64-211 Übung Eingebettete Systeme SoSe 2025

Aufgabenblatt 01 Termine: KW 17

Gruppe	
Name(n)	Matrikelnummer(n)

1 Ein-/Ausgabe digitaler Signale

Bereits auf dem optionalen Aufgabenblatt 0 wurde für die Ausgabe von Signalen bei dem ersten, beispielhaften Aufbau zur Demonstration eines Arduino-Sketches (siehe Blatt 0, Abb. 2 und 1) unkommentiert eine Leuchtdiode mit Vorwiderstand verwendet. Dieses Aufgabenblatt befasst sich nun mit elementaren elektrotechnischen Grundlagen, beispielsweise der Berechnung des o.g. Vorwiderstandes oder der Grundlagen, die beim Einlesen bzw. bei der Ausgabe digitaler Signale berücksichtigt werden müssen.

Aufgabe 1.0 Erstellen eines Blink-Sketches auf echter Hardware

Öffnen die die Arduino-IDE (auf den Uni-Rechnern bitte 1.8 benutzen, hier herunterladen) und darin via File > Examples > 01.Basics > Blink das Beispiel-Blink-Sketch. Schließen Sie den Arduino Due mit dem **Programming-Port** des Boards am Rechner an. Wählen Sie in der Arduino IDE unter Tools > Board > Arduino ARM (32 Bit) Boards > Arduino Due (Programming Port) das Board aus und selektieren unter Tools > Port den genutzten Port. Falls das Due Board nicht installiert ist, über Tools > Board > Board Manager nachinstallieren. Laden Sie den Sketch mittels der Taste mit dem Pfeil oder Ctrl+U auf das Arduino Board hoch und überprüfen, ob die LED (L) auf dem Aduino Board im sekundentakt blinkt.

Aufgabe 1.1 Eigene, komplexere Blink-Sequenz

Modifieren und erweitern Sie den Sketch aus Aufgabe 1.0 so, dass eine eigene, komplexere Blinksequenz dargestellt wird wie z.B. ein Morsecode.

Dabei kann die folgende Funktion sehr hilfreich sein:

* delay(⟨*ms*⟩) → delay

Sie unterbricht die Ausführung des Programms für `ms` Millisekunden. Auch macht es bei dieser Aufgabe Sinn, Gebrauch von for-Schleifen zu machen.

Etwas Elektrotechnik

Für die Lösung der folgenden Aufgaben könnte ein Versuchsaufbau, wie in Abbildung 1 exemplarisch mit einem Arduino-DUE Board dargestellt, aussehen. Mit diesem Aufbau lassen sich die folgenden Teilaufgaben dieses Blattes ohne Änderungen der Verdrahtung umsetzen.

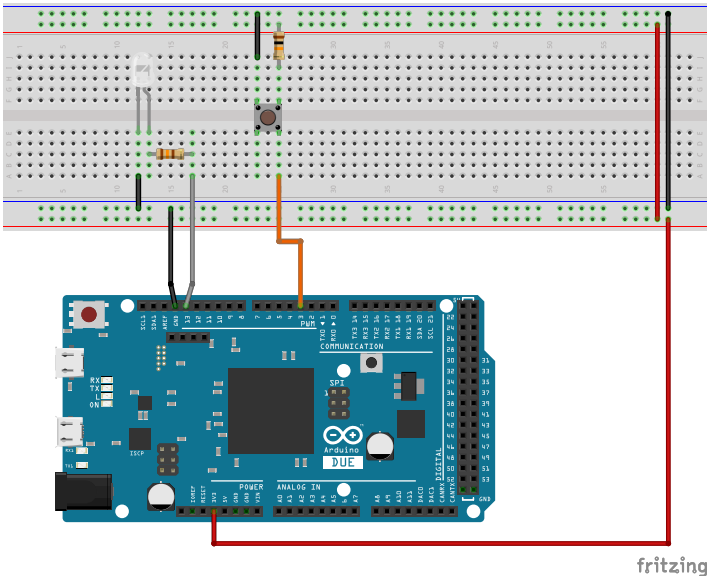

Abbildung 1: Vorschlag für die Verdrahtung des Versuchsaufbaus. Achten Sie auf die Farbmarkierungen der Widerstände in der Schaltung und die Einbaurichtung der LED.

