elektor MAG
design > share > earn
Woche 01
SEIT 1961
DEZ. 2023 & JAN. 2024
ELEKTORMAGAZINE.DE

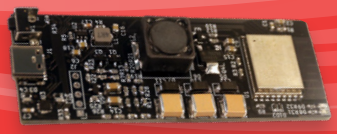
Gast-Ausgabe von

ESPRESSIF

Freigeschaltet: die Bonus-Ausgabe!

Home Automation mit **Espressif-Chips**

ADF, IDF und andere **SDKs**



ESP32 und **ChatGPT**

Bonus-Artikel für Profis, Maker und Studenten!

INELTEK

Design-In Expertise & Service

Ihr Partner für ESPRESSIF

Ineltek — Führender Franchise-Vertriebspartner von Espressif

Experience — Über 35 Jahre Erfahrung in der Halbleiterindustrie & Distribution

Latest News — Ineltek ist Up-To-Date bei Espressifs Produktneuheiten & Innovationen - Kontaktieren Sie uns!

Innovation — Ihnen fehlen Funktionen? Wir bringen Ihre Wünsche für nächste Produktgenerationen an

Application Support — Wir bieten spezialisierte, engagierte & zielgerichtete Kundenbetreuung für Ihre Projekte

Time to Market — Espressif & Inelteks FAE-Team unterstützen Sie von der Evaluierung bis zum Produkt

Information — Wir informieren Sie in Seminaren & Webinaren über Espressifs Lösungen und Wireless-Trends

Supply — Gängige Espressif SoCs, Module und EVKs ab Lager sofort verfügbar!

Kontaktieren Sie uns für Bauteile & Support!

www.ineltek.com

Wir bei Ineltek arbeiten als weltweiter agierender unabhängiger Distributor mit **Passion for Innovation** und unserem **Commitment to Service** an Sie.

Ineltek wurde 1987 gegründet und hat als technisch orientierter Halbleiter- und Design-In-Distributor bereits das Vertrauen tausender Industriekunden gewonnen. Zusammen mit Ihrem Team arbeiten wir gemeinsam daran, dass Ihre Produkte zur Markteinführung mit den bestmöglichen technischen Features ausgestattet sind, welche die Halbleiterindustrie aktuell bereithält.

Auf Jobsuche? Bewerben Sie sich!

Schicken Sie Ihre Bewerbung an personal@ineltek.com

- Vertriebsingenieur (m/w/d)
- Applikationsingenieur (m/w/d)
- Line Management / Marketing (m/w/d)

Folge uns!

INELTEK GmbH

Heidenheim, Wien, Castelfranco, London
Hamburg, München, Frankfurt, Dresden

IMPRESSUM

54. Jahrgang, Bonus-Ausgabe
Dezember 2023/Januar 2024
ISSN 0932-5468

Das Elektor Magazin wird 8 Mal im Jahr herausgegeben von
Elektor Verlag GmbH
Lukasstraße 1, 52070 Aachen (Deutschland)
Tel. +49 (0)241 95509190
www.elektor.de | www.elektormagazine.de

Für alle Ihre Fragen
service@elektor.de

Mitglied werden
www.elektormagazine.de/abo

Anzeigen
Büsra Kas
Tel. +49 (0)241 95509178
busra.kas@elektor.com
www.elektormagazine.de/mediadaten

Druck
Senefelder Misset, Mercuriusstraat 35
7006 RK Doetinchem (Niederlande)

Distribution
IPS Pressevertrieb GmbH, Carl-Zeiss-Straße 5
53340 Meckenheim (Deutschland)
Tel. +49 (0)2225 88010

Chefredakteur
Jens Nickel

Content Director
C. J. Abate

Herausgeber
Erik Jansen

EDITORIAL

Freigeschaltet: die Bonus-Ausgabe!

Im Laufe mehrerer Monate arbeitete das Elektor-Redaktionsteam eng mit Espressif zusammen, um ausführliche Artikel zu einer Vielzahl spannender Themen zu verfassen. Das Ergebnis dieser harten Arbeit ist eine von Espressif inspirierte Elektor-Ausgabe, die Anfang Dezember 2023 erscheint. Aber unsere Zusammenarbeit ist noch nicht zu Ende. Diese Elektor-Bonusausgabe ist vollgepackt mit zusätzlichen Projekten und Anleitungen, die Sie in den kommenden Monaten inspirieren sollen.
In typischer Elektor-Manier ist in dieser Bonus-Ausgabe für jeden etwas dabei, vom professionellen Ingenieur, der an der Entwicklung von AIoT-Produkten interessiert ist, bis hin zum Maker, der nach einem kreativen Wochenendprojekt sucht. Elektor und Espressif bieten Tipps für die Implementierung von ChatGPT mit Espressif SoCs, ein lustiges Dekatron-Projekt, Ratschläge für die Entwicklung von IoT-Anwendungen ohne Softwarekenntnisse und vieles mehr. Viel Spaß! Wenn Sie Ihr nächstes Espressif-Projekt starten, teilen Sie bitte Ihre Erfahrungen mit der Community auf der Online-Plattform Elektor Labs unter elektormagazine.de/labs - wir freuen uns darauf, zu sehen, was Sie schaffen!

C. J. Abate (Content Director, Elektor)

ESPRESSIF × elektor

INHALT

Entfesselung der Macht:

OpenAI für die ESP-BOX

Ein Leitfaden zur Verbindung von ChatGPT mit Espressif-SOCs

Von Ali Hassan Shah (Espressif)

Lassen Sie uns das Potenzial von ChatGPT mit der ESP-BOX erkunden, einer Entwicklungsplattform, die aus mit Mikrofonen ausgestatteten Boards und unter anderem einem umfangreichen Software-Ökosystem für Spracherkennung besteht. Dies bildet eine leistungsstarke Kombination, die IoT-Geräte auf die nächste Stufe heben kann!

Die Welt ist Zeuge einer technologischen Revolution, und OpenAI steht an der Spitze dieses Wandels. Eine der aufregendsten Innovationen ist ChatGPT, das die Verarbeitung natürlicher Sprache nutzt, um ansprechende und intuitive Benutzererfahrungen zu schaffen. Die Integration von OpenAI-APIs in IoT-Geräte hat eine Welt neuer Möglichkeiten eröffnet.

Dieser Artikel gliedert sich in drei Hauptabschnitte, die jeweils wesentliche Aspekte des Projekts behandeln. Der erste Abschnitt befasst sich mit der ESP-BOX-Entwicklungsplattform und liefert Details zu ihren Eigenschaften und Funktionalitäten. Der zweite Abschnitt ist eine Fallstudie, in der die Schritte beschrieben werden, die zum Aufbau des Projekts von Grund auf erforderlich waren. Der letzte Abschnitt enthält eine Liste zusätzlicher Informationsquellen, die der Leser nutzen kann, um sein Wissen und Verständnis des Projekts zu vertiefen.

ESP-BOX

Die ESP-BOX [1] ist eine AIoT-Entwicklungsplattform der nächsten Generation für die Entwicklungsboards ESP32-S3-BOX und ESP32-S3-BOX-Lite. Diese Boards basieren auf dem ESP32-S3-SoC [2] mit WLAN und Bluetooth 5 (BLE) und bieten eine flexible und anpassbare Lösung für die Entwicklung von AIoT-Anwendungen, die mit verschiedenen Sensoren, Controllern und Gateways versehen werden können.

Bild 1. Die ESP32-S3-BOX.

Die ESP-BOX (**Bild 1**) ist mit einer Vielzahl von Funktionen ausgestattet, die sie zu einer idealen AIoT-Entwicklungsplattform machen. Werfen wir einen genaueren Blick auf einige der wichtigsten Funktionen.

