ESP Zigbee SDK Programming Guide-

ESP Zigbee SDK ist das offizielle Zigbee-Entwicklungskonzept für Espressifs ESP32-Serie SoCs.

Es bietet vereinfachte APIs, häufig verwendete Peripheriegeräte, Werkzeuge und Dienstprogramme für Sicherheit, Fertigung und Produktion, begleitet von einer umfassenden Dokumentation. Es umfasst umfangreiche Produktionsreferenzen, die darauf abzielen, den Entwicklungsprozess der Zigbee-Produkte zu vereinfachen und es den Nutzern zu ermöglichen, in kürzester Zeit in die Produktion zu gehen.

Inhaltsverzeichnis-

1. Einführung-

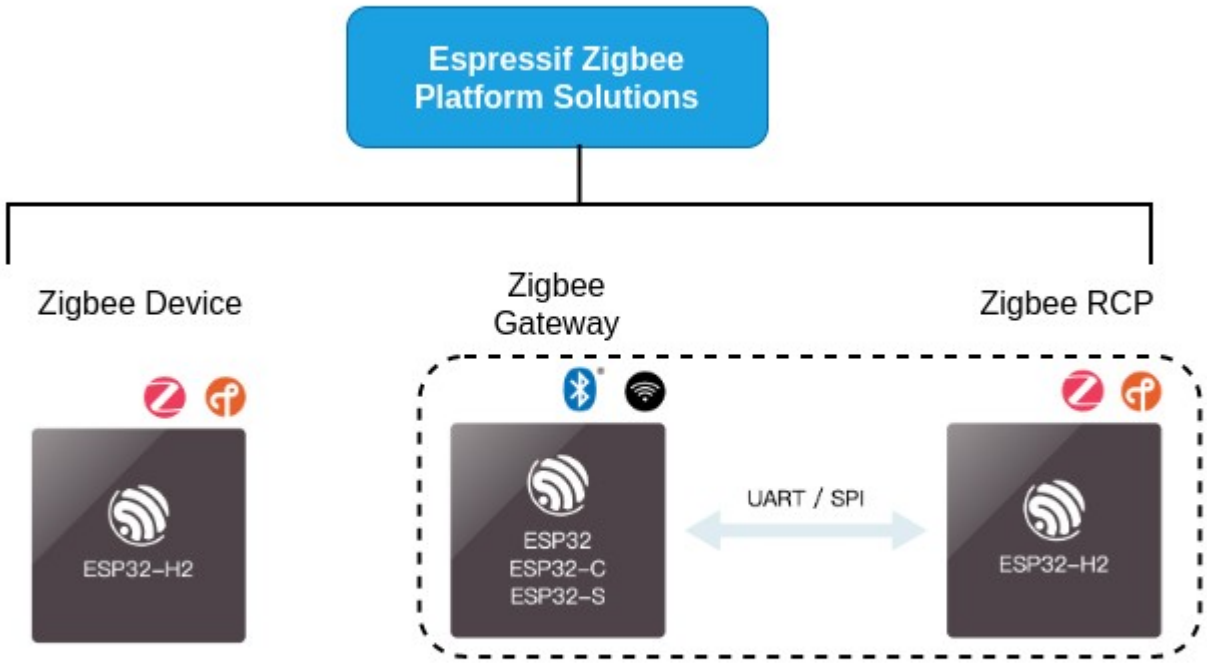
1.1 Espressif Zigbee-

Die Zigbee-Lösungen von Espressif bestehen aus:

- Ein ganzes Spektrum an Zigbee-Geräteplattformen
- Produktion fertig SDK
- Zigbee und ESP RainMaker

1.1.1 Espressif Zigbee Plattformen-

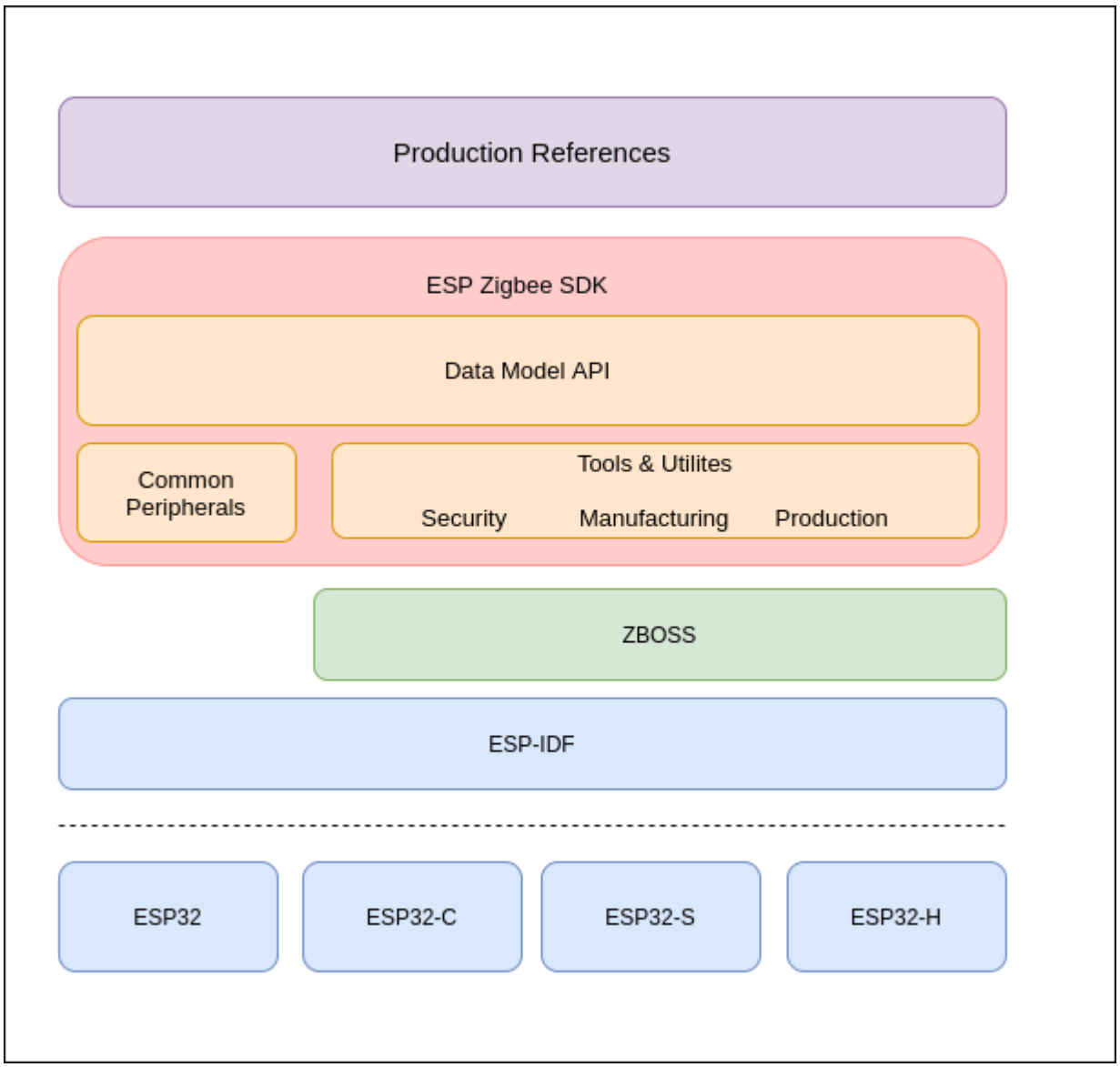
Espressif-Plattformlösungen sind wie unten gezeigt:



- 802.15.4 SoCs (ESP32-H2, ESP32-C6 usw.) können **zum** Bau **von Zigbee-Geräten** verwendet werden.
- Durch die effiziente Kombination von Espressif 802.15.4 und Wi-Fi SoCs (ESP32, ESP32-C3, ESP32-S3, etc.) kann **das Zigbee-Gateway** gebaut werden, um das Zigbee-Netzwerk mit dem Wi-Fi-Netzwerk zu verbinden.
- Wir bieten auch Matter-Zigbee-Brückenlösung, die Nicht-Materie-Geräte auf der Grundlage von Zigbee und anderen Protokollen die Verbindung zum Matter-Ökosystem ermöglichen. Siehe ESP Matter.

1.1.2 ESP Zigbee SDK-

Das Zigbee SDK von Espressif basiert auf dem ESP-IDF und Zboss Stack.



Das ZBOSS und Espressif Zigbee SDK wird als vorgefertigte Bibliothek bereitgestellt:

- esp-zboss-lib : ZBOSS Bibliotheken für ESP32 Serie SoCs
- esp-zigbee-lib : Espressif Zigbee SDK und APIs Bibliotheken

Diese beiden Bibliothekskomponenten werden von ESP Registry gehostet.

1.1.3 Zigbee und ESP RainMaker Integration-

Die AIoT-Cloud-Plattform ESP RainMaker von Espressif bietet Fernsteuerung und Cloud-basierte Geräteverwaltung für Zigbee-Geräte.

Durch die Kombination der oben genannten Zigbee Hardware- und Softwarelösungen mit ESP RainMaker bietet diese One-Stop-Zigebee-Ökosystemlösung einen vollwertigen Cloud-Bereit-Einsatz über Ihr eigenes privates Konto mit erweiterten Geräteverwaltungsfunktionen.

1.2 Unterstützte Features-

Die unterstützten Features im aktuellen ESP Zigbee SDK sind unten aufgeführt:

- Zigbee 3.0
- Zigbee Pro R22
- Zigbee Cluster Library (ZCL) v8
- Home Automation Geräte
- Touchlink
- Green Power Proxy, Sink, GPD
- Koordinator / Router / ZED / Sleepy Geräterollen
- Gateway und Radio Co-Prozessor (RCP) Beispiel
- Netzwerk-Co-Processor (NCP) und Host-Beispiel
- Sniffer

1.2.1 ZCL Cluster-

ZCL Cluster-

Cluster Name	Cluster ID
primc	0x0000
Power-Konfiguration	0x0001
identifizieren	0x0003
Gruppen	0x0004
Szenen	0x0005
on-off	0x0006
on-off-switch-cfg	0x0007
Maßhaltigkeit	0x0008
Zeit	0x000a
analoge	0x000c
analog-output	0x000d
Analog-Wert	0x000e
binary-input	0x000f

Cluster Name	Cluster ID
multistate-Wert	0x0014
Inbetriebnahme	0x0015
ota	0x0019
green-power	0x0021
shade-config	0x0100
Türschloss	0x0101
Fensterbelag	0x0102
Thermostat	0x0201
Fan-Steuerung	0x0202
Entfeuchtungssteuerung	0x0203
thermostat-user-interface	0x0204
color-control	0x0300
Beleuchtungsstärke	0x0400
Temperaturmessung	0x0402
Druckmessung	0x0403
flow-measur	0x040404
Feuchtemessung	0x0405
Belegungsmessung	0x0406
pH-Messung	0x0409
electric-conductivity – Messung	0x040a
Wind-Geschwindigkeit	0x040b
carbon-dioxid-measur	0x040d
PM2.5-Messung	0x042a

Cluster Name	Cluster ID
ias-zone	0x0500
iasace	0x0501
iaswd	0x0502
Preis	0x0700
Messing	0x0702
meter-identifikation	0x0b01
Elektro-Messung	0x0b04
Diagnose	0x0b05
touchlink-inbetrieblich	0x1000

1.2.1.1 Attribute und Befehle-

Siehe das unterstützte Attribut jedes Clusters in ZCL api-reference

1.2.2 HA Automation Geräte-

Zigbee Home Automation-

Geräte-ID	Name des Geräts	Standardcluster
0x0002	on-off-light	basic, identifizieren, Gruppen, Szenen, on-off
0x0000	on-off-switch	basic, identifizieren
0x0102	color-dimmable-light	basic, identifizieren, Gruppen, Szenen, on-off, level-control, color-control
0x0105	color-dimmable-switch	basic, identifizieren
0x0009	Power-Outlet	basic, identifizieren, Gruppen, Szenen, on-off
0x0200	Schatten	basic, identifizieren, Gruppen, Szenen, On-off, Level-control, Schatten
0x0201	shade-controler	basic, identifizieren
0x000A	Türschloss	basic, identifizieren, Gruppen, Szenen, Türsperre

Geräte-ID	Name des Geräts	Standardcluster
0x000B	door-lock-controller	basic, identifizieren
0x0301	Thermostat	Grund-, Identifizierung, Thermostat-Konfiguration
0x0302	Temperatursensor	Grund-, Identifizierung, Temperatur-Meas
0x0005	Konfiguration	basic, identifizieren, Gruppen, Szenen, on-off, level-control, color-control

Siehe Standard-PfNO-Attribute in jedem HA-Gerätecluster in HA api-reference

1.2.3 Mehr unterstützt-

Für neue Cluster- oder Geräteanforderungen öffnen Sie bitte ein Problem auf GitHub. Wir melden uns bald bei Ihnen.

2. Entwicklung mit ESP Zigbee SDK-

Bitte informieren Sie die Release Notes, um mehr über ESP Zigbee SDK-Releases zu erfahren. Überprüfen Sie README, um weitere Details zu erfahren.

2.1 Entwicklungs-Setup-

Dieser Abschnitt spricht über die Einrichtung Ihrer Entwicklungsumgebung, das Holen der Git-Repositorien und die Anweisungen zum Bauen und Aufblitzen.

2.1.1 Einrichtung der Repositories-

Folgen Sie dem ESP-IDF, der einen Leitfaden für die Einrichtung der IDF-Entwicklungsumgebung erhält. Für Anfänger überprüfen Sie bitte Installation Step by Step for esp-idf.

