

# Android App - LED Color Control - ESP32

Marco Bräuer

06.12.2021

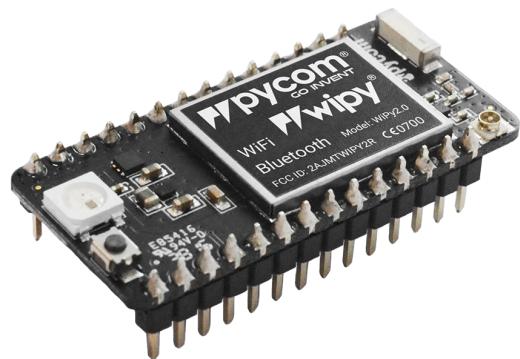


Abbildung 1: ESP-32 Mikrocontroller

# **Inhaltsverzeichnis**

<b>1</b>	<b>Analyse</b>	<b>3</b>
1.1	Anforderung an das Programm . . . . .	3
1.2	Anwendungsfälle . . . . .	3
1.3	Programmschnittstellen . . . . .	3
<b>2</b>	<b>Entwurf</b>	<b>4</b>
2.1	Zielplattform . . . . .	4
2.2	Entwurf der Benutzeroberflächen . . . . .	4
<b>3</b>	<b>Implementierung</b>	<b>5</b>
3.1	ESP32 . . . . .	5
3.2	Delphi . . . . .	5
3.2.1	Der Code . . . . .	6
3.2.2	Probleme . . . . .	7
<b>4</b>	<b>Anhang</b>	<b>8</b>

# **1 Analyse**

## **1.1 Anforderung an das Programm**

Die Anwendung soll auf einem Android Smartphone ausgeführt werden. In der App soll eine Farben über ein Bedienfeld ausgewählt und über den EP32- Mikrocontroller an den LED-Streifen übertragen werden.

Außerdem soll beim Start die letzte ausgewählte Farbe eingestellt und beim ausschalten die Farbe gespeichert werden.

## **1.2 Anwendungsfälle**

In einem Raum soll eine LED Beleuchtung durch die App gesteuert werden. Vorausgesetzt wird eine Internetverbindung in der sich die Geräte befinden.

## **1.3 Programmschnittstellen**

Der ESP32 soll als WLAN Webserver HTTP GET Anfragen von der, im gleichen Netzwerk befindlichen Android Applikation verarbeiten und an den angeschlossenen LED-Streifen weiterleiten.

## 2 Entwurf

### 2.1 Zielplattform

Die Zielplattform ist *Android* ab Version 6. Kompiliert wird mit der *Delphi 10.4.2 Community Edition*. Als Webserver wird ein *ESP32* Mikrocontroller mit der *Arduino IDE* in Wireing beschrieben.  
(C++ Derivat)

### 2.2 Entwurf der Benutzeroberflächen

Die Benutzeroberfläche soll über einen *RGB-Picker* und drei einfachen Buttons erfolgen.

*Siehe Abbildung 2 : Entwurf der Benutzeroberfläche*

Damit sich das Programm die Einstellung merken kann wird eine *.ini* Datei geschrieben.

## 3 Implementierung

### 3.1 ESP32

Nach der Installation der **Arduino IDE** habe ich die Bibliothek für den *ESP32* installiert. Zum testen habe ich ein leeres Programm kompiliert und auf den *ESP32* übertragen. Dabei war es notwendig bei dem *connecting...* den bootknopf am *ESP32* gedrückt zu halten.

Der Umgang mit analogen Ausgängen beim *ESP32* erschien schwierig und im Unterricht wurde nur der *Arduino Uno* behandelt. Daher habe ich eine erste Testschaltung mit 3 einzelnen Dioden, die jeweils Rot, Grün und Blau repräsentieren, am *Arduino Uno* durchgeführt.

*Siehe Abbildung 3 : Arduino Test LED Aufbau*

Danach habe ich den Aufbau mit einem kurzen, angelöteten LED-Segment getestet.

*Siehe Abbildung 4 : Arduino Aufbau*

Danach habe ich auf dem *ESP32* nach einer Anleitung von *randomnerdtutorial.com* und *techtutorialsx.com* einen Webserver aufgesetzt und mir genauer angeschaut, was welcher Befehl macht und was ich davon brauche. Beim *ESP32* habe ich dann, wie beim *Arduino Uno* eine Testumgebung mit 3 Dioden durchgeführt. Diese 3 Dioden habe ich an 3 Pins angeschlossen und mittels einer gesetzten Frequenz von 5Khz und einer Tiefe von 8 Bit (256 Farbwerte) mittels der LED Methoden des *ESP32* auf analog gesetzt.

Anschließend habe ich mir eine Methode geschrieben, die 3 integer Werte von 0 bis 255 als "r g b" Werte über den jeweiligen Kanal mit dem http request an die Led schreibt.

Danach habe ich meinen kurzen LED- Streifen angeschlossen und getestet mit den Befehlen über den Browser.

*Siehe Abbildung 5 bis 10*

### 3.2 Delphi

Nachdem ich über den HTTP Request mit der GET Methode die 3 Parameter für rot grün und blau Werte übergeben konnte und der LED-Streifen daraufhin seine Farbe ändern konnte, habe ich mir die Programmierung der Android App vorgenommen.

Dazu verwende ich die Delphi Community Edition mit installierten Android SDK und NDK. Mein Telefon wurde aber erst von der IDE erkannt, nachdem ich die Google USB Driver und die Samsung Driver nachinstalliert hatte.

*Siehe Abbildung 15 bis 17*

Außerdem musste ich die den Pfad zur keytool.exe ändern, damit das richtige Java gefunden werden konnte.

*Siehe Abbildung 18: Das richtige Java verwenden*

Danach habe ich eine simple Oberfläche erstellt. *Siehe Abbildung 10: Design Struktur*

Auf der Oberfläche liegt eine HTTP Komponente, über deren Instanz ich die *HTTP-Get* Anfrage abschicke. In den Einstellungen habe ich die Methode auf plain/text geändert. Die Browserverversion habe ich auf Mozilla 5.0 angepasst, da sonst die Anfragen nicht mehr verarbeitet werden können. (mit der Standard Einstellung auf Version 3.0)

Die gesamten Elemente habe ich in ein ScrollPane gelegt, dass auf verschiedenen großen Geräten bei Bedarf nach unten gescrollt werden kann und alle Elemente erreicht werden können. Als Layer verwende ich rectangle der Klasse TRectangle, da diese sehr flexibel in der Anpassung sind. Ich hätte auch in die Rectangle in denen meine Buttons liegen OnClick Events verwenden können und mir somit die Buttons sparen können. Ich habe mich aber für die Buttons entschieden, da es eine Standard animation beim Click gibt. Da die Buttons keine Möglichkeit der Farbanpassung besitzen habe ich die Background Color auf visible false über das Stil Menü für alle Button Instanzen gesetzt.

