Aufgabenblatt 03 Termine: KW 17, KW 18

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!

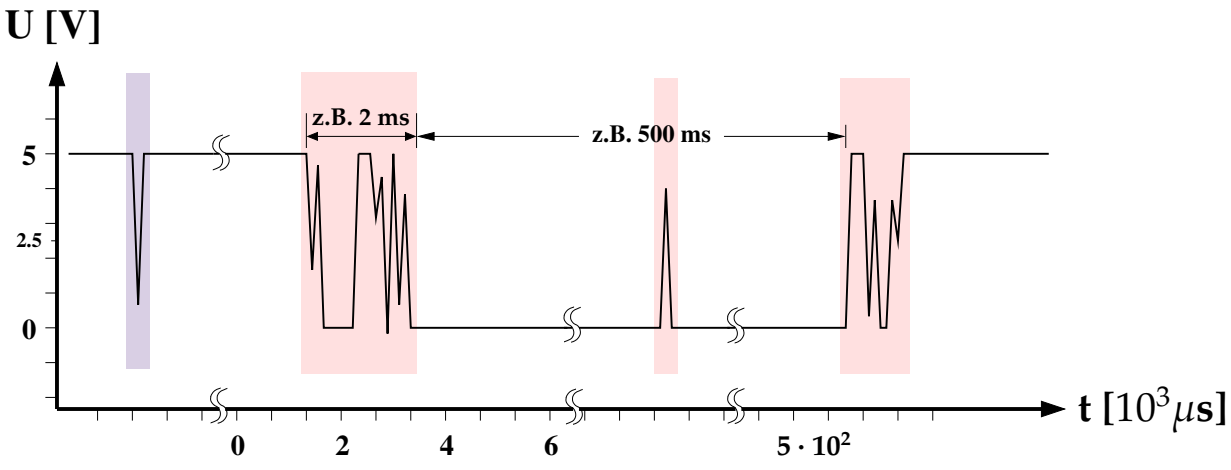

Abbildung 1: Signalverlauf / Prellen eines Tasters.

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 bei eingebetteten Systemen ein beliebtes Eingabegerät darstellen, 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.

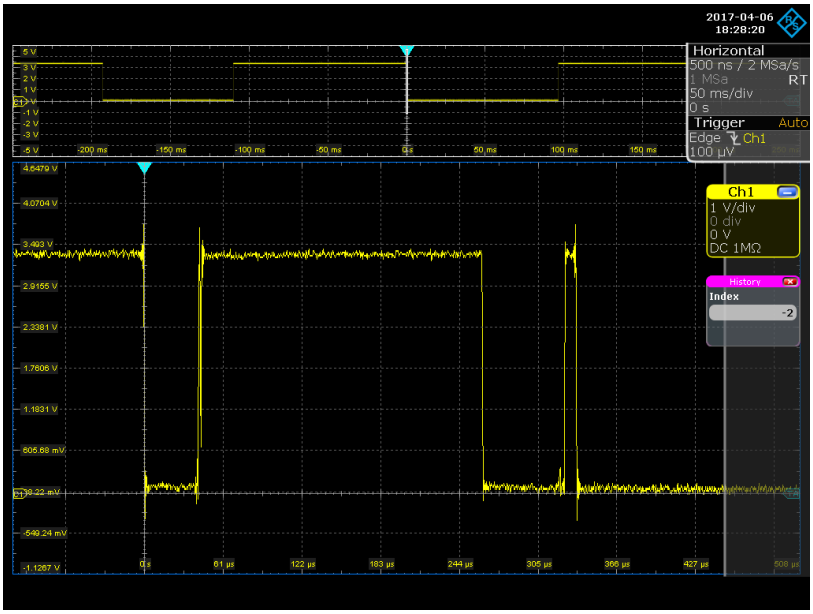

Abbildung 2: Oszillogramm des Signalverlauf am realen Taster. Der Taster prellt hier über ca. 350 μs, was bereits recht kurz ist. Je nach Bauart und Größe des Tasters/Schalters ist mit Prellzeiten bis in den zweistelligen Millisekunden Bereich zu rechnen.
Sollten Sie beispielsweise in Ihrer Lösung zu 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 Ermitteln der Dauer einer typischen Tasterbetätigung

Ermittlen Sie, wie lange Sie bei der Betätigung eines Tasters diesen typischerweise gedrückt halten.
Hierfür wird es erforderlich werden, das Eingangssignal (quasi)kontinuierlich zu beobachten. Dieses kann durch periodisches, äquidistantes Abtasten des Signals 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.
Beachten Sie hierfür die im folgenden Kapitel: **3.1 Hardware-Timer** gegebenen Hinweise.