- **Fernfeld-Sprachinteraktion mit zwei Mikrofonen**: Die ESP-BOX unterstützt Fernfeld-Sprachinteraktion mit zwei Mikrofonen, so dass Benutzer mit ihren Geräten aus der Ferne interagieren können.
- **Offline-Sprachbefehlserkennung in chinesischer und englischer Sprache mit hoher Erkennungsrate**: Die ESP-BOX bietet eine Offline-Sprachbefehlserkennung in chinesischer und englischer Sprache mit hoher Erkennungsrate, was die Entwicklung sprachgesteuerter Geräte erleichtert.
- **Über 200 rekonfigurierbare Sprachbefehle in chinesischer und englischer Sprache**: Entwickler können mehr als 200 Sprachbefehle in chinesischer und englischer Sprache nach ihren Bedürfnissen (re-) konfigurieren.
- **Kontinuierliche Identifizierung und Wakeup-Unterbrechung**: Die ESP-BOX unterstützt kontinuierliche Identifikation und Wakeup-Interrupts, damit die Geräte immer bereit sind, Sprachbefehle zu empfangen.
- **Flexibles und wiederverwendbares GUI-Framework**: Die ESP-BOX verfügt über ein flexibles und wiederverwendbares GUI-Framework, das es Entwicklern ermöglicht, eigene Benutzeroberflächen für ihre Anwendungen zu erstellen.
- **Durchgängiges AIoT-Entwicklungsframework ESP-RainMaker**: Die ESP-BOX basiert auf Espressifs End-to-End AIoT/IoT-Entwicklungsframework ESP-RainMaker, das Entwicklern alle Tools zur Verfügung stellt, die sie zur Entwicklung leistungsstarker und intelligenter Geräte benötigen.
- **Pmod-kompatible Header unterstützen die Erweiterung von Peripheriemodulen**: Die ESP-BOX ist mit Pmod-kompatiblen Headern ausgestattet, so dass sie mit einer breiten Palette von Peripheriemodulen erweitert werden kann.

Fallstudie

Diese Fallstudie beschreibt den Entwicklungsprozess für einen sprachgesteuerten Chatbot, der die Kombination aus ESP-BOX und der OpenAI-API nutzt.
Das System ist in der Lage, Sprachbefehle von Benutzern zu empfangen, sie auf dem Bildschirm anzuzeigen und sie durch die OpenAI-API zu verarbeiten, um eine Antwort zu generieren. Die Antwort wird dann auf dem Bildschirm angezeigt und über die ESP-BOX abgespielt. Der schrittweise Arbeitsablauf erklärt detailliert, wie diese Technologien integriert werden können, um einen effizienten und effektiven sprachgesteuerten Chatbot zu erstellen (siehe **Bild 2** und **Bild 3**).

Einrichten der Entwicklungsumgebung

Das Einrichten einer geeigneten Umgebung und die Installation der richtigen Version sind entscheidend, um Fehler zu vermeiden. In dieser Demo werden wir ESP-IDF-Version 5.0 (Master Branch) verwenden. Wenn Sie eine Anleitung zur Einrichtung von ESP-IDF benötigen, finden Sie detaillierte Informationen im offiziellen IDF-Programmierhandbuch [3]. Zum Zeitpunkt der Erstellung dieses Artikels lautet der aktuelle IDF-Commit-Head *df9310ada2*.
Um das leistungsfähige Sprachmodell ChatGPT, das auf der GPT-3.5-Architektur basiert, nutzen zu können, müssen Sie zunächst

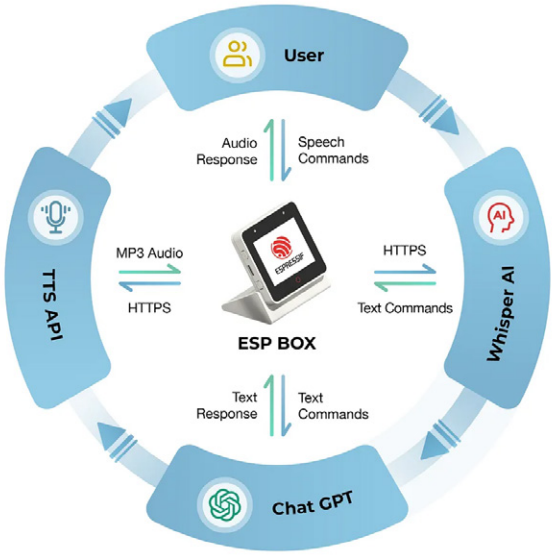


Bild 2. Arbeitsablauf der Fallstudie.

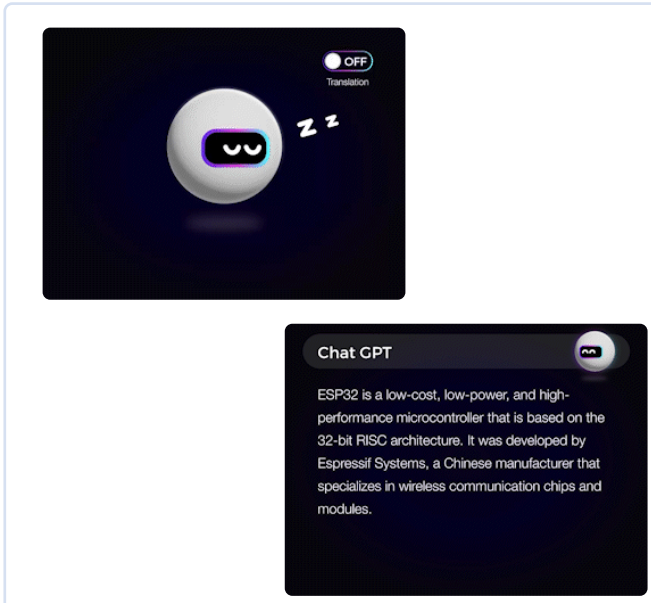


Bild 3. Demo der ESP-BOX als Chatbot.

einen sicheren API-Schlüssel erhalten. Dazu legt man ein Konto auf der OpenAI-Plattform [4] an. Mit einem API-Schlüssel erhalten Sie Zugang zu einer breiten Palette von Funktionen und Möglichkeiten, die an Ihre speziellen Bedürfnisse angepasst werden können, zum Beispiel die Verarbeitung und Generierung natürlicher Sprache, Textvervollständigung und Gesprächsmodellierung. Folgen Sie dem offiziellen API-Referenzlink [5]. Es versteht sich von selbst, dass die Wahrung der Vertraulichkeit und Sicherheit des API-Schlüssels von entscheidender Bedeutung ist, um einen unbefugten Zugriff auf das Benutzerkonto und die Daten des Benutzers zu verhindern.

Hinzufügen von Offline-Spracherkennungsfunktionen

Espressif Systems hat ein innovatives Spracherkennungssystem namens ESP-SR [6] entwickelt. Dieses Framework versetzt Geräte in die Lage, gesprochene Wörter und Phrasen zu erkennen, ohne auf externe Cloud-basierte Dienste angewiesen zu sein, was es zu einer idealen Lösung für Offline-Spracherkennungsanwendungen macht. Das ESP-SR-Framework besteht aus verschiedenen Modulen, darunter das Audio-Frontend (AFE), die Wake-Word-Engine (WakeNet), die Sprachbefehl-Worterkennung (MultiNet) und die Sprachsynthese (die derzeit nur die chinesische Sprache unterstützt). In der offiziellen Dokumentation [7] finden Sie weitere Informationen.

Einbindung der OpenAI-API

Die OpenAI-API bietet zahlreiche Funktionen, die Entwickler nutzen können, um ihre Anwendungen zu verbessern. In unserem Projekt haben wir die Audio-zu-Text- und Vervollständigungs-APIs verwendet und sie mit C-Code auf der Grundlage von ESP-IDF implementiert. Der folgende Abschnitt gibt einen kurzen Überblick über den von uns verwendeten Code.

Audio zu Text

Um Text aus Audio zu extrahieren, verwenden wir HTTPS und die Audio-API von OpenAI. Dazu wird folgender Code verwendet (**Listing 1**).

Bei diesem Code handelt es sich um eine Funktion namens `create_whisper_from_record()`, die einen Zeiger auf einen Puffer mit den Audiodaten und eine Ganzzahl `audio_len` mit der Länge der Audiodaten entgegennimmt. Diese Funktion sendet eine POST-Anfrage an den OpenAI-API-Endpoint, um die angegebenen Audiodaten zu transkribieren.

Die Funktion beginnt mit der Initialisierung der URL der OpenAI-API und dem Setzen der Autorisierungs-Header mit dem Bearer-Token `OPENAI_API_KEY`. Dann wird ein HTTP-Client konfiguriert und mit der bereitgestellten Konfiguration initialisiert, einschließlich URL, HTTP-Methode, Event-Handler, Puffergröße, Timeout und SSL-Zertifikat.

