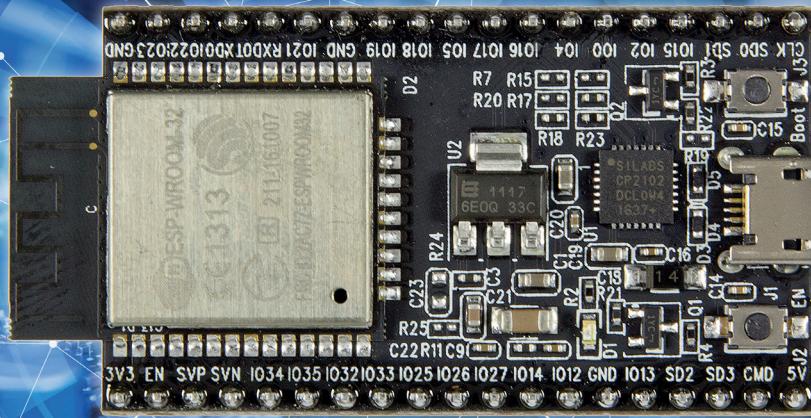


# ENTDECKE DEN IoT-CHIP

# DAS OFFIZIELLE ESP32-HANDBUCH



# Dogan Ibrahim und Ahmet Ibrahim

The Elektor logo features the word "elektor" in a lowercase, sans-serif font. The letter "e" is enclosed in a red circle, while the remaining letters are in a grey rectangle.

LEARN ➤ DESIGN ➤ SHARE

---

# **Das offizielle ESP32-Handbuch**



**Dogan Ibrahim  
Achmet Ibrahim**



an Elektor Publication

LEARN | DESIGN | SHARE

● © 2018: Elektor Verlag GmbH, Aachen.

1. Auflage 2018

● Alle Rechte vorbehalten.

Die in diesem Buch veröffentlichten Beiträge, insbesondere alle Aufsätze und Artikel sowie alle Entwürfe, Pläne, Zeichnungen und Illustrationen sind urheberrechtlich geschützt. Ihre auch auszugsweise Vervielfältigung und Verbreitung ist grundsätzlich nur mit vorheriger schriftlicher Zustimmung des Herausgebers gestattet.

Die Informationen im vorliegenden Buch werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Die in diesem Buch erwähnten Soft- und Hardwarebezeichnungen können auch dann eingetragene Warenzeichen sein, wenn darauf nicht besonders hingewiesen wird. Sie gehören dem jeweiligen Warenzeicheninhaber und unterliegen gesetzlichen Bestimmungen.

Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autor können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Für die Mitteilung eventueller Fehler sind Verlag und Autor dankbar.

Umschlaggestaltung: Elektor, Aachen

Satz und Aufmachung: D-Vision, Julian van den Berg | Oss (NL)

Druck: Media-Print Informationstechnologie GmbH, Paderborn

Printed in Germany

● ISBN 978-3-89576-329-8

Elektor-Verlag GmbH, Aachen

[www.elektor.de](http://www.elektor.de)



Elektor ist Teil der Unternehmensgruppe Elektor International Media (EIM), der weltweit wichtigsten Quelle für technische Informationen und Elektronik-Produkte für Ingenieure und Elektronik-Entwickler und für Firmen, die diese Fachleute beschäftigen. Das internationale Team von Elektor entwickelt Tag für Tag hochwertige Inhalte für Entwickler und DIY-Elektroniker, die über verschiedene Medien (Magazine, Videos, digitale Medien sowie Social Media) in zahlreichen Sprachen verbreitet werden. [www.elektor.de](http://www.elektor.de)

LEARN DESIGN SHARE

---

<b>VORWORT .....</b>	<b>19</b>
<b>Über die Autoren .....</b>	<b>21</b>
<b>KAPITEL 1 • DER ESP32-PROZESSOR.....</b>	<b>23</b>
1.1 Übersicht .....	23
1.2 Die Architektur des ESP32 .....	24
1.2.1 Die CPU.....	25
1.2.2 Interner Speicher .....	26
1.2.3 Externer Speicher.....	26
1.2.4 Universal-Timer .....	26
1.2.5 Watchdog-Timer .....	26
1.2.6 Der Systemtakt .....	26
1.2.7 Transceiver .....	26
1.2.8 Universal-Ein- und Ausgänge (GPIOs).....	26
1.2.9 Analog-Digital-Wandler (ADC) .....	27
1.2.10 Digital-Analog-Wandler (DAC).....	27
1.2.11 Hall-Sensor .....	27
1.2.12 Temperatursensor.....	27
1.2.13 Touch-Sensor .....	27
1.2.14 UART .....	27
1.2.15 I <sup>2</sup> C-Schnittstelle .....	27
1.2.16 I <sup>2</sup> S-Schnittstelle .....	27
1.2.17 Infrarot-Controller.....	27
1.2.18 Pulsweitenmodulation .....	28
1.2.19 LED-PWM .....	28
1.2.20 Impulszähler .....	28
1.2.21 SPI-Schnittstelle .....	28
1.2.22 Hardware-Beschleuniger .....	28
1.3 ESP32-Entwicklungsboards .....	28
1.3.1 SparkFun ESP32 Thing .....	28
1.3.2 Geekcreit ESP32 Development Board .....	29
1.3.3 LoLin32 ESP32 Development Board.....	30

1.3.4 Pycom LoPy Development Board . . . . .	30
1.3.5 ESP32 Test Board . . . . .	31
1.3.6 ESP32 Development Board von Pesky Products . . . . .	31
1.3.7 ESP32 OLED Development Board . . . . .	31
1.3.8 MakerHawk ESP32 Development Board . . . . .	32
1.3.9 ESP32-DevKitC . . . . .	32
1.3.10 Weitere Entwicklungsboards . . . . .	33
1.4 Zusammenfassung . . . . .	34
<b>KAPITEL 2 • DAS ESP32-DEVKITC DEVELOPMENT BOARD . . . . .</b>	<b>35</b>
2.1 Übersicht . . . . .	35
2.2 ESP32-DevKitC Hardware . . . . .	35
2.3 Einschalten des ESP32-DevKitC . . . . .	38
2.3.1 help . . . . .	40
2.3.2 op . . . . .	40
2.3.3 sta . . . . .	41
2.3.4 mac . . . . .	42
2.3.5 dhcp . . . . .	42
2.3.6 reboot . . . . .	42
2.3.7 ram . . . . .	42
2.3.8 ip . . . . .	43
2.3.9 ap . . . . .	43
2.4 Zusammenfassung . . . . .	43
<b>KAPITEL 3 • ARDUINO-IDE FÜR DAS ESP32-DEVKITC . . . . .</b>	<b>45</b>
3.1 Übersicht . . . . .	45
3.2 Installation der Arduino-IDE für das ESP32-DevKitC . . . . .	45
<b>KAPITEL 4 • BASISPROJEKTE MIT DER ARDUINO-IDE UND DEM ESP32-DEVKITC . . . . .</b>	<b>54</b>
4.1 Übersicht . . . . .	54
4.2 PROJEKT 1 – Blinkende LED . . . . .	54
4.2.1 Beschreibung . . . . .	54
4.2.2 Das Ziel . . . . .	54
4.2.3 Blockschaltbild . . . . .	54
4.2.4 Schaltplan . . . . .	55

4.2.5 Aufbau . . . . .	55
4.2.6 PDL des Projekts. . . . .	56
4.2.7 Programmlisting . . . . .	56
4.2.8 Programmbeschreibung. . . . .	57
4.2.9 Vorschläge. . . . .	57
4.3 PROJEKT 2 – Leuchtturm-LED . . . . .	57
4.3.1 Beschreibung . . . . .	57
4.3.2 Das Ziel. . . . .	58
4.3.3 Blockschaltbild . . . . .	58
4.3.4 Schaltplan . . . . .	58
4.3.5 Aufbau . . . . .	58
4.3.6 PDL des Projekts. . . . .	58
4.3.7 Programmlisting . . . . .	58
4.3.8 Programmbeschreibung. . . . .	59
4.4 PROJEKT 3 – Abwechselnd blinkende LEDs . . . . .	59
4.4.1 Beschreibung . . . . .	59
4.4.2 Das Ziel. . . . .	60
4.4.3 Blockschaltbild . . . . .	60
4.4.4 Schaltplan . . . . .	60
4.4.5 Aufbau . . . . .	60
4.4.6 PDL des Projekts. . . . .	61
4.4.7 Programmlisting . . . . .	61
4.4.8 Programmbeschreibung. . . . .	62
4.5 PROJEKT 4 – Rotierende LEDs . . . . .	62
4.5.1 Beschreibung . . . . .	62
4.5.2 Das Ziel. . . . .	63
4.5.3 Blockschaltbild . . . . .	63
4.5.4 Schaltplan . . . . .	63
4.5.5 Aufbau . . . . .	64
4.5.6 PDL des Projekts. . . . .	64
4.5.7 Programmlisting . . . . .	65
4.5.8 Programmbeschreibung. . . . .	66

4.5.9 Vorschläge . . . . .	66
4.6 PROJEKT 5 – Weihnachtsbeleuchtung . . . . .	66
4.6.1 Beschreibung . . . . .	66
4.6.2 Das Ziel . . . . .	66
4.6.3 Blockschaltbild . . . . .	66
4.6.4 Schaltplan . . . . .	66
4.6.5 Aufbau . . . . .	66
4.6.6 PDL des Projekts . . . . .	67
4.6.7 Programmlisting . . . . .	67
4.6.8 Programmbeschreibung . . . . .	69
4.6.9 Geändertes Programm . . . . .	69
4.6.10 Vorschläge . . . . .	71
4.7 PROJEKT 6 – Binärzähler mit LEDs . . . . .	71
4.7.1 Beschreibung . . . . .	71
4.7.2 Das Ziel . . . . .	71
4.7.3 Blockschaltbild . . . . .	71
4.7.4 Schaltplan . . . . .	71
4.7.5 Bau . . . . .	72
4.7.6 PDL des Projekts . . . . .	72
4.7.7 Programmlisting . . . . .	73
4.7.8 Programmbeschreibung . . . . .	74
4.8 PROJEKT 7 – Binärer Aufwärts-/Abwärtszähler mit LEDs . . . . .	75
4.8.1 Beschreibung . . . . .	75
4.8.2 Das Ziel . . . . .	75
4.8.3 Blockschaltbild . . . . .	75
4.8.4 Schaltplan . . . . .	76
4.8.5 Aufbau . . . . .	77
4.8.6 PDL des Projekts . . . . .	77
4.8.7 Programmlisting . . . . .	78
4.8.8 Programmbeschreibung . . . . .	80
4.9 PROJEKT 8 – Knight Rider LEDs . . . . .	80
4.9.1 Beschreibung . . . . .	80

---

4.9.2 Das Ziel . . . . .	80
4.9.3 Blockschaltbild . . . . .	80
4.9.4 Schaltplan . . . . .	80
4.9.5 Bau . . . . .	80
4.9.6 PDL des Projekts . . . . .	80
4.9.7 Programmlisting . . . . .	81
4.9.8 Programmbeschreibung . . . . .	82
4.9.9 Vorschläge . . . . .	82
4.10 PROJEKT 9 – Ändern der Helligkeit einer LED . . . . .	82
4.10.1 Beschreibung . . . . .	82
4.10.2 Das Ziel . . . . .	82
4.10.3 Blockschaltbild . . . . .	82
4.10.4 Schaltplan . . . . .	82
4.10.5 Aufbau . . . . .	83
4.10.6 PDL des Projekts . . . . .	83
4.10.7 Programmlisting . . . . .	83
4.10.8 Programmbeschreibung . . . . .	84
4.10.9 Vorschläge . . . . .	85
4.11 PROJEKT 10 – Zufällige Sounds mit einem Summer . . . . .	86
4.11.1 Beschreibung . . . . .	86
4.11.2 Das Ziel . . . . .	86
4.11.3 Blockschaltbild . . . . .	86
4.11.4 Schaltplan . . . . .	86
4.11.5 Aufbau . . . . .	86
4.11.6 PDL des Projekts . . . . .	87
4.11.7 Programmlisting . . . . .	88
4.11.8 Programmbeschreibung . . . . .	89
4.11.9 Vorschlag . . . . .	89
4.12 PROJEKT 11 – LED-Farbtafel . . . . .	89
4.12.1 Beschreibung . . . . .	89
4.12.2 Das Ziel . . . . .	90
4.12.3 Blockschaltbild . . . . .	90

4.12.4 Schaltplan . . . . .	90
4.12.5 Aufbau. . . . .	90
4.12.6 PDL des Projekts. . . . .	91
4.12.7 Programmlisting . . . . .	92
4.12.8 Programmbeschreibung . . . . .	93
4.12.9 Vorschläge . . . . .	93
4.13 Zusammenfassung . . . . .	93
<b>KAPITEL 5 • EINFACHE PROJEKTE MIT DER ARDUINO-IDE UND DEM ESP32-DEVKITC . . . . .</b>	<b>94</b>
5.1 Übersicht . . . . .	94
5.2 PROJEKT 1 – Thermometer mit seriellem Monitor . . . . .	94
5.2.1 Beschreibung . . . . .	94
5.2.2 Das Ziel. . . . .	94
5.2.3 Blockschaltbild . . . . .	94
5.2.4 Schaltplan . . . . .	94
5.2.5 Konstruktion . . . . .	95
5.2.6 PDL des Projekts. . . . .	96
5.2.7 Programmlisting . . . . .	96
5.2.8 Programmbeschreibung . . . . .	97
5.3 PROJEKT 2 – Temperatur und relative Luftfeuchtigkeit im seriellen Monitor. . . . .	99
5.3.1 Beschreibung . . . . .	99
5.3.2 Das Ziel. . . . .	99
5.3.3 Blockschaltbild . . . . .	99
5.3.4 Schaltplan . . . . .	99
5.3.5 Konstruktion . . . . .	101
5.3.6 PDL des Projekts. . . . .	101
5.3.7 Programmlisting . . . . .	102
5.3.8 Programmbeschreibung . . . . .	104
5.4 PROJEKT 3 – Messung der Lichtstärke . . . . .	104
5.4.1 Beschreibung . . . . .	104
5.4.2 Das Ziel. . . . .	104
5.4.3 Blockschaltbild . . . . .	104

---

5.4.4 Schaltplan . . . . .	105
5.4.5 Konstruktion . . . . .	106
5.4.6 PDL des Projekts. . . . .	106
5.4.7 Programmlisting . . . . .	107
5.4.8 Programmbeschreibung . . . . .	108
5.4.9 Vorschläge . . . . .	108
5.5 PROJEKT 4 – Dunkelheitserkennung. . . . .	108
5.5.1 Beschreibung . . . . .	108
5.5.2 Das Ziel. . . . .	108
5.5.3 Blockschaltbild . . . . .	108
5.5.4 Schaltplan . . . . .	109
5.5.5 Konstruktion . . . . .	109
5.5.6 PDL des Projekts. . . . .	110
5.5.7 Programmlisting . . . . .	111
5.5.8 Programmbeschreibung . . . . .	112
5.5.9 Vorschläge . . . . .	112
5.6 PROJEKT 5 – LED-Würfel . . . . .	112
5.6.1 Beschreibung . . . . .	112
5.6.2 Das Ziel. . . . .	112
5.6.3 Blockdiagramm: . . . . .	112
5.6.4 Schaltplan . . . . .	113
5.6.5 Konstruktion . . . . .	114
5.6.6 PDL des Projekts. . . . .	114
5.6.7 Programmlisting . . . . .	115
5.6.8 Programmbeschreibung . . . . .	117
5.7 PROJEKT 6 – Logiktester . . . . .	118
5.7.1 Beschreibung . . . . .	118
5.7.2 Das Ziel. . . . .	118
5.7.3 Blockschaltbild . . . . .	118
5.7.4 Schaltplan . . . . .	119
5.7.5 Konstruktion . . . . .	119

5.7.6 PDL des Projekts . . . . .	119
5.7.7 Programmlisting . . . . .	120
5.7.8 Programmbeschreibung . . . . .	121
5.7.9 Geändertes Programm . . . . .	121
5.8 PROJEKT 7 – Zähler mit 7-Segment-LED-Anzeige . . . . .	125
5.8.1 Beschreibung . . . . .	125
5.8.2 Das Ziel . . . . .	127
5.8.3 Blockschaltbild . . . . .	127
5.8.4 Schaltplan . . . . .	127
5.8.5 Aufbau . . . . .	128
5.8.6 PDL des Projekts . . . . .	129
5.8.7 Programmlisting . . . . .	129
5.8.8 Programmbeschreibung . . . . .	131
5.8.9 Geändertes Programm . . . . .	132
5.9 PROJEKT 8 – Klatschschalter . . . . .	134
5.9.1 Beschreibung . . . . .	134
5.9.2 Das Ziel . . . . .	134
5.9.3 Blockschaltbild . . . . .	134
5.9.4 Schaltplan . . . . .	135
5.9.5 Konstruktion . . . . .	135
5.9.6 PDL des Projekts . . . . .	136
5.9.7 Programmlisting . . . . .	136
5.9.8 Programmbeschreibung . . . . .	137
5.10 PROJEKT 9 – LCD „Hello from ESP32“ . . . . .	138
5.10.1 Beschreibung . . . . .	138
5.10.2 Das Ziel . . . . .	138
5.10.3 Blockschaltbild . . . . .	138
5.10.4 Schaltplan . . . . .	139
5.10.5 Konstruktion . . . . .	139
5.10.6 PDL des Projekts . . . . .	140
5.10.7 Programmlisting . . . . .	141

---

5.10.8 Programmbeschreibung . . . . .	142
5.11 PROJEKT 10 – LCD-Ereigniszähler . . . . .	142
5.11.1 Beschreibung . . . . .	142
5.11.2 Das Ziel . . . . .	142
5.11.3 Blockschaltbild . . . . .	142
5.11.4 Schaltplan . . . . .	143
5.11.5 Konstruktion . . . . .	143
5.11.6 PDL des Projekts . . . . .	143
5.11.7 Programmlisting . . . . .	144
5.11.8 Programmbeschreibung . . . . .	145
5.12 PROJEKT 11 – LCD-BEFEHLE . . . . .	145
5.12.1 Beschreibung . . . . .	145
5.12.2 Das Ziel . . . . .	145
5.12.3 Blockschaltbild . . . . .	146
5.12.4 Schaltplan . . . . .	146
5.12.5 Konstruktion . . . . .	146
5.12.6 LCD-Befehle . . . . .	146
5.12.7 Programmlisting . . . . .	147
5.12.8 Programmbeschreibung . . . . .	148
5.13 PROJEKT 12 – EXTERNE INTERRUPTS . . . . .	149
5.13.1 Beschreibung . . . . .	149
5.13.2 Das Ziel . . . . .	149
5.13.3 Blockschaltbild . . . . .	149
5.13.4 Schaltplan . . . . .	150
5.13.5 Konstruktion . . . . .	150
5.13.6 PDL des Projekts . . . . .	150
5.13.7 Programmlisting . . . . .	151
5.13.8 Programmbeschreibung . . . . .	153
5.14 PROJEKT 13 – TIMER-INTERRUPTS . . . . .	153
5.14.1 Beschreibung . . . . .	153
5.14.2 Das Ziel . . . . .	153

5.14.3 Blockdiagramm . . . . .	153
5.14.4 Schaltplan . . . . .	153
5.14.5 Konstruktion . . . . .	153
5.14.6 PDL des Projekts . . . . .	153
5.14.7 Programmlisting . . . . .	154
5.14.8 Programmbeschreibung . . . . .	155
5.15 Zusammenfassung . . . . .	155
<b>KAPITEL 6 • KOMPLEXERE PROJEKTE MIT DER ARDUINO-IDE UND DEM ESP32-DEVKITC . . . . .</b>	<b>156</b>
6.1 Übersicht . . . . .	156
6.2 PROJEKT 1 – Thermostat-Regelung . . . . .	156
6.2.1 Beschreibung . . . . .	156
6.2.2 Das Ziel . . . . .	156
6.2.3 Blockschaltbild: . . . . .	156
6.2.4 Schaltplan . . . . .	157
6.2.5 Konstruktion . . . . .	158
6.2.6 PDL des Projekts . . . . .	158
6.2.7 Programmlisting . . . . .	159
6.2.8 Programmbeschreibung . . . . .	162
6.3 PROJEKT 2 – Wellenformen erzeugen: Sägezahn . . . . .	163
6.3.1 Beschreibung . . . . .	163
6.3.2 Das Ziel . . . . .	164
6.3.3 Blockschaltbild: . . . . .	164
6.3.4 Der DAC . . . . .	164
6.3.5 Schaltplan . . . . .	165
6.3.6 Konstruktion . . . . .	165
6.3.7 PDL des Projekts . . . . .	165
6.3.8 Programmlisting . . . . .	166
6.3.9 Programmbeschreibung . . . . .	167
6.4 PROJEKT 3 – Wellenformen erzeugen: Dreieck . . . . .	167
6.4.1 Beschreibung . . . . .	167
6.4.2 Das Ziel . . . . .	167

---

6.4.3 Blockschaltbild:	167
6.4.4 Schaltplan	168
6.4.5 PDL des Projekts.	168
6.4.6 Programmlisting	168
6.4.7 Programmbeschreibung	169
6.5 PROJEKT 4 – Port-Expander	169
6.5.1 Beschreibung	169
6.5.2 Das Ziel	169
6.5.3 Blockschaltbild	170
6.5.4 Schaltplan	170
6.5.5 Der MCP23017	170
6.5.6 Konstruktion	172
6.5.7 PDL des Projekts.	173
6.5.8 Programmlisting	173
6.5.9 Programmbeschreibung	175
6.6 PROJEKT 5 – Elektronische Mini-Orgel	175
6.6.1 Beschreibung	175
6.6.2 Das Ziel	175
6.6.3 Blockschaltbild	176
6.6.4 Schaltplan	176
6.6.5 Konstruktion	178
6.6.6 PDL des Projekts.	178
6.6.7 Programmlisting	179
6.6.8 Programmbeschreibung	183
6.7 PROJEKT 6 – Rechner mit Tastatur und LCD	183
6.7.1 Beschreibung	183
6.7.2 Das Ziel	184
6.7.3 Blockschaltbild:	184
6.7.4 Schaltplan	184
6.7.5 Konstruktion	185
6.7.6 PDL des Projekts.	186

6.7.7 Programmlisting . . . . .	186
6.7.8 Programmbeschreibung . . . . .	193
6.8 PROJEKT 7 – HIGH-LOW-SPIEL . . . . .	194
6.8.1 Beschreibung . . . . .	194
6.8.2 Das Ziel . . . . .	195
6.8.3 Blockdiagramm: . . . . .	195
6.8.4 Schaltplan . . . . .	195
6.8.5 Konstruktion . . . . .	195
6.8.6 PDL des Projekts. . . . .	195
6.8.7 Programmlisting . . . . .	196
6.8.8 Programmbeschreibung . . . . .	199
6.9 PROJEKT 8 – Das kleine Einmaleins . . . . .	200
6.9.1 Beschreibung . . . . .	200
6.9.2 Das Ziel . . . . .	200
6.9.3 Blockschaltbild: . . . . .	200
6.9.4 Schaltplan . . . . .	201
6.9.5 Konstruktion . . . . .	201
6.9.6 PDL des Projekts. . . . .	201
6.9.7 Programmlisting . . . . .	201
6.9.8 Programmbeschreibung . . . . .	204
6.10 PROJEKT 9 – Lernen der Grundmathematik. . . . .	205
6.10.1 Beschreibung . . . . .	205
6.10.2 Das Ziel . . . . .	205
6.10.3 Blockschaltbild: . . . . .	205
6.10.4 Schaltplan . . . . .	205
6.10.5 Konstruktion. . . . .	205
6.10.6 PDL des Projekts. . . . .	205
6.10.7 Programmlisting . . . . .	206
6.10.8 Programmbeschreibung . . . . .	207
6.10.9 Vorschläge . . . . .	207
6.11 PROJEKT 10 – Code-Türschloss . . . . .	208

---

6.11.1 Beschreibung . . . . .	208
6.11.2 Das Ziel . . . . .	208
6.11.3 Blockschaltbild: . . . . .	208
6.11.4 Schaltplan . . . . .	208
6.11.5 Konstruktion. . . . .	209
6.11.6 PDL des Projekts . . . . .	209
6.11.7 Programmlisting . . . . .	210
6.11.8 Programmbeschreibung . . . . .	213
6.11.9 Vorschläge . . . . .	214
6.12 Zusammenfassung . . . . .	214
<b>KAPITEL 7 • ESP32-DEVKITC NETZWERKPROGRAMMIERUNG MIT DER ARDUINO-IDE . . . . .</b>	<b>215</b>
7.1 Übersicht . . . . .	215
7.2 Scannen der erreichbaren Wi-Fi-Netzwerke. . . . .	215
7.3 Verbindung zu einem Wi-Fi-Netzwerk herstellen . . . . .	217
7.4 HTTP-GET-Anfragen . . . . .	220
7.5 Verwenden der Socket-Bibliothek . . . . .	222
7.5.1 UDP-Programme . . . . .	223
7.5.2 TCP/IP-Programme . . . . .	227
7.6 Zusammenfassung. . . . .	229
<b>KAPITEL 8 • PROJEKT – TEMPERATUR UND FEUCHTE IN DER CLOUD . . . . .</b>	<b>230</b>
8.1 Übersicht . . . . .	230
8.2 Das Blockschaltungs . . . . .	230
8.3 Die Cloud . . . . .	230
8.4 Programmlisting . . . . .	232
8.5 Zusammenfassung. . . . .	236
<b>KAPITEL 9 • WEB-BASIERTE FERNSTEUERUNG . . . . .</b>	<b>237</b>
9.1 Übersicht . . . . .	237
9.2 Das Blockschaltbild . . . . .	237
9.3 HTTP-Webserver/Client. . . . .	238
9.4 Programmlisting ESP32-DevKitC . . . . .	239
9.5 Zusammenfassung. . . . .	243

<b>KAPITEL 10 • FERNBEDIENUNG MIT DEM MOBILTELEFON .....</b>	<b>244</b>
10.1 Überblick .....	244
10.2 Das Blockschaltbild .....	244
10.3 Smartphone-Anwendung .....	245
10.4 ESP32-DevKitC Programmlisting .....	245
10.5 Zusammenfassung.....	249
<b>KAPITEL 11 • TEMPERATUR UND LUFTFEUCHTE AN EIN HANDY SENDEN .....</b>	<b>250</b>
11.1 Übersicht .....	250
11.2 Die Blockschaltung.....	250
11.3 Handy-Anwendung.....	251
11.4 ESP32-DevKitC Programmlisting .....	251
11.5 Zusammenfassung.....	255
<b>KAPITEL 12 • MICROPYTHON AUF DEM ESP32-DEVKITC.....</b>	<b>256</b>
12.1 Übersicht .....	256
12.2 Installation von MicroPython auf ESP32-DevKitC .....	256
12.3 Testen der MicroPython-Installation .....	259
12.4 Blinkende LED .....	260
12.5 LED mit Drucktaster.....	260
12.6 Temperatur und Luftfeuchtigkeit .....	261
12.7 Verbindung zu einem WLAN herstellen .....	261
12.8 MicroPython-UDP-Programme .....	262
12.9 Speichern von Temperatur und Luftfeuchtigkeit in der Cloud.....	265
12.10 Fernbedienung über Mobiltelefon (Webserver).....	267
12.11 Laden von MicroPython-Programmen auf das ESP32-DevKitC .....	271
12.11.1 Verwenden der Aropy.....	272
12.11.2 Erstellen und Ausführen eines Programms .....	273
12.11.3 Ausführen eines Programms beim Booten .....	275
12.12 Zusammenfassung .....	278
<b>Index .....</b>	<b>279</b>

## VORWORT

WLAN-Netzwerke werden in den entwickelten Ländern in fast allen Häusern, Büros und öffentlichen Plätzen eingesetzt, um Geräte wie PCs, Smartphones, moderne Drucker und Tablets an das Internet anzuschließen. Ein drahtloser Access-Point (AP) stellt die Verbindung zwischen all diesen Geräten her. APs arbeiten normalerweise im 2,4-GHz-Frequenzband und besitzen Reichweiten bis maximal etwa 100 m.

Die Verbindung zu einem WLAN-AP erfordert ein kompatibles Drahtlosnetzwerk-Schnittstellenmodul auf den Geräten, die mit dem Internet verbunden werden sollen. Laptop- und Desktop-Computer verfügen über Drahtlosnetzwerk-Schnittstellenkarten (NICs), um über den AP mit dem Internet zu kommunizieren. Alles, was für die Kommunikation erforderlich ist, ist der WLAN-Name (SSID-Name) des AP und das Passwort. Früher war solche NIC-Hardware sperrig und teuer und hatte eine hohe Stromaufnahme, was zu voluminösen Stromversorgungen führte. Heutzutage gibt es NIC-Hardware in vielen verschiedenen Formen, Größen und Preisen, zum Beispiel als eingebaute Elektronikmodule, externe Karten oder externe Flash-Speicher in der Größe von USB-Modulen.

Mikrocontroller sind sehr beliebt und werden häufig in Haushaltsgeräten in kommerziellen und industriellen elektronischen Überwachungs- und Steuerungsanwendungen eingesetzt. Man geht davon aus, dass in den Industrieländern in jedem Haus mehr als 50 eingebettete Mikrocontroller ihren Dienst tun, in Mikrowellenherden, Druckern, Tastaturen, Computern, Tablets, Waschmaschinen, Geschirrspülern, Smart-TVs, Mobiltelefonen und vielen Geräten mehr.

Ein kleiner Chip namens ESP8266 ermöglicht es seit kurzer Zeit, jede Art von Mikrocontroller an einen WLAN-AP anzuschließen. Der ESP8266 ist ein kostengünstiger, winziger WLAN-Chip mit eingebautem TCP/IP-Stack und einem 32-Bit-Mikrocontroller. Dieser Chip, der vom in Shanghai ansässigen chinesischen Hersteller Espressif Systems produziert wird, ist kompatibel mit der WLAN-Norm IEEE 802.11 b/g/n, besitzt einen internen Programm- und Datenspeicher und universell nutzbare Input-Output-Ports. Mehrere Hersteller haben den ESP8266-Chip in ihre Hardware-Produkte (zum Beispiel ESP-xx, NodeMCU) integriert und bieten diese Produkte an, um ein Mikrocontroller-System wie Android, PIC oder andere an ein WLAN anzuschließen. Der ESP8266 ist ein Low-Power-Chip und kostet nur wenige Euro.

Zusätzlich zum ESP8266 hat Espressif Systems vor kurzem den neuen Mikrocontroller ESP32 entwickelt. Dabei handelt es sich um einen großen Bruder des ESP8266, der nicht nur in allen bisherigen ESP8266-Projekten verwendet werden kann, sondern auch zusätzliche Eigenschaften wie Bluetooth-Kommunikation, einen größeren SRAM-Datenspeicher, mehr GPIOs, mehr Schnittstellensignale, Touch-Sensoren, einen Temperatursensor, höhere CPU-Geschwindigkeit, CAN-Bus-Konnektivität, A/D-Wandler mit höherer Auflösung, D/A-Wandler und Sicherheitsfunktionen bietet.

Dieses Buch soll den ESP32-Prozessor einführen und die wichtigsten Hardware- und Software-Features dieses Chips beschreiben. Hauptziel des Buches ist es, dem Leser zu vermit-

teln, wie man Hard- und Software des ESP32 in praktischen Projekten einsetzt, vor allem mithilfe des hochmodernen ESP32-Entwicklungsboards. Viele grundlegende einfache bis mittelschwere Projekte in dem Buch basieren auf dem Entwicklungsboard ESP32-DevKitC und verwenden sowohl die sehr populäre Arduino-IDE als auch die Programmiersprache MicroPython. Die Projekte in diesem Buch weisen einen zunehmenden Schwierigkeitsgrad auf.

Bei Elektor ist ein Hardware-Kit speziell für dieses Buch erhältlich. Das Kit enthält alle Bauteile, die in den Projekten des Buchs verwendet werden. Mithilfe dieses Hardware-Kits sollte es problemlos und einfach gelingen, die Projekte im Buch aufzubauen.

Wir hoffen, dass Sie das Buch mit Vergnügen lesen und gleichzeitig lernen, wie Sie den ESP32 Prozessor in Ihren zukünftigen Projekten nutzen können.

*Dogan Ibrahim  
Ahmet Ibrahim*

*London, 2017*

## Über die Autoren

Prof. Dr. Dogan Ibrahim hat einen Abschluss als Bachelor of Science in Elektrotechnik, hält einen Master of Science in der automatischen Steuerungstechnik und einen Doktor in der digitalen Signalverarbeitung. Dogan hat lange in der Industrie gearbeitet, bevor er zum akademischen Leben zurückgekehrt ist. Professor Ibrahim ist Autor von über 60 Fachbüchern und über 200 Fachartikeln über Mikrocontroller, Mikroprozessoren und verwandte Bereiche. Er ist Diplomingenieur der Elektrotechnik und Mitglied der Institution of Engineering Technology.

Ahmet Ibrahim erhielt seinen Abschluss als Bachelor of Science von der Greenwich-Universität in London. Danach absolvierte er einen Master-of-Science-Kurs an derselben Universität. Auch Ahmet ist bei vielen Industrie-Unternehmen auf verschiedenen Ebenen tätig gewesen. Derzeit arbeitet er in einem großen Unternehmen im IT-Bereich. Er ist Autor mehrerer Fachbücher und Fachartikel.



## KAPITEL 1 • DER ESP32-PROZESSOR

### 1.1 Übersicht

Beim sehr beliebten ESP8266-Prozessor, der weniger als 5 Euro kostet, handelt es sich im Grunde um einen WLAN-fähigen Mikrocontroller mit GPIOs, der in kleinen Überwachungs- und Steuerungsanwendungen eingesetzt werden kann. Der ESP8266 wurde vom chinesischen Hersteller Espressif Systems aus Shanghai entwickelt und enthält einen vollständigen TCP/IP-Stack. Es gibt eine Vielzahl von Informationen, Tutorials, Datenblättern, Anwendungen, Applikationsschriften, Büchern und Projekten auf der Grundlage des ESP8266. Mehrere Unternehmen haben kleine Entwicklungsplatinen mit diesem Prozessor entwickelt, etwa den ESP8266-Arduino und die NodeMCU-Reihe.

Vor kurzem hat Espressif einen im Vergleich zum ESP8266 leistungsfähigeren Prozessor ESP32 vorgestellt. Obwohl der ESP32 nicht als Ersatz oder Nachfolger des ESP8266 entwickelt wurde, verbessert er ihn in vielerlei Hinsicht. Der neue ESP32-Prozessor verfügt nicht nur über WLAN-Unterstützung, sondern auch über ein Bluetooth-Kommunikationsmodul, so dass der Prozessor mit Bluetooth-kompatiblen Geräten kommunizieren kann. Im ESP32 arbeitet eine 32-Bit-CPU namens Xtensa LX6, die der ESP8266-CPU zwar sehr ähnlich ist, aber zwei Kerne, mehr Datenspeicher, mehr GPIOs, höhere CPU-Geschwindigkeit, A/D-Wandler mit höherer Auflösung, D/A-Wandler und CAN-Bus-Konnektivität besitzt. Die Haupteigenschaften des ESP32-Prozessors sind:

- 32-Bit Xtensa RISC-CPU: Dual-Core Mikrocontroller Tensilica Xtensa LX6
- Betriebsgeschwindigkeit von 160 ... 240 MHz
- 520 KB SRAM
- 448 KB ROM
- 16 KB SRAM (in der RTC)
- IEEE 802.11 b/g/ne/I-WLAN
- Bluetooth 4.2
- 12-Bit-ADC mit 18 Kanälen
- 8-Bit-DAC mit 2 Kanälen
- 10 Touch-Sensoren
- Temperatursensor
- 36 GPIOs
- 4 x SPI
- 2 x I<sup>2</sup>C
- 2 x I<sup>2</sup>S
- 3 x UART
- 1 x CAN Bus 2.0
- SD-Speicherkarten-Unterstützung
- Betrieb mit 2,2 ... 3,36 V
- RTC-Timer und Watchdog
- Hall-Sensor
- 16 PWM-Kanäle
- Ethernet-Schnittstelle
- Interner 8-MHz- und RC-Oszillator

- Externer 2 ... 60-MHz- und 32 kHz Oszillator
- Kryptografische Hardware-Beschleunigung (AES, HASH, RSA, ECC, RNG)
- IEEE 802.11 Sicherheitsmerkmale
- 5 µA Stromaufnahme im Sleep-Modus

Tabelle 1.1 Vergleich der Grundeigenschaften von ESP32- und ESP8266-Prozessoren.

Technische Daten	ESP32	ESP8266
CPU	32-Bit Xtensa L106 Single-Core	32-Bit Xtensa LX6 Dual-Core
Betriebsfrequenz	160 MHz	80 MHz
Bluetooth	Bluetooth 4.2	nein
WLAN	Ja (HT40)	Ja (HT20)
SRAM	512 KB	160 KB
GPIOs	36	17
Hardware-PWM	1	nein
Software-PWM	16	8
SPI/I2C/I2S/UART	4/2/2/2	2/1/2/2
CAN	1	nein
ADC	12-bit	10-bit
Touch-Sensor	10	nein
Temperatursensor	1	nein
Ethernet-MAC-Schnittstelle	1	nein

Tabelle 1.2. Vergleich von ESP32 und ESP8266

## 1.2 Die Architektur des ESP32

Bild 1.1 zeigt die Funktionsblöcke des ESP32-Prozessors (siehe **ESP32-Datasheet**, Espressif Systems, 2017). Im Mittelpunkt befindet sich die Dual-Core-CPU Xtensa LX6 mit dem Speicher. Auf der linken Seite sind die peripheren Schnittstellenblöcke wie SPI, I<sup>2</sup>C, I<sup>2</sup>S, SDIO, UART, CAN, ETH, IR, PWM, Temperatursensor, Touch-Sensor, DAC und ADC zu sehen. Die Bluetooth- und WLAN-Module befinden sich oben in der Mitte des Blockschaltbildes, der Taktgenerator und der HF-Transceiver rechts daneben. In der rechten Mitte sind die Module für kryptographische Hardware-Beschleunigung wie SHA, RSA, AES und RNG zu sehen. Und ganz unten befinden sich RTC, PMU, Co-Prozessor und der Recovery-Speicher.

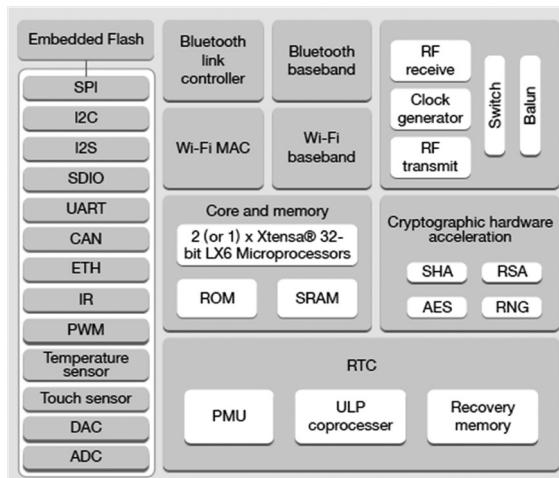


Bild 1.1. Funktionsblöcke des ESP32-Prozessors

Bild 1.2 zeigt die Struktur des Systems, bestehend aus der Dual-Core-CPU mit Harvard-Architektur mit den Bezeichnungen PRO\_CPU (für Protokoll-CPU) und APP\_CPU (für Applikations-CPU). Die Module dazwischen sind beiden CPUs gemeinsam. Detaillierte Informationen zur internen Architektur des ESP32 erhalten Sie im **ESP32 Technical Reference Manual** (Espressif Systems, 2017). Einige Informationen über die internen Module finden Sie weiter hinten in diesem Buch.

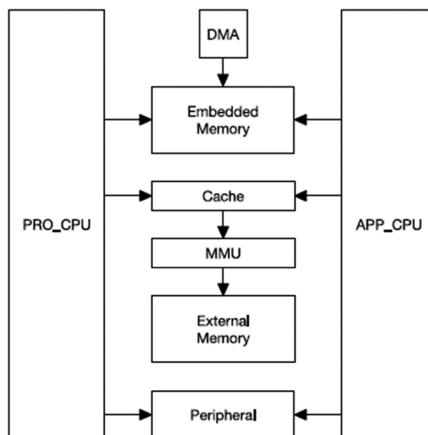


Bild 1.2. Systemstruktur

### 1.2.1 Die CPU

Die CPU kann mit bis zu 240 MHz betrieben werden und unterstützt 7-stufiges Pipelining mit einem 16/24-Bit-Befehlssatz. Gleitkomma-Operationen und DSP-Befehle wie 32-Bit-Multiplikator, 32-Bit-Treiler und 40-Bit-MAC werden unterstützt. Es stehen bis zu 70 externe und interne Interrupt-Quellen mit 32 Interrupt-Vektoren zur Verfügung. Das Debugging kann über die JTAG-Schnittstelle erfolgen.

### **1.2.2 Interner Speicher**

Auf dem Chip gibt es 520 KB SRAM und 448 KB ROM (zum Booten). Das RTC-Modul enthält je 8 KB langsamem und schnellen Speicher. Es ist 1 Kbit eFuse verfügbar, wobei 256 Bits für die MAC-Adresse und die Chip-Konfiguration reserviert sind, während die restlichen 768 Bits für User-Anwendungen offenstehen.

### **1.2.3 Externer Speicher**

Es werden bis zu 4x16 MB externer Flash- und SRAM-Speicher unterstützt, auf den über einen High-Speed-Cache zugegriffen werden kann. Bis zu 16 MB des externen Flash-Speichers werden auf den CPU-Code-Raum, bis zu 8 MB des externen Flash/SRAM-Speichers auf den CPU-Datenraum abgebildet. Das Lesen von Daten ist sowohl vom Flash als auch vom SRAM möglich, während Daten nur auf das SRAM geschrieben werden können.

### **1.2.4 Universal-Timer**

Vier 64-Bit-Universal-Timer können im ESP32-Prozessor software-gesteuert werden. Die Timer verfügen über 16-Bit-Vorteiler (2 ... 65535) und Auto-Reload-Auf- und Abwärtszähler. Die Timer können, wenn so konfiguriert, Interrupts erzeugen.

### **1.2.5 Watchdog-Timer**

Drei Watchdog-Timer mit programmierbaren Timeout-Werten stehen zur Verfügung. Zwei der Watchdog-Timer sind sogenannte „Main Watchdog Timer“ und befinden sich im Block der Universal-Timer, während der dritte, der „RTC Watchdog Timer“, sich im RTC-Modul befindet. Bei einem Reset der Watchdog-Timer können ein Interrupt, ein CPU-Reset, ein Core-Reset und ein System-Reset erzeugt werden.

### **1.2.6 Der Systemtakt**

Wenn der Prozessor zurückgesetzt wird, übernimmt ein externer Quarztakt das System-Timing. Die Taktfrequenz beträgt typischerweise 160 MHz und wird mithilfe einer PLL konfiguriert.

Ein genauer interner Takt (8 MHz) ist ebenfalls möglich. Dem Programmierer steht es frei, einen externen oder internen Takt zu wählen.

Die RTC kann mit einem externen 32-kHz-Quarz, einem internen RC-Oszillatator (typischerweise 150 kHz) oder einem internen 8-MHz-Oszillatator getaktet werden. Durch eine Teilung des internen 8-MHz-Taktes durch 256 ist auch eine Taktung der RTC mit 31,25 kHz möglich.

### **1.2.7 Transceiver**

Der ESP32-Prozessor verfügt über interne 2,4-GHz-Sende- und Empfangsmodule für WLAN- und Bluetooth-Kommunikation.

### **1.2.8 Universal-Ein- und Ausgänge (GPIOs)**

Es gibt 34 GPIOs, die für den digitalen, analogen oder den Einsatz als kapazitiver Touchscreen konfiguriert werden können. Digitalen GPIOs können durch Konfiguration interne Pull-up- oder Pull-Down-Widerstände hinzugefügt oder ein hochohmiger Zustand zugewiesen werden. Eingangs-Pins können so konfiguriert werden, dass sie Interrupts als Flanken oder Pegeländerung akzeptieren.

### **1.2.9 Analog-Digital-Wandler (ADC)**

Der ESP32-Prozessor enthält einen 12-Bit-A/D-Wandler mit 18 Kanälen. Um niedrige Analogspannungen zu messen, können einige der Eingänge als programmierbare Verstärker konfiguriert werden.

### **1.2.10 Digital-Analog-Wandler (DAC)**

Der ESP32-Prozessor verfügt über zwei unabhängige 8-Bit-D/A-Wandler.

### **1.2.11 Hall-Sensor**

Der Prozessor verfügt über einen resistiven Hall-Sensor. Befindet sich der Sensor in einem Magnetfeld, erzeugt er eine kleine Spannung, die vom A/D-Wandler gemessen werden kann.

### **1.2.12 Temperatursensor**

Weiterhin steht ein interner analoger Temperatursensor für Temperaturen im Bereich von -40 ... +125 °C zur Verfügung. Die gemessene Temperatur wird vom A/D-Wandler digitalisiert. Die Messung wird von der Temperatur des Chips und seiner aktiven Module beeinflusst, so dass der Temperatursensor nur zur Messung von Temperaturschwankungen und nicht der absoluten Temperatur geeignet ist.

### **1.2.13 Touch-Sensor**

Der Anschluss von bis zu zehn kapazitiven Touch-Sensoren ist möglich. Die GPIO-Pins können die kapazitiven Änderungen erfassen, die durch den direkten Kontakt mit einem Finger oder einem anderen geeigneten Objekt erzeugt werden.

### **1.2.14 UART**

Für die serielle Kommunikation (RS232, RS485 und IrDA) sind drei UARTs mit Geschwindigkeiten von bis zu 5 Mbps vorgesehen.

### **1.2.15 I<sup>2</sup>C-Schnittstelle**

ESP32-Prozessor bietet bis zu zwei I<sup>2</sup>C-Bus-Schnittstellen, die im Master- oder Slave-Modus konfiguriert werden können. Die Schnittstellen unterstützen den 400-Kbit/s-Fast-Transfer-Modus mit sieben oder zehn Bit breiter Adressierung. Auf diese Pins können externe Bauteile mit I<sup>2</sup>C-Bus-kompatibler Schnittstelle angeschlossen werden.

### **1.2.16 I<sup>2</sup>S-Schnittstelle**

Der ESP32-Prozessor unterstützt bis zu zwei I<sup>2</sup>S-Bus-Schnittstellen im Master- oder Slave-Modus, im Voll- oder Halbduplex-Betrieb. Die Taktfrequenz kann 10 kHz bis 40 MHz betragen.

### **1.2.17 Infrarot-Controller**

Bis zu 8 Kanäle einer programmierbaren Infrarot-Fernbedienung werden vom ESP32 unterstützt. Die Wellenformen beim Senden und Empfangen können im gemeinsam genutzten 512x32-Bit-Speicher niedergelegt werden.

### **1.2.18 Pulsweitenmodulation**

Pulsweitenmodulation (PWM) wird verwendet, um Geräte wie Motoren, elektrische Heizungen, intelligente Leuchten und Ähnliches zu steuern. Der ESP32 stellt dazu ein programmierbares Hardware-PWM-Modul und 16 per Software konfigurierbare PWM-Module bereit.

### **1.2.19 LED-PWM**

Mit der LED-PWM können bis zu 16 unabhängige digitale Wellenformen mit konfigurierbaren Tastverhältnissen und Perioden erzeugt werden. Das Tastverhältnis lässt sich per Software in einem Schritt-für-Schritt-Modus ändern.

### **1.2.20 Impulszähler**

Es gibt bis zu acht Impulszähler-Kanäle, um Impulse zu erfassen und Impulsflanken zu zählen. Wenn der Zähler einen vordefinierten Wert erreicht, kann ein Interrupt erzeugt werden.

### **1.2.21 SPI-Schnittstelle**

Bis zu vier SPI-Schnittstellen im Master- und Slave-Modus werden vom ESP32 unterstützt. An diese Pins können externe Bauteile mit SPI-Schnittstelle angeschlossen werden.

### **1.2.22 Hardware-Beschleuniger**

Der ESP32 unterstützt Hardware-Beschleuniger für mathematische Operationen mit Algorithmen wie AES, SHA, RSA und ECC. Diese Beschleuniger helfen, die Verarbeitungsgeschwindigkeit zu erhöhen und auch die Komplexität der Software zu verringern.

## **1.3 ESP32-Entwicklungsboards**

Der ESP32-Chip ist sehr komplex und lässt sich nicht ohne Hilfsmittel erforschen. Deshalb gibt es auf dem Markt mehrere Entwicklungsboards auf der Basis des ESP32-Chips. Diese Entwicklungsplatinen enthalten einen ESP32-Chip und die dazugehörige Hardware, um die Entwicklung von Projekten auf Basis des ESP32 zu vereinfachen. In diesem Kapitel wollen wir uns die Eigenschaften einiger populärer ESP32-Entwicklungsboards ansehen.

### **1.3.1 SparkFun ESP32 Thing**

Auf diesem Entwicklungsboard (siehe Bild 1.3) gibt es einen FTDI-Chip, der USB in serielle Signale umwandelt, damit der Computer mit dem Board kommunizieren kann. Zur Unterstützung der Projektentwicklung gibt es LEDs und Tasten auf dem Board. Ein LiPo-Lade-Modul ist ebenfalls enthalten, so dass die Platine mit einer geeigneten Batterie versorgt werden kann. Das Board bietet 28 GPIO und 4 MB Flash-Speicher.

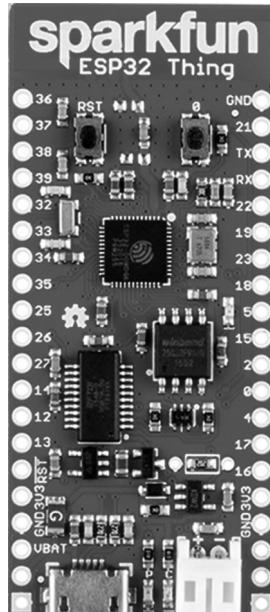


Bild 1.3 SparkFun ESP32 Thing

### 1.3.2 Geekcreit ESP32 Development Board

Dieses Entwicklungsboard in Bild 1.4 enthält alles, was der Einsatz eines ESP32-Prozessors in einem Projekt erfordert. Das Board wird über ein USB-Kabel an den PC angeschlossen und kostet nicht einmal 10 \$.



Bild 1.4 Geekcreit ESP32

### 1.3.3 LoLin32 ESP32 Development Board

Diese Entwicklungsplatine (Bild 1.5) ist mit 4 MB Flash-Speicher und einer Lithium-Batterie ausgestattet. 26 digitale I/Os und 12 analoge Eingänge sind vorhanden. Die Kommunikation mit dem PC erfolgt über die Mini-USB-Schnittstelle, über die das Board auch versorgt wird. Die Platine misst 5,8 cm x 2,54 cm.



Bild 1.5 LoLin32 ESP32

### 1.3.4 Pycom LoPy Development Board

Dieses Board in Bild 1.6 basiert auf dem ESP32-Chip und ist für die Programmiersprache MicroPython konfiguriert. Der Betrieb des Boards erfordert eine externe Antenne. Diese Entwicklungsplattform wurde hauptsächlich für IoT-Anwendungen entwickelt und enthält einen LoRa-Transceiver.



Bild 1.6 Pycom LoPy

### 1.3.5 ESP32 Test Board

Dieses Board (Bild 1.7) basiert auf ebenfalls dem ESP32-Chip. An allen seine I/O-Ports sind LEDs geschaltet.

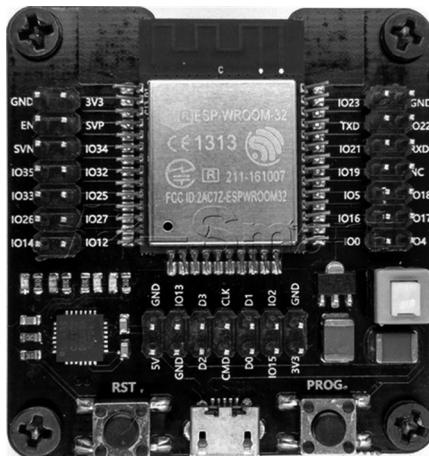


Bild 1.7 ESP32 Test Board

### 1.3.6 ESP32 Development Board von Pesky Products

Dies ist ein kleines Arduino-programmierbares ESP32-Board (Bild 1.8) mit einem 4 MB großen Flash-Speicher und einem USB-zu-seriell-Wandler. Auf der Platinen befinden sich auch ein LiPo-Batterielader und 26 GPIOs. Das Board ist Breadboard-kompatibel.

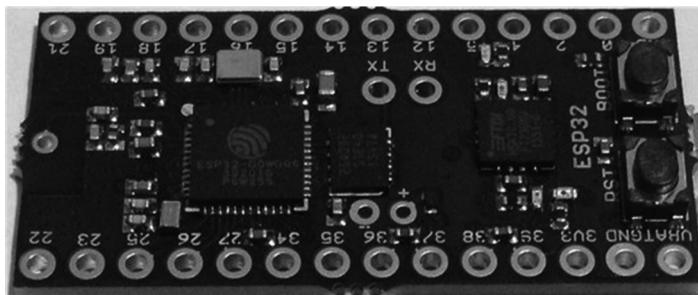


Bild 1.8 ESP32-Board von Pesky Products

### 1.3.7 ESP32 OLED Development Board

Dieses Board (Bild 1.9) enthält ein OLED-Display.



Bild 1.9 ESP32 OLED Board

### 1.3.8 MakerHawk ESP32 Development Board

Dies ist ein kleines Board (Bild 1.10), das nur 4 cm x 3 cm x 1 cm misst und den ESP32-Chip, eine Antenne und GPIOs besitzt. Für die PC-Schnittstelle steht eine Mini-USB-Buchse zur Verfügung.

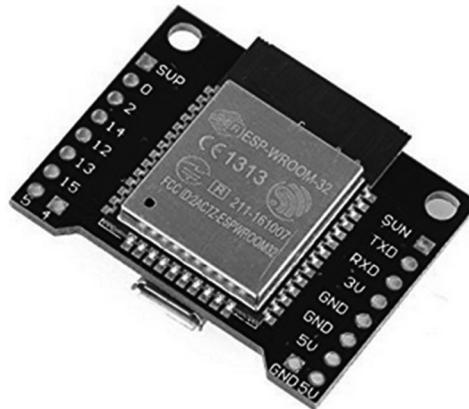


Bild 1.10 MakerHawk ESP32 Board

### 1.3.9 ESP32-DevKitC

Dies ist eines der beliebten ESP32-Entwicklungsboards (Bild 1.11), die von Espressif angeboten werden. Das Board ist Steckbrett-kompatibel und kann auch direkt in einem Projekt verwendet werden. Die Kommunikation mit dem PC erfolgt über eine Mini-USB-Buchse. Dieses Board wird in allen Projekten in diesem Buch verwendet. Deshalb wird es im nächsten Kapitel detailliert beschrieben.

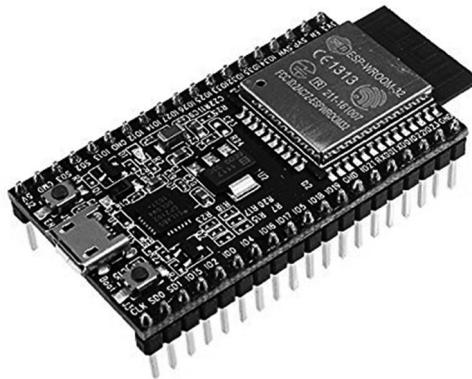


Bild 1.11 ESP32-DevKitC

### 1.3.10 Weitere Entwicklungsboards

Andere ESP32-Entwicklungsboards sind in Tabelle 1.2 zusammen mit dem Herstellern und der Webseite des Herstellers oder Distributors angegeben.

Development Board	Hersteller/Distributor	Webseite
HUZZAH32	Adafruit	<a href="https://adafruit.com">https://adafruit.com</a>
ESPea32	AprilBrother	<a href="https://www.aprbrother.com/en/index.htm">https://www.aprbrother.com/en/index.htm</a>
NodeMCU-32s	Ai-Thinker	<a href="http://en.ai-thinker.com/">http://en.ai-thinker.com/</a>
ESP32 Development Board	AnalogLamb	<a href="https://www.analoglamb.com/">https://www.analoglamb.com/</a>
Node32S	Ayarafun	<a href="http://www.ayarafun.com">www.ayarafun.com</a>
Node32S	LamLoei	<a href="https://github.com/lamloei/">https://github.com/lamloei/</a>
ESP32 Devkit	DOIT	<a href="http://en.doit.am/">http://en.doit.am/</a>
ESP32 Devkit	SmartArduino	<a href="http://www.smartarduino.com">www.smartarduino.com</a>
ESP32 WiFi/BLE	Elecrow	<a href="https://www.elecrow.com/">https://www.elecrow.com/</a>
Easy Kit ESP32	Geekworm	<a href="https://geekworm.aliexpress.com/store/1048722">https://geekworm.aliexpress.com/store/1048722</a>
ESP32 DevKit	MH-ET LIVE	<a href="http://mh.nodebb.com/">http://mh.nodebb.com/</a>
ESP32 MiniBoard	SunDUINO	<a href="http://www.sunduino.pl/">http://www.sunduino.pl/</a>
LoLin32	WEMOS	<a href="https://www.wemos.cc/">https://www.wemos.cc/</a>
ESP32 Development Board	iohippo	<a href="https://www.tindie.com/stores/ihippo/">https://www.tindie.com/stores/ihippo/</a>
ESP32 Breakout	Kilobyte	<a href="https://www.tindie.com/stores/kilabyte/">https://www.tindie.com/stores/kilabyte/</a>
ESP32 Breakout Kit	GNDTeknik	<a href="http://www.gndteknik.com/gndkits">http://www.gndteknik.com/gndkits</a>

Tabelle 1.3 Einige andere ESP32-Entwicklungsboards

## 1.4 Zusammenfassung

In diesem Kapitel haben wir kurz die Architektur des ESP32-Chips betrachtet. Zusätzlich wurden die Eigenschaften einiger beliebter ESP32-Entwicklungsboards zusammengefasst. Das ESP32-DevKitC wird im Folgenden als Entwicklungsboard verwendet. Im nächsten Kapitel sollen deshalb die Details der Hardware des ESP32-DevKitC detailliert untersucht und die Einsatzmöglichkeiten dieser Entwicklungsplatine in ESP32-basierten Projekten betrachtet werden.

## KAPITEL 2 • DAS ESP32-DEVKITC DEVELOPMENT BOARD

### 2.1 Übersicht

Im letzten Kapitel haben wir die Architektur des ESP32-Prozessors und seine grundlegenden Eigenschaften und Vorteile betrachtet. Wir haben auch kurz die verschiedenen ESP32-Entwicklungsboards kennengelernt, die es auf dem Markt gibt.

Derzeit ist das ESP32-DevKitC eines der beliebtesten Entwicklungsbretter auf ESP32-Basis. In diesem Buch sollen deshalb alle Projekte mit diesem Entwicklungsbrett durchgeführt werden. Es ist daher wichtig, dass wir den Aufbau und die Merkmale des Entwicklungsbretts ESP32-DevKitC im Detail kennenlernen.

### 2.2 ESP32-DevKitC Hardware

Das ESP32-DevKitC ist ein kleines, von Espressif entwickeltes ESP32-Prozessor-basiertes Board. Wie in Bild 2.1 zu sehen, ist die Platine mit den Abmessungen von 55 mm x 27,9 mm für die Aufnahme in ein Steckboard geeignet.

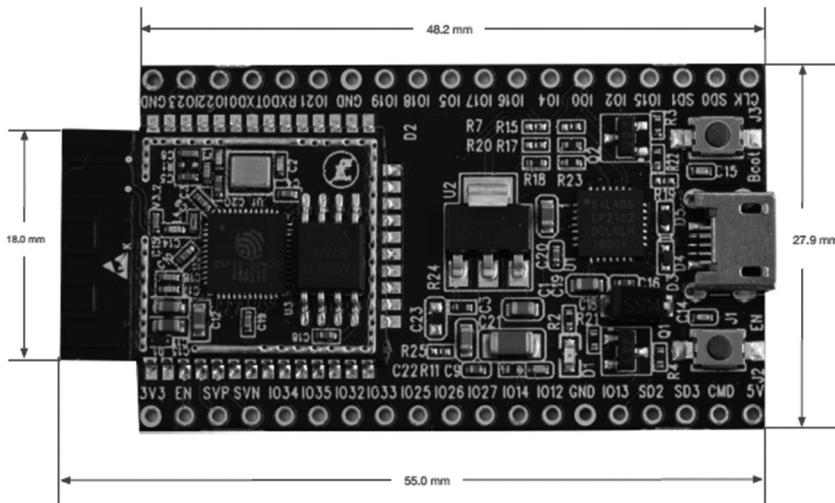


Bild 2.1 ESP32-DevKitC Entwicklungsplatine

Die Platine verfügt über je 19 Anschlüsse an jeder Seite für GPIO, Taktsignale und Spannungsversorgung. Wie in Bild 2.2 dargestellt, führen die beiden Steckverbinder folgende Signale:

Pin	Verbinder		Pin
	Links	Rechts	
1	+ 3,3V	GND	1
2	EN	IO23	2
3	SVP	IO22	3
4	SVN	TXD0	4

5	IO34	RXD0	5
6	IO35	IO21	6
7	IO32	GND	7
8	IO33	IO19	8
9	IO25	IO18	9
10	IO26	IO5	10
11	IO27	IO17	11
12	IO14	IO16	12
13	IO12	IO4	13
14	GND	IO0	14
15	IO13	IO2	15
16	SD2	IO15	16
17	SD3	SD1	17
18	CMD	SD0	18
19	+ 5V	CLK	19

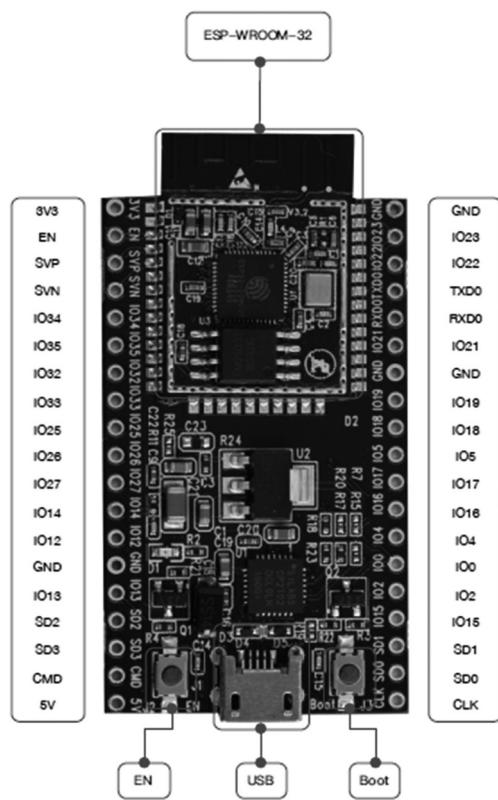


Bild 2.2 Anschlüsse des ESP32-DevKitC

Das Board wird über einen Mini-USB-Anschluss mit einem PC verbunden. Auch die Stromversorgung wird über den USB-Port vorgenommen. Die üblichen +5 V am USB-Port werden auf dem Board auf +3,3 V reduziert. Zusätzlich gibt es auf dem Board zwei Taster mit den Bezeichnungen EN und BOOT, die folgende Funktionen erfüllen:

**EN:** Dies ist die Reset-Taste. Das Board wird zurückgesetzt, wenn man diese Taste drückt.

**BOOT:** Dies ist die Download-Taste. Das Board befindet sich normalerweise im Betriebsmodus, wenn die Taste nicht gedrückt ist. Drückt und hält man diese Taste und drückt gleichzeitig auf die EN-Taste, wird der Firmware-Download-Modus gestartet, in dem Firmware über den seriellen USB-Port zum Prozessor heruntergeladen werden kann.

Die Pins auf dem ESP32-DevKitC-Board haben mehrere Funktionen, wie Bild 2.3 zeigt. So kann beispielsweise Pin 10, GPIO-Port 26, die Funktionen DAC-Kanal 2, ADC-Kanal 9, RTC-Kanal 7 und RX01 ausüben.

			3.3V																									
(pu)			RESET		EN												GND											
SVP		ADC0			GPIO36												GPIO23	VSPI MOSI								SPI MOSI		
SVN		ADC3			GPIO39												GPIO22									Wire SCL		
		ADC6			GPIO34												GPIO1	TX0								Serial TX		
		ADC7			GPIO35												GPIO3	RX0								Serial RX		
		TOUCH9	ADC4		GPIO32												GPIO21									Wire SDA		
		TOUCH8	ADC5		GPIO33												GPIO19	VSPI MISO								SPI MISO		
DAC1		ADC18			GPIO25												GPIO18	VSPI SCK								SPI SCK		
DAC2		ADC19			GPIO26												GPIO5	VSPI SS								(pu) SPI SS		
		TOUCH7	ADC17		GPIO27												GPIO17											
TMS	TDI	TOUCH6	ADC16	HSPI SCK	GPIO14												GPIO16											
(pd)		TOUCH5	ADC15	HSPI MISO	GPIO12												GPIO4		ADC10 TOUCH0								(pd)	
					GND												GPIO00	BOOT	ADC11 TOUCH1								(pu)	
		TCK	TOUCH4	ADC14	HSPI MOSI	GPIO13											GPIO2		ADC12 TOUCH2								(pd)	
					FLASH D2	GPIO9											GPIO15	HSPI SS	ADC13 TOUCH3	TDO	(pu)							
					FLASH D3	GPIO10											GPIO8		FLASH D1									
					FLASH CMD	GPIO11											GPIO7		FLASH D0									
					5V												GPIO6		FLASH SCK									

Bild 2.3 Ein Pin, mehrere Funktionen. Quelle: [www.cnx-software.co](http://www.cnx-software.co)

**Beachten Sie, dass GPIO34, GPIO35, GPIO36, GPIO37, GPIO38 und GPIO39 Ports ausschließlich Eingänge sind und nicht als Ausgangs-Ports verwendet werden können.**

Die Platine arbeitet mit einer Stromversorgung von typischen +3,3 V, obwohl das absolute Maximum mit +3,6 V angegeben ist. Es wird empfohlen, jedem Pin höchstens 6 mA zu entnehmen, obwohl der maximale Strom mit 12 mA angegeben ist. Es ist daher wichtig, Strombegrenzungswiderstände zu verwenden, wenn externe Lasten wie LEDs angeschlossen werden sollen.