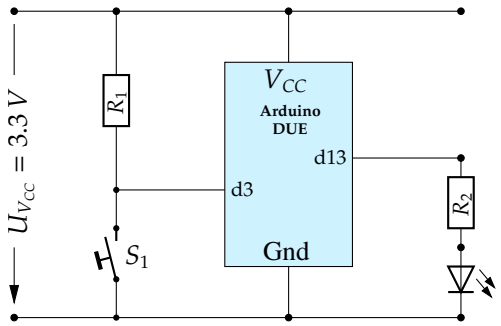

Abbildung 2: Dazugehöriger Schaltplan

Hinweis zur Verschaltung von Steckbrett (Abb. 3) und Taster (Abb. 4):
Die Steckkontakte zwischen der horizontalen roten und blauen Linie am unteren bzw. am oberen Rand des Steckbrettes sind jeweils horizontal durchkontaktiert und die sich daraus ergebenden beiden Zeilen pro Bereich stellen quasi „Stromschienen“ dar. Die übrigen Kontakte des Verdrahtungsfeldes sind spaltenweise jeweils in 5er-Gruppen verbunden.

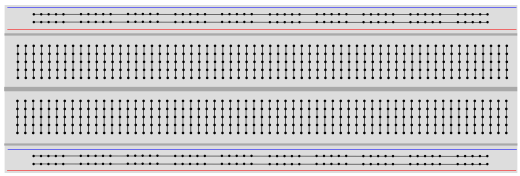

Abbildung 3: Verschaltung des Steckbrettes

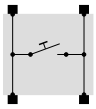

Abbildung 4: Verschaltung des Tasters (gleiche Orientierung wie im Verdrahtungsvorschlag)

Aufgabe 1.2 Schaltung mit Vorwiderstand

In dieser Aufgabe werden Sie mit dem Mikrocontroller die auf dem Steckbrett platzierte LED (Light-Emitting Diode) ansteuern (siehe Abb. 1). Hierbei handelt es sich um ein Bauelement, das keine annähernd lineare I-U-Kennlinie (graphische Darstellung der Strom-Spannungs-Paare) aufweist. Im Falle der Diode lässt sich der Zusammenhang von Strom und Spannung durch eine exponentielle Funktion approximieren.

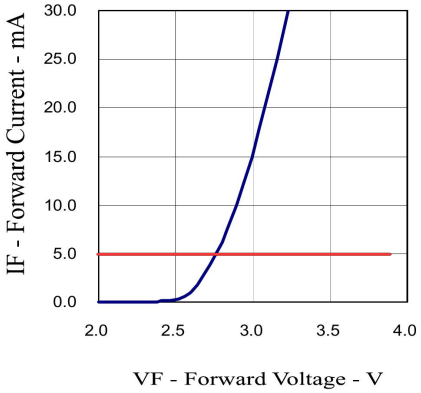

Abbildung 5: Kennlinie der LED (Datenblatt: L200-SIW-20D_LL)

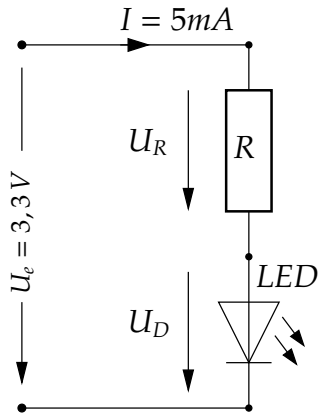

Abbildung 6: LED mit Vorwiderstand

Abb. 5 zeigt die Kennlinie der im Versuch verwendeten LED. Ein linearer Anstieg der Spannung über der LED hat also einen exponentiellen Anstieg des Stromes durch die Diode zu Folge. Die Ausgänge z. B. des Arduino DUE liefern Spannungswerte bis zu 3.3V, was also einen Stromfluss durch die Diode (siehe Abb. 5) von $> 30mA$ zur Folge hätte; dabei möchten wir den Strom zum Schutz der Ausgangstreiber des Arduino und der Diode selbst auf 5 mA begrenzen. Eine Lösung wäre auch hier die Verwendung eines Vorwiderstandes, wie in Abb. 6 gezeigt.
Berechnen Sie den erforderlichen Lastwiderstand R. Die Durchlassspannung U_D der Diode im geforderten Arbeitspunkt entnehmen Sie der Abb. 5. Zur Berechnung des benötigten Widerstands hilft uns das Ohmsche Gesetz $R = U/I$ mit dem der Widerstand R berechnet werden kann, welchen es für einen Spannungsabfall U mit einem Strom I braucht.

