64-211 Übung Eingebettete Systeme

Aufgabenblatt 2 Termine: KW 15

Gruppe	
Name(n)	Matrikelnummer(n)

Bei der Bearbeitung des Aufgaben 1.3 und 1.4 des ersten Aufgabenblattes sollte Ihnen aufgefallen sein, dass eine einzige Betätigung des Tasters in vielen Fällen zu mehreren Zustandswechseln der LED geführt hat. Dieses Verhalten ist auf das **Prellen** (de.wikipedia.org/wiki/Prellen) des Tasters zurückzuführen. Dabei kommt es im Inneren des Tasters zum mehrfachen Kontaktschluss bedingt durch mechanische Eigenschaften, Abnutzungserscheinungen, etc.

Ist die zeitliche Auflösung des digitalen Systems fein genug (z.B. Aufgabe 1.2: Frequenz des Aufrufs von `loop()`), so wird das Prellen des Tasters – mehrere Spikes im Signalverlauf (siehe Abbildung 1) - als mehrfaches Betätigen interpretiert. Da Taster im Bereich eingebetteter Systeme relativ häufig zum Einsatz kommen, sollen Sie im ersten Teil dieses Aufgabenblattes eine funktionierende Lösung für das Problem entwickeln.

Das Entprellen eines mechanischen Tasters kann sowohl in Hardware als auch in Software realisiert werden. Ihre Aufgabe ist es eine Software-basierte Lösung zu entwickeln.

Eine sinnvolle Variante zur Lösung des Problems kann durch periodisches Abtasten des Eingangssignals realisiert werden. Dabei wird ein Hardware-Timer verwendet, welcher periodisch einen Interrupt auslöst. Die Aufgabe der für den Interrupt zuständigen Behandlungsroutine soll das Abtasten und Auswerten des Eingangssignals über einen von Ihnen definierten Zeitraum sein. Wird ein konstanter Signalpegel (in unserem Fall: Logikpegel `LOW` nach einer fallenden Flanke) betrachtet, so soll diese Information zur Verarbeitung des Ereignisses an das Hauptprogramm weitergegeben werden.

Definieren Sie ein Kriterium für einen betätigten bzw. entlasteten Taster und gestalten Sie Ihre Lösung so, dass das Erkennen einer vollständigen Betätigung (d.h. gedrückt + losgelassen) des Tasters gewährleistet wird. Diskutieren Sie, bei welchem Zustandsübergang Ihrer Entprell-Routine im Hauptprogramm die gewünschte Aktion angestoßen werden sollte.

Aufgabe 2.1

Um Ihnen den Einstieg zu erleichtern haben wir eine kleine Arduino Bibliothek vorbereitet, welche die Verwendung der Hardware-Timer vereinfacht. Die `DueTimer` Bibliothek können Sie von der Webseite der Veranstaltung herunterladen: DueTimer Bibliothek. Folgender Beispielcode skizziert die Verwendung der `DueTimer` Bibliothek:

