

Arduino Cheat Sheet

Für Board-Support ab Version 3.x

Arduino Language Reference <https://arduino.cc/de/Reference/>
Espressif Reference <https://docs.espressif.com/projects/arduino-esp32/en/latest/>

Programmstruktur

Basis Programm Struktur

```
void setup ( ) {
  // einmal zu Beginn ausführen
}
void loop ( ) {
  // wiederholt ausführen
}
```

Kontroll-Strukturen

```
if (x < 5) { ... } // mehrere Beding.
else if (x < 10) { ... } // && bzw. ||
else { ... }
while (x < 5) { ... }
do { ... } while (x < 5); //mind. 1x
for (int i=0; i<10; i++) { ... }
break ; // Schleife abbrechen
continue ; // nächste Wiederholung
switch (var) {
  case 1: ... // wenn var = 1
  break; // Ende des Falles
  case 2: ... // wenn var = 2
  break;
  case 3: // wenn var = 3
  case 4: ... // oder var = 4
  break;
  default: ... // alle übrigen Fälle
  break;
}
```

Funktions-Definitionen

```
<ret. Type> <name>(<params>) { ... }
z.B. int doppel(int x) {return x*2;}
return x; // Datentyp wie <ret. Type>
return; // bei void-Funktion
```

Operatoren

Allgemeine Operatoren

```
= Zuweisung von rechts nach links
+ plus - minus
* mal / geteilt
% Modulo (Rest der GZ-Division)
== Gleich wie != Ungleich
< Kleiner als > Größer als
<= Kleiner oder gleich
>= Größer oder gleich
&& AND (log.) || OR (log.)
! NOT (log.) () Klammerung
```

Zusammengesetzte Operatoren

```
++ Inkrement / hochzählen
-- Dekrement / runterzählen
+= Addition mit Zuweisung
-= Subtraktion mit Zuweisung
*= Multiplikation mit Zuweisung
/= Division mit Zuweisung
&= Bitweise AND mit Zuweisung
|= Bitweise OR mit Zuweisung
<<= Links schieben mit Zuweisung
>>= Rechts schieben mit Zuweisung
X++ Post-Inkrement ++X Prä-Inkrement
X-- Post-Dekrement --X Prä-Dekrement
```

Bitweise Operatoren

```
& bitweise AND | bitweise OR
^ bitweise XOR ~ bitweise NOT
<< shift links >> shift rechts
```

Pointer Zugriff (Zeiger)

```
& Referenz: liefert einen Pointer
* Dereferenz: Inhalt, auf den der Pointer zeigt
```

Standard - Funktionen

Digital I/O - ESP32 Pins34-39 nur In
`pinMode (pin, [INPUT, OUTPUT, INPUT_PULLUP])`

```
bool digitalRead (pin)
digitalWrite (pin, [HIGH, LOW])
```

Analog in - ESP32 Pins A0-A9

```
int analogRead (pin)
int analogReadMillivolts (pin)
analogReadResolution (bits)
//Auflösung der Wandlung einstellen
//Default: ESP32 12Bit, EPS32-S3 13Bit
```

PWM out - Beim ESP32 mit `ledc()` !

```
bool ledcAttach (pin, freq, bits)
bool ledcAttachChannel (pin, freq, bits, channel) // Kanal 0-15
ledcWrite (pin, value)
ledcWriteChannel (channel, value)
ledcWriteTone (pin, freq)
ledcFade (pin,start,end,duration_ms)
analogWrite (pin, value) //kompatibel // zu Uno, Nano, ESP8266
```

Advanced I/O

```
tone (pin, freq_Hz) // Ton erzeugen
tone (pin, freq_Hz, duration_ms)
noTone (pin) // Ton ausschalten
shiftOut (dataPin, clockPin, [MSBFIRST, LSBFIRST], value)
unsigned long pulseIn (pin, [HIGH, LOW]) // Messe Dauer LOW- // oder HIGH-Teil eines Signals
```

Zeiten (ESP32-Timer siehe Rückseite)

```
delay (msec)
delayMicroseconds (usec)
unsigned long millis () // läuft nach 50 Tagen über
unsigned long micros () // läuft nach 70 Minuten über
```

Math

```
min (x, y) min (x, y) abs (x)
sin (rad) cos (rad) tan (rad)
sqrt (x) pow (base, exponent)
int constrain(x, minval, maxval)
int map (val, x1, x2, y1, y2)
```