Klonen esp-idf:

```
git clone --recursive https://github.com/espressif/esp-idf.git

cd esp-idf

git checkout v5.3.2

git submodule update --init --recursive

./install.sh

source ./export.sh

cd ..
```

Klonen esp-zigbee-sdk:

```
git clone https://github.com/espressif/esp-zigbee-sdk.git
```

2.1.2 Anwendungen bauen und blitzen-

Verschiedene Zigbee-Beispiele sind mit dem SDK versehen:

- Zickbee Beispiele

Wählen Sie unter einem Beispielordner IDF SoC Ziel.

```
idf.py set-target esp32
```

- Wenn das IDF-Ziel nicht explizit festgelegt wurde, dann `esp32`ist als Standard angesehen.

Bauen und blinken Sie das Beispiel.

```
idf.py -p PORT erase_flash flash monitor
```

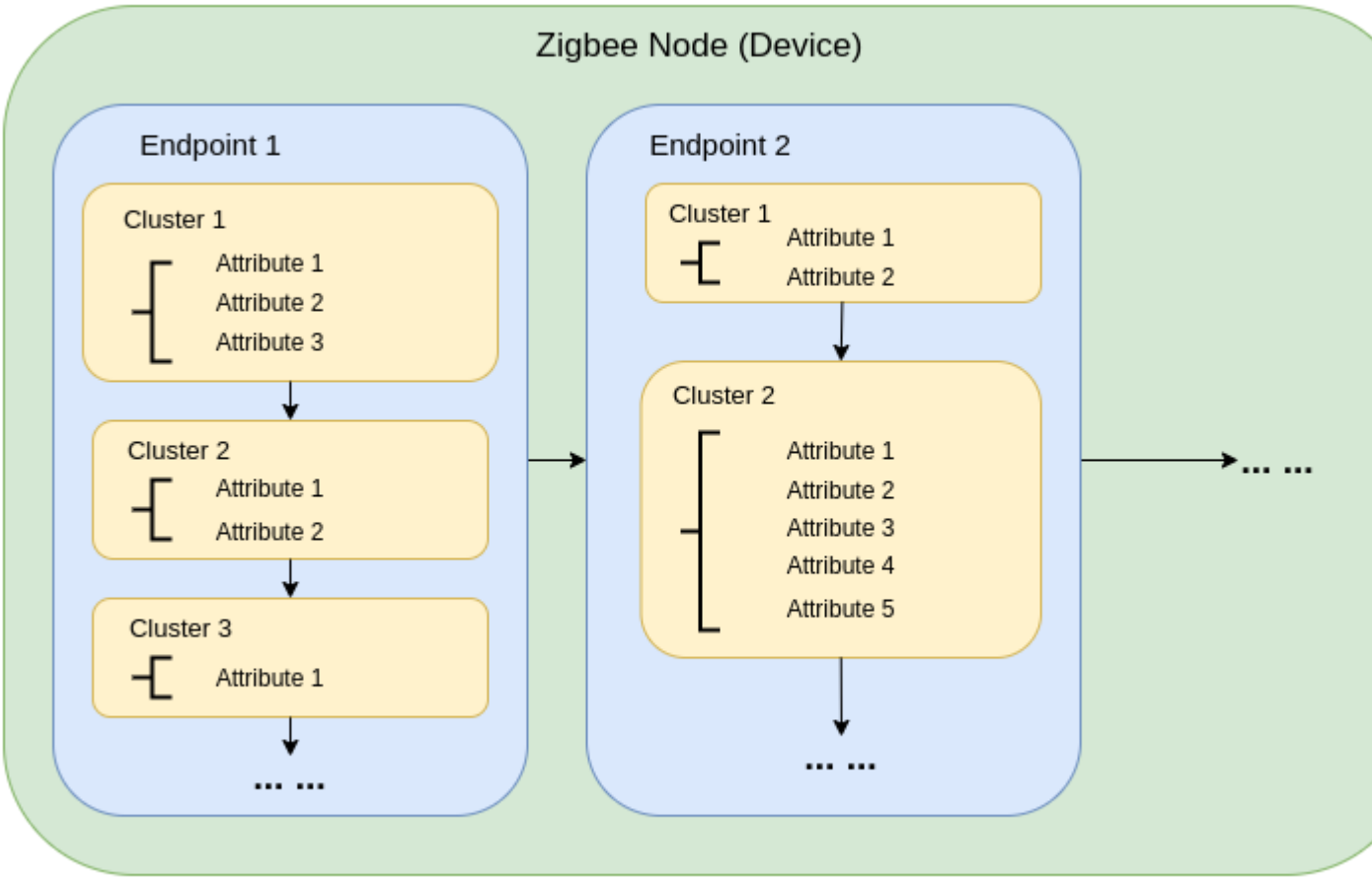
2.2 Laufende Beispiel-

Weitere Details finden Sie in den Beispielen readme: README

2.3 Produktentwicklung-

2.3.1 Datenmodell-

Die Struktur vor dem eigentlichen Ändern und Anpassen verstehen Das Gerät ist hilfreich.



Node-

Ein Knoten ist ein einzelnes ESP32-H2 basiertes Produkt. Es stellt einen Netzwerkknoten im Zigbee-Netzwerk dar. Ein einzelner Knoten kann mehrere Endpunkte freilegen.

Endpunkte-

Innerhalb jedes Knotens sind Endpunkte. Endpunkte, die durch eine Zahl zwischen 1 und 240 identifiziert wurden, definieren jede Anwendung, die in einem ZigBee-Knoten läuft (ja, ein einzelner ZigBee-Knoten kann mehrere Anwendungen ausführen). Endpunkte dienen drei Zwecken in ZigBee:

- Endpunkte ermöglichen es, verschiedene Anwendungsprofile in jedem Knoten zu erstellen.
- Endpunkte ermöglichen es, innerhalb jedes Knotens separate Kontrollpunkte zu erhalten.
- Endpunkte ermöglichen es, dass separate Geräte innerhalb jedes Knotens vorhanden sind.

Cluster-

Cluster, die durch eine 16-Bit-Kennung definiert sind, sind Anwendungsobjekte. Während NwkAddr und der Endpunkt Konzepte adressieren, definiert der Cluster Anwendungsbedeutung.

- Ein Endpunkt kann mehrere Cluster haben.

- Cluster haben neben der Kennung Richtung. Im SimpleDescriptor, der einen Endpunkt beschreibt, wird ein Cluster entweder als Ein- oder Ausgabe aufgeführt.
- Cluster enthalten sowohl Code (Befehle) als auch Daten (Attribute). Befehle verursachen Aktion. Attribute verfolgen den aktuellen Stand dieses Clusters.