Es gibt insgesamt 3 Buttons. Einen **ON**, einen **OFF** und einen **OK** Button, zum an- und ausschalten des Led-Streifens und zum senden der eingestellten Farbe.

### 3.2.1 Der Code

Zum Auswählen der Farbe verwende ich einen ColorPanel, eine vorgefertigte Komponente aus Delphi. Die Veränderung der Werte Frage ich im OnChange Event des Panels ab. Diese Werte lasse ich mir in Labels ausgeben. einmal den Hexadezimalwert der Farbe und die 3 zugehörigen Dezimalwerte für Rot, Grün und Blau. Das Umrechnen und Ausgeben der Werte erfolgt im Event selbst.

*Siehe Abbildung 11: Das Color Change Event*

Die 3 Farbwerte speichere ich mir in globalen Variablen, denn das sind die Werte, welche ich als Parameter mit der Get Anfrage sende.

Der HTTP-Request liegt dann in den 3 ButtonClick Events.

Beim Button **OFF** sende ich ein GET-Request mit jeweils 0 als Parameter, welches den Strom sozusagen auf 0 Volt an allen 3 Kanälen, bzw. zugeordneten Pins setzt. Beim Button **OK** verschicke ich die zwischengespeicherten Werte aus der Auswahl als Parameter und jede Farbe bekommt einen Wert zwischen 0 und 255, also keiner Spannung und voller Spannung.

Der **ON** Button und das Abschalten stellen noch ein Problem dar. Wie speichere ich mir die zuletzt eingestellte Farbe?

Als Lösung habe ich mich für eine ini- Datei entschieden, die ich beim kompilieren der apk. Datei mitgebe und die nach jedem An-und Abschalten überschreiben wird. Sie Enthält einen Knoten : [rgb] und die 3 Werte für den HTTP Request.

*Siehe Abbildung 12 bis 14*

### **3.2.2 Probleme**

Probleme traten auf, wenn ich zum Beispiel beim kompilieren die Verbindung zum Android Gerät verloren oder unterbrochen habe. Danach kommt es beim komplizieren zu Signatur Fehlermeldungen in der IDE. Das kann durch Bereinigen, Versionierung, geänderten Port oder eine komplette Neuinstallation der .apk auf dem Zielgerät erfolgen.