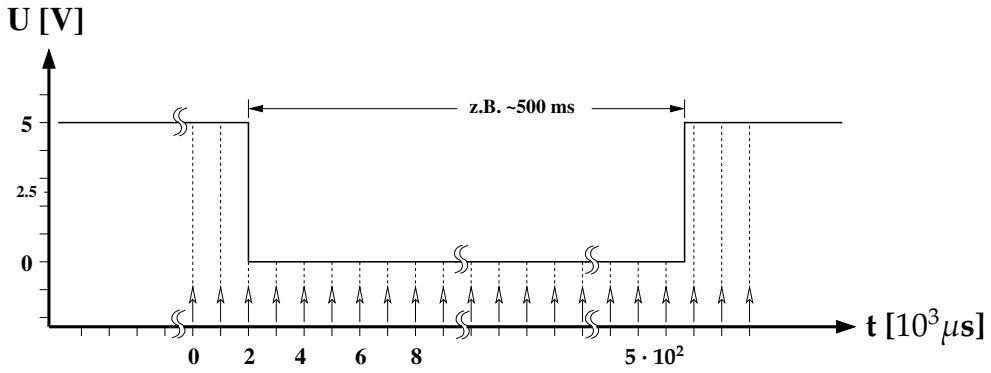

Abbildung 3: Abtasten eines Signals im Abstand 1 mS

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.
Um den Sketch für beide Zielarchitekturen, also das Arduino 2560-Board für die Proteus-Simulation und das Arduino DUE-Board für die Hardwarerealisierung, portabel zu halten, sind einige Besonderheiten der beiden Architekturen zu beachten:

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** (16bit, nur MEGA-Boards) sind frei verfügbar;
- **Timer5** (16bit, nur MEGA-Boards) frei verfügbar; aber die Servo-Library der MEGA-Boards benötigt diesen Timer.

Darüber hinaus werden bei Einsatz der PWM-Pins der AVR-Controller noch weitere Timer für die Erzeugung der PWM-Signale belegt. Beim ATmega2560 werden beispielsweise für die Generierung der PWM-Signale die Timer0,1,2,3,4 und 5 mit folgender Zuordnung eingesetzt:

Timer	PWM-Pin	Timer	PWM-Pin	Timer	PWM-Pin
Timer0	4	Timer2	9, 10	Timer4	6, 7, 8
Timer1	11, 12, 13	Timer3	2, 3, 5	Timer5	44, 45, 46

Beim Arduino DUE hingegen existieren weniger Einschränkungen bezüglich der Timer. Hier werden bei Verwendung der Servo-Library zwar die Timer 0,2,3,4 und 5 belegt, während bei Verwendung der PWM-Pins keine weiteren Einschränkungen auftreten sollten, da hier eigene Counter vorhanden sind. Allerdings interferieren auch hier auf Digitalpins herausgeführte Steuersignale der Timer/Counter mit der Generierung der PWM-Signale. Vollständig frei von Kreuzbezügen sind auch hier nur die PWM-Pins 6,7,8,9
Bei Verwendung eines Timers ist also (architekturübergreifend) grundsätzlich immer zu prüfen, ob dieser nicht bereits durch eine der verwendeten Funktionen belegt ist.

Um Ihnen den Einstieg zu erleichtern, werden die erforderlichen Libraries für die Verwendung der Timer bereitgestellt, welche die Verwendung der Hardware-Timer des Arduino-DUE als auch des Arduino-Mega vereinfacht.

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

- Die neun Timer des auf dem Arduino DUE-Board verwendeten Prozessors (Atmel SAM3X8E ARM Cortex-M3) ermöglichen einen weitgehend flexiblen Einsatz der Timer (s.o). Die `DueTimer` Bibliothek können Sie von der Webseite der Veranstaltung herunterladen: DueTimer Bibliothek.

Der folgende Beispielcode skizziert die Verwendung der Timer-Libraries, um den Code sowohl für den Mega 2560 als auch den Arm SAM3 des DUE kompillierbar zu halten.

```
// this sketch compiles with MEGA2560 as well as with DUE
//select timer library depending on the target architecture
#if defined(__SAM3X8E__)
  #include <DueTimer.h>
  // Objects for configuration of Arduino Due's hardware timers
  DueTimer Timer4;     //object named Timer4 to match Timer4 of MEGAs library
#else
  #include <TimerFour.h>
#endif

const uint32_t t4_frequency = 1;                  //set timer frequency [Hz]
const long t4_period = (long)(1e6/t4_frequency);  //calculate timer period [µs]

void timerISR(void) {
 //code of ISR
}

void setup() {
  // configuration of hardware timer
  #if defined(__SAM3X8E__)
    if (Timer4.configure(t4_frequency, timerISR)) {
      // Syntax: configure(timer_frequency , callback_function)
      // DUE-Timer is configured, but has not been started
      ; }
    else  { // timer configuration fails...
      ; } //if possible prepare an error message
  #else
   Timer4.initialize(t4_period);
   Timer4.attachInterrupt(timerISR);
   //attachInterrupt starts Timer, if not intended here, stopp timer!
   Timer4.stop();
  #endif

  Timer4.start();    //normally done on demand by application
}  //end setup

void loop() {
  //code of main loop
}  //end loop
```

