Aufgabenblatt 03 Termine: KW 18, KW 19

Gruppe	
Name(n)	Matrikelnummer(n)

3 Entprellen von Eingangssignalen

Bei der Bearbeitung der Aufgaben 1.3 und 1.4 des ersten Aufgabenblattes sollte Ihnen, falls Sie einen realen Aufbau mit echten Tastern zum Testen Ihrer Lösung benutzt haben, aufgefallen sein, dass eine einzige Betätigung des Tasters in vielen Fällen zu mehreren Zustandswechseln der LED geführt hat. Und auch in diesen Fällen sind Ihnen wiederum nur die Tasterbetätigungen mit einer ungeraden Anzahl an Zustandswechseln als Fehlfunktion aufgefallen.
Dieses Verhalten ist auf das **Prellen** (de.wikipedia.org/wiki/Prellen) des Tasters zurückzuführen. Dabei kommt es im Inneren des Tasters zu mehrfachem Kontaktschluss, bedingt durch mechanische Eigenschaften, Abnutzungserscheinungen, etc. Bei der Simulation mit Proteus werden Sie diese Effekte nicht bemerkt haben, da das Simulationsmodell des Tasters (nur) einen idealen Taster modelliert; diesen gibt es in der Realität leider nicht!

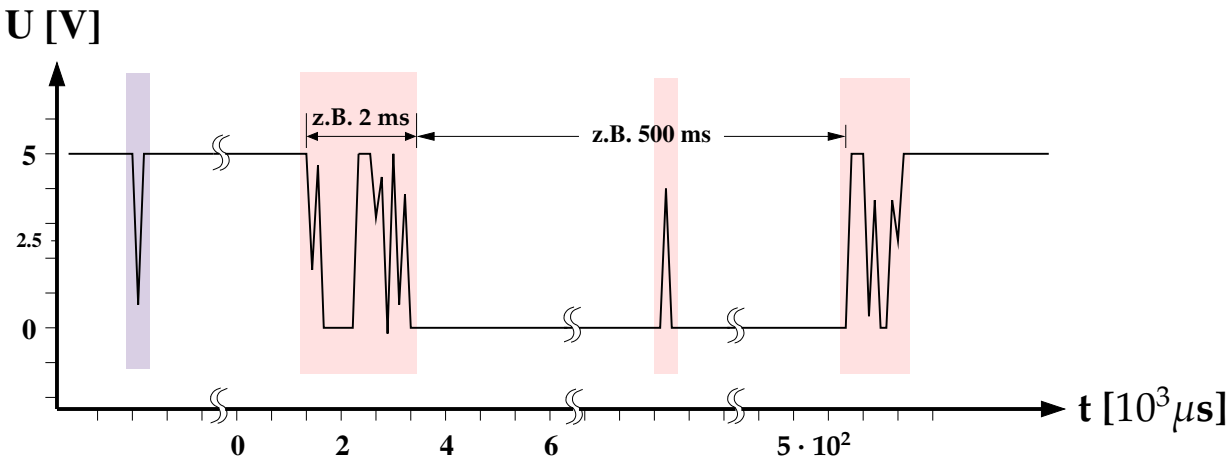

Abbildung 1: Signalverlauf / Prellen eines Tasters.

Abbildung 1 zeigt exemplarisch die Ausgabespannung eines realen Tasters, wenn dieser, wie in Aufgabe 1 eingeführt, mit einem Pullup-Widerstand gegen V_{CC} verschaltet ist. Die pink hinter-

legten Bereiche werden durch das nicht ideale Verhalten des Tasters bzw. des Kontaktmaterials im Übergangsbereich und auch z. T. im bereits kontaktierten Zustand hervorgerufen. Der distelfarben hinterlegte Spike ist hingegen eher auf eine unsaubere Versorgungsspannung zurückzuführen. Aber letztendlich darf keine dieser kurzzeitigen Änderungen des Spannungspegels als Betätigung des Tasters fehlinterpretiert werden.
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 auch Abbildung 2) - als mehrfaches Betätigen interpretiert. Da Taster im Bereich eingebetteter Systeme relativ häufig zum Einsatz kommen, sollen Sie im ersten Teil dieses Aufgabenblattes eine Lösung für das Problem entwickeln.