Abhängig von der Konfiguration beträgt die HF-Leistungsaufnahme während des Empfangs etwa 80 mA, beim Senden kann sie auf über 200 mA steigen.

## 2.3 Einschalten des ESP32-DevKitC

Zur Programmierung des ESP32-DevKitC muss das Board über den Mini-USB-Port mit dem PC verbunden werden. Die Kommunikation zwischen Board und PC erfolgt über das übliche serielle Protokoll.

Das ESP32-DevKitC wird mit vorinstallierter Firmware ausgeliefert, so dass die Platine sofort getestet werden kann. Diese Firmware wird standardmäßig beim Einschalten der Stromversorgung aktiviert. Um mit dem Board zu kommunizieren, muss eine Terminal-Emulationssoftware auf dem PC installiert sein. Es gibt etliche kostenlose Terminal-Emulationsprogramme wie HyperTerm, PuTTY, X-CTU und so weiter. In diesem Buch werden wir PuTTY benutzen.

PuTTY ist ein populäres Terminal-Emulationsprogramm, das von der Webseite [www.putty.org](http://www.putty.org) heruntergeladen und auf dem PC installiert werden kann.

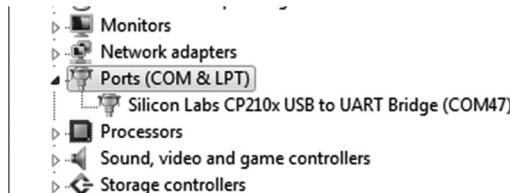


Bild 2.4 Geräte-Manager

Die Schritte zur Kommunikation mit der ESP32-DevKitC-Karte sind nachfolgend aufgeführt:

- Verbinden Sie Ihr ESP32-DevKitC mit einem der USB-Anschlüsse Ihres PCs. Auf der Platine sollte die rote LED aufleuchten und damit anzeigen, dass das Board mit Strom versorgt wird.
- Ermitteln Sie die serielle Port-Nummer, die dem USB-Port zugeordnet ist. Dies wird im **Geräte-Manager** wie in Bild 2.4 angezeigt. In diesem Beispiel ist die Port-Nummer COM47
- Starten Sie PuTTY durch einen Klick auf das PuTTY-Symbol
- Geben Sie in PuTTY folgende Kommunikationseinstellungen ein (siehe Bild 2.5):

Connection type: Serial

Host name: COM47

Speed: 115200

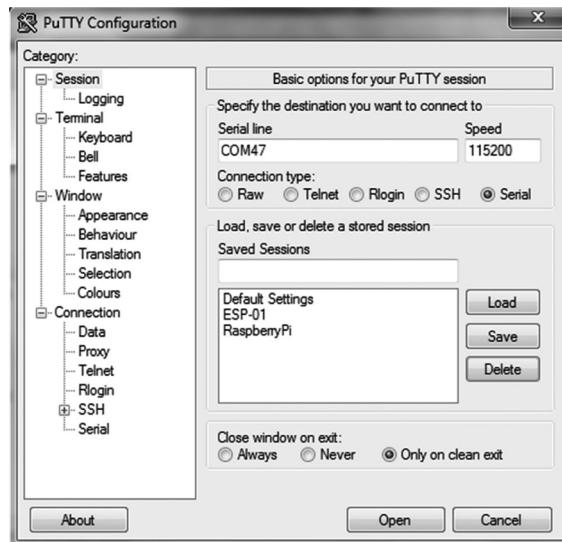


Bild 2.5 Putty-Kommunikationseinstellungen

Darüber hinaus können Sie PuTTY so konfigurieren, dass die Anzeige besser lesbar ist. Klicken Sie im linken Fensterbereich auf unter dem Punkt **Window** auf **Colours** und dann:

Klicken Sie auf **Default Foreground**

Klicken Sie auf **Modify** und wählen Sie schwarze Farbe

Klicken Sie auf **Default Background**

Klicken Sie auf **Modify** und wählen Sie weiße Farbe

Klicken Sie auf **Cursor Text**

Klicken Sie auf **Modify** und wählen Sie schwarze Farbe

Klicken Sie auf **Cursor Colour**

Klicken Sie auf **Modify** und wählen Sie schwarze Farbe

- Zusätzlich kann bei Bedarf die Schriftart/-größe eingestellt werden. Wählen Sie dafür im linken Fensterbereich unter **Window** den Punkt **Appearance**, klicken dann unter **Font settings** auf **Change** und wählen Sie zum Beispiel die Fontgröße 12 pt, bold.
- Klicken Sie im linken Fensterbereich auf **Session** und geben Sie Ihrer Sitzung einen Namen (zum Beispiel ESP32). Klicken Sie auf **Save**, um die Konfiguration zu speichern
- Klicken Sie auf **Open**, um die Terminal-Emulation zu starten

Sie sollten nun Ihr ESP32-DevKitC zurücksetzen, indem Sie die Reset-Taste unten links drücken. Das Board wird zurückgesetzt und sendet Nachrichten an Ihr Terminal, ähnlich wie in Bild 2.6 (hier ist nur ein Teil der Nachrichten abgebildet). Beachten Sie, dass die beiden Zeichen **:** > den Befehlsmodus einleiten.

```
:>ets Jun 8 2016 00:22:57
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun 8 2016 00:22:57

rst:0x10 (RTCWDT_RTC RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3ffc0000,len:0
load:0x3ffc0000,len:2304
load:0x40078000,len:3788
ho 0 tail 12 room 4
load:0x40098000,len:532
entry 0x4009813c

*****
*      hello espressif ESP32!
*      2nd boot is running!
*      version (V0.1)
*****
*****
```

Bild 2.6 Meldungen nach dem Reset des ESP32-DevKitC.

Wenn Sie auf Ihrem Display nichts sehen, sollten Sie die Verbindung zwischen ESP32-DevKitC und PC überprüfen. Stellen Sie außerdem sicher, dass die Port-Nummer und die Kommunikationsgeschwindigkeit in PuTTY korrekt gewählt sind.

Sie sollten mit der Firmware des ESP32-DevKitC kommunizieren können. Die am häufigsten verwendeten Firmware-Test-Befehle (die sogenannten SSC-Befehle) werden im folgenden Abschnitt erläutert. Die SSC-Befehle in den folgenden Abschnitten wurden der Espressif-Schrift **ESP32 SERIES Getting Started Guide for SDK based on FreeRTOS** (Version 1.0, 2015). entnommen.

### 2.3.1 help

Dieser Befehl zeigt eine Meldung mit einer Liste der SCS-Befehle an.

```
:> help
:>
:> supported command:
:>
:>
:> Please refer to document ssc_commands.xlsx for detail :>
```

### 2.3.2 op

Mit diesem Befehl wird der WLAN-Modus des Systems eingestellt und abgefragt. Das allgemeine Format des Befehls lautet:

op -S -o wmode

oder

op -Q

Folgende Parameter werden unterstützt:

-Q: Abfrage WLAN-Modus

-S: setzt den WLAN-Modus mit 1 = STA-Modus, 2 = AP-Modus, 3 = STA+AP-Modus

Bild 2.7 zeigt einige Beispiele.

```
:>op -Q
+CURMODE:1
+MODE:OK
:>op -s -o 1
+MODE:OK
:>■
```

*Bild 2.7 Verwendung des Befehls op*

### 2.3.3 sta

Dieser Befehl wird verwendet, um das STA-Netzwerk zu scannen, den AP anzuschließen oder zu trennen und den Status der STA-Netzwerkschnittstelle abzufragen. Das allgemeine Format des Befehls lautet:

```
sta -Q
sta -S -ssid -p password
sta -D
sta -S scan
```

wobei ssid und password der Name und das Passwort des zu verbindenden APs sind und

- Q den Status der Netzwerkverbindung zeigt
- D vom aktiven AP trennt
- S scan APs scannt

Ein Beispiel ist in Bild 2.8 zu sehen, wobei das ESP32-DevKitC mit einem Home-AP mit dem Namen **BTHomeSpot-XNH** und dem Passwort **49345abaeb** verbunden ist. Der Status wird nach der erfolgreichen Verbindung mit dem Befehl **sta -Q** überprüft.

```
:>sta -S -s BTHomeSpot-XNH -p 49345abaeb
+SCAN_DONE:OK!
:>
+SCAN:BTHomeSpot-XNH,00:07:26:c7:ac:24,4,6,-60,0,0
+SCANDONE
:>sta -Q
+JAP:CONNECTED,BTHomeSpot-XNH
:>■
```

*Bild 2.8 Anschluss an einen Home-Access Point*

Im Beispiel von Bild 2.9 werden die Access Points gescannt.

```
:>sta -S scan
+SCAN_DONE:OK!
:>
+SCAN:Chromecast7872.b,fa:8f:ca:56:8d:3f,0,6,-44,0,0
+SCAN:BTHomeSpot-XNH,00:07:26:c7:ac:20,4,11,-44,0,0
+SCAN:The Hotal,e0:b9:e5:53:d6:2d,4,1,-50,0,0
+SCAN:BTWifi-with-FON,c2:91:f9:66:5a:aa,0,6,-50,0,1
+SCAN:BTWifi-X,e2:91:f9:66:5a:aa,0,6,-52,0,1
+SCAN:HP-Print-19-LaserJet 200 color,9c:d2:1e:ae:86:19,0,6,-55,0,0
```

Bild 2.9 Scannen der Access Points

#### 2.3.4 mac

Dieser Befehl gibt die MAC-Adresse des ESP32-DevKitC zurück. Bild 2.10 zeigt ein Beispiel:

```
:>mac
+STAMAC:30:ae:a4:05:5b:e0
:>
```

Bild 2.10 Anzeige der MAC-Adresse

#### 2.3.5 dhcp

Dieser Befehl aktiviert oder deaktiviert oder fragt den Status des DHCP-Servers oder Clients ab.

Das Format des Befehls lautet:

```
dhcp -S -o mode
dhcp -E -o mode
dhcp -Q
```

Dabei startet -S den DHCP-Server, -E beendet ihn und -Q zeigt seinen Status an. Der Parameter „mode“ kann folgende Werte haben:

- 1: DHCP-Client der STA-Schnittstelle
- 2: DHCP-Server der STA-Schnittstelle
- 3: DHCP-Client und Server der STA-Schnittstelle

#### 2.3.6 reboot

Dieser Befehl setzt den ESP32-DevKitC zurück und startet es neu

#### 2.3.7 ram

Dieser Befehl gibt die Größe des freien dynamischen Speichers (heap) im System zurück. Ein Beispiel ist in Bild 2.11 zu sehen.

```
:>ram
+FREEHEAP:162632
:>
```

*Bild 2.11 Anzeige des freien Heap-Speichers*

### 2.3.8 ip

Mit diesem Befehl wird die dem ESP32-DevKitC zugewiesene IP-Adresse zugeordnet oder abgefragt. Das Format des Befehls lautet:

```
ip -Q [-o mode]
ip -S [-i ip] [-o mode] [-m mask] [-g gateway]
```

wobei

-Q die IP-Adresse, die Subnetzmaske und die Gateway-Adresse zurückgibt  
 -S die IP-Adresse (-i), die Maske (-m) und das Gateway (-g) setzt

mode ist 1 für STA, 2 für den AP und 3 für STA und AP

Im Bild 2.12 sind die IP-Adresse des ESP32-DevKitC und des Gateways auf 192.168.1.156 beziehungsweise 192.168.1.254, die Subnetzmaske auf 255.255.255.0 eingestellt.

```
:>ip -Q
+STAIP:192.168.1.156
+STAIPMASK:255.255.255.0
+STAIPGW:192.168.1.254
:>
```

*Bild 2.12 Abfrage der IP-Adresse*

### 2.3.9 ap

Mit diesem Befehl werden die Parameter des AP-Netzwerks-Interfaces festgelegt. Das Format des Befehls lautet:

```
ap -S [-s ssid] [-p password] [-t encrypt] [-n channel] [-h] [-m max]
ap -Q
ap -L
```

Dabei sind ssid der AP-Name, password das Passwort und encrypt der Verschlüsselungsmodus. -h versteckt die ssid, max legt die maximalen AP-Verbindungen fest, -Q zeigt die AP-Parameter, -L die MAC- und IP-Adresse der angeschlossenen Station an.

## 2.4 Zusammenfassung

In diesem Kapitel haben wir die grundlegenden Eigenschaften der Hardware-Erweiterungsplatine ESP32-DevKitC kennengelernt. Darüber hinaus wurden die SSC-Befehle des

ESP32-DevKitC behandelt, mit denen man das Board einstellen kann, sobald Spannung angelegt wird.

Im nächsten Kapitel werden wir lernen, wie man den ESP32-Prozessor programmiert und den ausführbaren Code zum Prozessor hochlädt.

## KAPITEL 3 • ARDUINO-IDE FÜR DAS ESP32-DEVKITC

### 3.1 Übersicht

Wenn man das ESP32-DevKitC erwirbt, gehört normalerweise keine Programmier-Software dazu. Es ist daher notwendig, eine Programmier-Software auf dem PC zu installieren, mit der der Benutzer Programme entwickeln und zum Prozessor hochladen kann. Genau wie der ESP8266 ist der ESP32-Prozessor mit verschiedenen Programmiersprachen wie C, MicroPython oder ähnlichen kompatibel.

Die Arduino-IDE ist eine der am häufigsten verwendeten Entwicklungsumgebungen für Mikrocontroller, vor allem für die Arduino-Familie. Diese IDE ist einfach zu bedienen, unterstützt viele Mikrocontroller und enthält sehr umfangreiche Bibliotheken von Funktionen, was die Programmierung enorm erleichtert. Die meisten Ingenieurstudenten der Elektrotechnik und viele Maker mit dem Hobby Elektronik kennen die Arduino-IDE. In diesem Abschnitt werden wir uns damit beschäftigen, wie man den ESP32-Prozessor in die Arduino-IDE auf einem Windows-PC aufnimmt.

In den nächsten Kapiteln finden Sie mehrere einfache bis mittelschwere Projekte mit dem ESP32-DevKitC-Board, die in der Arduino-IDE erstellt werden. Es ist wichtig zu wissen, dass die Arduino-IDE einige Beschränkungen aufweist, so dass nicht alle möglichen Funktionen des ESP32 mit dieser IDE programmiert werden können. Erst in den weiteren Kapiteln lernen wir das von Espressif entwickelte Framework für die ESP32-Prozessoren, die Entwicklungsumgebung ESP-IDF kennen, wie sie installiert wird und wie man damit umgeht. Obwohl die Programmierung mit der ESP-IDF komplexer ist als die mit der Arduino-IDE, hat sie den Vorteil, dass sie dem Programmierer Zugriff auf alle Funktionen des ESP32-Prozessors gewährt.

### 3.2 Installation der Arduino-IDE für das ESP32-DevKitC

Zunächst aber kümmern wir uns in diesem Abschnitt um die Unterstützung des ESP32 durch die Arduino-IDE auf einem Windows-PC.

- Laden Sie Python 2.7 von der Webseite <https://www.python.org/downloads/> (siehe Bild 3.1 und Bild 3.2) herunter und installieren Sie die Programmiersprache in dem Ordner **C:\Python27** auf Ihrem PC.



Bild 3.1 Installation von Python 2.7



Bild 3.2 Python 2.7 wird installiert

- Machen Sie Python von überall auf dem PC zugänglich, indem Sie die Umgebungsvariable PATH konfigurieren. Klicken Sie dazu auf **Systemsteuerung** -> **Alle Systemsteuerungselemente** -> **System** und dann auf **Erweiterte Systemeinstellungen**, wie in Bild 3.3 gezeigt.

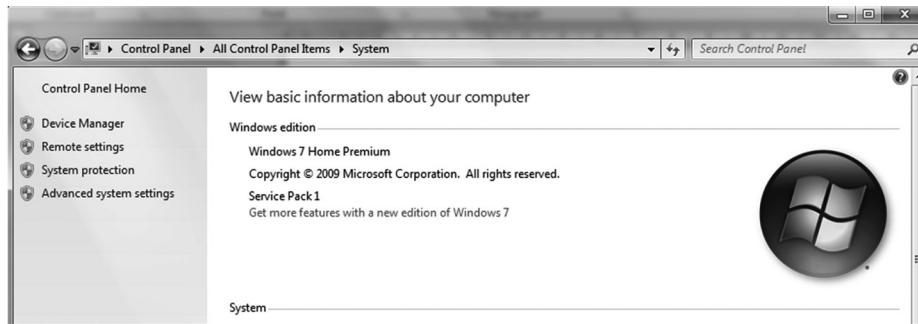


Bild 3.3 Klicken Sie auf „Erweiterte Systemeinstellungen“

- Klicken Sie auf **Erweitert** -> **Umgebungsvariablen**, wie in Bild 3.4 gezeigt

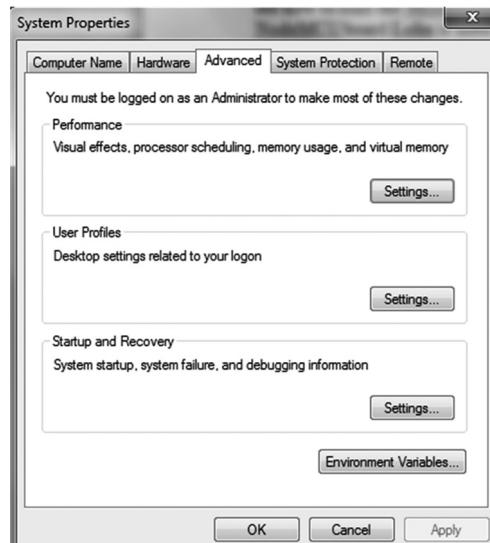


Bild 3.4 Klicken Sie auf „Umgebungsvariablen“

- Klicken Sie unter **Systemvariablen** auf **Path**, wie in Bild 3.5 gezeigt.

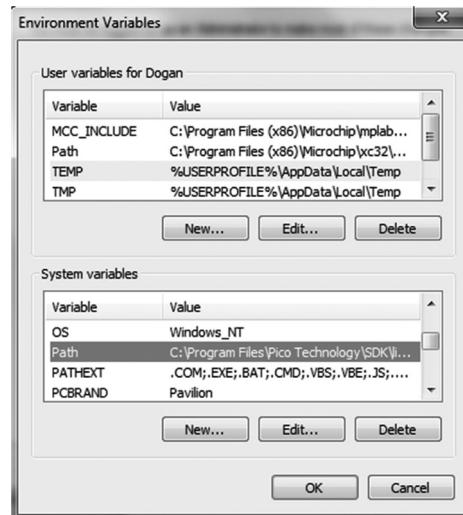


Bild 3.5 Klicken Sie unter Systemvariablen auf „Path“

- Klicken Sie auf **Bearbeiten** und fügen Sie folgende Anweisung am Ende der Zeile **Wert der Variablen** hinzu: ;**C:\Python27**, wie in Bild 3.6 gezeigt. Klicken Sie abschließend **OK**, um die Umgebungseinstellungen zu speichern und zu verlassen.

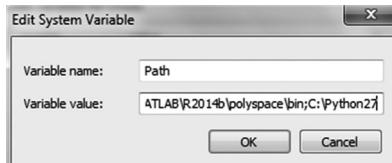


Bild 3.6 ;Python27 hinzufügen

- Python sollte jetzt von überall auf Ihrem PC zugänglich sein. Klicken Sie auf den **Startknopf** und geben Sie in die Befehlszeile **cmd** ein. Geben Sie **Python** ein, um den Python-Interpreter zu starten. Sie sollten ein Fenster ähnlich dem in Bild 3.7 gezeigten sehen.

```
Administrator: C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\Dogan>python
Python 2.7.13 (v2.7.13:a06454b1afaf, Dec 17 2016, 20:42:59) IMSC v.1500 32 bit
Intel>I on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Bild 3.7 Starten Sie Python auf Ihrem PC

- Geben Sie die Anweisung **print („Hello there“)** ein. Es erscheint die Zeichenfolge **Hello there**. Verlassen Sie Python, indem Sie **Strg-Z** eingeben, gefolgt von Enter.

- Dies zeigt, dass Python erfolgreich auf Ihrem PC installiert wurde. Als Nächstes müssen den ESP32 der Arduino-IDE vorstellen.
- Der nächste Schritt ist es, **Git** zu installieren. Dies ist zwar nicht unbedingt erforderlich, macht aber den Download und den Installationsvorgang einfacher. Laden Sie zunächst Git von <https://git-scm.com/download/win> und installieren Sie es mit den voreingestellten Optionen. Achten Sie darauf, dass Sie die richtige Version für Ihren Computer installieren, siehe Bild 3.8!

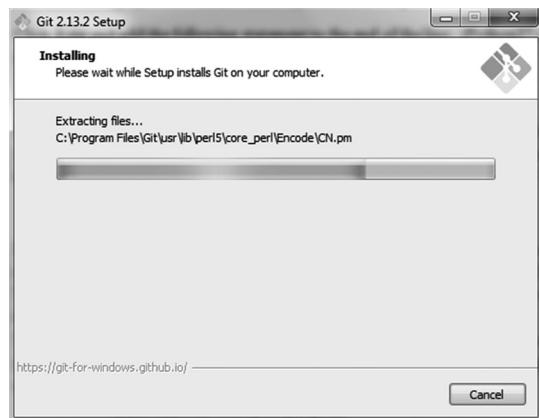


Bild 3.8 Git wird installiert

- Starten Sie **Git GUI** und wählen Sie die Option **Clone Existing Repository**, wie in Bild 3.9 gezeigt.

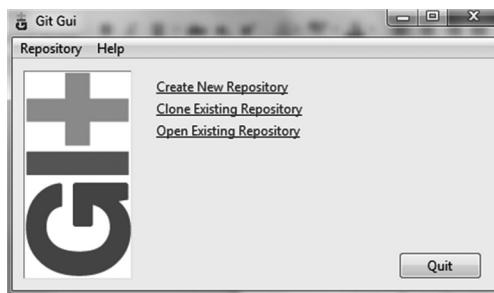


Bild 3.9 Wählen Sie die Option „Existing Repository“

- Tragen Sie die Namen von Quell- und Zielordner ein. Der Zielordner sollte, wie in Bild 3.10 zu sehen, mit dem Pfad zu Ihrer Arduino-IDE beginnen (Standard ist C:\Program Files (x86)\Arduino).

Source Location: <https://github.com/espressif/arduino-esp32.git>

Target Directory: **C:\Program Files (x86)\Arduino\hardware\espressif\esp32**

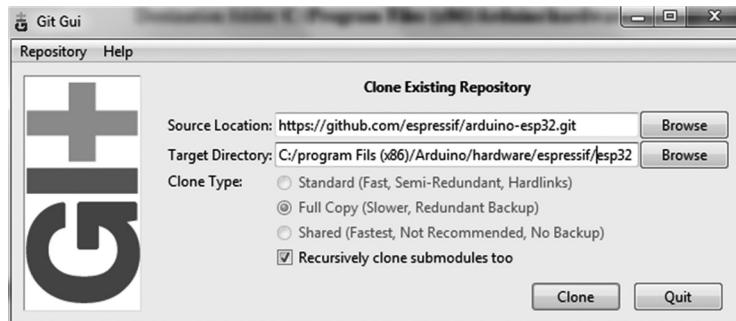


Bild 3.10 Wählen Sie Quell- und Zielordner aus

- Klicken Sie auf die Schaltfläche **Clone** und warten Sie, bis die Installation beendet ist.
- Schließen Sie **Git GUI**.
- Öffnen Sie den Ordner **C:\Program Files (x86)\Arduino\Hardware\espressif\esp32\tools**, wie in Bild 3.11 gezeigt.

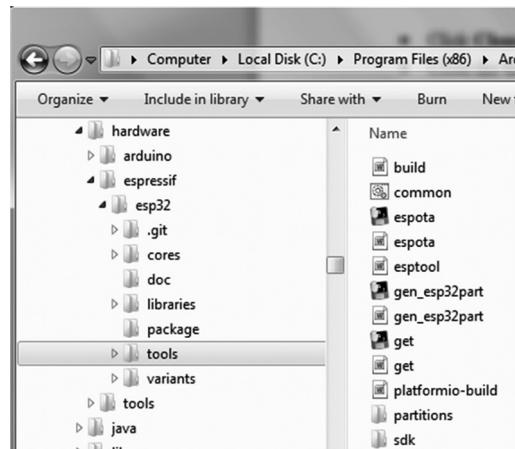


Bild 3.11 Öffnen des Ordners ... \tools

- Doppelklicken Sie auf die Anwendung **get.exe** und warten Sie, bis Download und Entpacken abgeschlossen sind (Bild 3.12).

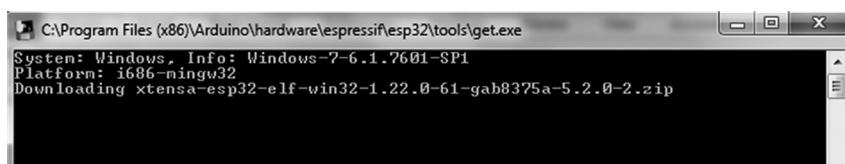


Bild 3.12 Download und Extrahieren

- Wir sollten nun die Installation testen, um sicherzustellen, dass alle benötigten Dateien geladen wurden. Wir verwenden eine der mitgelieferten Beispielanwendungen, um sicherzustellen, dass unser Programm auf den ESP32-Prozessor hochgeladen wird und korrekt funktioniert
- Schließen Sie das ESP32-DevKitC an Ihren PC an und starten Sie die Arduino-IDE.
- Wählen Sie **Werkzeuge -> Board -> ESP32 Dev Module**, wie in Bild 3.13 gezeigt.

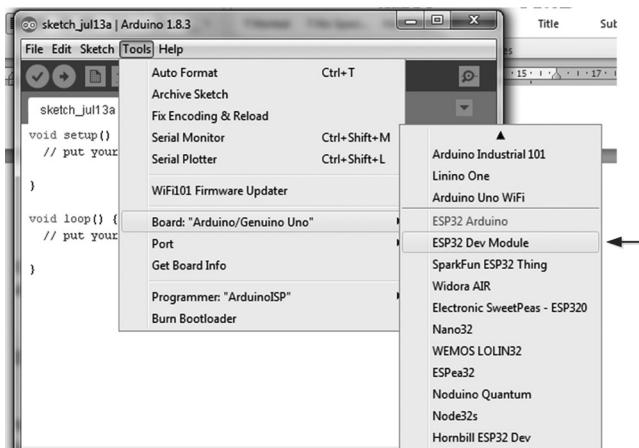


Bild 3.13 Wählen Sie das **ESP32 Dev Module** aus

- Wählen Sie den seriellen Port aus. Im Beispiel ist dies COM47, wie in Bild 3.14 zu sehen.

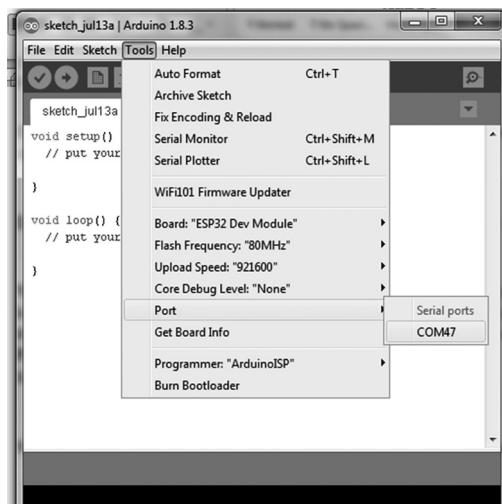


Bild 3.14 Wählen Sie den seriellen Port

- Öffnen Sie das Beispielprogramm in **Datei** -> **Beispiele** -> **WiFi** (unter ESP32) -> **WiFiScan**, wie in Bild 3.15 gezeigt.



Bild 3.15 Öffnen Sie das Beispielprogramm WiFi Scan

- Sie sollten einen Sketch wie in Bild 3.16 sehen.

```

// This sketch demonstrates how to scan WiFi networks.
// The API is almost the same as with the WiFi Shield library,
// the most obvious difference being the different file you need
*/
#include "WiFi.h"

void setup()
{
    Serial.begin(115200);

    // Set WiFi to station mode and disconnect from an AP if it was connected
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);

    Serial.println("Setup done");
}

```

Bild 3.16 Beispielprogramm zum Testen der Installation

- Jetzt wollen wir das Programm auf das ESP32-DevKitC laden, das dazu zunächst in den Firmware-Upload-Modus gesetzt werden muss. Halten Sie die **BOOT**-Taste auf der rechten Seite gedrückt und drücken Sie die **EN**-Taste auf der linken

Seite, um das Board neu zu starten. Wählen Sie in der Arduino-IDE **Sketch -> Hochladen**, um das Programm zu kompilieren und auf das ESP32-DevKitC zu spielen. Ist dies erfolgreich abgearbeitet, erscheint unten in der Arduino-IDE eine Meldung wie in Bild 3.17. Erst jetzt dürfen Sie die **BOOT**-Taste wieder loslassen.

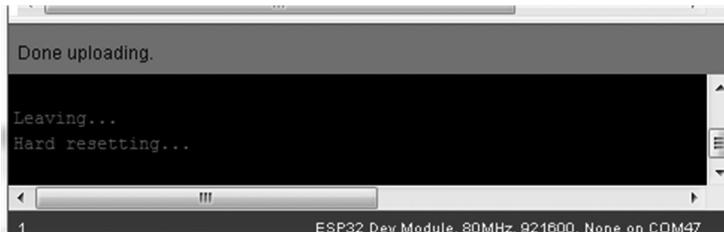


Bild 3.17 Ende des Uploads

- Öffnen Sie nun den Arduino-IDE-Monitor, indem Sie auf **Werkzeuge -> Serieller Monitor** klicken. Es sollte eine Liste der Access Points in Ihrer Nähe zu sehen sein (Bild 3.18). Die Liste wird alle fünf Sekunden aktualisiert.

 A screenshot of the Arduino IDE's Serial Monitor window titled "COM47". The window displays a list of 12 nearby access points. The output is as follows:
 

```

scan start
scan done
12 networks found
1: BTHomeSpot-XNH (-56)*
2: Chromecast7872.b (-65)
3: BTWifi-with-FON (-69)
4: BTWifi-X (-69)*
5: BTHomeSpot-XNH (-77)*
6: BTHub6-326G (-79)*
7: BTWifi-with-FON (-79)
8: BTWifi-X (-79)*
9: BTHub3-MFK8 (-81)*
10: BTWiFi (-86)
11: BTOpenzone-B (-86)
12: TALKTALK2C9964 (-86)*
  
```

Bild 3.18 Liste von Access Points in der Nähe

- Wenn alles erfolgreich verlaufen ist, können wir sicher sein, dass der ESP32-Prozessor in der Arduino-IDE korrekt integriert ist.

## KAPITEL 4 • BASISPROJEKTE MIT DER ARDUINO-IDE UND DEM ESP32-DEVKITC

### 4.1 Übersicht

Im letzten Kapitel haben wir gesehen, wie man die Arduino-IDE so einrichtet, dass sie die in der IDE entwickelten Programme kompiliert und zum ESP32-DevKitC Development Board hochlädt.

In diesem Kapitel werden wir grundlegende Projekte mit der Arduino-IDE als Entwicklungs-Software entwickeln, diese Projekte dann kompilieren und zum ESP32-DevKitC hochladen. Alle in diesem Buch enthaltenen Projekte wurden getestet und funktionieren. Jedes Projekt ist wie folgt gegliedert:

- Projekttitle
- Projektbeschreibung
- Ziel des Projekts
- Projekt-Blockschaltbild
- Projektschaltbild
- Bau des Projekts
- Betrieb des Projekts (PDL)
- Programmlisting
- Beschreibung des Programms
- Zusätzliche Arbeit (optional)

Der Ablauf der Projektprogramme wird anhand der Program Design Language (PDL) beschrieben. Dabei handelt es sich um eine Methode zum Entwurf und zur Dokumentation von Software-Methoden und -Prozeduren. PDL ist ähnlich wie Pseudocode und beschreibt ein Programm mit verständlichen Schlüsselwörtern wie BEGIN, END, IF, THEN, ELSE, WHILE, REPEAT, DO, DO FOREVER, verzichtet dabei aber auf sprachspezifische Eigenheiten. Die Schlüsselwörter sind in den PDLS fett gedruckt.

### 4.2 PROJEKT 1 – Blinkende LED

#### 4.2.1 Beschreibung

In diesem Projekt wird eine LED an den GPIO-Port 23 des ESP32-DevKitC angeschlossen, die im Sekundentakt blinken soll.

#### 4.2.2 Das Ziel

Ziel dieses Projekts ist es, zu zeigen, wie eine LED an einen GPIO-Port angeschlossen und wie sie ein- und ausgeschaltet werden kann.

#### 4.2.3 Blockschaltbild

Bild 4.1 zeigt das Blockschaltbild des Projekts.



Bild 4.1 Blockschaltbild des Projekts

#### 4.2.4 Schaltplan

Bild 4.2 zeigt die Pin-Belegung an der rechten oberen Seite des ESP32-DevKitC. Der Schaltplan des Projekts ist in Bild 4.3 dargestellt. Die LED ist über einen Strombegrenzungswiderstand mit dem GPIO-Port 23 (zweiter Pin oben rechts) verbunden. Geht man von einem Spannungsabfall über der LED von 2 V und einem LED-Strom von 4 mA aus, liegt der erforderliche Wert des Strombegrenzungswiderstandes R bei:

$$R = (3,3 \text{ V} - 2 \text{ V}) / 4 \text{ mA} = 325 \Omega.$$

Als nächstliegender Widerstand wird 330  $\Omega$  gewählt.

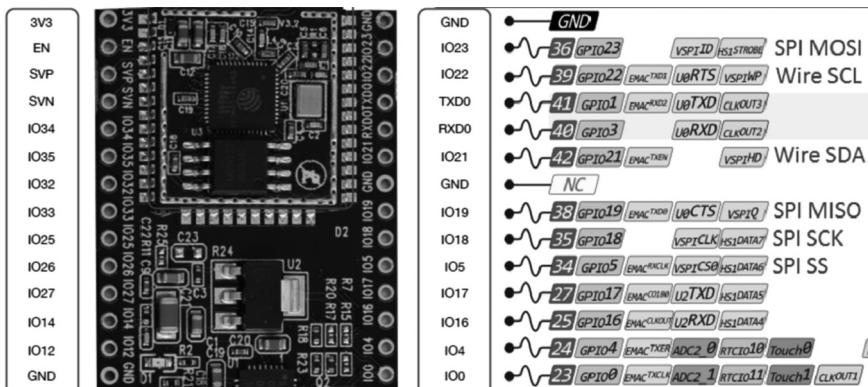


Bild 4.2 Die Anschlussbelegung an der rechten oberen Seite des ESP32-DevKitC

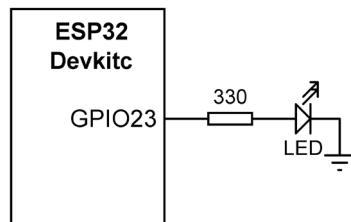


Bild 4.3 Schaltplan des Projekts

#### 4.2.5 Aufbau

Die ESP32-DevKitC-Platine wird auf einem Steckboard montiert (Bild 4.4) und die LED über den Strombegrenzungswiderstand mit der Platine verbunden.

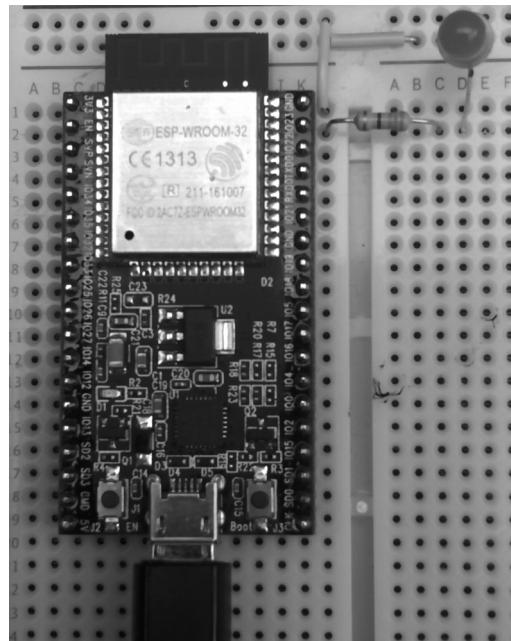


Bild 4.4 Projekt auf einem Steckbrett

#### 4.2.6 PDL des Projekts

Die PDL des Projekts ist in Bild 4.5 dargestellt. Das Programm läuft in einer Endlosschleife, die LED blinkt eine Sekunde lang auf und bleibt für eine weitere Sekunden dunkel.

```
BEGIN
    Assign LED to port pin GPIO23
    Configure port pin GPIO23 as output
    DO FOREVER
        Set LED HIGH
        Wait 1 second
        Set LED LOW
        Wait 1 second
    ENDDO
END
```

Bild 4.5 PDL des Projekts

#### 4.2.7 Programmlisting

Die Programmlisting FlashLED des Projekts ist sehr einfach, wie Bild 4.6 zeigt.

```
*****
*                               FLASHING LED
*                               =====
*
* In this program an LED is connected to port GPIO23 of the
```

```

* ESP32-DevKitC. The program flashes the LED every second
*
* Program: FlashLED
* Date : July, 2017
*****
```

```

#define LED 23

void setup()
{
    pinMode(LED, OUTPUT);
}

void loop()
{
    digitalWrite(LED, HIGH);
    delay(1000);
    digitalWrite(LED, LOW);
    delay(1000);
}
```

*Bild 4.6 Listing des Programms FlashLED*

#### **4.2.8 Programmbeschreibung**

Zu Beginn des Programms wird der Variablenname LED dem GPIO-Port 23 zugeordnet und dieser Port LED sofort als Ausgang konfiguriert. Das Hauptprogramm wird kontinuierlich in einer Schleife ausgeführt, wobei in dieser Schleife die LED eingeschaltet (HIGH) und ausgeschaltet (LOW) wird, jeweils mit einer Verzögerung.

Nur zur Erinnerung: Sie müssen die BOOT-Taste auf dem ESP32-DevKitC drücken und halten, während das Programm kompiliert und hochgeladen wird.

#### **4.2.9 Vorschläge**

Ändern Sie das Programm in Bild 4.6, so dass die LED-Einschaltzeit 2 s und die Ausschaltzeit 5 s beträgt.

### **4.3 PROJEKT 2 – Leuchtturm-LED**

#### **4.3.1 Beschreibung**

In diesem Projekt ist eine LED wie in Projekt 1 an den GPIO-Port 23 des ESP32-DevKitC angeschlossen. Die LED soll zweimal in einer Sekunde schnell aufblitzen. Die "Blitzzeit" soll dabei 200 ms betragen. Diese Art des Blinkens wird bei einem Leuchtturm **GpFl(2)** genannt. Die erforderliche Blitzsequenz kann wie folgt beschrieben werden:

LED EIN  
200 ms warten  
LED AUS

```
100 ms warten
LED EIN
200 ms warten
LED AUS
100 ms warten
400 ms warten
```

#### 4.3.2 Das Ziel

Ziel dieses Projekts ist es, zu zeigen, wie eine LED an einen GPIO-Port angeschlossen und wie sie in unterschiedlichen Sequenzen ein- und ausgeschaltet werden kann.

#### 4.3.3 Blockschaltbild

Das Blockschaltbild des Projekts ist in Bild 4.1 dargestellt.

#### 4.3.4 Schaltplan

Der Schaltplan des Projekts ist in Bild 4.3 zu sehen.

#### 4.3.5 Aufbau

Die LED wird wie in Bild 4.4 über den Strombegrenzungswiderstand an die ESP32-Dev-KitC-Platine angeschlossen.

#### 4.3.6 PDL des Projekts

Die PDL des Projekts ist in Bild 4.7 dargestellt. In der Endlosschleife des Programms blitzt die LED jede Sekunde mit der Leuchtturmfolge **GpFI(2)**.

```
BEGIN
    Assign LED to port pin GPIO23
    Configure port pin GPIO23 as output
    DO FOREVER
        Set LED HIGH
        Wait 200 ms
        Set LED LOW
        Wait 100 ms
        Set LED HIGH
        Wait 200 ms
        Set LED LOW
        Wait 100 ms
        Wait 400 ms
    ENDDO
END
```

*Bild 4.7 PDL des Projekts*

#### 4.3.7 Programmlisting

Die Programmlisting Lighthouse des Projekts ist sehr einfach und wird in Bild 4.8 gezeigt.

```

/*
 *          LIGHHOUSE FLASHING LED
 * =====
 *
 * In this program an LED is connected to port GPIO23 of the
 * ESP32-DevKitC. The program flashes the LED with the lighthouse
 * lighting sequence GpFl(2)
 *
 * Program: Lighthouse
 * Date : July, 2017
 */
*****
```

```

#define LED 23
#define ON HIGH
#define OFF LOW

void setup()
{
    pinMode(LED, OUTPUT);
}

void loop()
{
    digitalWrite(LED, ON);
    delay(200);
    digitalWrite(LED, OFF);
    delay(100);
    digitalWrite(LED, ON);
    delay(200);
    digitalWrite(LED, OFF);
    delay(100);
    delay(400);
}
```

*Bild 4.8 Programmlisting*

### 4.3.8 Programmbeschreibung

Am Anfang des Programms wird genau wie im Projekt 1 der Variablenname LED dem GPIO-Port 23 zugeordnet. Dann wird der LED-Port als Ausgang konfiguriert. Das Hauptprogramm läuft kontinuierlich in einer Schleife, in der die LED eingeschaltet (HIGH) und ausgeschaltet (LOW) wird, wie es die Leuchtturm-Sequenz **GpFl(2)** fordert.

## 4.4 PROJEKT 3 – Abwechselnd blinkende LEDs

### 4.4.1 Beschreibung

In diesem Projekt sind zwei LEDs an den GPIO-Ports 22 und 23 des ESP32-DevKitC angegeschlossen.

#### 4.4.2 Das Ziel

Dieses Projekts soll zeigen, wie LEDs an einen GPIO-Port angeschlossen und wie sie abwechselnd ein- und ausgeschaltet werden können.

#### 4.4.3 Blockschaltbild

Das Blockschaltbild des Projekts ist in Bild 4.9 zu sehen. LED1 ist an GPIO-Port 22, LED2 an GPIO-Port 23 angeschlossen.

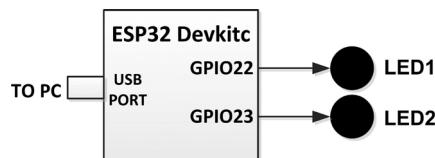


Bild 4.9 Blockschaltbild des Projekts

#### 4.4.4 Schaltplan

Der Schaltplan des Projekts in Bild 4.10 zeigt, dass die LEDs über 330- $\Omega$ -Strombegrenzungswiderstände mit den Port-Anschlüssen verbunden sind. Beachten Sie Bild 4.2 für Pin-Konfiguration des ESP32-DevKitC!

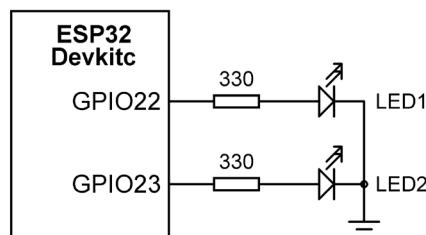
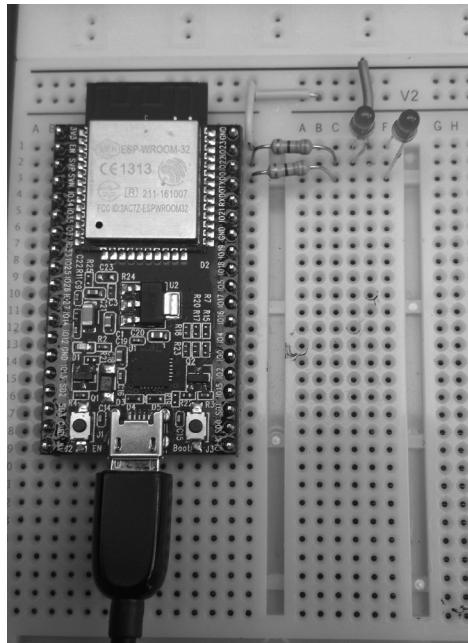


Bild 4.10 Schaltplan des Projekts

#### 4.4.5 Aufbau

Zum Aufbau des Projekts wird ein Steckboard verwendet. Die LEDs sind über die Strombegrenzungswiderstände mit dem ESP32 Dev-KitC verbunden, wie Bild 4.11 zeigt.

*Bild 4.11 Projekt auf einem Steckboard*

#### 4.4.6 PDL des Projekts

Die PDL des Projekts ist in Bild 4.12 zu sehen. Das Programm läuft in einer Endlosschleife, in der die LEDs abwechselnd jede Sekunde ein- und ausgeschaltet werden.

```

BEGIN
    Assign LED1 to port pin GPIO22
    Assign LED2 to port pin GPIO23
    Configure port pins GPIO22 and GPIO23 as outputs
DO FOREVER
    Set LED1 HIGH
    Set LED2 LOW
    Wait 1 second
    Set LED1 LOW
    Set LED2 HIGH
    Wait 1 second
ENDDO
END

```

*Bild 4.12 PDL des Projekts*

#### 4.4.7 Programmlisting

Das Programm Altflash ist sehr einfach aufgebaut, wie Bild 4.13 beweist.

```
/*****
 *          ALTERNATE FLASHING LEDs
 *=====
 *
 * In this program two LEDs are connected to port GPIO22 and
 * GPIO23 of the ESP32-DevKitC. The program flashes the LEDs
 * alternately with one second delay between each output
 *
 * Program: Altflash
 * Date : July, 2017
 *****/
#define LED1 22
#define LED2 23
#define ON HIGH
#define OFF LOW

void setup()
{
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
}

void loop()
{
    digitalWrite(LED1, HIGH);
    digitalWrite(LED2, LOW);
    delay(1000);
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, HIGH);
    delay(1000);
}
```

Bild 4.13 Programmlisting

#### 4.4.8 Programmbeschreibung

Zu Beginn des Programms werden den GPIO-Ports 22 und 23 die Variablennamen LED1 und LED2 zugeordnet, dann werden die Port-Pins als Ausgänge konfiguriert. Das Hauptprogramm wird kontinuierlich in einer Schleife ausgeführt, wobei innerhalb dieser Schleife LED1 und LED2 alternierend mit je einer Sekunde Verzögerung ein- und ausgeschaltet werden.

### 4.5 PROJEKT 4 – Rotierende LEDs

#### 4.5.1 Beschreibung

In diesem Projekt sind vier LEDs mit den – wie Bild 4.2 zeigt – nebeneinander liegenden GPIO-Ports 23, 22, 1 und 3 verbunden. Die LEDs werden sukzessive ein- und ausgeschaltet. Nur eine LED ist zu einem beliebigen Zeitpunkt eingeschaltet, so dass ein LED-Muster wie folgt erzeugt wird:

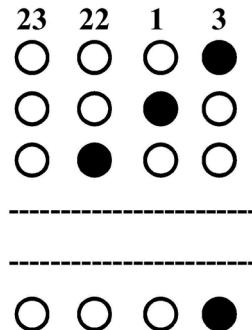


Bild 4.14 Rotierende LEDs

Eine LED soll, wenn sie „an der Reihe ist“, 500 ms leuchten und 100 ms dunkel bleiben.

#### 4.5.2 Das Ziel

Das Projekt soll zeigen, wie ein Array und eine **for**-Schleife in einem Programm zur Steuerung mehrerer Verbraucher (LEDs) an den GPIOs verwendet werden können.

#### 4.5.3 Blockschaltbild

Das Blockschaltbild des Projekts in Bild 4.15 zeigt, wie die vier LEDs LED1 ... LED4 mit den GPIO-Ports 23, 22, 1 und 3 verbunden sind.

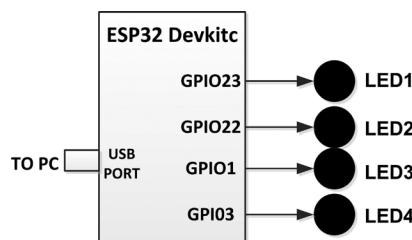


Bild 4.15 Blockschaltbild des Projekts

#### 4.5.4 Schaltplan

Der Schaltplan des Projekts in Bild 4.16 zeigt, dass die LEDs über 330- $\Omega$ -Strombegrenzungswiderstände mit den Port-Anschlüssen verbunden sind. Beachten Sie Bild 4.2 für Pin-Konfiguration des ESP32-DevKitC!

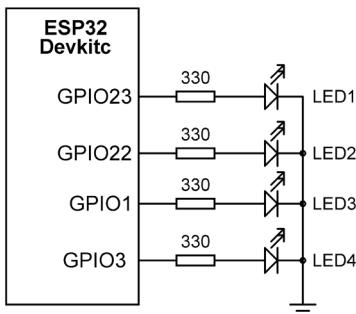


Bild 4.16 Schaltplan des Projekts

#### 4.5.5 Aufbau

Das Projekt wird auf einem Steckboard aufgebaut. Die LEDs werden über die Strombegrenzungswiderstände mit dem ESP32 Dev-KitC verbunden, wie in Bild 4.17 gezeigt.

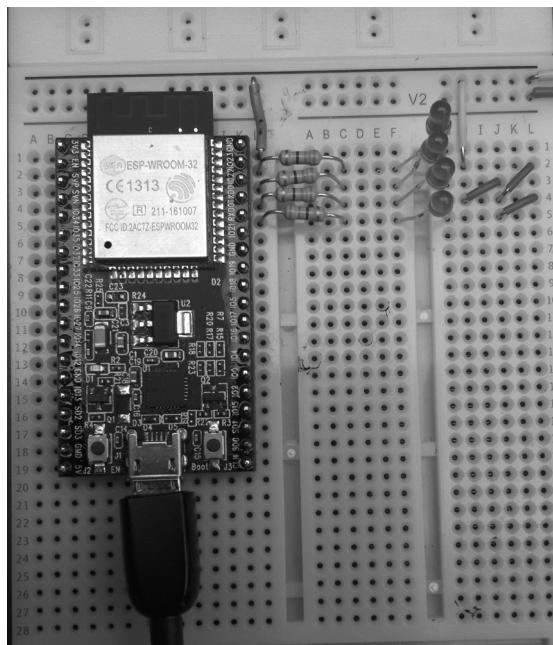


Bild 4.17 Projekt auf einem Steckbrett

#### 4.5.6 PDL des Projekts

Die PDL des Projekts ist in Bild 4.18 zu sehen. Das Programm läuft in einer Endlosschleife, in der die LEDs in einem „rotierenden“ Muster ein- und ausgeschaltet werden.

##### BEGIN

Store LED port numbers in array LEDs

Configure LED port pins as outputs and turn OFF the LEDs

##### DO FOREVER

```

DO FOR J = 0 TO 3
    Turn ON LED indexed by LEDs[J]
    Wait 500 ms
    Turn OFF LED indexed by LEDs[J]
    Wait 100 ms
ENDDO
ENDDO
END

```

*Bild 4.18 PDL des Projekts*

#### 4.5.7 Programmlisting

Das Programm des Projekts RotateLEDs in Bild 4.19 ist sehr einfach aufgebaut.

```

/*****
*          ROTATING LEDs
*          =====
*
* In this program four LEDs are connected to port GPIO23,
* GPIO22, GPIO1, and GPIO0 and of the ESP32-DevKitC. The
* program turns ON the LEDs in a rotating pattern. The ON
* and OFF times are chosen as 500 ms and 100 ms so that a
* nice rotating effect is displayed
*
* Program: RotateLEDs
* Date : July, 2017
*****
int LEDs[] = {23,22,1,3};
#define ON HIGH
#define OFF LOW

// Set GPIO pins 23,22,1,3 as outputs and turn OFF the LEDs
// to start with
//

void setup()
{
    unsigned char i;
    for(i=0; i <=3; i++)
    {
        pinMode(LEDs[i], OUTPUT);
        digitalWrite(LEDs[i], OFF);
    }
}

//
```

```
// Turn ON/OFF the LEDs in a rotating pattern
//
void loop()
{
    unsigned char j;
    for(j = 0; j <=3; j++)
    {
        digitalWrite(LEDs[j], ON);
        delay(500);
        digitalWrite(LEDs[j], OFF);
        delay(100);
    }
}
```

Bild 4.19 Listing des Programms RotateLEDs

#### 4.5.8 Programmbeschreibung

Zu Beginn des Programms wird ein Array mit der Bezeichnung LEDs eingerichtet, um die Port-Nummern der LEDs zu speichern. Dann werden diese GPIO-Ports als Ausgänge konfiguriert und alle LEDs ausgeschaltet. Innerhalb des Hauptprogramms sorgt eine **for**-Schleife dafür, dass die an den Pins 23, 22, 1 und 3 angeschlossenen LEDs in dieser Reihenfolge eingeschaltet werden. Als EIN- und AUS-Zeiten werden 500 ms und 100 ms gewählt, was einen schön rotierenden LED-Effekt zur Folge hat.

#### 4.5.9 Vorschläge

Verdoppeln Sie die Anzahl der LEDs und ändern Sie das Programm entsprechend.

### 4.6 PROJEKT 5 – Weihnachtsbeleuchtung

#### 4.6.1 Beschreibung

In diesem Projekt sind vier LEDs wie im vorherigen Projekt mit dem ESP32-DevKitC verbunden. Die LEDs werden jede Sekunde zufällig eingeschaltet und ausgeschaltet.

#### 4.6.2 Das Ziel

Das Ziel dieses Projekts soll es sein, die Funktion des Zufallszahlengenerators in einem Programm anzuwenden.

#### 4.6.3 Blockschaltbild

Das Blockschaltbild des Projekts ist in Bild 4.15 dargestellt.

#### 4.6.4 Schaltplan

Der Schaltplan des Projekts ist in Bildung 4.16 zu sehen.

#### 4.6.5 Aufbau

Auch dieses Projekt wird auf einem Steckboard aufgebaut, wie Bild 4.17 zeigt.

#### 4.6.6 PDL des Projekts

Die PDL des Projekts ist in Bild 4.20 dargestellt.

```
BEGIN/Main
    Store LED port numbers in array LEDs
    Configure LED port pins as outputs
    DO FOREVER
        Generate a random number between 1 and 15
        Call Display with the number to turn ON the appropriate LED
        Wait 1 second
    ENDDO
END/Main

BEGIN/Display
    Extract bit 3 of the number
    IF bit = 1 THEN
        Turn ON LED1
    ELSE
        Turn OFF LED1
    ENDIF
    Extract bit 2 of the number
    IF bit = 1 THEN
        Turn ON LED2
    ELSE
        Turn OFF LED2
    ENDIF
    Extract bit 1 of the number
    IF bit = 1 THEN
        Turn ON LED3
    ELSE
        Turn OFF LED3
    ENDIF
    Extract bit 0 of the number
    IF bit = 1 THEN
        Turn ON LED4
    ELSE
        Turn OFF LED4
    ENDIF
END/Display
```

*Bild 4.20 PDL des Projekts*

#### 4.6.7 Programmlisting

Das Listing des Programms Christmas ist sehr einfach und wird in Bild 4.21 gezeigt.

```
/*****  
*                               CHRISTMAS LIGHTS  
*                               =====  
*  
* In this program four LEDs are connected to port GPIO23,  
* GPIO22, GPIO1, and GPIO3 and of the ESP32-DevKitC. The  
* program turns ON the LEDs in a random manner every second  
* to give the effect of for example Christmas lights  
*  
* Program: Christmas  
* Date : July, 2017  
*****/  
  
int LEDs[] = {23,22,1,3};  
unsigned char Ran;  
  
//  
// Set GPIO pins 23,22,1,3 as outputs  
//  
  
void setup()  
{  
    unsigned char i;  
    for(i=0; i <=3; i++)  
    {  
        pinMode(LEDs[i], OUTPUT);  
    }  
    randomSeed(10);  
}  
  
//  
// Turn ON the appropriate LED  
//  
void Display(unsigned char No)  
{  
    digitalWrite(23,(No & B00001000));  
    digitalWrite(22,(No & B00000100));  
    digitalWrite(1, (No & B00000010));  
    digitalWrite(3, (No & B00000001));  
}  
  
//  
// Turn ON/OFF the LEDs randomly by generating a random number  
// between 1 and 15  
//  
void loop()
```

```

{
    Ran = random(1,16);
    Display(Ran);
    delay(1000);
}

```

*Bild 4.21 Programmlisting*

#### 4.6.8 Programmbeschreibung

Zu Beginn des Programms wird ein Array namens LEDs eingerichtet, um die Port-Nummern für die im Projekt verwendeten LEDs festzulegen. Dann werden die GPIO-Ports, an denen die LEDs angeschlossen sind, als Ausgangs-Ports konfiguriert. Die Funktion **randomSeed** wird mit einer Integer-Zahl aufgerufen, um den Zufallszahlengenerator zu starten. Innerhalb des Hauptprogramms wird eine zufällige Zahl zwischen 1 und 15 erzeugt (beachten Sie, dass die **Random**-Funktion stets die untere Grenze berücksichtigt, die obere Grenze aber ausschließt). Dann wird die Funktion **Display** aufgerufen, um die Bits der Zufallszahl zu extrahieren und entsprechend dieser Bits die LED ein- oder auszuschalten. Die Funktion **Display** empfängt so eine Integerzahl zwischen 1 und 15 und extrahiert die Bits dieser Zahl. Wenn zum Beispiel die Zahl 12 gewählt ist (binär 1100), dann werden die beiden LEDs links (LED1 und LED2) eingeschaltet.

#### 4.6.9 Geändertes Programm

Das Programm in Bild 4.21 kann effizienter gestaltet werden, wenn man die Funktion **Display** modifiziert. Das neue Programmlisting ist in Bild 4.22 (Programm Christmas2) dargestellt. Hier hat die Funktion **Display** zwei Argumente: **No** ist die Integerzahl, die als binäres Bitmuster durch die LEDs angezeigt wird, **L** ist die Breite der Integerzahl in Bits. Die Bits werden aus der Zahl extrahiert und an die LEDs gesendet, so dass die richtigen LEDs eingeschaltet werden. Das Programm läuft in einer Endlosschleife, in der eine Zufallszahl zwischen 1 und 15 erzeugt und in der Variable **Ran** gespeichert wird. Die Funktion **Display** wird dann als **Display(Ran,4)** aufgerufen, um die entsprechenden LEDs ein- und auszuschalten. Beachten Sie, dass in diesem Projekt die Zahl vier Bits breit ist, weil es ja vier LEDs gibt.

```

*****
*                               CHRISTMAS LIGHTS
*                               =====
*
* In this program four LEDs are connected to port GPIO23,
* GPIO22, GPIO1, and GPIO3 and of the ESP32-DevKitC. The
* program turns ON the LEDs in a random manner every second
* to give the effect of for example Christmas lights.
*
* In this modified program function Display is modified and
* the program is more efficient and easier to maintain
*
* Program: Christmas2
* Date : July, 2017

```

```
*****  
  
int LEDs[] = {23,22,1,3};  
unsigned char Ran;  
  
//  
// Set GPIO pins 23,22,1,3 as outputs  
//  
void setup()  
{  
    unsigned char i;  
    for(i=0; i <=3; i++)  
    {  
        pinMode(LEDs[i], OUTPUT);  
    }  
    randomSeed(10);  
}  
  
//  
// Turn ON the appropriate LED  
//  
void Display(unsigned char No, unsigned char L)  
{  
    unsigned char j, m, i;  
    m = L - 1;  
    for(i = 0; i < L; i++)  
    {  
        j = pow(2, m);  
        digitalWrite(LEDs[i], (No & j));  
        m--;  
    }  
}  
  
//  
// Turn ON/OFF the LEDs randomly by generating a random number  
// between 1 and 15  
//  
void loop()  
{  
    Ran = random(1, 15);  
    Display(Ran, 4);  
    delay(1000);  
}
```

Bild 4.22 Modifiziertes Programm

#### 4.6.10 Vorschläge

Die Anzahl der LEDs kann erhöht werden, indem man mehrere in Serie und parallel geschaltet zum Beispiel auf einem Weihnachtsbaum anordnet.

### 4.7 PROJEKT 6 – Binärzähler mit LEDs

#### 4.7.1 Beschreibung

In diesem Projekt sind acht LEDs an dem ESP32-DevKitC angeschlossen. Das Programm zählt von 0 bis 255, die entsprechende Binärzahl wird durch die acht LEDs dargestellt.

#### 4.7.2 Das Ziel

Ziel dieses Projekts ist es, 8 Port-Pins zusammenzufassen und wie ein 8-Bit-Ausgangs-Port zu behandeln.

#### 4.7.3 Blockschaltbild

Das Blockschaltbild des Projekts ist in Bild 4.23 zu sehen. Die LEDs sind mit folgenden GPIO-Anschlüssen verbunden (siehe ESP32-DevKitC-Pin-Konfiguration in Bild 4.2):

23 (MSB)  
22  
1  
3  
21  
19  
18  
5 (LSB)

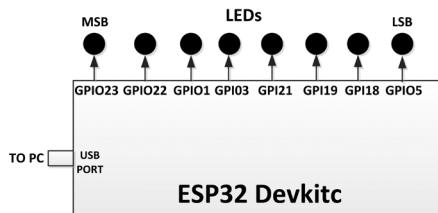


Bild 4.23 Blockschaltbild des Projekts

#### 4.7.4 Schaltplan

Der Schaltplan des Projekts in Bild 4.24 zeigt acht LEDs, die über 330- $\Omega$ -Strombegrenzungswiderstände an den GPIO-Ports angeschlossen sind.

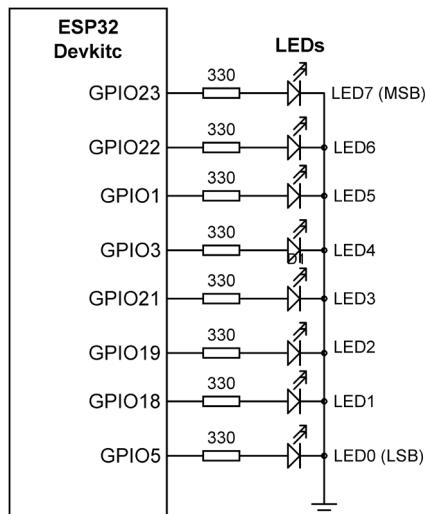


Bild 4.24 Schaltplan des Projekts

#### 4.7.5 Bau

Das Projekt ist wie in Bild 4.25 auf einem Steckbrett aufgebaut.

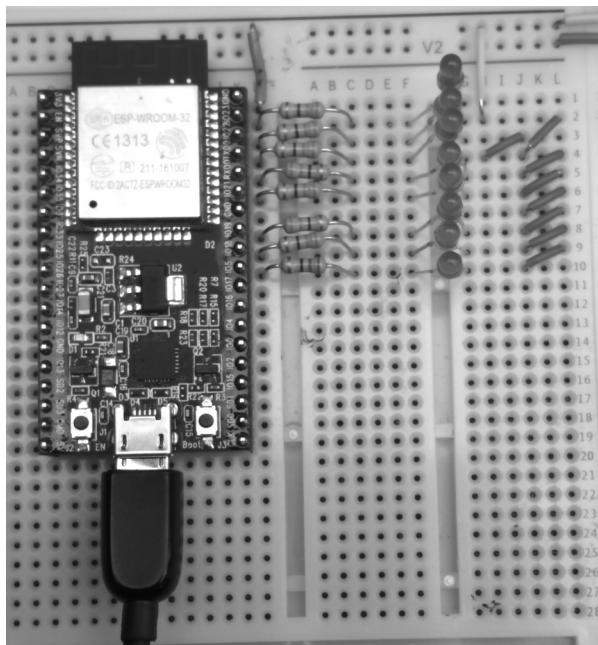


Bild 4.25 Aufbau des Projekts auf einem Steckboard

#### 4.7.6 PDL des Projekts

Die PDL des Projekts ist in Bild 4.26 dargestellt.

**BEGIN/Main**

Store LED port numbers in array LEDs

Configure LED port pins as outputs

Set Count to 0

**DO FOREVER**

**Call** Display with Count and 8 (width) as the arguments

Increment Count

Wait 1 second

**ENDDO****END/Main****BEGIN/Display**

Extract bits of the number and send to appropriate GPIO pins

**END/Display**

*Bild 4.26 PDL des Projekts*

#### 4.7.7 Programmlisting

Die einfache Listing des Programms Counter ist in Bild 4.27 zu sehen.

