64-211 Übung Eingebettete Systeme SoSe 2026

Aufgabenblatt 02 Termine: KW 17

Gruppe	
Name(n)	Matrikelnummer(n)

2 Erweiterung der Ein- Ausgabemöglichkeiten

Dieses Aufgabenblatt stellt weitere, zusätzliche Ein- und Ausgabemöglichkeiten eingebetteter Systeme vor: Das Einlesen bzw. die Ausgabe analoger Signale (Spannungen) und die Kommunikation über eine serielle Schnittstelle.

Serielle Schnittstellen

Auf Aufgabenblatt 1 wurde die einfachste Art der Kommunikation mit einem externen Gerät – das bloße Betrachten des Signalpegels der Kommunikationsleitung, z. B. mit Hilfe einer LED – eingeführt. Ein Merkmal eingebetteter Systeme ist die Fähigkeit zur protokollgesteuerten Kommunikation mit externer Hardware. Sensoren, Aktuatoren oder auch andere eingebettete Systeme kommen dabei als Kommunikationspartner in Frage. Hier spielen die **seriellen Protokolle** nach wie vor eine wichtige Rolle. Die bekanntesten sind:

- **UART**: **U**niversal **A**synchronous **R**eceiver/**T**ransmitter
  Beispielsweise **RS-232**: Ein Kommunikationspartner pro Schnittstelle, asynchroner Datentransfer
- **I^{2}C**: **I**nter-**I**ntegrated **C**ircuit
  Serieller Bus, (multi-)**Controller/Target** Kommunikation, synchroner Datentransfer
- **SPI**: **S**erial **P**eripheral **I**nterface
  Serieller Bus, **Controller/Target** Kommunikation, synchroner Datentransfer

Die UART-Schnittstelle nimmt bei den Arduino-Boards eine besondere Rolle ein und wird deshalb in diesem Aufgabenblatt eingeführt, während der SPI- und der I^{2}C-Bus in späteren Aufgaben vorgestellt wird. Alle Arduino-Boards haben mindestens eine serielle Schnittstelle, wobei UART0 immer für die Kommunikation über den USB-Port (auf dem Board integrierter USB-To-Serial Converter) mit der Arduino-IDE verwendet wird. Hierüber wird nicht nur die Firmware auf den Prozessor hochgeladen, sondern es können über die serielle Konsole der IDE (Tools > Serieller Monitor) Eingaben an das ausgeführte Programm übermittelt werden, bzw. vom ausgeführten Programm Ausgaben auf die serielle Konsole initiiert werden.

Das Arduino Framework macht Ihnen die Verwendung der seriellen UART-Schnittstellen einfach. Betrachten Sie die Dokumentation der Bibliothek `Serial` und verschaffen Sie sich einen Überblick über den Ihnen zur Verfügung stehenden Umfang an Funktionen.

```
Serial.begin(⟨speed⟩)                    → Serial.begin
Serial.write(⟨arg⟩)                      → Serial.write
Serial.print(⟨arg⟩)                      → Serial.print
Serial.println(⟨arg⟩)                    → Serial.println
```

Folgender Beispielcode skizziert die Verwendung der `Serial`-Bibliothek:

```
char ch_buffer[80] = "";
float pi_fl = 3.14159265359;

void ftoa(float num, char* res, int numDecPlaces)
{
  //code of ftoa
}

void print_float(float num, int numDecPlaces) {
  //code of print_float
}

void setup()
{
  // Initialisierung der ersten seriellen UART-Schnittstelle (9600 Baud)
  Serial.begin(9600);
  Serial.println("Test␣print␣Pi\n");
  Serial.print("print␣Pi␣using␣print_float:␣");
  print_float(pi_fl, 3);
  Serial.println("␣");
  Serial.print("print␣Pi␣using␣ftoa:␣");
  ftoa(pi_fl, ch_buffer, 5);
  Serial.println(ch_buffer);

  Serial.print("\ncross␣check:␣Serial.println(pi_ch,␣5):\t");
  Serial.println(pi_fl, 5);
}

void loop()
{
  //code of mainloop
}
```