(a) Definieren Sie die folgenden Größen:
$U_e =$
$I =$
$U_D =$

(b) Berechnen Sie:
$U_R =$
$R = U_R / I =$

Hinweis: Der in Abbildung 1 gezeigte Vorwiderstand von 130 Ω ist nicht die Umsetzung der exakten Berechnung, sondern lediglich der Tatsache geschuldet, dass dieser gerade in ausreichender Zahl vorhanden war. Weshalb ist dieses laxe Vorgehen im Falle des Widerstandes möglich?

Aufgabe 1.3 Aufbau einer Schaltung auf dem Breadboard

Erstellen Sie in Anlehnung an Abb. 1 den Aufbau zum Test des Blink-Sketches mit einer separaten LED mit eigenem Vorwiderstand. Achten Sie darauf, dass der richtige Pin (normalerweise 13) des Arduino an die LED angeschlossen wird. Sie sollte nun parallel zur LED auf dem Board selbst blinken.

Arduino `setup()` und `loop()`

Beim Blick in das Blink-Sketch fällt auf, dass zwei Funktionen definiert werden: setup und loop. Diese sind grundsätzlich Teil eines jeden Arduino-Sketches. Immer wenn man den Arduino startet, wird erst `setup()` aufgerufen. Diese wird genutzt, um Pins zu konfigurieren, Objekte anzulegen, auf denen später gearbeitet wird und einige Variablen zu definieren. Sie bereitet also alles für die `loop()`-Funktion vor. In `loop()` befindet sich der code, der also stetig in einer Endlosschleife ausgeführt wird nachdem, das System in `setup()` initialisiert wurde. Natürlich können andere Funktionen daneben definiert werden, die dann in `setup()` oder `loop()` aufgerufen werden.

Digitalpins des Arduino

Nahezu alle (aber nicht ganz alle!) Pins am Arduino lassen sich als digitale Ein- und Ausgabepins nutzen. Das bedeutet, dass man an ihnen ein digitales Signal (**HIGH** oder **LOW**) detektieren oder anlegen kann.

Für den grundlegenden Umgang mit Digitalpins gibt es drei Funktionen:

```
* pinMode(⟨pin⟩, ⟨mode⟩)                    → pinMode
* digitalRead(⟨pin⟩)                       → digitalRead
* digitalWrite(⟨pin⟩, ⟨value⟩)             → digitalWrite
```

Bevor ein Pin verwendet werden kann, muss der pinMode gesetzt werden. Dies geschieht üblicherweise, aber nicht immer, in der `setup()` Funktion. Verfügbare Pin-Modi sind: `OUTPUT` (macht den Pin zu einem Ausgangspin, den man mit digitalWrite mit den Werten HIGH oder

LOW belegen kann), `INPUT` (macht den Pin zu einem Eingangspin, von dem man mit digitalRead lesen kann) und `INPUT_PULLUP` (gleich wie INPUT, aber aktiviert einen internen Pullup Widerstand).

Aufgabe 1.4 Schaltung mit Pullup-Widerstand

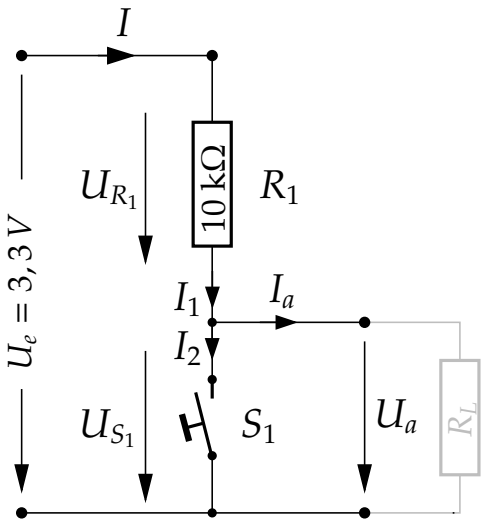

Abbildung 7: Taster mit Pullup-Widerstand

Vergegenwärtigen Sie sich die Schaltung nach Abb. 7 und stellen Sie den Bezug zu Abb. 1 her.