*Siehe Abbildung 19 bis 21*

## 4 Anhang

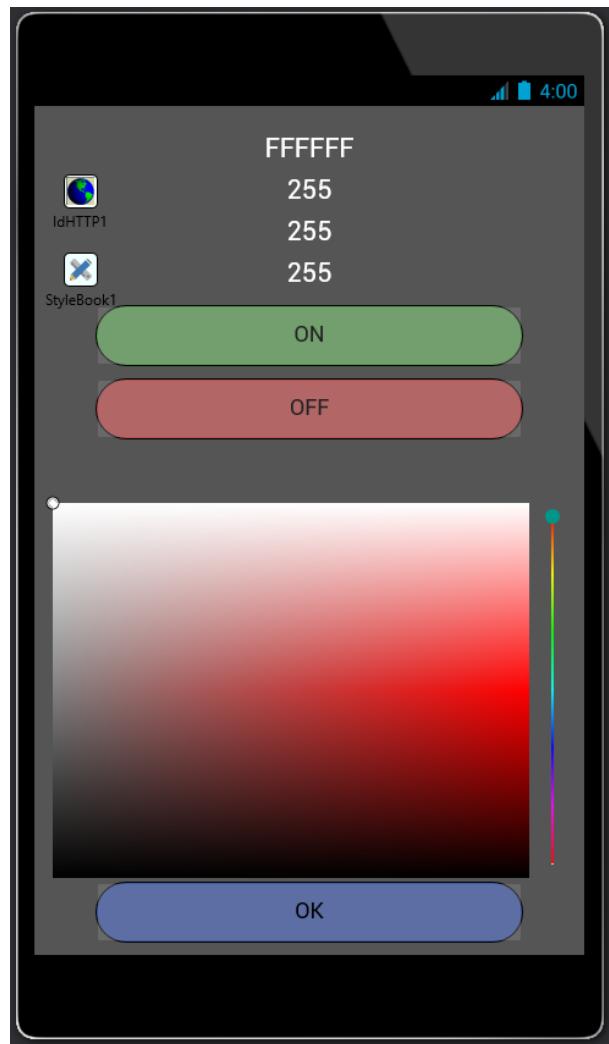


Abbildung 2: Entwurf

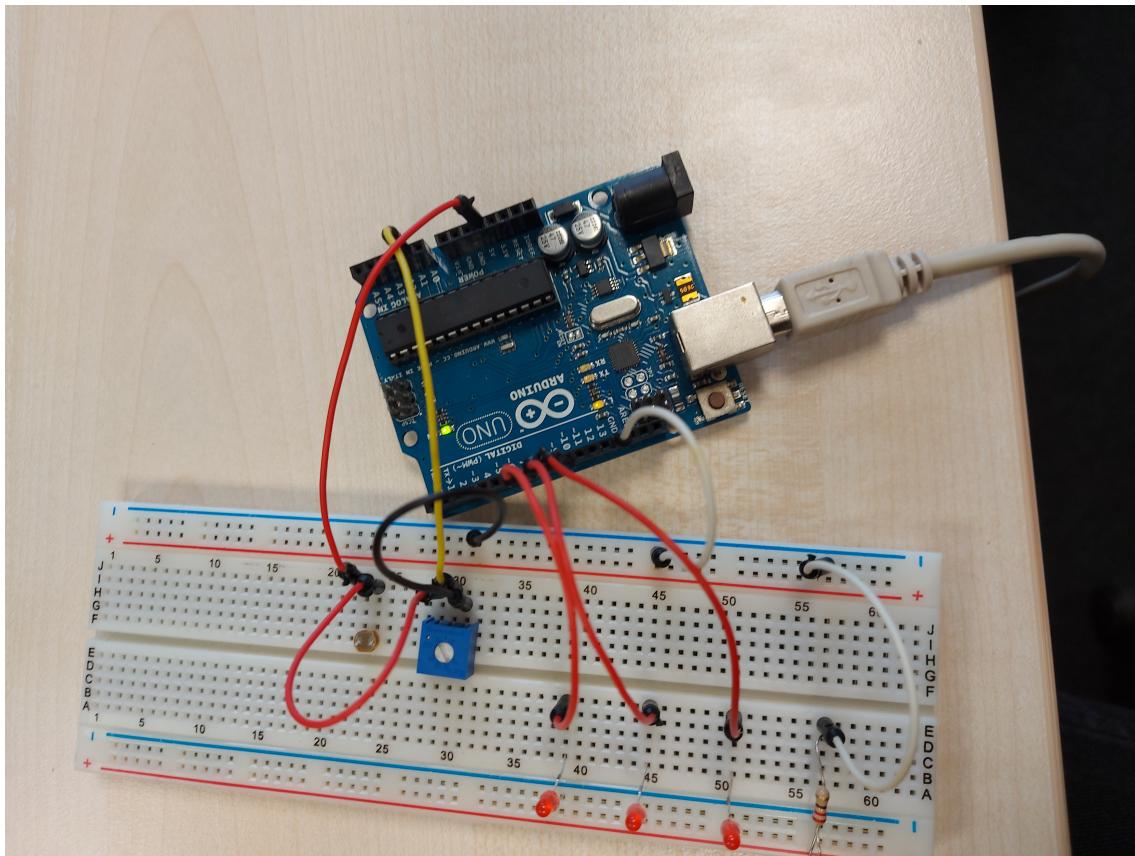


Abbildung 3: Arduino Test Led Aufbau

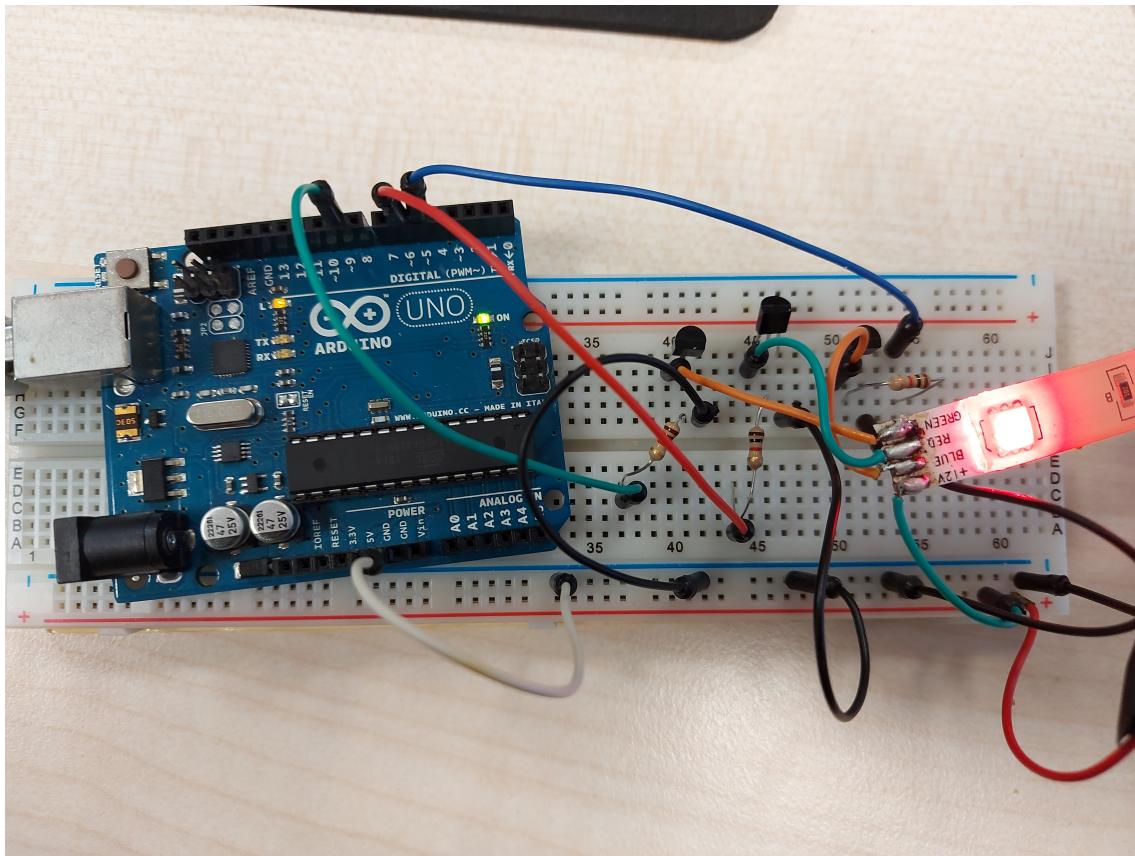


Abbildung 4: Arduino Aufbau

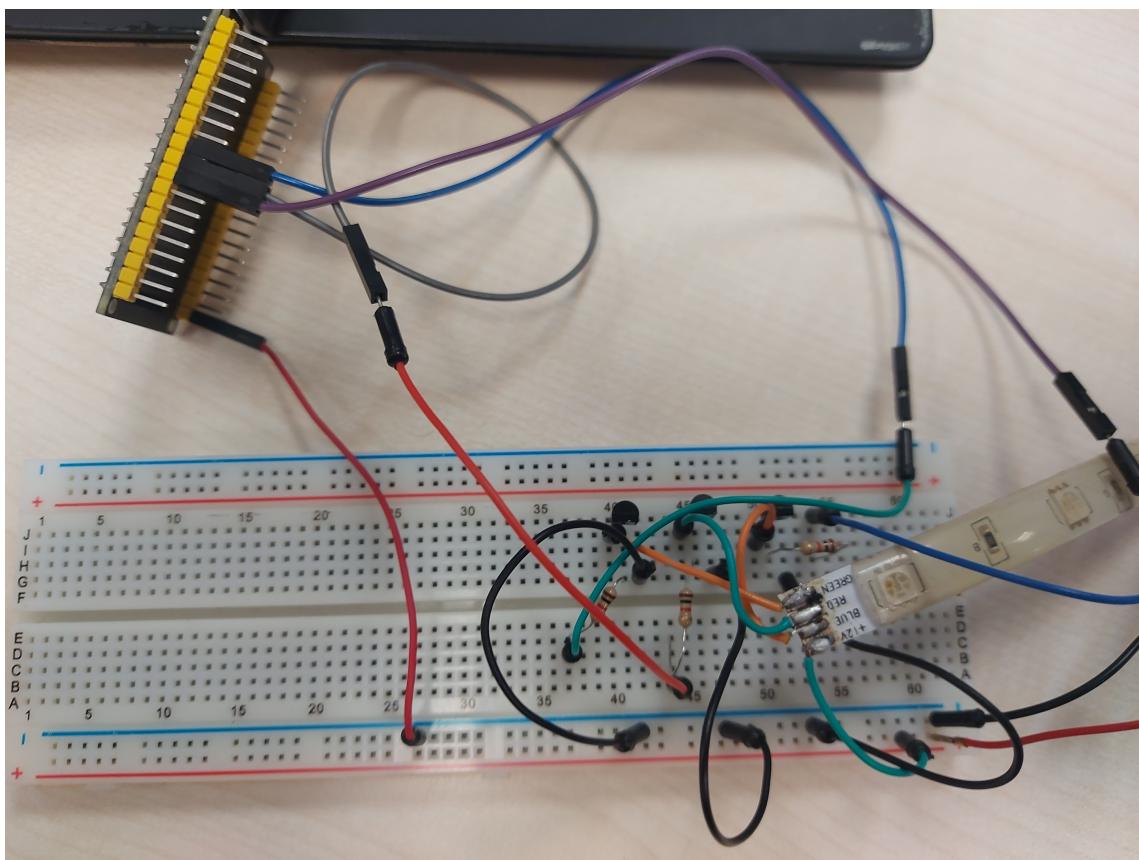


Abbildung 5: ESP-32 Aufbau

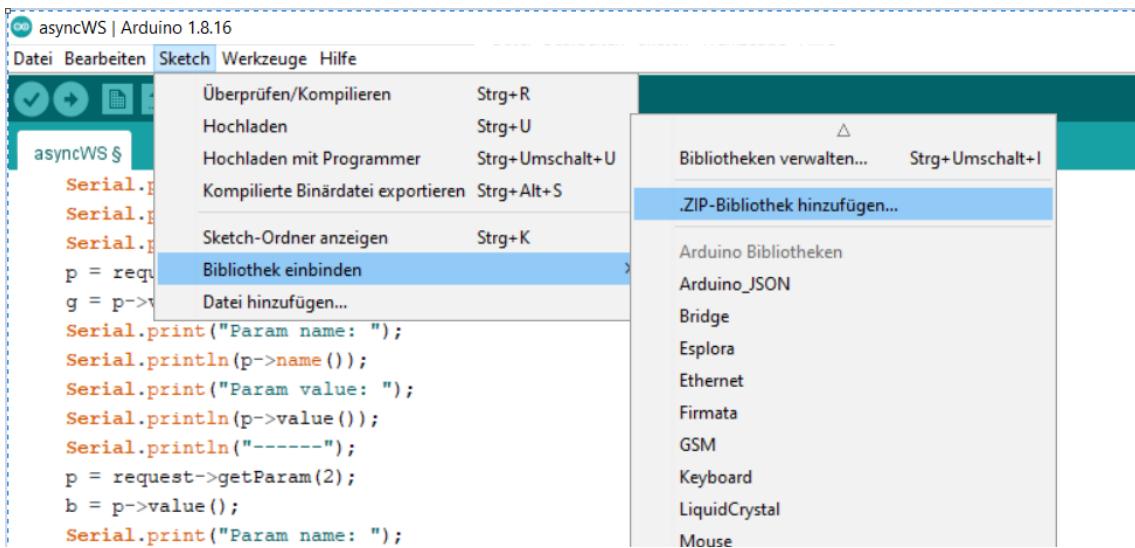


Abbildung 6: Setup Arduino

```

#include "WiFi.h"
#include "ESPAsyncWebServer.h"

const char* ssid = "IT-Network";
const char* password = "konradzuse";

AsyncWebServer server(80);

// Auxiliar variables to store the current output state
String ledState = "off";


// Assign output variables to GPIO pins
const int redPin = 25;
const int greenPin = 26;
const int bluePin = 27;
// setting PWM properties
const int freq = 5000;
const int ledChannel0 = 0;
const int ledChannell = 1;
const int ledChannel2 = 2;
const int resolution = 8;
String r, g, b;

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

    // configure LED PWM functionalitites
    ledcSetup(ledChannel0, freq, resolution);
    ledcSetup(ledChannell, freq, resolution);
    ledcSetup(ledChannel2, freq, resolution);

    // attach the channel to the GPIO to be controlled
    ledcAttachPin(redPin, ledChannel0);
    ledcAttachPin(greenPin, ledChannell);
    ledcAttachPin(bluePin, ledChannel2);

```

Abbildung 7: Arduino Code 1

```

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
}

Serial.println(WiFi.localIP());

server.on("/", HTTP_GET, [] (AsyncWebServerRequest * request) {

    int paramsNr = request->params();
    Serial.println(paramsNr);

    AsyncWebParameter* p = request->getParam(0);
    r = p->value();
    Serial.print("Param name: ");
    Serial.println(p->name());
    Serial.print("Param value: ");
    Serial.println(p->value());
    Serial.println("-----");
    p = request->getParam(1);
    g = p->value();
    Serial.print("Param name: ");
    Serial.println(p->name());
    Serial.print("Param value: ");
    Serial.println(p->value());
    Serial.println("-----");
    p = request->getParam(2);
    b = p->value();
    Serial.print("Param name: ");
    Serial.println(p->name());
    Serial.print("Param value: ");
    Serial.println(p->value());
    Serial.println("-----");
}