Aufgabe 2.1 Ausgabe eines float Wertes auf die serielle Konsole

Vergegenwärtigen Sie sich den obigen Beispielcode und ergänzen Sie den Code für die Funktionen ***ftoa*** und ***print_float***. ***ftoa*** soll einen float-Wert entgegen nehmen und in Form eines Character Arrays als druckbare Zeichen abspeichern. Dabei soll die Anzahl von berücksichtigten Nachkommastellen konfigurierbar sein. ***print_float*** verhält sich ähnlich, aber schreibt das Re-

sultat direkt auf die serielle Schnittstelle. Die serielle Konsole ist Teil der Arduino-IDE (Tools > Serieller Monitor).

Analoge Ein- und Ausgabespannungen

Im Aufgabenblatt 1 (Aufgabe 1.4, 1.5) wurde anhand einer an einem Eingabepin des Mikroprozessors anliegenden Spannung auf eine Betätigung eines Tasters geschlossen. Die Skizze in Abb. 1 verdeutlicht diesen Sachverhalt nochmals. Zur Erfassung der Eingangsspannung haben Sie bei dieser Aufgabe die Arduino-Funktion

```
digitalRead(⟨pin⟩)                                    → digitalRead
```

verwendet. Diese Funktion liefert einen boolschen Rückgabewert, also **HIGH** oder **LOW**. Der Rückgabewert **LOW** wird generiert, wenn die Eingangsspannung $U_e \leq 0.3V_{CC}$ ist und der Rückgabewert **HIGH** wird generiert, falls $U_e \geq 0.7V_{CC}$ (siehe SAM3X_Datasheet, p.1379). Im Unsicherheitsbereich ([0.3V . . 0.7V]; bei $V_{CC} = 3.3V$) wird natürlich auch einer der beiden Werte **HIGH** oder **LOW** ausgegeben; nur wird die Plausibilität des Rückgabewertes nicht mehr garantiert, und die Verantwortung wird an den Entwickler der externen Beschaltung delegiert, der sicherzustellen hat, dass der Unsicherheitsbereich bei einem Pegelwechsel möglichst schnell durchlaufen wird.

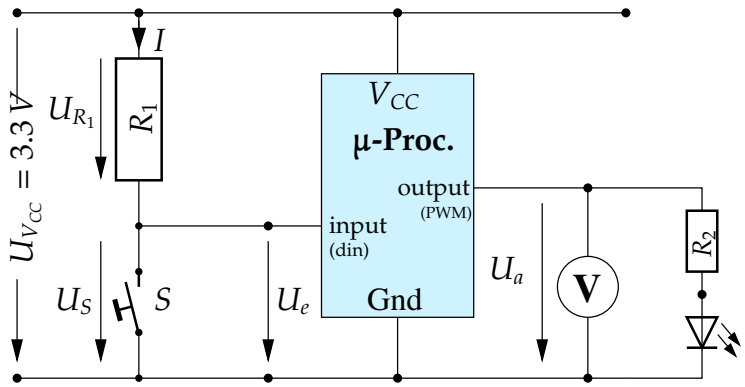

Abbildung 1: Erfassen der Eingangsspannung U_e und Generieren der Ausgangsspannung U_a

Entsprechend sind die durch die Arduino-Funktion

```
digitalWrite(⟨pin, value⟩)                            → digitalWrite
```

erzeugten Ausgabesignale der bisherigen Aufgaben ebenfalls von rein binärer Natur. Entweder wurde zwischen Ausgabepin und GND eine Spannung von etwa 0V oder eine Spannung von ~3.3V erzeugt (siehe Abb. 1).