(a) Welche Aufgabe hat der Widerstand R_1?

(b) Welche Spannungswerte erwarten Sie am Ausgang der Schaltung (U_a):
Taster S_1 gedrückt: $U_a =$
Taster S_1 Ruhestellung: $U_a =$

(c) Wie hoch ist der Querstrom I ($I = I_1 = I_2$; mit $I_a \to 0$)
Taster S_1 gedrückt: $I =$
Taster S_1 Ruhestellung: $I =$

Die Werte für die Eingangsspannung U_e und für den Widerstand R_1 entnehmen sie der Skizze in Abb. 7. Weiterhin soll angenommen werden, dass der Ausgangsstrom I_a vernachlässigbar klein ist ($I_a \to 0$).

Aufgabe 1.5 Verarbeitung von Eingaben mittels Polling

Erweitern Sie das Blink-Sketch um eine periodische Abfrage des Logikpegels am Anschlusspin des angeschlossenen Tasters (siehe Abb. 1). Wird der Taster betätigt, so soll die externe LED ein- bzw. ausgeschaltet werden (*state toggle*). Beachten Sie: **Eine** Betätigung des Tasters soll auch **nur einen** Zustandswechsel hervorrufen - die Dauer der Betätigung soll keinen Einfluss haben.

Bedenken Sie auch, dass an einem input Pin zum Auslesen eines Tasters jederzeit ein Spannungspegel anliegen muss, der eindeutig einem der Logikpegel **LOW** oder **HIGH** zugeordnet werden kann. Die im Verdrahtungsvorschlag abgebildeten Taster schließen bei Betätigung gegenüber dem Masse-Potenzial (`GND`, 0 V) kurz; es wird sicher ein **LOW** detektiert werden. Allerdings muss sichergestellt werden, dass im nicht betätigten Zustand des Tasters am Anschlusspin eine Spannung $\geq 0.7V_{CC}$ anliegt, damit sicher **HIGH** detektiert wird. Dieses kann mit einem **Pull-up Widerstand** umgesetzt werden. Angeschlossen an die Versorgungsspannung (`VCC` = 3,3 V) und unter der Annahme, dass der Ausgangsstrom $\rightarrow$ 0 geht, dürfte im nicht betätigten Zustand des Tasters eine Ausgangsspannung von $\rightarrow$`VCC` anliegen und somit sicher **HIGH** detektiert werden. Wird der Taster betätigt, so erfolgt eine Verbindung gegenüber dem Masse-Potenzial und der durch den **Pull-up Widerstand** begrenzte Strom fließt gegen Masse ab. Am Anschlusspin des Tasters liegt in diesem Fall eine Spannung $\rightarrow$ 0 an und es wird sicher ein **LOW** detektiert (siehe auch Aufgabe 1.4). Weitere Information zu dem Thema können Sie optional auch unter `rn-wissen.de/index.php/Pullup_Pulldown_Widerstand` erhalten.

Alternativ zu dem oben skizzierten Vorgehen lässt sich mittels Pin Mode `INPUT_PULLUP` eine Lösung entwickeln, die einen internen Pull-up Widerstand nutzt und somit keinen externen Widerstand am Taster benötigt. Gestalten Sie das Programm und die Schaltung entsprechend um.

Aufgabe 1.6 Hardware Interrupt auslösende Digitalpins

Die für die Aufgabe 1.5 entwickelte Lösung, bei der der Taster ständig, wiederkehrend mit `digitalRead` abgefragt wird (Polling), hat einen grundlegenden Nachteil. Erläutern Sie diesen.

Hardware Interrupts sind Interrupts, die als Reaktion auf ein externes Ereignis ausgelöst werden. So besitzen die Prozessoren der Arduino-Boards ausgewählte Inputpins, die in der Lage sind, auf einen Wechsel bzw. auf einen bestimmten Zustand des anliegenden Signals zu reagieren und einen Interrupt auszulösen.
Ein Hardware-Interrupt ist ein (in der Regel) asynchrones Ereignis, das die Ausführung des Programmcodes unterbricht und zu einer dem Interrupt zugewiesenen Interruptroutine springt. Nach Ausführung der Interruptroutine erfolgt ein Rücksprung an die Stelle im Programmcode, an der die Unterbrechung ausgelöst wurde.