```
/*
 *          BINARY UP COUNTER
 *=====
 *
 * In this program 8 LEDs are connected to port pins GPIO23,
 * GPIO22, GPIO1, GPIO3, GPIO21, GPIO19, GPIO18, and GPIO5
 * of the ESP32-DevKitC. The program counts up by one and
 * sends the count to the GPIO pins to turn ON/OFF the
 * appropriate LEDs so that the LEDs count up by one in binary.
 * A one second delay is inserted between each output
 *
 *
 * Program: Counter
 * Date : July, 2017
 */
int LEDs[] = {23,22,1,3,21,19,18,5};
unsigned char Count = 0;

// Set GPIO pins 23,22,1,3,21,19,18,5 as outputs
//
void setup()
{
    unsigned char i;
    for(i=0; i < 8; i++)
```

```
{  
    pinMode(LEDs[i], OUTPUT);  
}  
}  
  
//  
// Turn ON the appropriate LED  
//  
void Display(unsigned char No, unsigned char L)  
{  
    unsigned char j,m,i;  
    m = L - 1;  
    for(i = 0; i < L; i++)  
    {  
        j = pow(2, m);  
        digitalWrite(LEDs[i], (No & j));  
        m--;  
    }  
}  
  
//  
// Increment Count every second and send Count to GPIO  
// pins. Notice that the width is 8-bits (i.e. there are  
// 8 LEDs)  
//  
void loop()  
{  
    Display(Count,8);  
    if(Count == 255)  
        Count = 0;  
    else  
        Count++;  
    delay(1000);  
}
```

Bild 4.27 Programmlisting

#### 4.7.8 Programmbeschreibung

Am Anfang des Programms wird ein Array namens **LEDs** eingerichtet, um die Port-Nummern der verwendeten LEDs festzulegen. Die globale Variable **Count** wird auf den Anfangswert Null eingestellt. Dann werden die GPIO-Ports mit den LEDs als Ausgänge eingestellt. Innerhalb des Hauptprogramms wird die Funktion **Display** aufgerufen und **Count** als Argument angegeben. Die Funktion extrahiert die Bits von **Count** und schaltet dem binären Muster entsprechend die LEDs ein und aus. Die Zahl 8 ist auch ein Argument der Funktion **Display**, da es acht LEDs gibt, die Breite der Zahl also 8 Bit beträgt. Die LEDs zählen kontinuierlich binär von 0 bis 255, mit je einer Sekunde Verzögerung bei jedem Zählschritt. Es wird ein Muster erzeugt, wie es in Bild 4.28 dargestellt ist.

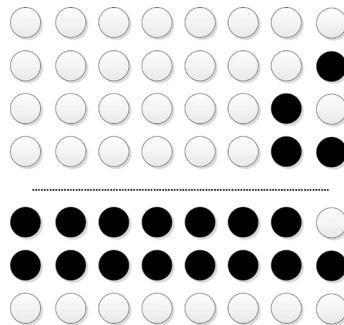


Bild 4.28 LED-Muster

## 4.8 PROJEKT 7 – Binärer Aufwärts-/Abwärtszähler mit LEDs

### 4.8.1 Beschreibung

In diesem Projekt sind wie im vorherigen Projekt acht LEDs mit dem ESP32-DevKitC verbunden. Zusätzlich ist ein Drucktaster (**button**) an GPIO-Pin 17 angeschlossen. Wenn die Taste nicht gedrückt wird, zählt das Programm aufwärts, wenn die Taste gedrückt wird, abwärts.

### 4.8.2 Das Ziel

Das Projekt soll zeigen, wie ein Drucktastenschalter an einen GPIO-Port des ESP32-DevKitC angeschlossen und wie dieser Port als Eingang konfiguriert werden kann.

### 4.8.3 Blockschaltbild

Das Blockschaltbild des Projekts in Bild 4.29 zeigt, wie die LEDs mit den folgenden GPIO-Anschlüssen verbunden sind. Beachten Sie Bild 4.2 für die Pin-Konfiguration des ESP32-DevKitC!

23 (MSB)  
22  
1  
3  
21  
19  
18  
5 (LSB)

Der Taster ist mit dem GPIO-Pin 17 verbunden.

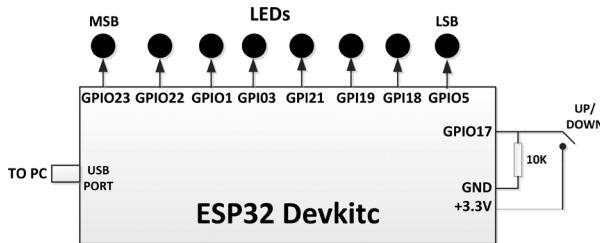


Bild 4.29 Blockschaltbild des Projekts

#### 4.8.4 Schaltplan

Der Schaltplan des Projekts in Bild 4.30 zeigt acht LEDs, die über 330- $\Omega$ -Strombegrenzungswiderstände an den GPIO-Ports angeschlossen sind. Der Taster kann auf zwei verschiedene Arten an den GPIO-Port angeschlossen werden. In Bild 4.31 liegt der Kontakt des Tasters (am GPIO-Pin) auf logisch 1 und geht, wenn der Taster gedrückt wird, auf logisch 0. Schließt man dagegen den Taster so an wie in Bild 4.32, ist es genau umgekehrt: Bei einem Druck auf den Taster geht der GPIO-Pin von logisch 0 auf logisch 1. In diesem Beispiel wird die zweite Methode verwendet. Der GPIO-Pin liegt also normalerweise auf logisch 0 und geht auf logisch 1, wenn der Taster gedrückt wird.

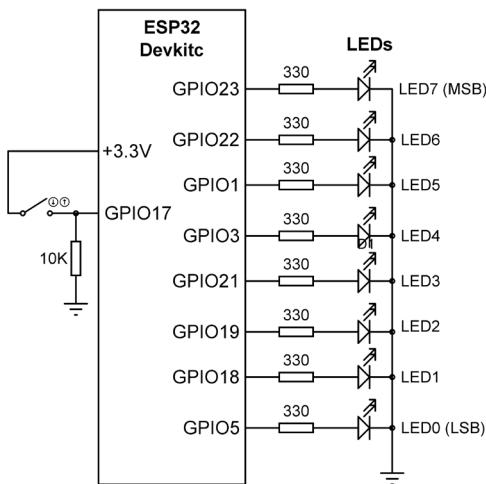


Bild 4.30 Schaltplan des Projekts

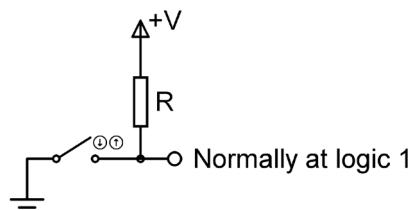


Bild 4.31 Der Tasterkontakt liegt normalerweise auf logisch 1

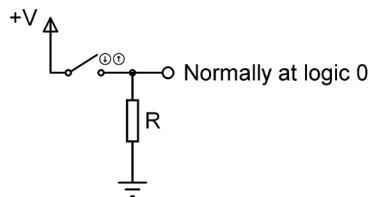


Bild 4.32 Der Tasterkontakt liegt normalerweise auf logisch 0

#### 4.8.5 Aufbau

Das Projekt ist wie in Bild 4.33 zu sehen auf einem Steckboard aufgebaut.

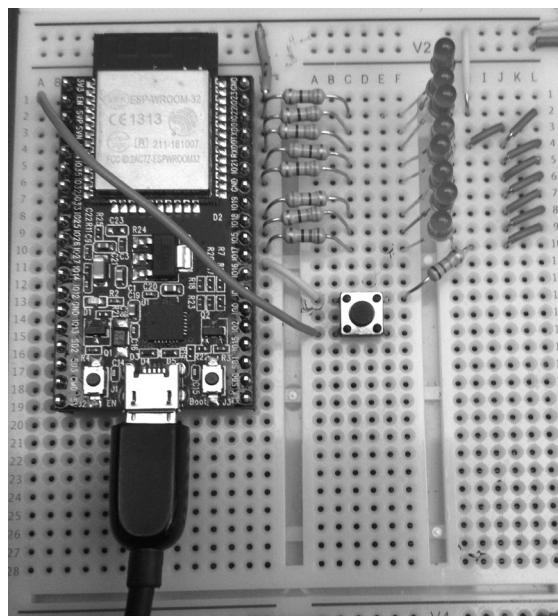


Bild 4.33 Aufbau des Projekts auf einem Steckboard

#### 4.8.6 PDL des Projekts

Die PDL des Projekts ist in Bild 4.34 dargestellt.

##### BEGIN/Main

Store LED port numbers in array LEDs

Configure LED port pins as outputs

Set Count to 0

##### DO FOREVER

Call Display with Count and 8 (width) as the arguments

**IF** Button not pressed **THEN**

    Increment Count

**ELSE IF** Button is pressed **THEN**

    Decrement Count

##### ENDIF

```
    Wait 1 second
ENDDO
END/Main

BEGIN/Display
    Extract bits of the number and send to appropriate GPIO pins
END/Display
```

*Bild 4.34 PDL des Projekts*

#### 4.8.7 Programmlisting

Die Programmlisting UpDown des Projekts ist sehr einfach und ist in Bild 4.35 dargestellt.

```
/*****
*                                BINARY UP/DOWN COUNTER
* =====
*
* In this program 8 LEDs are connected to port pins GPIO23, GPIO22, GPIO1,
* GPIO3, GPIO21, GPIO19, GPIO18, and GPIO5 of the ESP32-DevKitC. In
* addition, a push-button switch is connected to port pin 17. Normally the
* switch output is at logic 0 and when the switch is pressed its output goes
* to logic 1. The program normally counts up. When the switch is pressed and
* held down then the program starts to count down from its last value. The
* appropriate LEDs are turned ON/OFF
*
*
* Program: UpDown
* Date : July, 2017
*****/
```

```
#define Button 17
#define UP 0
#define DOWN 1
int LEDs[] = {23, 22, 1, 3, 21, 19, 18, 5};
unsigned char Count = 0;
unsigned char Button_State;

// 
// Set GPIO pins 23,22,1,3,21,19,18,5 as outputs and GPIO pin 17
// as input
//
void setup()
{
    unsigned char i;
    for(i=0; i < 8; i++)
    {
        pinMode(LEDs[i], OUTPUT);
```

```

        }
        pinMode(Button, INPUT);
    }

// 
// Turn ON the appropriate LED
//
void Display(unsigned char No, unsigned char L)
{
    unsigned char j, m, i;
    m = L - 1;
    for(i = 0; i < L; i++)
    {
        j = pow(2, m);
        digitalWrite(LEDs[i], (No & j));
        m--;
    }
}

// 
// When the button is not pressed increment Count every second
// and send Count to GPIO pins. When the button is pressed
// count down from the last value and send Count to GPIO pins
// Notice that the width is 8-bits (i.e. there are 8 LEDs)
//
void loop()
{
    Display(Count, 8);
    Button_State = digitalRead(Button); // Read Button state
    if(Button_State == UP) // If UP count
    {
        if(Count == 255)
            Count = 0; // Reset Count
        else
            Count++; // Increment Count
    }
    else if(Button_State == DOWN) // If DOWN count
    {
        if(Count == 0)
            Count = 255; // Reset Count
        else
            Count--; // Decrement Count
    }
    delay(1000); // Wait 1 second
}

```

*Bild 4.35 Programmlisting*

#### 4.8.8 Programmbeschreibung

Am Beginn des Programms wird ein Array namens LEDs eingerichtet, um die Port-Nummern der verwendeten LEDs festzulegen. Der Name **Button** (Taster) wird dem GPIO-Port 17, die Namen **UP** und **DOWN** werden die Werte 0 und 1 zugeordnet. Die globale Variable **Count** wird auf Null initialisiert. Dann werden die GPIO-Ports, an denen LEDs angeschlossen sind, als Ausgangs-Ports und der GPIO-Port mit dem **Button** als Eingangs-Port konfiguriert.

Im Hauptprogramm wird die Funktion **Display** mit **Count** als Argument aufgerufen. Anschließend wird der Zustand des Tasters ermittelt: Wenn der Taster nicht gedrückt ist (**Button\_State** gleich **UP**), wird aufwärts (Count++) gezählt, ist die Taste aber gedrückt (**Button\_State** gleich **DOWN**), wird abwärts gezählt (Count--). In beiden Fällen wird die Variable **Count** am Ende der Zählung zurückgesetzt. Erreicht der Wert während der Aufwärtszählung 255, wird sie im nächsten Zyklus auf 0 zurückgesetzt. Ähnlich geht es bei einer Abwärtszählung zu: Wenn der Zählerstand 0 erreicht, wird der Wert im nächsten Zyklus auf 255 gesetzt.

### 4.9 PROJEKT 8 – Knight Rider LEDs

#### 4.9.1 Beschreibung

Knight Rider ist ein TV-Action-Film mit einem super intelligenten, sprechenden und selbst navigierenden Auto namens K.I.T.T. An dessen Front ist ein Leuchtstreifen montiert, in dem eine Leuchte kontinuierlich hin und her läuft. In diesem Projekt sind acht LEDs mit dem ESP32-DevKitC (statt mit dem K.I.T.T.) verbunden, die dieses Lauflicht simulieren.

#### 4.9.2 Das Ziel

Ziel dieses Projekts ist es, ein Lauflicht à la Knight Rider mit dem ESP32-Prozessor zu erzeugen.

#### 4.9.3 Blockschaltbild

Die Blockschaltung des Projekts ist in Bild 4.23 dargestellt.

#### 4.9.4 Schaltplan

Der Schaltplan des Projekts ist in Bild 4.24 zu sehen. An den GPIO-Ports sind über  $330\text{-}\Omega$ -Strombegrenzungswiderstände acht LEDs angeschlossen.

#### 4.9.5 Bau

Das Projekt ist auf einem Steckboard (Bild 4.25) aufgebaut.

#### 4.9.6 PDL des Projekts

Die PDL des Projekts ist in Bild 4.36 dargestellt.

```
BEGIN
    Store LED port numbers in array LEDs
    Configure LED port pins as outputs
DO FOREVER
    Do k From 0 to 8
        Turn ON LED at index LEDs[k]
```

```

        Wait 100 ms
        Turn OFF LED at index LEDs[k]
ENDDO
DO k From 6 to 0
    TURN ON LED at index LEDs[k]
    Wait 100 ms
    Turn OFF LED at index LEDs[k]
ENDDO
END

```

*Bild 4.36 PDL des Projekts*

#### 4.9.7 Programmlisting

Die Programmlisting KnightRider des Projekts ist sehr einfach (Bild 4.37):

```

/*****
*
*                      KNIGHT RIDER LEDs
*                      =====
*
* In this program 8 LEDs are connected to port pins GPIO23,
* GPIO22, GPIO1, GPIO3, GPIO21, GPIO19, GPIO18, and GPIO5
* of the ESP32-DevKitC. The program simulates the lights of
* the Knight Rider car as in the TV action movie Knight Rider.
*
*
* Program: KnightRider
* Date : July, 2017
*****/


int LEDs[] = {23, 22, 1, 3, 21, 19, 18, 5};
unsigned char Count = 0;
unsigned char del = 100;

// 
// Set GPIO pins 23,22,1,3,21,19,18,5 as outputs
//
void setup()
{
    unsigned char i;
    for(i=0; i < 8; i++)
    {
        pinMode(LEDs[i], OUTPUT);
    }
}

// 
// Turn the LEDs ON/OFF to simulate the Knight Rider car

```

```
//  
void loop()  
{  
    for(int k = 0; k < 8; k++)  
    {  
        digitalWrite(LEDs[k], HIGH);  
        delay(del);  
        digitalWrite(LEDs[k], LOW);  
    }  
    for(int k = 6; k > 0; k--)  
    {  
        digitalWrite(LEDs[k], HIGH);  
        delay(del);  
        digitalWrite(LEDs[k], LOW);  
    }  
}
```

Bild 4.37 Programmlisting

#### 4.9.8 Programmbeschreibung

Zu Beginn des Programms wird ein Array namens LEDs eingerichtet, um die LEDs den Port-Nummern zuzuordnen. Anschließend werden diese GPIOs als Ausgangs-Ports konfiguriert. Im Hauptprogramm gibt es zwei **for**-Schleifen: In der ersten werden die LEDs vom MSB zum LSB für 100 ms eingeschaltet, in der zweiten **for**-Schleife vom LSB zum MSB, ebenfalls für 100 ms. Das Resultat ist, dass die LEDs in beide Richtungen jagen.

#### 4.9.9 Vorschläge

Im Programm ist die Verzögerungszeit bei jedem Wechsel auf 100 ms festgesetzt. Versuchen Sie, die Zeit zu ändern, und schauen Sie, was passiert.

### 4.10 PROJEKT 9 – Ändern der Helligkeit einer LED

#### 4.10.1 Beschreibung

In diesem Projekt ist eine LED an den ESP32-DevKitC angeschlossen. Das Programm ändert die an die LED angelegte Spannung und somit deren Helligkeit.

#### 4.10.2 Das Ziel

In diesem Projekt wird mit einem Programm eine PWM-Wellenform erzeugt. Diese Wellenform kann verwendet werden, um die an einer LED angelegte effektive Spannung zu variieren.

#### 4.10.3 Blockschaltbild

Das Blockschaltbild des Projekts ist das gleiche wie in Bild 4.1.

#### 4.10.4 Schaltplan

Der Schaltplan des Projekts ist in Bild 4.3 zu sehen. Eine LED ist über einen 330- $\Omega$ -Strombegrenzungswiderstand mit dem GPIO-Port 23 verbunden, genau wie in Projekt 1.

#### 4.10.5 Aufbau

Das Projekt ist wie in Bild 4.4 auf einem Steckboard aufgebaut.

#### 4.10.6 PDL des Projekts

Die PDL des Projekts ist in Bild 4.38 dargestellt.

```

BEGIN
    Define LED port
    Define PWM frequency, channel, and resolution
    Configure LED port pin as output
    Setup and attach port 23 to PWM channel 0
DO FOREVER
    DO 20 Times
        Increment the duty cycle by 5%, from 0% to 100%
        Send the waveform to port 23
        Wait 50 ms
ENDDO
DO 20 Times
    Decrement the duty cycle by 5%, from 100% to 0%
    Send the waveform to port 23
    Wait 50 ms
ENDDO
END

```

*Bild 4.38 PDL des Projekts*

#### 4.10.7 Programmlisting

FadeLED ist ein einfach aufgebautes Programm, wie das Listing 4.39 beweist.

```

/*
 *                               CHANGING THE LED BRIGHTNESS
 *=====
 *
 * In this program an LED is connected to GPIO port pin 23.
 * The program changes the brightness of the LED by varying
 * the voltage applied to the LED. The LED PWM function is used
 * in this project where the duty cycle of the PWM waveform is
 * varied from 0 to its full value of 255
 *
 * Program: Brightness
 * Date : July, 2017
 */
int LED = 23;
int frequency = 1000;
int resolution = 8;
int channel = 0;

```

```
//  
// Set GPIO pin 23 as output. Setup the PWM channel 0 and attach  
// GPIO pin 23 to this channel  
//  
void setup()  
{  
    pinMode(LED, OUTPUT);  
    ledcSetup(channel, frequency, resolution);  
    ledcAttachPin(LED, channel);  
}  
  
//  
// Change the brightness of the LED by varying the duty cycle  
// from 0% (0) to 100% (255)  
//  
void loop()  
{  
    for(int duty = 0; duty < 255; duty +=5)  
    {  
        ledcWrite(channel, duty);  
        delay(50);  
    }  
    for(int duty = 255; duty >= 0; duty -=5)  
    {  
        ledcWrite(channel, duty);  
        delay(50);  
    }  
    delay(500);  
}
```

Bild 4.39 Programmlisting

#### 4.10.8 Programmbeschreibung

Das Dimmen der LED erfolgt durch Anlegen eines PWM-Signals und der Änderung des Tastverhältnisses dieser Wellenform. Dadurch ändert sich auch die an der LED anliegende mittlere Spannung. Bild 4.40 zeigt eine typische PWM-Wellenform. Das Tastverhältnis oder der Duty-Cycle ist definiert als das Verhältnis von Einschaltzeit zur Periodendauer.

$$\text{Duty cycle} = \frac{\text{ONtime}}{\text{Period}} \times 100\%$$

oder

$$\text{Duty cycle} = \frac{\text{ONtime}}{(\text{ONtime} + \text{OFFtime})} \times 100\%$$

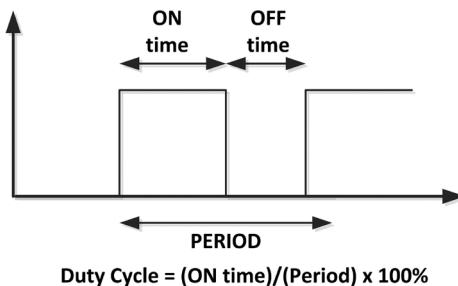


Bild 4.40 Eine typische PWM-Wellenform

Wenn zum Beispiel das Tastverhältnis 0 % ist, beträgt die Einschaltzeit ON 0, ist das Tastverhältnis dagegen 100 %, ist die Ausschaltzeit OFF 0.

Das Programm nutzt die LED-PWM-Funktion des ESP32, die aus 16 unabhängigen Kanälen mit konfigurierbaren Tastverhältnissen, Frequenzen und Auflösungen besteht. Es kann ein Kanal von 0 ... 15 und eine Auflösung von 1 ... 16 Bit ausgewählt werden. Die PWM-Setup-Funktion **ledcSetup** weist das folgende Format mit Integer-Parametern auf:

```
ledcSetup(channel,frequency,resolution)
```

Nach dem Setup muss der GPIO-Pin, den wir als PWM-Ausgang verwenden wollen, mit der Funktion **ledcAttach** im Format

```
ledcAttachPin(GPIO pin, channel)
```

definiert werden. Das PWM-Signal wird dann mit der Funktion **ledcWrite** an den gewählten GPIO-Pin gesendet:

```
ledcWrite(channel, duty cycle)
```

wobei bei einer Auflösung von 8 Bit das Tastverhältnis 0 für 0 % und 255 für 100 % steht.

In diesem Projekt wird die Frequenz auf 1000 Hz eingestellt und Kanal 0 mit dem GPIO-Pin 23 verwendet. Die Auflösung beträgt 8 Bit. Das Tastverhältnis wird alle 50 Millisekunden in Schritten von 5 % von 0 ... 100 % geändert. Dadurch blendet die LED (in besagten 5%-Schritten im 50-ms-Takt) von 0 (das heißt, die LED ist dunkel) bis zur vollen Helligkeit bei 100 % auf. Bei voller Helligkeit wartet das Programm 500 ms, dann wird die LED um 5 % alle 50 ms wieder dunkler.

#### 4.10.9 Vorschläge

In Bild 4.39 ist die Verzögerungszeit auf 50 ms und die Tastverhältnis-Stufe auf 5 % eingestellt. Modifizieren Sie beide Werte und beobachten Sie die Auswirkungen auf die LED-Helligkeit.

## 4.11 PROJEKT 10 – Zufällige Sounds mit einem Summer

### 4.11.1 Beschreibung

In diesem Projekt ist ein passiver Summer an GPIO-Pin 23 des ESP32-DevKitC angeschlossen. Der Summer ertönt in Frequenzen, die sich zufällig zwischen 100 Hz und 5 kHz ändern.

### 4.11.2 Das Ziel

Ziel dieses Projekts ist es, zu zeigen, wie ein passiver Summer an das ESP32-DevKitC angeschlossen und Töne mit verschiedenen Frequenzen erzeugt werden können.

### 4.11.3 Blockschaltbild

Das Blockschaltbild des Projekts ist in Bild 4.41 dargestellt.

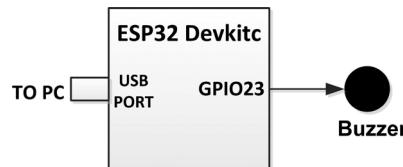


Bild 4.41 Blockschaltbild des Projekts

### 4.11.4 Schaltplan

Der Schaltplan des Projekts ist in Bild 4.42 zu sehen. Der Summer ist mit dem GPIO-Anschluss 23 verbunden.

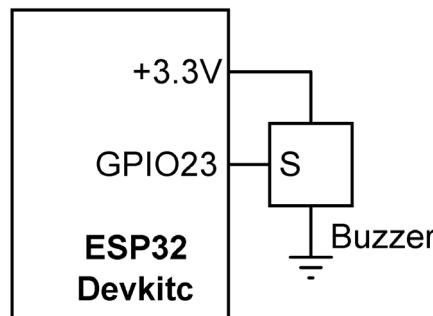


Bild 4.42 Schaltplan des Projekts

### 4.11.5 Aufbau

Das Projekt ist auf einem Steckbrett aufgebaut, wie in Bild 4.43 gezeigt. Bild 4.44 zeigt den passiven Summer, der in diesem Projekt verwendet wird.

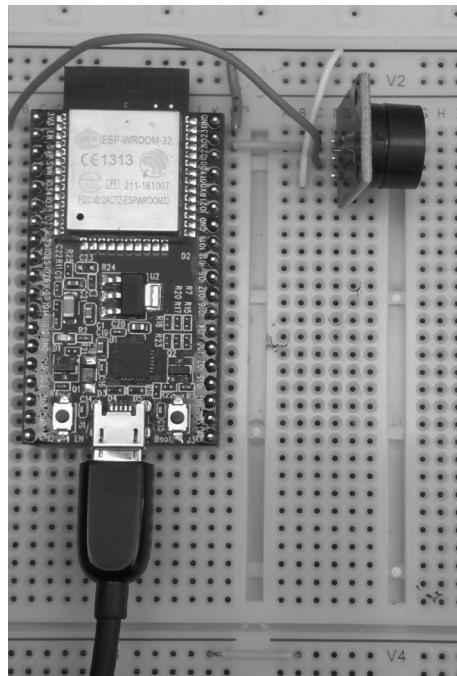


Bild 4.43 Das Projekt wurde auf einem Steckbrett aufgebaut

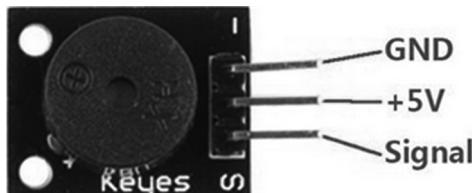


Bild 4.44 Der passive Summer, der im Projekt verwendet wird

#### 4.11.6 PDL des Projekts

Die PDL des Projekts ist in Bild 4.45 dargestellt.

```

BEGIN
    Define LED port
    Define PWM initial frequency, channel, and resolution
    Configure LED port pin as output
    Setup and attach port 23 to PWM channel 0
DO FOREVER
    Create a random number between 1000 and 5000
    Use this number as the frequency and send to the buzzer
    Wait 500 ms
ENDDO
END

```

Bild 4.45 PDL des Projekts

#### 4.11.7 Programmlisting

Das sehr einfache Programm des Projekts RandomBuzzer ist in Bild 4.46 dargestellt.

```
*****  
*          Random Buzzer  
*          ======  
*  
* In this program a passive Buzzer is connected to GPIO port  
* pin 23. The program generates random numbers between 1000  
* and 5000 and these numbers are used as the random frequency  
* to drive the Buzzer. The net effect is that the Buzzer  
* generates sound with random frequencies. The duty cycle of  
* the waveform is set to 50% (127). 500 ms delay is inserted  
* between each output.  
*  
*  
* Program: RandomBuzzer  
* Date   : July, 2017  
*****/  
int Buzzer = 23;                                // Buzzer on GPIO pin 23  
int duty = 127;                                  // 50% duty cycle  
int freq = 1000;                                 // Initial freq 1000 Hz  
int resolution = 8;                             // Resolution 8 bits  
int channel = 0;                                // Channel 0  
  
//  
// Set GPIO pin 23 as output. Setup the PWM channel 0 and attach  
// GPIO pin 23 to this channel  
//  
void setup()  
{  
    pinMode(Buzzer, OUTPUT);  
    ledcSetup(channel, freq, resolution);  
    ledcAttachPin(Buzzer, channel);  
    randomSeed(10);  
}  
  
void setColor(int redValue, int greenValue, int blueValue) {  
    analogWrite(redPin, redValue);  
    analogWrite(greenPin, greenValue);  
    analogWrite(bluePin, blueValue);  
}  
setColor(255, 0, 0); // Red Color  
delay(1000);  
setColor(0, 255, 0); // Green Color  
delay(1000);
```

```

    setColor(0, 0, 255); // Blue Color
    delay(1000);
    setColor(255, 255, 255); // White Color
    delay(1000);
    setColor(170, 0, 255); // Purple Color
    delay(1000);

    //
    // Use the random number generator to generate the frequency
    // values between 1000 Hz and 5000 Hz. The duty cycle is set
    // to 50% (127)
    //
    void loop()
{
}

}

```

*Bild 4.46 Listing des Programms RandomBuzzer*

#### **4.11.8 Programmbeschreibung**

Es gibt zwei Arten von Summern: passive und aktive. Passive Summer benötigen ein Wechselstromsignal im hörbaren Frequenzbereich, aktive Summer besitzen dagegen einen internen Oszillator mit einer festen Frequenz. Aktive Summer erzeugen einen festliegenden Ton, wenn eine logische 1 angelegt wird, ähnlich wie beim Einschalten einer LED. Die Frequenz eines aktiven Summers dagegen kann durch die Periodenfrequenz des angelegten 1/0-Logiksignals geändert werden. Dieses Projekt verwendet deshalb einen passiven Summer, dessen Tonhöhe wir mit dem PWM-Signal bestimmen. Wie im vorherigen Projekt werden die Funktionen **ledcSetup** und **ledcAttach** verwendet, um den erforderlichen GPIO-Pin an den PWM-Kanal anzuschließen. Mit der Funktion **ledcWriteTone** ändern wir die Frequenz des PWM-Signals. Das Format dieser Funktion lautet:

ledcWriteTone(channel, required frequency)

In diesem Projekt wird Kanal 0 mit dem GPIO-Pin 23 verwendet und die Auflösung auf 8 Bit gesetzt. Das Tastverhältnis ist fix auf 50 % (127) eingestellt. Eine Zufallszahl wird zwischen 1000 und 5000 wird erzeugt und diese Zahl als Wert der Frequenz verwendet, um den Summer zu aktivieren. Nach jedem Tonereignis tritt eine Verzögerung von 600 ms ein.

#### **4.11.9 Vorschlag**

In Bild 4.46 wird nur die Frequenz zufällig geändert. Ändern Sie auch das Tastverhältnis und beobachten Sie den Unterschied.

### **4.12 PROJEKT 11 – LED-Farbtafel**

#### **4.12.1 Beschreibung**

In diesem Projekt soll eine RGB-LED verschiedene Lichtfarben erzeugen, genau wie bei einer Farbtafel.

#### 4.12.2 Das Ziel

Dieses Projekt soll zeigen, wie eine RGB-LED in einem Programm verwendet werden kann, um verschiedene Lichtfarben zu erzeugen.

#### 4.12.3 Blockschaltbild

Das Blockschaltbild des Projekts ist in Bild 4.47 dargestellt.

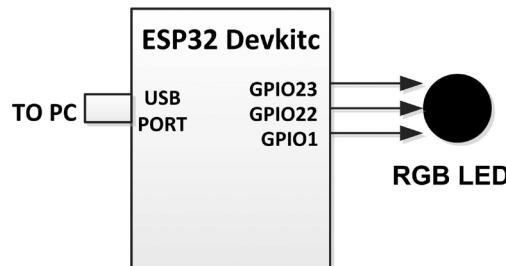


Bild 4.47 Blockschaltbild des Projekts

#### 4.12.4 Schaltplan

Der Schaltplan des Projekts in Bild 4.48 zeigt, dass die drei Anschlüsse RGB der LED mit den GPIO-Pins 23, 22 und 1 des ESP32-DevKitC verbunden sind. Nicht im Bild sind die  $330\text{-}\Omega$ -Strombegrenzungswiderstände.

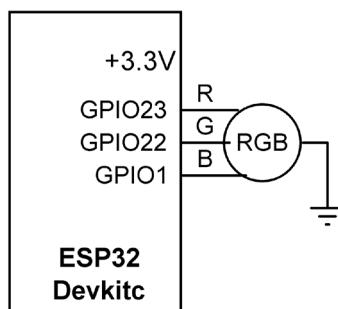


Bild 4.48 Schaltplan des Projekts

#### 4.12.5 Aufbau

Das Projekt wird auf einem Experimentier-Board wie in Bild 4.49 aufgebaut. Es kann in ein Spielzeug oder ein Spiel, in eine Blumenvase oder in ein Aquarium eingebaut werden. Bild 4.50 zeigt eine typische RGB-LED mit vier Beinchen. Das längste Beinchen 2 ist der gemeinsame Anschluss (je nach Typ Kathode oder Anode).

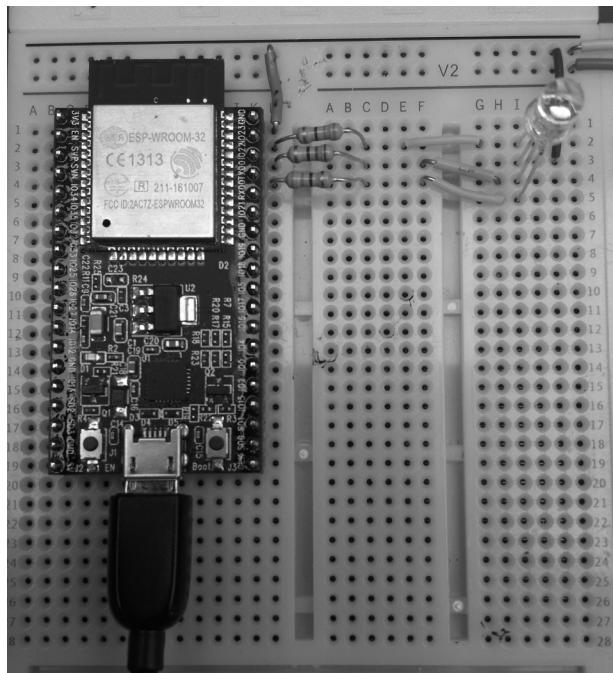


Bild 4.49 Das Projekt wurde auf einem Steckboard aufgebaut

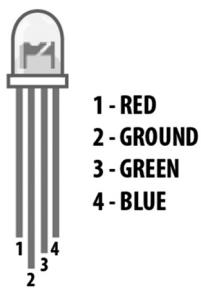


Bild 4.50 Eine typische RGB-LED mit gemeinsamer Kathode

#### 4.12.6 PDL des Projekts

Bild 4.51 zeigt die PDL des Projekts.

##### BEGIN

Define port numbers for Red, Green and Blue pins

Configure ports as outputs

##### DO FOREVER

Create a random number between 1 and 1 for Red

Create a random number between 1 and 1 for Green

Create a random number between 1 and 1 for Blue

Write Red value to Red port

Write Green value to Green port

```
        Write Blue value to Blue port
        Wait 500 ms
ENDDO
END
```

Bild 4.51 PDL des Projekts

#### 4.12.7 Programmlisting

Die Programmlisting des Projekts ist sehr einfach und ist in Bild 4.52 (Programm RGB) zu sehen.

```
/*********************************************
*
*           RGB LED
*
*=====
*
* In this program an RGB LED is connected to the ESP32-DevKitC
* where Red pin to GPIO23, Green pin to GPIO22, and the Blue pin
* to GPIO1. The LEDs are turned ON and OFF randomly.
*
*
* Program: RGB
* Date    : July, 2017
*****
#define RED 23
#define GREEN 22
#define BLUE 1
int R, G, B;

//
// Set GPIO pin 23, 22 and 1 as outputs
//
void setup()
{
    pinMode(RED, OUTPUT);
    pinMode(GREEN, OUTPUT);
    pinMode(BLUE, OUTPUT);
    randomSeed(10);
}

//
// Use the random number generator to generate ON (1) or OFF
// (0) values for the three colours
//
void loop()
{
    R = random(0, 2);
```

```
G = random(0, 2);
B = random(0, 2);

digitalWrite(RED, R);
digitalWrite(GREEN, G);
digitalWrite(BLUE, B);
delay(500);
}
```

Bild 4.52 Programmlisting

#### 4.12.8 Programmbeschreibung

Zu Beginn des Programms werden die GPIO-Pins definiert, an denen die LED-Pins für Rot, Grün und Blau angeschlossen sind, und anschließend als Ausgänge konfiguriert. Im Hauptteil des Programms, der Endlosschleife, werden zufällige Zahlen zwischen 0 und 1 erzeugt. Beachten Sie, dass bei der **random**-Funktion die untere Bereichsgrenze stets im Zufallszahlenraum eingeschlossen ist, die obere Bereichsgrenze dagegen nicht. Deshalb muss für die obere Grenze „2“ angegeben werden, damit am Ausgang entweder eine „0“ oder eine „1“ erscheint. Dann werden die erzeugten Zahlen für alle drei Farben an die entsprechenden Ports gesendet, die einzelnen Farben also entweder ein- oder ausgeschaltet.

#### 4.12.9 Vorschläge

Im Programm in Bild 4.52 ist die Verzögerungszeit auf 500 ms eingestellt. Versuchen Sie, diese Zeit zu ändern, und sehen Sie, was passiert.

### 4.13 Zusammenfassung

In diesem Kapitel haben wir die Entwicklung einiger grundlegender Projekte mit dem ESP32-DevKitC gesehen. Im nächsten Kapitel werden wir uns etwas komplexeren Projekten zuwenden.

## KAPITEL 5 • EINFÄCHE PROJEKTE MIT DER ARDUINO-IDE UND DEM ESP32-DEVKITC

### 5.1 Übersicht

Im letzten Kapitel haben wir einige Basisprojekte mit dem ESP32-DevKitC entwickelt. In diesem Kapitel kommen etwas komplexere Projekte mit der Arduino-IDE als Entwicklungs-Software an die Reihe.

Wie zuvor werden der Titel, die Beschreibung, das Ziel, das Blockdiagramm und so weiter bei allen Projekten angegeben.

### 5.2 PROJEKT 1 – Thermometer mit serielllem Monitor

#### 5.2.1 Beschreibung

In diesem Projekt wird ein analoger Temperatursensor-Chip verwendet, um die Umgebungstemperatur zu messen. Die Temperatur wird sekündlich mit dem seriellen Monitor der Arduino-IDE auf dem PC-Bildschirm angezeigt.

#### 5.2.2 Das Ziel

Das Projekt soll zeigen, wie ein analoger Eingang des ESP32-DevKitC verwendet werden kann, um analoge Daten zu lesen. Außerdem wird gezeigt, wie Daten auf dem PC-Monitor angezeigt werden.

#### 5.2.3 Blockschaltbild

Bild 5.1 zeigt das Blockdiagramm des Projekts.

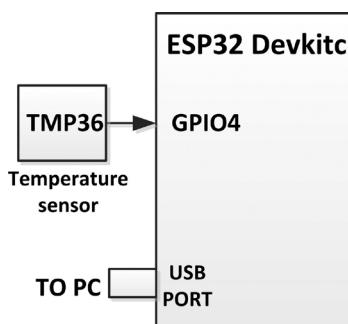


Bild 5.1 Blockdiagramm des Projekts

#### 5.2.4 Schaltplan

In diesem Projekt wird der analoge Temperatursensor TMP36 verwendet, um die Umgebungstemperatur zu messen. Der TMP36 besitzt drei Anschlüsse, deren Belegung in Bild 5.2 dargestellt ist. Der Sensor arbeitet mit einer Spannungsversorgung von 2,7 ... 5,5 V und kann Temperaturen im Bereich von -40 ... +125 °C messen. Die Ausgangsspannung des Gerätes ist proportional zur gemessenen Temperatur und folgt der Gleichung:

$$C = (V - 500) / 10$$

Wenn C die gemessene Temperatur in °C ist, ist V die Ausgangsspannung des Sensors in mV. So entspricht eine Ausgangsspannung von 800 mV einer Temperatur von 30 °C und so weiter.



PIN 1,  $+V_S$ ; PIN 2,  $V_{OUT}$ ; PIN 3, GND

Bild 5.2 Anschlussbelegung des Temperatursensor-Chips TMP36

Bild 5.3 zeigt den Schaltplan des Projekts. Zwei Pins des TMP36 sind an die Stromversorgung (+3,3 V und Masse) angeschlossen. Der Ausgangs-Pin des Temperatursensors TMP36 ist mit GPIO 4 verbunden, der auch der ADC2\_0 des ESP32-DevKitC ist (siehe Bild 4.2). Der A/D-Wandler ist 12 Bit breit und auf +3,3 V referenziert.

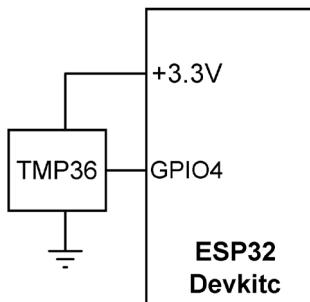


Bild 5.3 Schaltplan des Projekts

### 5.2.5 Konstruktion

Die ESP32-DevKitC-Platine wird wie in Bild 5.4 auf einer Steckplatine montiert. Der Ausgang des TMP36 ist an GPIO-Pin 4 angeschlossen.

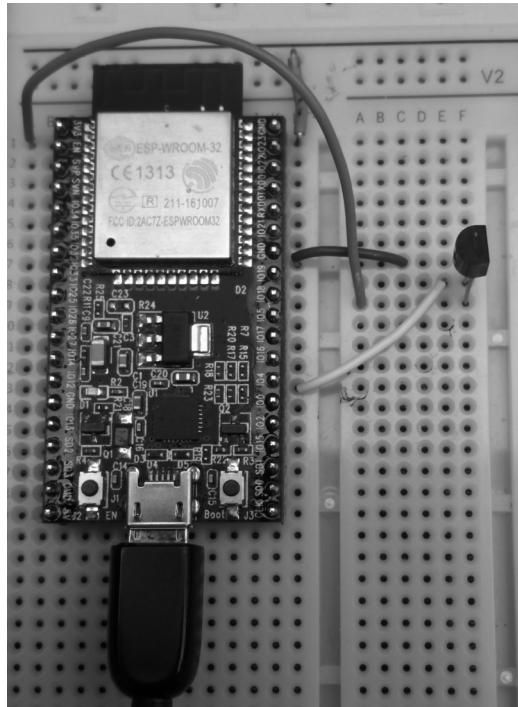


Bild 5.4 Das Projekt auf einem Steckboard

### 5.2.6 PDL des Projekts

Die PDL des Projekts ist in Bild 5.5 zu sehen. Das Programm läuft in einer Endlosschleife, in der jede Sekunde die Temperatur eingelesen und in Grad Celsius umgerechnet wird. Dann wird der Messwert vom seriellen Monitor zum PC-Bildschirm gesendet.

**BEGIN**

Define TMP36 as GPIO pin 4

Set the Serial Monitor baud rate to 9600

**DO FOREVER**

    Read the ambient temperature

    Convert the temperature into Degrees Centigrade

    Send the readings to Serial Monitor

    Wait 1 second

**ENDDO**

**END**

Bild 5.5 PDL des Projekts

### 5.2.7 Programmlisting

Das Listing des Programms TMP36 ist in Bild 5.6 dargestellt.

```

/*****
 *          THERMOMETER WITH SERIAL MONITOR
 * =====
 *
 * In this program a TMP36 type analog temperature sensor chip
 * is connected to GPIO pin 4 which is also an analog input.
 * The program reads the ambient temperature every second and
 * sends the readings to the Serial Monitor
 *
 *
 * Program: TMP36
 * Date   : July, 2017
 *****/
#include "dht.h"
dht DHT;

#define DHT11_PIN 7

void setup()
{
    Serial.begin(9600);
}

// 
// Read the ambient temperature, convert into Degrees Centigrade
// and send the readings to the Serial Monitor. Repeat this
// process every second
//
void loop()
{
    int rd = DHT.read11(DHT11_PIN);
    serial.print("Temperature = ");
    Serial.print(DHT.temperature);
    Serial.print("Humidity = ");
    Serial.println(DHT.humidity);
    delay(1000);
}

```

*Bild 5.6 Programmlisting*

### 5.2.8 Programmbeschreibung

Am Anfang des Programms wird der TMP36-Ausgang als Port GPIO 4 definiert, einer der analogen Eingänge des ESP32-DevKitC. Die Baudrate des seriellen Monitors wird auf 9600 eingestellt. Der Rest des Programms ist eine Endlosschleife, in der der analoge Ausgang des TMP36 mit der Bibliotheksfunktion **analogRead** ausgelesen und dann in Millivolt umgewandelt wird. Beachten Sie, dass die ADC-Referenzspannung 3300 mV beträgt und der

ADC 12 Bit auflöst (0 ... 4095). Der Messwert wird um 500 verringert und das Ergebnis durch 10 dividiert. Dies ergibt den Temperaturwert in Celsius. Die Temperaturwerte werden dann an den seriellen Monitor weitergeleitet und von diesem an den PC-Bildschirm gesendet. Klicken Sie dazu in der Arduino-IDE auf **Werkzeuge -> Serieller Monitor** und wählen Sie unten rechts eine Baudrate von 9600. Eine typische Ausgabe des seriellen Monitors zeigt Bild 5.7. Man kann die Anzeige benutzerfreundlicher gestalten, indem man zum Beispiel jeder Zeile den Text „**Temperature** =“ voranstellt. Der Code muss dazu wie folgt geändert werden. Das Ergebnis dieser Änderung ist in Bild 5.8 zu sehen.

```
float Temperatur = (mV - 500,0) / 10,0;  
int len = Serial.write („Temperature = „);  
Serial.println (Temperatur);
```

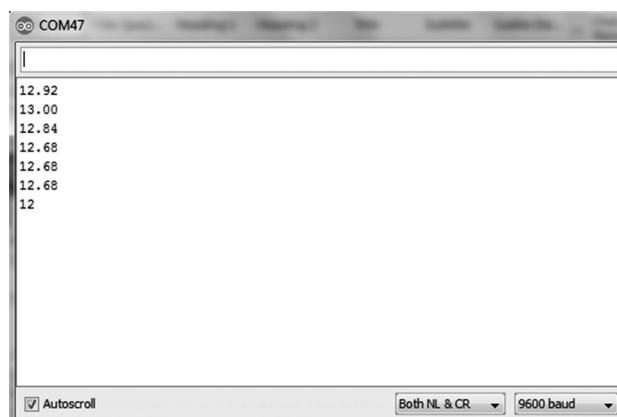


Bild 5.7 Typische Ausgabe des seriellen Monitors

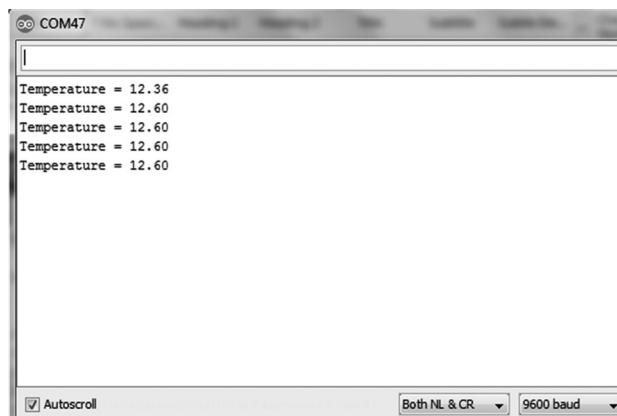


Bild 5.8 Modifizierte Ausgabe

An dieser Stelle ist es wichtig, zu erwähnen, dass das ESP32-DevKitC des Autors (und viele andere, wie im Internet berichtet wird) Probleme mit der Linearität des A/D-Wandlers hat, was zu ungenauen Messwerten führt. Um genaue Ergebnisse vom ADC zu erhalten, sollte

man in einem Experiment eine Tabelle oder eine Gleichung erarbeiten, die die tatsächliche Beziehung zwischen der analogen Eingangsspannung und den vom ADC zurückgegebenen digitalen Wert zeigt. Mit dieser Tabelle oder Gleichung können Sie anschließend die ADC-Werte korrigieren.

### 5.3 PROJEKT 2 – Temperatur und relative Luftfeuchtigkeit im seriellen Monitor

#### 5.3.1 Beschreibung

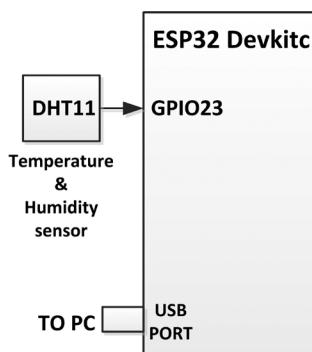
In diesem Projekt wird ein Sensor für Temperatur und relative Luftfeuchtigkeit mit der Bezeichnung DHT11 verwendet, um die Umgebungstemperatur und die relative Luftfeuchtigkeit zu ermitteln. Beide Messwerte werden sekündlich vom seriellen Monitor der Arduino-IDE auf dem PC-Bildschirm angezeigt.

#### 5.3.2 Das Ziel

Ziel dieses Projekts ist es, zu zeigen, wie der beliebte DHT11-Sensor für relative Feuchte und Temperatursensor in Projekten eingesetzt werden kann.

#### 5.3.3 Blockschaltbild

Bild 5.9 zeigt das Blockdiagramm des Projekts.



*Bild 5.9 Blockschaltung des Projekts*

#### 5.3.4 Schaltplan

In diesem Projekt wird der DHT11-Sensor für relative Feuchte und Temperatursensor verwendet. Der Standard-DHT11 ist in einem 4-poligen Gehäuse untergebracht, wobei lediglich drei Anschlüsse für die Versorgungsspannung (+V und GND) sowie als Datenausgang verwendet werden, wie Bild 5.10 zeigt. Der Daten-Pin muss über einen 10-kΩ-Widerstand auf +V hochgezogen werden. Im Chip arbeiten ein kapazitiver Feuchtesensor und ein Thermistor für die Umgebungstemperatur. Die gemessenen Daten sind nach rund einer Sekunde am Datenausgang verfügbar. Die grundlegenden Eigenschaften des DHT11 sind:

- Betriebsspannung 3 ... 5 V
- 2,5 mA Stromaufnahme (während einer Wandlung)
- Temperaturanzeige im Bereich 0 ... 50 °C mit einem Fehler von ±2 °C
- Feuchtigkeitsmesswert im Bereich 20 ... 80 % mit einem Fehler von 5 %

- Steckbrett-kompatibel durch 0,1-Zoll-Pinabstände

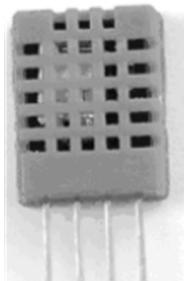


Bild 5.10 Der Standard-DHT11-Chip

In diesem Beispiel wird das bei Elektor erhältliche DHT11-Modul mit 10-k $\Omega$ -Pull-up-Widerstand verwendet. Das Modul hat anders als der Sensor selbst nur drei Anschlüsse, wie in Bild 5.11 dargestellt.

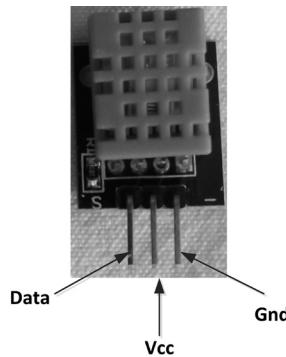


Bild 5.11 DHT11-Modul von Elektor

Bild 5.12 zeigt den Schaltplan des Projekts. Hier ist der Datenausgang des DHT11 mit GPIO 23 des ESP32-DevKitC verbunden.

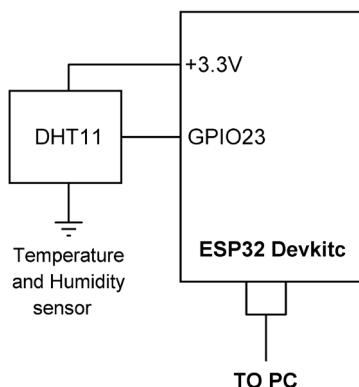


Bild 5.12 Schaltplan des Projekts

### 5.3.5 Konstruktion

Die ESP32-DevKitC-Platine wird auf ein Breadboard gesteckt (Bild 5.13) und das DHT11-Sensormodul an +3V, GND und GPIO23 angeschlossen.

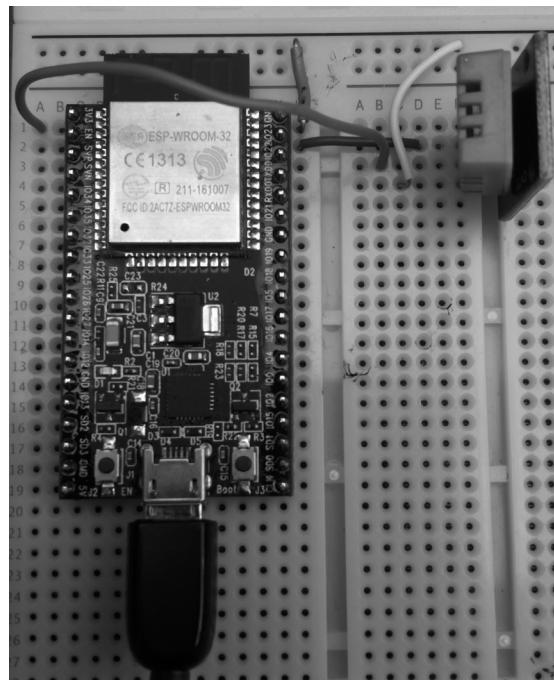


Bild 5.13 Das Projekt auf einem Steckbrett

### 5.3.6 PDL des Projekts

Die PDL des Projekts in Bild 5.14 zeigt eine Endlosschleife, in der jede Sekunde die Temperatur und die relative Luftfeuchtigkeit gelesen und die Messwerte an den seriellen Monitor gesendet werden.

#### BEGIN

```
Include the necessary header files
Define GPIO23 as the DHT11 port
Define the DHT type as DHT11
Set the Serial Monitor baud rate to 9600
Initialize the DHT library
```

#### DO FOREVER

```
Read the relative humidity
Read the ambient temperature
Display the relative humidity and temperature on Serial Monitor
Wait 1 second
```

#### ENDDO

#### END

Bild 5.14 PDL des Projekts

### 5.3.7 Programmlisting

Für den DHT11 muss die DHT-Sensorbibliothek installiert werden, bevor man den Sensor in einem Projekt mit dem ESP32-Prozessor verwenden kann. Die Schritte, um diese Bibliothek in der Arduino-IDE zu installieren, sind nachfolgend aufgeführt:

- Laden Sie die DHT11-Sensorbibliothek von <https://github.com/adafruit/DHT-sensor-library/archive/master.zip> herunter und speichern Sie die Datei **DHT-sensor-library-master.zip** in einem Ordner auf Ihrem PC.
- Entpacken Sie die Datei. Sie sollten einen Ordner mit dem Namen **DHT-sensor-library-master** erhalten.
- Benennen Sie den Ordner in **DHT** um.
- Verschieben Sie den Ordner **DHT** in den Ordner **Libraries** der Arduino-IDE-Ordner (siehe Bild 5.15).

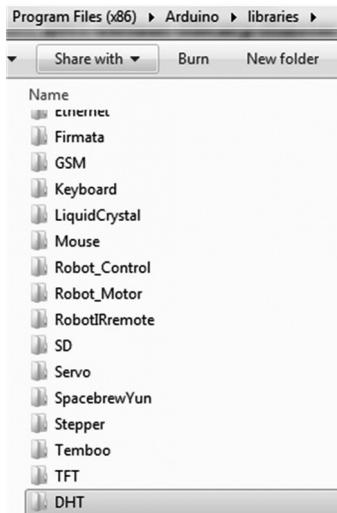


Bild 5.15: Ordner DHT in die Arduino-Bibliotheken verschieben

- Starten Sie die Arduino-IDE erneut.
- Jetzt müssen Sie die Adafruit Unified Sensor-Bibliothek installieren. Klicken Sie auf Sketch -> Bibliothek einbinden – > Bibliotheken verwalten
- Suchen Sie den Abschnitt Adafruit Unified Sensor (Bild 5.16) und klicken Sie auf Installieren, um diese Bibliothek zu installieren.

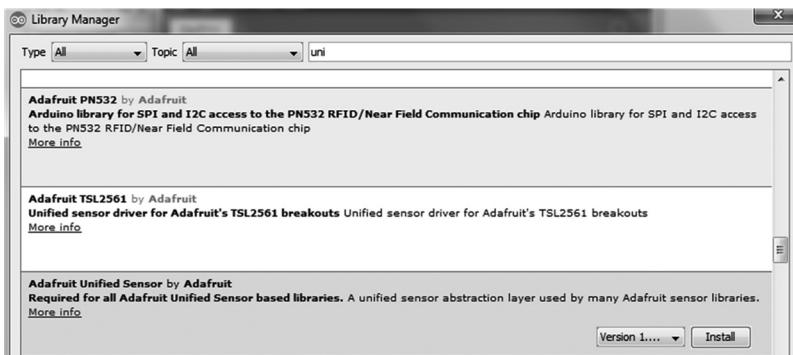


Bild 5.16: Installieren der Adafruit Unified Sensor-Bibliothek

Nach diesen Vorarbeiten kann man das Programm DHT11 entwickeln, um die relative Luftfeuchtigkeit und die Temperatur zu messen (Bild 5.17).

```
*****
*      RELATIVE HUMIDITY AND TEMPERATURE WITH SERIAL MONITOR
*      =====
*
* In this program a DHT11 type relative humidity and temperature
* sensor chip is connected to GPIO pin 23. The relative humidity
* and the ambient temperature are read every second and the
* readings are displayed on the Serial monitor.
*
* Program: DHT11
* Date   : July, 2017
*****
#define LDR 4
//
// Set the Serial Monitor baud rate to 9600 and initialize the
// DHT library
//
void setup()
{
    Serial.begin(9600);
}

//
// Read the relative humidity and the ambient temperature and
// display the results on the Serial Monitor
//
void loop()
{
    int ldr = analogRead(LDR);
```

```
    Serial.println(ldr);
    delay(1000);
}
```

Bild 5.17 Programmlisting

### 5.3.8 Programmbeschreibung

Im Programm werden zu Beginn die Bibliotheken **Adafruit\_Sensor.h** und **DHT.h** in das Programm eingebunden. Als GPIO-Pin, mit dem der DHT11 verbunden ist, wird Pin 23 definiert. Als DHT-Typ wird DHT11 angegeben. Dann wird die Baudrate des seriellen Monitors auf 9600 gesetzt und die DHT-Bibliothek initialisiert.

Der Rest des Programms ist eine Endlosschleife, in der die relative Luftfeuchtigkeit und die Umgebungstemperatur gelesen und die Messergebnisse durch den seriellen Monitor jede Sekunde angezeigt werden. Bild 5.18 zeigt, wie der serielle Monitor die Messwerte ausgibt.

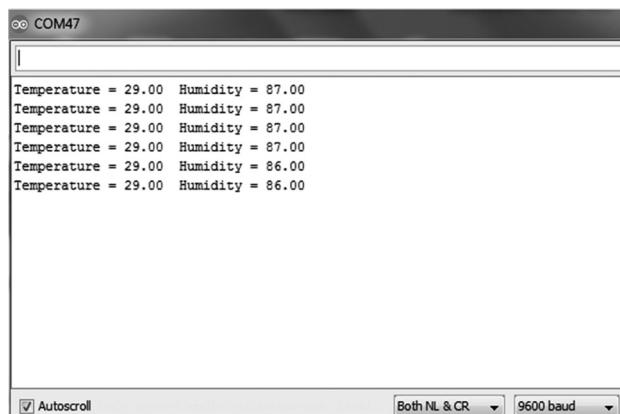


Bild 5.18 Zum Beispiel: Ausgabe des seriellen Monitors

## 5.4 PROJEKT 3 – Messung der Lichtstärke

### 5.4.1 Beschreibung

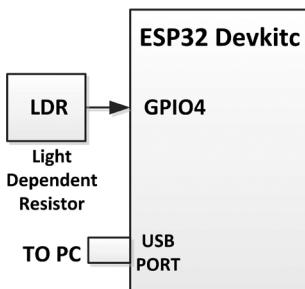
In diesem Projekt wird ein lichtabhängiger Widerstand (LDR) eingesetzt, um die Lichtstärke zu ermitteln und jede Sekunde mit dem seriellen Monitor anzuzeigen.

### 5.4.2 Das Ziel

Das Projekt soll zeigen, wie ein LDR in ESP32-basierten Schaltungen verwendet werden kann.

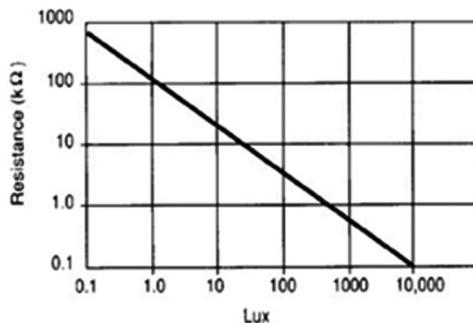
### 5.4.3 Blockschaltbild

Bild 5.19 zeigt die Blockschaltung des Projekts.

*Bild 5.19 Blockdiagramm des Projekts*

#### 5.4.4 Schaltplan

Ein LDR ist ein einfacher Widerstand (Bild 5.20), dessen Widerstandswert mit zunehmend einfallendem Licht abnimmt. Diese Bauteile werden normalerweise bei Lichtsteueranwendungen eingesetzt. Bild 5.21 zeigt die typische Kennlinie eines typischen LDRs. Wie Sie sehen, nimmt der Widerstand des LDRs ab, wenn die Lichtintensität erhöht wird.

*Bild 5.20 Ein typischer lichtabhängiger Widerstand**Bild 5.21 Kennlinie eines typischen LDR*

Der Schaltplan des Projekts in Bild 5.22 zeigt, dass ein Anschluss des LDRs mit GND (0 V), der andere an den Analogeingang GPIO4 des ESP32-DevKitC und gleichzeitig über einen 10-kΩ-Widerstand mit +3,3 V (3 V) verbunden ist. Beachten Sie, dass LDRs wie andere Widerstände keine Polarität aufweisen.

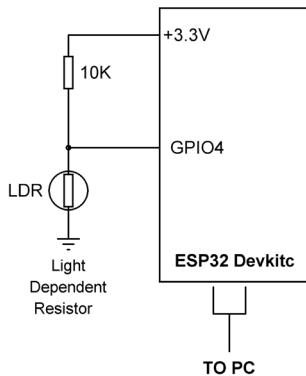


Bild 5.22 Schaltplan des Projekts

#### 5.4.5 Konstruktion

Das ESP32-DevKitC auf einem Steckboard ist, wie in Bild 5.23 gezeigt, über einen 10-kΩ-Widerstand mit einem LDR verbunden.

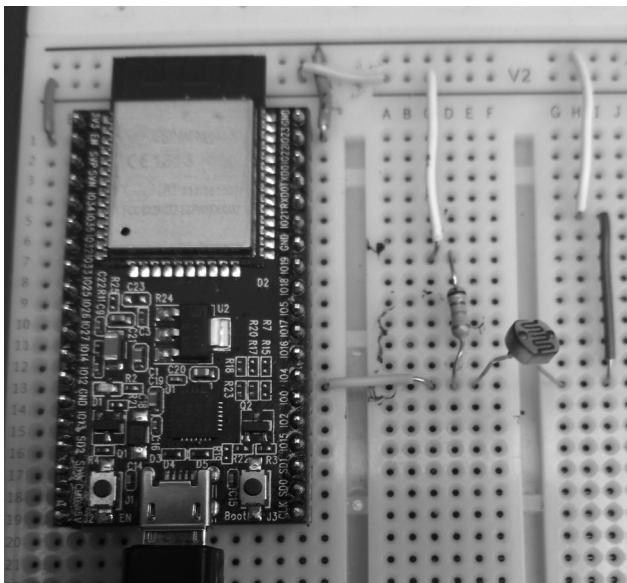


Bild 5.23 Projekt auf einem Experimentier-Board

#### 5.4.6 PDL des Projekts

Die PDL des Projekts ist in Bild 5.24 dargestellt. Das Programm läuft in einer Endlosschleife, in der die Umgebungshelligkeit gemessen und über den seriellen Monitor angezeigt wird.

##### BEGIN

Define LDR as GPIO4

Set the Serial Monitor baud rate to 9600

DO FOREVER

```

        Read the voltage across the LDR through the ADC
        Display the reading on the Serial Monitor
        Wait 1 second
ENDDO
END

```

*Bild 5.24 PDL des Projekts*

#### 5.4.7 Programmlisting

Das Programm LDR ist sehr einfach und in Bild 5.25 dargestellt.

```

/*****
*      MEASURING THE LIGHT LEVEL
*      =====
*
* In this program a light dependent resistor is connected
* to analog input GPIO4 of the ESP32-DevKitC. The program
* measures and then displays the light level through the
* Serial Monitor. The resistance of the LDR increases in
* dark and as a result a higher reading is obtained.
*
*
* Program: LDR
* Date   : July, 2017
*****/
#define LDR 4
//
// Set the Serial Monitor baud rate to 9600
//
void setup()
{
    Serial.begin(9600);
}

//
// Read the light level and display through the Serial Monitor
//
void loop()
{
    int ldr = analogRead(LDR);
    Serial.println(ldr);
    delay(1000);
}

```

*Bild 5.25 Programmlisting*

### 5.4.8 Programmbeschreibung

Zu Beginn des Programms wird der LDR dem analogen Eingangs-Port GPIO4 zugeordnet. Innerhalb des Hauptprogramms wird die Spannung über den LDR vom A/D-Wandler gelesen und auf dem seriellen Monitor angezeigt. Dabei wird der vom A/D-Wandler ermittelte Binärwert und nicht die tatsächliche physikalische Spannung am LDR angezeigt.

Bei normalen Lichtverhältnissen liegen diese Binärwerte bei einigen Hundert. Im Dunkeln nimmt der Widerstand des LDR zu, so dass der Messwert auf über 2000 steigt. Das Programm sollte vor einem ernsthaften Einsatz mit einem kommerziellen Belichtungsmesser kalibriert werden.

Bild 5.26 zeigt eine typische Ausgabe des Monitors.



Bild 5.26: Typischer Ausgang am seriellen Monitor

### 5.4.9 Vorschläge

Anstelle des 10-k $\Omega$ -Festwiderstands kann ein variabler Widerstand (Trimmpotentiometer) verwendet werden, um die Schaltung empfindlicher für andere Lichtstärken zu machen.

## 5.5 PROJEKT 4 – Dunkelheitserkennung

### 5.5.1 Beschreibung

In diesem Projekt wird wie im vorherigen Projekt ein lichtabhängiger Widerstand (LDR) verwendet. Hier sind zusätzlich ein aktiver Summer und eine LED an die GPIO-Pins 23 beziehungsweise 22 angeschlossen. Das Projekt erkennt, wenn der LDR Dunkelheit „misst“, und aktiviert daraufhin den Summer und die LED.

### 5.5.2 Das Ziel

Ziel dieses Projekts ist es, zu zeigen, wie eine einfache Dunkelheiterkennung mit dem ESP32-DevKitC entworfen werden kann.

### 5.5.3 Blockschaltbild

Bild 5.27 zeigt das Blockdiagramm des Projekts.

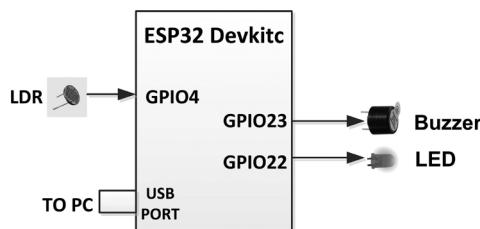


Bild 5.27 Blockschaltung des Projekts

#### 5.5.4 Schaltplan

Der Schaltplan des Projekts ist in Bild 5.28 zu sehen.

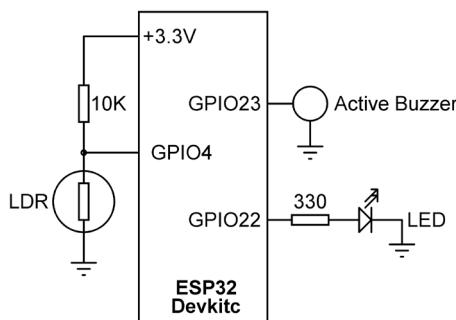


Bild 5.28 Schaltplan des Projekts

#### 5.5.5 Konstruktion

Das ESP32-DevKitC wird auf einem Steckbrett montiert, wie in Bild 5.29 gezeigt, und mit dem LDR, einem 10-kΩ-Widerstand und einem aktiven Summer ausgestattet.

Stellen Sie sicher, dass der aktive Summer nicht mehr als 5 mA benötigt und mit +3,3 V betrieben werden kann. Wenn Ihr Summer mehr als 5 mA benötigt, sollte ein Transistor-Schalter den Ausgang vor Überlastung schützen

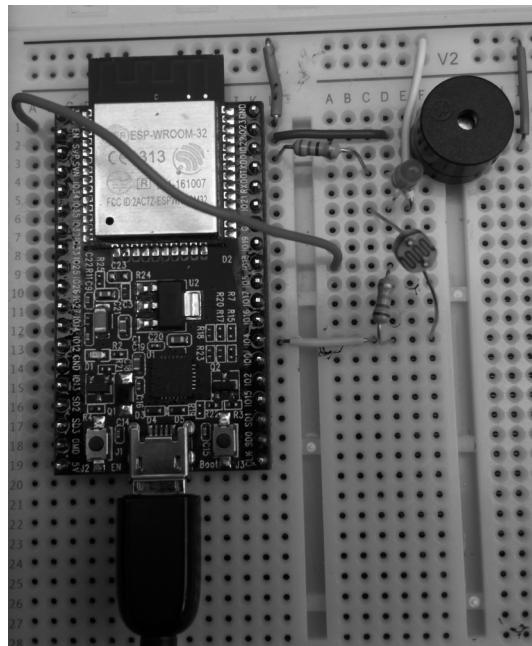


Bild 5.29 Das Projekt auf einem Steckboard

### 5.5.6 PDL des Projekts

Die PDL des Projekts in Bild 5.30 zeigt eine Endlosschleife, die den Umgebungslichtpegel misst. Wenn es dunkel ist (der LDR-Wert muss über 2000 liegen), werden sowohl Summer als auch LED eingeschaltet.