Random Numbers

```
randomSeed (seed); // long oder int
long random (max); // 0 bis max-1
long random (min, max);
```

Bits und Bytes

```
byte lowByte (x) byte highByte (x)
bool bitRead (x, bitn)
bitWrite (x, bitn, bit)
bitSet (x, bitn) bitClear (x,bitn)
byte bit (bitn) // bitn: 0=LSB 7=MSB
```

Typ Umwandlung

```
char (val) byte (val)
int (val) short (val)
long (val) float (val)
```

Externe Interrupts

```
Interrupt-Nummer bestimmen mit
intNr = digitalPinToInterrupt(pin)
attachInterrupt (intNr, &func, [LOW, CHANGE, RISING, FALLING])
detachInterrupt (intNr)
interrupts () //Interrupts zulassen
noInterrupts ()//Interrupts sperren
```

Variablen, Arrays und Definitionen

Datentypen

```
bool true || false
char -128 - 127, 'a', '?' (int8_t)
unsigned char 0 - 255 (uint8_t)
byte 0 - 255 (uint8_t)
short -32768 - 32767 (int16_t)
unsigned short 0 - 65535 (uint16_t)
int, unsigned int bei ESP32 mit 32 Bit
wie (unsigned) long (int32_t, uint32_t)
long -2147483648 - 2147483647 (231-1)
unsigned long 0 - 4294967295 (232-1)
float ±1.1755e-38 - ±3.4028e+38 (4Byte)
double ±5,0e-324 - ± 1,7e+308 (8Byte)
void Kein Rückgabewert oder Parameter
long long oder int64_t -263 - 263-1
unsigned long long o. uint64_t 0 - 264-1
```

Strings

```
char str1[8] = {'A', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
// C-String endet mit NULL-Zeichen \0
char str2[8] = "Arduino"; // \0 mitzählen
char str3[] = "Arduino";
String str4 = "Arduino"; // String-Klasse
String str5 = "Sensor-Wert: "
+ String(analogRead(A0)) + "\n";
```

Numerische Konstanten

```
123 Normale Dezimalzahl
0b01111011 Binäre schreibweise
0x7B Hexadezimalzahl (0 bis F)
0173 Oktalzahl - vermeiden!
123U Erzwinge unsigned
123L Erzwinge long
123UL Erzwinge unsigned long
123.0 Kommazahl
1.23e6 Exponent-Darstellung
1.23*106 = 1230000
```

