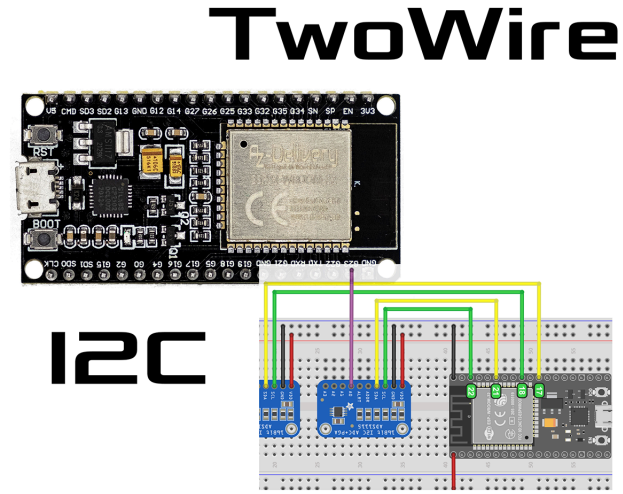
# I2C-Schnittstellen des ESP32 nutzen




## Über den Beitrag

In meinem letzten Beitrag habe ich darüber berichtet, wie ihr den TCA9548A oder einfache MOSFETs einsetzt, um mehrere Bausteine mit gleicher I2C Adresse zu steuern. In diesem Beitrag bleibe ich bei dem Thema und zeige, wie ihr die I2C-Schnittstellen des ESP32 nutzen könnt. Denn auch damit seid ihr unter Umständen in der Lage Adresskonflikten aus dem Wege zu gehen. Der Beitrag ist keine Einführung in den ESP32. Das werde ich in einem separaten Beitrag nachholen. Ich gehe also davon aus, dass ihr den ESP32 in eure Entwicklungsumgebung integriert habt.

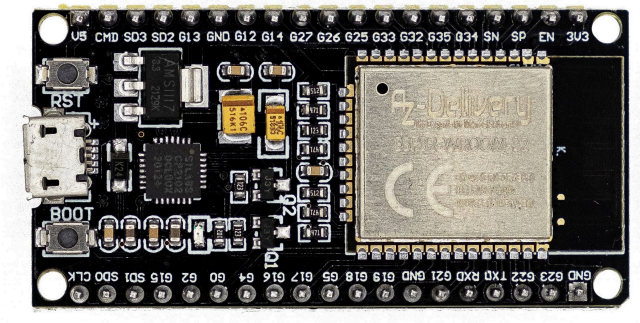
Ein weiterer Punkt, auf den ich in dem Beitrag besonders eingehe, ist die Übergabe von Objekten an Funktionen bzw. andere Objekte.

## I2C-Schnittstellen des ESP32





Es gibt eine Reihe verschiedener ESP32 Boards. Die I2C-Schnittstellen des ESP32 sind aber bei allen Ausführungen gleich organisiert. Die Standardschnittstelle befindet an den Pins 21 (SDA) und 22 (SCL). Sie kann aber auch anderen Pins zugeordnet werden. Die zweite Schnittstelle ist hinsichtlich der Pins nicht vordefiniert, d.h. ihr müsst sie festlegen bevor ihr sie nutzen könnt.



*Die I2C-Schnittstellen des ESP32 befinden sich an den Pins 21 und 22 und / oder an frei wählbaren Pins.*

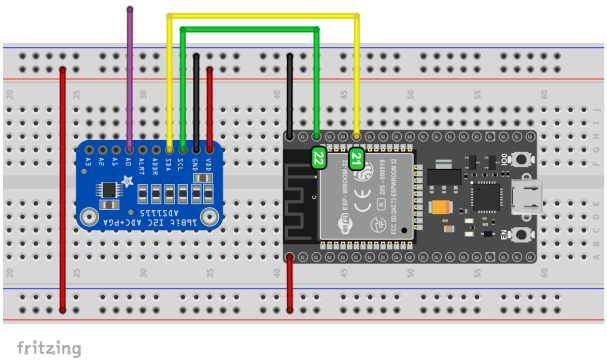
## Die Standard-I2C-Schnittstelle nutzen

Sofern ihr nur eine I2C-Schnittstelle nutzt, gibt es in der Handhabung keinen großen Unterschied zu den Arduino oder ESP8266 Boards (z.B. Wemos oder ESP-01). In den meisten Fällen werdet ihr für das angesteuerte I2C Bauteil eine Bibliothek verwenden.

Als Beispiel für einen I2C Baustein benutze ich wieder den A/D-Wandler ADS1115. Ihr müsst euch damit nicht eingehend beschäftigen, um den Beitrag zu verstehen. Nehmt einfach zur Kenntnis, dass das Teil einige einmalige Einstellungen benötigt und dann Spannungen wandelt, die am Anschluss A0 anliegen. Für die Schaltung werden keine Pull-Ups benötigt, da der ADS1115 schon welche mitbringt.








*ADS1115 an der Standard I2C-Schnittstelle des ESP32*

Ich verwende meine Bibliothek ADS1115_WE. Das ADS1115 Objekt adc wird mit `ADS1115_WE adc = ADS1115_WE (ADS1115_I2C_ADDRESS)` erzeugt. Die I2C Kommunikation wird wie gewohnt mit `Wire.begin()` intialisiert.