```
BEGIN
    Define LDR as GPIO04
    Define BUZZER as GPIO023
    Define LED as GPIO022
    Configure GPIO022 and GPIO023 as outputs
    Turn OFF both BUZZER and LED to start with
    Set the Serial Monitor baud rate to 9600
    DO FOREVER
        Read the voltage across the LDR through the ADC
        IF reading > 2000 THEN
            Turn ON BUZZER
            Turn ON LED
        ELSE
            Turn OFF BUZZER
            Turn OFF LED
        ENDIF
    ENDDO
END
```

Bild 5.30 PDL des Projekts

### 5.5.7 Programmlisting

Das Programm LDRALARM ist sehr einfach und sein Listing in Bild 5.31 dargestellt.

```
*****  
*          DARKNESS REMINDER  
*          =====  
*  
* In this program a light dependent resistor is connected  
* to analog input GPIO4 of the ESP32-DevKitC. In addition,  
* a buzzer and an LED are connected to GPIO ports 23 and 22  
* respectively. Normally both the buzzer and the LED are OFF  
* and they turn ON when it becomes dark. Darkness is assumed  
* when the LDR reading is above 2000.  
*  
*  
* Program: LDRALARM  
* Date   : July, 2017  
*****/  
#define LDR 4  
#define BUZZER 23  
#define LED 22  
  
//  
// Configure both the BUZZER and the LED as outputs  
//  
void setup()  
{  
    pinMode(BUZZER, OUTPUT);  
    pinMode(LED, OUTPUT);  
    digitalWrite(BUZZER, LOW);  
    digitalWrite(LED, LOW);  
}  
  
//  
// Read the light level and turn on both the BUZZER and the LED  
// when it becomes dark. In this project the darkness has been  
// assumed when the LDR reading is above 2000  
//  
void loop()  
{  
    int ldr = analogRead(LDR);  
    if(ldr > 2000)  
    {  
        digitalWrite(BUZZER, HIGH);  
        digitalWrite(LED, HIGH);  
    }  
}
```

```
    }
else
{
    digitalWrite(BUZZER, LOW);
    digitalWrite(LED, LOW);
}
}
```

Bild 5.31 Programmlisting

### 5.5.8 Programmbeschreibung

Am Anfang des Programms wird der LDR dem GPIO-Port 4, der Summer (BUZZER) dem Port 23 und die LED dem Port 22 zugeordnet. Die digitalen Ports 23 und 22 werden als Ausgänge eingestellt. Innerhalb des Hauptprogramms wird die Spannung über dem LDR gelesen und der Wert in der Integer-Variablen **ldr** gespeichert. Wenn **ldr** größer als 2000 ist (siehe vorheriges Projekt), wird Dunkelheit angenommen. In diesem Fall werden sowohl der Summer als auch die LED eingeschaltet, um Finsternis zu signalisieren.

### 5.5.9 Vorschläge

In diesem Projekt werden ein Summer und eine LED als Ausgabeinstrumente verwendet. Es ist möglich, ein Relais an einen Ausgang des ESP32-DevKitC anzuschließen und bei einsetzender Dunkelheit zum Beispiel die Raumbeleuchtung automatisch einzuschalten.

## 5.6 PROJEKT 5 – LED-Würfel

### 5.6.1 Beschreibung

Dies ist ein einfaches Würfelprojekt mit LEDs und einem Drucktaster. Die LEDs sind wie die Punkte eines echten Würfels angeordnet. Wenn man die Taste drückt, wird eine Zufallszahl zwischen 1 und 6 erzeugt und mit den LEDs angezeigt. Nach drei Sekunden verlöschen die LEDs wieder. In diesem Zustand, wenn alle LEDs ausgeschaltet sind, ist das System bereit, eine neue Zahl zu würfeln.

### 5.6.2 Das Ziel

In diesem Projekt soll ein LED-basierter Würfel entworfen werden.

### 5.6.3 Blockdiagramm:

Bild 5.32 zeigt die Blockschaltung des Projekts.

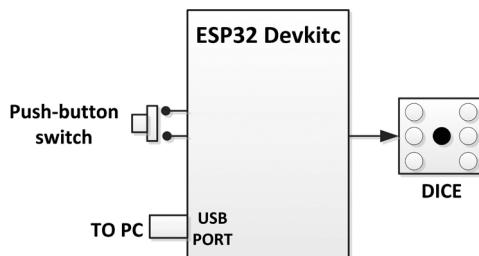


Bild 5.32 Blockdiagramm des Projekts

Bild 5.33 zeigt, dass die LEDs so angeordnet sind, dass sie Zahlen wie bei einem realen Würfel anzeigen. Die Arbeitsweise des Projekts: Normalerweise sind alle LEDs ausgeschaltet und zeigen damit an, dass das System bereit für einen neuen „Wurf“ ist. Dies geschieht durch einen Druck auf den Taster. Es wird eine Zufallszahl zwischen 1 und 6 erzeugt und drei Sekunden lang mit den LEDs angezeigt. Nach diesen drei Sekunden erlöschen alle LEDs wieder.

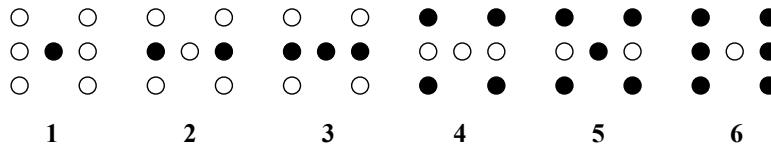


Bild 5.33 LED-Würfel

#### 5.6.4 Schaltplan

Der Schaltplan des Projekts ist in Bild 5.34 zu sehen. Die sieben LEDs sind an den GPIO-Pins 23, 22, 1, 3, 21, 19 und 18 jeweils über einen 330- $\Omega$ -Strombegrenzungswiderstand angeschlossen. Die Zuordnung ist wie folgt:

LED	GPIO Pin
D1	23
D2	1
D3	19
D4	21
D5	22
D6	3
D7	18

Der Taster ist über einen 10-k $\Omega$ -Pull-up-Widerstand mit dem GPIO-Pin 5 verbunden, so dass normalerweise der Tastenausgang logisch 1 ist und beim Druck auf die Taste auf logisch 0 geht.

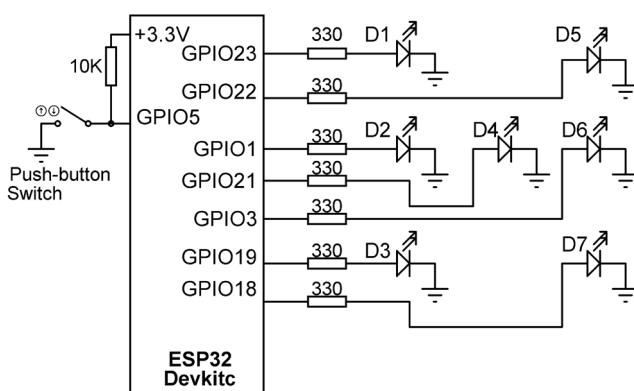


Bild 5.34 Schaltplan des Projekts

### 5.6.5 Konstruktion

Die ESP32-DevKitC-Platine ist mit den sieben LEDs und dem Drucktaster auf einem Steckboard montiert, wie in Bild 5.35 zu sehen.

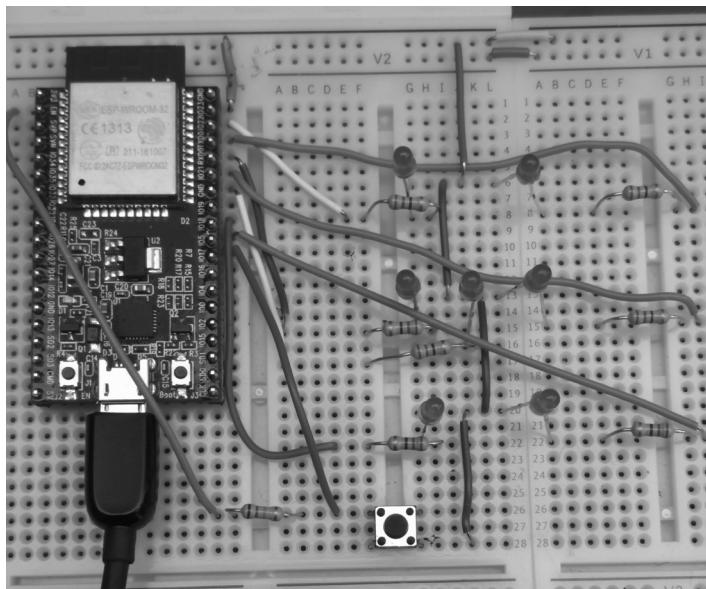


Bild 5.35 Das Würfelprojekt auf einem Steckboard

### 5.6.6 PDL des Projekts

Die PDL des Projekts in Bild 5.36 dargestellt. Tabelle 5.1 zeigt die Beziehung zwischen der gewürfelten Zahl und den einzuschaltenden LEDs. Soll zum Beispiel die Zahl 1 angezeigt werden, muss man nur LED D1 einschalten. In ähnlicher Weise müssen zur Anzeige der Zahl 3 die LEDs D2, D4, D6 eingeschaltet sein und so weiter.

Gewürfelte Zahl	LEDs, die eingeschaltet werden sollen
1	D4
2	D2, D6
3	D2, D4, D6
4	D1, D3, D5, D7
5	D1, D3, D4, D5, D7
6	D1, D2, D3, D5, D6, D7

Tabelle 5.1 Würfelnahlen und LEDs, die eingeschaltet werden sollen

#### BEGIN

- Assign Button to GPIO0
- Assign LEDs to the GPIOs
- Configure all LEDs as outputs
- Configure Button as input

```

DO FOREVER
    Turn OFF all LEDs
    Wait until the Button is pressed
    Generate a random number between 1 and 6
    Turn ON the appropriate LED to display the number
    Wait for 3 seconds
ENDDO
END

```

*Bild 5.36 PDL des Projekts*

### 5.6.7 Programmlisting

Das Listing des Programms DICE ist in Bild 5.37 zu sehen.

```

/*****
*          LED DICE
*          =====
*
* In this program 7 LEDs are connected in to the ESP32-DevKitC
* to simulate the faces of a dice. In addition, a push-button
* switch is used. When the switch is pressed, a random number
* is generated between 1 and 6 and the LEDs are turned ON to
* indicate the number as if it is a real dice. The LEDs display
* the number for 3 seconds and then turn OFF to indicate that
* the system is ready so that the user can press the button
* again if desired.
*
* The connections of the LEDs and the push-button switch are
* as follows:
*
* D1: GPIO23, D2: GPIO1, D3: GPIO19, D4: GPIO21, D5: GPIO22
* D6: GPIO3,   D7: GPIO18
* Push button switch: GPIO5
*
*
* Program: DICE
* Date   : July, 2017
*****
#define Button 5
#define D1 23
#define D5 22
#define D2 1
#define D6 3
#define D4 21
#define D3 19
#define D7 18
int LEDs[] = {23, 22, 1, 3, 21, 19, 18};

```

```
//  
// Configure the LEDs as outputs, and the Button as input  
//  
void setup()  
{  
    for(int i = 0; i < 7; i++)  
    {  
        pinMode(LEDs[i], OUTPUT);  
    }  
    pinMode(Button, INPUT);  
    randomSeed(10);  
  
}  
  
//  
// Turn all LEDs OFF  
//  
void ALL_OFF()  
{  
    for(int i = 0; i < 7; i++)  
    {  
        digitalWrite(LEDs[i], LOW);  
    }  
}  
  
//  
// Turn OFF all LEDS so that the system is ready to accept a  
// new button press. Wait until the Button is pressed. Then,  
// generate a random number between 1 and 6. Turn ON the  
// appropriate LEDs to display the number as a real dice face.  
// The program displays the number for 3 seconds and then all  
// the LEDs are turned OFF, ready for the next button press.  
//  
void loop()  
{  
    ALL_OFF();                                // Turn OFF all LEDs  
    while(digitalRead(Button) == 1);           // Wait for Button pressed  
    int dice = random(1, 7);                  // Generate a random number  
  
    switch (dice)  
    {  
        case 1:                               // Number 1  
            digitalWrite(D4, HIGH);           // Turn ON D4  
            break;  
        case 2:                               // Number 2
```

```

        digitalWrite(D2, HIGH);           // Turn ON D2,D6
        digitalWrite(D6, HIGH);
        break;
    case 3:                         // Number 3
        digitalWrite(D2, HIGH);       // Turn ON D2,D4,D6
        digitalWrite(D4, HIGH);
        digitalWrite(D6, HIGH);
        break;
    case 4:                         // Number 4
        digitalWrite(D1, HIGH);       // Turn ON D1,D3,D5,D7
        digitalWrite(D3, HIGH);
        digitalWrite(D5, HIGH);
        digitalWrite(D7, HIGH);
        break;
    case 5:                         // Number 5
        digitalWrite(D1, HIGH);       // Turn ON D1,D3,D4,D5,D7
        digitalWrite(D3, HIGH);
        digitalWrite(D4, HIGH);
        digitalWrite(D5, HIGH);
        digitalWrite(D7, HIGH);
        break;
    case 6:                         // Number 6
        digitalWrite(D1, HIGH);       // Turn ON D1,D2,D3,D5,D6,D7
        digitalWrite(D2, HIGH);
        digitalWrite(D3, HIGH);
        digitalWrite(D5, HIGH);
        digitalWrite(D6, HIGH);
        digitalWrite(D7, HIGH);
        break;
    }
    delay(3000);                   // Display for 3 seconds
}

```

*Bild 5.37 Programmlisting*

### 5.6.8 Programmbeschreibung

Zu Beginn des Programms wird Button dem GPIO-Port 5 zugewiesen, ebenso die LEDs D1, D2, D3, D4, D5, D6 den GPIO-Ports 23, 1, 19, 21, 22, 3 beziehungsweise 18. Alle LEDs werden als Ausgänge konfiguriert, die Taste Button als Eingang.

Das Hauptprogramm wird in einer Endlosschleife ausgeführt. In dieser Schleife werden alle LEDs ausgeschaltet und das Programm wartet, bis die Taste gedrückt wird. Dann wird eine Zufallszahl zwischen 1 und 6 erzeugt. Eine **switch**-Anweisung sorgt dann dafür, dass die der erzeugten Zufallszahl entsprechenden LEDs eingeschaltet werden. Das Programm zeigt die generierte Zahl für drei Sekunden auf dem Würfel an und springt dann zurück zum Anfang der Schleife.

Bild 5.38 zeigt ein Beispiel für die Anzeige der Zahl 6.



Bild 5.38 Anzeige der Zahl 6

## 5.7 PROJEKT 6 – Logiktester

### 5.7.1 Beschreibung

Dieses Projekt, ein einfacher Logiktester (Logic Probe), wird verwendet, um den logischen Zustand eines unbekannten digitalen Signals anzuzeigen. In einer typischen Anwendung wird eine Testleitung (Sonde) verwendet, um das unbekannte Signal zu erkennen, und zwei unterschiedliche Farb-LEDs, um den logischen Zustand anzuzeigen. Wenn das Signal zum Beispiel logisch 0 ist, dann leuchtet die rote LED, ist es dagegen logisch 1, wird die grüne LED eingeschaltet.

### 5.7.2 Das Ziel

Das Ziel dieses Projekts ist es, zu zeigen, wie ein Logiktester entworfen werden kann, um den logischen Zustand eines angelegten logischen Signals zu ermitteln.

### 5.7.3 Blockschaltbild

Bild 5.39 zeigt das Blockdiagramm des Projekts. RED LED und GREEN LED sind an die GPIO-Ports 23 und 22 angeschlossen. Das zu testende Logiksignal wird an den GPIO-Port 1 angelegt.

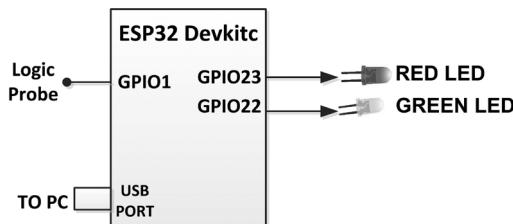


Bild 5.39 Blockdiagramm des Projekts

### 5.7.4 Schaltplan

Der Schaltplan des Projekts ist in Bild 5.40 zu sehen. Die LEDs sind über  $330\text{-}\Omega$ -Strombegrenzungswiderstände an den GPIO-Pins angeschlossen. Das zu testende Logiksignal wird direkt zu GPIO-Port 1 geführt.

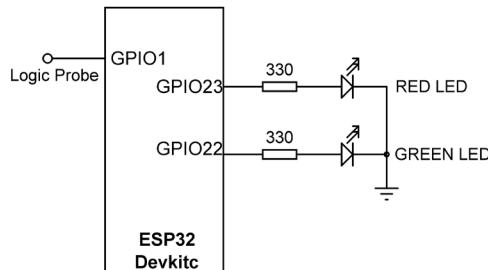


Bild 5.40 Schaltplan des Projekts

### 5.7.5 Konstruktion

Das ESP32-DevKitC wird mit den beiden LEDs auf einem Steckboard montiert, wie in Bild 5.41 dargestellt.

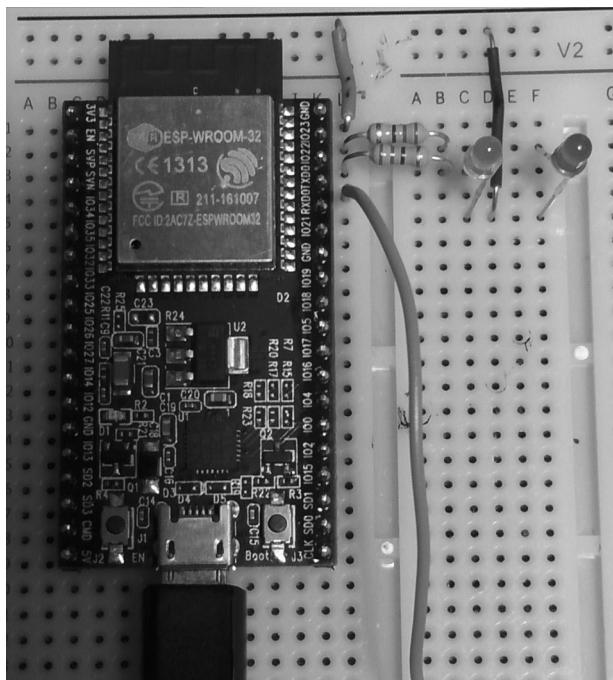


Bild 5.41 Das auf einem Breadboard aufgebaute Projekt

### 5.7.6 PDL des Projekts

Die PDL des Projekts ist in Bild 5.42 dargestellt. Wenn der externe logische Zustand 0 ist, wird die RED LED ein- und die GREEN LED ausgeschaltet. Wenn der externe logische Zu-

stand andererseits 1 ist, dann wird die GREEN LED ein- und die RED LED ausgeschaltet.

```
BEGIN
    Assign RED to GPIO port 23
    Assign GREEN to GPIO port 22
    Assign Probe to GPIO port 1
    Configure GPIO ports 22,23 as outputs
    Configure GPIO port 1 as input
    DO FOREVER
        IF Probe is logic 0 THEN
            Turn ON RED LED
            Turn Off GREEN LED
        ELSE
            Turn ON GREEN LED
            Turn OFF RED LED
        ENDIF
    ENDDO
END
```

*Bild 5.42 PDL des Projekts*

### 5.7.7 Programmlisting

Das Listing des Programms Probe ist in Bild 5.43 dargestellt.

```
/*****
 *          LOGIC PROBE
 *          =====
 *
 * This is a logic probe program. Two LEDs, one RED and one
 * GREEN are connected to GPIO ports 23 and 22 respectively.
 * The program determines the state of the logic signal applied
 * to GPIO port 5. If the applied logic level is 0 then the
 * RED LED is turned ON. If on the other hand the applied
 * logic level is 1 then the GREEN LED is turned ON.
 *
 *
 * Program: Probe
 * Date   : July, 2017
 *****/
#define Probe 1
#define RED 23
#define GREEN 22

//
// Configure the LEDs as outputs, and the Probe as input
//
void setup()
```

```

{
    pinMode(RED, OUTPUT);
    pinMode(GREEN, OUTPUT);
    pinMode(Probe, INPUT);
}

// Examine the state of the applied external logic signal. If
// the state is 0, then turn ON the RED LED. if on the other
// hand the external applied logic state is 1, then turn ON the
// GREEN LED
//
void loop()
{
    if(digitalRead(Probe) == 0) // Logic state is 0
    {
        digitalWrite(RED, HIGH); // RED ON
        digitalWrite(GREEN, LOW); // GREEN OFF
    }
    else // Logic state 1
    {
        digitalWrite(GREEN, HIGH); // GREEN ON
        digitalWrite(RED, LOW); // RED OFF
    }
}

```

*Bild 5.43 Programmliste*

### **5.7.8 Programmbeschreibung**

Zu Beginn des Programms wird der Logik-Tastkopf dem GPIO-Port 1 zugeordnet, dem GPIO-Port 23 die RED LED und dem GPIO-Port 22 die GREEN LED. Die LED-Ports werden danach als Ausgänge und Probe als Eingang konfiguriert.

Der Rest des Programms wird in einer Endlosschleife ausgeführt. Innerhalb dieser Schleife wird der Zustand des externen Logiksignals gelesen, und wenn es logisch 0 ist, wird die RED LED, andernfalls die GREEN LED eingeschaltet. Dieser Vorgang wird kontinuierlich wiederholt.

### **5.7.9 Geändertes Programm**

Das in Bild 5.43 dargestellte Programm Probe lässt eine der LEDs immer eingeschaltet, selbst wenn die Sonde nicht mit einem externen digitalen Signal verbunden, also zum Beispiel hochimpedant ist. Das Projekt lässt sich weiterentwickeln, so dass auch ein hochohmiger Zustand erfasst wird. Wenn kein Logikpegel anliegt, wird dann auch keine der LEDs eingeschaltet.

Bild 5.44 zeigt das modifizierte Schaltbild. Beachten Sie hier, dass ein Transistor (BC108 oder ein anderer npn-Transistor) vorgeschaltet ist.

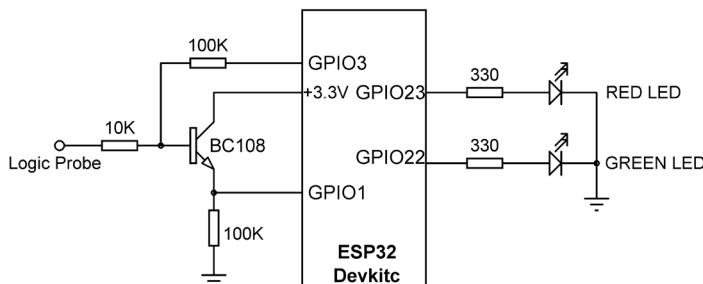


Bild 5.44 Modifizierter Schaltplan

Die Schaltung funktioniert wie folgt: Der Transistor ist als Emitterfolger geschaltet, wobei die Basis mit dem als Ausgang konfigurierten GPIO-Port 3 verbunden ist. Das externe Logiksignal wird über einen 10-k $\Omega$ -Widerstand an die Basis des Transistors angelegt. Der Emitter des Transistors ist mit GPIO-Port 1 verbunden, der als Eingang konfiguriert ist. Der GPIO-Port 3 legt einen logischen Pegel an die Basis des Transistors, während GPIO-Port 1 den Zustand des externen Signals bestimmt, wie in Tabelle 5.2 gezeigt. Wenn zum Beispiel, nachdem **port 3 = 1** gesetzt wurde, **port 1 = 1** ist und auch nachdem **port 3 = 0** gesetzt wurde, **port 1 = 0** ist, erkennen wir, dass das logische Signal am Eingang logisch 1 sein muss.

Logik-Tastkopf	Ausgangsspeigel an GPIO3	von GPIO1 erkannt
hohe Impedanz	1	1
	0	0
Logisch 1	1	1
	0	1
Logisch 0	1	0
	0	0

Tabelle 5.2 Angelegte und erfasste Logikpegel

Bild 5.45 zeigt das geänderte Programm Probe2. Zu Beginn des Programms wird Probe dem GPIO1, RED und GREEN GPIO23 und GPIO22 sowie CHECK dem GPIO3 zugewiesen. Die LEDs und CHECK werden dann als digitale Ausgänge, Probe als digitaler Eingang konfiguriert.

Der Rest des Programms wird in einer Endlosschleife ausgeführt und arbeitet logisch wie in Tabelle 5.2, um die LEDs zu aktivieren/deaktivieren.

```
/*
 *          LOGIC PROBE
 *          =====
 *
 * This is a logic probe program. Two LEDs, one RED and one
```

```
* GREEN are connected to GPIO ports 23 and 22 respectively.  
* The program determines the state of the logic signal applied  
* to GPIO port 5. If the applied logic level is 0 then the  
* RED LED is turned ON. If on the other hand the applied  
* logic level is 1 then the GREEN LED is turned ON.  
*  
* This modified program detects the high impedance state when  
* there is no logic level applied at the input  
*  
*  
* Program: Probe2  
* Date : July, 2017  
*****  
#define Probe 1  
#define RED 23  
#define GREEN 22  
#define CHECK 3  
  
//  
// Configure the LEDs as outputs, and the Probe as input  
//  
void setup()  
{  
    pinMode(RED, OUTPUT);  
    pinMode(GREEN, OUTPUT);  
    pinMode(CHECK, OUTPUT);  
    pinMode(Probe, INPUT);  
}  
  
//  
// Examine the state of the applied external logic signal. If  
// there is nothing connected at the input then assume high  
// impedance state and turn OFF both LEDs. If on the other  
// hand the state is 0, then turn ON the RED LED. if on the  
// other hand the external applied logic state is 1, then  
// turn ON the GREEN LED  
//  
void loop()  
{  
    digitalWrite(CHECK, HIGH);           // Set CHECK = 1  
    delay(1);  
    if(digitalRead(Probe) == 1)  
    {  
        digitalWrite(CHECK, LOW);  
        delay(1);  
        if(digitalRead(Probe) == 0)
```

```
{  
    digitalWrite(RED, LOW);           // High impedance state  
    digitalWrite(GREEN, LOW);  
}  
else  
{  
    digitalWrite(RED, LOW);  
    digitalWrite(GREEN, HIGH);  
}  
}  
else  
{  
    digitalWrite(CHECK, LOW);  
    delay(1);  
    if(digitalRead(Probe) == 0)  
    {  
        digitalWrite(GREEN, LOW);  
        digitalWrite(RED, HIGH);  
    }  
}  
}  
}
```

Bild 5.45 Geändertes Programm

Die modifizierte Schaltung auf dem Breadboard ist in Bild 5.46 dargestellt.

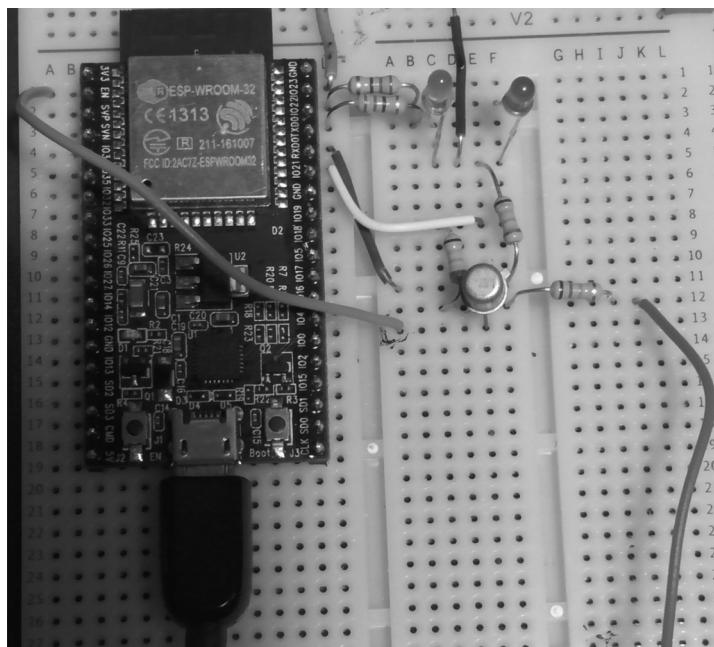


Bild 5.46 Modifizierte Schaltung auf einem Steckbrett

## 5.8 PROJEKT 7 – Zähler mit 7-Segment-LED-Anzeige

### 5.8.1 Beschreibung

Dieses einfache Projekt beschreibt einen Zähler mit einem 7-Segment-LED-Display, der im Sekundentakt kontinuierlich von 0 bis 9 zählt.

7-Segment-Anzeigen werden häufig in elektronischen Schaltungen verwendet, um numerische oder alphanumerische Werte anzuzeigen. Eine 7-Segment-Anzeige wie in Bild 5.47 besteht im Wesentlichen aus sieben LEDs, die so geformt und angeschlossen sind, dass die Ziffern von 0 bis 9 (und einige Buchstaben) angezeigt werden können. Die einzelnen Segmente werden üblicherweise so, wie in Bild 5.48 gezeigt, durch Buchstaben von **a** bis **g** gekennzeichnet.

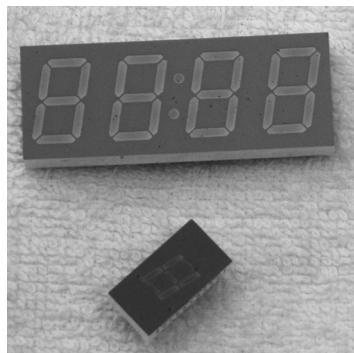


Bild 5.47 Einige 7-Segment-Anzeigen

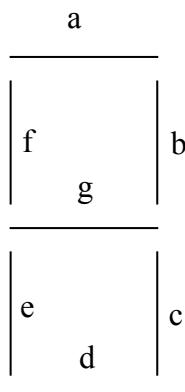


Bild 5.48 Segmente einer 7-Segment-Anzeige

Bild 5.49 zeigt, wie die Zahlen von 0 bis 9 durch Ein- und Ausschalten verschiedener Display-Segmente dargestellt werden.

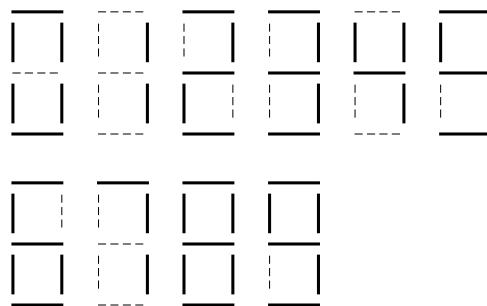


Bild 5.49 Anzeige der Ziffern 0 bis 9

Es gibt 7-Segment-Anzeigen mit gemeinsamer Kathode (**common cathode**) oder gemeinsamer Anode (**common anode**). Bei einer Anordnung mit gemeinsamer Kathode wie in Bild 5.50 sind die Kathoden aller Segment-LEDs mit Masse verbunden. Die einzelnen Segmente werden dann durch Anlegen einer logischen 1 (jeweils über Strombegrenzungswiderstände) eingeschaltet.

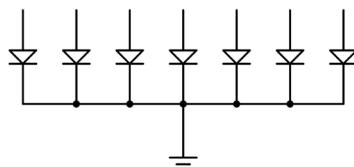


Bild 5.50 7-Segment-Anzeige mit gemeinsamer Kathode

Bei Displays mit gemeinsamer Anode sind die Anodenanschlüsse aller LEDs wie in Bild 5.51 miteinander verbunden. Diese verbundenen Anoden werden normalerweise an die Versorgungsspannung angeschlossen. Ein Segment wird durch Verbinden seines Kathodenanschlusses (über einen Strombegrenzungswiderstand) mit logisch 0 eingeschaltet.

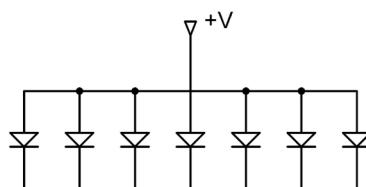


Bild 5.51 7-Segment-Anzeige mit gemeinsamer Anode

In diesem Projekt wird eine 7-Segment-Anzeige mit gemeinsamer Kathode mit der Bezeichnung **SMA42056** verwendet. Dies ist ein 14,20 mm (0,56 Zoll) hohes Display, das aus einer Entfernung von bis zu sieben Metern gut abgelesen werden kann. Die LED-Vorwärtsspannung beträgt ungefähr 2 V. Das Display besitzt zehn Anschlüsse, darunter auch einer für den Dezimalpunkt. Tabelle 5.3 zeigt die Pin-Belegung dieser Anzeige, wobei die Anschlussnummerierung in der unteren linken Ecke des Displays mit Pin 1 beginnt (siehe Bild 5.52). Die untere rechte Ecke ist Pin 5, die obere linke Ecke Pin 10.

Tabelle 5.3 SMA42056 7-Segment-LED-Pin-Konfiguration

Pinnummer	Segment
1	E
2	D
3	gemeinsame Kathode
4	C
5	Dezimalpunkt dp
6	B
7	A
8	gemeinsame Kathode
9	F
10	G

Bild 5.52 Pin-Belegung des SMA42056-Displays

### 5.8.2 Das Ziel

Das Projekt soll zeigen, wie eine 7-Segment-LED-Anzeige an einem ESP32-DevKitC angeschlossen und verwendet werden kann.

### 5.8.3 Blockschaltbild

Das Blockdiagramm des Projekts ist in Bild 5.53 dargestellt.

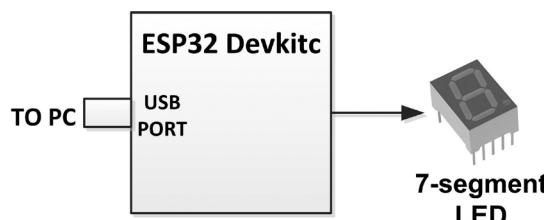


Bild 5.53 Blockschaltung des Projekts

### 5.8.4 Schaltplan

Die Schaltung in Bild 5.54 zeigt, dass die Segmente der LED-Anzeige über  $330\text{-}\Omega$ -Strombegrenzungswiderstände an GPIO Pins angeschlossen sind. Dabei sind die einzelnen Segmente folgenden GPIO-Pins zugeordnet:

Segment	GPIO-Pin
a	23
b	22
c	1
d	3

e	21
f	19
g	18

Bevor wir mit dem Display arbeiten, müssen wir die Beziehung zwischen den anzuzeigenden Ziffern und den GPIO-Pins kennen, die dafür HIGH sein sollten.

Anzuzeigende Ziffer	GPIO-Pins auf HIGH
0	23,22,1,3,21,19
1	22,1
2	23,22,18,21,3
3	23,22,18,1,3
4	19,18,22,1
5	23,19,18,1,3
6	19,18,1,3,21
7	23,22,1
8	23,22,1,3,21,19,18
9	23,22,19,18,1

Bild 5.54 Schaltplan des Projekts

### 5.8.5 Aufbau

Das ESP32-DevKitC wird zusammen mit der 7-Segment-LED-Anzeige und den Strombegrenzungswiderständen auf einem Steckbrett montiert (Bild 5.55).

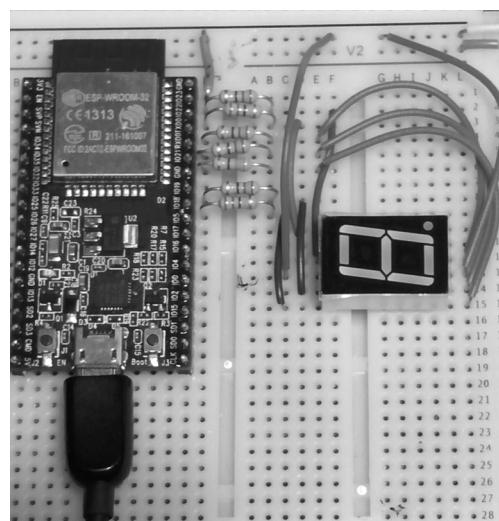


Bild 5.55 Das auf einem Steckboard aufgebaute Projekt

### 5.8.6 PDL des Projekts

Die PDL des Projekts ist in Bild 5.56 dargestellt. Auf der 7-Segment-Anzeige werden Zahlen von 0 bis 9 angezeigt, wobei zwischen jeder Zählung eine Sekunde vergeht.

```

BEGIN
    Declare array Num and store segments to be turned ON for every number
    Declare array Strt and store starting index of each number within Num
    Declare array LEDs and store all the GPIO port numbers used
    Set Count to 0
    Configure all used GPIO ports as outputs
DO FOREVER
    Display Count
    Wait one second
    Increment Count
    Turn OFF all segments
    IF Count = 10 THEN
        Count = 0
    ENDIF
ENDDO
END

```

*Bild 5.56 PDL des Projekts*

### 5.8.7 Programmlisting

Das Programm SevenSegment ist in Bild 5.57 zu sehen.

```

/*
*          7-SEGMENT LED COUNTER
*=====
*
* In this project a 7-segment LED is connected to the ESP32
* Devkitc. The program counts up from 0 to 9 with one second
* intervals. The connections between the GPIO pins and the
* 7-segment LED segments are as follows:
*
* Segment      GPIO pin
*   a          23
*   b          22
*   c           1
*   d           3
*   e          21
*   f          19
*   g          18
*
*
* Program: SevenSegment
* Date   : July, 2017

```

```
*****  
int Num[] = {6, 23, 22, 1, 3, 21, 19,           //0  
             2, 22, 1,                      //1  
             5, 23, 22, 18, 21, 3,          //2  
             5, 23, 22, 18, 1, 3,          //3  
             4, 19, 18, 22, 1,            //4  
             5, 23, 19, 18, 1, 3,          //5  
             5, 19, 18, 1, 3, 21,          //6  
             3, 23, 22, 1,                //7  
             7, 23, 22, 1, 3, 21, 19, 18, //8  
             5, 23, 22, 19, 18, 1};       //9  
  
int Strt[] = {0, 7, 10, 16, 22, 27, 33, 39, 43, 51};  
int LEDs[] = {23, 22, 1, 3, 21, 19, 18};  
int Count = 0;  
  
//  
// Configure the LEDs as outputs, and also turn OFF all segments  
//  
void setup()  
{  
    for(int i = 0; i < 7; i++)  
    {  
        pinMode(LEDs[i], OUTPUT);  
        ALL_OFF();  
    }  
}  
  
//  
// Turn OFF all segments  
//  
void ALL_OFF()  
{  
    for(int i = 0; i < 8; i++)  
    {  
        digitalWrite(LEDs[i], LOW);  
    }  
}  
//  
// This function display a given number N on the 7-segment display  
//  
void Display(int N)  
{  
    int i, k;  
    k = Strt[N];                      // Starting index  
    i = Num[k];                        // Length  
    for(int m = k + 1; m < k + 1 + i; m++)
```

```

    {
        digitalWrite(Num[m], HIGH);
    }
}

// Increment variable Count by 1 and display on the 7-segment
// display every second
//
//
void loop()
{
    Display(Count);
    delay(1000);
    ALL_OFF();
    Count++;
    if(Count == 10) Count = 0;
}

```

*Bild 5.57 Programmlisting*

### 5.8.8 Programmbeschreibung

Am Anfang des Programms werden drei Arrays deklariert: **Num**, **Strt** und **LEDs**. Das Array **Num** speichert die einzuschaltenden Segmente beziehungsweise ihre GPIO-Port-Nummern, wobei jede Zeile einer anzuzeigenden Ziffer entspricht. Die erste Zahl in jeder Zeile bezeichnet die Anzahl der Segmente, die für die gegebene Ziffer eingeschaltet werden müssen. So lautet zum Beispiel die Zeile für die Ziffer 3:

5,23,22,18,1,3, //3

Die erste Zahl 5 ist die Anzahl der einzuschaltenden Segmente, die mit den GPIO-Ports 23, 22, 18, 1 und 3 verbunden sind.

Das zweite Array **Strt** speichert den Startindex innerhalb des Arrays **Num** für jede anzuzeigende Ziffer. Beispielsweise ist der Startindex der Ziffer 3 = 16, weil das 16. Element des Arrays **Num** die Anzahl der Segmente bezeichnet, die wie oben beschrieben für die Darstellung der Ziffer 3 eingeschaltet werden müssen.

Das Array **LEDs** speichert die GPIO-Port-Adressen aller LEDs. Dieses Array wird verwendet, um alle Ports als Ausgänge zu konfigurieren und alle Segmente bei Bedarf auszuschalten. Die Funktion **Display** empfängt die Ziffer als Argument, die angezeigt werden soll. Diese Funktion ermittelt die einzuschaltenden Segmente und schaltet sie mit der Anweisung **digitalWrite** ein.

Das Hauptprogramm wird in einer Endlosschleife ausgeführt, in der der aktuelle Wert der Variablen **Count** angezeigt wird. Nach einer Verzögerung von einer Sekunde werden alle Segmente ausgeschaltet und die nächste Ziffer berechnet. Der Zählerstand **Count** wird zudem um 1 erhöht und auf 0 zurückgesetzt, wenn er 10 erreicht.

### 5.8.9 Geändertes Programm

Das in Bild 5.57 dargestellte Programm kann vereinfacht und benutzerfreundlicher gestaltet werden, wenn man die GPIO-Ports als 8-Bit-Byte gruppiert. Dann kann man auf alle Port-Bits zugreifen, indem man ein Byte an diesen Port sendet. Diese Technik wurde schon in Bild 4.22 in Kapitel 4 verwendet.

Die geänderte Programmlisting (SevenSegment2) ist in Bild 5.58 dargestellt. Die GPIO-Port-Bits sind hier wie folgt gruppiert:

MSB                            LSB  
X 23 22 1 3 21 19 18

Das MSB X wird nicht verwendet. Die Beziehung zwischen den anzuzeigenden Ziffern und den an den gruppierten Port zu sendenden Zahlen kann man wie folgt ableiten (X wird als 0 angenommen):

Ziffer	zu aktivierende Port-Bits	Zu sendende Zahl (in Hex)
	X 23 22 1 3 21 19 18	
0	x 1 1 1 1 1 1 0	7E
1	x 0 1 1 0 0 0 0	30
2	x 1 1 0 1 1 0 1	6D
3	x 1 1 1 1 0 0 1	79
4	x 0 1 1 0 0 1 1	33
5	x 1 0 1 1 0 1 1	5B
6	x 0 0 1 1 1 1 1	1F
7	x 1 1 1 0 0 0 0	70
8	x 1 1 1 1 1 1 1	7F
9	x 1 1 1 0 0 1 1	73

```
/*
 *           7-SEGMENT LED COUNTER
 * =====
 *
 * In this project a 7-segment LED is connected to the ESP32
 * Devkitc. The program counts up from 0 to 9 with one second
 * intervals. The connections between the GPIO pins and the
 * 7-segment LED segments are as follows:
 *
 * Segment   GPIO pin
 * a          23
 * b          22
 * c          1
 * d          3
```

```
*      e          21
*      f          19
*      g          18
*
*
* Program: SevenSegment2
* Date   : July, 2017
*****/*/
#include <Wire.h>
//#include <LCD.h>
#include <LiquidCrystal_I2C.h>

int Num[] = {0x7E, 0x30, 0x6D, 0x79, 0x33, 0x5B, 0x1F, 0x70, 0x7F, 0x73};
int LEDs[] = {23, 22, 1, 3, 21, 19, 18};
int Ports[] = {0, 23, 22, 1, 3, 21, 19, 18};
int Count = 0;
LiquidCrystal_I2C lcd(0x3F,16,2);

//
// Configure the LEDs as outputs
//
void setup()
{
    for(int i = 0; i < 7; i++)
    {
        pinMode(LEDs[i], OUTPUT);
    }
}

//
// Turn ON the appropriate LED
//
void Display(int No, unsigned char L)
{
    unsigned char j, m, i;
    m = L - 1;
    for(i = 0; i < L; i++)
    {
        j = pow(2, m);
        digitalWrite(Ports[i], (No & j));
        m--;
    }
}

//
```

```
// Increment variable Count by 1 and display on the 7-segment
// display every second
//
//
void loop()
{
    Display(Num[Count], 8);
    delay(1000);
    Count++;
    if(Count == 10)Count = 0;
}
```

Bild 5.58 Geändertes Programm

## 5.9 PROJEKT 8 – Klatschschalter

### 5.9.1 Beschreibung

In diesem Projekt werden ein Sound-Sensormodul (Small Microphone Module) und eine LED verwendet. Die LED ändert ihren Zustand (das heißt, sie wird ein- und ausgeschaltet), wenn das Modul ein Geräusch auffängt, zum Beispiel ein Händeklatschen. Wenn man in der Nähe des Mikrofons in die Hände klatscht, wird die LED eingeschaltet, wenn sie vorher dunkel war, und ausgeschaltet, wenn sie vorher leuchtete. In diesem Projekt wird eine LED verwendet, aber es gibt keinen Grund, warum sie nicht durch ein Relais ersetzt werden könnte, um damit zum Beispiel Lampen oder Garagentore zu steuern.

### 5.9.2 Das Ziel

Ziel dieses Projekts ist es, ein Sound-Sensormodul an den ESP32-DevKitC anzuschließen und es zur Steuerung eines externen Geräts wie einer LED, eines Relais oder eines anderen „logisch“ arbeitenden Bauteils zu verwenden.

### 5.9.3 Blockschaltbild

In diesem Projekt wird das kleine Mikrofonmodul in Bild 5.59 verwendet, das im Elektor-Shop erhältlich ist. Es handelt sich um ein 4-poliges Modul mit folgender Anschlussbelegung:

<b>A0</b>	Analogausgang
<b>G</b>	Versorgungsmasse
<b>+</b>	Spannungsversorgung (V+)
<b>D0</b>	Digitaler Ausgang

Der Analogausgang wird in diesem Projekt nicht verwendet. Ein kleines Potentiometer stellt den Auslösepunkt (die Empfindlichkeit) des digitalen Ausgangs so ein, dass er normalerweise auf logisch LOW gesetzt ist und auf logisch HIGH geht, wenn ein Geräusch vom Mikrofon erkannt wird.

Das Blockdiagramm in Bild 5.60 zeigt, dass GPIO-Pin 23 für das Sound-Sensormodul und GPIO-Pin 22 für die LED verwendet werden.



Bild 5.59 Kleines Mikrofonmodul

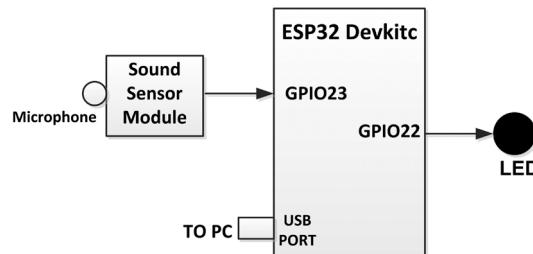


Bild 5.60 Blockschaltbild des Projekts

#### 5.9.4 Schaltplan

Das Schaltbild des Projekts ist in Bild 5.61 dargestellt. Der digitale Ausgang D0 des Sound-Sensormoduls ist an GPIO-Port 23 des ESP32-DevKitC angeschlossen. Die LED ist über einen  $330\text{-}\Omega$ -Strombegrenzungswiderstand mit GPIO-Port 22 verbunden.

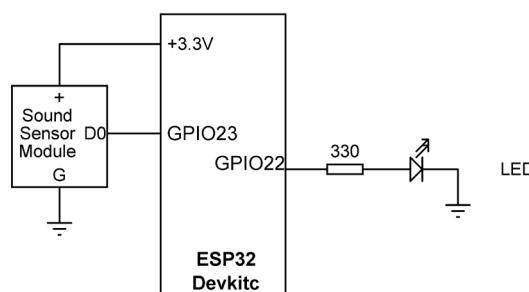


Bild 5.61 Schaltplan des Projekts

#### 5.9.5 Konstruktion

Das ESP32-DevKitC wird auf ein Breadboard gesteckt und mit dem Sound-Sensormodul und der daran angeschlossenen LED verbunden (siehe Bild 5.62).

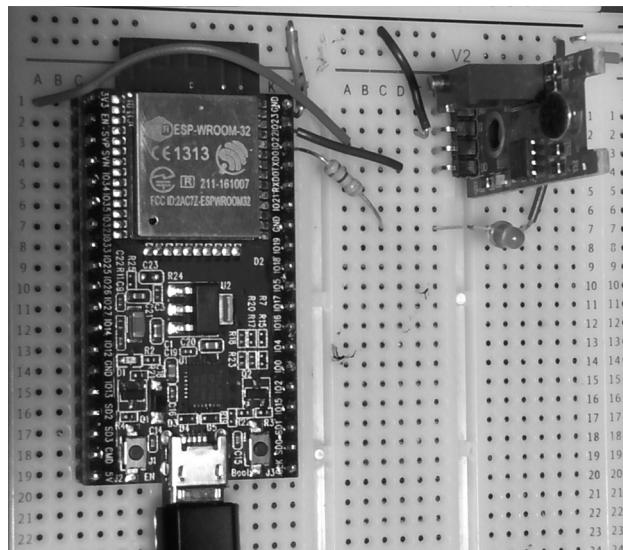


Bild 5.62 Das auf einem Steckbrett aufgebaute Projekt

### 5.9.6 PDL des Projekts

Die PDL des Projekts ist in Bild 5.63 dargestellt.

#### BEGIN

```
Assign GPIO port 22 to the LED  
Assign GPIO port 23 to the sound sensor module  
Configure GPIO22 as output  
Configure GPIO23 as input
```

#### DO FOREVER

```
    IF output of sound sensor module = 1  
        Toggle the LED  
        Wait 500 ms
```

#### ENDIF

#### ENDDO

#### END

Bild 5.63 PDL des Projekts

### 5.9.7 Programmlisting

Das Listing des Programms Sound ist in Bild 5.64 dargestellt.

```
*****  
* CLAP ON - CLAP OFF  
* ======  
*  
* In this project a sound sensor module (called the Small  
* Microphone Module) is connected to GPIO port 23 of the  
* ESP32-DevKitC. In addition, an LED is connected to GPIO
```

```
* port 22. The LED is normally OFF and is turned ON when
* sound is detected by the sensor (e.g. by clapping hands).
* The LED is toggled every time sound is detected by the
* sensor.
*
*****
#define LED 22
#define SoundSensor 23
int state = 0;

//
// Configure GPIO ports LED as output and SoundSensor as input
//
void setup ()
{
    pinMode (LED, OUTPUT);
    pinMode(SoundSensor, INPUT);
    digitalWrite(LED, LOW);
}

//
// Toggle the LED when sound is detected by the sensor
//
void loop ()
{
    if(digitalRead(SoundSensor) == 1)
    {
        digitalWrite(LED, (state) ? HIGH : LOW);
        state = !state;
        delay(500);
    }
}
```

Bild 5.64 Programmlisting

### 5.9.8 Programmbeschreibung

Zu Beginn des Programms werden GPIO 23 dem Sound-Sensormodul und GPIO 22 der LED zugeordnet. Der LED-Port wird als Ausgang, der Sound-Sensor-Port als Eingang konfiguriert.

Das Hauptprogramm läuft in einer Endlosschleife. In dieser Schleife wird der Ausgang des Sound-Sensors gelesen. Wenn der Ausgang logisch 1 ist, also ein Ton erkannt wird, wird der Zustand der LED umgeschaltet. Dieser Vorgang wird nach 500 Millisekunden wiederholt.

## 5.10 PROJEKT 9 – LCD „Hello from ESP32“

### 5.10.1 Beschreibung

Dieses einfache LCD-Projekt zeigt den Text **Hello from ESP32** in der ersten Zeile eines 16x2-Zeichen-LCDs.

### 5.10.2 Das Ziel

Dieses Projekt soll zeigen, wie ein alphanumerisches LCD an einem ESP32-DevKitC-Projekt angeschlossen und verwendet werden kann. In diesem Projekt wird ein zweizeiliges I<sup>2</sup>C-kompatibles LCD mit 16 Zeichen pro Zeile verwendet.

### 5.10.3 Blockschaltbild

Das LCD in diesem Projekt besitzt eine I<sup>2</sup>C-Schnittstelle. I<sup>2</sup>C ist ein Single-Ended-Bus, an dem bis zu 1008 Slaves und mehrere Master angeschlossen werden können. I<sup>2</sup>C erlaubt die Verbindung langsamer Peripheriegeräte mit einem Mikrocontroller. Der Bus besteht – abgesehen von der Spannungsversorgung – nur aus den beiden Drähten SDA und SCL, wobei SDA die Datenleitung und SCL die Takteleitung ist. Beide Leitungen müssen durch geeignete Widerstände auf die Versorgungsspannung gezogen werden. Das Taktsignal wird immer vom Master erzeugt. Master und Slaves am I<sup>2</sup>C-Bus können mit 100 kHz oder 400 kHz miteinander kommunizieren.

Die I<sup>2</sup>C-Bus-Schnittstelle des ESP32-DevKitC liegt auf den GPIO-Ports 21 und 22, wobei Port 21 die Datenleitung (SDA) und Port 22 die Takteleitung (SCL) ist.

Bild 5.65 zeigt Vorder- und Rückseite des I<sup>2</sup>C-LCDs. Beachten Sie die kleine Platine an der Rückseite des LCDs, die für die I<sup>2</sup>C-Schnittstelle sorgt. An dem kleinen Potentiometer auf dieser Platine wird auch der LCD-Kontrast eingestellt. Am Jumper daneben wird die Hintergrundbeleuchtung bei Bedarf deaktiviert. Das Blockdiagramm des Projekts ist in Bild 5.66 dargestellt.



Bild 5.65 LCD mit I<sup>2</sup>C-Schnittstelle (Vorder- und Rückansicht)

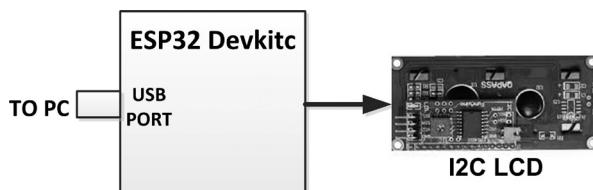


Bild 5.66 Blockdiagramm des Projekts

### 5.10.4 Schaltplan

Das Schaltbild des Projekts in Bild 5.67 zeigt, dass das I<sup>2</sup>C-LCD über vier Drähte mit dem Mikrocontroller verbunden ist. Zu den beiden Bus-Leitungen kommt noch die Versorgungsspannung, die +5 V am VCC-Pin (und nicht +3,3 V!) betragen muss. Die +5-V-Spannung steht am Pin links unten des ESP32-DevKitC zur Verfügung.

<b>GND</b>	Masse
<b>VCC</b>	+5-V-Versorgung
<b>SDA</b>	I <sup>2</sup> C-Datenleitung
<b>SCL</b>	I <sup>2</sup> C-Taktleitung

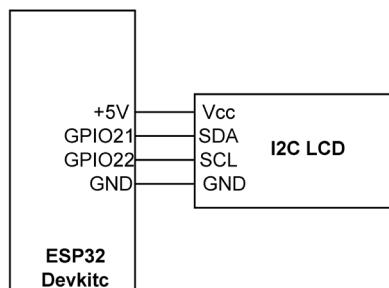


Bild 5.67 Schaltplan des Projekts

### 5.10.5 Konstruktion

Die ESP32-DevKitC-Platine auf dem Steckbrett wird über Jumperkabel mit dem I<sup>2</sup>C-LCD verbunden (siehe Bild 5.68).

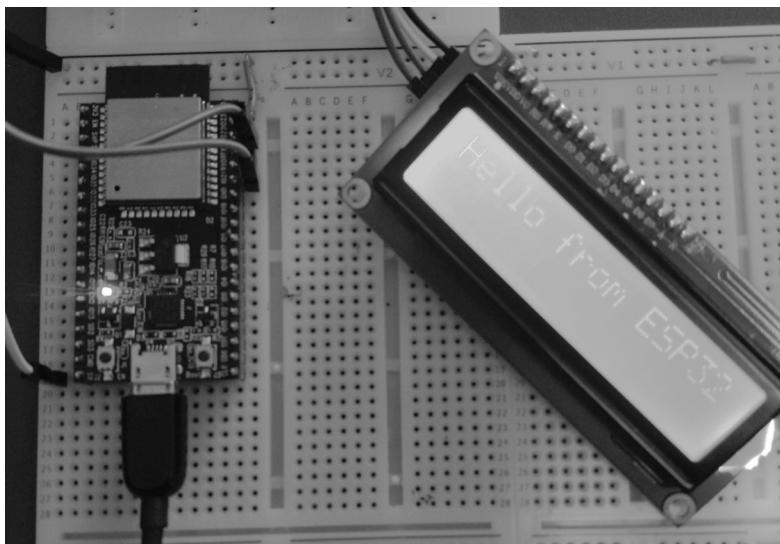


Bild 5.68 Das auf einem Steckbrett aufgebaute Projekt

### 5.10.6 PDL des Projekts

Bevor Sie den ESP32 für das I<sup>2</sup>C-LCD programmieren, müssen Sie die I<sup>2</sup>C-LCD-Bibliothek herunterladen und in den Arduino-IDE-Ordner einbinden. Dies erledigt man folgendermaßen:

- Erstellen Sie einen Ordner namens **LiquidCrystal\_I2C** unter dem Ordner „libraries“ der Arduino-IDE (siehe Bild 5.69).

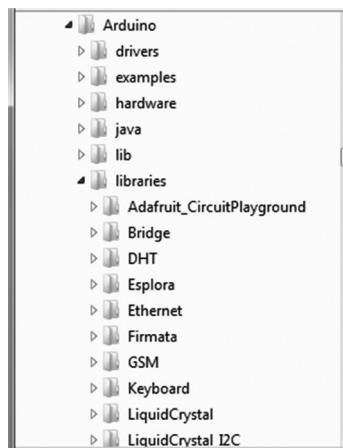


Bild 5.69 Ordner „LiquidCrystal\_I2C“ erstellen

- Gehen Sie zum folgenden Link, um die I<sup>2</sup>C-LCD-Bibliothek herunterzuladen: <https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library>
- Klicken Sie auf die Schaltfläche **Clone** oder **Download** (siehe Bild 5.70) und kopieren Sie alle Dateien in den Ordner **LiquidCrystal\_I2C**, wie es in Bild 5.71 gezeigt wird.

Branch: master		New pull request	Find file	Clone or download
	joao Pedroso committed with fdebrabander Update LiquidCrystal_I2C.cpp		Latest commit e3701fb on 9 Mar	
	examples Remove the call to backlight(), to demonstrate it is on by default.		6 years ago	
	LiquidCrystal_I2C.cpp Update LiquidCrystal_I2C.cpp		5 months ago	
	LiquidCrystal_I2C.h Update LiquidCrystal_I2C.h		5 months ago	
	README.md Explained how to install the library and include it in your project.		6 years ago	
	keywords.txt Initial commit.		6 years ago	

Bild 5.70 Laden Sie die I<sup>2</sup>C-Bibliotheksdateien herunter

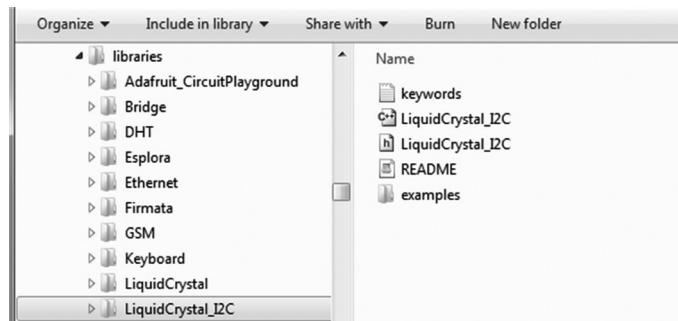


Bild 5.71 Dateien im Ordner LiquidCrystal\_I2C

Starten Sie die Arduino-IDE. Gehen Sie zu **Datei -> Beispiele**. Dort sollten Sie im Drop-down-Menü Beispiele für LiquidCrystal\_I2C sehen, wenn die Bibliothek korrekt installiert wurde.

Die kurze PDL des Projekts ist in Bild 5.72 zu sehen.

```
BEGIN
Include libraries Wire and LiquidCrystal_I2C
Set I2C LCD address and configuration
Initialize I2C LCD
Turn ON backlight
Display text Hello from ESP32
END
```

Bild 5.72 PDL des Projekts

### 5.10.7 Programmlisting

Das Listing des Programms LCD ist in Bild 5.73 dargestellt.

```
/*
 *          LCD TEXT
 *          =====
 *
 * This program displays text "Hello From ESP32" on an I2C LCD.
 * The LCD is connected to SDA and SCL port pins of the ESP32
 * Devkitc.
 *
 * File:    LCD
 * Date:    July 2017
 * Author:  Dogan Ibrahim
 */
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

//
```

```
// Set the LCD address to 0x27 and the configuration to
// 16 chars and 2 rows display
//
LiquidCrystal_I2C lcd(0x27, 16, 2);           // LCD address 0x27

void setup()
{
    lcd.begin();                           // Initialize LCD
    lcd.backlight();                      // Turn ON backlight
    lcd.print("Hello from ESP32");        // Display Text
}

void loop()
{
    // No code here
}
```

Bild 5.73 Programmlisting

### 5.10.8 Programmbeschreibung

Am Anfang werden die Bibliotheken **Wire** (I<sup>2</sup>C-Bibliothek) und **LiquidCrystal\_I2C** in das Programm eingebunden. In der I<sup>2</sup>C-LCD-Bibliothek werden dann die Adresse des LCDs (0x27) und seine Konfiguration (16 Spalten à 2 Zeilen) definiert. Das Programm initialisiert dann die I<sup>2</sup>C-LCD-Bibliothek, schaltet die Hintergrundbeleuchtung ein und zeigt die Nachricht „Hello from ESP32“ an.

## 5.11 PROJEKT 10 – LCD-Ereigniszähler

### 5.11.1 Beschreibung

Das LCD-Ereigniszählerprojekt zählt externe Ereignisse, die an einem der GPIO-Ports auftreten, und zeigt das Ergebnis der Zählung auf einem LCD an. Ein Ereignis besteht darin, dass der Zustand des GPIO-Ports 23 von logisch 1 auf logisch 0 wechselt. Hier werden externe Ereignisse durch einen Taster simuliert.

### 5.11.2 Das Ziel

Ziel dieses Projekts ist es, ein alphanumerisches LCD in einem ESP32-DevKitC anzuschließen und darin Zahlen anzuzeigen. Es wird das gleiche I<sup>2</sup>C-kompatible 16x2-Zeichen-LCD verwendet wie im vorherigen Projekt.

### 5.11.3 Blockschaltbild

Das Blockdiagramm des Projekts in Bild 5.74 zeigt wie im letzten Projekt die Verbindung der SDA- und SCL-Pins des I<sup>2</sup>C-LCDs mit dem ESP32-DevKitC. Zusätzlich ist ein Drucktaster über einen 10-kΩ-Pull-Up-Widerstand an das DevKitC angeschlossen, um das Auftreten externer Ereignisse zu simulieren.

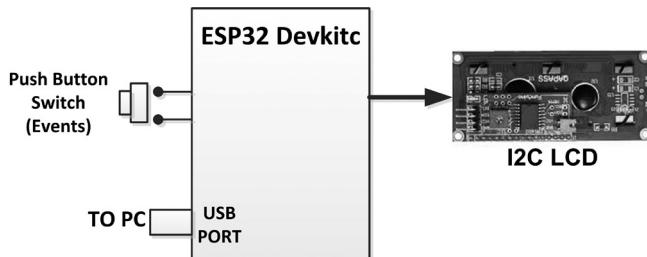


Bild 5.74 Blockschaltung des Projekts

#### 5.11.4 Schaltplan

Der Schaltplan des Projekts in Bild 5.75 zeigt, dass der SDA- und der CLK-Pin des I<sup>2</sup>C-LCDs an den GPIO-Ports 21 und 22 angeschlossen sind, der Drucktaster an den GPIO-Port 23.

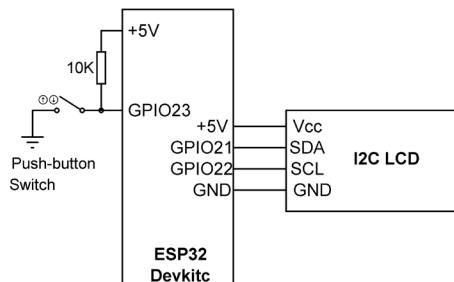


Bild 5.75 Schaltplan des Projekts

#### 5.11.5 Konstruktion

Die ESP32-DevKitC-Platine und der Taster sind auf einem Steckboard angebracht. Das I<sup>2</sup>C-LCD wird über Jumperkabel am DevKitC angeschlossen, wie in Bild 5.76 gezeigt.

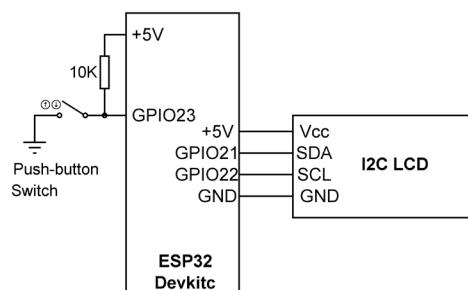


Bild 5.76 Das auf einem Steckboard aufgebaute Projekt

#### 5.11.6 PDL des Projekts

Die PDL des Projekts ist in Bild 5.77 dargestellt.