Danach werden der Inhaltstyp und die Begrenzungszeichenfolge für die mehrteilige Formulardatenanfrage als Header für den HTTP-Client festgelegt. Die Dateidaten und ihre Größe werden ebenfalls festgelegt, und eine Multipart/Form-Data-Anfrage wird erstellt. Der `form_data`-Puffer wird mit einer `malloc()`-Funktion zugewiesen, und die erforderlichen Informationen werden ihm hinzugefügt. Dazu gehören der Dateiname und der Content-Type der Audiodatei, der Inhalt der Datei und der Name des Modells, das für die Transkription verwendet werden soll.

Sobald die `form_data` erstellt ist, wird es als Post-Feld im HTTP-Client festgelegt, und der Client sendet die POST-Anfrage an den OpenAI-API-Endpunkt. Tritt während der Anfrage ein Fehler auf, protokolliert die Funktion eine Fehlermeldung. Schließlich wird der HTTP-Client aufgeräumt und die für `form_data` reservierten Ressourcen werden freigegeben.

Die Funktion gibt einen `esp_err_t`-Fehlercode zurück, der angibt, ob die HTTP-Anfrage erfolgreich war oder nicht.

Listing 1: Extrahieren von Text aus Audio.

```
esp_err_t create_whisper_request_from_record(uint8_t *audio, int audio_len)
{
      // Set the authorization headers
      char url[128] = "https://api.openai.com/v1/audio/transcriptions";
      char headers[256];
      snprintf(headers, sizeof(headers), "Bearer %s", OPENAI_API_KEY);
      // Configure the HTTP client
      esp_http_client_config_t config = {
         .url = url,
         .method = HTTP_METHOD_POST,
         .event_handler = response_handler,
         .buffer_size = MAX_HTTP_RECV_BUFFER,
         .timeout_ms = 60000,
         .crt_bundle_attach = esp_crt_bundle_attach,
      };
      // Initialize the HTTP client
      esp_http_client_handle_t client = esp_http_client_init(&config);
```

```
    // Set the headers
    esp_http_client_set_header(client, "Authorization", headers);
    // Set the content type and the boundary string
    char boundary[] = "boundary1234567890";
    char content_type[64];
    snprintf(content_type, sizeof(content_type), "multipart/form-data; boundary=%s", boundary);
    esp_http_client_set_header(client, "Content-Type", content_type);
    // Set the file data and size
    char *file_data = NULL;
    size_t file_size;
    file_data = (char *)audio;
    file_size = audio_len;
    // Build the multipart/form-data request
    char *form_data = (char *)malloc(MAX_HTTP_RECV_BUFFER);
    assert(form_data);
    ESP_LOGI(TAG, "Size of form_data buffer: %zu bytes", sizeof(*form_data) * MAX_HTTP_RECV_BUFFER);
    int form_data_len = 0;
    form_data_len += snprintf(form_data + form_data_len, MAX_HTTP_RECV_BUFFER - form_data_len,
                              "--%s\r\n"
                              "Content-Disposition: form-data; name=\"file\"; filename=\"%s\"\r\n"
                              "Content-Type: application/octet-stream\r\n"
                              "\r\n", boundary, get_file_format(file_type));
    ESP_LOGI(TAG, "form_data_len %d", form_data_len);
    ESP_LOGI(TAG, "form_data %s\n", form_data);
    // Append the audio file contents
    memcpy(form_data + form_data_len, file_data, file_size);
    form_data_len += file_size;
    ESP_LOGI(TAG, "Size of form_data: %zu", form_data_len);
    // Append the rest of the form-data
    form_data_len += snprintf(form_data + form_data_len, MAX_HTTP_RECV_BUFFER - form_data_len,
                              "\r\n"
                              "--%s\r\n"
                              "Content-Disposition: form-data; name=\"model\"\r\n"
                              "\r\n"
                              "whisper-1\r\n"
                              "--%s--\r\n", boundary, boundary);
    // Set the headers and post field
    esp_http_client_set_post_field(client, form_data, form_data_len);
    // Send the request
    esp_err_t err = esp_http_client_perform(client);
    if (err != ESP_OK) {
        ESP_LOGW(TAG, "HTTP POST request failed: %s\n", esp_err_to_name(err));
    }
    // Clean up client
    esp_http_client_cleanup(client);
    // Return error code
    return err;
}
```

Die Integration von OpenAIs ChatGPT mit Espressifs ESP-BOX eröffnet neue Möglichkeiten für die Entwicklung leistungsstarker und intelligenter IoT-Geräte.

Listing 2: HTTPS-Anfrage für den Chatabschluss.

```
esp_err_t create_chatgpt_request(const char *content)
{
        char url[128] = "https://api.openai.com/v1/chat/completions";
        char model[16] = "gpt-3.5-turbo";
        char headers[256];
        snprintf(headers, sizeof(headers), "Bearer %s", OPENAI_API_KEY);
        esp_http_client_config_t config = {
         .url = url,
         .method = HTTP_METHOD_POST,
         .event_handler = response_handler,
         .buffer_size = MAX_HTTP_RECV_BUFFER,
         .timeout_ms = 30000,
         .cert_pem = esp_crt_bundle_attach,
        };
        // Set the headers
        esp_http_client_handle_t client = esp_http_client_init(&config);
        esp_http_client_set_header(client, "Content-Type", "application/json");
        esp_http_client_set_header(client, "Authorization", headers);
        // Create JSON payload with model, max tokens, and content
        snprintf(json_payload, sizeof(json_payload), json_fmt, model, MAX_RESPONSE_TOKEN, content);
        esp_http_client_set_post_field(client, json_payload, strlen(json_payload));
        // Send the request
        esp_err_t err = esp_http_client_perform(client);
        if (err != ESP_OK) {
          ESP_LOGW(TAG, "HTTP POST request failed: %s\n", esp_err_to_name(err));
        }
        // Clean up client
        esp_http_client_cleanup(client);
        // Return error code
        return err;
}
```

Chat-Completion

Die OpenAI-API Chat Completion [8] wird verwendet, um HTTPS-Anfragen für den Chat-Abschluss zu senden. Dieser Prozess verwendet die Funktion `create_chatgpt_request()`, die einen Content-Parameter aufnimmt, der den Eingabetext für das GPT-3.5-Modell darstellt (**Listing 2**).

Die Funktion richtet zunächst die URL, das Modell und die Header ein, die für die HTTP-POST-Anfrage benötigt werden, und erstellt dann eine JSON-Nutzlast mit dem Modell, den maximalen Token und dem Inhalt. Als Nächstes setzt die Funktion den Header für die HTTP-Anforderung und legt die JSON-Nutzlast als Post-Feld für die Anforderung fest. Die HTTP-POST-Anforderung wird dann mit `esp_http_client_perform()` gesendet, und wenn die Anforderung fehlschlägt, wird eine Fehlermeldung protokolliert. Schließlich wird der HTTP-Client bereinigt und der Fehlercode zurückgegeben.

Behandlung der Antwort

Die Callback-Funktion `response_handler` wird von der ESP-IDF HTTP-Client-Bibliothek verwendet, um Ereignisse zu behandeln, die während eines HTTP-Request/Response-Austauschs auftreten (**Listing 3**).

Im Falle von `HTTP_EVENT_ON_DATA` weist die Funktion den eingehenden Daten Speicher zu, kopiert die Daten in den Puffer und erhöht die Variable `data_len` entsprechend. Dies geschieht, um die Antwortdaten zu akkumulieren.

Im Falle von `HTTP_EVENT_ON_FINISH` gibt die Funktion eine Meldung aus, die anzeigt, dass der HTTP-Austausch beendet ist, und ruft dann die Funktion `parsing_data()` auf, um die akkumulierten/rohen Daten zu verarbeiten. Anschließend gibt sie den Speicher frei und setzt die Variablen `data` und `data_len` auf null zurück. Schließlich meldet die Funktion `ESP_OK`, dass die Operation erfolgreich war.

Parsen von Rohdaten

Die JSON-`parser`-Komponente [9] wird verwendet, um die von der ChatGPT-API und der Whisper-AI-API über HTTPS erhaltene Rohantwort zu parsen. Zur Durchführung dieser Aufgabe wird eine Funktion verwendet, die die Parser-Komponente einsetzt. Weitere Details zu diesem Tool sind auf GitHub [10] zu finden (**Listing 4**).