1_ADS1115.ino
Plain text
Copy to clipboard
Open code in new window
EnlighterJS 3 Syntax Highlighter

```
#include<ADS1115_WE.h>
#include<Wire.h>
#define ADS1115_I2C_ADDRESS 0x48
ADS1115_WE adc = ADS1115_WE(ADS1115_I2C_ADDRESS);
void setup() {
Wire.begin();
Serial.begin(9600);
setupAdc();
}
void loop() {
float voltage = 0.0;
voltage = adc.getResult_V();
Serial.print("Voltage [V]: ");
Serial.println(voltage);
Serial.println("**********************************");
delay(1000);
}
void setupAdc(){
if(!adc.init()){
Serial.print("ADS1115 not connected!");
}
adc.setVoltageRange_mV(ADS1115_RANGE_6144);
adc.setCompareChannels(ADS1115_COMP_0_GND);
adc.setMeasureMode(ADS1115_CONTINUOUS);
}
```





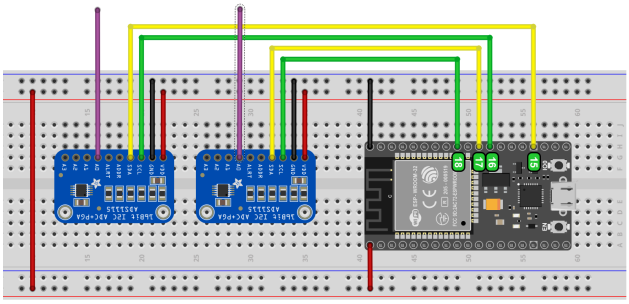
```
#include<ADS1115_WE.h> #include<Wire.h> #define ADS1115_I2C_ADDRESS 0x48
ADS1115_WE adc = ADS1115_WE(ADS1115_I2C_ADDRESS); void setup() { Wire.begin();
Serial.begin(9600); setupAdc(); } void loop() { float voltage = 0.0; voltage = adc.getResult_V();
Serial.print("Voltage [V]: "); Serial.println(voltage);
Serial.println("**********************************"); delay(1000); } void setupAdc(){
if(!adc.init()){ Serial.print("ADS1115 not connected!"); }
adc.setVoltageRange_mV(ADS1115_RANGE_6144);
adc.setCompareChannels(ADS1115_COMP_0_GND);
adc.setMeasureMode(ADS1115_CONTINUOUS); }
```

```
#include<ADS1115_WE.h>
#include<Wire.h>
#define ADS1115_I2C_ADDRESS  0x48

ADS1115_WE adc = ADS1115_WE(ADS1115_I2C_ADDRESS);

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

void loop() {
  float voltage = 0.0;

  voltage = adc.getResult_V();
  Serial.print("Voltage [V]: ");
  Serial.println(voltage);

  Serial.println("**********************************");
  delay(1000);
}

void setupAdc(){
  if(!adc.init()){
      Serial.print("ADS1115 not connected!");
    }
    adc.setVoltageRange_mV(ADS1115_RANGE_6144);
    adc.setCompareChannels(ADS1115_COMP_0_GND);
    adc.setMeasureMode(ADS1115_CONTINUOUS);
}
```

## Zwei I2C-Schnittstellen des ESP32 benutzen

### Variante 1: Zwei Schnittstellen definieren

Wenn ihr beide I2C-Schnittstellen des ESP32 nutzen wollt, dann müsst ihr euch zunächst zwei SDA und zwei SCL Pins aussuchen. Ihr könnt sie frei wählen. Meine Wahl fiel auf die Pins 15, 16, 17 und 18. Die Verdrahtung zweier ADS1115 ist wenig überraschend:




*Zwei ADS1115 an frei gewählten I2C Pins*

Für die Programmierung müsst ihr wissen, dass Wire ein Objekt der Klasse TwoWire ist. Da kümmert ihr euch normalerweise nicht drum, da das Objekt Wire mit dem Einbinden von Wire.h erzeugt wird. Im ersten Beispiel nutzen wir anstelle von Wire zwei selbsterzeugte Objekte, die wir I2C_1 und I2C_2 nennen:
Plain text
Copy to clipboard
Open code in new window
EnlighterJS 3 Syntax Highlighter
TwoWire I2C_1 = TwoWire(0);
TwoWire I2C_2 = TwoWire(1);
TwoWire I2C_1 = TwoWire(0); TwoWire I2C_2 = TwoWire(1);
```
TwoWire I2C_1 = TwoWire(0);
TwoWire I2C_2 = TwoWire(1);
```

Bevor ihr jetzt sagt: Super, dann kann ich ja nach diesem Schema munter weitere TwoWire Objekte erzeugen (`TwoWire I2C_3 = TwoWire(2)`, usw.), muss ich euch enttäuschen. Das funktioniert nicht. Nach 0 und 1 ist Schluss.