Qualifier

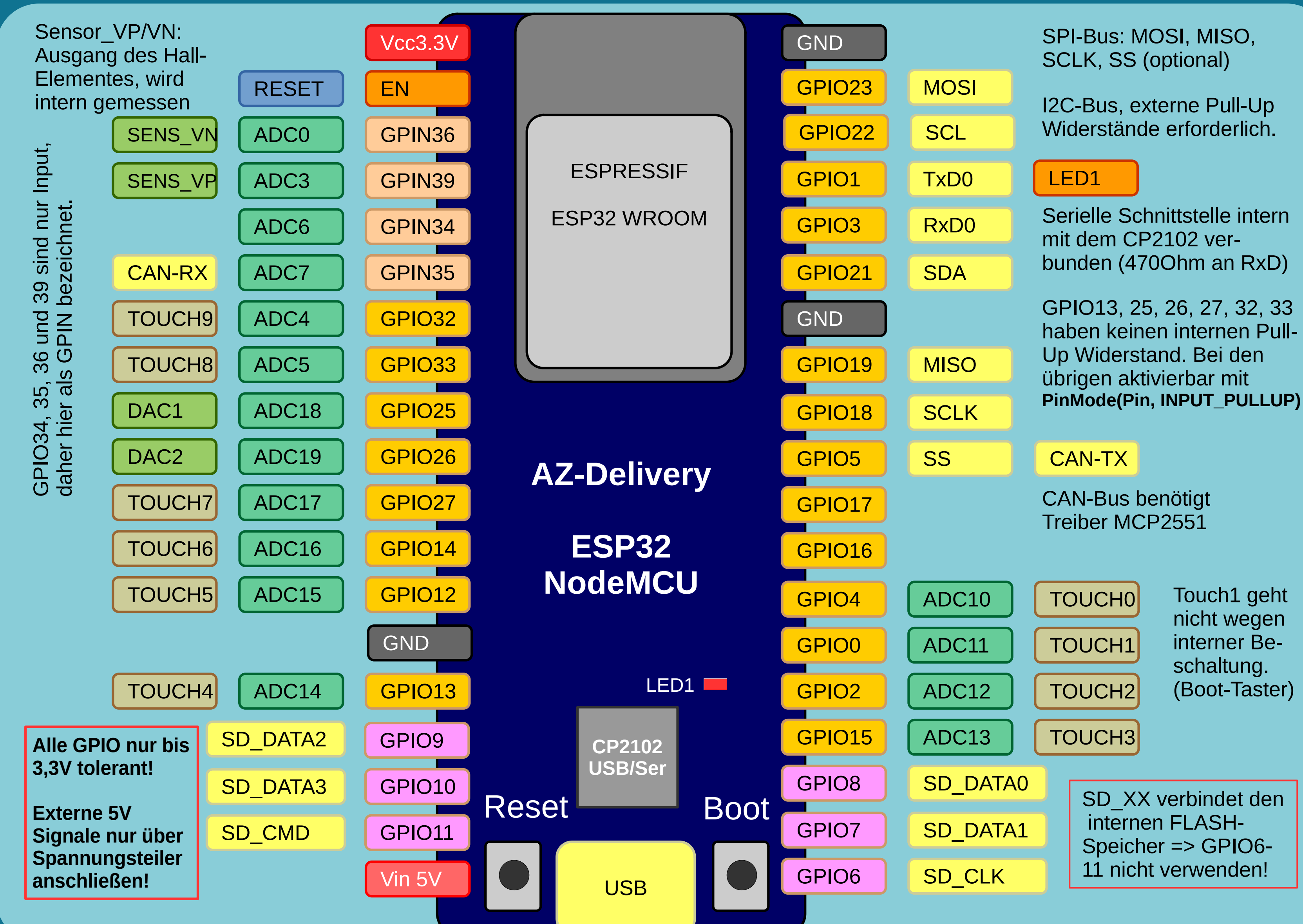
```
static Variable bleibt erhalten
volatile im RAM (notwendig für ISR)
const Nicht veränderbar (z.B.Pin)
```

Arrays

```
const int MyPins[] = { 2, 4, 8, 3, 6};
int MyInts[6]; // Array für 6 Werte
MyInts[0] = 42; // Zuweisung an den // ersten Wert
MyInts[6] = 21; // Fehler! Index // von 0 bis 5
```

Makros

```
#define Suche Ersatz // Suche ohne // Leerzeichen, nach Ersatz kein ;
```



Bibliotheken

Diese Bibliotheksfunktionen werden mit `Name.Methode` genutzt.

```
Serial - Komm. mit PC bzw. RX/TX
begin (long speed)//bis 921600Bit/s
begin (speed, config) //Config z.B. //SERIAL_7E1 oder SERIAL_8N1
end ( )
int available() //Bytes angekommen
int read () // -1, wenn leer
int peek () // lesen ohne löschen
flush () // Sendepuffer leeren
print (data) println (data)
write (byte) write (char *str)
write (byte *data, size)
```

```
SoftwareSerial.h - an bel. Pin
SoftwareSerial (rxPin, txPin)
begin (long speed)//bis 115200Bit/s
listen ()// umschalten v. mehreren
isListening () // SoftwareSerial
available, read, write, print etc.
// Genauso wie bei Serial
```

```
EEPROM.h nicht-flüchtiger Speicher
byte read (addr)
write (addr, byte)
EEPROM[index] // Array-Zugriff
```

```
Servo.h - Bib. für Uno/ESP8266
ESP32Servo.h - Bib. für ESP32
attach (pin, [min_uS, max_uS])
write (angle) // 0 bis 180°
writeMicroseconds (us)
//ca.1000-2000; 1500 ist die Mitte
int read () // 0 bis 180°
bool attached () // Kontrolle
detach () // Freigabe Pin
```

```
Wire.h - I2C Kommunikation
setPins(sda, scl) // vor begin
begin () // Als Master
begin (addr) // Slave mit addr
beginTransaction(addr)//schreiben
write (byte) // Schritt 2
write (char *str)
write (byte *data, size)
//Schritt 3: false->Repeated Start
endTransmission ([true,false])
int available() //Bytes angekommen
requestFrom (addr, count)// lesen
byte read () // ein Byte lesen
onReceive (handler) // Callback-
onRequest (handler) // Funktionen
```

by Bernhard Spitzer
Version: 2024-09-10

Quelle: <https://github.com/liffiton/Arduino-Cheat-Sheet/>

Adapted from:
- Original: Gavin Smith
- SVG version: Frederic Dufourg
- ESP32 functions / board drawing: Bernhard Spitzer

