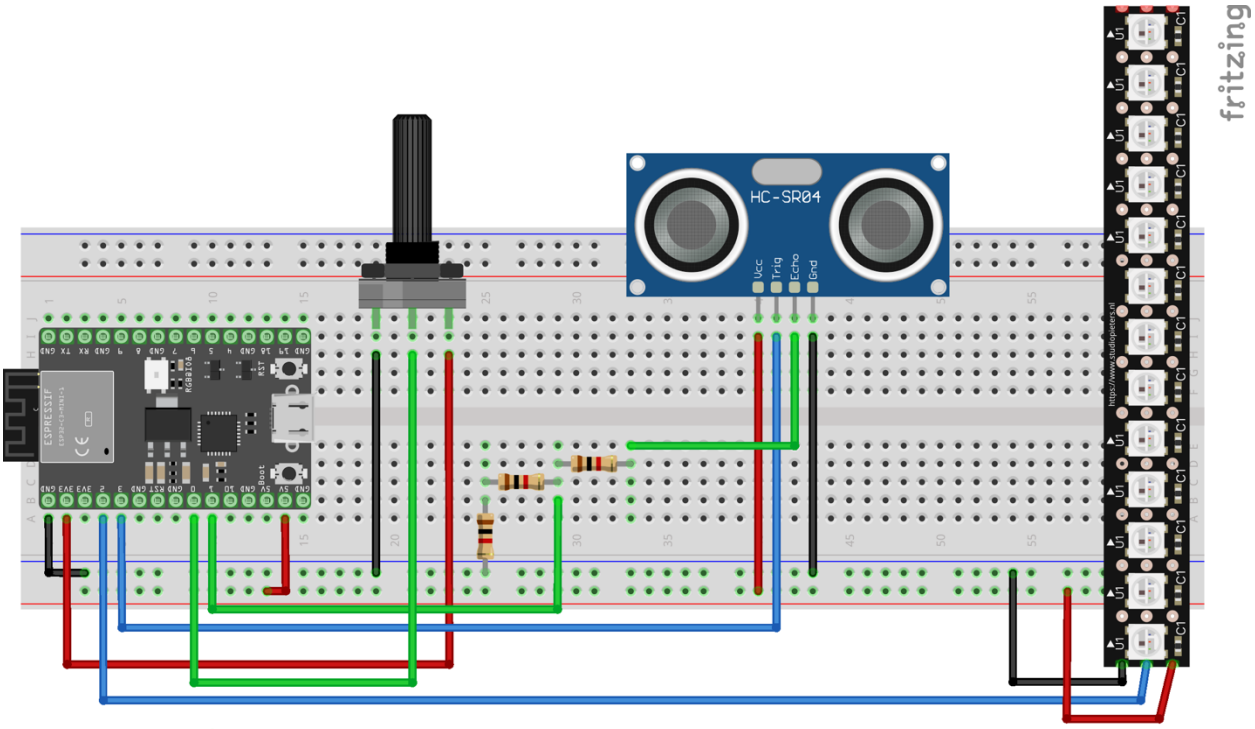
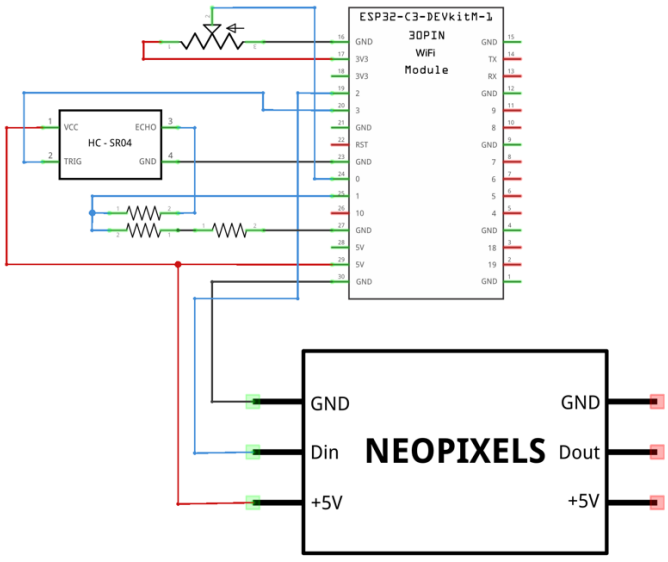
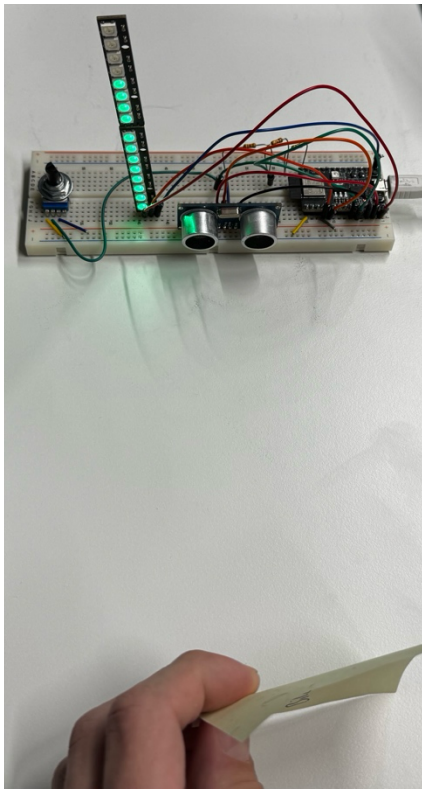
Systemnahe Programmierung 2