Attribute-

Attribute werden durch eine 16-Bit-Nummer identifiziert, speichern Sie den aktuellen "Staat" eines bestimmten Clusters. Ein Datenunternehmen, das eine physische Menge oder einen Zustand darstellt.

- Ein Cluster kann mehrere Attribute haben.
- Es gibt generische ZCL-Befehle zum Lesen und Schreiben von Attributen auf einen bestimmten Cluster.
- Attribute können sogar in regelmäßigen Abständen automatisch angezeigt werden, wenn sie sich ändern, oder beides.

2.3.2 Beispiel HA-on-off-light-

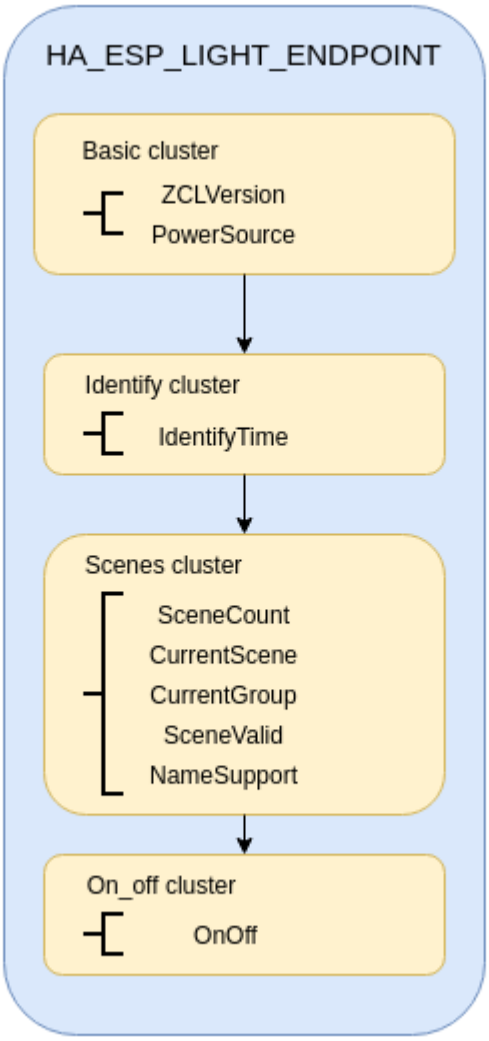
Dieser Abschnitt zeigt, wie ein Home Automation (HA) On-Off-Light-Beispiel als Referenz erstellt.

Siehe Beispiel: HA-on-off-light

2.3.2.1 Datenmodell-

Im Standardbeispiel HA-on-off-light verwenden wir `esp_zb_on_off_light_ep_create()`um einen HA auf einem einzigen Endpunkt zu schaffen.

Datenmodell sieht aus:



Oben ist die Endpunktliste, die wir erstellt haben, dann verwenden wir `esp_zb_device_register()`um ein Zigbee-Gerät zu registrieren.

2.3.2.2 Attribut Ruf-

Ein Attribut-Änderungsruf in `esp_zb_device_add_set_attr_value_cb()`um dem Benutzer die Benachrichtigung zu ermöglichen, dass sich bestimmte Attribute geändert haben.

Es hängt davon ab, wie man diese Attributänderungen basierend auf verschiedenen Umständen verarbeitet, siehe das Beispiel, um LED-Licht auf off zu handhaben.

```
void attr_cb(uint8_t status, uint8_t endpoint, uint16_t cluster_id, uint16_t
attr_id, void *new_value)
{
 if (cluster_id == ESP_ZB_ZCL_CLUSTER_ID_ON_OFF) {
     uint8_t value = *(uint8_t*)new_value;
     if (attr_id == ESP_ZB_ZCL_ATTR_ON_OFF_ON_OFF_ID) {
         /* implemented light on/off control */
         ESP_LOGI(TAG, "on/off light set to %hd", value);
         light_driver_set_power((bool)value);
     }
 } else {
     /* Implement some actions if needed when other cluster changed */
```

```
     ESP_LOGI(TAG, "cluster:0x%x, attribute:0x%x changed ", cluster_id,
attr_id);
 }
}
```

2.3.2.3 Zigbee stack Konfigurieren und Starten-

In diesem Beispiel ist ein Zigbee-Endgerät in HA-on-off-light konfiguriert. `ESP_ZB_ZED_CONFIG()`, siehe `esp_zb_cfg_t`Definition im API-Referenzkapitel unten.

Zigbee Stapel mit Verwendung `esp_zb_init()`, Start Zigbee Stack mit `esp_zb_start()`und Hauptschleife des Zigbee-Stacks mit `esp_zb_main_loop_iteration()`.

Nachdem Zigbee-Stack läuft, könnte der Benutzer durch die Überprüfung verschiedener Signale, die der Stack bereitgestellt hat, die BDB-Inbetriebnahme mit der Verwendung beginnen `esp_zb_bdb_start_top_level_commissioning()`.

2.3.3 Ein Beispiel für HA-on-off-Switch-

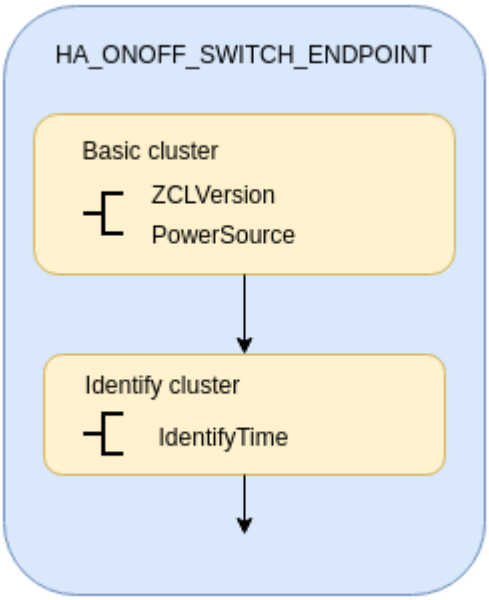
Dieser Abschnitt zeigt, wie ein Home Automation (HA) On-off-Switch-Beispiel als Referenz erzeugt.

Siehe Beispiel: HA-on-off-Schalter

2.3.2.1 Datenmodell-

Im Standardbeispiel HA-on-off-light verwenden wir `esp_zb_on_off_switch_ep_create()`um einen HA auf einem einzigen Endpunkt zu schaffen.

Datenmodell sieht aus:



Oben ist die Endpunktliste, die wir erstellt haben, dann verwenden wir `esp_zb_device_register()`um ein Zigbee-Gerät zu registrieren.

2.3.2.2 ZCL Befehle-

Varianten der ZCL-Befehle werden im Befehl esp-zigbee-zcl-cact bereitgestellt.

In diesem Beispiel zeigen wir einen Befehl ZCL on-off-toggle.