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

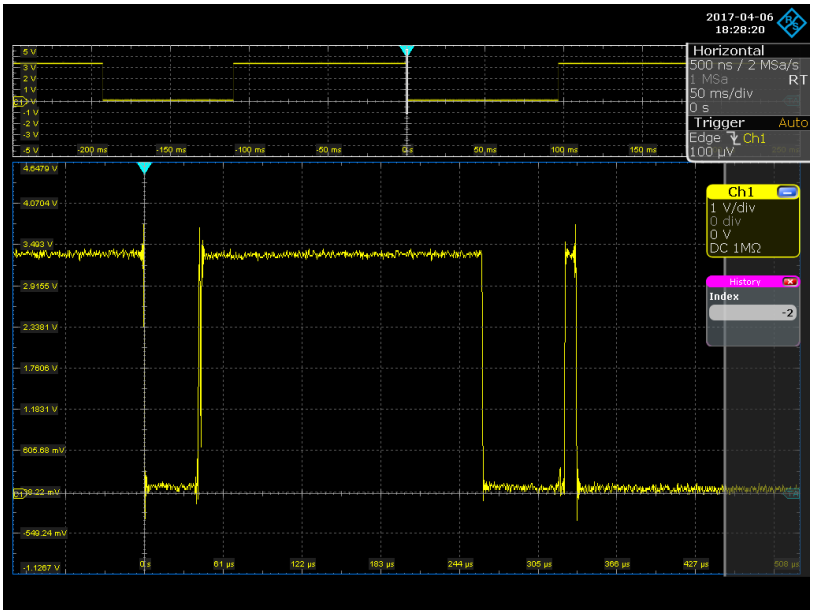

Abbildung 2: Oszillogramm des Signalverlauf am realen Taster. Der Taster prellt hier über ca. 350 μs, was bereits recht kurz ist. Je nach Bauart und Größe des Tasters/Schalters ist mit Prellzeiten bis in den zweistelligen Millisekunden Bereich zu rechnen.
Sollten Sie beispielsweise in Ihrer Lösung zu Augabe 1.4 lediglich die fallende Flanke zur Erkennung der Tasterbetätigung eingesetzt haben, werden bei obigem Signal mindestens 3 Tasterbetätigungen detektiert, wobei es sich tatsächlich um eine einzelne Betätigung handelt.

Aufgabe 3.1 Kriterium zur Erkennung des Tasterzustandes

Definieren Sie ein Kriterium für die sichere Erkennung des betätigten und des entlasteten Tasters. Hierfür wird die die kontinuierliche bzw. die quasi kontinuierliche Beobachtung des Eingangssignals erforderlich werden. Die Überwachung Ihrer Kriterien wird die Erkennung eines

vollständigen Betätigungszyklus (gedrückt + losgelassen) des Tasters ermöglichen.
Diskutieren und begründen Sie, bei welchem Zustandsübergang Ihrer Entprell-Routine im Hauptprogramm (der Main-Loop) die gewünschte Aktion angestoßen werden sollte.

Bei diesem Lösungsansatz wird es erforderlich werden, das Eingangssignal in für die Entprellung relevanten Zeitbereichen (quasi)kontinuierlich zu beobachten. Dieses kann durch periodisches, äquidistantes Abtasten des Eingangssignals realisiert werden. Dafür lassen sich Hardware-Timer verwenden, welche periodisch einen Interrupt auslösen. Die Aufgabe der dem entsprechenden Interrupt zugeordneten Behandlungsroutine wird dann das Abtasten des Eingangssignals sein.
Sollten Sie sich beispielsweise entschieden haben, die gewünschte Aktion bereits bei Betätigung des Tasters und nicht erst am Ende des Betätigungszyklus auszulösen, so soll nach Erkennung eines konstanten Signalpegels (in diesem Fall: Logikpegel `LOW` nach einer fallenden Flanke über den entsprechend Ihres Kriteriums definierten Zeitraum) diese Information zur Verarbeitung des Ereignisses an das Hauptprogramm weitergegeben werden.