```

Abbildung 8: Arduino Code 2

```
RGBcolor(r.toInt(), g.toInt(), b.toInt());  
  
    request->send(200, "text/plain", "message received");  
});  
  
server.begin();  
}  
  
void loop() {  
    // put your main code here, to run repeatedly:  
  
}  
  
void RGBcolor(int red, int green, int blue) {  
    ledcWrite(ledChannel0, red);  
    ledcWrite(ledChannel1, green);  
    ledcWrite(ledChannel2, blue);  
  
}
```

Abbildung 9: Arduino Code 3

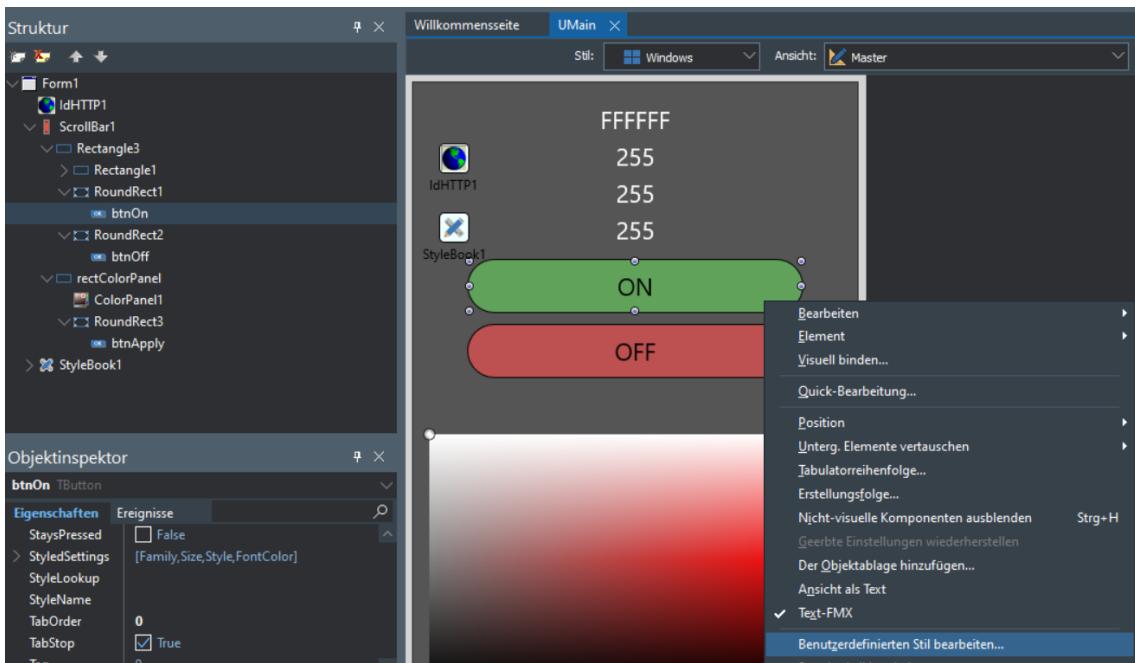


Abbildung 10: Design Struktur

```

procedure TForm1.ColorPanellChange(Sender: TObject);
var
  hexString, hexGreen, hexBlue, hexRed: String;
begin
  hexString := IntToHex(ColorPanell.Color);
  lblRgb.Text := Copy(hexString, 3, 7);
  hexGreen := Copy(hexString, 3, 2);
  i1 := StrToInt64('$' + hexGreen);
  lblGreen.Text := IntToStr(i1);
  hexBlue := Copy(hexString, 5, 2);
  i2 := StrToInt64('$' + hexBlue);
  lblBlue.Text := IntToStr(i2);
  hexRed := Copy(hexString, 7, 2);
  i3 := StrToInt64('$' + hexRed);
  lblRed.Text := IntToStr(i3);
end;

```

Abbildung 11: Das Color Change Event

```

150  -   begin
151    try
152      urlString := 'http://192.168.101.20/?r=' + IntToStr(i3) + '&g=' +
153          IntToStr(i1) + '&b=' + IntToStr(i2);
154      IdHTTP1.Get(urlString);
155    except
156      on e: EIdHTTPProtocolException do
157        begin
158          if e.ErrorCode = 404 then
159            ShowMessage('Bad Request');
160        end;
161    end;

```

Abbildung 12: Der HTTP Request

```

110  -   begin
111    appINI := TIniFile.Create(TPath.Combine(AppPath, 'mysettings.ini'));
112    try
113      h1 := appINI.ReadString('rgb', 'i1', '255'); // 255 is default
114      i1 := h1.ToInteger;
115      h2 := appINI.ReadString('rgb', 'i2', '255'); // 255 is default
116      i2 := h2.ToInteger;
117      h3 := appINI.ReadString('rgb', 'i3', '255'); // 255 is default
118      i3 := h3.ToInteger;
119    finally
120      appINI.Free;
121    end;
122  end;

130  -   begin
131    appINI := TIniFile.Create(TPath.Combine(AppPath, 'mysettings.ini'));
132    try
133      h1 := i1.ToString;
134      appINI.WriteString('rgb', 'i1', h1);
135      h2 := i2.ToString;
136      appINI.WriteString('rgb', 'i2', h2);
137      h3 := i3.ToString;
138      appINI.WriteString('rgb', 'i3', h3);
139    finally
140      appINI.Free;
141    end;
142  end;