Cheat Sheet für ESP32/ESP32-S3

Für Board-Support ab Version 3.x

Hauptquelle: Arduino Language Reference <https://arduino.cc/de/Reference/>

WLAN - Funktionen

WiFi.h - WLAN-Funktionen, allgemein
Zunächst das WLAN-Kennwort angeben:
`const char* ssid = "NetworkName";`
`const char* pass = "Password";`
Alternative (besser beim Weitergeben von Programmen):
`#include <credentials.h>` // im LIB-Verzeichnis anlegen,
// Inhalt: die beiden Zeilen von oben
`WiFi.begin(ssid, pass);` // Startet WLAN zu `ssid` als Client
`WiFi.begin(ssid);` // Verbindung mit offenem WLAN `ssid`
`IPAddress myIP = WiFi.localIP();` // IP-Adresse auslesen

`WiFi.status();` // Lese Status der Verbindung z.B. WL_CONNECTED
`WiFi.BSSID(bssid);` // Liest die MAC-Adresse des Routers
`long rssi = WiFi.RSSI();` // Liest die Signalstärke aus
`WiFi.disconnect();` // Verbindung trennen

WiFiAP.h - WLAN-Funktion Accesspoint
`WiFi.SoftAP(ssid, pass);` // Startet einen Accesspoint
`WiFi.SoftAP(ssid);` // offener Accesspoint
`IPAddress myIP = WiFi.softAPIP();` // IP-Adresse auslesen

`const char * softAPgetHostname();` // AP-Name auslesen
`bool softAPsetHostname(const char * hostname);` // AP-Name festlegen

`String softAPmacAddress(void);` // MAC-Adresse festlegen
`uint8_t * softAPmacAddress(uint8_t * mac);` // MAC-Adresse auslesen

HTTP(S) - Server

WebServer.h - für HTTP und HTTPS Verbindungen
Zunächst eine Objektinstanz erzeugen:
`WebServer server(80);` // Port 80 für HTTP, Port 443 für HTTPS

In `Setup()`:
`server.begin();` // Webserver starten, WLAN vorher verbinden
`server.onNotFound(Send404);` // Callback-Funktion für ungültige URL
`server.on("/", HandleRoot);` // Callback-Funktion für Web-Root
Weitere URLs: weitere Callback-Fkt. oder in `HandleRoot` abfragen

Webseiten ausliefern mit Status-Code und Inhaltstyp:
`server.send(200, "text/html", WebPage());` // WebPage liefert String
`void Send404(void) {` // hier als Funktion `Send404()`
`server.send(404, "text/plain", "Internet kaputt");`
`// sende unformatierten Text`
`}`

Auswerten von Server-Argumenten (für Formulare, Benutzereingaben):
`void HandleRoot(void) {`
`// DEBUG Ausgabe, zeigt alle Parameter des Aufrufs`
`Serial.printf("URI: %s, ", server.uri());`
`Serial.print("Method: ");`
`Serial.println((server.method() == HTTP_GET) ? "GET" : "POST");`
`Serial.printf("Arguments: %s\n", server.args());`
`// Alle Server-Argumente ausgeben`
`for (uint8_t i = 0; i < server.args(); i++) {`
`Serial.printf("%s : %s\n", server.argName(i), server.arg(i));`
`}`
`// ENDE DEBUG, jetzt nach bekannten Argumenten schauen;`
`if(server.hasArg("r") || server.hasArg("g") || server.hasArg("b"))`
`{ uint16_t red, green, blue; String Temp;`
`Temp = server.arg("r"); // Server sendet Strings`
`red = Temp.toInt(); // String in Zahlumwandeln`
`// jetzt was damit machen, z.B. LED-Farbe setzen`
`}`
`server.send(200, "text/html", WebPage());` // neue Webseite senden
`}`
Weitere Möglichkeiten siehe Programmbeispiele in Arduino!