Einbindung der TTS-API

Im Moment bietet OpenAI keinen öffentlichen Zugang zu ihrer Text-to-Speech-API (TTS). Es sind jedoch verschiedene andere TTS-APIs verfügbar, darunter Voicerss [11], TTSmaker [12] und TalkingGenie [13], die Sprache aus Texteingaben generieren können. Weitere Informationen über diese APIs sind auf ihren jeweiligen Websites finden. Für dieses Tutorial werden wir die TalkingGenie-API verwenden, die eine der besten Optionen für die Erzeugung hochwertiger, natürlich klingender Sprache sowohl in Englisch als auch in Chinesisch ist. Eine der einzigartigen Funktionen von TalkingGenie ist die Fähigkeit, gemischtsprachige Texte wie Chinesisch und Englisch nahtlos in

Listing 3: Behandlung von Ereignissen während eines HTTP-Anfrage/Antwort-Austauschs.

```
esp_err_t response_handler(esp_http_client_event_t *evt)
{
      static char *data = NULL; // Initialize data to NULL
      static int data_len = 0; // Initialize data to NULL
      switch (evt->event_id) {
      case HTTP_EVENT_ERROR:
          ESP_LOGI(TAG, "HTTP_EVENT_ERROR");
          break;
      case HTTP_EVENT_ON_CONNECTED:
          ESP_LOGI(TAG, "HTTP_EVENT_ON_CONNECTED");
          break;
      case HTTP_EVENT_HEADER_SENT:
          ESP_LOGI(TAG, "HTTP_EVENT_HEADER_SENT");
          break;
      case HTTP_EVENT_ON_HEADER:
          if (evt->data_len) {
              ESP_LOGI(TAG, "HTTP_EVENT_ON_HEADER");
              ESP_LOGI(TAG, "%.*s", evt->data_len, (char *)evt->data);
          }
          break;
      case HTTP_EVENT_ON_DATA:
          ESP_LOGI(TAG, "HTTP_EVENT_ON_DATA (%d +)%d\n", data_len, evt->data_len);
          ESP_LOGI(TAG, "Raw Response: data length: (%d +)%d: %.*s\n", data_len,
                  evt->data_len, evt->data_len, (char *)evt->data);

          // Allocate memory for the incoming data
          data = heap_caps_realloc(data, data_len + evt->data_len + 1,
                                   MALLOC_CAP_SPIRAM | MALLOC_CAP_8BIT);
          if (data == NULL) {
              ESP_LOGE(TAG, "data realloc failed");
              free(data);
              data = NULL;
              break;
          }
          memcpy(data + data_len, (char *)evt->data, evt->data_len);
          data_len += evt->data_len;
          data[data_len] = '\0';
          break;
      case HTTP_EVENT_ON_FINISH:
          ESP_LOGI(TAG, "HTTP_EVENT_ON_FINISH");
          if (data != NULL) {
              // Process the raw data
              parsing_data(data, strlen(data));
              // Free memory
              free(data);
              data = NULL;
              data_len = 0;
          }
          break;
      case HTTP_EVENT_DISCONNECTED:
          ESP_LOGI(TAG, "HTTP_EVENT_DISCONNECTED");
          break;
      default:
          break;
      }
      return ESP_OK;
}
```

Listing 4: Parsen der rohen Antwort, die von der ChatGPT-API und der Whisper-AI-API erhalten wurde.

```
void parse_response (const char *data, int len)
{
      jparse_ctx_t jctx;
      int ret = json_parse_start(&jctx, data, len);
      if (ret != OS_SUCCESS) {
          ESP_LOGE(TAG, "Parser failed");
          return;
      }
      printf("\n");
      int num_choices;
      /* Parsing Chat GPT response*/
      if (json_obj_get_array(&jctx, "choices", &num_choices) == OS_SUCCESS) {
          for (int i = 0; i < num_choices; i++) {
              if (json_arr_get_object(&jctx, i) == OS_SUCCESS &&
                  json_obj_get_object(&jctx, "message") == OS_SUCCESS &&
                  json_obj_get_string(&jctx, "content", message_content,
                 sizeof(message_content)) == OS_SUCCESS) {
                      ESP_LOGI(TAG, "ChatGPT message_content: %s\n", message_content);
                  }
                  json_arr_leave_object(&jctx);
          }
          json_obj_leave_array(&jctx);
      }
      /* Parsing Whisper AI response*/
      else if (json_obj_get_string(&jctx, "text", message_content,
              sizeof(message_content)) == OS_SUCCESS) {
          ESP_LOGI(TAG, "Whisper message_content: %s\n", message_content);
      } else if (json_obj_get_object(&jctx, "error") == OS_SUCCESS) {
          if (json_obj_get_string(&jctx, "type", message_content,
              sizeof(message_content)) == OS_SUCCESS) {
                  ESP_LOGE(TAG, "API returns an error: %s", message_content);
          }
      }
}
```

Sprache zu übersetzen. Dies kann ein wertvolles Werkzeug sein, um Inhalte zu erstellen, die ein globales Publikum ansprechen. Der folgende Code sendet eine von ChatGPT generierte Textantwort über HTTPS an die TalkingGenie-API und gibt dann die resultierende Sprache über eine ESP-BOX wieder (**Listing 5**).

Die Funktion `text_to_speech()` nimmt einen Nachrichtenstring und einen `AUDIO_CODECS_FORMAT`-Parameter als Eingabe. Der Message-String ist der Text, der in Sprache umgewandelt werden soll, während der `AUDIO_CODECS_FORMAT`-Parameter angibt, ob die Sprache im MP3- oder WAV-Format kodiert werden soll.

Die Funktion kodiert zunächst die Zeichenkette mit der Funktion `url_encode()`, die diverse ungültige Zeichen durch ihren ASCII-Code ersetzt, und wandelt diesen Code dann in eine zweistellige hexadezimale Darstellung um. Anschließend wird der Speicher für die resultierende kodierte Zeichenkette zugewiesen. Dann prüft sie den Parameter `AUDIO_CODECS_FORMAT` und setzt die entsprechende Zeichenkette für das Codec-Format, die in der `url` verwendet werden soll.

Als nächstes bestimmt die Funktion die Größe des `url`-Puffers, der für eine GET-Anfrage an die TalkingGenie API benötigt wird, und weist dementsprechend Speicher für den `url`-Puffer zu. Dann formatiert sie den `url`-String mit den entsprechenden Parametern, einschließlich der `voiceId` (die die zu verwendende Stimme angibt), dem kodierten Text, der Geschwindigkeit und Lautstärke der Sprache und dem Audiotyp (entweder MP3 oder WAV).

Die Funktion richtet dann eine `esp_http_client_config_t struct` mit der `url` und anderen Konfigurationsparametern ein, initialisiert ein `esp_http_client_handle_t` mit der `struct` und führt eine GET-Anfrage an die TalkingGenie API mit `esp_http_client_perform()` durch. Wenn die Anfrage erfolgreich ist, gibt die Funktion `ESP_OK` zurück, andernfalls einen Fehlercode. Schließlich gibt die Funktion den für den `url`-Puffer und die kodierte Nachricht reservierten Speicher frei, bereinigt den `esp_http_client_handle_t` und gibt den Fehlercode zurück.

Behandlung der TTS-Antwort

In ähnlicher Weise wird die Callback-Funktion `http_event_handler()` definiert, um Ereignisse zu behandeln, die während eines HTTP-Anfrage/Antwort-Austauschs auftreten (**Listing 6**).