```

Abbildung 13: Die Methoden iniload und inisave

Debug Konfiguration - Android 32 Bit Plattform						
Lokaler Pfad	Lokaler Name	Typ	Konfigurati...	Plattformen	Remote-Pfad	Remote-Name
\$(BDS)\bin\Artwork\...	FM_NotificationIcon_36...	Android_Notify...	Debug	[Android]	res\drawable-hdpi\	ic_notification.png
\$(BDS)\bin\Artwork\...	FM_LauncherIcon_48x4...	Android_Laun...	Debug	[Android]	res\drawable-mdpi\	ic_launcher.png
\$(BDS)\bin\Artwork\...	FM_NotificationIcon_96...	Android_Notify...	Debug	[Android]	res\drawable-xxxhdpi\	ic_notification.png
\$(BDS)\bin\Artwork\...	FM_LauncherIcon_144x...	Android_Laun...	Debug	[Android]	res\drawable-xxhdpi\	ic_launcher.png
\$(BDS)\bin\Artwork\...	FM_SplashImage_470x3...	Android_Splas...	Debug	[Android]	res\drawable-normal\	splash_image.png
\$(BDS)\bin\Artwork\...	FM_NotificationIcon_24...	Android_Notify...	Debug	[Android]	res\drawable-mdpi\	ic_notification.png
\$(BDS)\bin\Artwork\...	FM_LauncherIcon_96x9...	Android_Laun...	Debug	[Android]	res\drawable-xhdpi\	ic_launcher.png
\$(BDS)\bin\Artwork\...	FM_LauncherIcon_192x...	Android_Laun...	Debug	[Android]	res\drawable-xxxhdpi\	ic_launcher.png
\$(BDS)\bin\Artwork\...	FM_NotificationIcon_48...	Android_Notify...	Debug	[Android]	res\drawable-xhdpi\	ic_notification.png
\$(BDS)\lib\android\...	libnative-activity.so	AndroidLibnat...	Debug	[Android]	library\lib\armeabi\	libled.so
\$(BDS)\lib\android\...	libnative-activity.so	AndroidLibnat...	Debug	[Android]	library\lib\mips\	libled.so
\$(NDKBasePath)\pre...	gdbserver	AndroidGDBSe...	Debug	[Android]	library\lib\armeabi-v7a\	gdbserver
Android\Debug\	classes.dex	AndroidClasse...	Debug	[Android]	classes\	classes.dex
Android\Debug\	splash_image_def.xml	AndroidSplash...	Debug	[Android]	res\drawable\	splash_image_def.xml
Android\Debug\	styles.xml	AndroidSplash...	Debug	[Android]	res\values\	styles.xml
Android\Debug\	libled.so	ProjectOutput	Debug	[Android]	library\lib\armeabi-v7a\	libled.so
Android\Debug\	AndroidManifest.xml	ProjectAndroi...	Debug	[Android]	\	AndroidManifest.xml
Android\Debug\	colors.xml	Android_Colors	Debug	[Android]	res\values\	colors.xml
Android\Debug\	styles-v21.xml	AndroidSplash...	Debug	[Android]	res\values-v21\	styles.xml
Android\Debug\	strings.xml	Android.Strings	Debug	[Android]	res\values\	strings.xml
myAppData\	mysettings.ini	File	Debug	[Android]	\assets	mysettings.ini

Abbildung 14: Die Bereitstellung der ini für Android

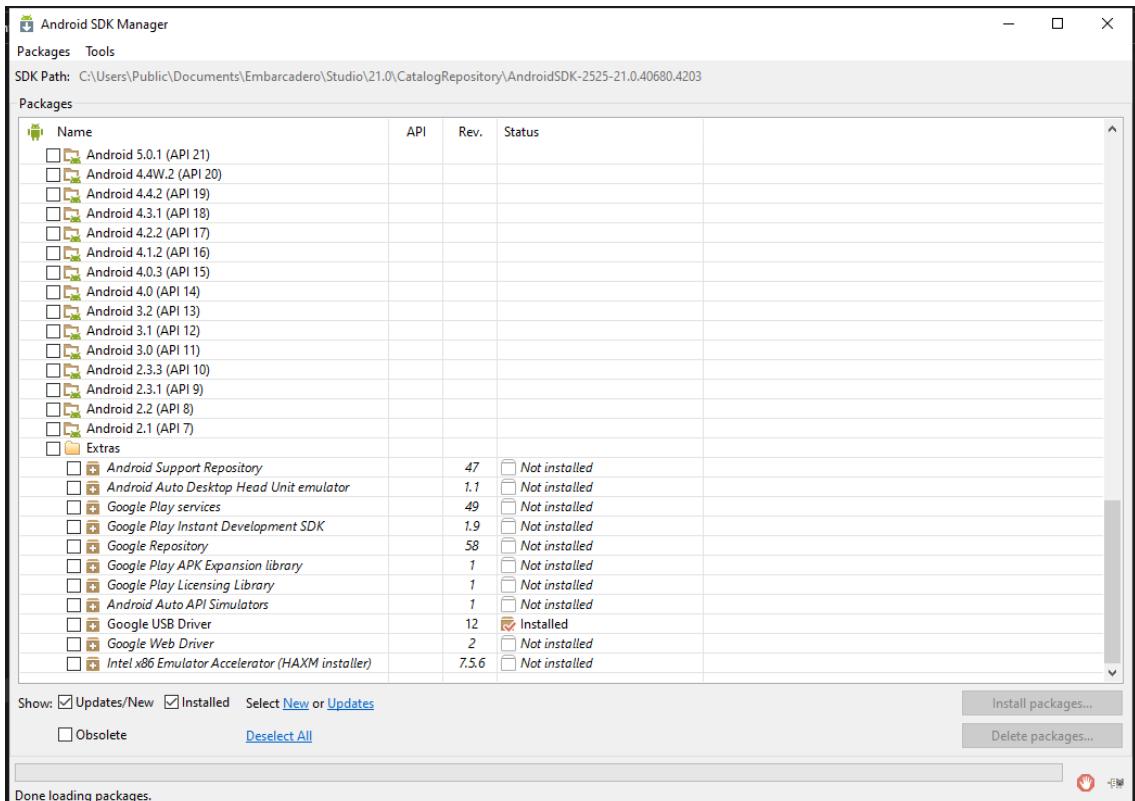


Abbildung 15: Der SDK Manager

The screenshot shows the Samsung Developers website at https://developer.samsung.com/android-usb-driver. The page has a navigation bar with links for Learn, Develop, Design, Distribute, Support (which is underlined), and Connect. The main title is "Samsung Android USB Driver".

## Samsung Android USB Driver

### Samsung Android USB Driver for Windows

You need the driver only if you are developing on Windows and want to connect a Samsung Android device to your development environment over USB.