```
BEGIN
    Include libraries Wire and LiquidCrystal_I2C
    Assign GPIO port 23 to EVENTS
    Clear variable Count to 0
    Configure port 23 as input
    Set I2C LCD address and configuration
    Initialize I2C LCD
    Turn ON backlight
    Display text Event Counter
DO FOREVER
    Wait until an event occurs
    Contact debounce
    Clear LCD
    Increment Count
    Convert Count into string
    Display Count on LCD
ENDDO
END
```

*Bild 5.77 PDL des Projekts*

### 5.11.7 Programmlisting

Bild 5.78 zeigt das Listing des Programms LCDCounter.

```
/****************************************************************************
 *          LCD EVENT COUNTER
 *=====
 *
 * This is an event counter program with an LCD. Events are
 * assumed to occur on GPIO port 23 where an event is the HIGH
 * to LOW transition of this pin. In this project events are
 * simulated using a push-button switch connected to GPIO23 and
 * pulled high. An I2C LCD is connected to the ESP32-DevKitC.
 *
 * File:    LCDCounter
 * Date:    July 2017
 * Author:  Dogan Ibrahim
 */
#define EVENTS 23
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
int Count = 0;

//
// Set the LCD address to 0x27 and the configuration to
// 16 chars and 2 rows display. Also configure port 23 as
// an input
```

```

//  

LiquidCrystal_I2C lcd(0x27, 16, 2);           // LCD address 0x27  
  

void setup()  
{  

    pinMode(EVENTS, INPUT);  

    lcd.begin();                         // Initialize LCD  

    lcd.backlight();                     // Turn ON backlight  

    lcd.print("Event Counter");         // Display Text  

}  
  

void loop()  
{  

    while(digitalRead(EVENTS) == 1);      // Wait until event  

    delay(150);                          // Contact debounce  

    lcd.clear();                         // Clear LCD  

    Count++;                            // Increment Count  

    lcd.print(Count);                   // Display on LCD  

}

```

*Bild 5.78 Programmlisting*

### 5.11.8 Programmbeschreibung

Zu Beginn des Programms wird EVENT dem GPIO-Port 23 zugewiesen, die I<sup>2</sup>C- und die LCD-Bibliothek werden eingebunden. Die Variable **Count** wird auf Null gesetzt. In der Setup-Routine wird Port 23 als Eingabe konfiguriert, das LCD initialisiert und die Textnachricht „Event Counter“ auf dem LCD angezeigt.

Der Rest des Programms läuft in einer Endlosschleife. Das Programm wartet, bis ein Ereignis eintritt. Eine kleine Verzögerung entprellt den Taster (auch als Kontaktentprellung bezeichnet). Das Programm löscht dann das LCD-, erhöht **Count** um 1, wandelt die numerische Variable mit der Funktion **snprintf** in einen String um und zeigt den Wert von **Count** auf dem LCD.

Wir könnten auch den Befehl **lcd.print (Count)** verwenden, um den Wert der Variable **Count** anzuzeigen, statt ihn in einen String zu verwandeln.

## 5.12 PROJEKT 11 – LCD-BEFEHLE

### 5.12.1 Beschreibung

In diesem Projekt lernen wir die verfügbaren LCD-Befehle kennen und zeigen im LCD Text in verschiedenen Formaten an.

### 5.12.2 Das Ziel

Das Ziel dieses Projekts ist eine Liste der verfügbaren LCD-Befehle und deren Verwendung in Programmen.

### 5.12.3 Blockschaltbild

Die Blockschaltung des Projekts ist in Bild 5.66 zu sehen.

### 5.12.4 Schaltplan

Bild 5.67 zeigt den Schaltplan des Projekts.

### 5.12.5 Konstruktion

Die ESP32-DevKitC Karte ist wie in Bild 5.68 auf einem Breadboard montiert.

### 5.12.6 LCD-Befehle

Die I<sup>2</sup>C-LCD-Bibliothek unterstützt die folgenden Befehle:

clear()	löscht das LCD und positioniert den Cursor an der ersten Position
home()	positioniert den Cursor an der ersten Position
setCursor(x, y)	setzt den Cursor auf Spalte x, Zeile y (Index beginnt bei 0)
print()	zeigt im LCD an
display()	zeigt Zeichen im Display an (Standard)
noDisplay()	zeigt keine Zeichen im Display an
blink()	Cursor blinkt
noBlink()	Cursor blinkt nicht
Cursor()	zeigt den Cursor-Indikator an
noCursor()	zeigt den Cursor-Indikator nicht an
scrollDisplayLeft()	verschiebt die Anzeige um eine Position nach links
scrollDisplayRight()	verschiebt die Anzeige um eine Position nach rechts
leftToRight()	Text fließt von der Cursorposition nach rechts, wenn das Display linksbündig ist (Standard)
rightToLeft()	Text fließt von der Cursorposition nach links, wenn das Display rechtsbündig ist
noBacklight()	schaltet Hintergrundbeleuchtung aus
backlight()	schaltet Hintergrundbeleuchtung ein
getBacklight()	Helligkeitswert der Hintergrundbeleuchtung lesen
autoscroll()	verschiebt den ganzen Text jedes Mal um ein Leerzeichen nach links, wenn ein Zeichen hinzugefügt wird (Standard)
noAutoscroll()	deaktiviertes automatisches Scrollen

In diesem Projekt sollen die folgenden Aktionen der LCD-Anzeige ausgeführt werden:

- Hintergrundbeleuchtung einschalten
- LCD löschen
- Zu Spalte 7, Zeile 0 gehen
- Anzeige des Textes „At 7,0“
- Zu Spalte 7, Zeile 1 gehen
- Anzeige des Textes „At 7,1“
- 1 Sekunde warten
- LCD löschen

Cursor deaktivieren  
 Displaytext „No Cursor“  
 1 Sekunde warten  
 Cursor aktivieren  
 Scrollen der Anzeige von rechts nach links aktivieren  
 1 Sekunde warten  
 LCD löschen  
 Cursor auf Spalte 10, Zeile 0 setzen  
 Displaytext „HELLO“  
 Scrollen der Anzeige von links nach rechts aktivieren  
 Count auf 100 setzen  
 LCD löschen  
 Count anzeigen

### 5.12.7 Programmlisting

Für diese Aktionen sorgt das Programm LCDCommands, dessen Listing in Bild 5.79 dargestellt ist.

```

*****
*          LCD COMMANDS
*          =====
*
* This program uses various LCD commands with the aim of making
* the programmer aware of the available commands.
*
* File:    LCDCommands
* Date:    July 2017
* Author:  Dogan Ibrahim
*****
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
int Count = 100;

//
// Set the LCD address to 0x27 and the configuration to
// 16 chars and 2 rows display
//
LiquidCrystal_I2C lcd(0x27, 16, 2);      // LCD address 0x27

void setup()
{
  lcd.begin();                           // Initialize LCD
  lcd.backlight();                      // Turn ON backlight
  lcd.clear();                          // Clear LCD
  lcd.setCursor(7, 0);                  // column 7, row 0
  lcd.print("At 7,0");                  // Display "At 7,0"
}

```

```

lcd.setCursor(7, 1);           // Column 7, row 1
lcd.print("At 7,1");          // Display "At 7,1"
delay(1000);                  // Wait 1 second
lcd.clear();                  // Clear LCD
lcd.noCursor();               // Disable cursor
lcd.print("No cursor");       // Display "No cursor"
delay(1000);                  // Wait 1 second
lcd.cursor();                 // Enable cursor
lcd.scrollDisplayRight();     // Scroll right
delay(1000);                  // Wait 1 second
lcd.clear();
lcd.setCursor(10,0);
lcd.rightToLeft();            // Right to left
lcd.print("Hello");           // Display "Hello"
lcd.leftToRight();            // Left to right
delay(1000);                  // Wait 1 second
Count = 100;                  // Count = 100
lcd.clear();                  // Clear LCD
lcd.print(Count);             // Display Count
}

void loop()
{
    // No code
}

```

*Bild 5.79 Programmlisting*

### 5.12.8 Programmbeschreibung

Die Bilder 5.80 ... 5.84 zeigen die verschiedenen Anzeigen, wenn das Programm in Bild 5.79 ausgeführt wird.

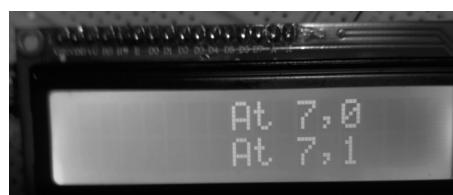
*Bild 5.80 LCD-Anzeige 1**Bild 5.81 LCD-Anzeige 2*



Bild 5.82 LCD-Anzeige 3



Bild 5.83 LCD-Anzeige 4



Bild 5.84 LCD-Anzeige 5

## 5.13 PROJEKT 12 – EXTERNE INTERRUPTS

### 5.13.1 Beschreibung

In diesem Projekt ist eine LED an den GPIO-Pin 23 angeschlossen, außerdem ein Tastschalter an GPIO-Pin 22. Die LED blinkt normalerweise alle 500 ms. Durch Drücken der Taste wird ein externer Interrupt erzeugt. Wird der Interrupt erzeugt, während die LED blinkt, hört sie auf zu blinken, ist die LED dagegen inaktiv, beginnt sie zu blinken, wenn die Taste gedrückt wird.

### 5.13.2 Das Ziel

Das Ziel dieses Projekts ist es, in der Arduino-IDE programmierte externe Interrupts am ESP32-DevKitC zu konfigurieren.

### 5.13.3 Blockschaltbild

Die Blockschaltung des Projekts ist in Bild 5.85 dargestellt.

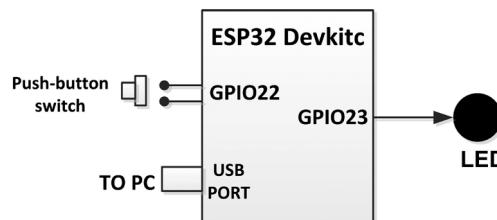


Bild 5.85 Blockschaltung des Projekts

#### 5.13.4 Schaltplan

Der Schaltplan des Projekts in Bild 5.86 zeigt eine LED, die über einen 330- $\Omega$ -Widerstand mit GPIO-Port 23 verbunden ist, und einen Taster an GPIO-Port 22. Port 22 wird dabei von einem internen Pull-up-Widerstand auf HIGH gezogen.

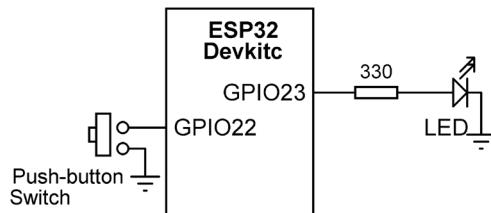


Bild 5.86 Schaltplan des Projekts

#### 5.13.5 Konstruktion

ESP32-DevKitC, LED und Taster sind wie in Bild 5.87 auf einem Steckboard montiert.

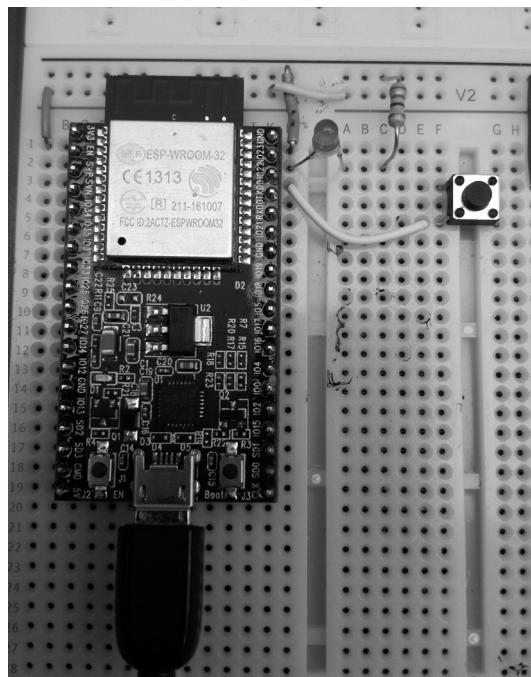


Bild 5.87 Das auf einem Steckbrett aufgebaute Projekt

#### 5.13.6 PDL des Projekts

Die PDL des Projekts ist in Bild 5.88 dargestellt.

##### BEGIN

    Assign LED to GPIO port 23

    Assign IntPin to GPIO port 22

```

        Configure LED port as output
        Configure IntPin as input
        Flag = 1
DO FOREVER
    IF Flag = 1 THEN
        Flash the LED
    ELSE
        Stop flashing
    ENDIF
ENDDO
END

Interrupt Service Routine: LEDControl
BEGIN
    Toggle Flag
END

```

*Bild 5.88 PDL des Projekts*

### 5.13.7 Programmlisting

Das Listing des Programms ExtInt ist in Bild 5.89 zu sehen.

```

/****************************************************************************
 *          EXTERNAL INTERRUPT
 *=====
 *
 * This is an example of using an external interrupt on the
 * ESP32-DevKitC. In this example an LED is connected to GPIO
 * port 23 through a 330 ohm resistor. In addition, a push-button
 * switch is connected to GPIO pin 22 and this button is then
 * configured as an interrupt pin such that pressing the button
 * generates an external interrupt in the program. Normally the
 * LED flashes every 500 ms. If the button is pressed when the
 * LED is flashing then the flashing stops. If on the other hand
 * the flashing is stopped, pressing the button will re-start
 * the LED to flash.
 *
 *
 * File:   ExtInt
 * Date:   July 2017
 * Author: Dogan Ibrahim
 *****/
#define LED 23                                // LED on GPIO 23
#define IntPin 22                               // Interrupt pin
int Flag = 1;                                // LED flag

//
```

```
// Configure LED port as output, interrupt pin as input, and
// PULL-UP this pin so that its state is at normally logic HIGH.
// Also, Attach the interrupt pin IntPin to function LEDControl,
// and accept interrupts on the FALLING edge (logic HIGH to LOW)
// of the interrupt pin
//
void setup()
{
    pinMode(LED, OUTPUT);                      // LED is output
    pinMode(IntPin, INPUT_PULLUP);              // Interrupt pin
    attachInterrupt(digitalPinToInterrupt(IntPin),LEDControl,FALLING);
}

//
// Flash the LED every 500 ms. The flashing is stopped and
// re-started by the interrupt pin. When interrupt occurs
// by pressing the IntPin, the state of variable Flag is
// changed (if 0, it becomes 1; and if 1 it becomes 0)
//
void LEDControl()
{
    if(Flag == 0)
        Flag = 1;
    else Flag = 0;
}

//
// Flash the LED every second unless topped by the interrupt
//
void loop()
{
    if(Flag == 1)
    {
        digitalWrite(LED, LOW);                // LED OFF
        delay(500);                          // 500 ms delay
        digitalWrite(LED, HIGH);              // LED ON
        delay(500);                          // 500 ms delay
    }
    else
    {
        digitalWrite(LED, LOW);                // LED OFF
    }
}
```

Bild 5.89 Programmlisting

### 5.13.8 Programmbeschreibung

Am Anfang des Programms wird die LED dem GPIO-Port 23 und der Interrupt-Anschluss **IntPin** dem GPIO-Pin 22 zugeordnet. Innerhalb der Setup-Routine wird die LED als Ausgang und der Interrupt-Pin als INPUT\_PULLUP konfiguriert, so dass der Pin normalerweise auf logisch HIGH liegt. Der externe Interrupt-Pin ist an die Funktion **LEDControl** gekoppelt, die in diesem Beispiel die Interrupt-Service-Routine ist. Ein Druck auf die Taste **IntPin** erzeugt einen externen Interrupt, so dass das Programm automatisch zur Funktion **LEDControl** springt. Innerhalb des Hauptprogramms blinkt die LED alle 500 ms, wenn die Variable **Flag** logisch 1 ist. Diese Variable wird innerhalb der Funktion **LEDControl** umgeschaltet. Wenn die Taste gedrückt wird, ändert die Variable **Flag** ihren Wert auf 0, so dass die LED aufhört zu blinken. Durch nochmaliges Drücken der Taste ändert sich **Flag** wieder auf 1 und das Blinken beginnt erneut.

Es werden vier Modi des externen Interrupts unterstützt:

- RISING:** Der Interrupt wird durch eine steigende Flanke (LOW zu HIGH) am Interrupt-Pins ausgelöst.
- FALLING:** Der Interrupt wird durch eine fallende Flanke (HIGH zu LOW) am Interrupt-Pins ausgelöst.
- CHANGE:** Der Interrupt wird ausgelöst, wenn sich der Zustand des Interrupt-Pins ändert (HIGH auf LOW oder LOW auf HIGH).
- LOW:** Der Interrupt wird ausgelöst, wenn der Interrupt-Pin LOW ist.

## 5.14 PROJEKT 13 – TIMER-INTERRUPTS

### 5.14.1 Beschreibung

In diesem Projekt wird wie im vorherigen Projekt eine LED an den GPIO Pin 23 angeschlossen. Das Projekt lässt die LED durch einen Timer-Interrupt alle 500 ms blinken.

### 5.14.2 Das Ziel

In diesem Projekt soll gezeigt werden, wie die Timer-Interrupts des ESP32-DevKitC mit der Arduino-IDE programmiert werden können.

### 5.14.3 Blockdiagramm

Die Blockschaltung des Projekts ist in Bild 4.1 zu sehen.

### 5.14.4 Schaltplan

Der Schaltplan des Projekts ist in Bild 4.3 zu sehen.

### 5.14.5 Konstruktion

Das ESP32-DevKitC und die LED sind wie in Bild 4.4 auf einem Steckbrett montiert.

### 5.14.6 PDL des Projekts

Bild 5.90 zeigt die PDL des Projekts. Der ESP32-Prozessor enthält zwei Gruppen von Hardware-Timern, die jeweils zwei universell einsetzbare Hardware-Timer enthalten. Bei allen Timern handelt es sich um generische 64-Bit-Timer, die auf 16-Bit-Vorteilern basieren und

über 64-Bit-Auto-Reload-Funktionen verfügen, die auf- und abwärts zählen können.

```
BEGIN
    Assign LED to GPIO port 23
    Configure LED port as output
    Configure timer 0 with pre-scaler 80 and counting up
    Attach the timer interrupt to function LEDControl
    Set timer alarm to interrupt every 500 ms
END
    Timer interrupt service routine (LEDControl):
BEGIN
    Toggle the LED
END
```

*Bild 5.90 PDL des Projekts*

### 5.14.7 Programmlisting

Bild 5.91 zeigt das Listing des Programms TimerInt.

```
/*
 *          TIMER INTERRUPT
 * =====
 *
 * This is an example of using a timer interrupt on the
 * ESP32 Devkitc. In this example an LED is connected to GPIO
 * port 23. The LED is flashed every 500 ms inside the interrupt
 * service routine.
 *
 *
 * File:   TimerInt
 * Date:   July 2017
 * Author: Dogan Ibrahim
 */
#include <WiFi.h>
const char* ssid = "BTHomeSpot-XNH";
const char* password = "49345abaeb";

// Configure LED port as output
void setup()
{
    Serial.begin(9600);
    WiFi.begin(ssid, password);
    while(WiFi.status() != WL_CONNECTED)
    {
```

```
    delay(500);
    Serial.println("Connecting");
}
Serial.println("Connected");

}

void loop()
{
    // no code
}
```

Bild 5.91 Programmlisting

### 5.14.8 Programmbeschreibung

Am Anfang des Programms wird die LED dem GPIO-Port 23 zugeordnet. Innerhalb der Setup-Routine wird die LED als Ausgang konfiguriert. Timer 0 wird mit einem Vorteilerwert von 80 und im Aufwärtszählmodus eingestellt. Da die Taktfrequenz 80 MHz beträgt, ergibt sich bei einem Pre-Scaler-Wert von 80 ein Intervall von einer Mikrosekunde. Die Funktion **LEDControl** ist an den Timer-Interrupt gekoppelt, so dass immer dann, wenn der Timer überläuft und einen Interrupt erzeugt, das Programm automatisch zur Funktion **LEDControl** springt. Der Timer-Alarm ist auf 500 ms (500000 Mikrosekunden) und auf automatische Wiederholung eingestellt, so dass der Timer-Interrupt bei jedem Überlauf wiederholt wird. Die Funktion **LEDControl** schaltet die LED immer um.

### 5.15 Zusammenfassung

In diesem Kapitel haben wir eine Reihe von einfachen Projekten mit dem ESP32-DevKitC entwickelt. Im nächsten Kapitel werden wir uns mit der Entwicklung von Projekten mit mittlerem Schwierigkeitsgrad befassen, die komplexer sind als die bisher gezeigten.

## KAPITEL 6 • KOMPLEXERE PROJEKTE MIT DER ARDUINO-IDE UND DEM ESP32-DEVKITC

### 6.1 Übersicht

Im letzten Kapitel haben wir einige einfache Projekte mit dem ESP32-DevKitC entwickelt. In diesem Kapitel gehen wir zu komplexeren Projekten mit der Arduino-IDE als Entwicklungs-Software über.

Wie bei den vorherigen Projekten werden der Titel, die Beschreibung, das Ziel, das Blockdiagramm und so weiter aller Projekte angegeben.

### 6.2 PROJEKT 1 – Thermostat-Regelung

#### 6.2.1 Beschreibung

Dies ist eine Temperatursteuerung mit Thermostatfunktion. Ein Temperatursensor misst die Temperatur beispielsweise eines Raums, in dem die Temperatur geregelt werden soll, und vergleicht sie mit einer eingestellten Temperatur. Ein Heizgerät ist über ein Relais mit dem ESP32-DevKitC verbunden. Wenn die gemessene Temperatur unter der eingestellten Temperatur liegt, wird die Heizung eingeschaltet. Steigt die gemessene Temperatur über die eingestellte, so wird die Heizung ausgeschaltet. Ein I<sup>2</sup>C-LCD zeigt sowohl die eingestellte als auch die gemessene Temperatur. Zusätzlich ist ein Drucktaster (SetPoint) am DevKitC angeschlossen, an dem die gewünschte Temperatur – angezeigt im LCD – eingestellt wird. Ein weiterer Taster (START) startet den Regelalgorithmus. Die Solltemperatur kann im Bereich +15 ... 30 °C eingestellt werden. Bei jedem Tastendruck wird die Solltemperatur um 1 Grad Celsius erhöht und dieser Wert auf dem LCD angezeigt.

#### 6.2.2 Das Ziel

Dieses Projekts soll zeigen, wie mit dem ESP32-DevKitC eine Thermostat-Regelung entworfen werden kann.

#### 6.2.3 Blockschaltbild:

Bild 6.1 zeigt das Blockdiagramm des Projekts. Die Temperatur wird wie im Projekt 5.3 mit dem Sensorchip DHT11 gemessen, der sowohl die Temperatur als auch die Luftfeuchtigkeit ermitteln kann. Der Temperatur-Regelbereich liegt zwischen 15 °C und 30 °C. Mit dem Taster SetPoint wird die gewünschte Temperatur in 1-Grad-Schritten eingestellt. Nach 30 °C springt der Wert der Solltemperatur zurück auf 15 °C. Die Einstellung ist im LCD zu sehen, während die SetPoint-Taste gedrückt wird. Ein weiterer Taster namens START startet den Steueralgorithmus. Die Heizung wird durch das Relais ein- und ausgeschaltet. Das LCD zeigt sowohl die gemessenen als auch die Soll-Temperaturwerte einmal pro Sekunde an. Zusätzlich wird der Status des Relais als ON oder OFF eingeblendet.

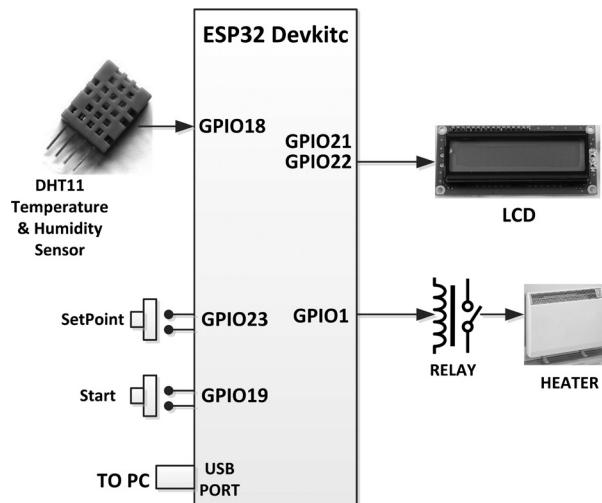


Bild 6.1 Blockschaltung des Projekts

### 6.2.4 Schaltplan

Der Schaltplan des Systems ist in Bild 6.2 zu sehen. Der Taster SetPoint ist an GPIO-Port 23, das Relais an GPIO-Port 1, der DHT11-Temperatursensor an GPIO-Port 18 und der START-Taster an GPIO-Port 19 angeschlossen. Die SDA- und SCL-Pins des I<sup>2</sup>C-LCDs sind mit den GPIO-Ports 21 respektive 22 verbunden. In diesem Projekt wird das Elektor-Relais aus dem **Sensor Kit** verwendet (siehe Bild 6.3). Die Relaisplatine besitzt drei Anschlüsse: + (Stromversorgung), S (Steuereingang) und – (Stromversorgung). Zur Messung der Temperatur wird der DHT11-Sensor aus dem gleichen Sensor-Kit verwendet (siehe Bild 6.4). Auch das DHT11-Modul weist drei Anschlüsse auf: + (Stromversorgung), S (Datenausgang) und – (Stromversorgung).

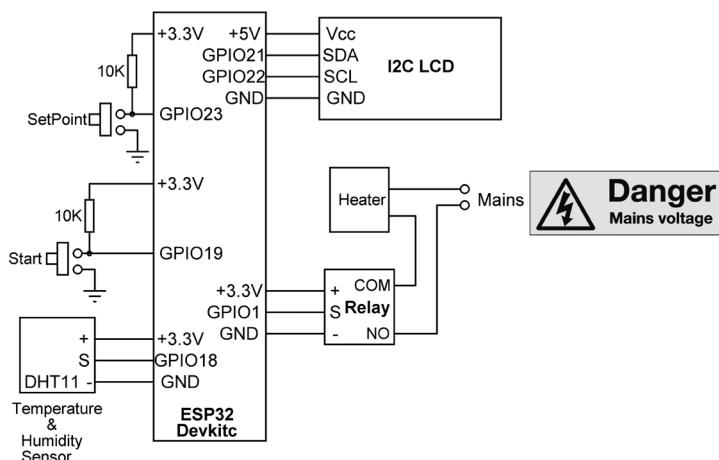


Bild 6.2 Schaltplan des Projekts



Bild 6.3 Das Relaismodul

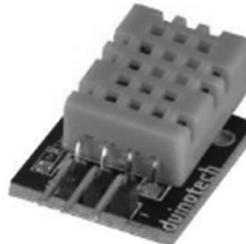


Bild 6.4 Das DHT11-Modul

### 6.2.5 Konstruktion

Das ESP32-DevKitC wird mit den Drucktastern und dem Temperatursensor-Chip DHT11 wie in Bild 6.5 zu sehen auf einem Steckboard montiert. Das I<sup>2</sup>C-LCD ist über Jumper-Drähte mit dem DevKitC verbunden.

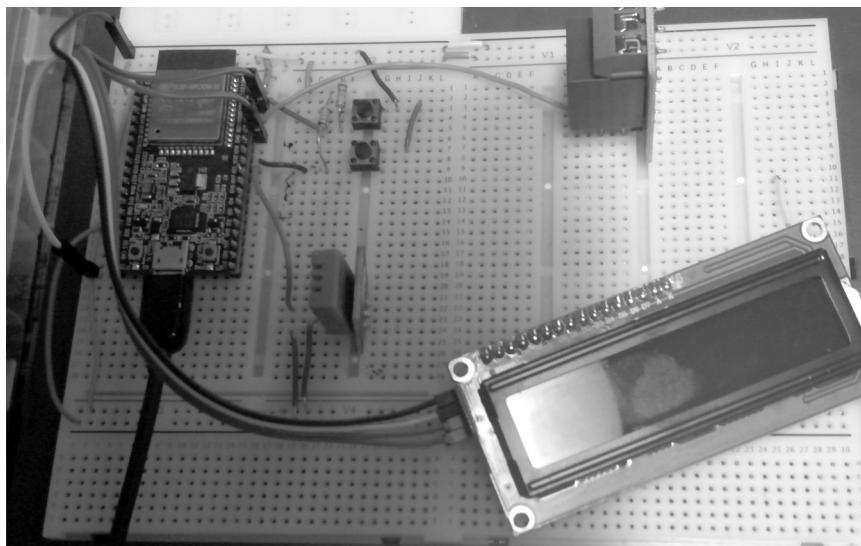


Bild 6.5 Projekt auf einem Steckbrett

### 6.2.6 PDL des Projekts

Die PDL des Projekts in Bild 6.6 zeigt, dass wie zuvor beschrieben die Heizung eingeschal-

tet wird, wenn die gemessene unter der eingestellten Temperatur liegt und ausgeschaltet, wenn die gemessene über der eingestellten Temperatur liegt. Die gewünschte Temperatur wird mit Taster SetPoint eingestellt, während der Taster START den Regelalgorithmus startet.

```

BEGIN
    Assign SetPoint to GPIO port 23
    Assign RELAY to GPIO port 1
    Assign START to GPIO port 19
    Include I2C and DHT11 libraries
    Configure SetPoint and START as input ports
    Configure RELAY as output port
    Turn OFF the relay to start with
DO FOREVER
    Read the required temperature setting
    Read the temperature from DHT11
    Display both the measured and the required temperatures
    IF required temperature > measured temperature THEN
        Turn ON relay
        Display text ON
    ELSE
        Turn OFF relay
        Display OFF
    ENDIF
    Wait 1 second
ENDDO
END

```

*Bild 6.6 PDL des Projekts*

### 6.2.7 Programmlisting

Das Listing des Programms ONOFF ist in Bild 6.7 dargestellt.

```

/*****
*          ON-OFF TEMPERATURE CONTROL
=====
*
* This is an ON-OFF temperature control project. A temperature
* sensor measures the temperature of the place (e.g. a room)
* where the temperature is to be controlled and compares it
* with a set temperature. A heater is connected to the ESP32
* Devkitc through a relay. If the measured temperature is below
* the set temperature then the heater is turned ON. If on the
* other hand the measured temperature is above the set point
* temperature then the heater is turned OFF. An I2C LCD is used
* to show both the set temperature and the measured temperature.
* In addition, a push-button switch is connected to the Devkitc

```

```
* and the required temperature is set using this push-button
* switch and the LCD. The temperature can be set between 15°C
* and 30°C. Every time the push-button switch is pressed the set
* point temperature is increased by one Degree Centigrade and
* this is shown on the LCD. The temperature control process
* starts when another push-button switch is pressed.
*
* File:    ONOFF
* Date:    July 2017
* Author: Dogan Ibrahim
*****// ****
#define SetPoint 23                                // SetPoint button
#define RELAY 1                                     // RELAY
#define START 19                                    // START button
//
// I2C LCD libraries
//
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
//
// DHT11 libraries
//
#include <Adafruit_Sensor.h>
#include "DHT.h"
#define DHT11_PIN 18
#define DHTTYPE DHT11
//
// Temperature variables
//
int MinTemperature = 15;                          // Min setting
int MaxTemperature = 30;                          // Max setting
int RequiredTemperature;
int First = 1;

DHT dht(DHT11_PIN, DHTTYPE);

//
// Set the LCD address to 0x27 and the configuration to
// 16 chars and 2 rows display. Also, initialize the LCD
// and the DHT11 sensor chip, turn the RELAY OFF to start
// with and clear the LCD
//
LiquidCrystal_I2C lcd(0x27, 16, 2);           // LCD address 0x27

void setup()
{
```

```
pinMode(RELAY, OUTPUT);           // Configure output
pinMode(SetPoint, INPUT);         // Configure input
pinMode(START, INPUT);           // Configure input
digitalWrite(RELAY, LOW);        // RELAY OFF to start with
dht.begin();                     // Initialize DHT11
lcd.begin();                     // Initialize LCD
lcd.backlight();                // Turn ON backlight
}

//  

// This function sets the required temperature. Pressing the  

// SetPoint button increases the required temperature value by  

// 1. When the required setting is reached, you should press  

// the START button to exit from this function. The required  

// temperature can be between 15 and 30 degrees Centigrade  

//  

int SetTemperature()
{
    int ExitFlag = 0;
    lcd.setCursor(0, 0);           // Go to col 0, row 0
    lcd.print("Req. Temperature"); // Display "Req. Temperature"
    lcd.setCursor(0, 1);           // Go to col 0, row 1
    RequiredTemperature = MinTemperature;
    lcd.print(RequiredTemperature); // Display req temp

    while(ExitFlag == 0)          // Do until START pressed
    {
        if(digitalRead(SetPoint) == 0)
        {
            RequiredTemperature++;
            if(RequiredTemperature > MaxTemperature)
            {
                RequiredTemperature = MinTemperature;
            }
            lcd.setCursor(0, 1);
            lcd.print(RequiredTemperature);
            delay(150);
        }
        else if(digitalRead(START) == 0)
        {
            ExitFlag = 1;
        }
    }
    lcd.clear();
    return RequiredTemperature;
}
```

```
//  
// Main program. At the beginning the required temperature is read  
// from the user. The program then implements the ON-OFF temp  
// algorithm to control the temperature. Both the measured and  
// the required temperature values are displayed on the LCD. If  
// the required temperature is greater than the measured one  
// then the relay is turned ON, otherwise the relay is turned  
// OFF. Texts ON or OFF are also displayed at the second row of  
// the LCD to show whether or not the relay is ON or OFF  
//  
void loop()  
{  
    if(First == 1)                                // If First time  
    {  
        RequiredTemperature = SetTemperature();  
        First = 0;  
    }  
    int Measured = dht.readTemperature();           // Read the temperature  
    lcd.setCursor(0,0);                            // Go to col 0, row 0  
    lcd.print("Measured=");                         // Display "Measured"  
    lcd.print(Measured);  
    lcd.print(char(223));                          // Display degree sign  
    lcd.setCursor(0,1);                            // Go to col 0, row 1  
    lcd.print("Required=");                         // Display "Required"  
    lcd.print(RequiredTemperature);  
    lcd.print(char(223));                          // Display degree sign  
    if(RequiredTemperature > Measured)  
    {  
        digitalWrite(RELAY, HIGH);                  // Turn RELAY ON  
        lcd.print(" ON ");                         // Display "ON"  
    }  
    else  
    {  
        digitalWrite(RELAY, LOW);                  // Turn RELAY OFF  
        lcd.print(" OFF");                        // Display "OFF"  
    }  
    delay(1000);                                 // Wait 1 second  
}
```

Bild 6.7 Programmlisting

## 6.2.8 Programmbeschreibung

Zu Beginn des Programms werden SetPoint, RELAY und START den GPIO-Ports 23, 1 und 19 zugewiesen. Dann werden die Bibliotheken für das I<sup>2</sup>C-LCD und den DHT11 in das Programm eingebunden. Für den Soll-Temperaturbereich werden **MinTemperatur** und **MaxTemperatur** auf 15 beziehungsweise 30 initialisiert.

In der Setup-Routine werden RELAY als Ausgang, die Taster SetPoint und START als Eingänge konfiguriert. Das Relais wird am Programmstart ausgeschaltet, die DHT11- und I<sup>2</sup>C-LCD-Bibliotheken werden gestartet und die LCD-Anzeige gelöscht.

Das restliche Programm läuft in einer Endlosschleife. Am Anfang der Schleife wird einmalig die Funktion **SetTemperature** aufgerufen, um die Soll-Temperatur einzustellen. Dabei wird die Variable **RequiredTemperature** auf den Wert von **MinTemperature** eingestellt und der Zustand der SetPoint-Taste überprüft. Bei jedem Tastendruck wird die Variable **RequiredTemperature** um 1 erhöht, bis die Grenzen von 30 erreicht ist. Die Funktion wird beendet und der Regelalgorithmus wird gestartet, wenn die START-Taste gedrückt wird (siehe Bild 6.8).



Bild 6.8 Einstellung der Soll-Temperatur

Im Hauptprogramm wird der Temperaturwert vom DHT11 gelesen und diese Ist-Temperatur mit der Soll-Temperatur angezeigt. Wenn die Soll-Temperatur größer als die Ist-Temperatur ist, wird das Relais eingeschaltet und damit auch die am Relais angeschlossene Heizung. Der Text ON wird in der zweiten Zeile des LCDs angezeigt (siehe Bild 6.9). Wenn andererseits die Soll- niedriger als die Ist-Temperatur ist, wird das Relais ausgeschaltet und der Text OFF in der zweiten Reihe des LCDs eingeblendet. Diese Schleife wird nach einer Sekunde Verzögerung wiederholt.

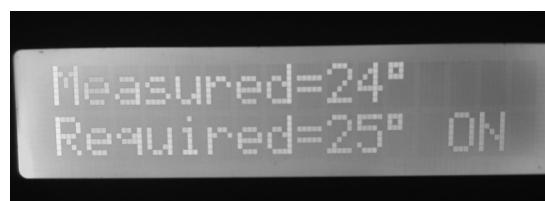


Bild 6.9 Das Relais ist eingeschaltet

**Hinweis:** Seien Sie vorsichtig und machen Sie sich klar, dass Sie es mit gefährlicher Netzspannung zu tun haben. Um keinen Stromschlag zu erleiden, sollten Sie professionellen Rat einholen, bevor Sie die Schaltung mit der Netzspannung verbinden!

## 6.3 PROJEKT 2 – Wellenformen erzeugen: Sägezahn

### 6.3.1 Beschreibung

In diesem Projekt wollen wir eine Sägezahn-Wellenform mit dem ESP32-DevKitC erzeugen.

### 6.3.2 Das Ziel

Das Ziel dieses Projekts ist es, den Digital-Analog-Wandler (DAC) des ESP32-DevKitC zu verwenden, um eine Sägezahnkurve zu erzeugen.

### 6.3.3 Blockschaltbild:

Bild 6.10 zeigt das Blockdiagramm des Projekts. Der ESP32-Prozessor erzeugt die erforderliche Sägezahn-Wellenform als digitales Signal, dann gibt der DAC dieses Signal am GPIO-Port aus, der für den DAC reserviert ist.

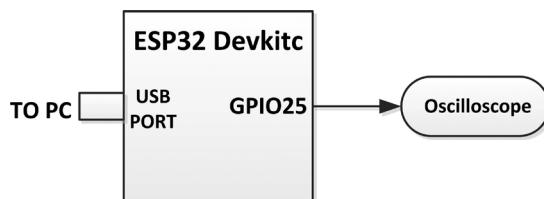


Bild 6.10 Blockdiagramm des Projekts

Grundsätzlich kann man zwei Methoden zur Erzeugung einer Wellenform verwenden:

- Der Prozessor berechnet die Punkte der Wellenform in Echtzeit und sendet sie an den DAC
- Die Punkte der Wellenform werden in einer sogenannten Lookup-Tabelle gespeichert. Der Prozessor liest die Punkte aus der Tabelle und sendet sie an den DAC. Mit dieser Methode kann man beliebige und auch höherfrequente Wellenformen erzeugen.

Die Geschwindigkeit, mit der die Wellenform-Punkte an den DAC gesendet werden, bestimmt die Frequenz der Wellenform.

### 6.3.4 Der DAC

Ein DAC konvertiert digitale in analoge Signale. Obwohl ein DAC ein externes Bauteil sein kann, verfügen die meisten Mikrocontroller heutzutage über eingebaute DACs. So gibt es beim ESP32-Prozessor zwei interne 8-Bit-DACs. Externe DACs können einen seriellen oder parallelen digitalen Eingang besitzen. Bei externen Wandlern mit Paralleleingang entspricht die Breite des digitalen Eingangs der Auflösung des Wandlers. 10-Bit-DACs beispielsweise haben zehn Eingangsbits. Externe serielle DACs verwenden normalerweise den SPI- oder den I<sup>2</sup>C-Bus, die eine Takt- und eine Datenleitung zum Senden der Daten verwenden, die vom DAC konvertiert werden sollen. Parallelwandler sind viel schneller, aber benötigen auch größere Bauteilgehäuse.

DACs werden in Bezug auf die Ausgangsspannungen entweder unipolar oder bipolar gefertigt. Unipolare Wandler können nur positive Spannungen ausgeben, Bipolar-Wandler sowohl positive als auch negative.

Die Beziehung zwischen dem digitalen Eingang/Ausgang und der Referenzspannung eines DACs ist gegeben durch:

$$V_o = \frac{DV_{ref}}{2^n}$$

$V_o$  ist die Ausgangsspannung,  $V_{ref}$  die Referenzspannung und  $n$  die Breite des Wandlers. D ist der digitale Wert. Bei einem 8-Bit-Wandler mit +3,3-V-Referenzspannung ist zum Beispiel

$$V_o = \frac{3.3D}{2^8} = 12.89D \text{ mV}$$

Wenn der digitale Wert beispielsweise 00000001 lautet, beträgt die analoge Ausgangsspannung 12,89 mV, wenn der digitale Wert 00000010 ist, liegt die analoge Ausgangsspannung bei 25,78 mV und so weiter.

In diesem Projekt wollen wir eine Sägezahn-Wellenform mit folgenden Spezifikationen erzeugen:

- Ausgangsspannung: 0 ... +3,3 V
- Frequenz: 100 Hz (Periode = 10 ms)
- Schrittweite: 0,1 ms

### 6.3.5 Schaltplan

Der Schaltplan des Systems ist in Bild 6.11 zu sehen. Von den beiden internen 8-Bit-DACs des ESP32-Prozessors an den GPIO-Ports 25 und 26 verwenden wir den an GPIO-Port 25.

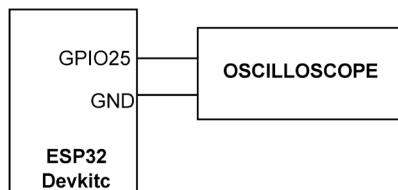


Bild 6.11 Schaltplan des Projekts

### 6.3.6 Konstruktion

Nur das ESP32-DevKitC befindet sich dem Steckboard. Die analoge Ausgangsspannung wurde/wird mit dem PC-Oszilloskop PSCGU250 erfasst und dargestellt.

### 6.3.7 PDL des Projekts

Da die Wellenform in 11 Schritten (0 ... 10) erzeugt wird und die erforderliche Frequenz 100 Hz (Periode von 10 ms) beträgt, sollte die Dauer jedes Schritts  $10.000/11 = 909 \mu\text{s}$  betragen. Die PDL des Projekts in Bild 6.12 zeigt diese erforderliche Verzögerung, die im Programm durch die Funktion **delayMicroseconds** erreicht wird.

```
BEGIN
    Assign Sawtooth to GPIO port 25
    Configure port 25 as analog
    DO FOREVER
        Send a step to the DAC
        Wait 909 microseconds
    ENDDO
END
```

*Bild 6.12 PDL des Projekts*

### 6.3.8 Programmlisting

Das Listing des Programms Sawtooth ist in Bild 6.13 dargestellt.

```
/*********************************************
*
*          SAWTOOTH WAVEFORM GENERATION
*
* =====
*
* In this project a Sawtooth waveform is generated through the
* DAC at GPIO port 25 of the ESP32-DevKitC.
*
* The specifications of the generated waveform are:
* Output voltage: 0 to +3.3V
* Frequency: 100 Hz (period 10 ms)
* Step size: 0.1 ms
*
* File:    Sawtooth
* Date:    July 2017
* Author: Dogan Ibrahim
*****
#define Sawtooth 25
int DAC_Value;
float Sample = 0.0, Inc = 0.1;

void setup()
{
    pinMode(Sawtooth, ANALOG);           // Configure as analog
}

void loop()
{
    DAC_Value = Sample * 255;
    dacWrite(Sawtooth, DAC_Value);       // Send to DAC
    Sample = Sample + Inc;
    if(Sample > 1.0 || Sample < 0.0)
    {
```

```

        Inc = -Inc;
        Sample = Sample + Inc;
    }
    delayMicroseconds(454); // 909 us delay
}

```

Bild 6.13 Programmlisting

### 6.3.9 Programmbeschreibung

Zu Beginn des Programms wird Sawtooth dem GPIO-Port 25 zugeordnet und dieser als analoger Port konfiguriert. Innerhalb des Hauptprogramms werden elf Schritte kontinuierlich mit einem Intervall von 909  $\mu$ s an den DAC gesendet. Mit der Funktion **dacWrite** werden die Schritte an den DAC gesendet. Diese Funktion hat die beiden Argumente DAC-Port-Nummer und Digitalwert, der in ein analoges Signal umgewandelt werden soll.

Bild 6.14 zeigt die erzeugte Sägezahnkurve auf dem PC-Oszilloskop PSCGU250.

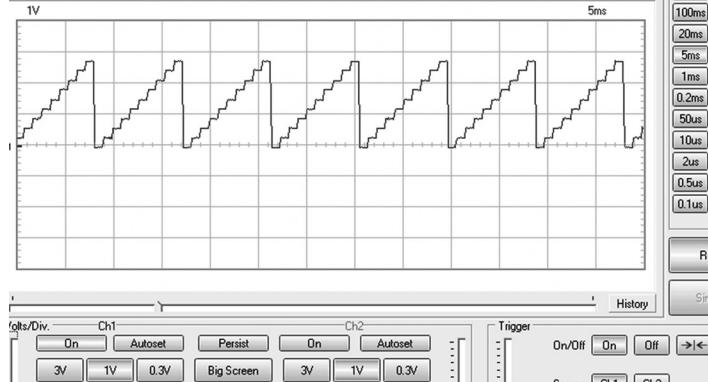


Bild 6.14 Generierte Sägezahn-Wellenform

## 6.4 PROJEKT 3 – Wellenformen erzeugen: Dreieck

### 6.4.1 Beschreibung

Dieses Projekt zeigt, wie eine Dreieck-Wellenform mit dem ESP32-DevKitC erzeugt werden kann.

### 6.4.2 Das Ziel

Dieses Projekt soll zeigen, wie der DAC des ESP32-DevKitC verwendet werden kann, um eine Dreieck-Wellenform zu erzeugen.

### 6.4.3 Blockschaltbild:

Das Blockschaltbild entspricht dem in Bild 6.10.

In diesem Projekt erzeugen wir eine Dreiecks-Wellenform mit folgenden Eigenschaften:

- Ausgangsspannung: 0 ... +3,3 V
- Frequenz: 100 Hz (Periode = 10 ms)
- Schrittweite: 0,1 ms

#### 6.4.4 Schaltplan

Der Schaltplan des Systems entspricht dem aus Bild 6.11.

#### 6.4.5 PDL des Projekts

Die PDL des Projekts ist in Bild 6.15 dargestellt. Da die erforderliche Periode 10 ms beträgt, sind die ansteigenden und abfallenden Flanken der Wellenform jeweils 454 µs lang. Auch hier wird die Funktion **delayMicroseconds** verwendet, um die erforderliche Verzögerung im Programm zu erzeugen.

```
BEGIN
    Assign Triangle to GPIO port 25
    Configure port 25 as analog
    DO FOREVER
        Send a step to the DAC
        Wait 454 microseconds
    ENDDO
END
```

*Bild 6.15 PDL des Projekts*

#### 6.4.6 Programmlisting

Das Listing des Projekts ist in Bild 6.16 (Programm Triangle) dargestellt.

```
*****
*                      TRIANGLE WAVEFORM GENERATION
*                      =====
*
* In this project a Triangle waveform is generated through the
* DAC at GPIO port 25 of the ESP32-DevKitC.
*
* The specifications of the generated waveform are:
* Output voltage: 0 to +3.3V
* Frequency: 100 Hz (period 10 ms)
* Step size: 0.1 ms
*
* File:   Triangle
* Date:   July 2017
* Author: Dogan Ibrahim
*****
#define Triangle 25
int DAC_Value;
float Sample = 0.0, Inc = 0.1;

void setup()
{
    pinMode(Triangle, ANALOG);           // Configure as analog
}
```

```

void loop()
{
    DAC_Value = Sample * 255;
    dacWrite(Triangle, DAC_Value);           // Send to DAC
    Sample = Sample + Inc;
    if(Sample > 1.0 || Sample < 0.0)
    {
        Inc = -Inc;
        Sample = Sample + Inc;
    }
    delayMicroseconds(454);                  // 909 us delay
}

```

Bild 6.16 Programmlisting

### 6.4.7 Programmbeschreibung

Am Anfang des Programms wird Triangle dem GPIO-Port 25 zugeordnet, der danach als analoger Port konfiguriert wird. Innerhalb des Hauptprogramms werden Samples in 454- $\mu$ s-Intervallen an den DAC gesendet. Dazu wird die Funktion **dacWrite** verwendet.

Bild 6.17 zeigt die erzeugte Dreieck-Wellenform, die mit dem PC-Oszilloskop PSCGU250 erfasst wurde.

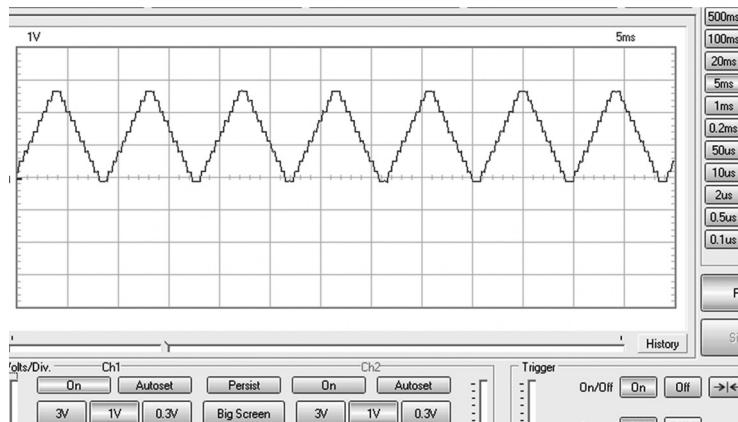


Bild 6.17 Generierte Dreieck-Wellenform

## 6.5 PROJEKT 4 – Port-Expander

### 6.5.1 Beschreibung

In diesem Projekt stellt ein Port-Expander des Typs MCP23017 dem ESP32-DevKitC zusätzliche 16 I/O-Ports zur Verfügung. An Port GPA0 (Pin 21) des MCP23017 ist eine LED angeschlossen, um Hard- und Software zu testen, die die LED im Sekundentakt blinken lässt. Ein 330- $\Omega$ -Strombegrenzungswiderstand wird in Reihe mit der LED geschaltet.

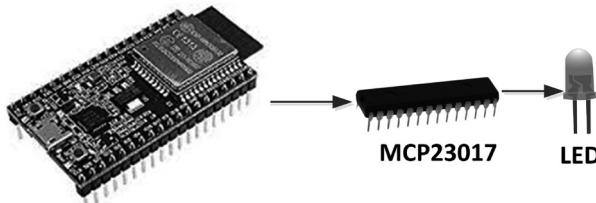
### 6.5.2 Das Ziel

Dieses Projekt soll zeigen, wie die I<sup>2</sup>C-Befehle mit dem ESP32-DevKitC und der Ardui-

no-IDE verwendet werden können und wie damit der Port-Expander MCP23017 programmiert werden kann.

### 6.5.3 Blockschaltbild

Bild 6.18 zeigt die Blockschaltung des Projekts, bei dem die LED an einen der Ports des MCP23017 angeschlossen ist, während 15 weitere I/O-Ports frei bleiben.

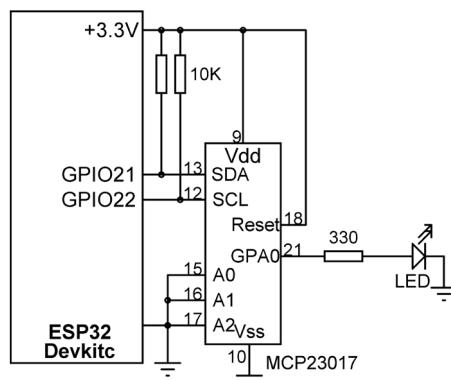


**ESP32 Devkitc**

*Bild 6.18 Blockschaltbild des Projekts*

### 6.5.4 Schaltplan

Der Schaltplan des Projekts ist in Bild 6.19 zu sehen. Der MCP23017 ist über den I<sup>2</sup>C-Bus mit ESP32-DevKitC verbunden, wobei die SDA- und SCL-Leitungen beider Bus-Teilnehmer miteinander gekoppelt sind. An diesen Leitungen sind 10-kΩ-Pull-up-Widerstände angegeschlossen. Die LED ist mit dem Port-Pin GPA0 des MCP23017 verbunden. Die Address-select-Bits des MCP23017 sind alle auf Masse gelegt, so dass die Slave-Adresse auf 0x20 eingestellt ist.



*Bild 6.19 Schaltplan des Projekts*

### 6.5.5 Der MCP23017

Der MCP23017 ist ein IC mit 28 Anschlässen. Bild 6.20 zeigt ein Bild des ICs, dessen Pin-Belegung in Bild 6.21 dargestellt ist. Das IC besitzt folgende Eigenschaften:

- 16 bidirektionale I/O-Ports
- Betrieb am I<sup>2</sup>C-Bus mit bis zu 1,7 MHz
- Interrupt-Fähigkeit

- Externer Reset-Eingang
- Niedriger Standby-Strom
- Betrieb bei +1,8 ... 5,5 V
- 3 Adresspins, so dass bis zu 8 MCP23017 an einem I<sup>2</sup>C-Bus betrieben werden können
- 28-poliges DIL-Gehäuse

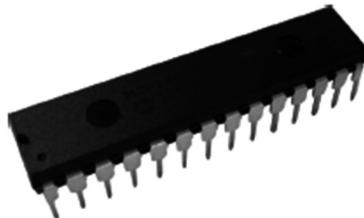


Bild 6.20 Der Port-Expander MCP23017

```

BEGIN
  Define MCP23017 registers
  Initialize the I2C bus
  Configure GPIOA as output
DO FOREVER
  Send 0 to GPIOA
  Wait 1 second
  Send 1 to GPIOA
  Wait 1 second
ENDDO
END

```

Bild 6.21 Anschlussbelegung des MCP23017

Die Anschlüsse werden in Tabelle 6.1 beschrieben.

Pin	Beschreibung
GPA0 ... GPA7	Anschlüsse Port A
GPB0 ... GPB7	Anschlüsse Port B
VDD	Stromversorgung
VSS	Masse
SDA	I2C-Daten
SCL	I2C-Takt
RESET	Reset
A0 ... A2	I2C-Adressierung

Tabelle 6.1 Anschlussbelegung des MCP23017

Der MCP23017 wird über die Pins A0 ... A2 adressiert. Tabelle 6.2 zeigt die Adressauswahl. In diesem Projekt sind alle Adresspins mit Masse verbunden, so dass die Adresse des ICs 0x20 lautet.

A2	A1	A0	Adresse
0	0	0	0x20
0	0	1	0x21
0	1	0	0x22
0	1	1	0x23
1	0	0	0x24
1	0	1	0x25
1	1	0	0x26
1	1	1	0x27

Tabelle 6.2 Adressauswahl des MCP23017

Der MCP23017 verfügt über acht interne Register, die für die Betriebsart konfiguriert werden können. Das IC kann je nach Konfiguration des Bits IOCON.BANK entweder im 16-Bit-Modus oder im 2x8-Bit-Modus betrieben werden. Beim Einschalten wird dieses Bit gelöscht, so dass standardmäßig der 2x8-Bit-Modus gewählt ist.

Die I/O-Richtung der Port-Pins wird mit den Registern IODIRA (Adresse 0x00) und IODIRB (Adresse 0x01) festgelegt. Wenn in diesen Registern ein Bit auf 0 gesetzt wird, wird der korrespondierende Anschluss-Pin zum Ausgang. Setzt man dagegen ein Bit in diesen Registern auf 1, so wird der entsprechende Port-Pin zum Eingang. Die Registeradressen für GPIOA und GPIOB lauten 0x12 beziehungsweise 0x13. Dies ist in Bild 6.22 dargestellt.

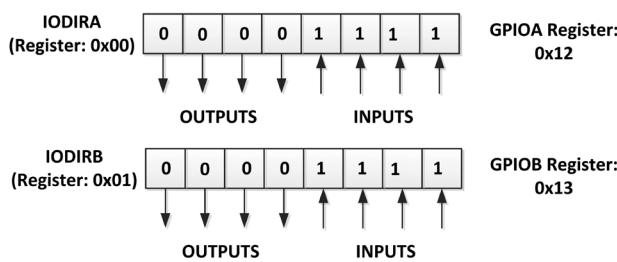


Bild 6.22 Konfigurieren der I/O-Leitungen

Weitere Informationen zum MCP23017 können dem Datenblatt <http://docs-europe.electrocomponents.com/webdocs/137e/0900766b8137eed4.pdf> entnommen werden

### 6.5.6 Konstruktion

Das ESP32-DevKitC wird mit dem Port-Expander MCP23017 und der LED wie in Bild 6.23 gezeigt auf einem Steckbrett angebracht.

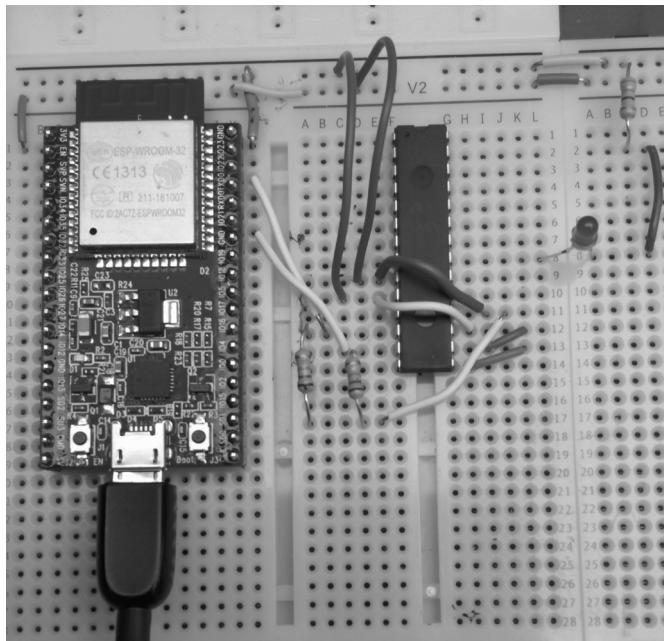


Bild 6.23 Das Projekt auf einem Steckboard

### 6.5.7 PDL des Projekts

Bild 6.24 zeigt die PDL des Projekts.

```

BEGIN
    Include I2C library
    Define slave address and register addresses
    Initialize I2C bus
    Configure GPIOA as output
DO FOREVER
    Turn OFF LED (Send 0)
    Wait 1 second
    Turn ON LED (Send 1)
    Wait 1 second
ENDDO
END

```

Bild 6.24 PDL des Projekts

### 6.5.8 Programmlisting

Das Listing des Programms Expander ist in Bild 6.25 dargestellt.

```

*****
* PORT EXPANDER
* =====
*

```

```
* In this project an MCP23017 Port Expander chip is used to
* add 16 more I/O ports to the ESP32-DevKitC. This chip is
* interfaced to a processor via the I2C bus (SDA and SCL lines).
* 10K pull-up resistors are used at these lines. In order to
* test the hardware and the software, an LED is connected to
* port GPA0 (pin 21) of the MCP23017 chip. The program flashes
* the LED every second.
*
*
* File: Expander
* Date: July 2017
* Author: Dogan Ibrahim
***** */
#include <Wire.h> // Include I2C library
const byte slave_addr=0x20; // MCP23017 I2C address
const byte IODIRA=0x00; // IODIRA address
const byte GPIOA=0x12; // GPIOA address
//
// This function configures the MCP23017 chip such that Port pin
// GPA0 is an output pin. PortDir is the address of the port
// direction register (IODIRA or IODIRB), and Data is the
// direction data such that 0 sets the corresponding port pin to
// output, and 1 sets the corresponding port pin to input
//
void Configure(char PortDir, char Data)
{
    Wire.beginTransmission(slave_addr);
    Wire.write(PortDir);
    Wire.write(Data);
    Wire.endTransmission();
}
//
// This function sends data to the specified port. The port can be
// GPIOA or GPIOB and data is a byte
//
void Send(char Port, char Data)
{
    Wire.beginTransmission(slave_addr);
    Wire.write(Port);
    Wire.write(Data);
    Wire.endTransmission();
}
//
// Initialize the I2C bus, configure port GPIOA as an output port
//
void setup()
```

```
{  
    Wire.begin();  
    Configure(IODIRA, 0xFE);  
}  
//  
// Main program initializes the I2C, configures the GPIOA port  
// and flashes the LED connected to port pin GPA0 every second  
//  
void loop()  
{  
    Send(GPIOA, 0); // LED OFF  
    delay(1000); // Wait 1 second  
    Send(GPIOA, 1); // LED ON  
    delay(1000); // Wait 1 second  
}
```

Bild 6.25 Programmlisting

### 6.5.9 Programmbeschreibung

Das Senden von Daten an einen I<sup>2</sup>C-Slave erfolgt in einem fünf Schritte umfassenden Verfahren:

- Übertragung starten
- Geräteadresse senden
- Registeradresse senden
- Daten an das Register senden
- Ende der Übertragung

Zu Beginn des Programms wird die I<sup>2</sup>C-Bibliothek in das Programm eingebunden, die Slave-Adresse ist auf 0x20 gesetzt und die Adressen in IODIRA und GPIOA auf 0x00 und 0x12 eingestellt. Innerhalb der Setup-Routine wird der I<sup>2</sup>C-Bus initialisiert und der GPIOA-Pin 0 des MCP23017 als Ausgangs-Port konfiguriert. Beachten Sie, dass die Hexadezimalzahl 0xFF (binär: 1111 1110) an das Register gesendet wird, so dass Bit 0 Ausgang und alle anderen Bits Eingänge werden. Innerhalb der Hauptprogrammfunktion wird **Send** aufgerufen, um die LED ein- und auszuschalten. Nach jedem Wechsel wird eine Pause von einer Sekunde eingefügt.

## 6.6 PROJEKT 5 – Elektronische Mini-Orgel

### 6.6.1 Beschreibung

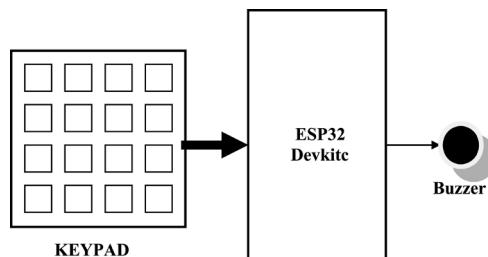
In diesem Projekt wird eine elektronische Mini-Orgel entwickelt, die mit dem ESP32-DevKitC und einer 4x4-Tastatur mit 16 Tasten Musiktöne innerhalb einer Oktave erzeugt.

### 6.6.2 Das Ziel

Ziel dieses Projekts ist es, eine Tastatur an einem ESP32-DevKitC anzuschließen und zu verwenden, um mit dieser Tastatur eine elektronische Mini-Orgel aufzubauen.

### 6.6.3 Blockschaltbild

Bild 6.26 zeigt das Blockschaltbild des Projekts, bei dem die Tastatur und ein passiver Summen mit dem ESP32-DevKitC verbunden sind.



*Bild 6.26 Blockschaltbild des Projekts*

### 6.6.4 Schaltplan

Bild 6.27 zeigt die Tastatur, die in diesem Entwurf verwendet wird. Sie besitzt 16 Tasten, die von S1 bis S16 wie folgt gekennzeichnet sind:

- Zeile 1: S1 bis S4
- Zeile 2: S5 bis S8
- Zeile 3: S9 bis S12
- Zeile 4: S13 bis S16



*Bild 6.27 Die Tastatur im Projekt*

Der Schaltplan der Tastatur in Bild 6.28 zeigt, dass die Zeilen von oben nach unten und die Spalten von links nach rechts jeweils von 1 bis 4 durchnummieriert sind.

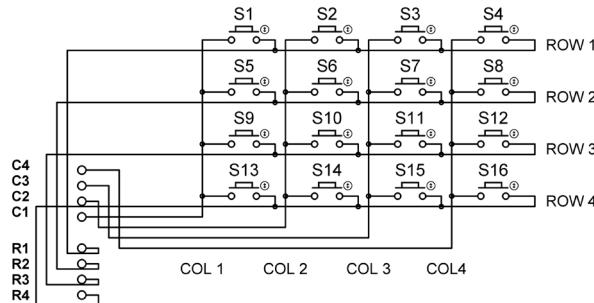


Bild 6.28 Schaltplan der Tastatur

Im Schaltplan des Projekts in Bild 6.29 ist zu sehen, dass die Zeilen und Spalten der Tastatur wie folgt verbunden sind. Pin 1 ist dabei der obere Pin, wenn sich die Anschlüsse Pins links vom Keypad befinden. Bild 4.2 zeigt die Anschlussbelegung des ESP32-DevKitC.

Zeile	Tastatur-Pin	ESP32-GPIO-Port
C4	1	23
C3	2	22
C2	3	21
C1	4	19
R1	5	18
R2	6	5
R3	7	17
R4	8	16

Der passive Summer wird an den GPIO-Port 4 angeschlossen.

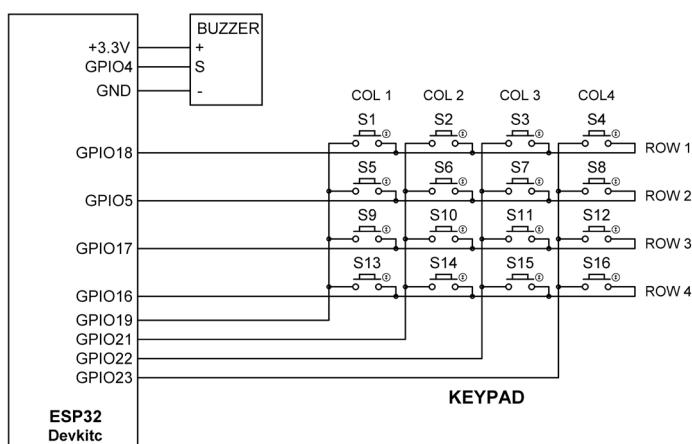


Bild 6.29 Schaltplan des Projekts

Die Musiktöne werden auf der Tastatur wie folgt konfiguriert:

c <sup>1</sup>	d <sup>1</sup>	e <sup>1</sup>	f <sup>1</sup>
g <sup>1</sup>	a <sup>1</sup>	h <sup>1</sup>	c <sup>2</sup>
cis <sup>1</sup>	dis <sup>1</sup>	fis <sup>1</sup>	gis <sup>1</sup>
b <sup>1</sup>	x	x	x

Die Frequenzen der Noten sind:

Note (engl. Notation)	C4	C4#	D4	D4#	E4	F4	F4#	G4	G4#	A4	A4#	B4	C5
Note (deutsche Notation)	c1	cis1	d1	dis1	e1	f1	fis1	g1	gis1	a1	ais1	h1	c2
Frequenz [Hz]	261,63	277,18	293,66	311,13	329,63	349,23	369	392	415,3	440	466,16	493,88	523,25

## 6.6.5 Konstruktion

Das ESP32-DevKitC wird zusammen mit der Tastatur und dem passiven Summer wie in Bild 6.30 zu sehen auf einem Steckboard aufgebaut.

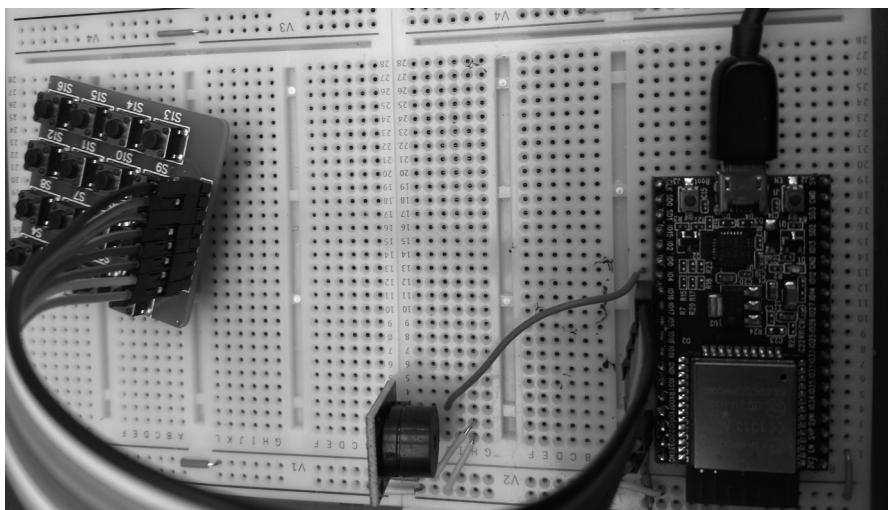


Bild 6.30 Projekt auf einem Steckbrett

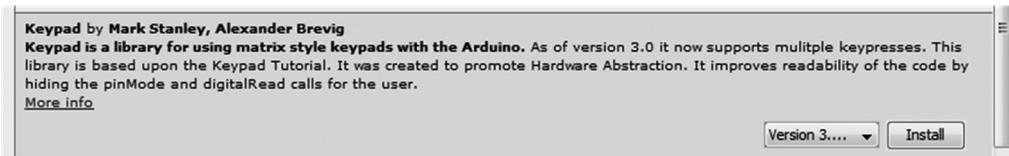
## 6.6.6 PDL des Projekts

Bevor die Tastaturfunktionen verwendet werden können, muss die Tastaturlibrary in die Arduino-IDE geladen werden. Befolgen Sie dazu folgende Schritte:

- Starten Sie die Arduino-IDE.
- Klicken Sie auf **Sketch -> Bibliothek einbinden -> Bibliotheken verwalten**.
- Suche Sie nach der **Keypad**-Bibliothek.

- Klicken Sie auf den Text und dann auf **Installieren**, um die Bibliothek zu installieren (siehe Bild 6.31).
- Schließen Sie den Bibliotheksverwalter.

Nach der Installation dieser Tastaturbibliothek können Sie die Headerdatei **keypad.h** in Ihre Programme aufnehmen.



*Bild 6.31 Installieren der Tastaturbibliothek*

Die PDL des Projekts ist in Bild 6.32 dargestellt.