Das Ereignis `HTTP_EVENT_ON_DATA` wird verwendet, um die vom Server empfangenen Audiodaten zu verarbeiten. Die Audiodaten werden in einem Puffer namens `record_audio_buffer` und die Gesamtlänge der empfangenen Audiodaten in einer Variablen namens `file_total_len`

Listing 5: Text to Speech.

```
esp_err_t text_to_speech_request(const char *message, AUDIO_CODECS_FORMAT code_format)
{
      int j = 0;
      size_t message_len = strlen(message);
      char *encoded_message;
      char *language_format_str, *voice_format_str, *codec_format_str;
      // Encode the message for URL transmission
      encoded_message = heap_caps_malloc((3 * message_len + 1), MALLOC_CAP_SPIRAM | MALLOC_CAP_8BIT);
      url_encode(message, encoded_message);
      // Determine the audio codec format
      if (AUDIO_CODECS_MP3 == code_format) {
          codec_format_str = "MP3";
      } else {
          codec_format_str = "WAV";
      }
      // Determine the required size of the URL bu
      int url_size = snprintf(NULL, 0,
          "https://dds.dui.ai/runtime/v1/synthesize?voiceId=%s&text=%s&speed=1&volume=%d&audiotype=%s", \
                               VOICE_ID, \
                               encoded_message, \
                               VOLUME, \
                               codec_format_str);
      // Allocate memory for the URL buffer
      char *url = heap_caps_malloc((url_size + 1), MALLOC_CAP_SPIRAM | MALLOC_CAP_8BIT);
      if (url == NULL) {
          ESP_LOGE(TAG, "Failed to allocate memory for URL");
          return ESP_ERR_NO_MEM;
      }
      // Format the URL string
      snprintf(url, url_size + 1,
          "https://dds.dui.ai/runtime/v1/synthesize?voiceId=%s&text=%s&speed=1&volume=%d&audiotype=%s", \
              VOICE_ID, \
              encoded_message, \
              VOLUME, \
              codec_format_str);
      // Configure the HTTP client
      esp_http_client_config_t config = {
          .url = url,
          .method = HTTP_METHOD_GET,
          .event_handler = http_event_handler,
          .buffer_size = MAX_FILE_SIZE,
          .buffer_size_tx = 4000,
          .timeout_ms = 30000,
          .crt_bundle_attach = esp_crt_bundle_attach,
      };
      // Initialize and perform the HTTP request
      esp_http_client_handle_t client = esp_http_client_init(&config);
      esp_err_t err = esp_http_client_perform(client);
      if (err != ESP_OK) {
          ESP_LOGE(TAG, "HTTP GET request failed: %s", esp_err_to_name(err));
      }
      // Free allocated memory and clean up the HT
      heap_caps_free(url);
      heap_caps_free(encoded_message);
      esp_http_client_cleanup(client);
      // Return the result of the function call
      return err;
}
```

gespeichert. Wenn die Gesamtlänge der empfangenen Audiodaten geringer ist als eine vordefinierte `MAX_FILE_SIZE`, werden die Daten in den `record_audio_buffer` kopiert.
Schließlich wird das Ereignis `HTTP_EVENT_ON_FINISH` verwendet, um das Ende der HTTP-Antwort zu behandeln. In diesem Fall wird der `record_audio_buffer` an eine Funktion namens `audio_player_play()` übergeben, die den Ton abspielt.

Display

Für die Darstellung im Display verwenden wir LVGL, eine Open-Source-Bibliothek für eingebettete Grafiken, die aufgrund ihrer leistungsstarken und visuell ansprechenden Funktionen und ihres geringen Speicherbedarfs immer beliebter wird. LVGL hat auch einen visuellen Drag-and-Drop-UI-Editor namens SquareLine-Studio [14] veröffentlicht. Es handelt sich um ein leistungsstarkes Werkzeug, mit dem Sie auf einfache Weise schöne GUIs für Ihre Anwendungen erstellen können. Um LVGL in Ihr Projekt zu integrieren, bietet Espressif Systems ein offizielles Paketmanager-Tool [15] an. Mit diesem Tool können Sie LVGL und die zugehörigen Portierungskomponenten direkt zu Ihrem Projekt hinzufügen, was Ihnen viel Zeit und Mühe erspart. Weitere Informationen finden Sie in den offiziellen Blogs [16] und Dokumentationen [17].

Intelligente IoT-Geräte erstellen

Die Integration von OpenAIs ChatGPT mit Espressifs ESP-BOX hat neue Möglichkeiten für die Erstellung von leistungsstarken und intelligenten IoT-Geräten eröffnet. Die ESP-BOX bietet eine flexible und anpassbare AIoT-Entwicklungsplattform mit Funktionen wie Fernfeld-Sprachinteraktion, Offline-Sprachbefehlserkennung und einem wiederverwendbaren GUI-Framework. Durch die Kombination dieser Funktionen mit der OpenAI API können Entwickler sprachgesteuerte Chatbots erstellen und die Benutzererfahrung in IoT-Anwendungen verbessern. Vergessen Sie nicht, das GitHub-Repository von Espressif [18] Systems [19] für weitere Open-Source-Demos zu ESP-IoT-Solution [20], ESP-SR und ESP-BOX zu besuchen. Der Quellcode für dieses Projekt ist auf GitHub [21] zu finden. Als Teil unserer Zukunftspläne streben wir die Einführung einer Komponente für die OpenAI-API an, die benutzerfreundliche Funktionen bieten wird.

RG – 230462-02

Haben Sie Fragen oder Kommentare?

Wenn Sie technische Fragen oder Kommentare zu diesem Artikel haben, wenden Sie sich bitte per E-Mail an den Autor unter ali.shah@espressif.com oder an das Elektor-Redaktionsteam unter redaktion@elektor.de.

Über den Autor

Ali Hassan Shah ist ein Embedded-Software-Ingenieur, der von einer tiefen Leidenschaft für IoT-Technologie erfüllt ist. Als geschätztes Mitglied des Application Engineering-Teams von Espressif Systems bringt er sein Fachwissen in die Aufgabe ein, Technologie für alle mühelos zugänglich und benutzerfreundlich zu machen, ganz nach dem Motto „Vereinfachung der Technologie für alle".

WEBLINKS

[1] ESP-Box: https://github.com/espressif/esp-box
[2] ESP32-S3 Produktauswahl: https://tinyurl.com/esp32s3prodsel
[3] IDF-Programmierhandbuch: https://docs.espressif.com/projects/esp-idf/en/release-v5.0/esp32/index.html
[4] OpenAI-Plattform: https://openai.com
[5] Offizielle API-Referenz : https://platform.openai.com/docs/api-reference
[6] ESP-SR: https://github.com/espressif/esp-sr
[7] Benutzerhandbuch ESP-SR: https://docs.espressif.com/projects/esp-sr/en/latest/esp32/index.html
[8] API für Chatabschluss: https://platform.openai.com/docs/api-reference/chat/create
[9] JSON-Parser: https://components.espressif.com/components/espressif/json_parser
[10] JSON-Parser auf GitHub: https://github.com/espressif/json_parser
[11] Voicerss: https://voicerss.org/api
[12] TTSmaker: https://ttsmaker.com/zh-cn
[13] TalkingGenie: https://talkinggenie.com
[14] SquareLine Studio: https://squareline.io
[15] Offizielles Paketmanager-Tool für LVGL: https://components.espressif.com/components/lvgl/lvgl
[16] Blog über LVGL: https://tinyurl.com/espfancyui
[17] LVGL-Dokumentation: https://docs.lvgl.io/master/index.html
[18] Espressif Systems: https://espressif.com/
[19] GitHub-Repository von Espressif Systems: https://github.com/orgs/espressif/repositories
[20] ESP-IoT-Solution: https://github.com/espressif/esp-iot-solution
[21] Quellcode für dieses Projekt: https://github.com/espressif/esp-box/tree/master/examples

Passendes Produkt

› **ESP32-S3-BOX-3**
www.elektor.de/20627

Listing 6: Behandlung der TTS-Antwort.

```
static esp_err_t http_event_handler(esp_http_client_event_t *evt)
{
      switch (evt->event_id) {
          // Handle errors that occur during the HTTP request
          case HTTP_EVENT_ERROR:
              ESP_LOGE(TAG, "HTTP_EVENT_ERROR");
              break;
      // Handle when the HTTP client is connected
      case HTTP_EVENT_ON_CONNECTED:
          ESP_LOGI(TAG, "HTTP_EVENT_ON_CONNECTED");
          break;
      // Handle when the header of the HTTP request is sent
      case HTTP_EVENT_HEADER_SENT:
          ESP_LOGI(TAG, "HTTP_EVENT_HEADER_SENT");
          break;
      // Handle when the header of the HTTP response is received
      case HTTP_EVENT_ON_HEADER:
          ESP_LOGI(TAG, "HTTP_EVENT_ON_HEADER");
          file_total_len = 0;
          break;
      // Handle when data is received in the HTTP response
      case HTTP_EVENT_ON_DATA:
          ESP_LOGI(TAG, "HTTP_EVENT_ON_DATA, len=%d", evt->data_len);
          if ((file_total_len + evt->data_len) < MAX_FILE_SIZE) {
              memcpy(record_audio_buffer + file_total_len, (char *)evt->data, evt->data_len);
              file_total_len += evt->data_len;
          }
          break;
      // Handle when the HTTP request finishes
      case HTTP_EVENT_ON_FINISH:
          ESP_LOGI(TAG, "HTTP_EVENT_ON_FINISH:%d, %d K", file_total_len, file_total_len / 1024);
          audio_player_play(record_audio_buffer, file_total_len);
          break;
      // Handle when the HTTP client is disconnected
      case HTTP_EVENT_DISCONNECTED:
          ESP_LOGI(TAG, "HTTP_EVENT_DISCONNECTED");
          break;
      // Handle when a redirection occurs in the HTTP request
      case HTTP_EVENT_REDIRECT:
          ESP_LOGI(TAG, "HTTP_EVENT_REDIRECT");
          break;
      }
      return ESP_OK;
}
```