[Samsung Android USB Driver for Windows v1.7.48 \(36.89MB\) | Oct 25, 2021](#)

Abbildung 16: USB Treiber für Samsung Android

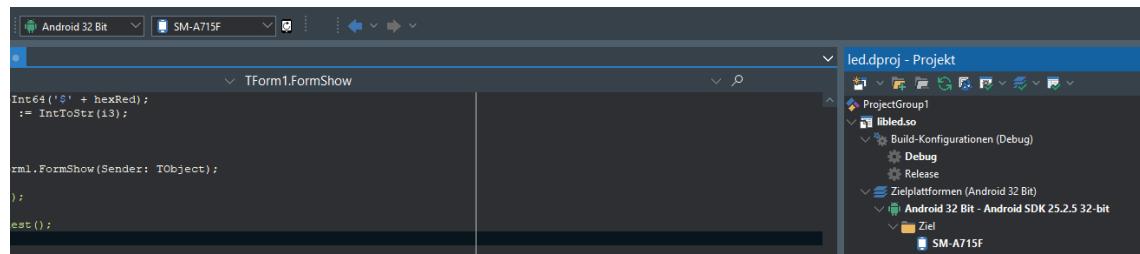


Abbildung 17: Auswahl des Android Gerätes in der IDE

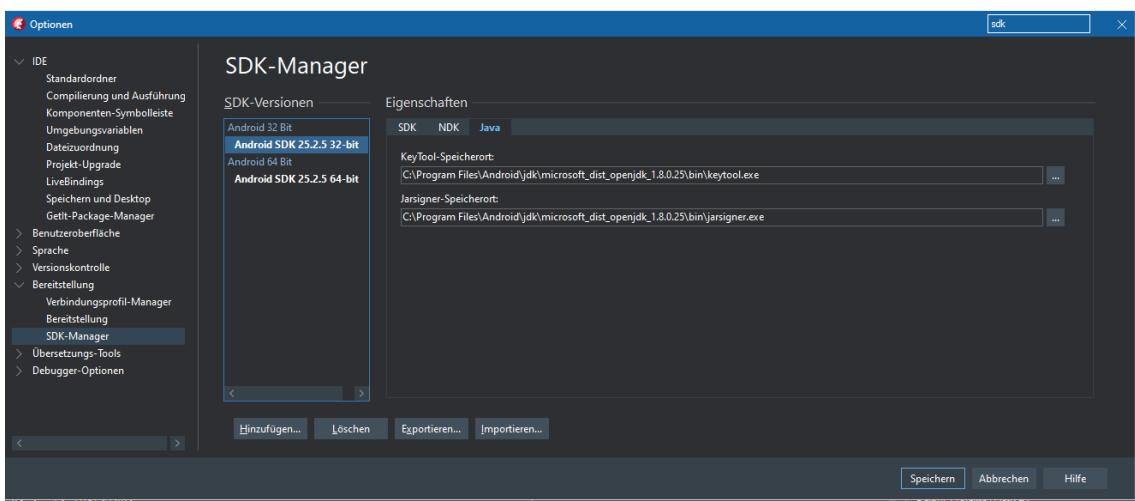


Abbildung 18: Das richtige Java verwenden