```

BEGIN
    Include Keypad.h in the program
    Assign BUZZER to GPIO 4
    Define musical tones for an octave
    Define PWM parameters
    Define keypad parameters
    Define the interface between the keypad and the GPIO ports
DO FOREVER
    IF a key is pressed THEN
        Find the numeric value of the key
        Find the required tone frequency
        Send tone to the buzzer for 250 ms
    ENDIF
ENDDO
END

```

*Bild 6.32 PDL des Projekts*

### 6.6.7 Programmlisting

Bevor man das komplette Programm schreibt, sollte man die Hardware aufbauen und dann ein Programm zum Test der Tastaturschnittstelle verfassen. Dieses Programm zeigt die Tastennummer im seriellen Monitor der Arduino-IDE an, wenn Sie die Tasten nacheinander drücken. Dieser Programmcode (TestKeypad) kann dem Bild 6.33 entnommen werden. Führen Sie das Programm aus und drücken Sie die Tasten auf dem Keyboard. Auf dem Monitor erscheinen die Tastennummern (1 bis 9 oder a bis g), siehe Bild 6.34.

```

/***** MINI ELECTRONIC ORGAN *****
*                                     MINI ELECTRONIC ORGAN
*                                     =====
*
* In this project a 4x4 keypad is connected to the ESP32-DevKitC

```

```
* boards. In addition, a passive buzzer is connected to GPIO
* port 17. The program configures the keypad so that the
* musical notes for one octave can be played by using the
* keypad.
*
* The keypad is configured as follows (A to G are the musical
* notes):
* C4  D4  E4  F4
* G4  A4  B4  C5
* C4# D4# F4# G4#
* A4#
*
*
* File:  TestKeypad
* Date:  July 2017
* Author: Dogan Ibrahim
*****
#include "Keypad.h"                                // Include Keypad library
#define BUZZER 17
const byte Rows = 4;                                // Number of rows
const byte Cols = 4;                                // Number of columns
//
// Define the keymap on the keypad
//
char keys[Rows][Cols] =
{
    {'1', '2', '3', '4'},
    {'5', '6', '7', '8'},
    {'9', 'a', 'b', 'c'},
    {'d', 'e', 'f', 'g'}
};
//
// Define the Keypad connections for the rows and columns
//
byte rowPins[Rows] = {18, 5, 17, 16};           // Rows 1 to 4
byte colPins[Cols] = {19, 21, 22, 23};           // Cols 1 to 4
//
// Initialize the Keypad library with the row and columns defs.
// Here, we have to specify the key map name (keys), row pin
// assignments to GPIO ports (rowPins), and the column pin
// assignments to GPIO ports (colPins)
//
Keypad kpd=Keypad(makeKeymap(keys),rowPins,colPins,Rows,Cols);

//
// Configure BUZZER as an output port
```

```

// 
void setup()
{
    pinMode(BUZZER, OUTPUT);
    Serial.begin(9600); // Start the Monitor
}

// 
void loop()
{
    char keypressed = kpd.getKey(); // Look for key press
    if(keypressed != NO_KEY) // If a key is pressed
    {
        Serial.println(keypressed); // Display the key
    }
}

```

Bild 6.33 Tastatur-Testprogramm



Bild 6.34 Im seriellen Monitor angezeigte Tastendrücke

Die vollständige Listing des Programms Organ zu diesem Projekt ist in Bild 6.35 zu sehen.

```

*****
*          MINI ELECTRONIC ORGAN
*
* In this project a 4x4 keypad is connected to the ESP32-DevKitC
* boards. In addition, a passive buzzer is connected to GPIO
* port 17. The program configures the keypad so that the
* musical notes for one octave can be played by using the
* keypad.
*

```

```
* The keypad is configured as follows (A to G are the musical
* notes):
* C4  D4  E4  F4
* G4  A4  B4  C5
* C4# D4# F4# G4#
* A4#
*
*
* File:  Organ
* Date:  July 2017
* Author: Dogan Ibrahim
***** */
#include "Keypad.h"                                // Include Keypad library
#define BUZZER 17
const byte Rows = 4;                               // Number of rows
const byte Cols = 4;                               // Number of columns
//
// Define the keymap on the keypad
//
char keys[Rows][Cols] =
{
    {'1', '2', '3', '4'},
    {'5', '6', '7', '8'},
    {'9', 'a', 'b', 'c'},
    {'d', 'e', 'f', 'g'}
};
//
// Define the Keypad connections for the rows and columns
//
byte rowPins[Rows] = {18, 5, 17, 16};           // Rows 1 to 4
byte colPins[Cols] = {19, 21, 22, 23};           // Cols 1 to 4
//
// Initialize the Keypad library with the row and columns defs.
// Here, we have to specify the key map name (keys), row pin
// assignments to GPIO ports (rowPins), and the column pin
// assignments to GPIO ports (colPins)
//
Keypad kpd=Keypad(makeKeymap(keys),rowPins,colPins,Rows,Cols);

//
// Configure BUZZER as an output port
//
void setup()
{
    pinMode(BUZZER, OUTPUT);
    Serial.begin(9600);
```

```

}

// 
void loop()
{
    char keypressed = kpd.getKey();
    if(keypressed != NO_KEY)
    {
        Serial.println(keypressed);
    }
}

```

*Bild 6.35 Programmlisting*

### 6.6.8 Programmbeschreibung

Im Programmkopf wird **Keypad.h** eingebunden und BUZZER dem GPIO-Port 4 zugeordnet. Die (ungefähren) Frequenzen der Noten einer Oktave werden im Array **Notes** gespeichert (das erste Array-Element wird nicht verwendet). Dann werden die PWM-Parameter zur Tonerzeugung definiert: Das Tastverhältnis wird auf 50 %, die Auflösung auf 8 Bit und die Anfangsfrequenz auf 1000 Hz eingestellt. Außerdem wird der PWM-Kanal 0 ausgewählt.

Dann wird eine Tastatur, bestehend aus vier Zeilen und vier Spalten, eingestellt. Das Zeichenarray **keys** definiert die Zeichen, die vom Programm empfangen werden sollen, wenn eine Taste der Tastatur gedrückt wird. Wenn Sie zum Beispiel die Taste S1 oben links drücken, wird das Zeichen „1“ an das Programm gesendet. Das Interface zwischen den GPIO-Ports und der Tastatur ist in dem Array **rowPins** für die Zeilen 1 ... 4 und in **colPins** für die Spalten 1 ... 4 definiert. In der folgenden Anweisung beispielsweise wird Zeile 1 mit GPIO-Port 18, Zeile 2 mit GPIO-Port 5 und so weiter verbunden.

```
byte rowPins[Rows] = {18, 5, 17, 16};
```

Innerhalb des Hauptprogramms sucht der Befehl **kpd.getKey** nach einem Tastendruck. Wenn eine Taste gedrückt wird, ist das, was das Programm empfängt, ein Zeichen zwischen „1“ und „9“ oder „a“ und „g“. Das Zeichen wird in einen numerischen Wert zwischen 1 und 13 umgewandelt und dieser Wert wird dann als Index für das Array **Notes** verwendet, um die korrespondierende Tonfrequenz zu erhalten. Dieser Ton wird dann an den Summer gesendet. Die Tonlänge ist fest auf 250 ms eingestellt, danach wird das Tastverhältnis auf 0 gesetzt, so dass der Summer schweigt.

## 6.7 PROJEKT 6 – Rechner mit Tastatur und LCD

### 6.7.1 Beschreibung

In diesem Projekt wird ein einfacher Rechner für ganze Zahlen entworfen. Dieser „Taschenrechner“ kann ganze Zahlen addieren, subtrahieren, multiplizieren, dividieren und die Ergebnisse auf dem LCD anzeigen. Der Taschenrechner funktioniert wie folgt: Wenn das System eingeschaltet wird, erscheint auf dem LCD zwei Sekunden lang der Text **CALCULATOR**. Dann wird der Text **No1:** in der ersten Zeile des LCDs angezeigt. Der Benutzer gibt

die erste Zahl ein und drückt die **ENTER**-Taste. Daraufhin wird der Text **No2:** in der zweiten Zeile des LCDs angezeigt. Der Benutzer gibt die zweite Zahl ein und drückt abermals auf **ENTER**. Abschließend muss die gewünschte Operator-Taste gedrückt werden. Das Ergebnis erscheint für fünf Sekunden auf dem LC-Display, dann wird es gelöscht und der Rechner ist bereit für die nächste Berechnung. Das folgende Beispiel zeigt, wie die Zahlen 12 und 20 addiert werden können (vom Benutzer einzugebenden Zeichen sind aus Gründen der Übersichtlichkeit **fett** gedruckt):

No1: **15 ENTER**

No2: **20 ENTER**

Op: +

Res = 35

In diesem Projekt haben die Tasten der Tastatur folgende Bedeutung:

1	2	3	4
5	6	7	8
9	0		ENTER
+	-	x	/

Die Taste zwischen 0 und ENTER wird im Projekt nicht verwendet.

### 6.7.2 Das Ziel

Das Projekt soll zeigen, wie eine Tastatur und ein I<sup>2</sup>C-LCD in einem Projekt zusammen verwendet werden können und wie man einen einfachen Taschenrechner entwirft.

### 6.7.3 Blockschaltbild:

Bild 6.36 zeigt das Blockschaltbild des Projekts, bei dem die Tastatur und das I<sup>2</sup>C-LCD mit dem ESP32-DevKitC verbunden werden.

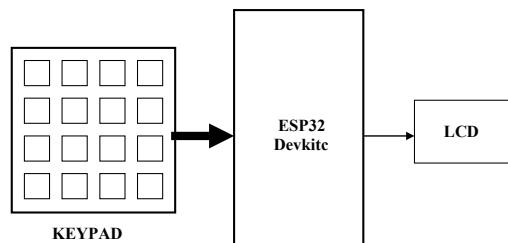


Bild 6.36 Blockschaltung des Projekts

### 6.7.4 Schaltplan

Der Schaltplan des Projekts ist in Bild 6.37 dargestellt. Pin 1 ist der obere Anschluss, wenn sich die Pins links vom Keypad befinden. Beachten Sie Bild 4.2 für die Pin-Belegung des ESP32-DevKitC. Die Anschlüsse unterscheiden sich hier vom vorherigen Projekt, da die Pins 21 und 22 vom I<sup>2</sup>C-LCD-Modul belegt werden. Die Zeilen und Spalten der Tastatur sind wie folgt verbunden:

Tastatur-Zeile	Tastatur-Pin	ESP32 GPIO Port
C4	5	23
C3	6	19
C2	7	18
C1	8	5
R1	4	17
R2	3	16
R3	2	4
R4	1	0

Die I<sup>2</sup>C-LCD-Pins SDA und SCL sind mit den GPIO-Pins 21 und 22 verbunden.

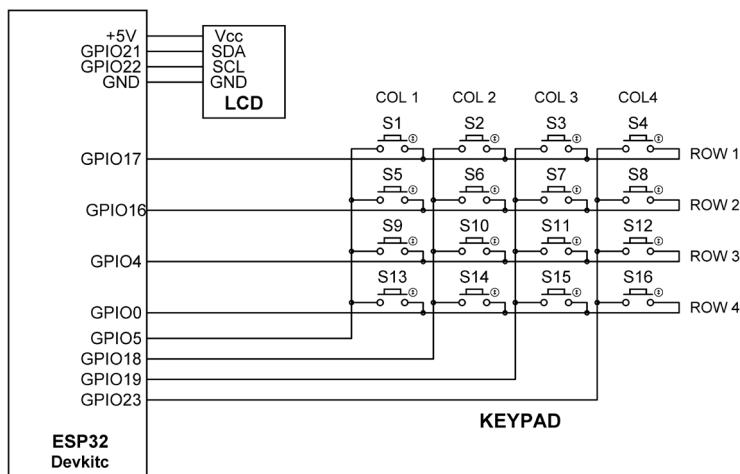


Bild 6.37 Schaltplan des Projekts

### 6.7.5 Konstruktion

ESP32-DevKitC wird zusammen mit dem Tastenfeld und dem I<sup>2</sup>C-LCD auf einem Steckboard montiert (siehe Bild 6.38).

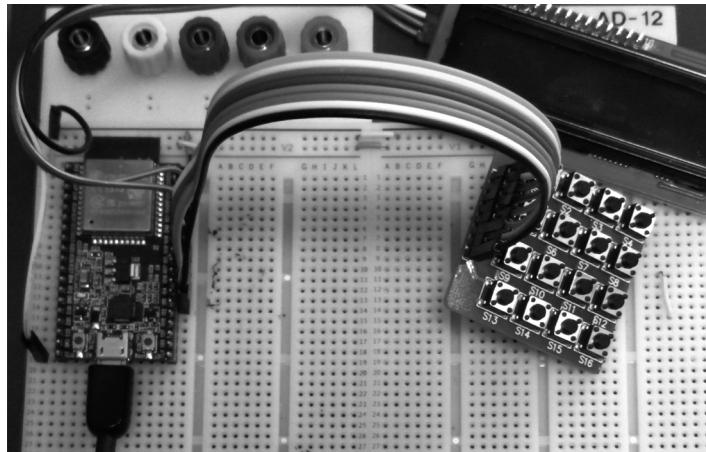


Bild 6.38 Projekt auf einem Steckbrett

### 6.7.6 PDL des Projekts

Stellen Sie wie im vorherigen Projekt sicher, dass die Tastaturlibrary bereits in die Arduino-IDE geladen wurde. Die PDL des Projekts ist in Bild 6.39 zu sehen.

```
BEGIN
    Include the Keypad and the I2C libraries
    Define the keypad and the I2C LCD connections
    Initialize the LCD
    Initialize the Keypad
    Display heading CALCULATOR
DO FOREVER
    Display No1:
    Read the first number and save in Op1
    Display No2:
    Read the second number and store in Op2
    Display Op:
    Read the required operation
    Perform the required operation
    Display the result on the LCD
ENDDO
END
```

Bild 6.39 PDL des Projekts

### 6.7.7 Programmlisting

Vor dem Schreiben des kompletten Programms sollte man die Hardware aufbauen und dann ein Programm verfassen, das die Tastaturschnittstelle testet, indem es die Nummern der Tasten im seriellen Monitor der Arduino-IDE anzeigt, die nacheinander gedrückt werden. Der Programmcode TestKeypad2 ist in Bild 6.40 dargestellt. Führen Sie das Programm aus und drücken Sie die Tasten auf dem Tastenfeld. Im seriellen Monitor sollten die Tastennummern (1 bis 9 oder a bis g) angezeigt werden.

```
*****  
*          CALCULATOR WITH KEYPAD AND LCD  
*          ======  
*  
* In this project a 4x4 keypad is connected to the ESP32-DevKitC  
* boards. In addition, an I2C LCD is connected. The program is a  
* simple integer calculator that can perform the arithmetic  
* operations of addition, subtraction, multiplication and  
* division.  
*  
* The keypad is configured as follows (the key between 0 and  
* ENTER is not used:  
*  
* 1 2 3 4  
* 5 6 7 8  
* 9 0   ENTER  
* + - X /  
*  
*  
* File:  Testkeypad2  
* Date:  July 2017  
* Author: Dogan Ibrahim  
*****/  
#include "Keypad.h"                      // Include Keypad library  
#include <Wire.h>                         // Include I2C library  
#include <LiquidCrystal_I2C.h>  
//  
// Keypad keys  
//  
#define Enter 'E'  
#define Plus '+'  
#define Minus '-'  
#define Multiply '*'  
#define Divide '/'  
  
//  
// Set the LCD address to 0x27 and the configuration to 16 chars  
// by 2 rows display  
//  
LiquidCrystal_I2C lcd(0x27, 16, 2);  
  
//  
// Define Keypad parameters  
//  
const byte Rows = 4;                      // Number of rows  
const byte Cols = 4;                      // Number of columns
```

```
//  
// Define the keymap on the keypad  
//  
char keys[Rows][Cols] =  
{  
    {'1', '2', '3', '4'},  
    {'5', '6', '7', '8'},  
    {'9', '0', ' ', 'E'},  
    {'+', '-', 'X', '/'}  
};  
  
//  
// Define the Keypad connections for the rows and columns  
//  
byte rowPins[Rows] = {17, 16, 4, 0}; // Rows 1 to 4  
byte colPins[Cols] = {5, 18, 19, 23}; // Cols 1 to 4  
  
// Initialize the Keypad library with the row and columns defs.  
// Here, we have to specify the key map name (keys), row pin  
// assignments to GPIO ports (rowPins), and the column pin  
// assignments to GPIO ports (colPins)  
  
//  
Keypad kpd=Keypad(makeKeymap(keys),rowPins,colPins,Rows,Cols);  
  
char MyKey, i, j, op[12], Op1, Op2;  
int Calc;  
  
//  
// Initialize the I2C LCD, turn ON backlight, clear LCD, display  
// heading (CALCULATOR), wait 2 seconds and then clear LCD  
//  
void setup()  
{  
    lcd.begin();  
    lcd.backlight();  
    lcd.clear();  
    lcd.print("CALCULATOR");  
    delay(2000);  
    lcd.clear();  
    Serial.print(9600);  
}  
  
//  
// Inside the main program, read the keypad  
//  
void loop()  
{
```

```
Op1 = 0;
Op2 = 0;
MyKey = 0;

lcd.setCursor(0,0); // Col 0, row 0
lcd.print("No1: "); // Display No1:
while(1)
{
    do
        MyKey = kpd.getKey();
    while(!MyKey); // Wait until key pressed
    if(MyKey == Enter)break; // ENTER key
    if(MyKey == '0')MyKey = 0; // '0' key
    lcd.print(MyKey);
    Op1 = 10*Op1 + MyKey - '0';
}

lcd.setCursor(0,1); // Col0, row 1
lcd.print("No2: "); // Display No2:
while(1)
{
    do
        MyKey = kpd.getKey();
    while(!MyKey); // Wait until key pressed
    if(MyKey == Enter)break; // ENTER key
    if(MyKey == '0')MyKey = 0; // '0' key
    lcd.print(MyKey);
    Op2 = 10*Op2 + MyKey - '0';
}

lcd.clear();
lcd.setCursor(0,0); // Col 0, row 0
lcd.print("Op: "); // Display Op:
do
    MyKey = kpd.getKey();
while(!MyKey);
lcd.clear();
lcd.setCursor(0,0);
lcd.print("Res=");

switch(MyKey)
{
    case Plus:
        Calc = Op1 + Op2;
        break;
    case Minus:
```

```
    Calc = Op1 - Op2;
    break;
  case Multiply:
    Calc = Op1 * Op2;
    break;
  case Divide:
    Calc = Op1 / Op2;
    break;
}
lcd.print(Calc);
delay(5000);
lcd.clear();
}
```

Bild 6.40 Tastatur-Testprogramm

Das vollständige Programmlisting zum Projekt Calculator ist in Bild 6.41 dargestellt.

```
*****
*          CALCULATOR WITH KEYPAD AND LCD
*          =====
*
* In this project a 4x4 keypad is connected to the ESP32-DevKitC
* boards. In addition, an I2C LCD is connected. The program is a
* simple integer calculator that can perform the arithmetic
* operations of addition, subtraction, multiplication and
* division.
*
* The keypad is configured as follows (the key between 0 and
* ENTER is not used:
*
* 1 2 3 4
* 5 6 7 8
* 9 0   ENTER
* + - X /
*
*
* File:  Calculator
* Date:  July 2017
* Author: Dogan Ibrahim
*****
#include "Keypad.h"                      // Include Keypad library
#include <Wire.h>                         // Include I2C library
#include <LiquidCrystal_I2C.h>

// 
// Keypad non-numeric keys
```

```
//  
#define Enter 'E'  
#define Plus '+'  
#define Minus '-'  
#define Multiply '*'  
#define Divide '/'  
  
//  
// Set the LCD address to 0x27 and the configuration to 16 chars  
// by 2 rows display  
//  
LiquidCrystal_I2C lcd(0x27, 16, 2);  
  
//  
// Define Keypad parameters  
//  
const byte Rows = 4; // Number of rows  
const byte Cols = 4; // Number of columns  
//  
// Define the keymap on the keypad  
//  
char keys[Rows][Cols] =  
{  
    {'1', '2', '3', '4'},  
    {'5', '6', '7', '8'},  
    {'9', '0', ' ', 'E'},  
    {'+', '-', '*', '/'}  
};  
//  
// Define the Keypad connections for the rows and columns  
//  
byte rowPins[Rows] = {17, 16, 4, 0}; // Rows 1 to 4  
byte colPins[Cols] = {5, 18, 19, 23}; // Cols 1 to 4  
//  
// Initialize the Keypad library with the row and columns defs.  
// Here, we have to specify the key map name (keys), row pin  
// assignments to GPIO ports (rowPins), and the column pin  
// assignments to GPIO ports (colPins)  
//  
Keypad kpd=Keypad(makeKeymap(keys),rowPins,colPins,Rows,Cols);  
  
char MyKey;  
long Calc, Op1, Op2;  
  
//  
// Initialize the I2C LCD, turn ON backlight, clear LCD, display
```

```
// heading (CALCULATOR), wait 2 seconds and then clear LCD
//
void setup()
{
    lcd.begin();
    lcd.backlight();
    lcd.clear();
    lcd.print("CALCULATOR");           // Display heading
    delay(2000);
    lcd.clear();
}

//
// Inside the main program, read the first number, the second
// number, and the required operation. The program displays the
// result for 5 seconds and after this time the process is
// repeated
//
void loop()
{
    Op1 = 0;
    Op2 = 0;
    MyKey = 0;

    lcd.setCursor(0,0);             // Col 0, row 0
    lcd.print("No1: ");            // Display No1:
    while(1)
    {
        do
            MyKey = kpd.getKey();
        while(!MyKey);              // Wait until key pressed
        if(MyKey == Enter)break;    // ENTER key
        lcd.print(MyKey);          // Echo the key
        Op1 = 10*Op1 + MyKey - '0'; // Entered number so far
    }

    lcd.setCursor(0,1);             // Col0, row 1
    lcd.print("No2: ");            // Display No2:
    while(1)
    {
        do
            MyKey = kpd.getKey();
        while(!MyKey);              // Wait until key pressed
        if(MyKey == Enter)break;    // ENTER key
        lcd.print(MyKey);          // Echo the key
        Op2 = 10*Op2 + MyKey - '0'; // Entered number so far
    }
}
```

```

}

lcd.clear();
lcd.setCursor(0,0);                                // Col 0, row 0
lcd.print("Op: ");                               // Display Op:
do
    MyKey = kpd.getKey();
while(!MyKey);
lcd.clear();
lcd.setCursor(0,0);
lcd.print("Res=");                                // Display "Res="
//
// Calculate the result and display on the LCD
//
switch(MyKey)
{
    case Plus:
        Calc = Op1 + Op2;
        break;
    case Minus:
        Calc = Op1 - Op2;
        break;
    case Multiply:
        Calc = Op1 * Op2;
        break;
    case Divide:
        Calc = Op1 / Op2;
        break;
}
lcd.print(Calc);                                // Display result
delay(5000);                                    // Wait 5 seconds
lcd.clear();
}

```

*Bild 6.41 Programmlisting*

### 6.7.8 Programmbeschreibung

Am Anfang des Programms werden die Dateien **Keypad.h**, **Wire.h** und **LiquidCrystal\_I2C.h** integriert. Dann werden die nicht-numerischen Tastaturtasten definiert. Das LCD erhält die Adresse 0x27 und wird mit 16 Zeichen und zwei Zeilen konfiguriert.

Für die Tastatur werden je vier Zeilen und Spalten eingestellt. Das Zeichen-Array **keys** definiert die Zeichen, die vom Programm empfangen werden sollen, wenn eine Taste der Tastatur gedrückt wird. Wenn man beispielsweise auf der Tastatur die Taste oben links (S1) drückt, wird das Zeichen „1“ an das Programm gesendet. Die Schnittstelle zwischen den GPIO-Ports und der Tastatur ist im Array **rowPins** für die Zeilen 1 ... 4 und im Array **colPins** für die Spalten 1 ... 4 definiert. So ist beispielsweise in der folgenden Anweisung

Zeile 1 mit GPIO-Port 17, Zeile 2 mit GPIO-Port 16 und so weiter verbunden.

```
byte rowPins [Rows] = {17, 16, 4, 0};
```

In der Setup-Routine wird der Text CALCULATOR für zwei Sekunden angezeigt. Der Rest des Programmcodes wird innerhalb der Hauptprogrammschleife ausgeführt. Hier wird **No1:** angezeigt und vom Benutzer erwartet, dass er die erste ganze Zahl eingibt. Die gelesene Zahl wird in der Variablen **Op1** gespeichert. In ähnlicher Weise wird die zweite Zahl gelesen und in der Variablen **Op2** gespeichert. Dann fordert das Programm mit dem Text **Op:** den Benutzer auf, den gewünschten Operator einzugeben. Das Ergebnis wird in einer **Switch**-Anweisung berechnet und für fünf Sekunden in der Form **Res=nn** angezeigt. Danach wird das LCD gelöscht und der Prozess beginnt von neuem.

Bild 6.42 zeigt ein typisches Display, bei der die Zahlen 12 und 2 multipliziert werden müssen. Das Ergebnis wird in Bild 6.43 als **Res=24** angezeigt.

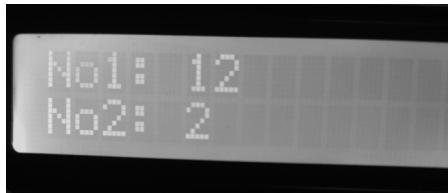


Bild 6.42 Geben Sie die zu berechnenden Zahlen ein



Bild 6.43 Ergebnis der Multiplikation

## 6.8 PROJEKT 7 – HIGH-LOW-SPIEL

### 6.8.1 Beschreibung

Dieses Projekt verwendet eine 4x4-Tastatur und ein LCD wie im vorherigen Projekt, um das klassische High-Low-Spiel zu implementieren. Hier die Spielregeln:

- Der Computer generiert eine geheime Zufallszahl zwischen 1 und 100
- In der oberen Zeile des LCDs erscheint „Guess Now ...“
- Der Spieler versucht, die Zahl zu erraten, indem er diese Zahl mit der Tastatur eingibt und dann die ENTER-Taste drückt.
- Wenn die getippte Zahl höher ist als die Zufallszahl, erscheint in der unteren Zeile des LCDs „HIGH – Try Again“.
- Wenn die getippte Zahl niedriger ist als die Zufallszahl, erscheint in der unteren Zeile des LCDs „LOW – Try Again“.

- Wenn der Spieler die Zahl errät, wird in der unteren Zeile „Well done ...“ angezeigt.
- Das Programm wartet fünf Sekunden und startet das Spiel neu.

In diesem Projekt ist die Tastatur wie folgt gekennzeichnet:

1	2	3	4
5	6	7	8
9	0	NU	ENTER
NU	NU	NU	NU

NU bezeichnet Tasten ohne Funktion (not used).

### 6.8.2 Das Ziel

Das Projekt zeigt, wie ein einfaches Spiel mit einer Tastatur und einem LCD erstellt werden kann.

### 6.8.3 Blockdiagramm:

Die Blockschaltung entspricht der in Bild 6.36.

### 6.8.4 Schaltplan

Der Schaltplan des Projekts ist der gleiche wie in Bild 6.37, in dem Tastatur und LCD mit dem ESP32-DevKitC verbunden sind.

### 6.8.5 Konstruktion

Das ESP32-DevKitC, das Tastenfeld und das I<sup>2</sup>C-LCD werden wie in Bild 6.38 auf einem Steckboard montiert.

### 6.8.6 PDL des Projekts

Die PDL des Projekts ist in Bild 6.44 dargestellt.

```

BEGIN
    Include the Keypad and the I2C libraries
    Define the keypad and the I2C LCD connections
    Initialize the LCD
    Initialize the Keypad
    Display heading HIGH-LOW GAME
    NewNumber = 1
DO FOREVER
    IF NewNumber = 1 THEN
        Generate a random number between 1 and 100
        NewNumber = 0
    ENDIF
    Display message "Guess Now"
    Read a number
    Calculate the difference between the generated and guessed number
  
```

```
    IF the difference > 0 THEN
        Display "HIGH - TRY AGAIN"
    ELSE IF the difference < 0 THEN
        Display "LOW -TRY AGAIN"
    ELSE IF the difference = 0 THEN
        Display "Well done"
        Wait 5 seconds
        NewNumber = 1
    ENDIF
ENDDO
END
```

Bild 6.44 PDL des Projekts

### 6.8.7 Programmlisting

Das Listing des Programms HighLow ist in Bild 6.45 dargestellt.

```
/*
 *          HIGH-LOW GAME
 *=====
 *
 * In this project a 4x4 keypad is connected to the ESP32-DevKitC
 * boards. In addition, an I2C LCD is connected as in the previous
 * project. This project is the classical High-Low game. The
 * processor generates a random number between 1 and 100. The
 * user is required to guess this number. If the number entered
 * is smaller or greater than the generated number then a
 * message is displayed to help the user guess the correct number.
 * The game continues until the user finds the generated number
 *
 * The keypad is configured as follows (NU keys are not used):
 *
 * 1 2 3 4
 * 5 6 7 8
 * 9 0 NU ENTER
 * NU NU NU NU
 *
 *
 * File:    HighLow
 * Date:    July 2017
 * Author:  Dogan Ibrahim
 */
#include "Keypad.h"                      // Include Keypad library
#include <Wire.h>                         // Include I2C library
#include <LiquidCrystal_I2C.h>
// Keypad non-numeric keys
```

```
//  
#define Enter 'E'  
  
//  
// Set the LCD address to 0x27 and the configuration to 16 chars  
// by 2 rows display  
//  
LiquidCrystal_I2C lcd(0x27, 16, 2);  
  
//  
// Define Keypad parameters  
//  
const byte Rows = 4; // Number of rows  
const byte Cols = 4; // Number of columns  
//  
// Define the keymap on the keypad  
//  
char keys[Rows][Cols] =  
{  
    {'1', '2', '3', '4'},  
    {'5', '6', '7', '8'},  
    {'9', '0', ' ', 'E'},  
    {' ', ' ', ' ', ' '}  
};  
//  
// Define the Keypad connections for the rows and columns  
//  
byte rowPins[Rows] = {17, 16, 4, 0}; // Rows 1 to 4  
byte colPins[Cols] = {5, 18, 19, 23}; // Cols 1 to 4  
//  
// Initialize the Keypad library with the row and columns defs.  
// Here, we have to specify the key map name (keys), row pin  
// assignments to GPIO ports (rowPins), and the column pin  
// assignments to GPIO ports (colPins)  
//  
Keypad kpd=Keypad(makeKeymap(keys),rowPins,colPins,Rows,Cols);  
  
long MyGuess;  
char MyKey, NewGame = 1;  
int Guess, Diff;  
  
//  
// Initialize the I2C LCD, turn ON backlight, clear LCD, display  
// heading (HIGH-LOW GAME), wait 2 seconds and then clear LCD  
//  
void setup()
```

```
{  
    lcd.begin();  
    lcd.backlight();  
    lcd.clear();  
    lcd.print("HIGH-LOW GAME");           // Display heading  
    delay(2000);  
    lcd.clear();  
    randomSeed(10);  
}  
  
//  
// Inside the main program, generate a random number between 1  
// and 100. The user attempts to guess this number and sees  
// messages such as HIGH or LOW to help him in making a choice  
// for the next guess  
//  
void loop()  
{  
    if(NewGame == 1)  
    {  
        lcd.clear();  
        lcd.print("Guess Now...");  
        Guess = random(1, 101);           // Generate between 1-100  
        NewGame = 0;  
    }  
    MyGuess = 0;  
    MyKey = 0;  
  
    lcd.setCursor(0,1);                // Col 0, row 0  
    while(1)  
    {  
        do  
            MyKey = kpd.getKey();  
        while(!MyKey);                  // Wait until key pressed  
        if(MyKey == Enter)break;         // ENTER key  
        lcd.print(MyKey);              // Echo the key  
        MyGuess = 10*MyGuess + MyKey - '0'; // Entered number so far  
    }  
  
    Diff = MyGuess - Guess;           // Difference  
    if(Diff > 0)                    // High Guess ?  
    {  
        lcd.setCursor(0,1);  
        lcd.print("HIGH - TRY AGAIN"); // Display HIGH  
        delay(2000);  
        lcd.setCursor(0,1);
```

```

        lcd.print("                ");
    }
    else if(Diff < 0)
    {
        lcd.setCursor(0,1);
        lcd.print("LOW - TRY AGAIN");           // Display LOW
        delay(2000);
        lcd.setCursor(0,1);
        lcd.print("                ");
    }
    else if(Diff == 0)
    {
        lcd.setCursor(0,1);
        lcd.print("Well Done...");            // Correct
        delay(5000);                      // Wait 5 seconds
        NewGame = 1;                      // New game flag set
    }
}

```

*Bild 6.45 Programmlisting*

### 6.8.8 Programmbeschreibung

Im Header des Programms werden die Bibliotheken **Keypad.h**, **Wire.h** und **LiquidCrystal\_i2C.h** aufgenommen, dann die nicht-numerischen Tasten der Tastatur definiert. Das LCD erhält die Adresse 0x27 und wird mit 16 Zeichen und 2 Zeilen konfiguriert.

Innerhalb des Hauptprogramms wird eine zufällige ganze Zahl zwischen 1 und 100 erzeugt und in der Variablen **Guess** gespeichert. Der Spieler gibt eine Zahl ein, die in der Variablen **MyGuess** gespeichert wird. Der Unterschied zwischen der generierten und der geschätzten Zahl wird in der Variablen **Diff** untergebracht. Ist **Diff** größer als Null (die geratene Zahl ist größer als die Zufallszahl), so wird der Text **HIGH – TRY AGAIN** angezeigt und der Benutzer muss es erneut versuchen. Wenn **Diff** kleiner als Null ist (die geratene Zahl ist kleiner als die Zufallszahl), erscheint die Meldung **LOW – TRY AGAIN**. Erst wenn **Diff** gleich Null ist, man die Zahl also endlich erraten hat, wird der Text **Well Done ...** angezeigt. Das Spiel startet nach fünf Sekunden mit einer neuen Zahl.

Die Bilder 6.46, 6.47 und 6.48 zeigen beispielhaft Anzeigen des LC-Displays.

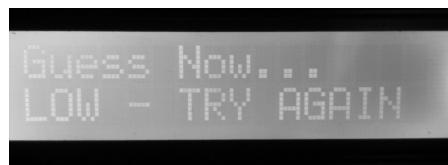
*Bild 6.46 Die geratene Zahl ist niedriger*



Bild 6.47 Die geratene Zahl ist höher



Bild 6.48 Die geratene Zahl stimmt mit der Zufallszahl überein!

## 6.9 PROJEKT 8 – Das kleine Einmaleins

### 6.9.1 Beschreibung

Die meisten Kinder lernen das kleine Einmaleins (engl. times table) in der Grundschule vor ihrem 12. Lebensjahr. Dieses Programm zeigt für eine bestimmte gewünschte Zahl (hier die 5) das kleine Einmaleins auf dem LCD in folgendem Format:

```
1 × 5 = 5  
2 × 5 = 10  
3 × 5 = 15  
.....  
.....  
12 × 5 = 60
```

Dieses Projekt nutzt eine 4x4-Tastatur und ein LCD wie im vorherigen Projekt. Jede Berechnung wird für eine Sekunde angezeigt. Das kleine Einmaleins reicht hier von 1 bis 12 (Zugabe für Hochbegabte!)

In diesem Projekt ist die Tastatur wie folgt aufgebaut:

1	2	3	4
5	6	7	8
9	0	NU	ENTER
NU	NU	NU	NU

NU sind wiederum die nicht im Projekt verwendeten Tasten.

### 6.9.2 Das Ziel

Das Ziel dieses Projekts ist es, Kindern auf spielerische Weise das kleine Einmaleins zu vermitteln (und Eltern, dies zu programmieren).

### 6.9.3 Blockschaltbild:

Die Blockschaltung entspricht der von Bild 6.36.

### 6.9.4 Schaltplan

Der Schaltplan des Projekts ist der gleiche wie in Bild 6.37. Die Tastatur und das LCD sind mit dem ESP32-DevKitC wie im vorherigen Projekt verbunden.

### 6.9.5 Konstruktion

Das ESP32-DevKitC wird zusammen mit dem Tastenfeld und dem I<sup>2</sup>C-LCD auf einem Steckboard montiert (siehe Bild 6.38).

### 6.9.6 PDL des Projekts

Bild 6.49 zeigt die PDL des Projekts.

```
BEGIN
    Include the Keypad and the I2C libraries
    Define the keypad and the I2C LCD connections
    Initialize the LCD
    Initialize the Keypad
    Display heading TIMES TABLE
    NewTable = 1
DO FOREVER
    IF NewTable = 1 THEN
        Prompt for the required number
        NewTable = 0
    ENDIF
    FOR i=1 to 12
        Res = i * required number
        Display i, "X", required number, "=", Res
        Wait 1 second
    ENDFOR
    NewTable = 1
ENDDO
END
```

*Bild 6.49 PDL des Projekts*

### 6.9.7 Programmlisting

Das Listing des Programms TimesTable ist in Bild 6.50 dargestellt.

```
/*
 *          LEARNING THE TIMES TABLE
 *          =====
 *
 * In this project a 4x4 keypad is connected to the ESP32-DevKitC
 * boards. In addition, an I2C LCD is connected as in the previous
 * project. This project displays the times table for a chosen
 * number. Each output is displayed for one second.
 *
 * The keypad is configured as follows (NU keys are not used):
 */
```

```
*  
* 1 2 3 4  
* 5 6 7 8  
* 9 0 NU ENTER  
* NU NU NU NU  
*  
*  
* File: TimesTable  
* Date: July 2017  
* Author: Dogan Ibrahim  
*****  
#include "Keypad.h" // Include Keypad library  
#include <Wire.h> // Include I2C library  
#include <LiquidCrystal_I2C.h>  
//  
// Keypad non-numeric keys  
//  
#define Enter 'E'  
  
//  
// Set the LCD address to 0x27 and the configuration to 16 chars  
// by 2 rows display  
//  
LiquidCrystal_I2C lcd(0x27, 16, 2);  
  
//  
// Define Keypad parameters  
//  
const byte Rows = 4; // Number of rows  
const byte Cols = 4; // Number of columns  
//  
// Define the keymap on the keypad  
//  
char keys[Rows][Cols] =  
{  
    {'1', '2', '3', '4'},  
    {'5', '6', '7', '8'},  
    {'9', '0', ' ', 'E'},  
    {' ', ' ', ' ', ' '}  
};  
//  
// Define the Keypad connections for the rows and columns  
//  
byte rowPins[Rows] = {17, 16, 4, 0}; // Rows 1 to 4  
byte colPins[Cols] = {5, 18, 19, 23}; // Cols 1 to 4  
//
```

```
// Initialize the Keypad library with the row and columns defs.  
// Here, we have to specify the key map name (keys), row pin  
// assignments to GPIO ports (rowPins), and the column pin  
// assignments to GPIO ports (colPins)  
//  
Keypad kpd=Keypad(makeKeymap(keys),rowPins,colPins,Rows,Cols);  
  
int Res, MyChoice, NewTable = 1;  
char MyKey;  
  
//  
// Initialize the I2C LCD, turn ON backlight, clear LCD, display  
// heading (TIMES TABLE) for 2 seconds  
//  
void setup()  
{  
    lcd.begin();  
    lcd.backlight();  
    lcd.clear();  
    lcd.print("TIMES TABLE"); // Display heading  
    delay(2000); // Wait 2 seconds  
}  
  
//  
// Inside the main program, read the number whose times table is  
// required and then display the times table for this number  
//  
void loop()  
{  
    if(NewTable == 1) // If new table  
    {  
        lcd.clear();  
        lcd.print("Which Number ?"); // No for the table  
        NewTable = 0;  
    }  
  
    MyKey = 0;  
    MyChoice = 0;  
  
    lcd.setCursor(0,1); // Col 0, row 1  
    while(1)  
    {  
        do  
            MyKey = kpd.getKey();  
        while(!MyKey); // Wait until key pressed  
        if(MyKey == Enter)break; // ENTER key
```

```

lcd.print(MyKey);                                // Echo the key
MyChoice = 10*MyChoice + MyKey - '0';           // Entered number so far
}

lcd.clear();
lcd.print("Timetable for ");                    // Display heading
lcd.print(MyChoice,10);                         // Display no
for(int i = 1; i <=12; i++)                     // Do for 1 to 12
{
    lcd.setCursor(0, 1);                         // Goto col0, row 1
    lcd.print("      ");
    lcd.setCursor(0,1);
    Res = i * MyChoice;                         // Calculate the result
    lcd.print(i,DEC);                           // Display the times table
    lcd.print(" X ");
    lcd.print(MyChoice, 10);
    lcd.print(" = ");
    lcd.print(Res, 10);
    delay(1000);                               // Wait 1 second
}
lcd.clear();
NewTable = 1;                                    // Set new table flag
}

```

Bild 6.50 Programmlisting

### 6.9.8 Programmbeschreibung

Im Header des Programms werden genau wie in den vorherigen Tastatur- und LCD-Projekten die Dateien **Keypad.h**, **Wire.h** und **LiquidCrystal\_i2C.h** eingebunden und dann die nicht-numerischen Tasten definiert. Das LCD mit der Adresse 0x27 wird mit 16 Zeichen und 2 Zeilen konfiguriert. In der Setup-Routine wird das LCD initialisiert, die Hintergrundbeleuchtung eingeschaltet und der Text TIMES TABLE wird für zwei Sekunden eingeblendet. Im Hauptprogramm wird der Benutzer aufgefordert, die Zahl einzugeben, deren Einmaleins gezeigt werden soll. Diese Zahl wird in der Variablen **MyChoice** gespeichert. Dann sorgt eine **for**-Schleife dafür, dass im Sekudentakt alle Multiplikatoren von 1 bis 12 durchlaufen und das entsprechende Einmaleins angezeigt wird. Wenn das Einmaleins beendet ist, wird das LCD gelöscht und das Flag **NewTable** auf 1 gesetzt, so dass ein neues Einmaleins angezeigt werden kann.

Bild 6.51 zeigt beispielhaft eine Anzeige aus dem Programm.



Bild 6.51 Beispieldarstellung

## 6.10 PROJEKT 9 – Lernen der Grundmathematik

### 6.10.1 Beschreibung

Dieses Projekt zielt darauf ab, Kindern grundlegende Mathematik (hier die Multiplikation) zu vermitteln. Es werden zwei Integer-Zufallszahlen zwischen 1 und 100 erzeugt. Diese Zahlen werden multipliziert, aber das Ergebnis für 30 Sekunden nicht angezeigt. In dieser Zeit soll das Kind das Resultat herausfinden und danach mit der vom Programm angezeigten Lösung vergleichen.

Das Programm läuft typisch ab wie folgt:

```
Multiplication  
26 X 5 =  
(wartet 30 Sekunden)  
26 X 5 = 130
```

### 6.10.2 Das Ziel

Ziel dieses Projekts ist es, Kindern grundlegende Mathematik (hier Multiplikation) zu vermitteln.

### 6.10.3 Blockschaltbild:

Die Blockschaltung entspricht der aus Bild 5.65.

### 6.10.4 Schaltplan

Der Schaltplan des Projekts ist der gleiche wie in Bild 5.66.

### 6.10.5 Konstruktion

Das ESP32-DevKitC wird mit dem I<sup>2</sup>C-LCD auf einem Steckboard angebracht, wie in Bild 5.67 zu sehen.

### 6.10.6 PDL des Projekts

Die PDL des Projekts ist in Bild 6.52 dargestellt.

```
BEGIN
    Include the I2C library
    Define the I2C LCD connections
    Initialize the LCD
    Display heading BASIC MATHS
    DO FOREVER
        Generate two integer random numbers between 1 and 100
        Multiply these two numbers
        Display the numbers but not the result of the multiplication
        Wait 30 seconds
        Display the result
    ENDDO
END
```

*Bild 6.52 PDL des Projekts*

### 6.10.7 Programmlisting

Die Programmlisting Maths des Projekts ist in Bild 6.53 dargestellt.

```
*****  
*          BASIC MATHEMATICS  
*          =====  
*  
* In this project an I2C LCD is connected to the ESP32-DevKitC.  
* The program generates two integer random numbers between 1 and  
* 100 and multiplies these numbers. The result is not shown for  
* 30 seconds where the program waits so that the user can  
* calculate the result. After 30 seconds the result is displayed  
* so that the user can check with his/her own result.  
*  
* The I2C LCD is connected to the ESP32 Devkitc as in the  
* previous projects, i.e. using the SDA and SCL I2C lines  
*  
* File:    Maths  
* Date:    July 2017  
* Author:  Dogan Ibrahim  
*****  
#include <Wire.h>                                // Include I2C library  
#include <LiquidCrystal_I2C.h>  
  
//  
// Set the LCD address to 0x27 and the configuration to 16 chars  
// by 2 rows display  
//  
LiquidCrystal_I2C lcd(0x27, 16, 2);  
  
int FirstNumber, SecondNumber, Res;  
  
//  
// Initialize the I2C LCD, turn ON backlight, clear LCD, display  
// heading (MATHS) for 2 seconds  
//  
void setup()  
{  
    lcd.begin();                                     // Initialize LCD  
    lcd.backlight();                                 // Backlight ON  
    lcd.clear();                                    // Clear LCD  
    lcd.print("BASIC MATHS");                      // Display heading  
    delay(2000);                                    // Wait 2 seconds  
    randomSeed(10);  
}
```

```

// Inside the main program, read the number whose times table is
// required and then display the times table for this number
//
void loop()
{
    lcd.clear();
    lcd.print("Multiplication"); // Display heading
    FirstNumber = random(1, 101); // First number
    SecondNumber = random(1, 101); // Second number
    Res = FirstNumber * SecondNumber; // Result
    lcd.setCursor(0, 1); // Go to col 0, row 1
    lcd.print("      ");
    lcd.setCursor(0,1);
    lcd.print(FirstNumber,DEC); // Display the question
    lcd.print(" X ");
    lcd.print(SecondNumber, 10);
    lcd.print(" = ");
    delay(30000); // Wait 30 seconds
    lcd.print(Res, 10);
    delay(2000); // Wait 2 seconds
}

```

Bild 6.53 Programmlisting

### 6.10.8 Programmbeschreibung

Im Header des Programms werden wie in den vorherigen LCD-Projekten wieder die Daten **Wire.h** und **LiquidCrystal\_i2C.h** eingebunden. Das LCD wird in der Setup-Routine initialisiert, die Hintergrundbeleuchtung eingeschaltet und der Text BASIC MATHS im LCD angezeigt.

Innerhalb des Hauptprogramms werden zwei Integer-Zufallszahlen mit der Bezeichnung **FirstNumber** und **SecondNumber** generiert und deren Produkt in der Variablen **Res** gespeichert. Das Programm zeigt die Zahlen in der zweiten LCD-Zeile an, aber für 30 Sekunden nicht das Ergebnis. Während dieser Zeit soll das Kind das Ergebnis berechnen. Dann wird das Ergebnis auch angezeigt, damit das Kind es mit dem eigenen Ergebnis vergleichen kann.

### 6.10.9 Vorschläge

Obwohl in diesem Projekt nur multipliziert werden kann, lässt sich das Programm so modifizieren, dass alle vier Operatoren für Addition, Subtraktion, Multiplikation und Division möglich sind.

Bild 6.54 zeigt eine typische Anzeige.

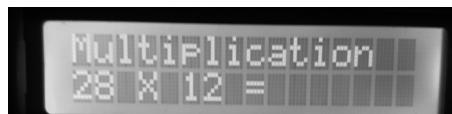


Bild 6.54 Eine typische Anzeige

## 6.11 PROJEKT 10 – Code-Türschloss

### 6.11.1 Beschreibung

Dieses Projekt ist ein elektronisches Code-Türschloss mit einer Tastatur und einem LCD. Eine vierstellige Geheimnummer (kann geändert werden) wird im System gespeichert. Schaltet man das System ein, wird die Meldung **Enter Code:** angezeigt. Der Benutzer muss den richtigen Code eingeben, damit (in diesem Projekt) ein Relais aktiviert wird, das das Schloss öffnet. Es wird davon ausgegangen, dass ein elektrischer Verriegelungsmechanismus an das Relais angeschlossen ist. Wenn der korrekte Code eingegeben und das Relais aktiviert wird, erscheint die Meldung **Door Opened** im Display. Bei einem falschen Code wird dagegen die Meldung **Wrong Code** angezeigt. Aus Sicherheitsgründen wird nach drei aufeinanderfolgenden Fehleingaben das Schloss für eine Minute gesperrt und die Meldung **Failed to open** angezeigt. Als Geheimcode wird in diesem Beispiel 1234 gewählt.

### 6.11.2 Das Ziel

Ziel dieses Projekts ist es, ein elektronisches Code-Türschloss mit dem ESP32-DevKitC, einer Tastatur, einem LCD und einem Relais zu entwerfen.

### 6.11.3 Blockschaltbild:

Das Blockschaltbild des Projekts ist in Bild 6.55 dargestellt.

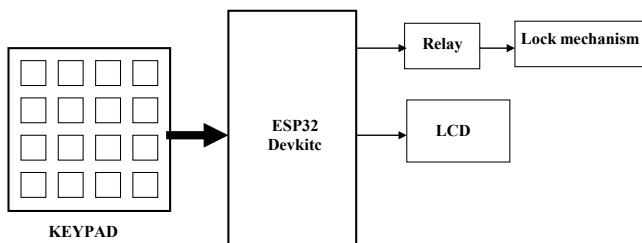


Bild 6.55 Blockschaltung des Projekts

### 6.11.4 Schaltplan

Der Schaltplan des Projekts ist der gleiche wie in Bild 6.56. Die Tastatur und das LCD sind wie in Projekt 8 mit ESP32-DevKitC verbunden, Anschluss S des Relais (siehe Bild 6.3) mit GPIO-Port 2.

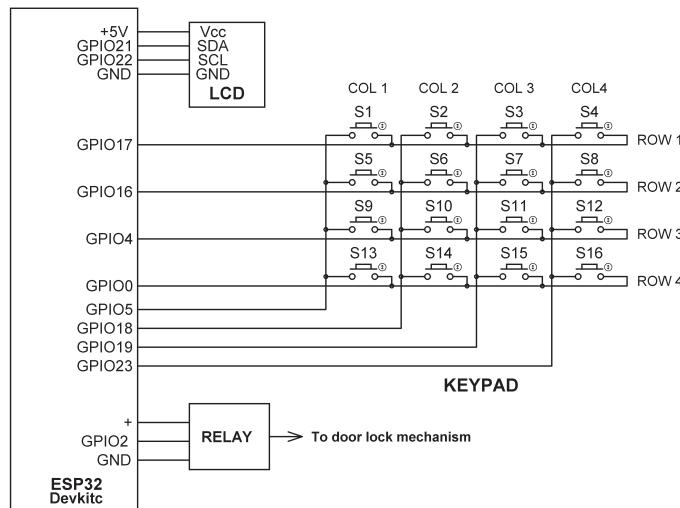


Bild 6.56 Schaltplan des Projekts

### 6.11.5 Konstruktion

ESP32-DevKitC, Tastatur, LCD und Relais sind wie in Bild 6.57 auf einem Steckboard montiert.

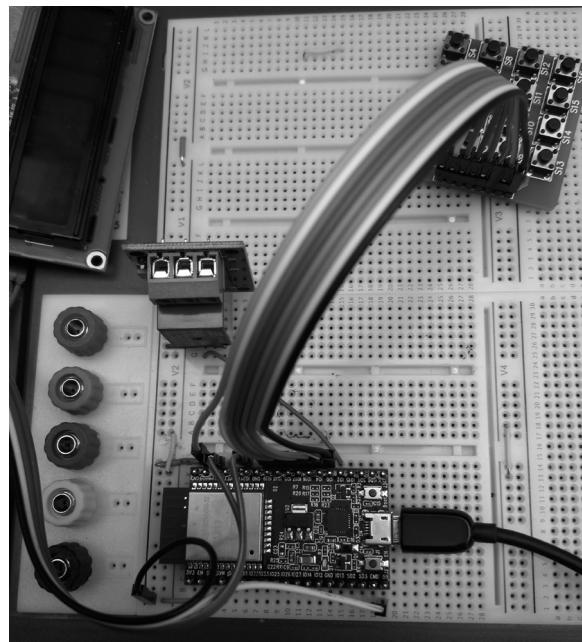


Bild 6.57 Das Projekt auf einem Steckboard

### 6.11.6 PDL des Projekts

Die PDL des Projekts ist in Bild 6.58 dargestellt.

```
BEGIN
    Include the Keypad and the I2C libraries
    Define the keypad and the I2C LCD connections
    Define the secret number
    Initialize the LCD
    Initialize the Keypad
    Clear attempt number
    DO FOREVER
        Display Enter code:
        Read user code
        IF correct code entered THEN
            Activate the relay
            Wait 10 seconds
            Deactivate the relay
            Clear attempt number
        ELSE IF wrong code entered THEN
            IF 3rd attempt THEN
                Display failure message
                Wait for 1 minute
                Clear attempt number
            ELSE
                Increment attempt number
                Display wrong code message
                Wait 2 seconds
            ENDIF
        ENDIF
    ENDDO
END
```

*Bild 6.58 Die PDL des Projekts*

### 6.11.7 Programmlisting

Das Programmlisting Lock des Projekts ist in Bild 6.59 zu sehen.

```
/*
 *          KEYPAD LOCK
 *=====
 *
 * This is an electronic lock project using a keypad, LCD, and
 * a buzzer. A 4 digit secret number is stored in memory. The
 * lock mechanism is electrically operated and is connected to
 * a relay. The user must enter the correct code in order to
 * activate the relay and open the door. If wrong code is entered
 * on three consequent attempts then 30 second delay is introduced
 * to the system for security.
 *
 * The relay is connected to GPIO port 2 of the ESP32-DevKitC.
```

```
*  
* The keypad is configured as follows (NU keys are not used):  
*  
* 1 2 3 4  
* 5 6 7 8  
* 9 0 NU ENTER  
* NU NU NU NU  
*  
* File: Lock  
* Date: July 2017  
* Author: Dogan Ibrahim  
*****  
#define SecretNumber 1234 // Secret number  
#define RELAY 2 // Relay at port 0  
#include "Keypad.h" // Include Keypad library  
#include <Wire.h> // Include I2C library  
#include <LiquidCrystal_I2C.h>  
//  
// Keypad non-numeric keys  
//  
#define Enter 'E'  
  
//  
// Set the LCD address to 0x27 and the configuration to 16 chars  
// by 2 rows display  
//  
LiquidCrystal_I2C lcd(0x27, 16, 2);  
  
//  
// Define Keypad parameters  
//  
const byte Rows = 4; // Number of rows  
const byte Cols = 4; // Number of columns  
//  
// Define the keymap on the keypad  
//  
char keys[Rows][Cols] =  
{  
    {'1', '2', '3', '4'},  
    {'5', '6', '7', '8'},  
    {'9', '0', ' ', 'E'},  
    {' ', ' ', ' ', ' '}  
};  
//  
// Define the Keypad connections for the rows and columns  
//
```

```
byte rowPins[Rows] = {17, 16, 4, 0};           // Rows 1 to 4
byte colPins[Cols] = {5, 18, 19, 23};          // Cols 1 to 4
//
// Initialize the Keypad library with the row and columns defs.
// Here, we have to specify the key map name (keys), row pin
// assignments to GPIO ports (rowPins), and the column pin
// assignments to GPIO ports (colPins)
//
Keypad kpd=Keypad(makeKeymap(keys),rowPins,colPins,Rows,Cols);

int Res, MyNumber, Attempts = 0;
char MyKey;

//
// Initialize the I2C LCD, turn ON backlight, clear LCD
//
void setup()
{
    pinMode(RELAY, OUTPUT);                      // Relay is output
    digitalWrite(RELAY, LOW);                     // Relay off
    lcd.begin();                                // Initialize LCD
    lcd.backlight();                            // Backlight ON
    lcd.clear();                                // Clear LCD
    attachInterrupt(digitalPinToInterrupt(4),blink,CHANGE);
}

void blink()
{
}

//
// Inside the main program, prompt for the code to be entered.
// If the code is correct then open the door (activate the relay).
// If the code is wrong, display message to say that it is wrong.
// After three failures stop the system for 1 minute for security.
//
void loop()
{
    lcd.clear();
    lcd.print("Enter code:");                   // Prompt for the code

    MyKey = 0;
    MyNumber = 0;

    lcd.setCursor(0,1);                        // Col 0, row 1
    while(1)                                  // Get user code
```

```

{
    do
        MyKey = kpd.getKey();
        while(!MyKey);                                // Wait until key pressed
        if(MyKey == Enter)break;                      // ENTER key
        lcd.print(MyKey);                            // Echo the key
        MyNumber = 10*MyNumber + MyKey - '0';         // Entered number so far
    }

    if(MyNumber == SecretNumber)                  // Correct code ?
    {
        Attempts = 0;
        digitalWrite(RELAY, HIGH);                  // Open door
        lcd.clear();
        lcd.print("Door Opened");                 // Display opened
        delay(10000);                            // Wait 10 seconds
        digitalWrite(RELAY, LOW);                  // Close door
    }
    else                                         // Wrong code
    {
        Attempts++;
        if(Attempts == 3)                         // 3 failures ?
        {
            Attempts = 0;                        // Clear attempts
            lcd.setCursor(0,1);                  // Go to col 0, row 1
            lcd.print("Failed to open");        // Display Wrong code
            delay(60000);                       // Wait 1 minute
        }
        else                                     // Less than 3 attempts
        {
            lcd.setCursor(0,1);                  // Goto col 0, row 1
            lcd.print("Wrong code...");        // Display wrong code
            delay(2000);                        // Wait 2 seconds
            lcd.clear();                        // Clear LCD
        }
    }
}
}

```

Bild 6.59 Programmlisting

### 6.11.8 Programmbeschreibung

Zu Beginn des Programms wird in der Variablen **SecretNumber** der Geheimcode (in diesem Beispiel 1234, kann aber auf Wunsch geändert werden) gespeichert. Dann werden genau wie in den vorherigen Tastatur- und LCD-Projekten die Header-Dateien **Keypad.h**, **Wire.h** und **LiquidCrystal\_i2C.h** in das Programm aufgenommen. In der Setup-Routine wird auch das Relais als Ausgangs-Port konfiguriert und zunächst ausgeschaltet. Das LCD wird initialisiert, die Hintergrundbeleuchtung eingeschaltet und der Display-Inhalt gelöscht.

Innerhalb des Hauptprogramms wird die Meldung **Enter code:** ausgegeben und der Benutzer damit aufgefordert, den Geheimcode einzugeben, wobei eingegebene Code dann in der Variablen **MyNumber** gespeichert wird. Ist der eingegebene Code korrekt, wird das Relais aktiviert, um die Tür zu öffnen. Wenn der falsche Code eingegeben wird, hat der Benutzer noch zwei Versuche frei. Nach drei vergeblichen Versuchen jedoch wird das System für eine Minute blockiert. Dies geschieht, um die Sicherheit zu erhöhen, damit der Benutzer nicht verschiedene Codes ausprobieren kann. Bild 6.60 zeigt das Beispiel einer Anzeige im Display.



Bild 6.60 Beispieldisplay

### 6.11.9 Vorschläge

Versuchen Sie, das Programm zu ändern, indem Sie die Stellen der Geheimzahl auf sechs erhöhen. Fügen Sie dem System einen Summer hinzu, der ertönt, wenn ein falscher Code eingegeben wurde.

### 6.12 Zusammenfassung

In diesem Kapitel haben wir den Entwurf etwas komplizierterer Projekte mit dem ESP32-DevKitC und der Arduino-IDE-Programmierumgebung angesehen. Im nächsten Kapitel soll es um die Entwicklung netzwerkbasierter Anwendungen mit der Arduino-IDE für das ESP32-DevKitC gehen.

## KAPITEL 7 • ESP32-DEVKITC NETZWERKPROGRAMMIERUNG MIT DER ARDUINO-IDE

### 7.1 Übersicht

Im letzten Kapitel haben wir komplexere ESP32-Devkit-Projekte mit der Arduino-IDE als Programmierumgebung entwickelt.

Die Wi-Fi-Fähigkeit ist eine leistungsstarke Funktion und ein Pluspunkt des ESP32-Prozessors. In diesem Kapitel werden wir uns deshalb Wi-Fi-Netzanwendungen des ESP32-DevKitC ansehen, die wiederum in der Arduino-IDE entwickelt werden.

Die Abkürzung Wi-Fi (Wireless Fidelity) für ein drahtloses Funknetzwerk ist besonders im nicht-deutschen Sprachraum gebräuchlich, aber eigentlich falsch. Richtig wäre WLAN (Wi-reless Local Area Network), wobei Wi-Fi nur einen technischen Standard für bestimmte, meist öffentliche WLANs bezeichnet. Der Unterschied zwischen WLAN und Wi-Fi lässt sich vergleichen mit dem zwischen Musik und HiFi. Im Folgenden werden aber nur lokale Funknetzwerke nach dem Wi-Fi-Standard behandelt, so dass die Abkürzung Wi-Fi hier durchaus gerechtfertigt ist.

### 7.2 Scannen der erreichbaren Wi-Fi-Netzwerke

In einigen Anwendungen möchten wir die nähere Umgebung nach Wi-Fi-Netzwerken durchsuchen und deren Eigenschaften ermitteln. Dazu kann man die Funktion **WiFi.scanNetworks()** verwenden. Wenn Netzwerke gefunden werden, gibt diese Funktion die folgenden Parameter zurück:

- Gesamtzahl der gefundenen Netzwerke
- Netzwerknamen (SSID)
- Netzwerk-Signalstärken (RSSI)
- MAC-Adressen (BSSIDstr)
- Verschlüsselungsarten

Bild 7.1 zeigt das Programm SCAN, das die nähere Umgebung nach Wi-Fi-Netzwerken durchsucht und die verschiedenen Netzwerkparameter zurückgibt.