3.1 Hardware-Timer

Die folgenden Aufgaben benötigen zur Realisierung Hardware-Timer, die von allen auf den Arduino-Boards eingesetzten Prozessoren in unterschiedlicher Zahl zur Verfügung gestellt werden.
Timer/Counter sind spezielle Hardwareeinheiten der Prozessoren, die, getrieben durch die Systemclock, deren Frequenz ggf. heruntergeteilt werden kann, ein dem jeweiligen Timer fest zugeordnetes Zähl-Register inkrementieren. Die Timer können über Zählfrequenz, Startwert, Vergleichswert und verschiedene Interruptbedingungen flexibel konfiguriert werden. Die auf dem Atmel ATmega328P basierenden Arduino-Boards wie der Arduino UNO besitzen drei Timer (timer0,1,2), die Arduino MEAG-Boards, die auf den ATmega1280 oder ATmega2560 basieren, verfügen über sechs Timer, und das Arduino DUE-Board mit dem Atmel SAM3X8E ARM-Prozessor über neun Timer.
Allerdings sind nicht alle Timer beliebig verfügbar. Bei den AVR-Prozessoren wird:

- **Timer0** (8bit Timer) für Arduino-Zeitfunktionen wie `delay(), millis() und micros()` verwendet,
- **Timer1** (16bit Timer) auf dem UNO für die Servo-Library verwendet,
- **Timer2** (8bit Timer) von der Arduino `tone()`-Funktion benutzt,
- **Timer3, Timer4, Timer5** (16bit, nur MEGA-Boards) sind frei verfügbar; bei Verwendung der Servo-Library wird hier Timer5 belegt.

Darüber hinaus werden bei Einsatz der PWM-Pins noch weitere Timer für die Erzeugung der PWM-Signale belegt. Beim ATmega2560 beispielsweise wird für die Generierung der PWM-Signale der Timer0 für den Pin 4, der Timer1 für die Pins 11, 12, 13, der Timer2 für die Pins 9, 10, der Timer3 für die Pins 2, 3, 5, der Timer4 für die Pins 6, 7, 8 und der Timer5 für die Pins 44, 45 und 46 werwendet. Bei Verwendung eines Timers ist also immer zu prüfen, ob dieser nicht bereits durch eine der verwendeten Funktionen belegt ist.

Um Ihnen den Einstieg zu erleichtern, haben wir die erforderlichen Libraries für die Verwendung der Timer bereitgestellt, welche die Verwendung der Hardware-Timer des

Arduino-MEGA vereinfacht. Die sechs Timer des auf dem MEGA2560-Board eingesetzten Prozessor ATmega2560 ermöglicht einen weitgehend flexiblen Einsatz der Timer. Die MEGA-Timer Bibliotheken können Sie von der Webseite der Veranstaltung herunterladen: MegaTimer Bibliotheken Die Timer1- und die Timer3-Library des Tar-Archives finden Sie auch unter playground.arduino.cc. Die TimerFour- und TimerFive-Library ist von der TimerThree-Library abgeleitet.

Folgender Beispielcode skizziert die Verwendung der `Mega-Timer` Bibliotheken:

```
#include <TimerThree.h>

static const uint8_t led_pin = 13;  // the pin with a LED
volatile bool led_state = LOW;

void changeLedState(void){
  led_state = !led_state;               //change of led state
 }

void setup(void)
{
  //configuration of digital I/O pin #13
  pinMode(led_pin, OUTPUT);
  //configuration of timer3 (@ 1Hz, 1E6us)
  Timer3.initialize(1E6);
  //attatch ISR
  Timer3.attachInterrupt(changeLedState); // attatch ISR & start timer
  //Timer3.stop(); Timer3.start()
}

void loop(void)
{
  digitalWrite(led_pin, led_state);
  delay(10);
}
```

Um das skizzierte Beispiel compilieren zu können, müssen Sie die `MegaTimer` Bibliotheken in den dafür von der Entwicklungsumgebung vorgegebenen Ordner verschieben. Dieser Ordner befindet sich im Home-Verzeichnis des Benutzers.
Der Pfad des Ordners ist unter Linux: `~/Arduino/libraries` und unter Win10/11: `~\Documents\Arduino\libraries`.
Betrachten Sie den Quellcode der `MegaTimer` Bibliotheken und parallel dazu das Datenblatt des auf dem Arduino MEGA-Board verwendeten Mikrocontrollers: Atmel ATmega2560. Schauen Sie sich insbesondere das Kapitel 17 (ab Seite 133) an. Versuchen Sie den Quellcode der `Timer-Bibliotheken` zu verstehen, um diesen auf Anfrage erläutern zu können.