ESP-Unlock

Zwei-Faktor-Authentifizierungs-Dongle mit dem ESP32-C3

Von Jakob Hasse, Espressif

Viele Online-Dienste bieten heutzutage eine Zwei-Faktor-Authentifizierung an. Die Smartphone-Authentifizierungs-App ist wahrscheinlich die beliebteste Wahl, aber ein Smartphone kann gestohlen werden, und oft werden private und geschäftliche Angelegenheiten vermischt. An dieser Stelle sind Hardware-Token in Form eines USB-Sticks nützlich. In diesem Projekt wird ein solcher Token auf der Basis eines günstigen ESP32-C3-Boards realisiert.

Stellen Sie sich vor, Sie sind Cyberkrimineller von Beruf und haben sich Zugang zum Benutzernamen und Passwort Ihres Opfers für einen beliebigen Online-Dienst verschafft. Nun versuchen Sie, sich einzuloggen. Es klappt! Aber dann fragt der Online-Dienst nach einem „Verifizierungscode". Und warum? Weil dieser Online-Dienst eine Zwei-Faktor-Authentifizierung anbietet und Ihr Opfer diese aktiviert hat. Die Zwei-Faktor-Authentifizierung - gängige Akronyme sind 2FA, TFA oder MFA - bedeutet im Grunde, dass man einen zusätzlichen Schritt benötigt, um sich gegenüber einem Online-Dienst zu authentifizieren. Man authentifiziert sich nicht nur mit etwas, das man kennt, sondern auch mit etwas, das man besitzt. In der Praxis beinhaltet dieser zusätzliche Schritt oft ein Hardware-Gerät, zu dem man Zugang hat. Ein gutes Beispiel ist das Abheben von Geld an einem Geldautomaten, für das man eine PIN (Wissen) und die dazugehörige Debitkarte (Besitz) benötigt. Alleinige Kenntnis der PIN oder der Besitz der Karte reichen nicht aus, um auf das Bankkonto des Opfers zuzugreifen

Aber nicht nur Banken verwenden die Zwei-Faktor-Authentifizierung. Auch viele Online-Dienste bieten heutzutage eine Zwei-Faktor-Authentifizierung an. Anders als Banken geben sie keine Bankkarten aus. Stattdessen kann eine Smartphone-Authentifizierungs-App oder ein handelsübliches elektronisches Gerät verwendet werden, das wir von nun an „Hardware-Token" nennen werden. Man muss sich immer noch an die normalen Anmeldedaten erinnern, um sich zu authentifizieren (Wissen), aber man muss auch eine Smartphone- Authentifizierungs-App oder Ihren Hardware-Token verwenden (Besitz).

Die Smartphone-Authentifizierungs-App ist wahrscheinlich die beliebteste Wahl, da praktisch jeder ein Smartphone besitzt. Aber wie sieht es mit einem Backup aus, wenn das Smartphone gestohlen wird? Wie sieht es mit der Trennung der beruflichen Konten von Ihrem privaten Telefon aus? Was ist, wenn das Telefon keine Energie mehr hat? Oder was ist, wenn es zu mühsam ist, das badezimmerkachelgroße Flaggschiff-Smartphone aus der Tasche zu fischen?

An dieser Stelle kommen Hardware-Token ins Spiel. Diese machen dich nicht nur unabhängig von einem Smartphone, sondern können auch klein und billig sein. Ein kleiner Hardware-Token kann in verschiedenen Situationen immer mitgeführt werden. Ein preiswerter Hardware-Token ermöglicht es mehr Menschen mit einem geringen Budget, ihn zu nutzen, und er ermöglicht es auch, mehrere Hardware-Token für verschiedene Zwecke oder zur gegenseitigen Unterstützung zu besitzen, ohne dass der Besitzer arm wird.

Wie Hardware-Token funktionieren

Sie stellen sich vielleicht die Frage: Wie kann der Online-Dienst den Hardware-Token von anderen Token unterscheiden? Die Antwort ist, dass der Hardware-Token tatsächlich „intelligent" ist, da er einen kleinen Mikroprozessor und etwas Speicher enthält. Bei der Einrichtung des Hardware-Tokens als Authentifikator für Ihren Online-Dienst erhält man vom Online-Dienst einen kryptografischen Schlüssel, der in den Speicher geschrieben wird (dazu später mehr). Der Schlüssel wird dann verwendet, um die Authentizität des Hardware-Tokens zu beweisen. Sie können sich ihn als ein weiteres Passwort vorstellen, das sich der Hardware-Token merkt, um sich beim Online-Dienst anzumelden. Die Art und Weise, wie man diese Art von Hardware-Token verwendet, ist jedoch der Verwendung eines physischen Schlüssels sehr ähnlich. Wenn man stattdessen eine Smartphone-Authentifizierungs-App verwendet, wird der Schlüssel irgendwo auf dem Smartphone gespeichert, aber die hier erläuterten Grundsätze sind dieselben.

Ein Standard für die Zwei-Faktor-Authentifizierung ist der weit verbreitete Standard für zeitbasierte Einmalpasswörter (Time-based One-time Passwords, TOTP), die aus einem kryptografischen Schlüssel und der aktuellen Uhrzeit erstellt werden. Der Schlüssel wird zwischen dem Online-Dienst und dem Hardware-Token ausgetauscht. Eine Hash-Funktion erstellt ein numerisches sechsstelliges OTP aus dem Schlüssel und der aktuellen Uhrzeit. Der Onlinedienst führt die gleiche Berechnung mit seiner eigenen Kopie des Schlüssels und der aktuellen Uhrzeit durch. Das vom Hardware-Token berechnete OTP wird an den Online-Dienst gesendet, der dieses OTP mit dem OTP vergleicht, das aus seiner eigenen Kopie des Schlüssels berechnet wurde. Nur wenn die beiden OTPs

übereinstimmen, wird der Benutzer erfolgreich authentifiziert. Bei der OTP-Berechnung wird eine Hash-Funktion verwendet, damit der Originalschlüssel bei der Übertragung vom Hardware-Token zum Online-Dienst nicht beschädigt wird. Weitere Informationen über den TOTP-Standard findet man unter [1].

ESP-Unlock Hardware-Token

Da Hardware-Token für die Zwei-Faktor-Authentifizierung so praktisch sind und der TOTP-Standard weit verbreitet und einfach zu implementieren ist, habe ich den „ESP-Unlock" entwickelt, ein Open-Source TOTP-kompatibler Hardware-Token für die Zwei-Faktor-Authentifizierung.
Ich habe den relativ preiswerten ESP32-C3-Chip für den Hardware-Token gewählt, weil ich mit dem Chip und dem *Espressif IoT Development Framework* (ESP-IDF) [2] vertraut bin, mit dem ein ESP32-C3 programmiert werden kann. Genauer gesagt, habe ich ein ESP32-C3-WROOM-02U-Modul gewählt, das den ESP32-C3 und andere nützliche Komponenten vereint. Der ESP32-C3 bietet ein paar sehr nützliche Funktionen:

- Einen integrierten USB-Seriell-Wandler, der die Kommunikation über USB ermöglicht, eine Schnittstelle, die auf praktisch jedem Computer verfügbar ist.
- Eine CPU mit kryptographischen Beschleunigern zur Berechnung des OTPs.
- Hardware, die Flash-Verschlüsselung zur Verschlüsselung des geheimen Schlüssels und sicheres Booten ermöglicht, um die Ausführung von nicht autorisiertem Code auf dem Hardware-Token zu verhindern.
- Das ESP32-C3-WROOM-02U Modul verfügt über einen Flash-Speicher zur Speicherung von Programmcode und Schlüssel.