Dann müsst ihr euren Objekten I2C_1 und I2C_2 noch die SDA und SCL Pins zuordnen. Das macht ihr mit dem Aufruf der `begin()` Funktion. Optional könnt ihr die I2C Frequenz übergeben. So oder ähnlich sieht das dann aus:
Plain text
Copy to clipboard
Open code in new window
EnlighterJS 3 Syntax Highlighter
#define SDA_1 15
#define SCL_1 16
#define SDA_2 17
#define SCL_2 18
#define I2C_FREQ 400000
....
....
I2C_1.begin(SDA_1, SCL_1, I2C_FREQ);
I2C_2.begin(SDA_2, SCL_2, I2C_FREQ);
#define SDA_1 15 #define SCL_1 16 #define SDA_2 17 #define SCL_2 18 #define I2C_FREQ 400000 .... .... I2C_1.begin(SDA_1, SCL_1, I2C_FREQ); I2C_2.begin(SDA_2, SCL_2,





I2C_FREQ);
#define SDA_1 15
#define SCL_1 16
#define SDA_2 17
#define SCL_2 18
#define I2C_FREQ 400000
....
....
I2C_1.begin(SDA_1, SCL_1, I2C_FREQ);
I2C_2.begin(SDA_2, SCL_2, I2C_FREQ);

Nun müsst ihr den Objekten eures I2C Bauteils noch die I2C-Schnittstellen, also I2C_1 beziehungsweise I2C_2 zuordnen (was allerdings nicht jede Bibliothek zulässt!). Dazu übergebt ihr I2C_1 und I2C_2. Je nach Bibliothek passiert das üblicherweise bei der Initialisierung des Objektes oder mit einer `begin()` oder `init()` Funktion. Meistens wird die Vorgehensweise in Beispielsketchen zu den Bibliotheken erklärt. Bei der ADS1115_WE Bibliothek sieht die Übergabe so aus:
Plain text
Copy to clipboard
Open code in new window
EnlighterJS 3 Syntax Highlighter
ADS1115_WE adc_1 = ADS1115_WE(&I2C_1, ADS1115_I2C_ADDRESS);
ADS1115_WE adc_2 = ADS1115_WE(&I2C_2, ADS1115_I2C_ADDRESS);
ADS1115_WE adc_1 = ADS1115_WE(&I2C_1, ADS1115_I2C_ADDRESS); ADS1115_WE adc_2 = ADS1115_WE(&I2C_2, ADS1115_I2C_ADDRESS);
```
ADS1115_WE adc_1 = ADS1115_WE(&I2C_1, ADS1115_I2C_ADDRESS);
ADS1115_WE adc_2 = ADS1115_WE(&I2C_2, ADS1115_I2C_ADDRESS);
```

Ein TwoWire Objekt hat eine gewisse Größe. Um Speicherplatz zu sparen, arbeiten die meisten Bibliotheken deshalb nicht mit lokalen Kopien der übergebenen Objekte, sondern mit den Objekten selbst. Dazu wird das TwoWire Objekt als Zeiger übergeben, d.h. die empfangende Funktion benutzt Zeiger als Parameter. Beim Funktionsaufruf muss dann I2C_1 und I2C_2 der Adressoperator „&“ vorangestellt werden. Falls ihr damit keine Erfahrung habt, ist das sicherlich zunächst verwirrend. Weiter unten komme ich darauf noch einmal zurück.

Und so sieht dann der ganze Sketch aus:
2_ADS1115.ino
Plain text
Copy to clipboard
Open code in new window
EnlighterJS 3 Syntax Highlighter
#include<ADS1115_WE.h>
#include<Wire.h>
#define ADS1115_I2C_ADDRESS 0x48
#define I2C_FREQ 400000
#define SDA_1 15
#define SCL_1 16
#define SDA_2 17
#define SCL_2 18
TwoWire I2C_1 = TwoWire(0);
TwoWire I2C_2 = TwoWire(1);
ADS1115_WE adc_1 = ADS1115_WE(&I2C_1, ADS1115_I2C_ADDRESS);
ADS1115_WE adc_2 = ADS1115_WE(&I2C_2, ADS1115_I2C_ADDRESS);
void setup() {
Serial.begin(9600);
I2C_1.begin(SDA_1, SCL_1, I2C_FREQ);
I2C_2.begin(SDA_2, SCL_2, I2C_FREQ);
setupAdc_1();
setupAdc_2();
}





