**Dokumentation der Aufgaben 2.x**

2.3:

Aus bereits vorhandenen Erfahrungen aus der Praxisphase 1 wissen wir bereits, wie ein Arduino-Programm aufgebaut ist.

Auf der Konsole wird gemäß unserem Code alle 500ms der String „Hello SSTS Arduino

World“ auf dem Serial Monitor ausgegeben.

Das hochladen eines Programms lässt sich an der Anzeige „Uploading Sketch“ und der anschließenden erfolgs-Meldung „Done uploading“ in der unteren rechten Ecke der Arudino IDE erkennen. Während des Uploads blinken auf dem Arduino außerdem die RX und TX LEDs, welche bei erfolgtem Upload wieder aufhören zu blinken.

2.4:

Um eine LED mit 1 Hz blinken zu lassen, ist es erforderlich, deren Zustand im 500msec Takt zu alternieren.

Dazu setzen wir anfangs den Zustand der LED auf eine Variable mit frei gewählten Wahrheitswert, in unserem Fall True. In der loop() Funktion wird der Wert der Zustands-Variablen auf den entsprechendem Pin ausgegeben und invertiert.

Um ein sofortiges alternieren zu vermeiden endet die loop() Funktion mit einer delay(500) Anweisung, damit zwischen zwei Alternierungen gewartet wird.

2.5: ­

Um den Zustand eines Tasters und eines einzelnen Dip Switches auszulesen und auszugeben müssen deren Werte in jeder Iteration der loop() Funktion neu ausgelesen werden.

Wir haben uns für den Taster auf Pin 10 und den Dip Switch auf Pin 3 entschieden.

Ist der Zustand beim Auslesen gleich HIGH, so wird der Zustandsvariablen der Wert 1 zugewiesen, andernfalls 0. Anschließend wird die entsprechende Ausgabe auf die Konsole geprintet.

2.6:

Der Zustand der LED soll nur dann alterniert werden, wenn der Taster gedrückt ist. Um ein zu schnelles blinken zu vermeiden muss eine Alternative zur delay() Funktion gefunden werden.

Hierzu definieren wir eine Zeitspanne, um die gewartet werden soll, und initialisieren eine Anfangszeit seit dem Programmstart mit 0.

Sofern die Differenz aus dem Rückgabewert der millis() Funktion (der aktuellen Zeit seit dem Programmstart in ms) und dem Zeitpunkt der letzten Ausführung größer/gleich dem Intervall ist, wird der Zustand tatsächlich alterniert. Die Alternierung erfolgt dabei analog zu 2.4.

Ist der Taster nicht gedrückt, so wird einfach nur LOW auf den Pin ausgegeben.

2.7:

Es sollen ein oder zwei LEDs von links nach rechts oder umgekehrt wandern, wobei die wandernden LEDs entweder an oder ausgeschaltet sein sollen. Über verschiedene Taster oder Dip switch Betätigungen soll der Benutzer dies auswählen können.

Um ein sofortiges Umschalten der verschiedenen Modi zu erreichen verwenden wir die Funktion instant\_input\_reaction(), welche in jeder loop() Iteration aufgerufen wird und Variablen die die Zustände Programintern speichern sofort entsprechend umschaltet.

Beim Bewegen der LED wird abhängig vom aktuellen Status der Einstellung ein oder zwei LEDs in laufrichtung angeschaltet oder ausgeschaltet.

Beim Berechnen der Position wird die aktuelle Position immer eins in aktueller Laufrichtung erhöht. Die Berechnung erfolgt dabei anhand der Hilfsvariablen i, welche dynamisch zurückgesetzt wird. Handelt es sich um den Fall, dass die LED an einem der Ränder angesprochen werden sollte und mehrere LEDs aktiv sind, so wird dies gesondert behandelt.

2.8:

Es soll anhand eines DIP-Switches ein Wert zwischen 0 und 255 eingestellt werden, und anhand dieses Wertes eine LED über PWM gesteuert werden.

Dazu wird anfangs ein Array mit den Pin-Nummern der einzelnen Dip Switches initialisiert sowie der LED-Pin. Die Konfiguration als In-/Output wird entsprechend durchgeführt. Wichtig ist, dass die LED an einem PWM fähigen Ausgang angeschlossen ist.

In der loop() Funktion muss in jeder Iteration der aktuelle Wert mit 0 initialisiert werden. Anschließend wird jeder DIP-Switch als Binärwert ausgelesen, um die Wertigkeit nach links geshiftet und mit dem berechneten Wert oder-verknüpft.

Eine Änderung der Auflösung der PWM-Ausgänge von 8 Bit auf 16 Bit ist zwar theoretisch möglich, aber in der Praxis nicht einfach umsetzbar.