Das ESP-IDF-Entwicklungsframework ermöglicht Flash-Verschlüsselung und sicheres Booten als leicht zugängliche Funktion auf der Hardware. Die einzigen fehlenden Teile waren eine Platine, um die gesamte Hardware zusammenzusetzen, und Software, um die einzelnen Teile miteinander zu verbinden. Das ESP-Unlock-Projekt kombiniert alle Teile, erstellt einen einfachen und billigen Open-Source-TOTP-Hardware-Token und eine Authentifizierungsanwendung für einen Host-Computer, die später besprochen wird. Wir besprechen nun die Architektur des gesamten Projekts, die Hardware und die Software, stellen dann einige Überlegungen zur Sicherheit an und erfahren schließlich, wie man ESP-Unlock benutzt.

Projekt-Architektur

Um das OTP zu berechnen und anzuzeigen, benötigt man den Hardware-Token ESP-Unlock selbst sowie einen Host-Computer, der über einen USB-A-Anschluss verfügt. Der ESP-Unlock speichert die Schlüssel, um OTPs für verschiedene Online-Dienste zu erzeugen. Der Host stellt über seine Echtzeituhr die aktuelle Wanduhrzeit zur Verfügung. Die Kommunikation zwischen den beiden wird als serielle Kommunikation über USB realisiert, unter Verwendung der USB-seriellen Peripherie des ESP32-C3. Die Berechnung eines OTPs erfordert vier Schritte (siehe **Bild 1**):

1. Der Host-Computer erhält die aktuelle Uhrzeit von seiner eigenen Echtzeituhr
2. Der Host-Computer fordert die OTP-Berechnung gegenüber dem ESP-Unlock an
3. Das ESP-Unlock verwendet den entsprechenden Schlüssel zur Berechnung des OTPs
4. Das ESP-Unlock sendet das OTP in einer Ergebnisnachricht zurück

Die Kommunikation (siehe Schritt 2 und Schritt 4 in Bild 1) zwischen dem Host und dem ESP-Unlock besteht aus einem recht einfachen Request-Response-Protokoll, wobei der Host immer als Initiator fungiert, während der ESP-Unlock immer nur Anfragen beantwortet. Alle Nachrichten werden als Klartext gesendet, um die Fehlersuche zu erleichtern. Ein Service-Name in der Anforderungsnachricht ist notwendig, da der Hardware-Token mit mehreren Online-Diensten arbeiten kann und einen Schlüssel pro Dienst benötigt. Die Beispielanforderung in Bild 1 lautet:

```
TOTP:github,1690975870<nl>
```

wobei `TOTP:` und das Zeilenumbruch-Zeichen am Ende Trennzeichen für das Parsing sind, `github` der Dienstname des zu wählenden Schlüssels ist und `1690975870` die aktuelle UNIX-Zeit ist, formatiert als Dezimalzahl. Die entsprechende Antwortnachricht in Bild 1 lautet:

```
TOTP:123456<nl>
```

wobei `TOTP:` und der Zeilenumbruch wieder Trennzeichen sind und `123456` das OTP ist. Weitere Informationen über dieses Protokoll und die zusätzlichen Nachrichten zur Auflistung von Dienstnamen und zum Hinzufügen neuer Sätze von Schlüssel- und Dienstnamen findet man im Repository des Projekts [3].

Hardware

Da bei der Hardware eine geringe Größe oberstes Gebot war, enthält die Platine des ESP-Unlocks nur die notwendigen Komponenten: ein ESP32-C3-WROOM-Modul (ohne Antenne), die entsprechende Bootstrap-Schaltung, die Stromversorgung (3,3 V von der USB-5-V-Versorgung), zwei LEDs und einen USB-A-Stecker zum Anschluss an einen Computer. Es gibt auch einen Platz für eine optionale Taste, die von zukünftigen Softwareversionen verwendet werden kann, um OTP-Berechnungen nur dann zu ermöglichen, wenn die Taste gedrückt wird, was die Sicherheit erhöht. Einschließlich des USB-Anschlusses misst die Hardware

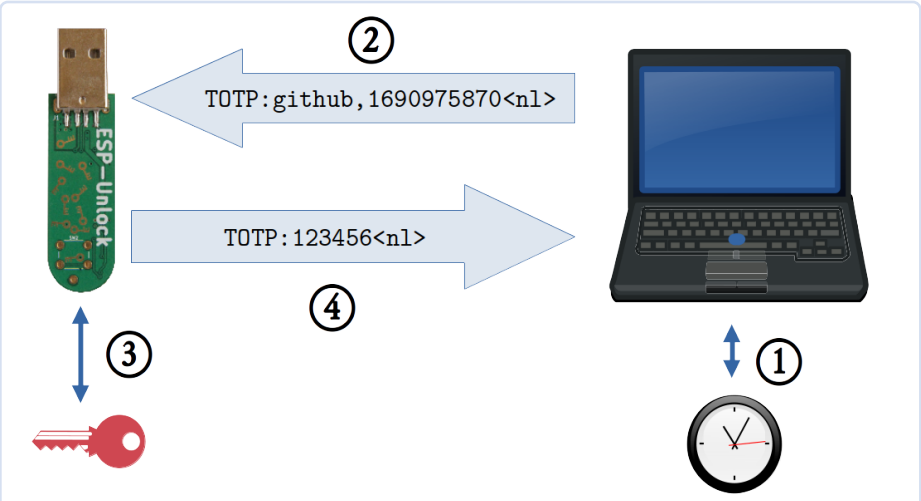


Bild 1. Kommunikation zwischen einem ESP-Unlock-Hardware-Token und dem Host-Computer.

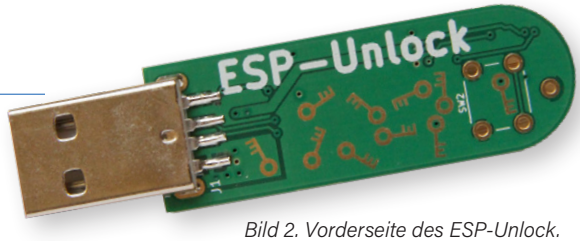


Bild 2. Vorderseite des ESP-Unlock.

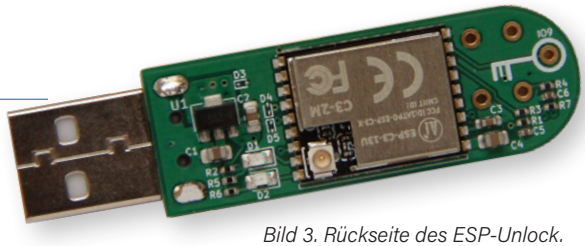


Bild 3. Rückseite des ESP-Unlock.

6×1,6 cm, was in etwa der Größe eines normalen Türschlüssels entspricht (**Bild 2** und **Bild 3**). Die Hardware wurde mit der Open-Source-Software KiCad entworfen, mit der Schaltungen und Platinenlayouts erstellt werden können. Das Platinenlayout von ESP-Unlock ist doppelseitig (**Bild 4**), da dies normalerweise die billigste Option ist. Alle Komponenten mit Ausnahme der bedrahteten Bauteile (J1) befinden sich auf einer Seite, so dass alle SMD-Komponenten einfach im Reflow-Lötverfahren gelötet werden können.

Software

Zwei Softwareanwendungen sind für dieses Projekt relevant: eine Firmware, die auf dem ESP-Unlock läuft und es „intelligent" macht, und die ESP-Unlock-Authentifizierungsanwendung, die auf dem Host-Computer läuft. Die Firmware auf dem ESP-Unlock ist in C++ geschrieben, die ESP-IDF [2] und ESP-IDF-C++ [4] verwendet. Sobald die ESP-Unlock-Firmware bootet, beginnt die Anwendung, auf OTP-Anfragen zu warten. Die beiden LEDs blinken einmal abwechselnd, wenn das Gerät bereit für Anfragen ist. Sobald eine OTP-Anfrage empfangen wurde, prüft die ESP-Unlock-Firmware, ob sie einen Schlüssel hat, der dem Namen in der Anfrage entspricht. Wenn ja, sucht sie den entsprechenden Schlüssel. Dann berechnet sie das OTP unter Verwendung des gerade gelesenen Schlüssels und der Zeit aus der OTP-Anfrage. Als Ergebnis wird eine Antwortnachricht mit dem OTP an den Host zurückgeschickt (vergleiche Bild 1). Beachten Sie, dass die gesamte OTP-Berechnung auf dem ESP-Unlock durchgeführt wird - der Schlüssel verlässt ihn nie.