Einlesen analoger Signale

Häufig sind die in eingebetteten Systemen verarbeiteten Signale **analoger**, also **kontinuierlicher** Natur. Bei Eingaben ist es besonders häufig der Fall, wenn es sich um Signale externer Komponenten (z.B. Sensoren wie einen **Joystick**) handelt. Um analoge Signale innerhalb eines

digitalen Systems verarbeiten zu können, ist eine Analog-Digital-Wandlung (de.wikipedia.org/wiki/Analog-Digital-Umsetzer) durch Abtastung/Quantisierung notwendig. Die Aufgabe der Analog-Digital-Wandlung ist es, ein analoges Eingangssignal in eine diskrete Repräsentation zu wandeln. Die A/D-Konverter der Arduino-Prozessoren sind standardmäßig auf 10 bit-Wortbreite konfiguriert, d. h. für die Quantisierung stehen einschließlich der Stufe 0 1024 Schritte zur Verfügung.

`analogRead(⟨pin⟩)` → `analogRead`

Die Funktion analogRead liefert also einen ganzzahligen Rückgabewert im Bereich [0..1023] abhängig von der Eingangsspannung am Pin. Nicht jeder Pin am Arduino verfügt über einen ADC. Die analogen Eingangspins des Arduinos sind als solche beschriftet, lassen sich aber auch als digitalpins nutzen. Es ist weiterhin nötig, den pinMode auf INPUT zu setzen.

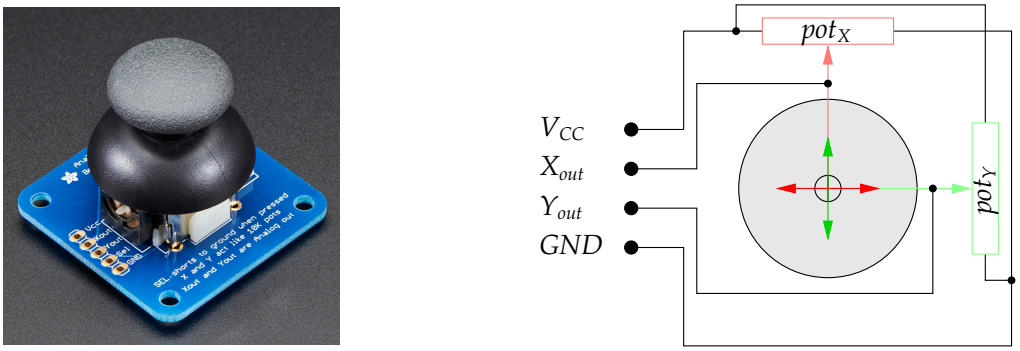

Abbildung 2: 2-Achsen Joystick basierend auf zwei Drehpotentiometern.

Als Beispiel für einen analogen Sensor ist in Abb. 2 ein Joystick gezeigt, der zwei analoge Spannungen jeweils proportional zur Auslenkung der x- und y-Achse, erzeugt. Die beiden Spannungen werden durch Potentiometer (veränderbare Spannungsteiler, de.wikipedia.org/wiki/Potentiometer) erzeugt.

Aufgabe 2.2 Messen einer analogen Spannung

Implementieren Sie ein Programm, dass es Ihnen ermöglicht, die am Analog-Pin *A9* anliegende Spannung einzulesen und geben Sie bei Veränderung des Wertes die Spannung in Volt [V] und auch in Millivolt [mv] auf der seriellen Konsole aus. Die Ausgabe könnte etwa folgendermaßen aussehen: ***pin A9 – ADC-value: 257; input Voltage: 1.25 V, 1256 mV***.
Die analoge Spannung generieren Sie beim praktischen Aufbau beispielsweise mit Hilfe eines der beiden Potentiometer des Joysticks (siehe Abb. 2.2).

Ausgeben analoger Signale

Wie bereits das Auswerten eines analogen Eingangssignals weist auch das Generieren eines analogen Ausgangssignals einige Besonderheiten auf. Viele auf den Arduino-Boards verwendeten Prozessoren weisen zwar einen oder zwei Digital-Analog-Converter (DAC) auf, allerdings werden diese nur in besonderen Anwendungen eingesetzt, z.B. wenn für eine Referenzspannung oder eine leistungslose Spannungssteuerung eine besonders genaue Analogspannung benötigt