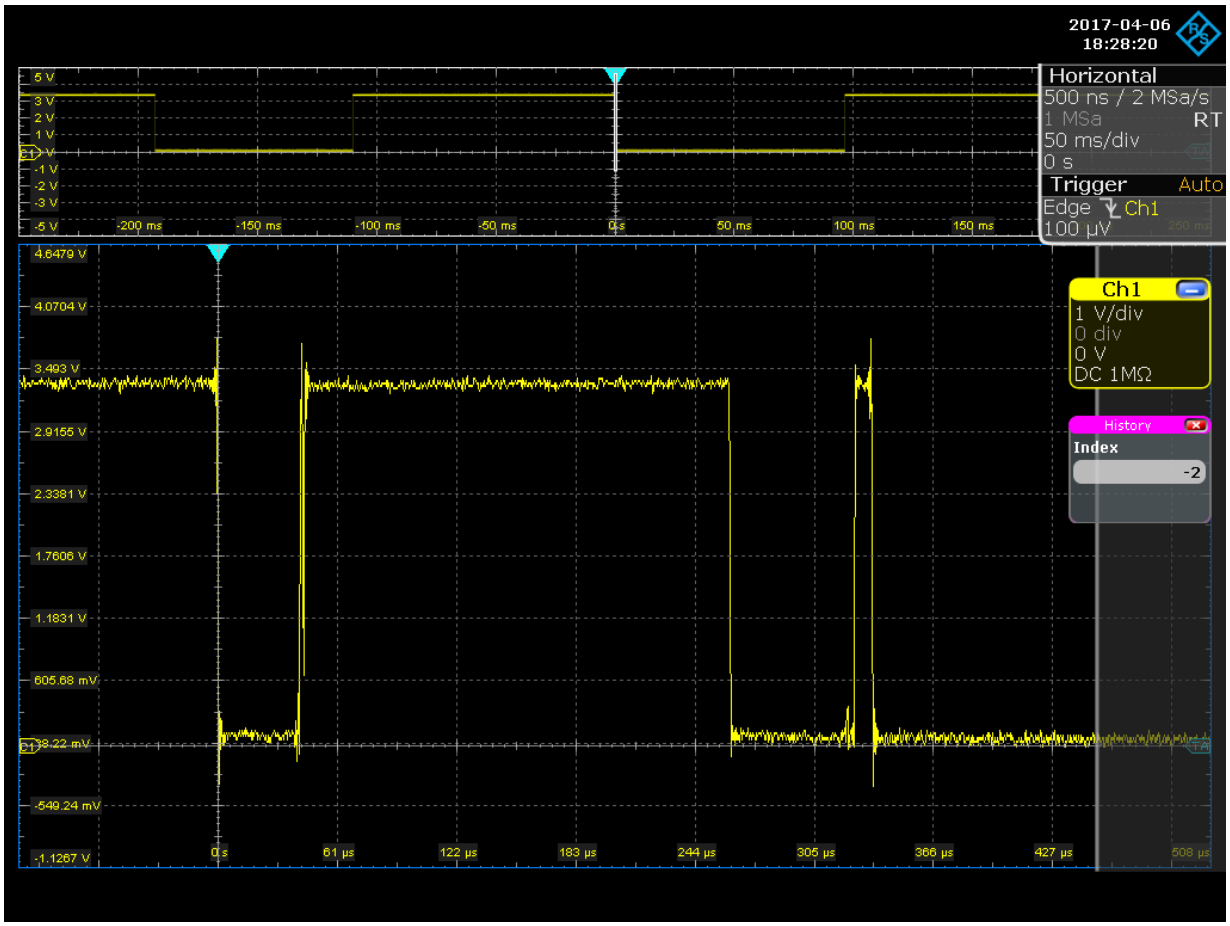

Abbildung 1: Signalverlauf / Prellen eines Tasters.

```
#include <DueTimer.h>

auto led_pin = 13;
volatile auto led_state = LOW;
DueTimer timer;

void changeLedState(void)
{
  // Wechsel des LED Zustands (ON/OFF)
  led_state = !led_state;
}

void setup()
{
  // Konfiguration der Richtung des digitalen I/O Anschlusspins 13
  pinMode(led_pin, OUTPUT);
```

```
  // Konfiguration des Hardware-Timers (@ 1 Hz)
  // Syntax: configure(timer_frequency, callback_function)
  if (timer.configure(1, changeLedState))
  {
    timer.start();
  }
}

void loop()
{
  // Wechsel des Logikpegels am Ausgangspin 13
  digitalWrite(led_pin, led_state);
}
```

Um das skizzierte Beispiel compilieren zu können, müssen Sie die `DueTimer` Bibliothek in den dafür von der Entwicklungsumgebung designierten Ordner verschieben. Dieser Ordner ist `~/Arduino/libraries`.

Betrachten Sie den Quellcode der `DueTimer` Bibliothek und parallel dazu das Datenblatt des auf dem Arduino Due verwendeten Mikrocontrollers: Atmel SAM3X8E ARM Cortex-M3 Datenblatt. Schauen Sie sich insbesondere das Kapitel 37 (ab Seite 869) an. Versuchen Sie den Quellcode der `DueTimer` Bibliothek zu verstehen, um diesen auf Anfrage erläutern zu können.

Wählen Sie ausgehend von der Konfiguration des Timers (empfohlen wird eine Frequenz von 1 KHz) einen sinnvollen Zeitraum für die Betrachtung des Signals. Vergessen Sie nicht beide Zustände des Betätigungsvorgangs zu betrachten (Taster gedrückt + Taster losgelassen).

Verwenden Sie den Versuchsaufbau aus dem ersten Aufgabenblatt und erstellen Sie ein Programm, das einen zuverlässigen Zustandswechsel der LED (Aufgabe 1.2 + 1.3) implementiert.

Aufgabe 2.2

Ein digitaler Anschlusspin, konfiguriert als Ausgang, lässt sich verwenden um eine angeschlossene Komponente entweder mit `GND` (0 V, Logikpegel `LOW`) oder mit `VCC` (3,3 V, Logikpegel `HIGH`) zu verbinden. Liefert eine Komponente (z.B. eine LED) bei 3,3 V Spannung 100 % ihrer Leistung, so besteht zunächst die Möglichkeit diese Komponente auf 0 % (`LOW`) oder 100 % (`HIGH`) Leistung einzustellen. Jedoch ist gerade bei LEDs die Möglichkeit zur variablen Regelung der Leistung (Leuchtintensität) sehr interessant.

Eine Art solcher variablen 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 2 stellt ein pulsweitenmoduliertes Signal mit einem positiven Tastverhältnis/Tastgrad von ca. 50 %, generiert mit dem Mikrocontroller des Arduino Due. Das Tastverhältnis stellt die Impulsdauer im Verhältnis zur Periodendauer dar. Ein positives 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`.