void loop() {
float voltage = 0.0;
voltage = adc_1.getResult_V();
Serial.print("Voltage [V], ADS1115 No 1: ");
Serial.println(voltage);
voltage = adc_2.getResult_V();
Serial.print("Voltage [V], ADS1115 No 2: ");
Serial.println(voltage);
Serial.println("**********");
delay(1000);
}
void setupAdc_1(){
if(!adc_1.init()){
Serial.println("ADS1115 No 1 not connected!");
}
adc_1.setVoltageRange_mV(ADS1115_RANGE_6144);
adc_1.setCompareChannels(ADS1115_COMP_0_GND);
adc_1.setMeasureMode(ADS1115_CONTINUOUS);
}
void setupAdc_2(){
if(!adc_2.init()){
Serial.println("ADS1115 No 2 not connected!");
}
adc_2.setVoltageRange_mV(ADS1115_RANGE_6144);
adc_2.setCompareChannels(ADS1115_COMP_0_GND);
adc_2.setMeasureMode(ADS1115_CONTINUOUS);
}
#include<ADS1115_WE.h> #include<Wire.h> #define ADS1115_I2C_ADDRESS 0x48 #define I2C_FREQ 400000 #define SDA_1 15 #define SCL_1 16 #define SDA_2 17 #define SCL_2 18 TwoWire I2C_1 = TwoWire(0); TwoWire I2C_2 = TwoWire(1); ADS1115_WE adc_1 = ADS1115_WE(&I2C_1, ADS1115_I2C_ADDRESS); ADS1115_WE adc_2 = ADS1115_WE(&I2C_2, ADS1115_I2C_ADDRESS); void setup() { Serial.begin(9600); I2C_1.begin(SDA_1, SCL_1, I2C_FREQ); I2C_2.begin(SDA_2, SCL_2, I2C_FREQ); setupAdc_1(); setupAdc_2(); } void loop() { float voltage = 0.0; voltage = adc_1.getResult_V(); Serial.print("Voltage [V], ADS1115 No 1: "); Serial.println(voltage); voltage = adc_2.getResult_V(); Serial.print("Voltage [V], ADS1115 No 2: "); Serial.println(voltage); Serial.println("**********"); delay(1000); } void setupAdc_1(){ if(!adc_1.init()){ Serial.println("ADS1115 No 1 not connected!"); } adc_1.setVoltageRange_mV(ADS1115_RANGE_6144); adc_1.setCompareChannels(ADS1115_COMP_0_GND); adc_1.setMeasureMode(ADS1115_CONTINUOUS); } void setupAdc_2(){ if(!adc_2.init()){ Serial.println("ADS1115 No 2 not connected!"); } adc_2.setVoltageRange_mV(ADS1115_RANGE_6144); adc_2.setCompareChannels(ADS1115_COMP_0_GND); adc_2.setMeasureMode(ADS1115_CONTINUOUS); }
```
#include<ADS1115_WE.h>
#include<Wire.h>
#define ADS1115_I2C_ADDRESS  0x48
#define I2C_FREQ 400000

#define SDA_1 15
#define SCL_1 16
#define SDA_2 17
#define SCL_2 18

TwoWire I2C_1 = TwoWire(0);
TwoWire I2C_2 = TwoWire(1);

ADS1115_WE adc_1 = ADS1115_WE(&I2C_1, ADS1115_I2C_ADDRESS);
ADS1115_WE adc_2 = ADS1115_WE(&I2C_2, ADS1115_I2C_ADDRESS);
```





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

  I2C_1.begin(SDA_1, SCL_1, I2C_FREQ);
  I2C_2.begin(SDA_2, SCL_2, I2C_FREQ);

  setupAdc_1();
  setupAdc_2();
}

void loop() {
  float voltage = 0.0;

  voltage = adc_1.getResult_V();
  Serial.print("Voltage [V], ADS1115 No 1: ");
  Serial.println(voltage);

  voltage = adc_2.getResult_V();
  Serial.print("Voltage [V], ADS1115 No 2: ");
  Serial.println(voltage);

  Serial.println("**********");
  delay(1000);
}

void setupAdc_1(){
  if(!adc_1.init()){
    Serial.println("ADS1115 No 1 not connected!");
  }
  adc_1.setVoltageRange_mV(ADS1115_RANGE_6144);
  adc_1.setCompareChannels(ADS1115_COMP_0_GND);
  adc_1.setMeasureMode(ADS1115_CONTINUOUS);
}

void setupAdc_2(){
  if(!adc_2.init()){
    Serial.println("ADS1115 No 2 not connected!");
  }
  adc_2.setVoltageRange_mV(ADS1115_RANGE_6144);
  adc_2.setCompareChannels(ADS1115_COMP_0_GND);
  adc_2.setMeasureMode(ADS1115_CONTINUOUS);
}
```

Der Code lässt sich noch kürzen, da sich in Bezug auf die Objekte adc_1 und adc_2 einiges wiederholt. Also führe ich Funktionen ein, denen ich diese Objekte übergebe. Auch hier arbeite ich nicht mit lokalen Kopien der Objekte, sondern mit den Originalen. Allerdings wähle hier eine andere Methode, nämlich die Übergabe mit Referenzen als Parametern. Dem Funktionsaufruf, z.B.:
Plain text
Copy to clipboard
Open code in new window
EnlighterJS 3 Syntax Highlighter
setupAdc(adc_1, 1);
setupAdc(adc_1, 1);
```
setupAdc(adc_1, 1);
```

sieht man nichts von der Referenz an. In der Funktion selbst taucht dann aber wieder der Adressoperator auf:
Plain text
Copy to clipboard
Open code in new window
EnlighterJS 3 Syntax Highlighter
void setupAdc(ADS1115_WE &adc, byte i)
void setupAdc(ADS1115_WE &adc, byte i)
void setupAdc(ADS1115_WE &adc, byte i)

Dadurch wird innerhalb der Funktion lediglich ein anderer Bezeichner für das Objekt eingeführt.

2_ADS1115_short.ino

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