Entwerfen Sie eine abgewandelte Variante des bisherigen Programms, in der Sie die Betätigung des Tasters als **Hardware-Interrupt** innerhalb einer entsprechenden **Interruptroutine** behandeln.

Für Ihr Programm werden die folgende Funktone benötigen:

* attachInterrupt(digitalPinToInterrupt(⟨*pin*⟩), ⟨*function*⟩, ⟨*mode*⟩) $\rightarrow$ attachInterrupt

Dabei ist `pin` der Pin, an den der Button angeschlossen ist, `function` die Funktion, die aufgerufen wird, wenn der Interrupt ausgelöst wird und `mode` der Modus des Interrupts, also der Typ Ereignis, das ihn auslöst. Zur Verfügung stahen folgende Modi:

- **LOW**: Löst aus, wenn der Pin mit LOW belegt ist.
- **HIGH**: Löst aus, wenn der Pin mit HIGH belegt ist. (Nicht bei allen Boards verfügbar, aber beim Due schon.)
- **CHANGE**: Löst aus, wenn sich die Belegung des Pins ändert.
- **RISING**: Löst aus, wenn der Pin mit LOW belegt war und zu HIGH wechselt.
- **FALLING**: Löst aus, wenn der Pin mit HIGH belegt war und zu LOW wechselt.

In der Arduino Referenz unter `https://docs.arduino.cc/language-reference/en/functions/external-interrupts/attachInterrupt/`, gibt es weitere Informationen zu Interrupts.

Welche Aufgabe hat die Funktion *digitalPinToInterrupt(pin)*, als erster Parameter von *attachInterrupt()*? Bei welchen Prozessoren kann die Funktion entfallen?

Anhang für Proteus-Nutzer

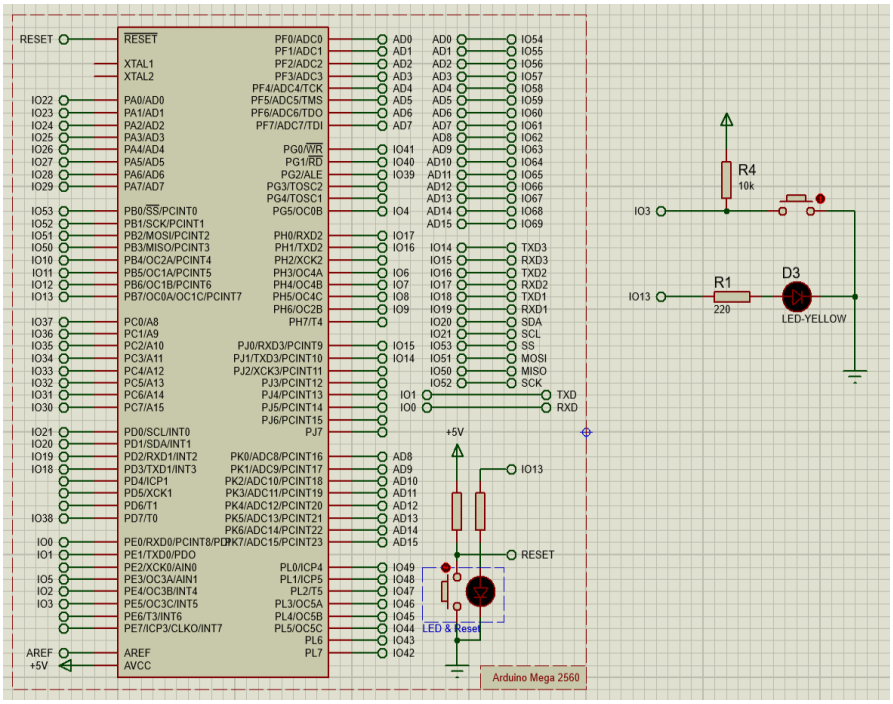

Abbildung 8: Proteus Firmware-Projekt: Aufbau zum Ein- und Ausschalten einer LED mittels Taster