Aufgabe 3.2 Entprellen eines Tasters

Erweitern Sie den Versuchsaufbau aus dem ersten Aufgabenblatt und erstellen Sie ein Programm, das einen zuverlässigen Zustandswechsel der LED in Abhängigkeit der Tasterbetätigung (Aufgabe 1.3 + 1.4) realisiert. Implementieren einen Zähler erkannter Betätigungen des Tasters.
Wählen Sie ausgehend von der Konfiguration des Timers (empfohlen wird eine Frequenz von 1 KHz) einen sinnvollen Zeitraum für die Beobachtung des Signals in den relevanten Zeiträumen. Vergessen Sie nicht, beide Zustandsübergänge des Betätigungsvorgangs für einen vollen Betätigungszyklus zu betrachten (Taster gedrückt + Taster losgelassen).
Unter Berücksichtigung des Nyquist-Shannon-Abtasttheorems müsste das Signal aus Abb. 2 selbst bei Außerachtlassen der Unsauberkeiten im Bereich der Flanken allein wegen des Spikes bei ~330μs mit mindestens 100 KHz abgetastet werden. Weshalb erreichen wir selbst ohne vorgeschaltete Entprellung durch zusätzliche Hardware selbst bei einer Abtastrate von 1 kHz eine wirkungsvolle Entprellung? Da die Taster der Proteus Bibliothek ideale Taster modellieren, eignen sich diese auch nicht zum Test Ihrer Entprellroutine. Der in unserer Lizenz zur Verfügung stehende Patterngenerator erlaubt leider keine algorithmischen Muster, sondern nur kurze hardkodierte Musterfolgen. Verwenden Sie deshalb bitte zum Testen ihres Entprellverfahrens

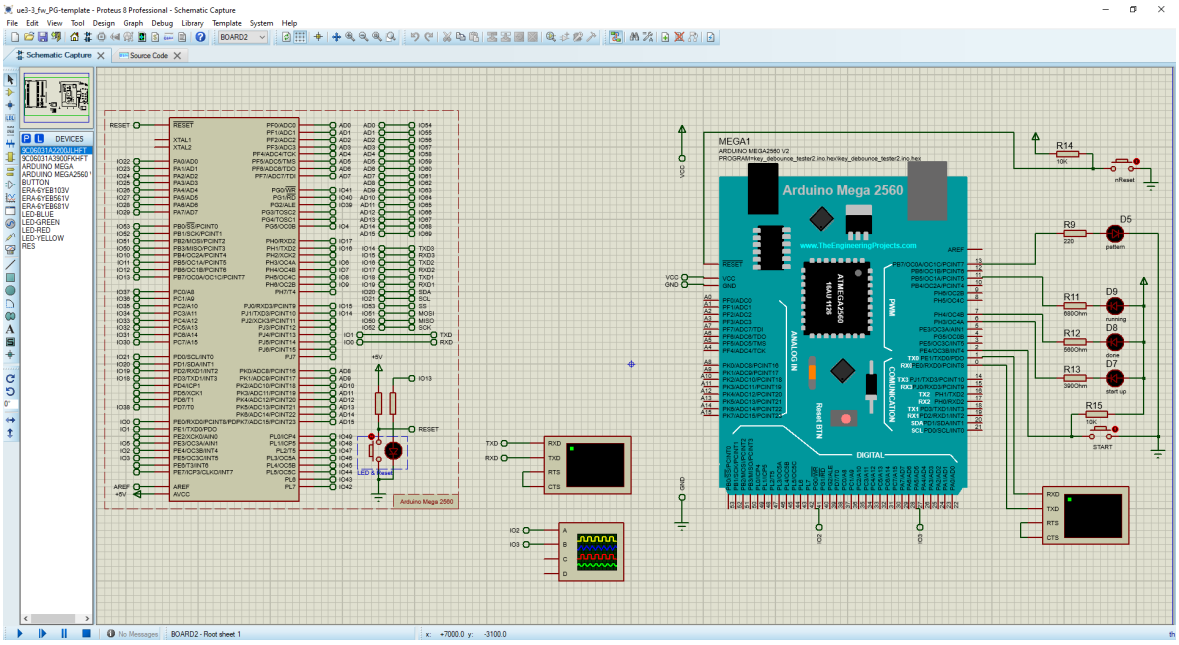

Abbildung 3: Test bench zur Validierung des Entprellverfahrens bestehend aus zwei Prozessoren: (rechts) – Patterngenerator; (links) – Device under Test (Prozessor zur Implementierung des zu testenden Entprellverfahrens