```
#include<ADS1115_WE.h>
#include<Wire.h>
#define ADS1115_I2C_ADDRESS 0x48
#define I2C_FREQ 400000
#define SDA_1 15
#define SCL_1 16
#define SDA_2 17
#define SCL_2 18
TwoWire I2C_1 = TwoWire(0);
TwoWire I2C_2 = TwoWire(1);
ADS1115_WE adc_1 = ADS1115_WE(&I2C_1, ADS1115_I2C_ADDRESS);
ADS1115_WE adc_2 = ADS1115_WE(&I2C_2, ADS1115_I2C_ADDRESS);
void setup() {
Serial.begin(9600);
I2C_1.begin(SDA_1, SCL_1, I2C_FREQ);
I2C_2.begin(SDA_2, SCL_2, I2C_FREQ);
setupAdc(adc_1, 1);
setupAdc(adc_2, 2);
}
void loop() {
queryAdc(adc_1, 1);
queryAdc(adc_2, 2);
Serial.println("**********************************");
delay(1000);
}
void setupAdc(ADS1115_WE &adc, byte i){
if(!adc.init()){
Serial.print("ADS1115 No ");
Serial.print(i);
Serial.println(" not connected!");
}
adc.setVoltageRange_mV(ADS1115_RANGE_6144);
adc.setCompareChannels(ADS1115_COMP_0_GND);
adc.setMeasureMode(ADS1115_CONTINUOUS);
}
void queryAdc(ADS1115_WE &adc, byte i){
float voltage = 0.0;
voltage = adc.getResult_V();
Serial.print("Voltage [V], ADS1115 No ");
Serial.print(i);
Serial.print(": ");
Serial.println(voltage);
}
```

#include<ADS1115_WE.h> #include<Wire.h> #define ADS1115_I2C_ADDRESS 0x48 #define I2C_FREQ 400000 #define SDA_1 15 #define SCL_1 16 #define SDA_2 17 #define SCL_2 18 TwoWire I2C_1 = TwoWire(0); TwoWire I2C_2 = TwoWire(1); ADS1115_WE adc_1 = ADS1115_WE(&I2C_1, ADS1115_I2C_ADDRESS); ADS1115_WE adc_2 = ADS1115_WE(&I2C_2, ADS1115_I2C_ADDRESS); void setup() { Serial.begin(9600); I2C_1.begin(SDA_1, SCL_1, I2C_FREQ); I2C_2.begin(SDA_2, SCL_2, I2C_FREQ); setupAdc(adc_1, 1); setupAdc(adc_2, 2); } void loop() { queryAdc(adc_1, 1); queryAdc(adc_2, 2); Serial.println("**********************************"); delay(1000); } void setupAdc(ADS1115_WE &adc, byte i){ if(!adc.init()){ Serial.print("ADS1115 No ");





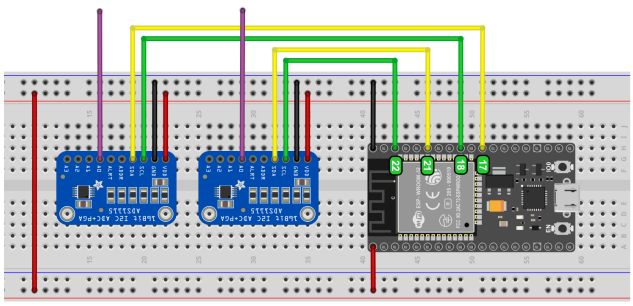
```
Serial.print(i); Serial.println(" not connected!"); }
adc.setVoltageRange_mV(ADS1115_RANGE_6144);
adc.setCompareChannels(ADS1115_COMP_0_GND);
adc.setMeasureMode(ADS1115_CONTINUOUS); } void queryAdc(ADS1115_WE &adc, byte i){
float voltage = 0.0; voltage = adc.getResult_V(); Serial.print("Voltage [V], ADS1115 No ");
Serial.print(i); Serial.print(": "); Serial.println(voltage); }
#include<ADS1115_WE.h>
#include<Wire.h>
#define ADS1115_I2C_ADDRESS  0x48
#define I2C_FREQ 400000

#define SDA_1 15
#define SCL_1 16
#define SDA_2 17
#define SCL_2 18

TwoWire I2C_1 = TwoWire(0);
TwoWire I2C_2 = TwoWire(1);
ADS1115_WE adc_1 = ADS1115_WE(&I2C_1, ADS1115_I2C_ADDRESS);
ADS1115_WE adc_2 = ADS1115_WE(&I2C_2, ADS1115_I2C_ADDRESS);

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

  I2C_1.begin(SDA_1, SCL_1, I2C_FREQ);
  I2C_2.begin(SDA_2, SCL_2, I2C_FREQ);

  setupAdc(adc_1, 1);
  setupAdc(adc_2, 2);
}

void loop() {
  queryAdc(adc_1, 1);
  queryAdc(adc_2, 2);

  Serial.println("**********************************");
  delay(1000);
}

void setupAdc(ADS1115_WE &adc, byte i){
  if(!adc.init()){
    Serial.print("ADS1115 No ");
    Serial.print(i);
    Serial.println(" not connected!");
  }
  adc.setVoltageRange_mV(ADS1115_RANGE_6144);
  adc.setCompareChannels(ADS1115_COMP_0_GND);
  adc.setMeasureMode(ADS1115_CONTINUOUS);
}

void queryAdc(ADS1115_WE &adc, byte i){
  float voltage = 0.0;

  voltage = adc.getResult_V();
  Serial.print("Voltage [V], ADS1115 No ");
  Serial.print(i);
  Serial.print(": ");
  Serial.println(voltage);
}
```

## Variante 2: Wire und eine zusätzliche Schnittstelle





Nicht weil es Vorteile brächte, aber der Vollständigkeit halber möchte ich noch zeigen, dass ihr natürlich auch das vordefinierte Wire Objekt in Verbindung mit einem zusätzlich kreiertem TwoWire Objekt benutzen könnt. Gegenüber der letzten Schaltung ändern sich nur die I2C Pins:




*Zwei ADS1115 an Wire und einer zuätzlichen I2C Schnittstelle*

Das zusätzliche TwoWire Objekt habe ich Wire1 genannt. Es muss mit `TwoWire(1)` erzeugt werden. An einigen Stellen habe ich gelesen, dass auch Wire1 genau wie Wire vordefiniert ist. Ohne die Zeile `TwoWire Wire1 = TwoWire(1);` funktionierte es bei mir aber nicht.

So sieht dann der Sketch dazu aus:

2_ADS1115_wire_wire1.ino

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