Grundsätzlich sollten Sie folgendes beachten, um den Code für beide Zielarchitekturen – trotz zum Teil unterschiedlicher Libraries – problemlos kompilieren zu können:

- Verwenden Sie keine architekturspezifischen Zusatzfunktionen, wie beispielsweise die Enhanced SPI-Library des Due.
- Binden Sie architekturspezifische Librarys mittels Präprozessordirektiven ein.
  Versuchen Sie möglichst eine architekturübergreifende Notation zu verwenden. Benennen Sie beispielsweise das Due-Timerobjekt entsprechend der Mega-Notation. Auf diese

Weise sind die Methodenaufrufe im Code identisch.
Der Beispielsketch DueMegaTimerTemplate.ino, bzw. das Proteusprojekt DueMegaTimerTemplate.pdsprj zeigen es exemplarisch.

Auf diese Weise kann der Code weiterhin unter Proteus für den ATmega2560 entwickelt und validiert werden und anschließen auf dem Arduino DUE ausgeführt werden, wobei über die für das Kompilieren gewählte Zielarchitektur die Codeauswahl gesteuert wird.

Um das skizzierte Beispiel compilieren zu können, müssen Sie die MegaTimer Bibliotheken und die DueTimer Bibliothek 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 Timer Bibliotheken und parallel dazu das Datenblatt des auf dem Arduino-Board verwendeten Mikrocontrollers: Im Falles des Arduino DUE, das des Atmel SAM3X8E ARM Cortex-M3, Kapitel 36 (ab Seite 856), und im Falle des Arduino MEGA 2560, das des Atmel ATmega2560, Kapitel 17 (ab Seite 133). Versuchen Sie den Quellcode der Timer-Bibliotheken zu verstehen.

Aufgabe 3.2 Kriterium zur Erkennung des Tasterzustandes

Für die Entprellung eines digitalen Eingangssignals ist es erforderlich, ein leicht überprüfbares Kriterium für die sichere Erkennung des LOW-Pegels und des HIGH-Pegels vorliegen zu haben. Definieren Sie für die Entprellung eines Tasters ein Kriterium für die sichere Erkennung des betätigten und auch 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 erfordern.
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, wie es bereits in Aufg. 3.1 für die Ermittlung der Betätigungsdauer des Tasters eingesetzt wurde.

Diskutieren und begründen Sie, bei welchem Zustandsübergang des Tasters im Hauptprogramm (der Main-Loop) die bei Tasterbetätigung gewünschte Aktion angestoßen werden sollte. Grundsätzlich ist es hier möglich, die bei Betätigung des Tasters anzustoßende Aktion bereits bei erkannter sicherer Betätigung, oder aber erst nach sicherer Deaktivierung des Tasters anzustoßen. Diskutieren Sie die Vor- und Nachteile beider Varianten.
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: konstanter Logikpegel `LOW` nach einer fallenden Flanke über den entsprechend Ihres Kriteriums definierten Zeitraum) diese Information zur Verarbeitung des Ereignisses an das Hauptprogramm weitergegeben werden, wobei natürlich der Taster bis zum Abschluss der Betätigung überwacht werden muss.

Aufgabe 3.3 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.

Wie bereits unter Aufg. 3.2 diskutiert, ist die Überwachung eines vollständigen Betätigungszyklus (gedrückt + losgelassen) des Tasters erforderlich. Implementieren Sie hierfür einen endlichen Automaten und beginnen Sie die Entwicklung des Automaten mit dem Aufstellen des Zustandsdiagramms. Bei der Entscheidung, ob Sie einen Moore- oder Mealy-Automaten entwerfen, bedenken Sie, dass bei der unter Aufgabe 3.4 geforderten gleichzeitigen Betätigung zweier Taster die Automaten beider Taster untereinander kommunizieren müssen. Die Zustandsüberführungsfunktion des Automaten lässt sich sehr übersichtlich mit Hilfe des Arduino-Switch Statements formulieren:

```
switch (state) {      //Zustandsvariable: state_t state;
  case idle:          //              enum state_t {idle, active, pressed, ...};
    state = ...;
    ...;
    break;
  case active:
    state = ...;
    ...;
    break;
  case pressed:
    state = ...;
    ...;
    break;
  case ...:
  ...;
  default:
    state = ...;
    break;
  }
```

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 das vorbereitete Proteus-Projekt ue3-3_fw_PG-template.pdsprj (siehe Abb. 4). 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. Die drei Taster *button0, button1* und *double actuation* sollen das anfängliche Testen ihrer Entprellroutine erleichtern. Bedenken Sie hierbei aber, dass es sich im ideale Taster handelt.