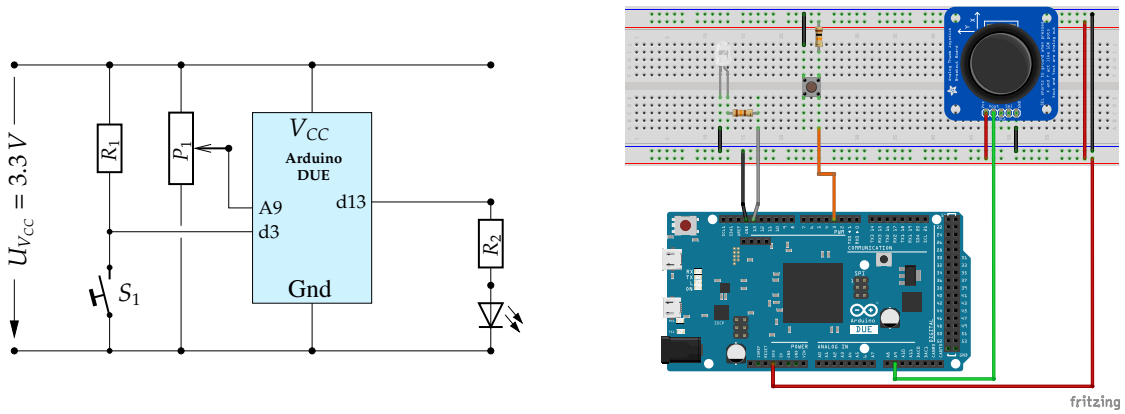

Abbildung 3: Vorschlag für die Verdrahtung des Versuchsaufbaus zu Aufg. 2.2
links: Schaltplan, rechts: praktischer Aufbau

wird.
Soll beispielsweise die Helligkeit einer LED verändert werden, ist es zum einen von der Auflösung her nicht erforderlich und zum anderen aufgrund der geringen Treiberleistung des ADC-Ausganges auch gar nicht möglich! Diese Aufgabe ließe sich mit der Technik der **Pulsweitenmodulation (PWM)** lösen.

Ein digitaler Anschlusspin, konfiguriert als Ausgang, lässt sich verwenden, um eine angeschlossene Komponente entweder mit `GND` (0 V, Logikpegel `LOW`) oder mit `VCC` (3.3V, Logikpegel `HIGH`) zu verbinden. Liefert eine Komponente aufgrund ihrer Beschaltung (z.B. eine LED) bei 3.3V Spannung 100 % ihrer Leistung, so besteht zunächst nur die Möglichkeit, diese Komponente auf 0 % (`LOW`) oder 100 % (`HIGH`) Leistung einzustellen. Jedoch ist gerade bei LEDs die Möglichkeit der variablen Steuerung der Leistung (Leuchtintensität) sehr interessant.

Eine Art der Leistungsregelung ermöglicht die **Pulsweitenmodulation (PWM)** des einfachen Rechtecksignals. Dabei wird die Frequenz des Rechtecksignals derart gewählt, dass aufgrund der Trägheit der Komponente, diese das periodische Ein-/Ausschalten mit einer sich verändernden Pulsweite als lineare Veränderung der Spannungsversorgung interpretiert. Die Trägheit der Komponente agiert dabei als Tiefpassfilter (**Hinweis:** Im Falle der LED ist es das menschliche Auge, welches als Tiefpassfilter agiert).

Abbildung 4 stellt ein pulsweitenmoduliertes Signal mit einem Tastverhältnis von ca. 50 %, generiert mit dem Mikrocontroller des Arduino Due. Das Tastverhältnis stellt die Impulsdauer im Verhältnis zur Periodendauer dar. Ein Tastverhältnis von 50 % bedeutet also: innerhalb einer Periode des Rechtecksignals hat dieses den Pegel `HIGH` für 50 % der Periodendauer. Integriert man über die Periode, so erhält man die über den Tiefpass wahrgenommene Spannungshöhe. Weitere Information zu dem Thema der Pulsweitenmodulation finden Sie unter `de.wikipedia.org/wiki/Pulsweitenmodulation`.

