ATmega 644.