```
#include<ADS1115_WE.h>
#include<Wire.h>
#define ADS1115_I2C_ADDRESS 0x48
#define I2C_FREQ 400000
#define SDA_2 17
#define SCL_2 18
TwoWire Wire1 = TwoWire(1);
ADS1115_WE adc_1 = ADS1115_WE(ADS1115_I2C_ADDRESS);
ADS1115_WE adc_2 = ADS1115_WE(&Wire1, ADS1115_I2C_ADDRESS);
void setup() {
Serial.begin(9600);
Wire.begin();
Wire1.begin(SDA_2, SCL_2, I2C_FREQ);
setupAdc(adc_1, 1);
setupAdc(adc_2, 2);
}
void loop() {
queryAdc(adc_1, 1);
queryAdc(adc_2, 2);
Serial.println("**********************************");
delay(1000);
}
```





```
void setupAdc(ADS1115_WE &adc, byte i){
if(!adc.init()){
Serial.print("ADS1115 No ");
Serial.print(i);
Serial.println(" not connected!");
}
adc.setVoltageRange_mV(ADS1115_RANGE_6144);
adc.setCompareChannels(ADS1115_COMP_0_GND);
adc.setMeasureMode(ADS1115_CONTINUOUS);
}
void queryAdc(ADS1115_WE &adc, byte i){
float voltage = 0.0;
voltage = adc.getResult_V();
Serial.print("Voltage [V], ADS1115 No ");
Serial.print(i);
Serial.print(": ");
Serial.println(voltage);
}
```

#include<ADS1115_WE.h> #include<Wire.h> #define ADS1115_I2C_ADDRESS 0x48 #define I2C_FREQ 400000 #define SDA_2 17 #define SCL_2 18 TwoWire Wire1 = TwoWire(1); ADS1115_WE adc_1 = ADS1115_WE(ADS1115_I2C_ADDRESS); ADS1115_WE adc_2 = ADS1115_WE(&Wire1, ADS1115_I2C_ADDRESS); void setup() { Serial.begin(9600); Wire.begin(); Wire1.begin(SDA_2, SCL_2, I2C_FREQ); setupAdc(adc_1, 1); setupAdc(adc_2, 2); } void loop() { queryAdc(adc_1, 1); queryAdc(adc_2, 2); Serial.println("**********************************"); delay(1000); } void setupAdc(ADS1115_WE &adc, byte i){ if(!adc.init()){ Serial.print("ADS1115 No ");

```
Serial.print(i); Serial.println(" not connected!"); }
adc.setVoltageRange_mV(ADS1115_RANGE_6144);
adc.setCompareChannels(ADS1115_COMP_0_GND);
adc.setMeasureMode(ADS1115_CONTINUOUS); } void queryAdc(ADS1115_WE &adc, byte i){
float voltage = 0.0; voltage = adc.getResult_V(); Serial.print("Voltage [V], ADS1115 No ");
Serial.print(i); Serial.print(": "); Serial.println(voltage); }
#include<ADS1115_WE.h>
#include<Wire.h>
#define ADS1115_I2C_ADDRESS  0x48
#define I2C_FREQ 400000
#define SDA_2 17
#define SCL_2 18

TwoWire Wire1 = TwoWire(1);

ADS1115_WE adc_1 = ADS1115_WE(ADS1115_I2C_ADDRESS);
ADS1115_WE adc_2 = ADS1115_WE(&Wire1, ADS1115_I2C_ADDRESS);

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

  Wire.begin();
  Wire1.begin(SDA_2, SCL_2, I2C_FREQ);

  setupAdc(adc_1, 1);
  setupAdc(adc_2, 2);
}

void loop() {
  queryAdc(adc_1, 1);
  queryAdc(adc_2, 2);
  Serial.println("**********************************");
  delay(1000);
}

void setupAdc(ADS1115_WE &adc, byte i){
  if(!adc.init()){
```

```
        Serial.print("ADS1115 No ");
        Serial.print(i);
        Serial.println(" not connected!");
      }
      adc.setVoltageRange_mV(ADS1115_RANGE_6144);
      adc.setCompareChannels(ADS1115_COMP_0_GND);
      adc.setMeasureMode(ADS1115_CONTINUOUS);
}

void queryAdc(ADS1115_WE &adc, byte i){
  float voltage = 0.0;