Matr.-Nr.: 5289911 und 9431638

Aufbau:







Allgemeine Funktionsweise:

Die Kommunikation mit dem Ultraschallsensor erfolgt durch ein kurzes Senden eines Signals auf den Trigger-Output. Daraufhin wird mittels eines Interrupts auf ein ECHO-Signal reagiert. Sobald dieses kommt, wird die ISR gestartet. Mithilfe des Timer wird die Zeit zwischen steigender und fallender Flanke gemessen und daraus die Distanz bestimmt. Diese wird dann für die weitere Verwendung in cm umgerechnet.
Mit dem Potentiometer kann die Helligkeit der LED-Anzeige variiert werden. Die Helligkeit kann Werte zwischen 0 und 255 annehmen.
Die Kommunikation mit dem LED-Strip erfolgt über SPI und den von Espressif zur Verfügung gestellten APIs. Die Funktion "displayLeds()“ steuert die LED-Anzeige. Als Input erwartet die Funktion ein Pointer auf ein Array mit Platz für 48 Elementen (Anzahl LEDs * 3 Farben), Anzahl der leuchtenden LEDs und die Farbe als Array mit den 8-Bit-Werten für Grün, Rot und Blau. In der Funktion wird dann das große Array berechnet, indem die Farb-Werte für die Anzahl an LEDs gemappt werden. Dieser Teil der Funktion ist in extrended asm syntax implementiert. Die restlichen LEDs werden Schwarz (0,0,0) gesetzt. Dann wird ein SPI-Buffer allokiert und initialisiert. Dieser wird mit der Funktion "encodeLedSKData()“ gefüllt. Diese verwendet die Zeitkonstanten aus dem Datenblatt und mappt diese auf die Farben. Am Ende fügt die Funktion eine RESET-Periode hinzu. Dieser nun vollständige Buffer wird in den MOSI-GPIO Pin synchron geschrieben. Dieser MOSI-GPIO Pin ist abhängig davon, ob der LED-Strip oder die Onboard LED beschaltet wird. Für das Wechseln des GPIO-Pins muss der Bus befreit und anschließend mit neuer Konfiguration initialisiert werden. Für die Onboard LED kann die gleiche Logik verwendet werden, jedoch die Zeitkonstanten müssen angepasst werden, denn es ist eine LED anderen Typs (WS2812). Hierfür gibt es in der "defines.h“ die jeweiligen Werte.

Umgesetzte Zusatzaufgaben:

1. Durch Mittelwertbildung und der Einsatz eines Buffers wird die kalkulierte Distanz stabilisiert.
2. Bei Änderung der Distanz im Bereich von 50 bis 5 cm wird neben der Anzahl der LEDs auch die Farben geändert. Eir haben ein Farbverlauf von Blau über Rot zu Grün.
3. Für GPIO1 (ECHO-Signal) ist der Interrupt für sowohl steigende als auch fallende Flanke des Signals aktiviert. Wenn dieser Auslöst startet die ISR. Hier wird überprüft, ob der Interrupt tatsächlich vom ECHO-Signal kommt und dann der Timer gestartet. Bei Fallender Flanke wird der Timer gestoppt und dann die Distanz aus der Zeit berechnet. Zum Schluss wird der Interrupt wieder zurückgesetzt.
4. Über ein Potentiometer, dessen Ausgang an GPIO0 angeschlossen ist, lässt sich die Helligkeit der Anzeige variieren. Hierfür kommt der interne ADC zum Einsatz, der das analoge Signal in digitale Werte umwandelt. Die Minimal- und Maximal-Werte stehen ebenfalls in der "defines.h“ und können, wie unten in ADC-Setup beschrieben, konfiguriert werden.

5. Bei weniger als 5 cm kommt die Onboard-LED des ESP32C3-DevkitM zum Einsatz. Diese zeigt dann mit den Farben Rot, Gelb, Grün die Distanz zwischen den Werten 2 cm und 5 cm. Wenn sowohl die Onboard-LED als auch der LED-Strip rot leuchten, liegt ein Fehlerhafter Wert vor. (größer als 400 oder kleiner als 2, der Wert -1 wird als Fehler-Flag benutzt)

ADC-Setup:

Wenn Potentiometer oder Microcontroller gewechselt werden, kann mithilfe der "setupADC()“ Funktion der Minimum- und Maximum-Wert bestimmt werden. Dafür muss in der "defines.h“ die Flags "DEBUG“ und "SETUP“ auf 1 gesetzt werden. Dann gibt das Programm beim Start interaktiv die beiden Grenz-Werte des ADC aus. Außerdem werden im DEBUG-Mode die Werte für Distanz, Helligkeit und die Anzahl der leuchtenden LEDs auf dem Monitor ausgegeben.