Bluetooth - Funktionen

BluetoothSerial.h - Bluetooth zur Kommunikation
Zunächst eine Objektinstanz erzeugen:
`BluetoothSerial SerialBT;` // Name ist beliebig
In `Setup()`:
`SerialBT.begin("Geräte_1234");` // Eindeutiger Name

Danach können die gleichen Funktionen wie bei der seriellen Schnittstelle genutzt werden:
`SerialBT.available();` // Empfangsdaten vorhanden?
`SerialBT.println("Text");`
Zeichen = `SerialBT.read();`
`StringVar = SerialBT.readString();`
`SerialBT.end();`

Weitere Funktionen für die Verbindungssteuerung:
`void enableSSP();` // Secure Pairing einschalten
`bool setPin(*pin);` // z.B. `SerialBT.setPin("1234");`
`bool connect(remoteName);` // neue Verbindung
`bool connect();` // gespeicherte Verbindung
`bool connected(timeout=0);` // Verbindung ok?
`bool disconnect();`
`bool unpairDevice(remoteAddress[]);`

Callback-Funktion für Verbindungs-Ereignisse:
`SerialBT.register_callback(BT_Comm);` // in `Setup()`
`void BT_Comm(esp_spp_cb_event_t event,`
`esp_spp_cb_param_t *param) {`
`if(event == ESP_SPP_SRV_OPEN_EVT){`
`Serial.println("Client verbunden");`
`}`
`if(event == ESP_SPP_CLOSE_EVT){`
`Serial.println("Client getrennt");`
`}`

Weitere Event-Nummern und Definitionen:
docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/esp_spp.html
BluetoothA2DPSource.h und **BluetoothA2DPSink.h**
Bluetooth zur Audio-Übertragung (Quelle, Senke)
z.B. github.com/pschatzmann/ESP32-A2DP

Sleep - Modes

Der ESP32 kann in einen Sleep-Mode geschickt werden, in dem die Stromaufnahme stark reduziert ist. Vorher muss eine Quelle für den Wakeup definiert werden. Man kann auch eine feste Zeit einstellen, nach der der ESP wieder aufwacht.

`esp_sleep_enable_ext0_wakeup (pin, [HIGH, LOW]);`
`esp_sleep_enable_timer_wakeup (microSeconds);`
`esp_deep_sleep_start ();` // schlafen schicken

Touch - Eingänge

Der ESP32 hat 10 nutzbare Touch-Eingänge (ESP32-S3: 14). Die Touch-Pins müssen nicht mit `PinMode` initialisiert werden. Die Werte haben teilweise Störungen und sollten gefiltert werden.

`byte TouchValue = touchRead(T0);` // T0-T9

Rückgabewert 0 - 100, Touch-Ereignis bei Schwellwert 20-30 (je nach Beschaltung):
`bool digitalTouch(T_Chan) {`
`if(touchRead(T_Chan) < 30) return true;`
`else return false;`
`}`

Timer - Funktionen

Der ESP32 /ESP32-S3 hat 4 Universaltimer mit 64 Bit Zählumfang Eingangstakt 80MHz mit Vorteiler (:1 bis :65535)

`hw_timer_t *MeinTimer = NULL;` // Zeiger auf eine Timer-Struktur
`MeinTimer = timerBegin (Frequenz);` // Timerfrequenz in Hz, wenn
// `MeinTimer` danach noch NULL ist, liegt ein Fehler vor
`TimerStart (MeinTimer);` // Timer starten
`TimerStop (MeinTimer);` // Timer stoppen
`timerAlarm(MeinTimer, Alarmwert, Reload, Count);` // 64 Bit Alarmwert
// Reload true: nach Alarm bei 0 anfangen, Count -> Anzahl
// der automatischen Reloads (0 -> unendlich)
`timerAttachInterrupt(MeinTimer, &onTimer);` // ISR onTimer()
`uint64_t timerRead(MeinTimer);` // Zählerstand lesen (64Bit!)
`timerWrite(MeinTimer, uint64_t Wert);` // Zählerstand setzen
`timerRestart(MeinTimer);` // Zählerstand auf 0 setzen
`uint64_t timerReadMicros(MeinTimer);` // Mikrosekunden
`uint64_t timerReadMillis(MeinTimer);` // Millisekunden
`double timerReadSeconds(MeinTimer);` // Sekunden