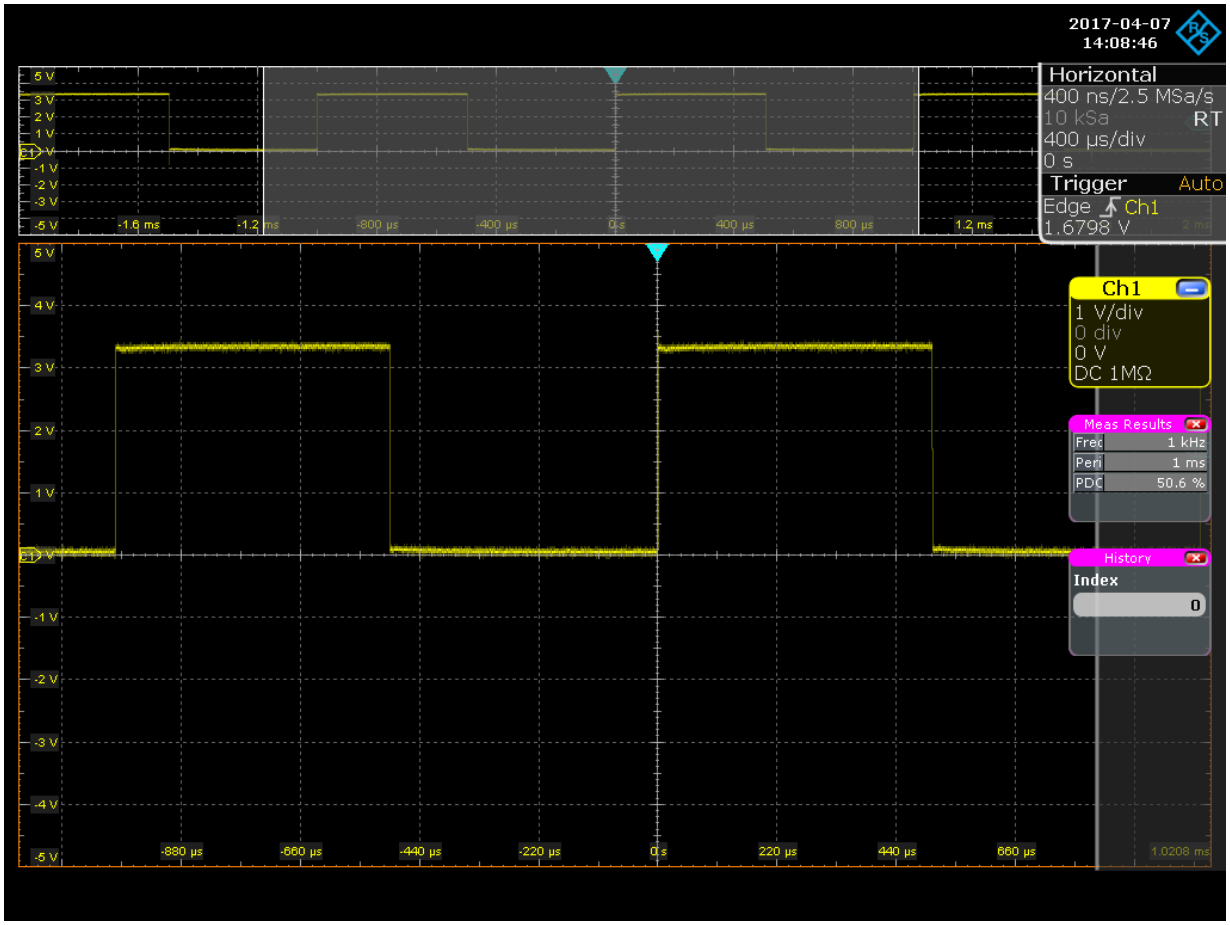

Abbildung 2: Pulsweitenmoduliertes Signal (@ 1 kHz).

Ziel der aktuellen Aufgabe ist es die Leuchtintensität der extern angeschlossenen LED mit einem **pulsweitenmodulierten Signal** variabel zu regeln.

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`

`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.

Erstellen Sie ein Programm, das die Leuchtintensität der LED mit einer Sägezahnfunktion regelt. Der Funktionsparameter ⟨*value*⟩ hat eine Auflösung von 8-bit, Ihnen steht damit ein Intervall von 0…255 zur Verfügung. Modifizieren/Erweitern Sie Ihre Lösung der Aufgabe 2.1 indem Sie **zwei entprellte Taster** in Ihr Programm einbinden. Die Betätigung der Taster soll folgendes Verhalten produzieren:

- Taster 1 dekrementiert **mit jeder separaten Betätigung** die Leuchtintensität der LED um eine von Ihnen festgelegte Schrittweite. An der unteren Grenze angekommen, soll die Leuchtintensität bei 0 % verbleiben.
- Taster 2 verhält sich entgegengesetzt zur Funktion von Taster 1: die Helligkeit wird Schrittweise erhöht und nach Erreichen von 100 % soll die LED bei ihrer max. Intensität verbleiben.