```
/*
 *           SCAN WI-FI NETWORK
 *
 * This program does a scan and displays the surrounding Wi-Fi
 * networks. The following parameters of the found networks
 * are displayed: Network name (SSID), signal strength (RSSI),
 * and MAC address (BSSIDstr).
 *
 * File:   SCAN
 * Author: Dogan Ibrahim
 * Date:   August, 2017
```

```
*  
*****  
#include "WiFi.h"  
  
//  
// Set mode to station, scan the surrounding Wi-Fi networks and  
// display their details on the Serial Monitor  
//  
void setup()  
{  
    Serial.begin(9600);  
    WiFi.mode(WIFI_STA); // Mode station  
  
    int N = WiFi.scanNetworks(); // Scan networks  
    if(N > 0) // If networks found  
    {  
        Serial.println("");  
        Serial.print(N);  
        Serial.println(" Networks found");  
        for (int i = 0; i < N; ++i)  
        {  
            Serial.print(i + 1);  
            Serial.print(": ");  
            Serial.print("Network Name: ");  
            Serial.print(WiFi.SSID(i));  
            Serial.print(" ");  
            Serial.print("Signal Strength: ");  
            Serial.print(WiFi.RSSI(i));  
            Serial.print(" ");  
            Serial.print("MAC Address: ");  
            Serial.println(WiFi.BSSIDstr(i));  
            delay(50);  
        }  
    }  
}  
  
void loop()  
{  
    // no code  
}
```

*Bild 7.1 Scan-Programm für Wi-Fi-Netzwerke*

In dem obigen Programm werden die zurückgegebenen Daten im seriellen Monitor angezeigt. Ein Beispiel (nur ein Ausschnitt) davon ist in Bild 7.2 zu sehen.

```

17 Networks found
1: Network Name: Chromecast7872.b Signal Strength: -43 MAC Address: FA:8F:CA:56:8D:3F
2: Network Name: BTWifi-with-FON Signal Strength: -51 MAC Address: CA:91:F9:66:5A:B1
3: Network Name: BTHomeSpot-XNH Signal Strength: -56 MAC Address: 00:07:26:C7:AC:20
4: Network Name: BTWifi-X Signal Strength: -60 MAC Address: EA:91:F9:66:5A:B1
5: Network Name: HP-Print-19-LaserJet 200 color Signal Strength: -66 MAC Address: 9C:D2
6: Network Name: BTOpenzone-B Signal Strength: -69 MAC Address: 12:81:D8:78:27:BA
7: Network Name: BTWifi Signal Strength: -74 MAC Address: 02:81:D8:78:27:BA
8: Network Name: BTHub3-MFK8 Signal Strength: -75 MAC Address: 00:81:D8:78:27:BA
9: Network Name: BTHomeSpot-XNH Signal Strength: -76 MAC Address: 00:07:26:C7:AC:24
10: Network Name: DIRECT-CC-HB OfficeJet Pro 6960 Signal Strength: -81 MAC Address: 30:1
11: Network Name: BTHub4-GMP9 Signal Strength: -83 MAC Address: 44:E9:DD:5D:1D:90
12: Network Name: BTWifi-with-FON Signal Strength: -84 MAC Address: 44:E9:DD:5D:1D:93
13: Network Name: VM4929888 Signal Strength: -89 MAC Address: 84:16:F9:65:FF:00

```

Bild 7.2 Ausgabe der ermittelten Netzwerkparameter

### 7.3 Verbindung zu einem Wi-Fi-Netzwerk herstellen

Die Verbindung zu einem Wi-Fi-Netzwerk ist einfach herzustellen. Wir müssen nur die Funktion **WiFi.begin** mit dem spezifischen Netzwerknamen und dem spezifischen Passwort als Argumente aufrufen, um den ESP32 mit dem gewünschten Netzwerk zu verbinden. Dabei sollte man immer den Wi-Fi-Status überprüfen, um sicherzustellen, dass die Verbindung erfolgreich aufgenommen wurde. Um den Wi-Fi-Status einer Verbindung zu überprüfen, kann die Konstante **WL\_CONNECTED** verwendet werden.

Im Bild 7.3 ist das Programm CONNECT zu sehen, das eine Verbindung zum lokalen Wi-Fi-Netzwerk mit dem Namen **BTHomeSpot-XNH** herstellt. In diesem Programm wird der Verbindungsstatus auf dem seriellen Monitor angezeigt (siehe Bild 7.4).

```

/*
 *          CONNECT TO A WI-FI NETWORK
 *=====
 *
 * This program connect to the local Wi-Fi network with the
 * name BTHomeSpot-XNH. The connection status is displayed on
 * the Serial Monitor.
 *
 *
 * File:    CONNECT
 * Author:  Dogan Ibrahim
 * Date:   August, 2017
 *
 */
#include "WiFi.h"
const char* ssid = "BTHomeSpot-XNH";
const char* password = "49345abaeb";

//
// Set mode to station, scan the surrounding Wi-Fi networks and
// display their details on the Serial Monitor
//

```

```
void setup()
{
    Serial.begin(9600);
    Serial.println("");
    WiFi.begin(ssid, password);
    while(WiFi.status() != WL_CONNECTED)
    {
        Serial.println("Attempting to connect...");
        delay(1000);
    }
    Serial.println("Connected to Wi-Fi network");
    //
    // Display the local IP address and the MAC address, and then
    // disconnect from the Wi-Fi
    //
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
    Serial.print("Mac address: ");
    Serial.println(WiFi.macAddress());
    WiFi.disconnect(true);
}

void loop()
{
    // no code
}
```

Bild 7.3 Verbindung zu einem Wi-Fi-Netzwerk herstellen

```
Attempting to connect...
Attempting to connect...
Attempting to connect...
Attempting to connect...
Connected to Wi-Fi network
```

Bild 7.4 Anzeige des Verbindungsstatus

Die lokale IP-Adresse und die MAC-Adresse können nach einer Verbindung zu einem Wi-Fi-Netzwerk einfach ermittelt werden. Die Funktion **WiFi.localIP()** gibt die lokale IP-Adresse zurück, die Funktion **WiFi.macAddress()** in ähnlicher Weise die im Programm CONNECT2 (Bild 7.5) angezeigte MAC-Adresse. Bild 7.6 zeigt beispielhaft eine Ausgabe des Programms.

```
*****
*                      CONNECT TO A WI-FI NETWORK
*                      =====
*
* This program connect to the local Wi-Fi network with the
* name BTHomeSpot-XNH. The connection status is displayed on
* the Serial Monitor. In addition, the local IP address and the
```

```
* MAC address are displayed. The program disconnects from the
* network before stopping
*
*
* File: CONNECT2
* Author: Dogan Ibrahim
* Date: August, 2017
*
*****
#include "WiFi.h"
const char* ssid = "BTHomeSpot-XNH";
const char* password = "49341abaeb";

//
// Set mode to station, scan the surrounding Wi-Fi networks and
// display their details on the Serial Monitor
//
void setup()
{
    Serial.begin(9600);
    Serial.println("");
    WiFi.begin(ssid, password);
    while(WiFi.status() != WL_CONNECTED)
    {
        Serial.println("Attempting to connect...");
        delay(1000);
    }
    Serial.println("Connected to Wi-Fi network");
//
// Display the local IP address and the MAC address, and then
// disconnect from the Wi-Fi
//
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
    Serial.print("Mac address: ");
    Serial.println(WiFi.macAddress());
    WiFi.disconnect(true);
}

void loop()
{
    // no code
}
```

Bild 7.5 Das Programm gibt die IP-Adresse und die MAC-Adresse zurück

```
Attempting to connect...
Attempting to connect...
Attempting to connect...
Connected to Wi-Fi network
IP address: 192.168.1.156
Mac address: 30:AE:A4:05:5B:E0
```

Bild 7.6 Ausgabe des Programms

## 7.4 HTTP-GET-Anfragen

Das HTTP (HyperText Transfer Protocol) wurde entwickelt, um die Kommunikation zwischen Client-Servern in einem Netzwerk zu ermöglichen. Es gibt mehrere Operationen, die mit HTTP ausgeführt werden können, wie GET, PUT, LOAD, DELETE, HEAD, TRACE und so weiter. Die GET-Methode wird zum Abrufen von Informationen von einem Webserver verwendet, wobei die URL des Servers angegeben wird. Die Information wird nur abgerufen, der Inhalt des Servers aber nicht geändert.

Bild 7.7 zeigt das Programm **HTTPGet**, das Informationen von der Website <http://httpbin.org/ip> abruft. Diese Website gibt die lokale IP-Adresse des Anrufers zurück. Zu Beginn des Programms werden die Bibliotheken WiFi.h und HTTPClient.h in das Programm aufgenommen. In der Setup-Routine verbindet sich das Programm dann mit dem lokalen Wi-Fi. Im Hauptprogramm wird das Objekt **HTTPClient** deklariert und die URL der Website mit der Anweisung **http.begin** angegeben. Der Rückgabewert der HTTP-GET-Anfrage wird in der Variablen **HttpRetCode** gespeichert. Zurückgegebene Daten werden in der Variablen **Contents** gespeichert und dann mit dem HTTP-Rückgabewert im seriellen Monitor angezeigt.

```
/****************************************************************************
 *          HTTP GET EXAMPLE
 *=====
 *
 * This is an example of using the HTTP GET service to retrieve
 * information from a web site. In this example the web site:
 * http://httpbin.org/ip is used as an example, where this URL
 * returns the local IP address. The retrieved information is
 * displayed on the Serial Monitor together with the HTTP return
 * code
 *
 * File:    HTTPGet
 * Author:  Dogan Ibrahim
 * Date:   August, 2017
 *
 */
#include "WiFi.h"
#include <HTTPClient.h>

const char* ssid = "BTHomeSpot-XNH";
const char* password = "49345abaeb";
```

```
//  
// Set mode to station, and connect to the local Wi-Fi  
//  
void setup()  
{  
    Serial.begin(9600);  
    WiFi.begin(ssid, password);  
    while(WiFi.status() != WL_CONNECTED)  
    {  
        Serial.println("Attempting to connect...");  
        delay(1000);  
    }  
    Serial.println("Connected to local Wi-Fi");  
  
}  
  
//  
// Retrieve information from site:  
// http://httpbin.org/ip and display on the Serial Monitor  
//  
void loop()  
{  
    HTTPClient http;  
    http.begin("http://httpbin.org/ip");           // Specify site  
    int HttpRetCode = http.GET();                  // HTTP GET request  
  
    if(HttpRetCode > 0)                          // If data returned  
    {  
        Serial.println("Received data...");         // Display the data  
        String Contents = http.getString();  
        Serial.println(HttpRetCode);                // Display the code  
        Serial.println(Contents);                   // Display the contents  
        http.end();                                // End the connection  
    }  
}
```

Bild 7.7 Das Programm HTTP Get Request

Bild 7.8 zeigt die Ausgabe auf dem seriellen Monitor. In diesem Beispiel lautet der Rückgabewert 200, der dem Erfolg entspricht. Einige der anderen wichtigen Rückgabecodes sind:

<b>200</b>	OK	erfolgreich
<b>202</b>	Accepted	Die Anfrage wurde akzeptiert, aber die Verarbeitung ist noch nicht abgeschlossen.
<b>206</b>	Partial Content	Die Anfrage muss ein Content-Range-Header-Feld enthalten, um den gewünschten Bereich anzuzeigen.
<b>301</b>	Moved Permanently	Die angeforderte Ressource hat eine neue permanente URL, alle zukünftigen Anfragen sollten an diese Adresse gerichtet werden.
<b>404</b>	Not Found	URL wurde nicht gefunden/wurde abgelehnt.
<b>408</b>	Request Time-out	Zeitüberschreitung der Anfrage
<b>409</b>	Conflict	Es besteht ein Konflikt mit der Ressource.
<b>414</b>	URI Too Long	Der URL ist zu lang.
<b>500</b>	Internal Server Error	Unerwarteter interner Serverfehler
<b>505</b>	HTTP Version not supported	Die HTTP-Version wird vom Server nicht unterstützt.

```

Attempting to connect...
Attempting to connect...
Connected to local Wi-Fi
Received data...
200
{
    "origin": "109.159.164.148"
}

Received data...
200
{
    "origin": "109.159.164.148"
}

```

Bild 7.8 Ausgabe des Programms

## 7.5 Verwenden der Socket-Bibliothek

Die Socket-Bibliothek wird hauptsächlich in der Netzwerk-Kommunikation verwendet. Die beiden am häufigsten verwendeten Protokolle zum Senden und Empfangen von Daten über ein Netzwerk sind UDP und TCP/IP. TCP/IP ist ein zuverlässiges Protokoll mit Handshaking, das die zuverlässige Zustellung von Datenpaketen an das gewünschte Ziel gewährleistet. UDP ist zwar nicht so zuverlässig, aber ein schnelleres Protokoll.

Tabelle 7.1 Vergleich der Kommunikation von UDP und TCP/IP.

<b>TCP/IP</b>	<b>UDP</b>
verbindungsorientiertes Protokoll	verbindungsloses Protokoll
langsam	schnell
sehr zuverlässige Datenübertragung	nicht so zuverlässig

Pakete geordnet	Pakete nicht geordnet
Headergröße 20 Byte	Headergröße 8 Byte
Fehlerprüfung und erneute Übertragung	keine Fehlerprüfung
Bestätigung des Datenempfang	keine Bestätigung
Verwendung in HTTP, HTTPS, FTP usw.	Verwendung in DNS, DHCP, TFTP usw.

Tabelle 7.1 Vergleich von UDP und TCP/IP

Programme mit dem UDP- oder TCP/IP-Protokoll sind Server-Client-basiert, wobei ein Knoten Daten sendet und der andere Knoten diese Daten empfängt (und umgekehrt). Daten werden über Ports übertragen, die bei Server und Clients gleich sein müssen.

In den nächsten Abschnitten werden wir untersuchen, wie Programme mit UDP- und TCP/IP-Protokoll für den ESP32-DevKitC mit der Programmierumgebung Arduino-IDE geschrieben werden können.

### 7.5.1 UDP-Programme

UDP ist ein verbindungsloses Protokoll. Das bedeutet, dass keine Verbindung zum Zielknoten hergestellt werden muss, bevor ein Paket gesendet werden kann. Die Kommunikation zwischen einem Server und einem Client funktioniert im Prinzip wie folgt:

#### Server

- Definieren der IP-Adresse und der Port-Nummer des Zielknotens
- Konstruieren des Datenpakets
- Einen Sockel erstellen
- Socket an den lokalen Port binden
- Daten vom Client empfangen
- Daten an den Client senden
- Verbindung schließen

#### Client

- Definieren der IP-Adresse und der Port-Nummer des Zielknotens
- Einen Sockel erstellen
- Daten an den Server senden
- Daten vom Server empfangen
- Schließen der Verbindung

Beachten Sie, dass sowohl der Server als auch der Client Datenpakete senden und voneinander empfangen können. Das folgende Beispiel zeigt, wie Daten von einem UDP-Programm gesendet und empfangen werden.

#### Beispiel 7.1

Schreiben Sie ein UDP-Programm, um eine Textnachricht von einem UDP-Client zu empfangen, und geben Sie die Nachricht auf dem seriellen Monitor aus. Senden Sie dann die Nachricht **Hello From the Server** an den Client. Verwenden Sie die folgenden IP-Adressen

und Port-Nummern für Server und Client:

IP-Adresse des Servers (ESP32-DevKitC)	192.168.1.156
IP-Adresse des Client (PC)	192.168.1.71

In diesem Beispiel ist der ESP32-DevKitC der UDP-Server und der PC der UDP-Client. Die lokale Port-Nummer wird auf 5000 eingestellt.

### Lösung 7.1

Bild 7.9 zeigt den Aufbau des Testsystems mit den IP-Adressen und den Port-Nummern. Um unser Programm zu testen, benötigen wir ein weiteres Programm. Doch anstatt ein anderes ESP32-Programm zu schreiben, können wir auf ein fertiges UDP-Paketprogramm für den PC zurückgreifen, das UDP- und TCP/IP-Pakete senden und empfangen kann. Ein solches Programm ist **Packet Sender**, das kostenlos unter dem Link: <https://packetsender.com/download> zur Verfügung steht.

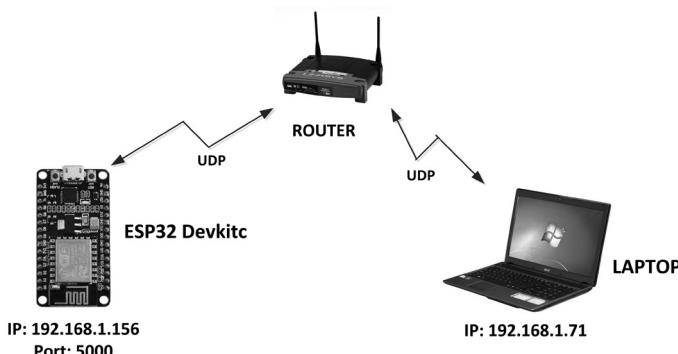


Bild 7.9 Aufstellung des Testsystems

Laden Sie das Programm herunter und extrahieren es in einen Ordner. Wir verwenden nun das Programm **Packet Sender**, um UDP-Pakete zu senden und von unserem ESP32-DevKitC-Programm zu empfangen. Doppelklicken Sie dann auf die Datei **Packet Sender**, um das Programm zu starten. Geben Sie Ihrem Test einen Namen (zum Beispiel **Test packet**) und dann die Nachricht ein, die an das DevKitC-Serverprogramm gesendet werden soll (etwa **Hello from the client**). Geben Sie unten das Ziel (IP des DevKitC) und daneben das Protokoll UDP ein (siehe Bild 7.10).

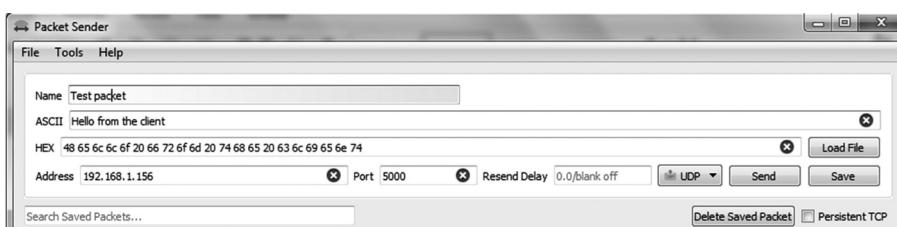


Bild 7.10 Konfiguration im Programm Packet Sender

Das UDP-Programm UDP für das ESSP32 DevKitC ist in Bild 7.11 zu sehen. Zu Beginn des Programms werden die Bibliotheken WiFi und WiFiUDP eingebunden, Netzwerkname und Passwort des lokalen Wi-Fi-Routers angegeben und die lokale Port-Nummer sowie die entfernte IP-Adresse eingetragen. In der Setup-Routine wird eine Verbindung zum lokalen Netzwerk hergestellt und das Programm wird so eingestellt, dass es auf eine Verbindung wartet.

Im Hauptprogramm wird das empfangene Paket analysiert, die Parameter wie Paketlänge, entfernte IP-Adresse und entfernte Port-Nummer werden extrahiert und auf dem seriellen Monitor angezeigt. Das Programm liest dann den Inhalt des empfangenen Datenpakets (**Hello from the client**) und zeigt ihn an. Mit **udp.printf** wird dann die Nachricht **Hello from the server** an den Client gesendet. Schließlich trennt das Programm die Verbindung.

```
/*
 *          UDP SERVER EXAMPLE
 *
 * This is an UDP server example. The program sends the message
 * "Hello from the Server" to an UDP client. The IP addresses
 * and port numbers of the server and the client are:
 *
 * Server (ESP32-DevKitC) IP address: 192.168.1.156 Port: 5000
 * Client (PC) IP address:           192.168.1.71
 *
 *
 * File:    UDP
 * Author:  Dogan Ibrahim
 * Date:   August, 2017
 *
 */
#include "WiFi.h"
#include <WiFiUDP.h>

const char* ssid = "BTHomeSpot-XNH";
const char* password = "49345abaeb";
const int UDPPort = 5000;
const char* DestUDPAddress = "192.168.1.71";
char Buffer[80];
WiFiUDP udp;

// Set mode to station, and connect to the local Wi-Fi. Display
// the local IP address
//
void setup()
{
```

```
Serial.begin(9600);
WiFi.begin(ssid, password);
while(WiFi.status() != WL_CONNECTED)
{
    Serial.println("Attempting to connect...");
    delay(1000);
}
Serial.println("Connected to local Wi-Fi");
Serial.print("Local IP address is: ");
Serial.println(WiFi.localIP());
udp.begin(UDPPort);                                // listen to incoming packets
}

// Parse the received packet and get its details, such as the
// packet size, remote IP address, and remote port number. Then
// display the contents of the received packet which is stored
// in variable Buffer
//
void loop()
{
    udp.beginPacket(DestUDPAddress, UDPPort);
    int packetSize=udp.parsePacket();
    if(packetSize)
    {
        Serial.print("Received packet size is: ");
        Serial.print(packetSize);
        Serial.println(" characters");
        Serial.print("Packet received from: ");
        Serial.println(udp.remoteIP());
        Serial.print("Port address is: ");
        Serial.println(udp.remotePort());
    }
    int len = udp.read(Buffer, 80);
    if(len > 0)
    {
        Serial.println(Buffer);
    }
    udp.printf("Hello from the Server");
    delay(2000);
    udp.endPacket();
}
```

Bild 7.11 Programmlisting

Klickt man im **Packet Sender** auf den **Send**-Knopf, wird das Paket an das ESP32-DevKitC gesendet. Bild 7.12 zeigt die empfangenen Daten, die vom ESP32-DevKitC auf dem seriell-

len Monitor angezeigt werden. Bild 7.13 zeigt den **Packet Sender** mit dem empfangenen und dem an den ESP32-DevKitC gesendeten UDP-Paket.

```
Attempting to connect...
Attempting to connect...
Connected to local Wi-Fi
Local IP address is: 192.168.1.156
Received packet size is: 21 characters
Packet received from: 192.168.1.71
Port address is: 55184
Hello from the client
```

Bild 7.12 Anzeige im seriellen Monitor

							<input checked="" type="checkbox"/> Log Traffic	Save Log	Save Traffic Packet	Copy to Clipboard
Time	From IP	From Port	To IP	To Port	Method	Error	ASCII	Hex		
1 12:09:29.905 pm	192.168.1.156	5000	You	55184	UDP		Hello from the Server	48 65 6C 6C 6f 20 66 72 6f 6D 20 74 68		
2 12:09:27.423 pm	You	55184	192.168.1.156	5000	UDP		Hello from the client	48 65 6c 6c 6f 20 66 72 6f 6d 20 74 68 6		
3 12:08:50.215 pm	You	55184	192.168.1.156	5000	UDP		Hello from the client	48 65 6c 6c 6f 20 66 72 6f 6d 20 74 68 6		

Bild 7.13 Anzeige im Packet Sender

### 7.5.2 TCP/IP-Programme

TCP/IP ist ein verbindungsbasierter und zuverlässiges Protokoll. Das folgende Beispiel soll zeigen, wie dieses Protokoll für den Datenaustausch mit dem ESP32-DevKitC verwendet werden kann.

#### Beispiel 7.2

Schreiben Sie ein TCP/IP-Programm, um eine Textnachricht von einem TCP/IP-Client zu empfangen, und zeigen Sie diese Nachricht auf dem seriellen Monitor an. Senden Sie daraufhin die Nachricht **Hello From the server** an den Client.

Verwenden Sie die IP-Adressen und Port-Nummern für den Server und den Client:

IP-Adresse Server (ESP32-DevKitC)	192.168.1.156
IP-Adresse Client (PC)	192.168.1.71

In diesem Beispiel ist der ESP32-DevKitC der TCP/IP-Server, der PC der TCP/IP-Client. Die lokale Port-Nummer ist auf 5000 eingestellt.

#### Lösung 7.2

Das Testsystem ist wie in Bild 7.9 aufgebaut, allerdings wird hier TCP/IP anstelle von UDP verwendet. Bild 7.14 zeigt das Listing des Programms TCP. Wieder wird das Programm mit dem **Packet Sender** getestet, aber das Protokoll ist auf TCP gesetzt. Am Ende der Nachricht ist ein Punkt eingefügt. Die Nachricht lautet also „**Hello from the client**“. Am Anfang des Programms wird die Portadresse des Servers definiert, in der Setup-Routine eine Verbindung zum lokalen Wi-Fi-Router hergestellt, die lokale IP-Adresse angezeigt und der Server so eingestellt, dass er auf eine Verbindung zum Client wartet. Innerhalb des Hauptprogramms wird ein Paket vom Client gelesen, bis das Punkt-Zeichen erkannt wird. Dann wird eine Nachricht zum Client gesendet.

```
/*****
 *          TCP/IP SERVER EXAMPLE
 *          =====
 *
 * This is a TCP/IP server example. The program sends the message
 * "Hello from the Server" to a TCP/IP client. The IP addresses
 * and port numbers of the server and the client are:
 *
 * Server (ESP32-DevKitC) IP address: 192.168.1.156 Port: 5000
 * Client (PC) IP address:           192.168.1.71
 *
 *
 * File:    TCP
 * Author:  Dogan Ibrahim
 * Date:   August, 2017
 *
 *****/
#include "WiFi.h"

const char* ssid = "BTHomeSpot-XNH";
const char* password = "49345abaeb";
const int TCPPort = 5000;
char Buffer[80];
WiFiServer server(TCPPort);

// 
// Set mode to station, and connect to the local Wi-Fi. Display
// the local IP address, and begin listening for clients
//
void setup()
{
    Serial.begin(9600);
    WiFi.begin(ssid, password);
    while(WiFi.status() != WL_CONNECTED)
    {
        Serial.println("Attempting to connect...");
        delay(1000);
    }
    Serial.println("Connected to local Wi-Fi");
    Serial.print("Local IP address is: ");
    Serial.println(WiFi.localIP());
    server.begin();                                // Begin listening for
clients
}

//
```

```

// Read the packet from the client until character "." detected,
// and then send the message "Hello from the Server" to the
// client
//
void loop()
{
    WiFiClient client = server.available();           // A client connected?
    while(client.connected() && !client.available())
    {
        delay(1);
    }
    String line=client.readStringUntil('.');
    Serial.println(line);
    client.printf("Hello from the Server");
    client.stop();

}

```

*Bild 7.14 Programmlisting*

Durch einen Klick auf die Schaltfläche **Send** im **Packet Sender** wird das Paket an das ESP32-DevKitC gesendet. Bild 7.15 zeigt die Daten, die vom ESP32-DevKitC empfangen und auf dem seriellen Monitor angezeigt werden. Bild 7.16 zeigt das vom ESP32-DevKitC empfangene TCP-Paket und die zurückgeschickte Antwort.

```

Attempting to connect...
Connected to local Wi-Fi
Local IP address is: 192.168.1.156

Hello from the client

```

*Bild 7.15 Anzeige des seriellen Monitors*

Clear Log							<input checked="" type="checkbox"/> Log Traffic	Save Log	Save Traffic Packet	Copy to Clipboard
Time	From IP	From Port	To IP	To Port	Method	Error	ASCII			Hex
1 3:35:47.052 pm	192.168.1.156	5000	You	14211	TCP					
2 3:35:46.923 pm	192.168.1.156	5000	You	14211	TCP		Hello from the Server	48 65 6C 6C 6F 20 66 72 6F 6D 20 74 68 65		
3 3:35:46.808 pm	You	14211	192.168.1.156	5000	TCP		Hello from the client.	48 65 6C 6C 6F 20 66 72 6F 6D 20 74 68 65		

*Bild 7.16 Fenster des Packet Sender*

## 7.6 Zusammenfassung

In diesem Kapitel haben wir die Netzwerkprogrammierung des Entwicklungsbretts ESP32-DevKitC mit der Arduino-IDE als Programmierumgebung kennengelernt. In den nächsten Kapiteln werden wir weitere Projekte mit dem ESP32-DevKitC in der Arduino-IDE entwickeln.

## KAPITEL 8 • PROJEKT – TEMPERATUR UND FEUCHTE IN DER CLOUD

### 8.1 Übersicht

Im letzten Kapitel haben wir gesehen, wie man Netzwerkprogramme für das ESP32-DevKitC Entwicklungsboard schreibt.

In diesem Kapitel werden wir eine weitere interessante Netzwerk-Anwendung entwickeln: Das System ermittelt die Umgebungstemperatur und die Luftfeuchtigkeit mit einem Sensor und speichert diese Daten alle 30 Sekunden in einer Cloud, so dass man von überall auf die Daten zugreifen kann.

### 8.2 Das Blockschaltbild

Das Blockschaltbild des Systems ist in Bild 8.1 dargestellt. In diesem Projekt wird wie in Kapitel 5.3 ein Temperatur- und Feuchtesensor DHT11 verwendet.

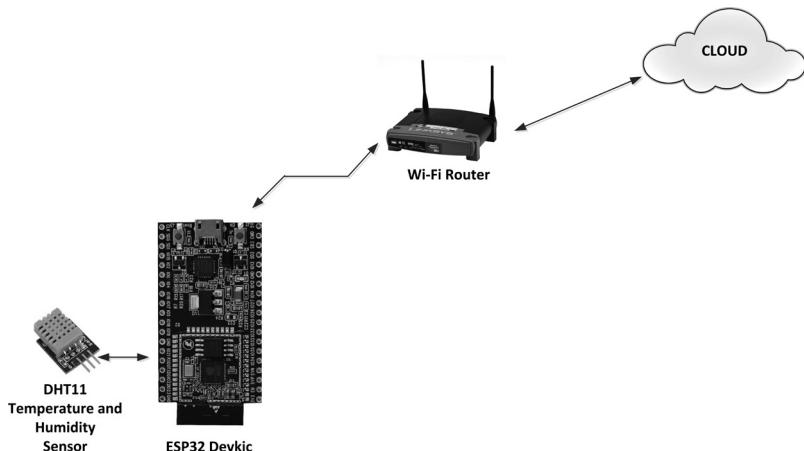


Bild 8.1 Blockschaltbild des Projekts

Der Schaltplan des Projekts ist der gleiche wie in Bild 5.12. Der Datenausgang des DHT11 ist am GPIO-Port 23 des Entwicklungsboards ESP32-DevKitC angeschlossen. Die Schaltung ist auf einem Steckbrett aufgebaut, wie in Bild 5.13 zu sehen.

### 8.3 Die Cloud

Es gibt mehrere Cloud-Dienste, die zum Speichern von Daten verwendet werden können, zum Beispiel **SparkFun**, **Thingspeak**, **Cloudino**, **Bluemix**. In diesem Projekt nehmen wir die Dienste des kostenlosen Cloud-Service Thingspeak in Anspruch, bei dem die Sensorsdaten mit einfachen HTTP-Anfragen gespeichert und abgerufen werden können. Bevor Sie den Dienst Thingspeak einsetzen können, muss auf dessen Webseite ein Konto erstellt werden, bei dem Sie sich immer anmelden können. Öffnen Sie ein neues Konto, indem Sie Ihre E-Mail-Adresse auf der Webseite

[https://thingspeak.com/users/sign\\_up](https://thingspeak.com/users/sign_up)

angeben und ein Passwort wählen. Sie sollten eine E-Mail erhalten, um Ihr Konto zu überprüfen und zu aktivieren. Klicken Sie danach auf **Continue**, erhalten Sie eine Mitteilung wie in Bild 8.2, dass die Anmeldung erfolgreich verlaufen ist. Klicken Sie auf OK, um die Bedingungen zu akzeptieren.

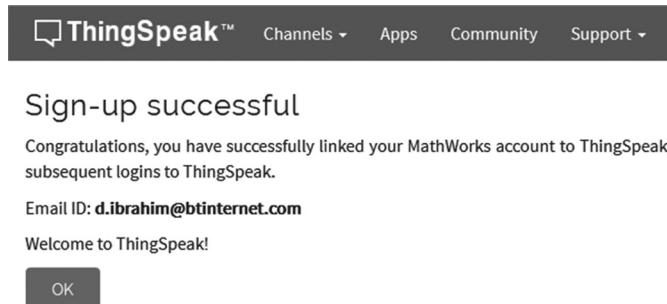


Bild 8.2 Erfolgreiche Anmeldung bei Thingspeak

Dann sollten Sie im Tab **New Channel** einen neuen Kanal erstellen. Füllen Sie das Formular wie in Bild 8.3 gezeigt aus. Geben Sie der Anwendung den Namen **MyWeather** und erstellen Sie zwei Kanäle mit den Namen **Temperature** und **Humidity**. Aktivieren Sie diese Kanäle, um sie öffentlich zu machen.

 A screenshot of the 'New Channel' creation form on the ThingSpeak website. The header bar is identical to the one in the previous screenshot. The main form has fields for 'Name' (set to 'MyWeather'), 'Description' (set to 'Get the ambient temperature and humidity data'), and three 'Field' sections. 'Field 1' is 'Temperature' with a checked checkbox. 'Field 2' is 'Humidity' with a checked checkbox. 'Field 3' is empty with an unchecked checkbox.

Bild 8.3 Einen neuen Kanal erstellen (nur ein Teil des angezeigten Formulars)

Klicken Sie unten im Formular auf **Save Channel**. Jetzt können die beiden Kanäle mit Ihren Daten verwendet werden. Sie sehen nun Registerkarten mit den folgenden Namen:

- **Private View:** Diese Registerkarte zeigt private Informationen zu Ihrem Kanal an, die nur Sie sehen können.
- **Public View:** Wenn Ihr Kanal öffentlich ist, können Sie auf dieser Registerkarte ausgewählte Felder und Kanalvisualisierungen anzeigen.

- **Channel Settings:** Auf dieser Registerkarte werden alle Kanaloptionen angezeigt, die Sie bei der Erstellung festgelegt haben. Sie können den Kanal auf dieser Registerkarte bearbeiten, die Daten löschen oder den Kanal löschen.
- **API Keys:** Diese Registerkarte zeigt Ihre Kanal-API-Schlüssel an. Verwenden Sie diese Schlüssel, um Ihre Kanäle zu lesen und zu beschreiben.
- **Data Import/Export:** Über diese Registerkarte können Sie Kanaldaten importieren und exportieren.

Sie können auf diese Registerkarten klicken und den Inhalt sehen, um bei Bedarf Korrekturen vorzunehmen. Klicken Sie auf die Registerkarte **API Keys** und notieren Sie Ihre **Write API**, **Read API** sowie die **Channel ID** (und bringen Sie die Notizen an einem sicheren Ort unter). Der API-Schlüssel und die Kanalnummer in diesem Projekt sind:

API Key = QAAHT5XIK3ZB8736  
Channel Number = 265433

#### 8.4 Programmlisting

Schon als dieses Buch verfasst wurde, unterstützte die Arduino-IDE die Thingspeak-Bibliothek. Diese Bibliothek macht die Programmierung eigentlich sehr einfach. Aber leider war der Support nur für die Arduino-Hardware-Familie und den ESP8266-Prozessor gedacht, es gab keine Bibliotheksunterstützung für den ESP32-Prozessor. Aus diesem Grund mussten wir von Grund auf programmieren und die Daten mithilfe von TCP/IP-Datenpaketen an die Cloud senden.

Bild 8.4 zeigt das vollständige Programmlisting CLOUD des Projekts. Zu Beginn des Programms werden die Bibliotheks-Header für Wi-Fi und den DHT11-Sensor eingebunden. Der lokale Wi-Fi-Name und das Kennwort werden angegeben, der API-Schlüssel und der Thingspeak-Hostname definiert.

Die Funktion **Connect\_WiFi** stellt eine Verbindung zum lokalen WLAN-Router her. Die Funktion **TempHum** liest Umgebungstemperatur und Luftfeuchtigkeit und speichert die Werte in den Variablen **T** und **H**. Innerhalb der Setup-Routine wird der DHT11 initialisiert, die Funktion **Connect\_WiFi** aufgerufen und über Port 80 eine Verbindung zum HTTP hergestellt.

Das Hauptprogramm läuft in einer Endlosschleife, deren Code alle 30 Sekunden ausgeführt wird. Umgebungstemperatur und Luftfeuchtigkeit werden durch Aufruf der Funktion **TempHum** gelesen. Mit dem folgenden Code wird ein HTTP-GET-Request an die Thingspeak-Website gesendet:

```
client.print ("GET /update?api_key=QAAHT5XIK3ZB8736&field1=");  
client.print (String (T));  
client.print ("& field2=");  
client.print (String(H));  
client.print ("HTTP/1.0\r\nHost:api.thingspeak.com\r\n\r\n");
```

Der API-Schlüssel und die aktuellen Messwerte für Temperatur und Luftfeuchte werden in Strings konvertiert und als field1 beziehungsweise field2 in den HTTP-GET-Request aufgenommen. Der eigentliche String lautet:

```
GET /Update?api_key=QAAHT5XIK3ZB8736&field1=String(T)&-
field2=String(H)
HTTP/1.0\r\nHost:api.thingspeak.com\r\n\r\n\r\n
```

Die Daten werden von der Thingspeak-Website automatisch gespeichert. Der folgende Link kann verwendet werden, um die geplotteten Daten zu sehen:

[https://api.thingspeak.com/channels/YOUR\\_CHANNEL\\_ID](https://api.thingspeak.com/channels/YOUR_CHANNEL_ID)

wobei in diesem Projekt die Channel ID 265433 lautet. Deshalb lautet die Verknüpfung zu den Daten:

<https://api.thingspeak.com/channels/265433>

```
*****
*          TEMPERATURE AND HUMIDITY ON THE CLOUD
*          =====
*
* In this project a DHT11 type temperature and humidity sensor
* chip is connected to GPIO port 23 of the ESP32-DevKitC. The
* program reads the ambient temperature and humidity and sends
* this data to the cloud every 60 seconds where it can be
* accessed from anywhere on Earth. In addition, change of the
* temperature and the humidity can be plotted using the cloud
* services.
*
* The program uses the Thingspeak cloud service. The ESP32
* must be connected to the Wi-Fi network before we can send
* data to the cloud
*
*
* File:    CLOUD
* Author:  Dogan Ibrahim
* Date:   August, 2017
*
*****
#include "WiFi.h"
//
// DHT11 sensor library includes
//
#include <Adafruit_Sensor.h>
#include "DHT.h"
```

```
#define DHT11_PIN 23
#define DHTTYPE DHT11
//
// Local Wi-Fi name and password
//
const char* ssid = "BTHomeSpot-XNH";
const char* password = "49341abaeb";
//
// Thingspeak API key, channel number, and host name
//
unsigned long myChannelNumber = 265433;
String APIKEY = "QAAHT5XIK3ZB8736";
const char* host = "api.thingspeak.com";

int T, H;
DHT dht(DHT11_PIN, DHTTYPE);
WiFiClient client;

//
// This function connects the ESP32-DevKitC to the local Wi-Fi
// network. The network name and passwords are specified earlier
void Connect_WiFi()
{
    WiFi.begin(ssid, password);
    while(WiFi.status() != WL_CONNECTED)
    {
        delay(1000);
    }
}

//
// This function reads the ambient temperature and humidity and
// stores in variables T and H respectively
//
void TempHum()
{
    H = dht.readHumidity();                      // Read humidity
    T = dht.readTemperature();                    // Read temperature
}

//
// Initialize DHT11, connect to local Wi-Fi, connect to HTTP
//
void setup()
```

```

{
    dht.begin();
    Connect_WiFi();
    client.connect(host, 80);
}

// This is the main program. Read the ambient temperature and
// humidity and send the data to Thingspeak every 30 seconds
//
void loop()
{
    client.connect(host,80);
    TempHum();
    client.print("GET /update?api_key=QAAHT5XIK3ZB8736&field1=");
    client.print(String(T));
    client.print("&field2=");
    client.print(String(H));
    client.print(" HTTP/1.0\r\nHost: api.thingspeak.com\r\n\r\n");
    delay(30000);
}

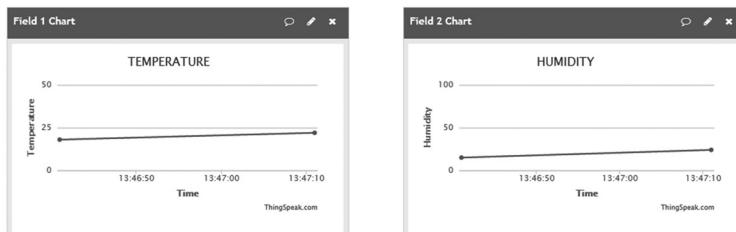
```

*Bild 8.4 Programmlisting des Projekts*

Eine vom Thingspeak gezeichnete Darstellung der Messwerte ist in Bild 8.5 dargestellt.

#### Channel Stats

Created: about 5 hours ago  
 Updated: 30 minutes ago  
 Last entry: 30 minutes ago  
 Entries: 14

*Bild 8.5 Eine Beispieldarstellung von Thingspeak*

Als dieses Buch geschrieben wurde, war es nicht möglich, die Diagramme zu konfigurieren, um zum Beispiel die vertikale oder die horizontale Achse, das Startdatum der Graphen und Ähnliches zu ändern. Es kann jedoch auch eine separate Graphik mit dem gewünschten Startdatum (und der Zeit) erstellt werden, indem man einen Weblink angibt, in dem die Kanalnummer und das Startdatum angegeben sind (weitere Informationen dazu finden Sie auf der Thingspeak-Website):

<HTTP://pi.thingspeak.com/channels/265433/charts/1?start=2017-07-31>

Es besteht auch die Möglichkeit, die Diagrammdaten beispielsweise in ein Excel-Tabellenkalkulationsprogramm zu exportieren und dann ein Diagramm der Daten zu zeichnen. Dies lässt sich erreichen, indem man auf Data Export klickt und das Format CSV wählt. Mit dieser Option hat man mehr Einfluss auf die Konfiguration des Diagrammtyps, der Achsen, Beschriftungen und so weiter.

## **8.5 Zusammenfassung**

In diesem Kapitel haben wir eine interessante Anwendung entwickelt, bei der die Umgebungstemperatur und die Luftfeuchtigkeit mit dem DHT11-Sensorchip ermittelt und diese Daten dann an die Cloud gesendet werden, so dass jederzeit und überall auf der Erde darauf zugegriffen werden kann.

Im nächsten Kapitel geht es um eine Webserver-Anwendung, bei der zwei LEDs am ESP32-DevKitC-Entwicklungsboard von einer Web-Anwendung ferngesteuert werden.

## KAPITEL 9 • WEB-BASIERTE FERNSTEUERUNG

### 9.1 Übersicht

Im letzten Kapitel haben wir den Entwurf und die Entwicklung einer interessanten Wi-Fi-Anwendung kennengelernt, die Umgebungstemperatur und Luftfeuchtigkeit erfasst und in der Cloud speichert.

In diesem Kapitel werden wir eine weitere interessante Anwendung entwickeln, bei der das System über eine HTTP-Webserver-Anwendung zwei LEDs an einer ESP32-DevKitC-Entwicklungsplatine steuert. Obwohl in diesem Beispiel zwei LEDs verwendet werden, können natürlich auch andere Bauteile wie zum Beispiel Relais an das DevKitC angeschlossen und jede Art von elektrischen Geräten ferngesteuert werden.

### 9.2 Das Blockschaltbild

Das Blockschaltbild des Systems ist in Bild 9.1 dargestellt, Bild 9.2 zeigt den Schaltplan. LED0 und LED1 sind über  $330\text{-}\Omega$ -Strombegrenzungswiderstände an den GPIO-Ports 23 (LED0) und 22 (LED1) angeschlossen. Die LEDs leuchten, wenn der Ausgang des Ports logisch 1 (high) ist, und sind dunkel, wenn der Ausgang des Ports logisch 0 (low) ist.

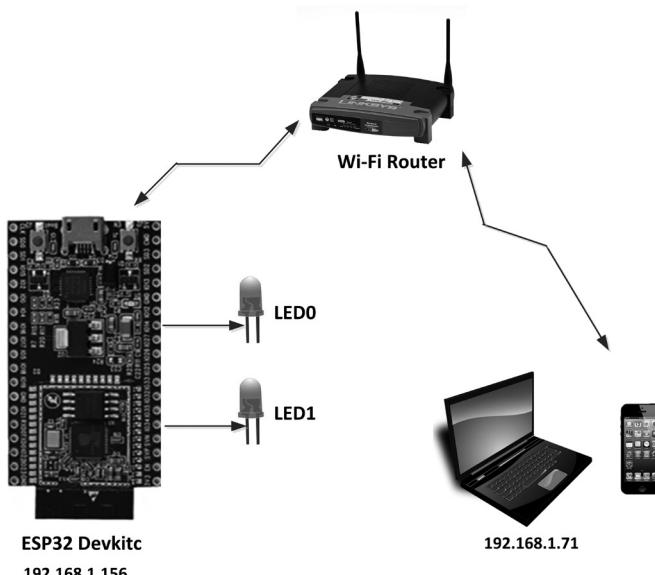


Bild 9.1 Blockschaltbild des Systems

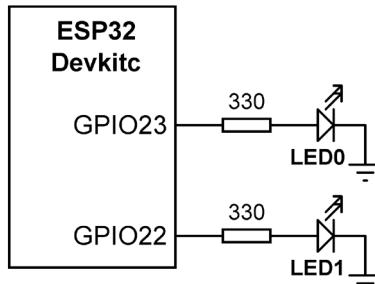


Bild 9.2 Schaltplan des Systems

### 9.3 HTTP-Webserver/Client

Ein Webserver ist ein Programm, das im HTTP (Hypertext Transfer Protocol) Benutzern auf ihre Anforderung hin Webseiten zur Verfügung stellt. Diese Anforderungen werden von HTTP-Clients weitergeleitet. Mit einem solchen HTTP-Server/Client-Paar können alle Geräte, die mit einem Webserver-Prozessor verbunden sind, über das Internet gesteuert werden.

Bild 9.3 zeigt die Struktur eines Webservers/Clients. In diesem Bild ist das ESP32-DevKitC der Webserver und der PC (oder Laptop, Tablet, Handy) der Webclient. Das zu steuernde Gerät, in diesem Fall die beiden LEDs, ist mit dem Webserver-Prozessor aka der Entwicklungsplatine ESP32-DevKitC verbunden. Das System funktioniert wie folgt:

- Der Webserver befindet sich im „Listen“-Modus und hört auf Anfragen des Webclients.
- Der Webclient stellt eine Anfrage an den Webserver, indem er ein HTTP-Request sendet.
- Als Antwort sendet der Webserver einen HTTP-Code an den Webclient, der vom Webbrowser auf dem Webclient aktiviert wird und auf dem Display des Webclients als Formular angezeigt wird.
- Der Benutzer sendet einen Befehl (er klickt beispielsweise auf eine Schaltfläche im Webclient-Formular, um eine LED einzuschalten). Der Webclient sendet einen Code an den Webserver, damit der die erforderliche Operation ausführen kann.

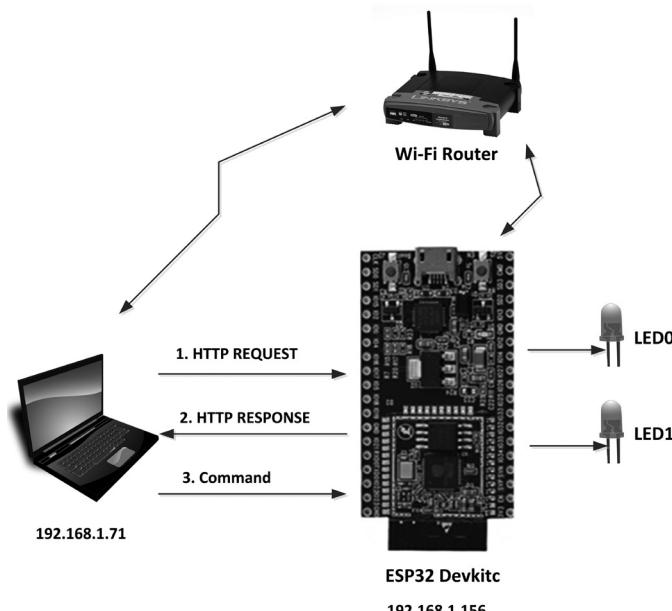


Bild 9.3 Webserver/Client-Struktur

In diesem Programm werden ein Laptop mit der IP-Adresse 192.168.1.71 als Webclient und das ESP32-DevKitC mit der IP-Adresse 192.168.1.156 als Webserver verwendet.

#### 9.4 Programmlisting ESP32-DevKitC

Bild 9.4 zeigt das vollständige Programm WEBSERVER. Zu Beginn des Programms werden die Wi-Fi-Bibliothek in das Programm aufgenommen, LED0 und LED1 den GPIO-Ports 23 beziehungsweise 22 zugewiesen und der lokale Wi-Fi-Name und das Passwort definiert.

```
/*
 *      WEB SERVER TO CONTROL 2 LEDs
 *      =====
 *
 * This is a simple web server program. 2 LEDs are connected to
 * GPIO ports 23 and 22 of the ESP32-DevKitC board. The project
 * controls these LEDs (turns ON or OFF) from a web server
 * application. For example, the LEDs can be controlled from any
 * device that is on the web, for example, a PC, tablet, mobile
 * phone etc. when activated, a form will appear on the device
 * with buttons and clicking these buttons will control the
 * LEDs. The LEDs are named LED0 and LED1, connected to GPIO
 * ports 23 and 22 respectively.
 *
 *
 * File:    WEBSERVER
 * Author: Dogan Ibrahim
```

```
* Date: August, 2017
*
*****/
```

```
#include "WiFi.h"
#define LED0 23
#define LED1 22

// Local Wi-Fi name and password
//
const char* ssid = "BTHomeSpot-XNH";
const char* password = "49345abaeb";
WiFiServer server(80);

// The HTML code. This code will display two buttons on user's
// device which can be clicked to control the LEDs
//
String html ="<!DOCTYPE html> \
<html> \
<body> \
<center><h1>ESP32-DevKitC LED ON/OFF</h1></center> \
<center><h2>Web Server Example with 2 LEDs</h2></center> \
<form> \
<button name=\"LED0\" button style=\"color:green\" value=\"ON\" type=\"submit\">LED0 ON</button> \
<button name=\"LED0\" button style=\"color:red\" value=\"OFF\" type=\"submit\">LED0 OFF</button><br><br> \
<button name=\"LED1\" button style=\"color:green\" value=\"ON\" type=\"submit\">LED1 ON</button> \
<button name=\"LED1\" button style=\"color:red\" value=\"OFF\" type=\"submit\">LED1 OFF</button> \
</form> \
</body> \
</html>";

// This function connects the ESP32-DevKitC to the local Wi-Fi
// network. The network name and passwords are as specified earlier
void Connect_WiFi()
{
    WiFi.begin(ssid, password);
    while(WiFi.status() != WL_CONNECTED)
    {
        delay(1000);
    }
}
```

```
}

// Configure LEDs as outputs, turn OFF the LEDs to start with,
// Connect to the local Wi-Fi, start the server
//
void setup()
{
    pinMode(LED0, OUTPUT);
    pinMode(LED1, OUTPUT);
    digitalWrite(LED0, LOW);
    digitalWrite(LED1, LOW);
    Connect_WiFi();
    server.begin();
}

// This is the main program loop. Inside the main program we
// check for the client connection and send the HTML file so
// that it is displayed on user's device. The user clicks the
// buttons to control the LEDs.
//
void loop()
{
    WiFiClient client=server.available();
    String request = client.readStringUntil('\r');
    client.flush();

    // We control the LEDs depending upon the key click. Variable
    // request holds the request and we search this string to see
    // which LED should be turned ON/OFF. The contents of request
    // is of the form (for example, to turn OFF LED0):
    // "/?LED0=OFF", or similarly, to turn LED 1: "/?LED1=ON"
    //

    if(request.indexOf("LED0=ON") != -1)digitalWrite(LED0, HIGH);
    if(request.indexOf("LED0=OFF") != -1)digitalWrite(LED0, LOW);
    if(request.indexOf("LED1=ON") != -1)digitalWrite(LED1, HIGH);
    if(request.indexOf("LED1=OFF") != -1)digitalWrite(LED1, LOW);

    //
    // The HTML page to be displayed on user's device
    //
    client.print(html);
}
```

Bild 9.4 Listing des Programms WEB SERVER

Das Programm definiert dann den HTML-Code, der an den anfragenden Webclient gesendet werden soll. Dieser Code ist in der Variablen **html** gespeichert. Der Webbrower (beispielsweise der Internet-Explorer) auf dem Webclient zeigt dann wie in Bild 9.5 die Meldung auf dem Client-Bildschirm an.

In der Mitte des Fensters werden eine Überschrift und auf der linken Seite Schaltflächen angezeigt. Zwei Tastenpaare zum Einschalten (grün) und Ausschalten (rot) der beiden LEDs werden angezeigt. Das Webserver-Programm sendet die HTML-Datei (Anweisung **client.print(html)**) an den Client und wartet auf dessen Befehl. Wenn eine Schaltfläche im Client-Fenster angeklickt wird, wird (zusätzlich zu einigen anderen Daten) ein Befehl an den Webserver gesendet. Hier interessiert uns aber nur der eigentliche Befehl, der je nach gedrückter Schaltfläche an den Webserver gesendet wird:

gedrückte Schaltfläche	zum Webserver gesendeter Befehl
LED0 ON	?LED0=ON
LED0 OFF	?LED0=OFF
LED1 ON	?LED1=ON
LED1 OFF	?LED1=OFF

Das Webserver-Programm auf dem ESP32-DevKitC stellt eine TCP-Verbindung her und liest den Befehl des WebClient als String (**String request = client.readStringUntil('\r')**). Das Programm durchsucht dann den empfangenen String, um festzustellen, ob darin einer der obigen Befehle enthalten ist. Wenn kein gültiger Sub-String gefunden wird, gibt der Webserver -1 zurück. Betrachten Sie beispielsweise folgende Anweisung:

```
if (request.indexOf("LED0=ON") != -1) digitalWrite(LED0, HIGH);
```

Hier wird der Sub-String **LED0=ON** im String **request** gesucht. Wenn der Sub-String **LED0=ON** in **request** nicht existiert, wird -1 zurückgegeben. Das Programm sucht für alle vier Befehle eine Übereinstimmung. Wenn ein Befehl gefunden wird, wird die passende LED entsprechend ein- oder ausgeschaltet:

```
if(request.indexOf("LED0=ON") != -1) digitalWrite(LED0,HIGH);
if(request.indexOf("LED0=OFF") != -1) digitalWrite(LED0,LOW);
if(request.indexOf("LED1=ON") != -1) digitalWrite(LED1,HIGH);
if(request.indexOf("LED1=OFF") != -1) digitalWrite(LED1,LOW);
```

## ESP32 Devkit LED ON/OFF

### Web Server Example with 2 LEDs



Bild 9.5 Anzeige des HTML-Codes im Browser des Client

In diesem Projekt lautete die IP-Adresse des Webservers 192.168.1.156. Die Prozedur, um das System zu testen, verläuft wie folgt:

- Das Programm kompilieren, auf das ESP32-DevKitC hochladen und ausführen.
- Öffnen Sie den Webbrower auf Ihrem PC und geben Sie die Adresse der Webseite 192.168.1.156 ein.
- Das in Bild 9.5 gezeigte Formular erscheint auf Ihrem PC-Bildschirm.
- Klicken Sie auf eine Schaltfläche, zum Beispiel LED0 ON, um LED0 einzuschalten.

Bild 9.6 und Bild 9.7 zeigen die Befehle, die an den Webserver gesendet werden, wenn auf die Taste LED0 ON beziehungsweise die Taste LED0 OFF geklickt wird.

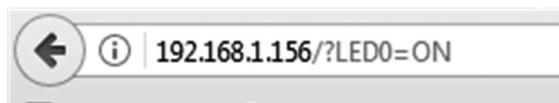


Bild 9.6 Klick auf die Taste LED0 ON



Bild 9.7 Klick auf die Taste LED0 OFF

## 9.5 Zusammenfassung

In diesem Kapitel haben wir eine interessante Webserver-Anwendung kennengelernt, bei der zwei LEDs aus dem Internet ferngesteuert werden. Statt der LEDs in diesem Projekt können auch beispielsweise Relais verwendet und damit alle elektrischen Geräte ferngesteuert werden.

Im nächsten Kapitel werden wir uns mit der Fernsteuerung von Geräten auseinandersetzen, indem wir UDP-Pakete von einem Mobiltelefon senden.

## KAPITEL 10 • FERNBEDIENUNG MIT DEM MOBILTELEFON

### 10.1 Überblick

Im letzten Kapitel haben wir eine interessante Wi-Fi-Anwendung kennengelernt, die zwei LEDs von einer Webserver-Anwendung aus steuert. Obwohl in diesem einfachen Beispiel LEDs verwendet wurden, gibt es keinen Grund, statt LEDs Relais anzuschließen, um irgend-eine Art elektrischer Geräte fernzusteuern.

In diesem Kapitel werden wir eine weitere interessante Anwendung entwickeln. Das System steuert zwei LEDs an einem ESP32-DevKitC-Entwicklungsboard über ein Mobiltelefon. Die Idee dabei ist, dass das DevKitC als UDP-Client konfiguriert wird. Das Mobiltelefon sendet UDP-Pakete, um diese LEDs zu steuern. Auch hier können diese LEDs durch Relais ersetzt werden, so dass das System zur Steuerung beliebiger elektrischer Geräte eingesetzt werden kann. Folgende Befehle sind gültig, alle anderen Befehle werden vom Programm ignoriert:

0=ON	LED0 einschalten
0=OFF	LED0 ausschalten
1=ON	LED1 einschalten
1=OFF	LED1 ausschalten

### 10.2 Das Blockschaltbild

Das Blockschaltbild des Projekts ist in Bild 10.1 dargestellt. Der Schaltplan entspricht dem vorherigen Projekt (Bild 9.2). LED0 und LED1 werden ebenfalls über 330- $\Omega$ -Strombegrenzungswiderstände an den GPIO-Ports 23 (LED0) und GPIO-Ports 22 (LED1) angeschlossen. Die LEDs werden eingeschaltet, wenn der Port-Ausgang logisch 1 (high) ist und ausgeschaltet, wenn der Port-Ausgang logisch 0 (low) ist.

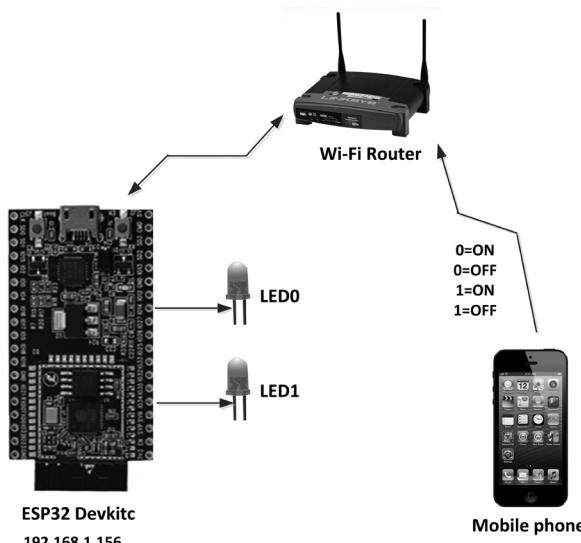


Bild 10.1 Blockschaltung des Systems

### 10.3 Smartphone-Anwendung

In diesem Projekt wird ein Android-Mobiltelefon verwendet, um die UDP-Befehle zu senden. Obwohl man eine App zum Senden der UDP-Befehle selber entwickeln könnte, gibt es im **Play Store** viele freie UDP-Apps, die man zum Senden und/oder Empfangen von UDP-Paketen verwenden kann. In unserem Projekt kommt die App **UDP RECEIVER and SENDER** von Wezzi Studios (Version 4.0) zum Einsatz. Diese App kann UDP-Pakete von jedem anderen Gerät, das das UDP-Protokoll ausführt, senden und empfangen. Der Bildschirm ist zweigeteilt: oben der Sender und unten der Empfänger (siehe Bild 10.2). Der Benutzer muss die Ziel-IP-Adresse, die Port-Nummer und die zu sendende Nachricht angeben. Wenn er dann auf die Schaltfläche SEND UDP MESSAGE klickt, wird das Paket an sein Ziel gesendet. Im Empfängerteil ist die lokale IP-Adresse schon eingetragen, so dass der Benutzer nur die Port-Nummer angeben muss, damit empfangene Pakete auf dem Bildschirm angezeigt werden.

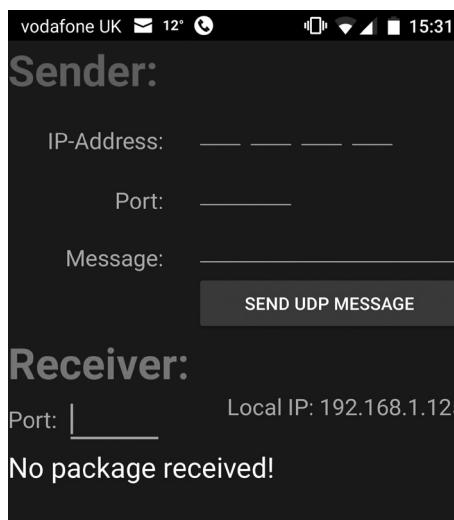


Bild 10.2 Die Android-App **UDP RECEIVER and SENDER**

In diesem Programm lautet die IP-Adresse des ESP32-DevKitC: 192.168.1.156.

### 10.4 ESP32-DevKitC Programmlisting

Bild 10.3 zeigt das vollständige Programm UDPControl. Zu Beginn werden Wi-Fi-Bibliothek-Header in das Programm eingebunden, LED0 und LED1 den GPIO-Ports 23 beziehungsweise 22 zugeordnet sowie die lokale Port-Nummer, der lokale Wi-Fi-Name und das Passwort definiert. Die Funktion **Connect\_WiFi** stellt eine Verbindung zum lokalen Wi-Fi-Netzwerk her.

Innerhalb der Setup-Routine werden die LED-Ports als Ausgänge konfiguriert und beide LEDs zunächst ausgeschaltet. Dann wird die Funktion **Connect\_WiFi** aufgerufen und UDP wird am lokalen Port gestartet.

Der Rest des Programms wird in einer Endlosschleife ausgeführt, in der das Programm auf

den Empfang eines Pakets (Befehls) vom UDP-Server, dem Mobiltelefon wartet. Trifft ein Paket ein, wird es im Zeichenarray **Packet** gespeichert. Das Programm überprüft den Befehl auf seine Gültigkeit und schaltet entsprechend die LEDs ein oder aus.

Der folgende Code wird zum Dekodieren des empfangenen Befehls verwendet:

```
if(Packet[1] == '=')
{
    if(Packet[0] == '0')
    {
        if(Packet[2] == '0' && Packet[3] == 'N')
        {
            digitalWrite(LED0,HIGH);
        }
        else if(Packet[2] == '0' && Packet[3] == 'F' && Packet[4] == 'F')
        {
            digitalWrite(LED0,LOW);
        }
    }
    else if(Packet[0] == '1')
    {
        if(Packet[2] == '0' && Packet[3] == 'N')
        {
            digitalWrite(LED1,HIGH);
        }
        else if(Packet[2] == '0' && Packet[3] == 'F' && Packet[4] == 'F')
        {
            digitalWrite(LED1,LOW);
        }
    }
}

/*****
 *          UDP BASED CONTROL FROM MOBILE PHONE
 *          =====
 *
 * This is an UDP based control program where 2 LEDs are connected
 * to GPIO ports 23 and 22 of the ESP32-DevKitC board, and named
 * as LED0 and LED1. Commands are sent from an Android type
 * mobile phone to turn the LEDs ON/OFF. The ESP32-DevKitC is
 * configured as an UDP client in this application. The format of
 * the commands are as follows:
 *
 * 0=ON   turn ON LED0
 * 0=OFF  turn OFF LED0
 * 1=ON   turn ON LED1
 * 1=OFF  turn OFF LED1
 *
```

```
*  
* File: UDPControl  
* Author: Dogan Ibrahim  
* Date: August, 2017  
*  
*****  
#include "WiFi.h"  
#include <WiFiUdp.h>  
//  
// LED assignments  
//  
#define LED0 23  
#define LED1 22  
WiFiUDP udp;  
//  
// Use local port 5000  
//  
const int Port = 5000;  
char Packet[80];  
  
//  
// Local Wi-Fi name and password  
//  
const char* ssid = "BTHomeSpot-XNH";  
const char* password = "49345abaeb";  
  
//  
// This function connects the ESP32-DevKitC to the local Wi-Fi  
// network. The network name and passwords are as specified earlier  
void Connect_WiFi()  
{  
    WiFi.begin(ssid, password);  
    while(WiFi.status() != WL_CONNECTED)  
    {  
        delay(1000);  
    }  
}  
  
//  
// Configure LEDs as outputs, turn OFF the LEDs to start with,  
// Connect to the local Wi-Fi. Also, UDP is started in local port  
//  
void setup()  
{  
    pinMode(LED0, OUTPUT);  
}
```

```
pinMode(LED1, OUTPUT);
digitalWrite(LED0, LOW);
digitalWrite(LED1, LOW);
Connect_WiFi();
udp.begin(Port);
}