  voltage = adc.getResult_V();
  Serial.print("Voltage [V], ADS1115 No ");
  Serial.print(i);
  Serial.print(": ");
  Serial.println(voltage);
}
```

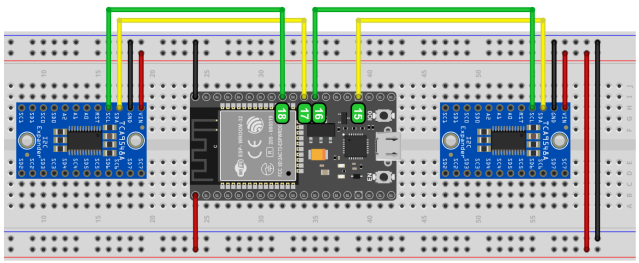
## I2C-Schnittstellen des ESP32 ohne Bibliotheken

In den bisherigen Beispielen war das I2C Bauteil über eine Bibliothek als Objekt definiert. Wenn ihr ohne Bibliothek arbeitet, sind die Dinge einfacher. Ich gehe am Beispiel des I2C Multiplexers TCA9548A trotzdem noch einmal darauf ein, da ich in dem Zuge die Übergabe von Zeigern näher erläutern kann.

Den TCA9548A habe ich ausgewählt, weil ihr damit noch weitere I2C-Bauteile mit gleicher Adresse ansteuern könnt. Wir bleiben also beim Thema. Darüber hinaus ist es hinsichtlich der Ansteuerung das einfachste Bauteil, dass ich finden konnte.

Da ich den TCA9548A in meinem letzten Beitrag behandelt habe, beschreibe ich ihn hier nur ganz grob: Der TCA9548A hat einen I2C Eingang und acht I2C Ausgänge. Die Kanäle werden geöffnet, indem das für den jeweiligen Kanal zuständige Bit im Kontrollregister gesetzt wird. Auf diese Weise können mit einem TCA9548A acht I2C Bauteile mit gleicher Adresse angesprochen werden.

So sieht kann es aussehen, wenn ihr zwei TCA9548A an den ESP32 anschließt:




*Zwei TCA9548A am ESP32*

Der folgende Sketch öffnet Kanal 3 des einen und Kanal 7 des anderen Moduls:
Plain text





Copy to clipboard
Open code in new window
EnlighterJS 3 Syntax Highlighter

```
#include<Wire.h>
#define TCA_I2C_ADDRESS 0x70
#define TCA_1_CHANNEL 3
#define TCA_2_CHANNEL 7
#define I2C_FREQ_1 400000
#define I2C_FREQ_2 400000
#define SDA_1 15
#define SCL_1 16
#define SDA_2 17
#define SCL_2 18
TwoWire I2C_1 = TwoWire(0);
TwoWire I2C_2 = TwoWire(1);
void setup() {
I2C_1.begin(SDA_1, SCL_1, I2C_FREQ_1);
I2C_2.begin(SDA_2, SCL_2, I2C_FREQ_2);
setTCAChannel_1(TCA_1_CHANNEL);
setTCAChannel_2(TCA_2_CHANNEL);
}
void loop() {
}
void setTCAChannel_1(byte i){
I2C_1.beginTransmission(TCA_I2C_ADDRESS);
I2C_1.write(1 << i);
I2C_1.endTransmission();
}
void setTCAChannel_2(byte i){
I2C_2.beginTransmission(TCA_I2C_ADDRESS);
I2C_2.write(1 << i);
I2C_2.endTransmission();
}
```

#include<Wire.h> #define TCA_I2C_ADDRESS 0x70 #define TCA_1_CHANNEL 3 #define TCA_2_CHANNEL 7 #define I2C_FREQ_1 400000 #define I2C_FREQ_2 400000 #define SDA_1 15 #define SCL_1 16 #define SDA_2 17 #define SCL_2 18 TwoWire I2C_1 = TwoWire(0); TwoWire I2C_2 = TwoWire(1); void setup() { I2C_1.begin(SDA_1, SCL_1, I2C_FREQ_1); I2C_2.begin(SDA_2, SCL_2, I2C_FREQ_2); setTCAChannel_1(TCA_1_CHANNEL); setTCAChannel_2(TCA_2_CHANNEL); } void loop() { } void setTCAChannel_1(byte i){ I2C_1.beginTransmission(TCA_I2C_ADDRESS); I2C_1.write(1 << i); I2C_1.endTransmission(); } void setTCAChannel_2(byte i){ I2C_2.beginTransmission(TCA_I2C_ADDRESS); I2C_2.write(1 << i); I2C_2.endTransmission(); }

```
#include<Wire.h>
#define TCA_I2C_ADDRESS  0x70
#define TCA_1_CHANNEL  3
#define TCA_2_CHANNEL  7
#define I2C_FREQ_1 400000
#define I2C_FREQ_2 400000

#define SDA_1 15
#define SCL_1 16
#define SDA_2 17
#define SCL_2 18

TwoWire I2C_1 = TwoWire(0);
TwoWire I2C_2 = TwoWire(1);

void setup() {
  I2C_1.begin(SDA_1, SCL_1, I2C_FREQ_1);
  I2C_2.begin(SDA_2, SCL_2, I2C_FREQ_2);
```





```
  setTCAChannel_1(TCA_1_CHANNEL);
  setTCAChannel_2(TCA_2_CHANNEL);
}

void loop() {
}

void setTCAChannel_1(byte i){
  I2C_1.beginTransmission(TCA_I2C_ADDRESS);
  I2C_1.write(1 << i);
  I2C_1.endTransmission();
}

void setTCAChannel_2(byte i){
  I2C_2.beginTransmission(TCA_I2C_ADDRESS);
  I2C_2.write(1 << i);
  I2C_2.endTransmission();
}
```

Die beiden setTCAChannel() Funktionen lassen sich zusammenfassen, indem man das TwoWire Objekt als Referenz übergibt:
Plain text
Copy to clipboard
Open code in new window
EnlighterJS 3 Syntax Highlighter

```
void setTCAChannel(byte i, TwoWire &I2C){
I2C.beginTransmission(TCA_I2C_ADDRESS);
I2C.write(1 << i);
I2C.endTransmission();
}
```

void setTCAChannel(byte i, TwoWire &I2C){ I2C.beginTransmission(TCA_I2C_ADDRESS); I2C.write(1 << i); I2C.endTransmission(); }