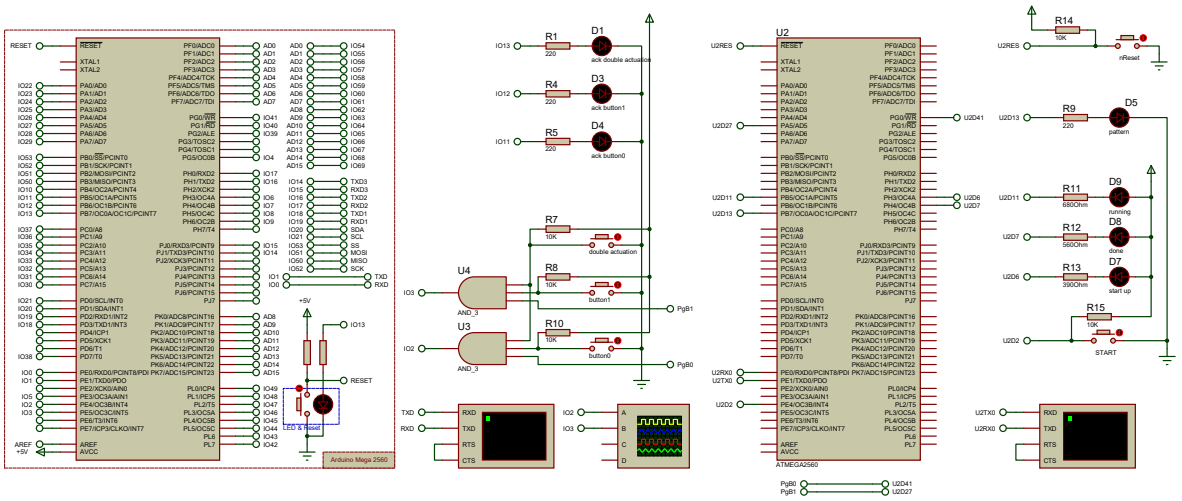

Abbildung 4: Test bench zur Validierung des Entprellverfahrens bestehend aus zwei Prozessoren: (rechts) – Patterngenerator; (links) – Device under Test (Prozessor zur Implementierung des zu testenden Entprellverfahrens

Aufgabe 3.4 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.

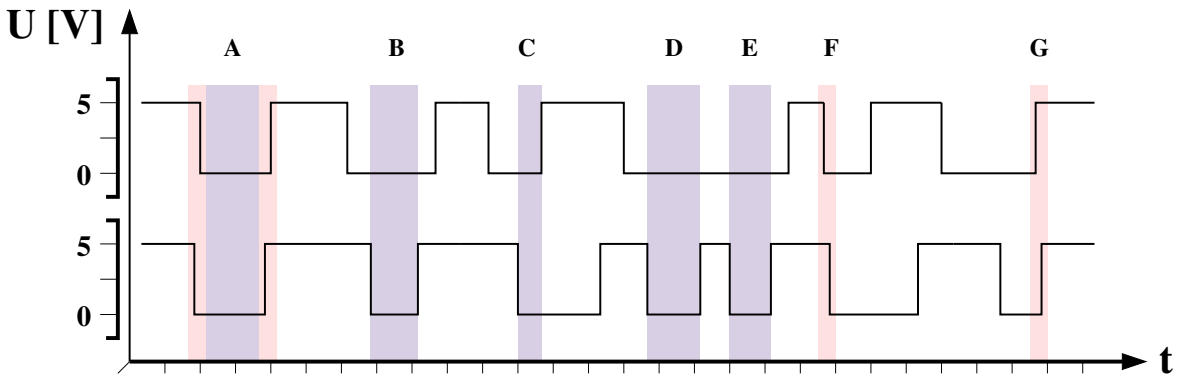

Abbildung 5: Beispielszenarien für Doppel- bzw. Einzelbetätigungen

Hier müssen Sie Fragen diskutieren wie: „Wie definiert sich eine Doppelbetätigung: Be-

ginn in einem festgelegten Zeitfenster, und/oder Ende im festgelegten Zeitfenster, oder eine einfache Überlappung usw.“. Abbildung 5 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. .