Das pulsweitenmodulierte Signal wird im Mikrocontroller in der Regel mit Hilfe von PWM-Generatoren oder Hardware-Timern erzeugt. Das Arduino Framework macht Ihnen die Verwendung dieser Blöcke einfach. Verwenden Sie für Ihr Programm folgende Funktion:

```
analogWrite(⟨pin⟩, ⟨value⟩)                                    → analogWrite
```

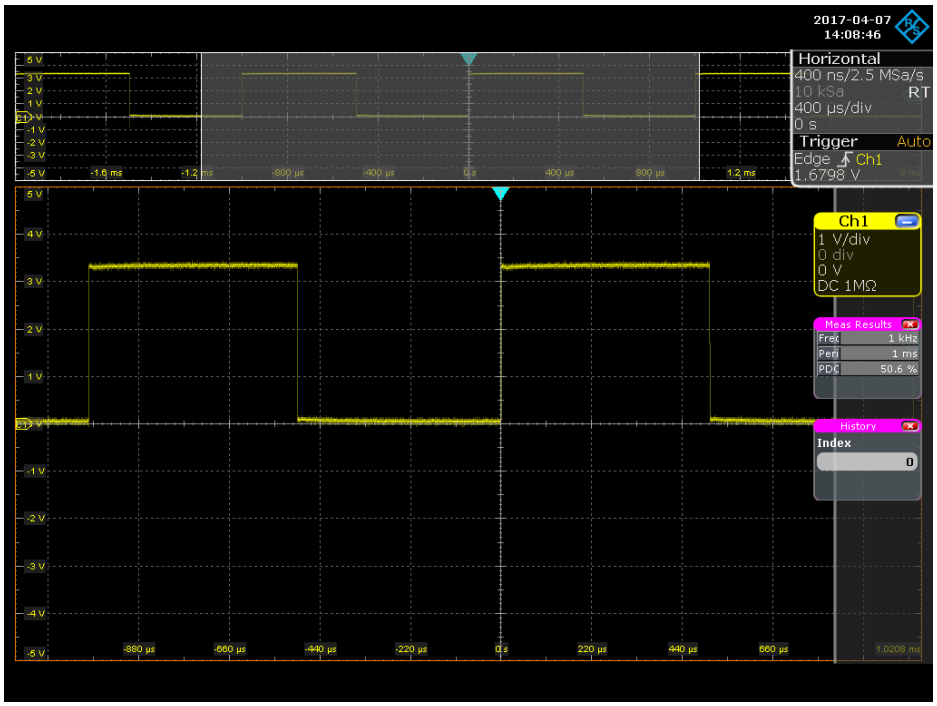

Abbildung 4: Pulsweitenmoduliertes Signal (@ 1 kHz).

Beachten Sie: Bei der Verwendung der Funktion `analogWrite(`⟨*pin*⟩`,` ⟨*value*⟩`)` ist es nicht notwendig, den verwendeten Anschlusspin explizit als Ausgang zu konfigurieren; dies übernimmt die Funktion intern. Nicht alle Pins am Arduino unterstützen PWM. Beim Due sind das nur die Pins 2-13 (sind auch als solche markiert). Value ist ein Integer [0..255] wobei 0 ganz aus und 255 ganz an entspricht.

Aufgabe 2.3 Erzeugen einer analogen Spannung

Erweitern Sie die vorige Aufgabe um die Erzeugung eines pulsweitenmodulierten Ausgangssignals, das der gemessenen Eingangsspannung entspricht. Bei einer anliegenden Eingangsspannung von 3.3V sollte das Tastverhältnis also 100% betragen, bei einer Eingangsspannung von 1.65V entsprechend 50% usw.
Steuern Sie auf diese Weise die Helligkeit der LED aus Augabe 1.6 mittels eines Potentiometers (in Abb. 2.2 wird hierfür das X-Potentiometers des Joysticks verwendet). Dimensionieren Sie den Vorwiderstand der LED (R_2 in Abb. 2.2) wie in Aufgabe 1.2 berechnet.