Nach dem Knopf auf dem Brett gedrückt, `esp_zb_buttons_handler()`erzeugt einen On-off-Schalterbefehl `esp_zb_zcl_on_off_cmd_req()`. Der Benutzer muss einen Remote-Knotenendpunkt, einen lokalen Node-Endpunkt, eine kurze Adresse und einen Befehlstyp bereitstellen, um die Befehlsanfrage zu packen. Siehe `esp_zb_zcl_on_off_cmd_t`.

```
static void esp_zb_buttons_handler(switch_func_pair_t *button_func_pair)
{
    if (button_func_pair->func == SWITCH_ONOFF_TOGGLE_CONTROL) {
        /* implemented light switch toggle functionality */
        esp_zb_zcl_on_off_cmd_t cmd_req;
        cmd_req.zcl_basic_cmd.dst_addr_u.addr_short = on_off_light.short_addr;
        cmd_req.zcl_basic_cmd.dst_endpoint = on_off_light.endpoint;
        cmd_req.zcl_basic_cmd.src_endpoint = HA_ONOFF_SWITCH_ENDPOINT;
        cmd_req.address_mode = ESP_ZB_APS_ADDR_MODE_16_ENDP_PRESENT;
        cmd_req.on_off_cmd_id = ESP_ZB_ZCL_CMD_ON_OFF_TOGGLE_ID;
        ESP_EARLY_LOGI(TAG, "send 'on_off toggle' command");
        esp_zb_zcl_on_off_cmd_req(&cmd_req);
    }
}
```

2.3.2.3 Zigbee stack Konfigurieren und Starten-

In diesem Beispiel ist ein Zigbee-Koordinator auf HA-on-off-Switch konfiguriert. `ESP_ZB_ZC_CONFIG()`, siehe `esp_zb_cfg_t`Definition im API-Referenzkapitel unten.

Zigbee Stapel mit Verwendung `esp_zb_init()`, Start Zigbee Stack mit `esp_zb_start()`und Hauptschleife des Zigbee-Stacks mit `esp_zb_main_loop_iteration()`.

Nachdem Zigbee-Stack läuft, könnte der Benutzer durch die Überprüfung verschiedener Signale, die der Stack bereitgestellt hat, die BDB-Inbetriebnahme mit der Verwendung beginnen `esp_zb_bdb_start_top_level_commissioning()`.

2.4 Debugging-

Wenn Sie mit dem Zigbee SDK auf Funktionsprobleme stoßen, können folgende Debugging-Tipps hilfreich sein.

2.4.1 Zigbee API Lock-

Die Zigbee SDK APIs sind nicht fadensicher, daher ist es obligatorisch, die Sperre zu erwerben, bevor Sie Zigbee APIs aufrufen, mit Ausnahme der folgenden Fälle:

- Die Call-Site ist in Zigbee-Rückrufe, die von Zigbee-Aufgabe stammen.
- Aufruf der Zeitplanalarm-APIs, zu denen auch *esp-zb-scheduler-alarm()* und *esp-zb-scheduler-alarm-cancel()*.

Ein Beispielcodeblock:

```
#include "esp_zigbee_core.h"
```

```
void application_task(void *pvParameters)
{
   ......
   esp_zb_lock_acquire(portMAX_DELAY);

   esp_zb_zcl_on_off_cmd_req(cmd_req);

   esp_zb_lock_release();
   ......
}
```

Das gleiche Schloss wird erworben *esp-zb-main-loop-iteration()* wenn die Zigbee-Aufgabe nicht untätig ist.

2.4.2 Stapelgröße-

Unzureichende Stack-Größe führt oft zu unerwarteten Laufzeitproblemen, Sie können uxTaskGetGtackHighWaterMark() FreeRTOS API verwenden, um die Stapelnutzung von Aufgaben zu überwachen.

2.4.3 Sniffer und Wireshark-

Die Analyse des Paketflusses, der von einem Schnüffel erfasst wird, ist eine effektive Methode, um Zigbee-Protokoll- und Fehlerbehebungsprobleme zu verstehen.

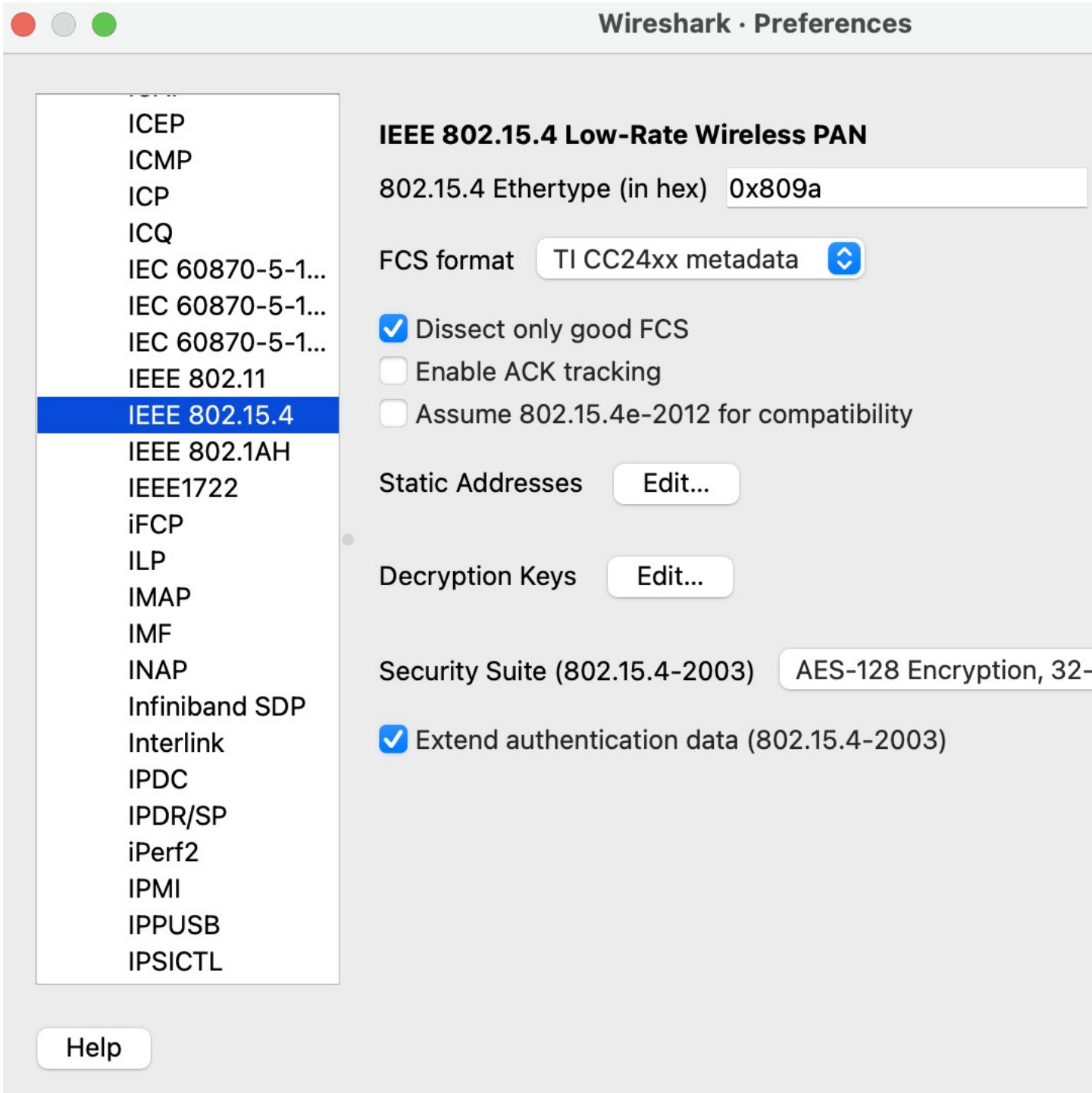
Um einen Sniffer für 802.15.4 einzurichten, benötigen Sie Folgendes:

- Eine Hostmaschine mit Pyspinel und Wireshark
- Ein 802.15.4 enabled valkit (ESP32-H2, ESP32-C6 usw.)ot_rcp

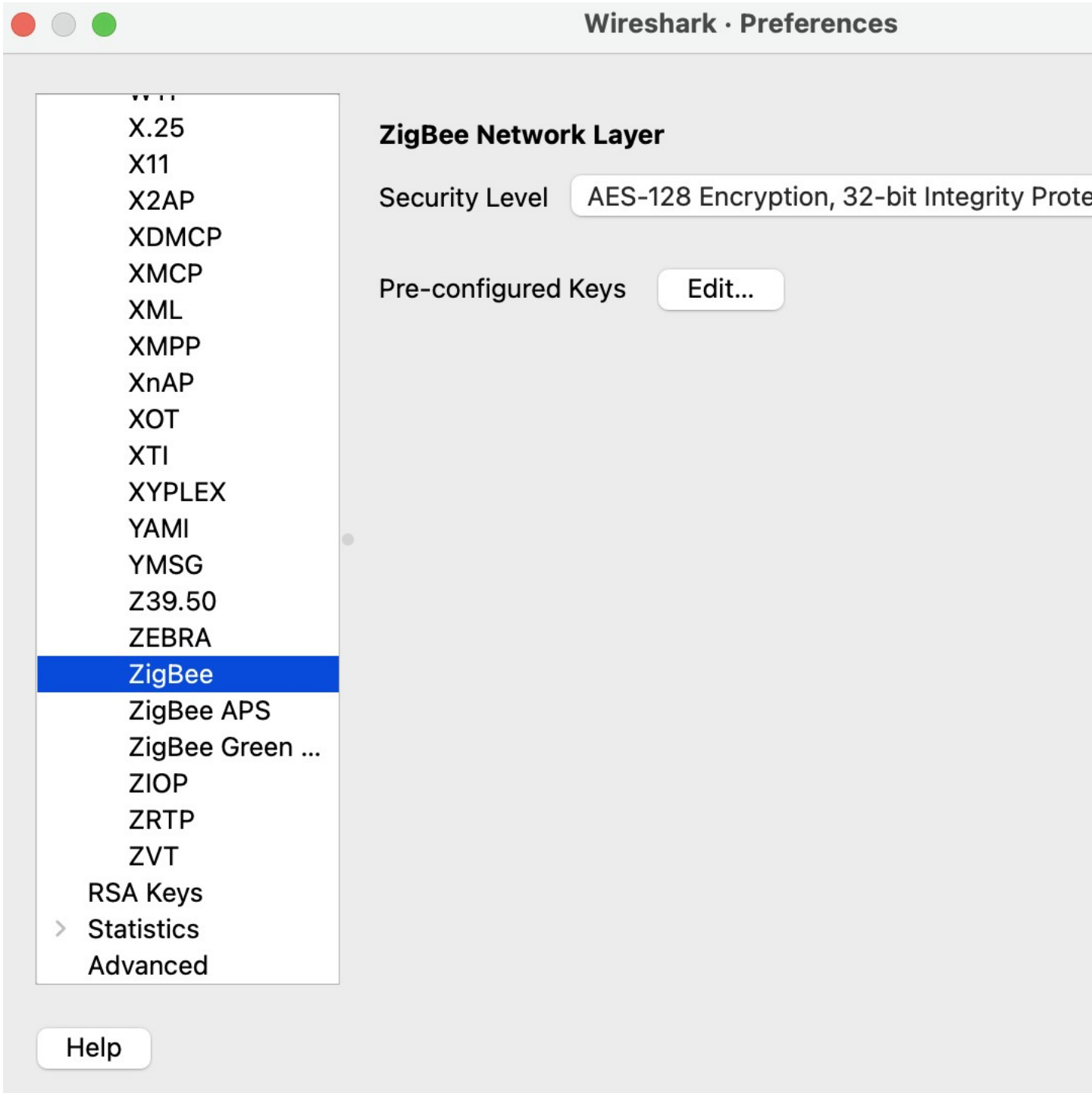
Folgen Sie den Schritten in Paket Sniffing mit Pyspinel, um den Schniffer aufzustellen.

Bitte beachten Sie, dass die im obigen Link angegebene Wireshark-Konfiguration für das Gewindeprotokoll bestimmt ist. Für Zigbee müssen Sie folgende Konfiguration vornehmen:

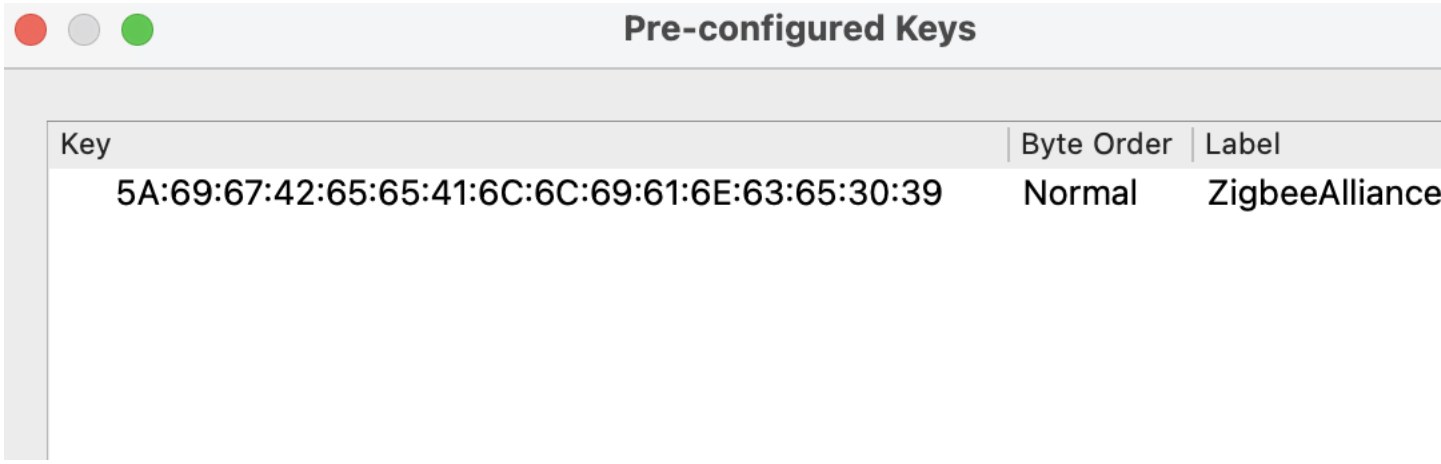
1. Gehen Sie zu den **Wireshark-Einstellungen** > **Protokolle** > **IEEE 802.15.4**, konfigurieren Sie die 802.15.4 als bellow:



2. Gehen Sie zu den **Wireshark-Einstellungen** > **Protokolle** > **ZigBee** :



3. Fügen Sie die vorkonfigurierten Tasten für die Paketentschlüsselung hinzu, der Standardschlüssel in den Beispielen ist *5A:69:67:42:65:65:41:6C:6C:69:61:6E:63:65:30:39* („ZigbeeAlliance09“)



Jetzt können Sie den Zigbee-Paketfluss in Wireshark überprüfen.

2.4.4 Entwesenmodus und Trace-Lackierung aktivieren-

Standardmäßig werden die Release-Versionenbibliotheken zum Bauen verwendet. Aktiv *ZB-DEBUG-MODE* Option, stattdessen Debug-Versionsbibliotheken zu verwenden, die mehr Protokolle für Debugging ausgeben.

Die Stack trace-Logging-Funktion gibt zusätzliche Protokolle aus, hier nehmen Sie das HA-on-off-light-light als Beispiel. Um die Protokollierung von Trace zu aktivieren, gehen Sie wie folgt vor:

1. Navigieren Sie zum Beispielverzeichnis und führen Sie den Befehl aus:

```
idf.py menuconfig
```

2. Gehen Sie zu **Component config** > **Zigbee** > **Zigbee Enable** > **Zigbee Debug Mode**, aktivieren Sie die `Zigbee Debug Mode`Option.
3. Anruf `esp_zb_set_trace_level_mask()`vor `esp_zb_init()`um die Spurenlage und Maske zu konfigurieren. Bitte beachten Sie die Masken in der Masken in der Lage.

```
#include "esp_zigbee_trace.h"

static void esp_zb_task(void *pvParameters)
{
#if CONFIG_ESP_ZB_TRACE_ENABLE
   esp_zb_set_trace_level_mask(ESP_ZB_TRACE_LEVEL_CRITICAL,
ESP_ZB_TRACE_SUBSYSTEM_MAC | ESP_ZB_TRACE_SUBSYSTEM_APP);
#endif

   /* initialize Zigbee stack */
   esp_zb_cfg_t zb_nwk_cfg = ESP_ZB_ZED_CONFIG();
   esp_zb_init(&zb_nwk_cfg);
   ......
}
```

4. Die Aktivierung der Spurprotokollierung erhöht die Codegröße. Möglicherweise müssen Sie erhöhen *Fabrik* Trenngröße im `partitions.csv`Datei:

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,        data, nvs,      0x9000,  0x6000,
phy_init,   data, phy,      0xf000,  0x1000,
factory,    app,  factory,  , 1200K,
zb_storage, data, fat,      , 16K,
```

```
zb_fct,     data, fat,      , 1K,
```

5. Übermäßiges Protokollieren kann zu Watchdog-Timeout für die Leerlaufaufgabe führen. Daher vorübergehend den untätigen Task-Watchdog deaktivieren:

`ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0` and `ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1`.

Schließlich bauen und führen Sie das Beispiel. Sie werden nun mehr Debugging-Logs in der Ausgabe sehen.

2.4.5 Schneidbarkeiten-

Es gibt bestimmte Behauptungen im SDK, die verhindern, dass der Stapel in bestimmte Situationen gerät. Typischerweise, Logs und Backtraces von *idf.py Monitor* helfen Sie, den Ort des Durchsetzungsproblems zu identifizieren, damit Sie herausfinden können, was mit der Umsetzung nicht stimmt.

Dieser Ansatz ist jedoch möglicherweise nicht wirksam, wenn die Behauptung in der Zigbee-Bibliothek aufgrund unvollständiger Debug-Informationen in der Bibliothek auftritt. In solchen Fällen können Sie beim Debuggen helfen, indem Sie uns die Protokolle zusammen mit der entsprechenden ELF-Datei teilen (es ist im Projekt *Bau* Ordner nach der Zusammenstellung, z.B., buil/on-off-light-bulb.elf).

Bitte erfassen Sie das gesamte Protokoll mit einem seriellen Werkzeug wie *Bildschirm* oder *minicom*. Der Output ähnelt folgendem:

```
^[[0;32mI (579) ESP_ZB_ON_OFF_LIGHT: Start network steering^[[0m
^[[0;32mI (2959) ESP_ZB_ON_OFF_LIGHT: Network steering was not successful
(status: ESP_FAIL)^[[0m

assert failed: esp_zb_app_signal_handler esp_zb_light.c:70 (false)
Core  0 register dump:
MEPC    : 0x4080063e  RA      : 0x408074c6  SP      : 0x4084f090  GP      :
0x4080d5a0
TP      : 0x4083e428  T0      : 0x37363534  T1      : 0x7271706f  T2      :
0x33323130
S0/FP   : 0x00000085  S1      : 0x00000001  A0      : 0x4084f0cc  A1      :
0x4080da59
A2      : 0x00000001  A3      : 0x00000029  A4      : 0x00000001  A5      :
0x40817000
A6      : 0x00000004  A7      : 0x76757473  S2      : 0x00000009  S3      :
0x4084f1e2
S4      : 0x4080da58  S5      : 0x00000000  S6      : 0x00000000  S7      :
0x00000000
S8      : 0x00000000  S9      : 0x00000000  S10     : 0x00000000  S11     :
0x00000000
T3      : 0x6e6d6c6b  T4      : 0x6a696867  T5      : 0x66656463  T6      :
0x62613938
MSTATUS : 0x00001881  MTVEC   : 0x40800001  MCAUSE  : 0x00000007  MTVAL   :
0x00000000
MHARTID : 0x00000000

Stack memory:
4084f090: 0x40809aa6 0x40809ad2 0x42073910 0x4080bdea 0x4080dd04 0x42073910
0x4080dce8 0x4207382c
4084f0b0: 0x4080dd14 0x4084f0c4 0x4080dd18 0x4207381c 0x4080da58 0x00003037
0x4084f520 0x65737361
4084f0d0: 0x66207472 0x656c6961 0x65203a64 0x7a5f7073 0x70615f62 0x69735f70
0x6c616e67 0x6e61685f
```