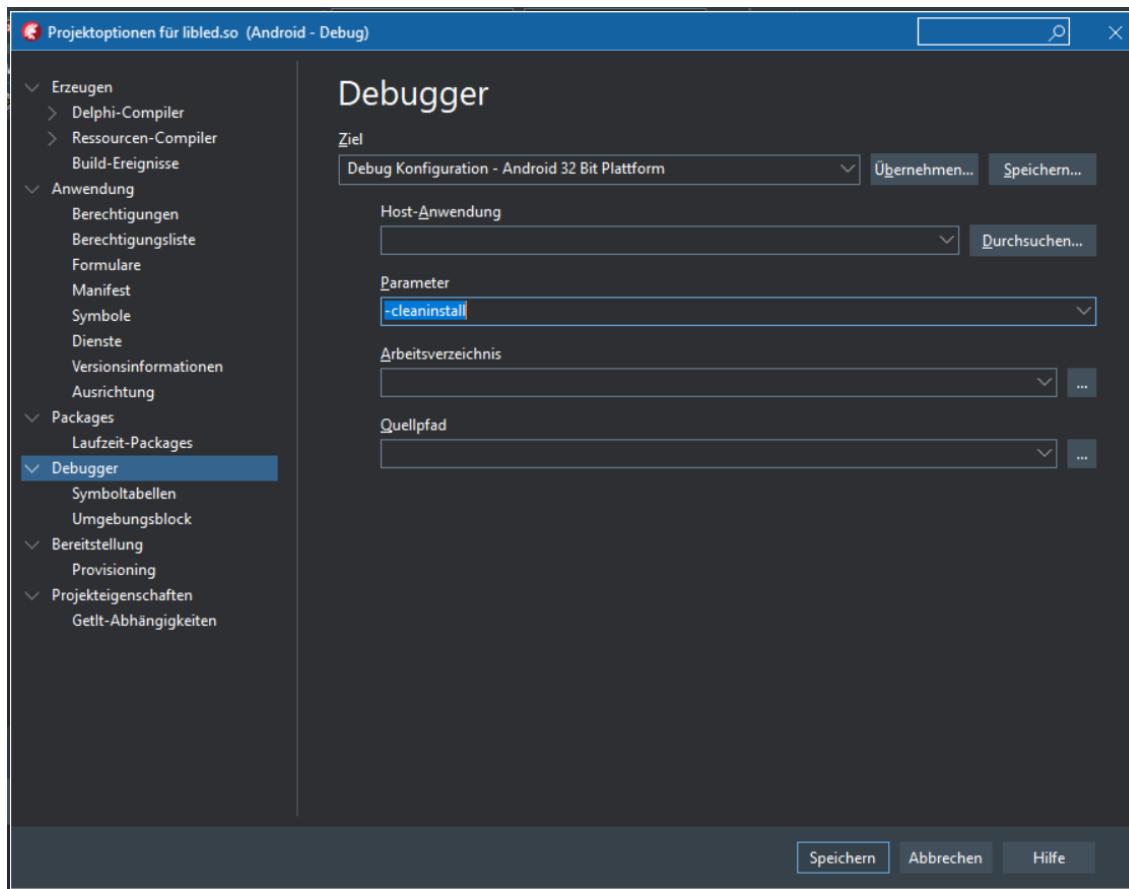


Abbildung 19: Alle Daten komplett neu mitgeben

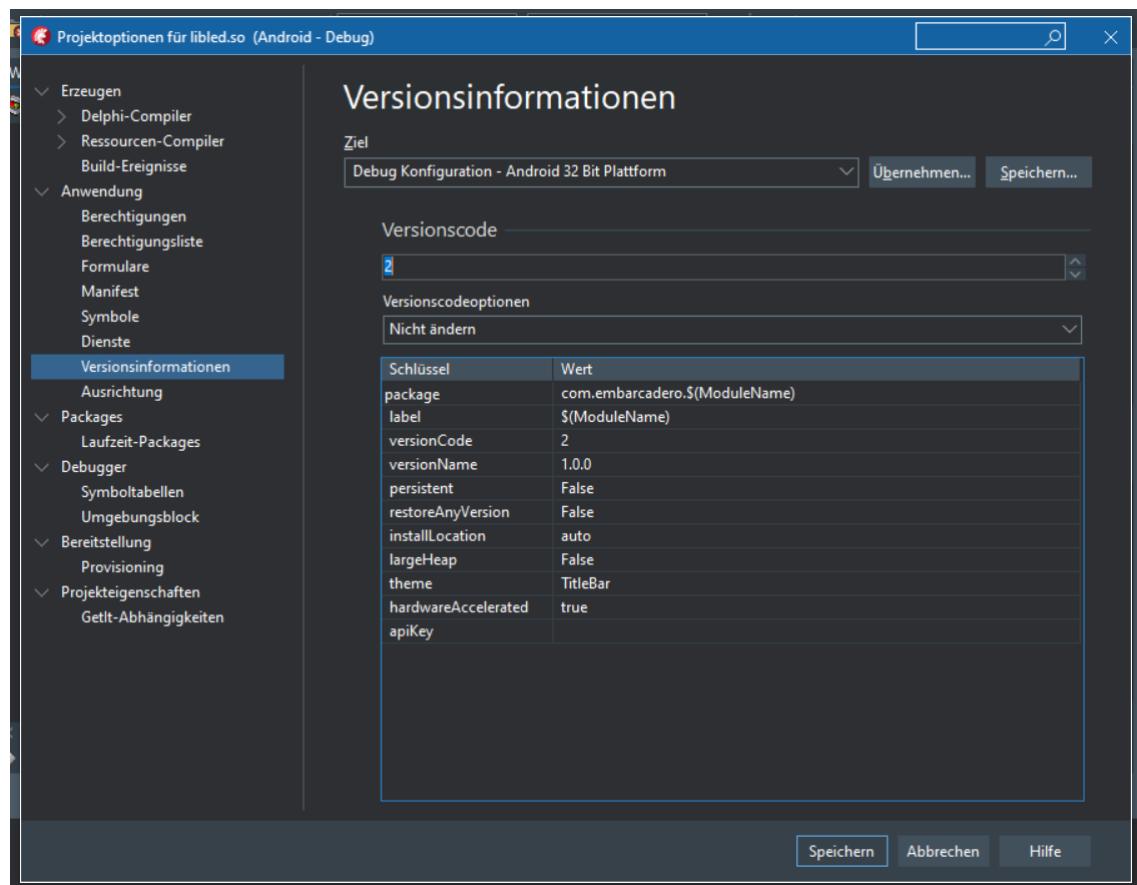


Abbildung 20: Version beim kompilieren hoch setzen nach Signatur-Fehler

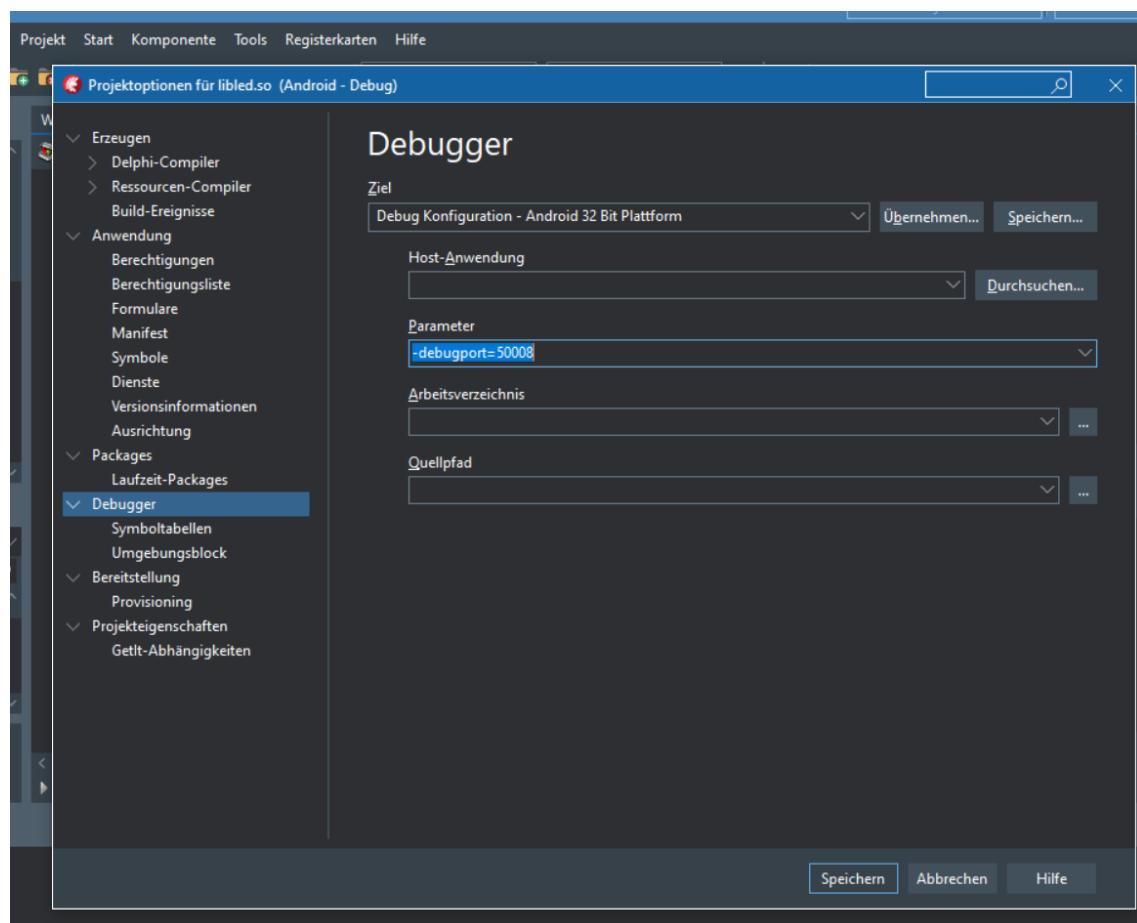


Abbildung 21: Den Port ändern nach Signatur Fehler