das vorbereitete Proteus-Projekt ue3-3_fw_PG-template.pdsprj (siehe Abb. 3). Laden Sie die vorbereitete Firmware (key_debounce_tester2.ino.hex in Prozessor des Patterngenerators hoch. Der linke Prozessor steht Ihnen für ein Firmware-Projekt zur Implementierung Ihrer Entprellroutine zur Verfügung. Vervollständigen Sie das Scematic, auch ggf. noch mit zusätzlichen Tastern, um das anfängliche Testen zu erleichtern.

Aufgabe 3.3 gleichzeitige Tasterbetätigung

Um Resourcen zu sparen, werden die Benutzerinterface eingebettete Systeme meist recht einfach gehalten, was sich auch in einer ökonomischen Gestaltung der Eingabemöglichkeiten widerspiegelt. Beispielsweise lässt sich ein dritter Taster einsparen, wenn die „geichzeitige“ Betätigung zweier bereits vorhandener Taster ausgewertet werden kann.

(a) Erweitern Sie Ihr Programm aus der vorigen Aufgabe um die Entprellung eines zweiten Tasters. Beide Taster sollen, wie bereits diskutiert, einzeln ausgelöst werden können.

(b) Zusätzlich soll auch die Möglichkeit geschaffen werden, eine Doppel- oder Parallelbetätigung beider Taster auszuwerten. Definieren Sie hierfür ein Kriterium, das erlaubt, sowohl eine parallele Betätigung der Taster als auch Einzelbetätigungen zu erkennen bzw. zu unterscheiden.

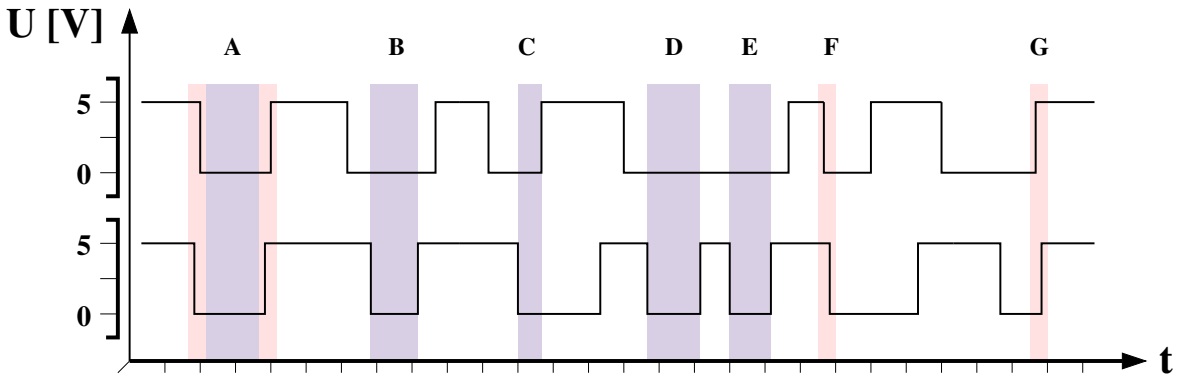

Abbildung 4: Beispielszenarien für Doppel- bzw. Einzelbetätigungen

Hier müssen Sie Fragen diskutieren wie: „Wie definiert sich eine Doppelbetätigung: Beginn in einem festgelegten Zeitfenster, und/oder Ende im festgelegten Zeitfenster, oder eine einfache Überlappung usw.“. Abbildung 4 zeigt eine kleine Auswahl an Szenarien, die an Hand Ihres Kriteriums eindeutig als Doppel oder Einzelbetätigung klassifiziert werden müssen. Eine Tasterbetätigung darf natürlich auch nur einmal ausgewertet werden: Entweder ist die Betätigung eines Tasters Bestandteil einer Doppelbetätigung (hier stellen die Fälle **D** und **E** ggf. einen Sonderfall dar), oder es ist eine Einzelbetätigung.

(c) Implementieren Sie Ihr Kriterium zur Erkennung einer Doppelbetätigung und zählen Sie die Doppelbetätigungen. .