```
4084f0f0: 0x72656c64 0x70736520 0x5f627a5f 0x6867696c 0x3a632e74 0x28203037
0x736c6166 0x42002965
4084f110: 0x00000000 0xffffffff 0x4080f198 0x4084f368 0x00000008 0x4084f158
0x00000003 0x42004ce4
4084f130: 0x00000000 0x00000000 0x00000000 0x0000004b 0x4080f759 0x00000000
0x00000339 0x4204ba5e
4084f150: 0x420737d0 0x420734b4 0x00000042 0x4204be28 0x40850000 0x4084f1e8
0x4080f759 0x4201f83a
4084f170: 0x00000019 0x00000000 0x00000042 0x4201ebb6 0x00000000 0x00000000
0x0000004d 0x000000c0
4084f190: 0x00000019 0x00000000 0x00000000 0x42000000 0x4084fd94 0x40850000
0x0000004d 0x000000c0
4084f1b0: 0x00000019 0xffffffff 0x00000b8f 0x4200756e 0x00000000 0x00001800
0x40817944 0x40800a9c
4084f1d0: 0x00000008 0x4084f208 0x00000003 0x000000c0 0x00001800 0x00000008
0x00000019 0x40800b1c
4084f1f0: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0000004d 0x000000c0
4084f210: 0x00000019 0xffffffff 0x4084fd94 0x4200cc44 0x00000000 0x00000000
0x000000aa 0x408107d8
4084f230: 0x00000000 0x00000000 0x00000019 0x4203bc0c 0x00000001 0x00000001
0x00000001 0x4201f05a
4084f250: 0x00000000 0x4203bbb2 0x00190000 0x404f4d19 0x00000000 0x00000000
0x00000000 0x00000000
4084f270: 0x00000000 0x00000000 0x00000000 0x4203b852 0x00000000 0x00000000
0x4084fd74 0x4200ca7e
4084f290: 0x00000000 0x00000000 0x00000000 0x42007178 0x00000008 0x00000000
0x00000000 0x00000000
4084f2b0: 0x00000002 0x00000000 0x00000006 0x00000bb8 0x00000000 0x00000000
0x00000000 0x4080995a
4084f2d0: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000
4084f2f0: 0x00000000 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5 0xa5a5a5a5
0x00000154 0x4084f0e0
4084f310: 0x000000f4 0x4080e534 0x4080e534 0x4084f30c 0x4080e52c 0x00000014
0x4084fe34 0x4084fe34
4084f330: 0x4084f30c 0x00000000 0x00000005 0x4084e308 0x6267695a 0x6d5f6565
0x006e6961 0x00000000
4084f350: 0x00000000 0x4084f300 0x00000005 0x00000001 0x00000000 0x00000000
0x00000009 0x40817bf4
4084f370: 0x40817c5c 0x40817cc4 0x00000000 0x00000000 0x00000001 0x00000000
0x00000000 0x00000000
4084f390: 0x4205ef9e 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000
4084f3b0: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000
4084f3d0: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000
4084f3f0: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000
4084f410: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000
4084f430: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x00000000 0x00000000
4084f450: 0x00000000 0x00000000 0x00000000 0x40000000 0x00000054 0x00000000
0x4084f464 0x4084f30c
4084f470: 0x00000001 0x00000000 0x4084f47c 0xffffffff 0x4084f47c 0x4084f47c
0x00000000 0x4084f490
```

Hinweis

Wenn Sie auf Schwierigkeiten stoßen und Hilfe benötigen, zögern Sie bitte nicht, ein Github-Problem zu öffnen und die Spürer-Capture-Datei, Protokolle und die ELF-Datei einzuschließen. Alternativ kontaktieren Sie uns bitte über technische Anfragen.

3. Zigbee Zertifizierung-

3.1 Die Zertifizierung der Espressif Zigbee Plattform-

Espressif Systems ESP32-H2 wurde nach der entsprechenden Zertifizierung von CSA (Connectivity Standards Alliance) als „Zigbee-Compliant Platform“ zertifiziert. Siehe Zertifizierung. Daher wurde offiziell anerkannt, dass ESP32-H2 den Nutzern rigoros getestete Zigbee-Lösungen bietet.

Darüber hinaus hat ESP32-H2 das offizielle „Thread Interoperability Certificate“ erhalten, das sicherstellt, dass ESP32-H2 alle notwendigen Tests für Compliance und Interoperabilität mit marktführenden Thread-Referenz-Implementierungen bestanden hat.

3.2 Espressif Zigbee Produktionszertifizierung-

Bitte beachten Sie die ZUTHZUTH-Tools.

4. Überlegungen zur Produktion-

4.1 Espressif Zigbee Fertigungskonfiguration-

Espressif bietet auch ein Tool mfg-Tool für den Benutzer, um die fertigungsbezogene Konfiguration festzulegen.

Derzeit, kann Benutzer Config installieren Code, MAC-Adresse, Kanalmaske, Herstellungsname, Herstellungscode zu einem Flash Binärdatei in der Partitionstabelle `zb_fct`.

- **Code installieren**

Zigbee-Installationscodes, manchmal auch als „Installationscodes“ bezeichnet, dienen als Mittel für ein Gerät, um einem Zigbee-Netzwerk auf vernünftige Weise beizutreten.

Der Installationscode selbst ist ein zufälliger Wert, der zur Herstellungszeit auf dem Fügegerät installiert ist und verwendet wird, um den ersten Netzwerkschlüsseltransport vom zentralen Trust Center-Gerät des Zigbee-Netzwerks (der Koordinator) zum Fügegerät zu verschlüsseln.

Mit der Erstellung der Zölpe 3.0-Norm Ende 2016 müssen alle Zigbee-Geräte, die Netzwerke verbinden können (im Gegensatz zur Bildung), die Verwendung von Installationscodes während des Fügens unterstützen, da dies eine Voraussetzung für die Einhaltung von Zigbee 3.0 ist.

ESP Zigbee SDK bietet Tools zur Konfiguration in der Fertigungszeit und sicherheitsrelevanten APIs für Set/Add/Remove-Installationscode.

- **MAC Adresse**

Die MAC-Adresse, auch IEEE-Adresse, lange Adresse oder erweiterte Adresse genannt, ist eine 64-Bit-Nummer, die dieses Board eindeutig von allen anderen ZigBee-Boards der Welt identifiziert.

Jede Espressifs SOC schreibt bereits („brennen“) einmalige MAC-Adresse im eFuse-Speicher. Hier bietet ESP Zigbee SDK eine Möglichkeit, dass der Benutzer seine eigene MAC-Adresse während der Herstellungszeit konfigurieren kann.

5. User Guide_

This section provides a guide to the ESP Zigbee SDK, detailing common usage scenarios to support various stages of Zigbee product development and production.