```
void setTCAChannel(byte i, TwoWire &I2C){
  I2C.beginTransmission(TCA_I2C_ADDRESS);
  I2C.write(1 << i);
  I2C.endTransmission();
}
```

**Übergabe von Zeigern**

Es gibt aber noch eine andere Methode, nämlich die Übergabe von Zeigern. Das sieht bei der empfangenden Funktion folgermaßen aus:
Plain text
Copy to clipboard
Open code in new window
EnlighterJS 3 Syntax Highlighter

```
void setTCAChannel(byte i, TwoWire *I2C){
TwoWire *wire = I2C;
wire->beginTransmission(TCA_I2C_ADDRESS);
wire->write(1 << i);
wire->endTransmission();
}
```

void setTCAChannel(byte i, TwoWire *I2C){ TwoWire *wire = I2C; wire->beginTransmission(TCA_I2C_ADDRESS); wire->write(1 << i); wire->endTransmission(); }

```
void setTCAChannel(byte i, TwoWire *I2C){
  TwoWire *wire = I2C;
  wire->beginTransmission(TCA_I2C_ADDRESS);
  wire->write(1 << i);
  wire->endTransmission();
}
```





Die Übergabe des Objektes als Zeiger hat zwei Konsequenzen:

1. Auf Zeiger können keine Punktoperatoren angewendet werden. Wollt ihr eine Funktion des Objektes verwenden, auf die der Zeiger zeigt, dann ist die Entsprechung der Pfeiloperator.
2. Beim Aufruf der obigen Funktion muss dem zu übergebenden Objekt der Adressoperator vorangestellt werden.

Und warum erzähle ich das alles? Weil die Übergabe des TwoWire Objektes bei den meisten Bibliotheken so funktioniert und ich das an so einem einfachen Beispiel am besten zeigen kann.

Der vollständige Sketch sieht dann so aus:
Plain text
Copy to clipboard
Open code in new window
EnlighterJS 3 Syntax Highlighter

```
#include<Wire.h>
#define TCA_I2C_ADDRESS 0x70
#define TCA_1_CHANNEL 3
#define TCA_2_CHANNEL 7
#define I2C_FREQ_1 400000
#define I2C_FREQ_2 400000
#define SDA_1 15
#define SCL_1 16
#define SDA_2 17
#define SCL_2 18
TwoWire I2C_1 = TwoWire(0);
TwoWire I2C_2 = TwoWire(1);
void setup() {
I2C_1.begin(SDA_1, SCL_1, I2C_FREQ_1);
I2C_2.begin(SDA_2, SCL_2, I2C_FREQ_2);
setTCAChannel(TCA_1_CHANNEL, &I2C_1);
setTCAChannel(TCA_2_CHANNEL, &I2C_2);
}
void loop() {
}
void setTCAChannel(byte i, TwoWire *I2C){
TwoWire *wire = I2C;
wire->beginTransmission(TCA_I2C_ADDRESS);
wire->write(1 << i);
wire->endTransmission();
}
```

#include<Wire.h> #define TCA_I2C_ADDRESS 0x70 #define TCA_1_CHANNEL 3 #define TCA_2_CHANNEL 7 #define I2C_FREQ_1 400000 #define I2C_FREQ_2 400000 #define SDA_1 15 #define SCL_1 16 #define SDA_2 17 #define SCL_2 18 TwoWire I2C_1 = TwoWire(0); TwoWire I2C_2 = TwoWire(1); void setup() { I2C_1.begin(SDA_1, SCL_1, I2C_FREQ_1); I2C_2.begin(SDA_2, SCL_2, I2C_FREQ_2); setTCAChannel(TCA_1_CHANNEL, &I2C_1); setTCAChannel(TCA_2_CHANNEL, &I2C_2); } void loop() { } void setTCAChannel(byte i, TwoWire *I2C){ TwoWire *wire = I2C; wire->beginTransmission(TCA_I2C_ADDRESS); wire->write(1 << i); wire->endTransmission(); }

```
#include<Wire.h>
#define TCA_I2C_ADDRESS  0x70
#define TCA_1_CHANNEL  3
#define TCA_2_CHANNEL  7
#define I2C_FREQ_1 400000
#define I2C_FREQ_2 400000

#define SDA_1 15
#define SCL_1 16
```

```
#define SDA_2 17
#define SCL_2 18

TwoWire I2C_1 = TwoWire(0);
TwoWire I2C_2 = TwoWire(1);

void setup() {
  I2C_1.begin(SDA_1, SCL_1, I2C_FREQ_1);
  I2C_2.begin(SDA_2, SCL_2, I2C_FREQ_2);
  setTCAChannel(TCA_1_CHANNEL, &I2C_1);
  setTCAChannel(TCA_2_CHANNEL, &I2C_2);
}

void loop() {
}

void setTCAChannel(byte i, TwoWire *I2C){
  TwoWire *wire = I2C;
  wire->beginTransmission(TCA_I2C_ADDRESS);
  wire->write(1 << i);
  wire->endTransmission();
}
```

## Update meiner Bibliotheken

Mittlerweile habe ich eine ganze Reihe von Bibliotheken auf GitHub veröffentlicht. Bis auf eine Ausnahme habe ich sie in den letzten Wochen überarbeitet, sodass sie die Übergabe von TwoWire Objekten optional zulassen. Ihr findet die Bibliotheken hier.

## Danksagung

Das Fritzing Bauteil für den ESP32 habe ich hier im Fritzing Forum gefunden. Den Designern vielen Dank.