// This is the main program loop. Inside the main program we read
// UDP packets and then control the LEDs as requested. The format
// of the control commands are:
//
// 0=ON or 0=OFF for LED0
// 1=ON or 1=OFF for LED1
//
// Any other commands are simply ignored by the program
//
void loop()
{
    int PacketSize = udp.parsePacket();
    if(PacketSize)
    {
        udp.read(Packet, PacketSize);

        if(Packet[1]=='=')
        {
            if(Packet[0]=='0')
            {
                if(Packet[2]=='0' && Packet[3]=='N')
                {
                    digitalWrite(LED0, HIGH);
                }
                else if(Packet[2]=='0' && Packet[3]=='F' && Packet[4]=='F')
                {
                    digitalWrite(LED0, LOW);
                }
            }
            else if(Packet[0]=='1')
            {
                if(Packet[2]=='0' && Packet[3]=='N')
                {
                    digitalWrite(LED1, HIGH);
                }
                else if(Packet[2]=='0' && Packet[3]=='F' && Packet[4]=='F')
                {
                    digitalWrite(LED1, LOW);
                }
            }
        }
    }
}
```

}

### *Bild 10.3 Programmlisting*

Als Beispiel zeigt Bild 10.4 den Befehl, der von der Mobiltelefonanwendung gesendet wurde, um LED1 einzuschalten (Befehl: 1=ON).

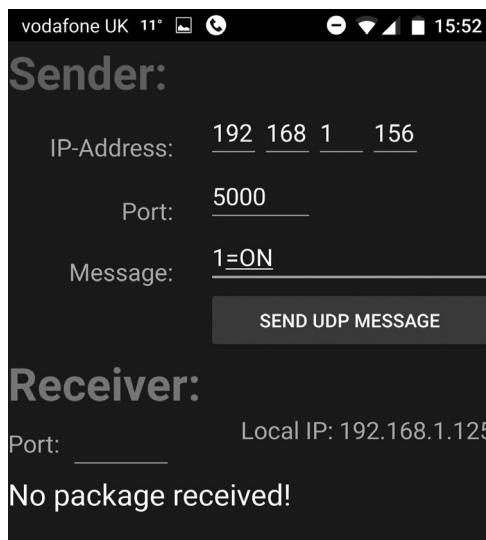


Bild 10.4 Befehl zum Einschalten der LED1

## 10.5 Zusammenfassung

In diesem Kapitel haben wir eine Webserver-Anwendung gesehen, bei der zwei LEDs per UDP-Paket von einem Mobiltelefon aus ferngesteuert werden. Obwohl in diesem Projekt LEDs verwendet werden, kann man auch Relais an das ESP32-DevKitC anschließen, über die alle elektrischen Geräte ferngesteuert werden können.

Im nächsten Kapitel erfahren Sie, wie Sie wiederum mit UDP-Paketen, Temperatur- und Luftfeuchtigkeitswerte auf ein Mobiltelefon übertragen können.

## KAPITEL 11 • TEMPERATUR UND LUFTFEUCHTE AN EIN HANDY SENDEN

### 11.1 Übersicht

Im vorherigen Kapitel haben wir eine interessante Wi-Fi-Anwendung entwickelt, bei der zwei LEDs von einem Android-Handy gesteuert werden.

In diesem Kapitel geht es um eine weitere interessante Anwendung. Wir ermitteln die Umgebungstemperatur und die Luftfeuchte und senden diese Daten dann an ein Mobiltelefon, wenn eine Anfrage gestellt wird. In diesem Programm werden Daten mit UDP-Paketen ausgetauscht.

Die Daten werden in folgendem Format gesendet, wobei T die Temperatur- und H die Luftfeuchtigkeitswerte sind:

$$T = nn \quad H = nn$$

Eine Datenanfrage des Mobiltelefons liegt dann vor, wenn das Zeichen „S“ zum ESP32-DevKitC gesendet wird.

Dieses Projekt verwendet eine bidirektionale UDP-Kommunikation zum Empfangen und Senden von Daten.

### 11.2 Die Blockschaltung

Das Blockschaltbild des Projekts ist in Bild 11.1 zu sehen. Bild 5.12 zeigt den Schaltplan des Projekts, bei dem der DHT11-Ausgang (Pin S) an den GPIO-Port 23 des ESP32-DevKitC angeschlossen ist.

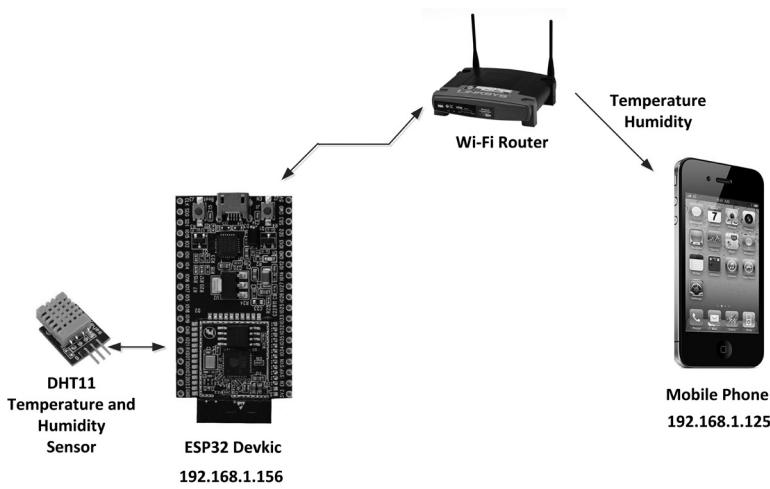


Bild 11.1 Blockschaltbild des Systems

### 11.3 Handy-Anwendung

In diesem Projekt wird ein Android-Mobiltelefon verwendet, um die UDP-Pakete vom ESP32-DevKitC zu empfangen. Wie in Kapitel 9 wird auch hier die Android-App **UDP RECEIVER and SENDER** von Wezzi Studios (Version 4.0) verwendet.

Die IP-Adressen des ESP32-DevKitC und des Mobiltelefons lauten 192.168.1.156 und 192.168.1.125. Beachten Sie, dass die IP-Adresse des Mobiltelefons auf dem Display erscheint, wenn die UDP-Anwendungen gestartet werden.

### 11.4 ESP32-DevKitC Programmlisting

Bild 11.2 zeigt das vollständige Programm UDPMonitor. Zu Beginn des Programms werden die Wi-Fi- und DHT11-Bibliotheks-Header aufgenommen sowie die lokale Port-Nummer angegeben, der lokale Wi-Fi-Name und das Passwort definiert. Die Funktion **Connect\_WiFi** stellt eine Verbindung zum lokalen Wi-Fi-Netzwerk her. Die Funktion **TempHum** liest die Umgebungstemperatur und die Luftfeuchtigkeit und speichert sie in den Variablen **T** und **H**. In der Setup-Routine wird die Funktion **Connect\_WiFi** zur Verbindung mit dem lokalem Wi-Fi aufgerufen, dann werden UDP und DHT11 gestartet.

Der Rest des Programms wird in einer Endlosschleife ausgeführt. Innerhalb dieser Schleife wartet das Programm auf den Empfang eines Pakets (Befehl zum Zurückgeben der Temperatur- und Luftfeuchtigkeitswerte) vom UDP-Server, dem Mobiltelefon. Dazu muss das Mobiltelefon das Zeichen S senden. Wenn dieses Zeichen eintrifft, werden Temperatur und die Feuchtigkeit gelesen und mit dem Zeichenarray **Packet wird** ein Datenpaket mit den Temperatur- und Luftfeuchtigkeitswerten im erforderlichen Format zusammengestellt. Das Paket wird dann über Port 5000 an das Mobiltelefon gesendet.

Das Zeichenarray **Packet** wird wie folgt gebildet:

```
Packet[0] = 'T'; // Anzeige T
Packet[1] = '='; // Anzeige =
int msd = T / 10; // Hohe Ziffer extrahieren
int lsd = T - msd * 10; // Niedrige Ziffer extrahieren
Packet[2] = msd + '0';// msd in ASCII-Zeichen konvertieren
Packet[3] = lsd + '0';// lsd in ASCII-Zeichen konvertieren
Packet[4] = ''; // Leerzeichen
Packet[5] = ''; // Leerzeichen
Packet[6] = 'H'; // Anzeige H
Packet[7] = '='; // Anzeige =
msd = H/10; // Hohe Ziffer extrahieren
lsd = H - msd*10; // Niedrige Ziffer extrahieren
Packet[8] = msd + '0';// msd in ASCII-Zeichen konvertieren
Packet[9] = lsd + '0';// lsd in ASCII-Zeichen konvertieren
Packet[10] = 0; // NULL-Terminator
```

```
*****  
*          SEND TEMPERATURE AND HUMIDITY TO A MOBILE PHONE
```

```
* =====
*
* This is an UDP based monitoring program where a DHT11 type
* temperature and humidity sensor chip is connected to GPIO
* port 23 of the ESP32-DevKitC. The program reads the ambient
* temperature and humidity and sends the readings to an Android
* mobile phone using UDP packets. The data on the mobile phone
* is displayed using an UDP apps, available on the Play Store.
* The data is sent to the mobile phone whenever it is requested.
*
* A request for data is made to the ESP32-DevKitC when character
* S is sent to the ESp32 Devkit over the UDP link. The ESP32
* Devkitc sends the temperature and humidity data to the device
* that made the request. Port 5000 is used for the communication
*
* The IP addresses of the ESP32-DevKitC and mobile phone are:
* 192.168.1.56 and 192.168.1.125 respectively.
*
* File: UDPMonitor
* Author: Dogan Ibrahim
* Date: August, 2017
*
*****
#include "WiFi.h"
#include <WiFiUdp.h>
//
// DHT11 sensor library includes
//
#include <Adafruit_Sensor.h>
#include "DHT.h"
#define DHT11_PIN 23
#define DHTTYPE DHT11
DHT dht(DHT11_PIN, DHTTYPE);

WiFiUDP udp;

//
// Use local port 5000
//
const int Port = 5000;
char Packet[20];

//
// Local Wi-Fi name and password
//
const char* ssid = "BTHomeSpot-XNH";
```

```
const char* password = "49345abaeb";
int T, H;

// 
// Read the ambient temperature and humidity and store in variables
// T and H respectively
//
void TempHum()
{
    H = dht.readHumidity();                                // Read humidity
    T = dht.readTemperature();                             // Read temperature
}

// 
// This function connects the ESP32-DevKitC to the local Wi-Fi
// network. The network name and passwords are as specified earlier
void Connect_WiFi()
{
    WiFi.begin(ssid, password);
    while(WiFi.status() != WL_CONNECTED)
    {
        delay(1000);
    }
}

// 
// Connect to the local Wi-Fi. Also, UDP is started in local
// port and DHT11 is started
//
void setup()
{
    Connect_WiFi();
    udp.begin(Port);
    dht.begin();
}

// 
// This is the main program loop. Inside the main program we read
// the temperature and humidity and send them as a packet to the
// mobile phone when a request is made by the mobile phone.
// A request is made when character S is sent by the mobile
// phone.
//
// The packet format is:
//
```

```
//  T=nn H=nn
//
void loop()
{
    int PacketSize = udp.parsePacket();
    if(PacketSize)
    {
        udp.read(Packet, PacketSize);
        if(Packet[0] == 'S')
        {
            TempHum();
            Packet[0] = 'T';
            Packet[1] = '=';
            int msd = T/10;
            int lsd = T - msd*10;
            Packet[2] = msd + '0';
            Packet[3] = lsd + '0';
            Packet[4] = ' ';
            Packet[5] = ' ';
            Packet[6] = 'H';
            Packet[7] = '=';
            msd = H/10;
            lsd = H - msd*10;
            Packet[8] = msd + '0';
            Packet[9] = lsd + '0';
            Packet[10] = 0;
            udp.beginPacket(udp.remoteIP(), Port);
            udp.print(Packet);
            udp.endPacket();
        }
    }
}
```

*Bild 11.2 Programmlisting*

Bild 11.3 zeigt beispielhaft die Anfrage **S** der Mobiltelefonanwendung. Die zurückgegebenen Temperatur- und Feuchtigkeitsdaten werden im unteren Teil des Displays angezeigt.

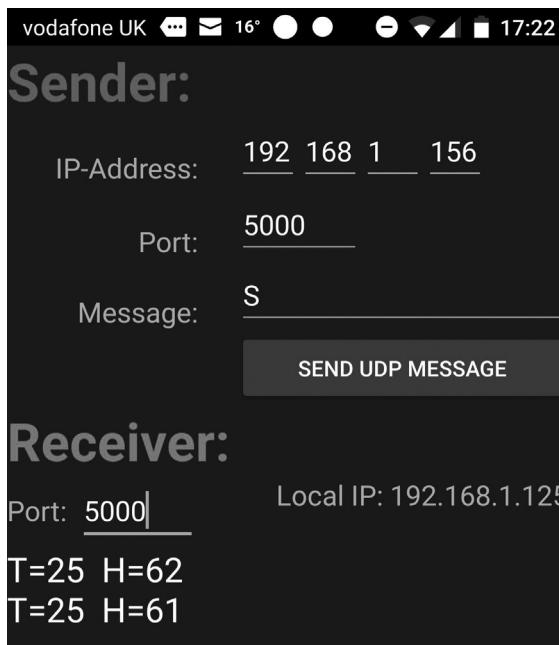


Bild 11.3 Auf Anfrage S werden T und H zurückgegeben.

### 11.5 Zusammenfassung

In diesem Kapitel haben wir eine Webserver-Anwendung kennengelernt, bei der die Umgebungstemperatur- und Luftfeuchtigkeitswerte vom ESP32-DevKitC über UDP-Pakete an ein Android-Mobiltelefon gesendet werden. Dieses Projekt zeigt, wie eine Zwei-Wege-Kommunikation mit UDP-Paketen aufgebaut werden kann.

Im nächsten Kapitel werden wir die MicroPython-Software auf unserem ESP32-DevKitC installieren und sehen, wie MicroPython-Programme entwickelt und auf das DevKitC hochgeladen werden können.

## KAPITEL 12 • MICROPYTHON AUF DEM ESP32-DEVKITC

### 12.1 Übersicht

Bis jetzt haben wir in der Arduino-IDE Programme für unser ESP32-DevKitC entwickelt. Der ESP32-Prozessor kann programmiert werden mit:

- **ESP-IDF:** Dies ist die native Entwicklungsumgebung, die für die ESP32-Prozessoren von Espressif Systems entwickelt wurde. Diese Entwicklungsumgebung basiert auf FreeRTOS und ermöglicht dem Benutzer den Zugriff auf alle Module des ESP32-Prozessors einschließlich Bluetooth. Obwohl die ESP-IDF sehr leistungsfähig ist, ist die Entwicklung von Programmen in dieser IDE ziemlich schwierig.
- **Mongoose OS:** Dies ist ein Open-Source-Betriebssystem, das ESP32, ESP8266, STM32 und andere Prozessoren unterstützt, die für die Internet-of-Things-Anwendungen entwickelt wurden.
- **Arduino-IDE:** Diese Entwicklungsumgebung haben wir bisher genutzt, um Programme für den ESP32-Prozessor zu schreiben.
- **MicroPython:** Dies ist derzeit eine der beliebtesten Programmiersprachen, die weltweit in den meisten Universitäten in Einführungskursen verwendet wird.

In diesem Kapitel werden wir MicroPython auf unserem ESP32-DevKitC installieren und einige einfache Projekte mit MicroPython durchführen.

### 12.2 Installation von MicroPython auf ESP32-DevKitC

Vor der Installation von MicroPython auf dem ESP32-DevKitC muss man **Python 2.7** auf dem PC installieren, wie es schon in Kapitel 3.2 beschrieben wurde.

Wir benötigen auch eine Software namens **esptool** auf unserem PC, um die MicroPython-Binaries auf den ESP32 zu flashen. Die Schritte zur Installation von **esptool** auf den PC sind folgende:

- Starten Sie die Kommandozeile (cmd.exe) auf Ihrem PC und geben Sie CD\ ein, um ins Stammverzeichnis zu wechseln (siehe Bild 12.1).

```
C:\ Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\Dogan>cd\

C:\>
```

Bild 12.1 Ins Stammverzeichnis wechseln

- Geben Sie folgenden Befehl ein, um die neueste Version des **esptools** auf dem PC zu installieren (siehe Bild 12.2). Es muss mindestens die Version 2.0 sein, da

Version 1.x von esptool den ESP32-Prozessor nicht unterstützt. Nach einiger Zeit sollte die Meldung **Successful installed esptool** erscheinen.

C:\>pip install esptool.  
oder  
C:\>python -m install esptool

```
C:\>python -m pip install esptool
Collecting esptool
  Downloading esptool-2.0.1.tar.gz (67kB)
    100% :#####: 71kB 217kB/s
Requirement already satisfied: pyserial>=2.5 in c:\python27\lib
ron esptool>
Collecting pyaes (from esptool)
  Downloading pyaes-1.6.0.tar.gz
Collecting ecdsa (from esptool)
  Downloading ecdsa-0.13-py2.py3-none-any.whl (86kB)
    100% :#####: 92kB 708kB/s
Installing collected packages: pyaes, ecdsa, esptool
  Running setup.py install for pyaes ... done
  Running setup.py install for esptool ... done
Successfully installed ecdsa-0.13 esptool-2.0.1 pyaes-1.6.0
C:\>
```

Bild 12.2 Installation von esptool

- Es wird empfohlen, den gesamten Flash-Speicher des ESP32 vor dem Hochladen der neuen Firmware zu löschen. Schließen Sie dazu das ESP32-DevKitC mit dem Mini-USB-Kabel an den PC an. Ermitteln Sie die Nummer des seriellen Ports des DevKitC im Geräte-Manager und geben Sie den folgenden Befehl ein. Hier wird angenommen, dass der DevKitC am seriellen Port COM47 angeschlossen ist. Bild 12.3 zeigt den Befehl zum Löschen des Flash-Speichers. Die Löschvorgänge sollten etwa zehn Sekunden dauern.

C: \>esptool.py --port COM47 erase\_flash  
oder  
C: \>python c:\ python27\lib\site-packages\esptool.py --port COM47  
erase\_flash

```
C:\>python c:\python27\lib\site-packages\esptool.py --port COM47 erase_flash
esptool.py v2.0.1
Connecting.....-----
Detecting chip type... ESP32
Chip is ESP32DWDQ6 (revision 0)
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 2.7s
Hard resetting...
C:\>
```

Bild 12.3 Löschen des gesamten Flash-Speichers des ESP32-DevKitC

- Die neueste MicroPython-Binärdatei, die unter dem nachfolgenden Link zu finden ist, wird in einen Ordner (hier zum Beispiel C:\ESP32) kopiert. Als diese Zeilen geschrieben wurde, hieß die aktuelle Version: esp32-20170803-v1.9.1-394-g79feb956.bin. (siehe Bild 12.4).

<https://micropython.org/download/#esp32>



Bild 12.4 Kopieren der MicroPython-Binärdatei in einen Ordner

- Wir sind jetzt bereit, die MicroPython-Firmware auf unser ESP32-DevKitC hochzuladen. Schließen Sie, wenn noch nicht geschehen, das DevKitC an den USB-Port an und geben Sie folgenden Befehl ein (siehe Bild 12.5). Die Größe des Flash-Speichers wird automatisch erkannt und die MicroPython-Firmware wird auf den ESP32-DevKitC hochgeladen:

```
C:\>esptool.py --port COM47 --baud 460800 write_flash -fm dio 0x1000
C:\ESP32\esp32-20170803-v1.9.1-394-g79feb956.bin
```

oder

```
C:\>python c:\python27\lib\site-packages\esptool.py --port COM47 -
baud 460800 write_flash -fm dio 0x1000 C:\ESP32\esp32-20170803-
v1.9.1-394-g79feb956.bin
```

```
C:\>python c:\python27\lib\site-packages\esptool.py --port COM47 --baud 460800 w
rite_flash -fm dio 0x1000 c:\esp32\esp32-20170803-v1.9.1-394-g79feb956.bin
esptool.py v2.0.1
Connecting.....
Detecting chip type... ESP32
Chip is ESP32D0WDQ6 (revision 0)
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 1085936 bytes to 603738...
Wrote 1085936 bytes <603738 compressed> at 0x00001000 in 16.8 seconds (effective
516.1 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting...
```

Bild 12.5 Hochladen der MicroPython-Firmware

Beachten Sie, dass in Bild 12.5 angenommen wird, dass die Größe des Flash-Speichers 4 MB beträgt, weshalb der Parameter **dio** angegeben wird. Dieser Parameter sollte für 512-Kbyte-Module **qio** lauten. Die Größe des Flash-Speichers kann mit folgendem Befehl ermittelt werden:

```
C:\>python c:\python27\lib\site-packages\esptool.py --port COM47 flash_id
```

Die Größe des Flash-Speichers in diesem Beispiel ist in Bild 12.6 dargestellt.

```
C:\>python c:\python27\lib\site-packages\esptool.py --port COM47 flash_id
esptool.py v2.0.1
Connecting.....
Detecting chip type... ESP32
Chip is ESP32D0WDQ6 (revision 0)
Uploading stub...
Running stub...
Stub running...
Manufacturer: c8
Device: 4016
Detected flash size: 4MB
Hard resetting...
```

Bild 12.6 Anzeige der Größe des Flash-Speichers

- Drücken Sie nach dem Hochladen die Taste RESET am ESP32-DevKitC.

### 12.3 Testen der MicroPython-Installation

Um die MicroPython-Installation zu testen, wird das Programm **PuTTY** gestartet. Stellen Sie den Verbindungstyp auf **Serial** ein, geben in **Serial line** die serielle Port-Nummer (hier COM47) und die Geschwindigkeit **Speed** 115200 ein. Drücken Sie die RESET-Taste an Ihrem ESP32-DevKitC. Sie sollten dann den MicroPython-Interpreter-Bildschirm wie in Bild 12.7 sehen. Geben Sie den folgenden Befehl ein, um Ihre Installation zu testen:

```
print ("Hello from MicroPython")
```

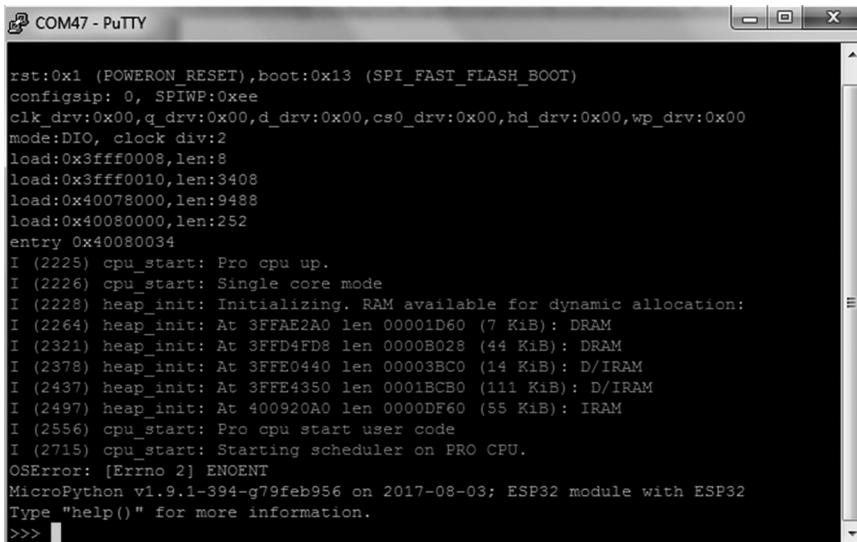


Bild 12.7 Bildschirm des MicroPython-Interpreters

Es ist nicht die Absicht dieses Buches, die MicroPython-Programmiersprache zu lehren. Es gibt viele Bücher (zum Beispiel ESP8266 AND MICROPYTHON von Elektor), Tutorials, Projekte und Anwendungshinweise zu diesem Thema. In den nächsten Abschnitten werden einige Projekte durchgeführt, die die Anwendung von MicroPython auf dem ESP32-DevKitC demonstrieren.

## 12.4 Blinkende LED

Dieses Projekt entspricht dem Projekt 1 in Kapitel 4.2, bei dem eine LED über einen  $330\text{-}\Omega$ -Strombegrenzungswiderstand am GPIO-Port 23 angeschlossen ist und jede Sekunde blinkt.

Das Programm wird interaktiv mit dem PuTTY-Terminal-Emulator geschrieben, der eine Verbindung zum ESP32-DevKitC herstellt. Bild 12.8 zeigt das Listing des Programms.

```
>>>
>>> from machine import Pin
>>> import time
>>> p23 = Pin(23, Pin.OUT)
>>> while True:
...     p23.value(0)
...     time.sleep(1)
...     p23.value(1)
...     time.sleep(1)
...
...
```

Bild 12.8 MicroPython-Programmlisting

## 12.5 LED mit Drucktaster

In diesem Projekt wird wie im vorherigen Projekt eine LED an den GPIO-Port 23 angeschlossen. Zusätzlich ist ein Taster am GPIO-Port 22 angeschlossen. Die LED wird bei jedem Drücken der Taste eingeschaltet.

Bild 12.9 zeigt den Schaltplan des Projekts, Bild 12.10 das Programmlisting. GPIO-Port 22 ist als Eingang konfiguriert und wird controller-intern auf HIGH gezogen, so dass der GPIO-Pin normalerweise logisch 1 und die LED ausgeschaltet ist. Durch Drücken der Taste ändert sich der Status des Tastereingangs auf logisch 0, so dass die LED eingeschaltet wird.

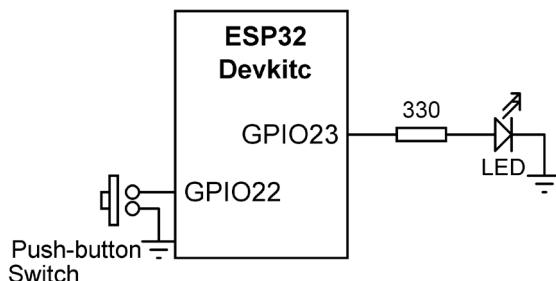


Bild 12.9 Schaltplan des Projekts

```

>>>
>>> from machine import Pin
>>> p23 = Pin(23, Pin.OUT)
>>> p22 = Pin(22, Pin.IN, Pin.PULL_UP)
>>> p23.value(0)
>>> while True:
...     if p22.value() == 0:
...         p23.value(1)
...     else:
...         p23.value(0)
...

```

Bild 12.10 MicroPython-Programmlisting

## 12.6 Temperatur und Luftfeuchtigkeit

In diesem Projekt werden Temperatur und Luftfeuchtigkeit vom Sensorchip DHT11 gemessen und alle fünf Sekunden auf dem PuTTY-Bildschirm angezeigt. Das Blockschaltbild und der Schaltplan des Projekts entsprechen Bild 5.10 und Bild 5.13.

Bild 12.11 zeigt das MicroPython-Programmlisting, das interaktiv über PuTTY eingegeben wird. Die Bibliotheken dht, Pin und time werden am Anfang des Programms importiert.

```

>>> import dht
>>> import time
>>> from machine import Pin
>>> d = dht.DHT11(Pin(23))
>>> while True:
...     d.measure()
...     t = d.temperature()
...     h = d.humidity()
...     print("Temperature=%fC, Humidity=%f" %(t, h))
...     time.sleep(5)
...
...
...
...
Temperature=23.000000C, Humidity=60.999999
Temperature=23.000000C, Humidity=60.999999
Temperature=23.000000C, Humidity=60.999999

```

Bild 12.11 Programmlisting

## 12.7 Verbindung zu einem WLAN herstellen

Bevor man mit anderen Knoten in einem Netzwerk kommunizieren kann, muss man sich selbst als Knoten im Netzwerk anmelden und eine IP-Adresse erhalten. Die MicroPython-Bibliothek „network“ stellt netzwerkbezogene Funktionen zur Verfügung, verbindet zum Beispiel mit einem Netzwerk, sucht nach der IP-Adresse und so weiter. Diese Bibliothek muss in unsere MicroPython-Programme importiert werden, bevor man auf die Netzwerkfunktionen zugreifen kann.

Die Netzwerkbibliothek hat eine Klasse mit dem Namen WLAN. Einige der häufig verwendeten Methoden dieser Klasse sind:

- **network.WLAN(network.STA\_IF)**: erstellt ein WLAN-Netzwerkobjekt zur Verbindung mit einem WLAN
- **network.WLAN(network.AP\_IF)**: erstellt ein WLAN-Netzwerkobjekt zur Verbindung mit einem AP, andere Wi-Fi-Clients können sich verbinden
- **active(True)**: aktiviert die Netzwerkschnittstelle

- **active(False)**: deaktiviert die Netzwerkschnittstelle
- **connect(„ssid“, „password“)**: stellt eine Verbindung zum angegebenen WLAN her
- **disconnect()**: Verbindung zum aktuell verbundenen Netzwerk trennen
- **isconnected()**: findet heraus, ob man mit einem WLAN verbunden ist
- **ifconfig()**: gibt die IP-Adresse der zugewiesenen Verbindung zurück

Das Beispiel in Bild 12.12 zeigt, wie die Methode connect durch Angeben der SSID (in diesem Beispiel BTHomeSpot-XNH) und des Passworts (hier 49345abaeb) die Verbindung zu einem Wi-Fi-Netzwerk herstellt, woraufhin die zugewiesene IP-Adresse 192.168.1.156 dargestellt wird.

```
>>> import network
>>> net=network.WLAN(network.STA_IF)
>>> net.isconnected()
False
>>> net.active(True)
I (1524367) wifi: mode : sta (30:ae:a4:05:5b:e0)
I (1524367) wifi: STA_START
True
>>> net.connect("BTHomeSpot-XNH", "49345abaeb")
>>> I (1561557) wifi: n:6 0, o:1 0, ap:255 255, sta:6 0, prof:1
I (1562117) wifi: state: init -> auth (b0)
I (1562127) wifi: state: auth -> assoc (0)
I (1562137) wifi: state: assoc -> run (10)
I (1562157) wifi: connected with BTHomeSpot-XNH, channel 6
I (1562157) wifi: event 4
I (1564817) event: ip: 192.168.1.156, mask: 255.255.255.0, gw: 192.168.1.254
I (1564817) wifi: GOT_IP
I (1572137) wifi: pm start, type:0

>>> net.ifconfig()
('192.168.1.156', '255.255.255.0', '192.168.1.254', '192.168.1.254')
>>> █
```

Bild 12.12 Verbindung zu einem Wi-Fi-Netzwerk herstellen

Die folgende MicroPython-Funktion kann von einem Programm aufgerufen werden, um eine Verbindung zu einem Wi-Fi-Netzwerk herzustellen. Die zugewiesene IP-Adresse wird am Ende der Funktion angezeigt:

```
def Connect_WiFi():
    import network
    net = network.WLAN(network.STA_IF)
    if not net.isconnected():
        net.active(True)
    ....net.connect("ssid", "password")
    print (net.ifconfig())
```

## 12.8 MicroPython-UDP-Programme

Wie in Kapitel 7 beschrieben wurde, ist UDP ein verbindungsloses Protokoll und wird üblicherweise zur Kommunikation über ein Netzwerk verwendet, indem Datenpakete gesendet und empfangen werden.

Das Beispielprogramm in diesem Abschnitt zeigt – wie das Programm in Kapitel 11 –, wie Temperatur- und Feuchtigkeitswerte mit UDP-Paketen an ein Android-Mobiltelefon gesen-

det werden. Das Programm geht dazu in folgenden Schritten vor:

- Das Mobiltelefon fordert Daten an, indem es das Zeichen S an das ESP32-DevKitC sendet.
- DevKitC liest die Werte von Temperatur und Luftfeuchtigkeit, die vom DHT11 gemessen wurden und sendet sie ans Handy.
- Dieser Prozess wird solange wiederholt, bis er angehalten wird.

Bild 12.13 zeigt das Listing dieses Programms, das interaktiv über den PuTTY-Terminal-Emulator eingegeben wurde. Am Anfang des Programms werden die Bibliotheken socket, dht und Pin importiert. Die Funktion **Connect\_WiFi** stellt eine Verbindung zum lokalen WLAN her, die Funktion **tempHum** liest Umgebungstemperatur und Luftfeuchtigkeit und gibt die Werte in den Variablen **T** und **H** zurück. Das Programm erstellt dann einen UDP-Socket und bindet diesen. Der Rest des Programms wird innerhalb der **while**-Schleife ausgeführt. Hier wartet das Programm auf den Empfang der Anfrage S des Mobiltelefons. Bei einem Empfang dieses Requests wird die Funktion **tempHum** aufgerufen und beide Messwerte wie in Bild 11.3 an das Mobiltelefon gesendet.

```
>>> import socket
I (95754) modsocket: Initializing
>>> import dht
>>> from machine import Pin
>>> def Connect_WiFi():
...     import network
...     net=network.WLAN(network.STA_IF)
...     if not net.isconnected():
...         net.active(True)
...         net.connect("BTHomeSpot-XNH","49345abaeb")
...
...
>>> def tempHum():
...     d=dht.DHT11(Pin(23))
...     d.measure()
...     t=d.temperature()
...     h=d.humidity()
...     return t,h
...
...
>>> Connect_WiFi()
>>> port=5000
>>> UDP="",
port
>>> sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
>>> sock.bind(UDP)
>>> while True:
...     data,addr=sock.recvfrom(80)
...     if(data.decode() == 'S'):
...         T,H=tempHum()
...         dat=str(T)+" "+str(H)
...         tup=(addr[0],port)
...         sock.sendto(dat,tup)
...
...
...
5
5
```

*Bild 12.13 Das Programm wird interaktiv eingegeben*

Bild 12.14 zeigt das Programmlisting als kommentierte Datei

```
#-----
#  
# SEND TEMPERATURE AND HUMIDITY TO A MOBILE PHONE  
# -----  
#  
# In this program a DHT11 sensor chip is connected to GPIO port 23 of the  
# ESP32-DevKitC. The MicroPython program reads the ambient temperature and  
# humidity from the sensor chip and sends to the mobile phone when  
# requested.  
# A request is made by the mobile phone it sends character S. The data is  
# displayed on the mobile phone in the following format:  
#  
# T=nn H=nn  
#  
# Author: Dogan Ibrahim  
# Date : August 2017  
#-----  
#  
# Import libraries used in the program  
#  
import socket  
import dht  
from machine import Pin  
#  
# Function to connect to Wi-Fi  
#  
def Connect_WiFi():  
    import network  
    net = network.WLAN(network.STA_IF)  
    if not net.isconnected():  
        net.active(True)  
        net.connect("BTHomeSpot-XNH", "49345abaeb")  
#  
# Function returns the temperature and humidity  
#  
def tempHum():  
    d = dht.DHT11(Pin(23))  
    d.measure()  
    t = d.temperature()  
    h = d.humidity()  
    return t,h  
#  
# Start of main program. Define port number, create a socket, and bind to it  
#  
Connect_WiFi()  
port = 5000
```

```

UDP = ("", port)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(UDP)
#
# This part of the program runs in an endless loop, sending the temperature
# and humidity readings to the mobile phone when it receives a request (S)
#
while True:
    data, addr = sock.recvfrom(80)
    if(data.decode() == 'S'):
        T,H = TempHum()
        dat = str(T) + „ „ + str(H)
        tup = (addr[0], port)
        sock.sendto(dat, tup)

```

*Bild 12.14 Programmlisting mit Kommentaren*

## 12.9 Speichern von Temperatur und Luftfeuchtigkeit in der Cloud

In diesem Projekt werden die Luftfeuchtigkeit und die Temperatur gelesen und die Werte in der Cloud gespeichert. Das Blockschaltbild und der Schaltplan des Projekts entsprechen Bild 8.1 beziehungsweise Bild 5.12.

Die Werte werden wie in Kapitel 8 in der Thingspeak-Cloud gespeichert. Sie sollten dafür ein Thingspeak-Konto erstellen und, wie ebenfalls in Kapitel 8 beschrieben, einen API-Schlüssel und eine Kanalnummer besitzen. In diesem Abschnitt wird nur das MicroPython-Programm behandelt und API-Key wie Kanalnummer aus Kapitel 8 verwendet.

Bild 12.15 zeigt das MicroPython-Programm, das interaktiv über den PuTTY-Terminal-Emulator eingegeben wurde. Das Programm sorgt grundsätzlich für einen HTTP-GET-Request zum Hochladen der Temperatur- und Feuchtigkeitswerte.

Das Programmlisting mit Kommentaren ist in Bild 12.16 zu sehen. Am Anfang des Programms werden die Netzwerkbibliotheken, die Time-Bibliothek, die DHT11-Bibliothek und die Pin-Bibliothek importiert. Die Funktion **Connect\_WiFi** stellt wieder eine Verbindung zum lokalen WLAN her. Die Funktion **TempHum** gibt die Temperatur- und die Feuchtigkeitswerte zurück. Die Hauptprogrammschleife beginnt mit der **while**-Anweisung, innerhalb der die IP-Adresse der Thingspeak-Website extrahiert und eine Verbindung zu dieser Site hergestellt wird. Dann werden die Temperatur- und Feuchtigkeitswerte in das **path**-Statement eingesetzt. Die **sock.send**-Anweisung sendet eine HTTP-GET-Anfrage an die Thingspeak-Website und lädt die Temperatur- und Feuchtigkeitswerte hoch. Dieser Vorgang wird alle 30 s wiederholt.

Der Link zur Anzeige der Datendiagramme in der Cloud lautet wie in Kapitel 8:

<https://api.thingspeak.com/channels/265433>

```
>>> import network
>>> import socket
>>> import time
>>> import dht
>>> from machine import Pin
>>> APIKEY="QAAHT5XIK3ZB8736"
>>> def Connect_WiFi():
...     net=network.WLAN(network.STA_IF)
...     net.active(True)
...     net.connect("BTHomeSpot-XNH","49345abaeb")
>>> def TempHum():
...     d=dht.DHT11(Pin(23))
...     d.measure()
...     t=d.temperature()
...     h=d.humidity()
...     return t,h
>>> while True:
...     Connect_WiFi()
...     sock=socket.socket()
...     addr=socket.getaddrinfo("api.thingspeak.com",80)[0][-1]
...     sock.connect(addr)
...     t,h=TempHum()
...     host="api.thingspeak.com"
...     path="api_key="+APIKEY+"&field1="+str(t)+"&field2="+str(h)
...     sock.send(bytes("GET /update?"+path+" HTTP/1.0\r\nHost: "+host+"\r\n\r\n", "utf8"))
...     sock.close()
...     time.sleep(30)
...
95
95
95
```

Bild 12.15: Das MicroPython-Programm wurde interaktiv eingegeben

```
-----
# TEMPERATURE AND HUMIDITY ON THE CLOUD
# =====
#
# In this project temperature and humidity sensor is connected to
# GPIO port 23 of the ESP32-DevKitC board. The project reads the
# temperature and humidity and sends to the cloud where it can be accessed
# from anywhere. In addition, change of the temperature and the humidity
# can be plotted in the cloud.
#
# The DHT11 sensor chip is used to get the ambient temperature and the
# humidity.
#
# The program uses the Thingspeak cloud service. The ESP32-DevKitC must be
# connected to the Wi-Fi router before we can send data to the cloud.
#
#
# Date : May, 2017
# Programmer: Dogan Ibrahim
#-----
import network
import socket
import time
import dht
from machine import Pin
```

```

APIKEY = "QAAHT5XIK3ZB8736"
#
# This function connects to the Wi-Fi
#
def Connect_WiFi():
    net = network.WLAN(network.STA_IF)
    if not net.isconnected():
        net.active(True)
        net.connect(iBTHomeSpot-XNHi, i49345abaebi)
        print(net.ifconfig())
    #
# This function returns the temperature and humidity
#
def TempHun():
    d = dht.DHT11(Pin(0))
    d.measure()
    t = d.temperature()
    h = d.humidity()
    return t, h
#
# Send data to Thingspeak. This function sends the temperature and
# humidity data to the cloud every 30 seconds
#
while True:
    Connect_WiFi()
    sock = socket.socket()
    addr = socket.getaddrinfo("api.thingspeak.com", 80)[0][-1]
    sock.connect(addr)
    (t, h) = TempHun()
    host = "api.thingspeak.com"
    path = "api_key="+APIKEY+"&field1="+str(t)+"&field2="+str(h)
    sock.send(bytes("GET /update?"+path+" HTTP/1.0\r\nHost: "+host+"\r\n\r\n", "utf8"))
    sock.close()
    time.sleep(30)

```

*Bild 12.16 Programmlisting mit Kommentaren*

## 12.10 Fernbedienung über Mobiltelefon (Webserver)

Dieses Projekt entspricht dem Projekt aus Kapitel 10. Bei diesem Projekt sind zwei LEDs an einem ESP32-DevKitC-Entwicklungsboard angeschlossen. Die Idee dabei ist, dass das DevKitC als UDP-Client konfiguriert wird und UDP-Pakete von einem Mobiltelefon gesendet werden, um die LEDs zu steuern. Wie bereits erwähnt, können diese LEDs durch Relais ersetzt und das Projekt so zur Steuerung aller möglichen elektrischen Geräte verwendet werden, die man an das ESP32-DevKitC angeschließen.

Gültige Befehle sind wie folgt (alle anderen Befehle werden vom Programm ignoriert):

```
0=ON  LED0 einschalten  
0=OFF LED0 ausschalten  
1=ON  LED1 einschalten  
1=OFF LED1 ausschalten
```

Das Blockschaltbild und der Schaltplan entsprechen denen aus Bild 10.1 beziehungsweise Bild 9.2. Die LEDs sind wieder über  $330\text{-}\Omega$ -Strombegrenzungswiderstände an den GPIO-Ports 23 (LED0) und 22 (LED1) angeschlossen. Die LEDs leuchten, wenn der Ausgang des Ports logisch 1 (high) ist, und sind dunkel, wenn der Ausgang des Ports logisch 0 (low) ist.

In diesem Projekt wird ein Android-basiertes Handy verwendet, auf dem die UDP-Apps **UDP RECEIVER and SENDER** von Wezzi Studios (Version 4.0) installiert ist, um Befehle an das ESP32-DevKitC zu senden. Wenn Sie auf die Schaltfläche SEND UDP MESSAGE klicken, wird das Paket an sein Ziel gesendet. Im Empfängerteil wird die lokale IP-Adresse angegeben. Der Benutzer muss nur die erforderliche Port-Nummer angeben. Empfangene Pakete werden auf dem Bildschirm angezeigt (weitere Informationen hierzu finden Sie in Kapitel 10).

Bild 12.17 zeigt das vollständige MicroPython-Programm.

```
-----  
#           WEB SERVER TO CONTROL 2 LEDs  
#-----  
  
# This is a simple web server program. 2 LEDs are connected to GPIO ports 23  
# 22 of the ESP32-DevKitC development board. The project controls these LEDs  
# (turns ON or OFF) from a web server application. For example, the LEDs can  
be  
# controlled from any device that is on the web, for example a PC, tablet,  
# mobile phone etc. When activated, a form will appear on the device with  
# buttons and clicking these button will control the LEDs. The LEDs are  
named as  
# LED0 and LED1, connected to ports 23 and 22 respectively.  
#  
#  
# Date      : May, 2017  
# Programmer: Dogan Ibrahim  
#-----  
import network  
import socket  
from machine import Pin  
#  
# Configure the pins as outputs  
#  
LED0 = Pin(23, Pin.OUT)  
LED1 = Pin(22, Pin.OUT)
```

```
#  
# Turn OFF the LEDs to start with  
#  
LED0.low()  
LED1.low()  
  
#  
# The HTML code  
#  
html = """<!DOCTYPE html>  
<html>  
<body>  
<center><h1>ESP32 DECKITC LED ON/OFF</h1></center>  
<center><h2>MicroPython Web Server Example with 2 LEDs</h2></center>  
<form>  
  
<button name="LED0" button style="color:green" value="ON" type="submit">LED0  
ON</button>  
<button name="LED0" button style="color:red" value="OFF" type="submit">LED0  
OFF</button><br><br>  
  
<button name="LED1" button style="color:green" value="ON" type="submit">LED1  
ON</button>  
<button name="LED1" button style="color:red" value="OFF" type="submit">LED1  
OFF</button>  
</form>  
</body>  
</html>  
"""  
  
#  
# This function connects to the Wi-Fi  
#  
def Connect_WiFi():  
    net = network.WLAN(network.STA_IF)  
    if not net.isconnected():  
        net.active(True)  
        net.connect("BTHomeSpot-XNH", "49345abaeb")  
  
    #  
    #  
    #  
Connect_WiFi()  
addr=socket.getaddrinfo("0.0.0.0",80)[0][-1]  
s = socket.socket()
```

```
s.bind(addr)
s.listen(1)
while True:
    conn, address = s.accept()
    request = conn.recv(1024)
    request = str(request)
    LEDON0 = request.find("/?LED0=ON")
    LEDOFF0 = request.find("/?LED0=OFF")
    LEDON1 = request.find("/?LED1=ON")
    LEDOFF1 = request.find("/?LED1=OFF")
    if LEDON0 != -1:
        LED0.high()
    if LEDOFF0 != -1:
        LED0.low()
    if LEDON1 != -1:
        LED1.high()
    if LEDOFF1 != -1:
        LED1.low()
    resp = html
    conn.send(resp)
    conn.close()
```

Bild 12.17 Programmlisting mit Kommentaren

Zu Beginn des Programms werden die Netzwerkbibliotheken und die Pin-Bibliothek eingebunden, LED0 und LED1 den GPIO-Ports 23 und 22 zugewiesen, diese Ports als Ausgänge konfiguriert und die LEDs zunächst ausgeschaltet. Das Programm definiert dann den HTML-Code, den er auf ein Request hin an den Web-Client sendet. Dieser Code wird in der Variablen `html` gespeichert.

Das Webserverprogramm sendet die HTML-Datei (Anweisungen `resp=html` und `conn.send (resp)`) an den Client und wartet auf dessen Befehl. Wenn man beim Client eine Schaltfläche drückt, wird zusätzlich zu einigen anderen Daten ein Befehl an den Webserver gesendet. Hier interessiert nur dieser Befehl. Die folgenden Befehle werden je nach gedrückter Schaltfläche an den Webserver gesendet:

Gedrückter Button	Zum Webserver gesendeter Befehl
LED0 ON	?LED0=ON
LED0 OFF	?LED0=OFF
LED2 ON	?LED2=ON
LED2 OFF	?LED2=OFF

Das Webserverprogramm auf dem ESP32-DevKitC stellt eine TCP-Verbindung her und liest den Befehl vom Webclient (`request = conn.recv(1024)`). Das Programm durchsucht den empfangenen Befehl, um festzustellen, ob einer der obigen Zeichenfolgen in den empfangenen Daten enthalten ist. Beachten Sie, dass `find` einen String nach einem Sub-String durchsucht und die Startposition des Sub-Strings im String zurückgibt. Wenn kein gültiger

Sub-String gefunden wurde, wird -1 zurückgegeben. Betrachten Sie beispielsweise die folgende Anweisung:

```
LEDON0 = request.find("/?LED0=ON")
```

Hier wird im String **request** der Sub-String **/?LED0=ON** gesucht und, falls gefunden, die Startposition des Sub-Strings in der Zeichenfolge angegeben. Eine -1 wird zurückgegeben, wenn der Sub-String **/?LED0=ON** nicht vorhanden ist. Das Programm sucht nach einer Übereinstimmung für alle vier gültigen Befehle und schaltet, wenn ein Befehl gefunden wurde, die entsprechende LED entsprechend ein oder aus:

```
LEDON0 = request.find("/?LED0=ON")
LEDOFF0 = request.find("/?LED0=OFF")
LEDON2 = request.find("/?LED2=ON")
LEDOFF2 = request.find("/?LED2=OFF")
if LEDON0 != -1:                                # Wenn ON0-Befehl gefunden wurde
    LED0.high()                                 # LED0 einschalten
if LEDOFF0 != -1:                               # Wenn OFF0-Befehl gefunden wurde
    LED0.low()                                  # LED0 ausschalten
if LEDON2 != -1:                                # Wenn ON2-Befehl gefunden wurde
    LED2.high()                                 # LED2 einschalten
if LEDOFF2 != -1:                               # Wenn OFF2-Befehl gefunden wurde
    LED2.low()                                  # LED2 ausschalten
```

In diesem Projekt war die IP-Adresse des Webservers 192.168.1.156. Das System wird folgendermaßen getestet:

- Führen Sie das MicroPython-Programm entweder interaktiv oder von einer Datei aus.
- Öffnen Sie den Webbrowser auf Ihrem PC und geben Sie die Adresse 192.168.1.156 der Webseite ein.
- Die HTML-Seite zur Steuerung wird auf dem PC-Bildschirm angezeigt.
- Klicken Sie auf eine Taste (zum Beispiel LED0 ON), um eine LED ein- oder auszuschalten.

### **12.11 Laden von MicroPython-Programmen auf das ESP32-DevKitC**

Bisher haben wir die MicroPython-Programme interaktiv ausgeführt, wobei das Programm über den Terminalemulator PuTTY eingegeben wurde. Es gibt aber viele Anwendungen, in denen ein Programm auf das ESP32-DevKitC hochgeladen und beim Booten oder nach einem RESET automatisch ausgeführt werden soll. In diesem Abschnitt wird beschrieben, wie dies realisiert werden kann. Als Beispielprojekt soll ein kleiner Servo-Motor mit einer Verzögerung von drei Sekunden um 180 Grad in jede Richtung drehen. Dies soll automatisch nach dem (Neu-) Start des ESP32-DevKitC geschehen.

### 12.11.1 Verwenden der Ampy

Das Adafruit MicroPython Tool **ampy** ist ein Hilfsprogramm, das seriell mit einer ESP32-Karte interagiert, auf der MicroPython geladen ist. Ampy ist ein einfaches Befehlszeilen-Tool, mit dem die folgenden Operationen auf dem MicroPython-Board ausgeführt werden können:

- Inhalt eines Verzeichnisses auf der MicroPython-Board anzeigen (Befehl: **ls**)
- Ordner auf dem MicroPython-Board anlegen (Befehl: **mkdir**)
- Datei vom MicroPython-Board abrufen (Befehl: **get**)
- Datei vom PC an das MicroPython-Board senden (Befehl: **put**)
- Ordner samt Inhalt vom MicroPython-Board entfernen
- Skript auf dem PC ausführen und dessen Ausgabe anzeigen (Befehl: **run**)
- Reset des Boards durchführen (Befehl: **reset**)

Bevor man es verwenden kann, sollte man ampym natürlich auf dem PC installieren, was aber erst geschehen darf, wenn Python (2.7.x oder 3.x) bereits auf dem PC installiert ist. In diesem Abschnitt gehen wir davon aus, dass Python 2.7 bereits auf dem PC installiert ist. Um **ampy** zu installieren, geben Sie folgenden Befehl über die Befehlszeile auf Ihrem PC ein (siehe Bild 12.18):

```
C:\> cd python27\scripts  
C:\> Python27\Scripts> pip install adafruit-ampy
```

```
C:\>cd Python27\scripts  
C:\Python27\Scripts>pip install adafruit-ampy  
Collecting adafruit-ampy  
  Downloading adafruit_ampy-1.0.1-py2.py3-none-any.whl  
Collecting click (from adafruit-ampy)  
  Downloading click-6.7-py2.py3-none-any.whl (71kB)  
    100% :#####: 71kB 484kB/s  
Requirement already satisfied: pyserial in c:\python27\lib\site-packages (from adafruit-ampy)  
Installing collected packages: click, adafruit-ampy  
Successfully installed adafruit-ampy-1.0.1 click-6.7  
C:\Python27\Scripts>
```

Bild 12.18 Installation von ampym

Sie können die erfolgreiche Installation von **ampy** mit dem folgenden Befehl überprüfen (siehe Bild 12.19):

```
C:\Python27\Scripts>ampy --help
```

```
C:\Python27\Scripts>ampy --help
Usage: ampy [OPTIONS] COMMAND [ARGS]...
      ampy - Adafruit MicroPython Tool

      Ampy is a tool to control MicroPython boards over a serial connection.
      Using ampy you can manipulate files on the board's internal filesystem
      and even run scripts.

Options:
  -p, --port PORT  Name of serial port for connected board. Can optionally
                   specify with AMPY_PORT environment variable. [required]
  -b, --baud BAUD  Baud rate for the serial connection (default 115200). Can
                   optionally specify with AMPY_BAUD environment variable.
  --version         Show the version and exit.
  --help            Show this message and exit.

Commands:
  get   Retrieve a file from the board.
  ls    List contents of a directory on the board.
  mkdir Create a directory on the board.
  put   Put a file or folder and its contents on the...
  reset Perform soft reset/reboot of the board.
  rm    Remove a file from the board.
```

Bild 12.19 AmPy-Hilfe

Ein Beispiel für die Verwendung von ampy ist unten aufgeführt. Bevor Sie diese Befehle verwenden, sollten Sie das ESP32-DevKitC an den PC anschließen und die Nummer der seriellen Schnittstelle (COM47 in den folgenden Beispielen) ermitteln.

Ein Beispiel finden Sie in Bild 12.20, in dem mit dem List-Befehl die Dateien auf dem MicroPython-Board aufgeführt werden:

```
C:\Python27\Scripts> ampy --port COM47 ls
```

```
C:\Python27\Scripts>ampy --port COM2 ls
boot.py
C:\Python27\Scripts>
```

Bild 12.20 Auflisten von Dateien auf dem Board

### 12.11.2 Erstellen und Ausführen eines Programms

Mit ampy kann man ein MicroPython-Programm auf dem MicroPython-Board ausführen. Das Programm kann mit unserem bevorzugten Texteditor Notepad (oder sogar ESPlorer) erstellt werden. Ein Beispielprogramm im Editor ist in Bild 12.21 dargestellt. Das Programm heißt **sinetable.py** und wird im Unterordner **C:\Python\Scripts** gespeichert. Dieses Programm zeigt die Sinuswerte für Winkel von 0 ... 90 Grad in Schritten von 5 Grad an.

```
#-----
#
#          TRIGONOMETRIC SINE TABLE
#          =====
#
# This program tabulates the trigonometric sine of angles from 0
# to 90 degrees in steps of 5 degrees
#
#
# Program: sinetables.py
```

```
# Date    : August, 2017
# Author  : Dogan Ibrahim
#-----
import math

print ("TABLE OF TRIGONOMETRIC SINE")
print ("=====")
print ("")
print ("  ANGLE      SINE")

for angle in range (0, 95, 5):
    r = math.radians(angle)
    s = math.sin(r)
    print (" %d      %f" %(angle, s))

print(„End of program”)
```

*Bild 12.21 Das Programm sinetable.py*

Mit dem folgenden Befehl kann man das Programm ausführen:

```
C:\Python27\Scripts>ampy --port COM47 run sinetable.py
```

Die Ausgabe des Programms auf dem PC-Bildschirm ist in Bild 12.22 dargestellt. Standardmäßig wartet der Befehl run mit der Anzeige, bis das Programm auf dem Board abgearbeitet ist. In vielen Mikrocontroller-Anwendungen wird aber ein Programm in einer Endlosschleife ausgeführt und endet nie. In solchen Anwendungen müssen Sie die Option **--no-output** im Befehl run einsetzen. Wenn Sie diese Option wählen, wird ampy angewiesen, nicht auf eine Ausgabe zu warten, sondern nur das Programm wiederholt auszuführen. Man muss dabei auch berücksichtigen, dass keine Tastatureingaben an das Programm gesendet werden können.

Die Option -no-output wird wie folgt verwendet:

```
C:\Python27\Scripts>ampy --port COM2 run --no-output sinetable.py
```

TABLE OF TRIGONOMETRIC SINE	
<hr/>	
ANGLE	SINE
0	0.000000
5	0.087156
10	0.173648
15	0.258819
20	0.342020
25	0.422618
30	0.500000
35	0.573576
40	0.642787
45	0.707107
50	0.766044
55	0.819152
60	0.866025
65	0.906308
70	0.939692
75	0.965926
80	0.984808
85	0.996195
90	1.000000

Bild 12.22 Ausgabe des Programms

### 12.11.3 Ausführen eines Programms beim Booten

In einigen Anwendungen möchten Sie vielleicht ein Programm automatisch ausführen, wenn das ESP32-DevKitC eingeschaltet wird. Dies kann wie unten beschrieben ganz einfach durchgeführt werden.

Wenn das Board mit installiertem MicroPython hochgefahren wird, führt es automatisch die Firmware-Datei **boot.py** aus. Diese Datei sollte normalerweise nicht geändert werden, es sei denn, Sie möchten den Boot-Vorgang anpassen. Sie können den ampy-Befehl **get** verwenden, um den Inhalt der Datei **boot.py** anzuzeigen. Wenn die Ausführung von **boot.py** abgeschlossen ist, ruft es ein MicroPython-Programm namens **main.py** auf, das sich auf dem Board befindet. Oder auch nicht: Wenn es kein Programm **main.py** gibt, können wir unser Programm in **main.py** umbenennen und es dann auf das Board hochladen. Auf diese Weise startet unser Programm automatisch beim Booten. Die Schritte sind unten angegeben

- Schreiben Sie das gewünschte Programm (zum Beispiel **sinetable.py** wie in Bild 12.20).
- Benennen Sie das Programm in **main.py** um.
- Schließen Sie das ESB32-DevKitC an Ihren PC an.
- Laden Sie die Datei mit dem folgenden ampy-Befehl auf das Board hoch:

```
C:\Python27\Scripts> ampy --port COM47 put main.py
```

- Überzeugen Sie sich davon, dass die Datei **main.py** hochgeladen wurde. Geben Sie folgenden Befehl in ampy ein, damit Sie die eine Liste mit den beiden Dateien **boot.py** und **main.py** erhalten:

```
C:\Python27\Scripts> ampy --port COM47 ls
```

- Starten Sie das Terminal-Emulationsprogramm (PuTTY).
- Setzen Sie das Board zurück und starten Sie es neu. Nun sollte die Ausgabe im Fenster des Terminalemulators zu sehen sein.

Das folgende Beispiel zeigt einen kleinen, preisgünstigen Servo-Motor Tower Pro SG90 (siehe Bild 12.23), der so angesteuert wird, dass er vollständig nach links und rechts dreht, jeweils mit Pausen von drei Sekunden.



Bild 12.23 Der Servo Tower Pro SG90

Das Servo hat folgende Anschlüsse

- Schwarz: Masse
- Rot: +3,3 V
- Braun: Signal

Einige wichtige Eigenschaften dieses Servos sind:

- Abmessungen: 22 mm x 12 mm x 31 mm
- Betriebsgeschwindigkeit: 0,3 s für einen 60-Grad-Winkel
- Gewicht: 9 g
- Betriebsspannung: 4,2 ... 6 V

Obwohl als minimale Betriebsspannung des Servos 4,2 V angegeben sind, arbeitet er auch an +3,3 V. Deshalb können in diesem Projekt der Signalanschluss des Servos direkt an GPIO-Pin 23 und die anderen Anschlüsse auf Masse und +3,3 V gelegt werden.

Der Servo wird mit einem PWM-Signal mit einer Frequenz von 50 Hz (Periode = 20 ms) angesteuert. Wenn die Impulsbreite ungefähr 2 ms beträgt, dreht sich der Servo vollständig in eine Richtung, bei einer Impulsbreite von 1 ms in die entgegengesetzte Richtung. Man kann das PWM-Tastverhältnis (duty cycle) so ausdrücken:

Tastverhältnis = (Impulsbreite / Periode) × 100 %  
 oder  
 Tastverhältnis = (Impulsbreite \* Frequenz) × 100 %

Für die 2-ms-Impulse wäre also:

$$\text{Tastverhältnis} = (0,002 * 50) \times 100 \% = 10 \%$$

und für die 1-ms-Impulse

$$\text{Tastverhältnis} = (0,001 * 50) \times 100 \% = 5 \%$$

Da die PWM-Auflösung 10 Bit beträgt, entspricht der Maximalwert (100 %) dem Tastverhältnis 1024. Für die Tastverhältnisse von 10 % und 5 % müssen also die Werte 102 beziehungsweise 51 eingesetzt werden. In der Praxis verhielt es sich jedoch so, dass der Servo bei einem Tastverhältnis von 126 vollständig in die eine und bei 15 vollständig in die andere Richtung gedreht war.

Bild 12.24 zeigt das Programmlisting. Der GPIO-Pin 23 wird als PWM-Kanal konfiguriert und die Frequenz auf 50 Hz eingestellt. Das Tastverhältnis wird in einer Schleife geändert, so dass der Servo in beide Richtungen dreht. Nach jeder Rotation wird eine Pause von drei Sekunden eingelegt.

```
#-----
#                               SERVO CONTROL
#
#
# In this program a small servo motor is connected to GPIO
# port 23 of the ESP32-DevKitC. The program sends PWM pulses
# to the servo to rotate it fully in one direction, and after
# 3 seconds in the opposite direction. This process is repeated
# until stopped manually.
#
#
# File  : main.py
# Author: Dogan Ibrahim
# Date  : August, 2017
#-----

import machine
import time
p23 = machine.Pin(23)
pwm23 = machine.PWM(p23)
pwm23.freq(50)      # set frequency to 50 Hz
while True:          # Do Forever
    pwm23.duty(20)   # Turn in one direction
```

```
time.sleep(3)      # Wait 3 seconds
pwm23.duty(128)   # Turn in opposite direction
time.sleep(3)      # Wait 3 seconds
```

*Bild 12.24 MicroPython-Programmlisting*

Erstellen Sie das Programm mit dem Editor und speichern Sie es unter dem Namen **main.py** im Ordner **C:\Python27\Scripts**. Kopieren Sie dann das Programm mit dem folgenden Befehl auf das ESP32-DevKitC. Starten Sie das ESP32-DevKitC neu und sehen Sie, wie sich der Servo erwartungsgemäß dreht.

```
C:\Python27\Scripts> ampy --port COM47 put main.py
```

Sie können das Programm aus dem Stammverzeichnis des ESP32-DevKitC mit dem folgenden Befehl entfernen, damit es beim Booten nicht mehr ausgeführt wird.

```
C:\Python27\Scripts> ampy --port COM47 rm main.py
```

## 12.12 Zusammenfassung

In diesem Kapitel haben wir den MicroPython-Interpreter auf einem ESP32-DevKitC installiert. Außerdem zeigen einige Beispiele und Projekte, wie ein MicroPython-Programm entwickelt und auf dem ESP32-DevKitC ausgeführt werden kann.

## Index

### Symbole

7-Segment-Anzeige

### H

Hall-Sensor

27

HTTP

220

### A

Ampy

272

### I

Analog-Digital-Wandler

27

27

Android

245

27

Anode

126

28

Arduino-IDE

45

27

### B

Binärzähler

71

### K

Bluemix

230

Kathode

126

80

### C

Cloud

230

### L

Cloud-Dienste

230

### LED

Cloudino

230

LED-Farbtafel

55

CPU

25

LED-Würfel

89

Leuchtturm

112

LoLin32 ESP32 Development Board

57

30

### D

Digital-Analog-Wandler

27

### M

### E

Ereigniszähler

MAC-Adressen

215

ESP32

142

MakerHawk

32

ESP32-Datasheet

23

Mathematik

205

ESP32-DevKitC

24

MicroPython

256

ESP32-DevKitC Hardware

32

Mini-Orgel

175

ESP32 OLED Development Board

35

Mobiltelefon

244

ESP32 Technical Reference Manual

31

Mongoose OS

256

ESP32 Test Board

25

### N

ESP8266

31

Netzanwendungen

215

ESP-IDF

246

Netzspannung

163

Netzwerke

215

Netzwerknamen

215

### F

Fernsteuerung

237

### P

Funknetzwerk

215

### Pycom LoPy

30

### G

Geekcreit ESP32 Development Board

29

Pesky Products

31

Geräte-Manager

38

Pulsweitenmodulation

28

Git

49

Pycom LoPy

45

GPIO

26

Python 2.7

**S**

Sägezahn	163
Servo	276
SMA42056	126
Sound-Sensormodul	134
SparkFun	230
SparkFun ESP32 Thing	28
SPI-Schnittstelle	28
Summer	86
Systemtakt	26

**T**

Takt	26
Temperatursensor	27, 95
Thermostat	156
Thingspeak	230
Timer-Interrupt	153
TMP36	95
Touch-Sensor	27
Transceiver	26
Türschloss	208

**U**

UART	27
UDP	262
UDP RECEIVER and SENDER	251
Universal-Timer	26

**W**

Watchdog-Timer	26
Webserver	267
Weihnachtsbeleuchtung	66
Wellenform	163
Wezzi Studios	251
Wi-Fi	215
WLAN	215





# ENTDECKE DEN IoT-CHIP

# DAS OFFIZIELLE ESP32-HANDBUCH

# Dogan and Ahmet Ibrahim

Prof.Dr. Dogan Ibrahim

hat einen Bachelor-Abschluss in der Elektroniktechnik, einen Master in Steuerungstechnik und hat sich über digitalen Signalverarbeitung promoviert. Er ist Autor von über 60 Fachbüchern und über 200 Fachartikeln über Mikrocontroller, Mikroprozessoren und verwandte Gebiete.

**Ahmet Ibrahim** erhielt einen Bachelor-Abschluss von der Greenwich University in London. Danach absolvierte er dort einen Master-Kurs. Ahmet erwarb sich Berufserfahrung durch die Arbeit bei vielen Industrieorganisationen auf verschiedenen Ebenen und ist derzeit in einer großen IT-Organisation aktiv.

ISBN 978-3-89576-329-8



9 783895 763298

Elektor-Verlag GmbH  
52072 Aachen  
[www.elektor.de](http://www.elektor.de)

Die chinesische Firma Espressif Systems hat vor kurzem den neuen Mikrocontroller ESP32 entwickelt, das Nachfolgemodell des ESP8266-Wifi-Chips. Zusätzlich zu allen ESP8266-Funktionen stellt der ESP32 nun Bluetooth-Funk, einen größeren SRAM-Datenspeicher, mehr GPIOs, mehr Schnittstellen, einen Touch-Sensor, einen Temperatursensor, höhere CPU-Geschwindigkeit, einen CAN-Bus, ADCs mit höherer Auflösung, DACs und Sicherheitsfunktionen bereit. Der ESP32 ist Arduino-kompatibel.

Dieses Buch ist eine Einführung in den ESP32-Prozessor und beschreibt die wichtigsten Hardware- und Software-Features dieses Chips. Das Hauptziel des Buches ist es, dem Leser zu vermitteln, wie man ESP32-Hard- und Software in praktischen Projekten einsetzt, vor allem durch Verwendung des hochmodernen ESP32-Entwicklungsboards. Im Buch sind viele einfache, grundlegende und weitere Projekte mittlerer Schwierigkeit auf Basis des Entwicklungsboards ESP32 DevKitC samt der beliebten Arduino-IDE und der Programmiersprache MicroPython enthalten. Die Projekte sind nach aufsteigender Komplexität organisiert.

Das speziell für das Buch vorbereitete **ESP32-Hardware-Kit** ist bei Elektor erhältlich. Das Kit enthält alle Bauteile, die in den Buchprojekten verwendet werden.