Serielle Schnittstellen (Fortsetzung)

Im Abschnitt 2 dieses Aufgabenglattes wurde die UART-Schnittstelle lediglich zur Ausgabe von Programminformationen zur Ausgabe von Programminformationen auf die serielle Konsole der Arduino-IDE eingesetzt. Die UART-Schnittstelle erlaubt allerdings nicht nur eine unidirektionale Datenübertragung, sondern sie erlaubt auch eine bidirektionale Kommunikation. So verfügt die Arduino `Serial`-Library auch über Funktionen, die das Lesen vom seriellen Port erlauben.

Zusätzlich zu den bereits bekannten Funktionen sind hier insbesondere die `Serial.available`- und die `Serial.read`-Funktion interessant:

```
Serial.begin(⟨speed⟩)                    → Serial.begin
Serial.available()                       → Serial.available
Serial.read()                            → Serial.read
Serial.write(⟨arg⟩)                      → Serial.write
Serial.print(⟨arg⟩)                      → Serial.print
Serial.println(⟨arg⟩)                    → Serial.println
```

Bitte beachten Sie, dass sich die seriellen Monitore der Arduino-IDEs untereinander, andere Terminals für serielle Kommunikation () und das virtuelle Proteus-Terminal sich wiederum leicht unterschiedlich zu einander verhalten:

- Die seriellen Monitore der Arduino-IDEs der Versionen < 2.0 übertragen die Zeichen erst nach Eingabe von **`ret`**, bei der Version 2.0 erst nach Eingabe von **`strg + ret`**. Soll die Eingabe zur Erkennung des Zeilenendes zusätzlich durch ein **LF** (line feed – 0x0A) oder **CR** (carriage return – 0x0D) abgeschlossen werden, wählen Sie bitte im `Serial Monitor` die Option `New Line` oder `Carriage Return`.
- Das virtuelle Terminal von Proteus 8 überträgt jedes eingetippte Zeichen sofort und die Eingabe von **`ret`** resultiert in der Übertragung von **LF** (0x0A), während ein **`strg + ret`** die Übertragung von **CR** (0x0D) bewirkt. Wenn Sie die eingegebenen Zeichen vom Terminal selbst angezeigt bekommen möchten, aktivieren Sie bitte den Echo-Mode.

Aufgabe 2.4 Entwicklung eines String-Parsers

Entwickeln Sie einen String-Parser, welcher ein über das Eingabefeld des seriellen Monitors abgeschicktes Kommando entgegennimmt und dieses auf Korrektheit überprüft. Bei einem korrekt erkannten Kommando veranlassen Sie die Ausführung der Funktion, andernfalls geben Sie, soweit möglich, eine möglichst aussagekräftige Fehlermeldung auf der seriellen Konsole aus. Nutzen Sie auch die Dokumentation der `Serial`-Bibliothek und die Dokumentation ihrer Funktionen (s. o.).
Der Befehlssatz soll folgende Kommandos enthalten:

- `help()`
  Listet die gültigen Befehle mit Angabe von Information zur Nutzung der Befehle in der Ausgabe des seriellen Monitors auf.
- `LEDon()`
  schaltet die LED mit dem voreingestellten Helligkeitswert ein.
- `LEDoff()`
  schaltet die LED aus
- `illuminance(value)`
  Lichtintensität (`value`) der LED in [%]

Die Namen der Befehle dürfen Sie, insbesondere für Proteus, gerne (tipp)ökonomischer wählen.

ACHTUNG: Vergessen Sie nicht, die Benutzereingabe auf Korrektheit zu überprüfen! Für erkannte Eingabefehler soll ein „sinnvolles Feedback“ ausgegeben werden. Berücksichtigen Sie

bei der Entwicklung des Parsers eine leichte Erweiterbarkeit des Befehlssatzes für spätere Aufgaben.

Erweitern Sie die Aufgabe 2.3 um die zusätzliche Steuerung der LED über die serielle Konsole mit obigen Kommandos. Es soll parallel die Steuerung über Taster und Poti erhalten bleiben.