Die ESP-Unlock-Authentifizierungsanwendung stellt eine einfache Benutzeroberfläche zur Verfügung und ist für die Koordinierung der Kommunikation mit dem ESP-Unlock verantwortlich. Die Benutzeroberfläche listet alle Dienste auf, für die Schlüssel auf dem aktuell verbundenen ESP-Unlock verfügbar sind, zeigt OTPs für die entsprechenden Dienste an und ermöglicht das Hinzufügen neuer Schlüssel. Sie ist nur für Linux geschrieben, aber die ESP-Unlock-Firmware kümmert sich nicht um das Betriebssystem des Hosts, so dass die Anwendung für jedes andere Betriebssystem neu implementiert oder portiert werden könnte.

Überlegungen zur Sicherheit

Bitte beachten Sie, dass es sich hierbei nicht um eine formale Sicherheitsanalyse handelt, da dies den Rahmen des Projekts in seinem derzeitigen Stadium sprengen würde. Sie soll auf bestehende Gefahren hinweisen und diese verständlich machen.

Die kritischen Daten auf dem ESP-Unlock sind die Schlüssel, die immer geheim gehalten werden sollten. Ein direkter Zugriff auf den Schlüssel kann durch die Verwendung einer Flash-Verschlüsselung verhindert werden. Aber auch die Software, die auf dem ESP-Unlock läuft, hat Zugriff auf die Schlüssel. Daher sollte nur autorisierte Software ausgeführt werden, um zu verhindern, dass unautorisierte Software die Schlüssel ausspäht. Secure Boot stellt sicher, dass nur autorisierter Code auf dem ESP32-C3 läuft. Wenn Secure Boot und Flash-Verschlüsselung aktiviert sind, kann selbst ein Angreifer, der physischen Zugriff auf den ESP-Unlock hat, die Schlüsseldaten nicht lesen.

Ein weiterer Faktor ist, dass die aktuelle Software es jedem, der physischen Zugriff auf den ESP-Unlock hat, erlaubt, OTPs zu generieren und auszulesen. Wenn Sie es verlieren und jemand anderes es findet, kann diese Person OTPs auf die gleiche Weise wie Sie selbst erzeugen.

Die auf dem ESP-Unlock laufende Firmware könnte Sicherheitslücken (Exploits) enthalten. Im schlimmsten Fall würde ein solcher Exploit einem Angreifer erlauben, seinen eigenen Code auf dem ESP-Unlock auszuführen. Eine Maßnahme gegen bestimmte Arten von Exploits ist die Aktivierung des Stack-Schutzes in der Firmware, der in der Konfiguration der Firmware eingestellt werden kann. Eine weitere Abhilfemaßnahme wäre eine sicherheitsorientierte Überprüfung des Codes, die nicht implementiert wurde, da es sich bei dem Code nur um einen frühen Prototyp handelt. Beachten Sie, dass Secure Boot nicht vor Code-Exploits schützt, sondern nur vor der Ausführung von nicht autorisiertem Code auf dem Hardware-Token. Es schützt nicht davor, dass sich autorisierter Code falsch verhält und die Ausführung von Random-Code ermöglicht. Angriffe auf den Code sind am wahrscheinlichsten mit physischem Zugang zum ESP-Unlock, aber auch Angriffe, die Software auf dem Host-Computer als Proxy verwenden, sollten ebenfalls in Betracht gezogen werden, sind aber aufgrund der zusätzlichen Schritte weniger wahrscheinlich.

Wenn ein Angreifer physischen Zugang zu einer ESP-Sperre erlangt, sollte der Besitzer eine neue ESP-Sperre einrichten und die geheimen Schlüssel in allen zugehörigen Online-Diensten so schnell wie möglich ändern. Solange der Angreifer jedoch nicht in den Besitz der Benutzerdaten (Benutzername und Passwort) gelangt, kann er sich nicht anmelden. Schließlich ist das OTP nur der zweite Faktor, und man sollte immer noch so sicher sein wie ohne jegliche Zwei-Faktor-Authentifizierung.

Verwendung des ESP-Unlock

Bevor ESP-Unlock OTPs generieren kann, muss der *ESP-Unlock Authenticator* installiert und der Schlüssel für die Zwei-Faktor-Authentifizierung hinzugefügt werden. Weitere Informationen über die Installation des *ESP-Unlock Authenticators* findet man in der Anleitung im Repository [5]. Um den Schlüssel hinzuzufügen, muss dieser zunächst von dem kompatiblen Online-Dienst abgerufen werden, den Sie verwenden möchten. Wie das funktioniert,

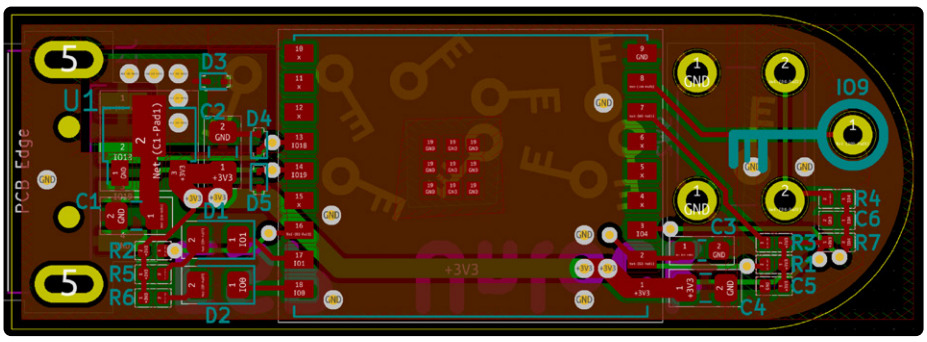


Bild 4. Ober- und Unterseite sowie der Bestückungsaufdruck der Platine.

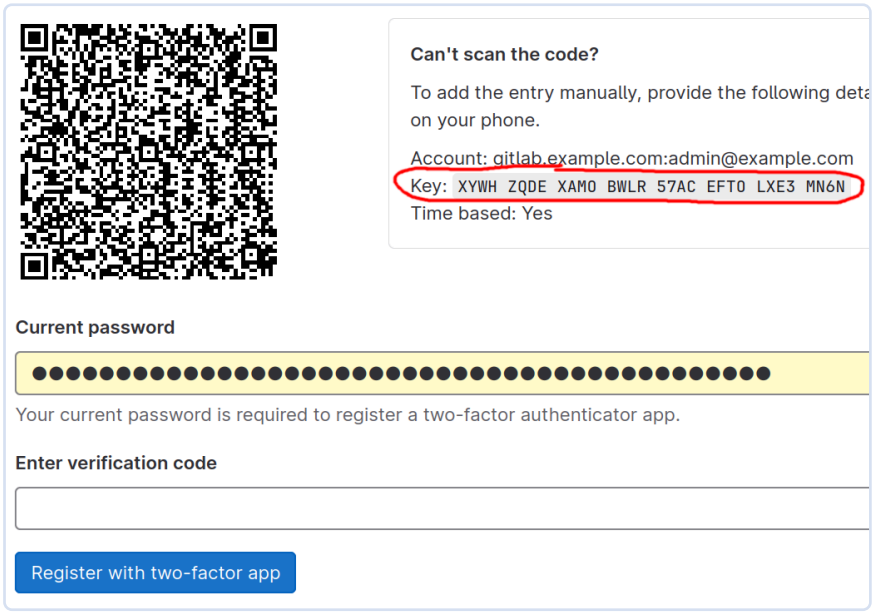


Bild 5. Base32-kodierter Schlüssel in GitLab (rot markiert).

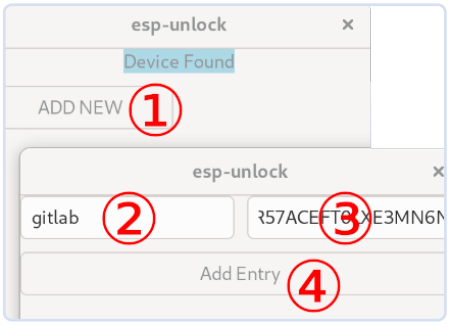


Bild 6. Hinzufügen eines neuen Schlüssels.

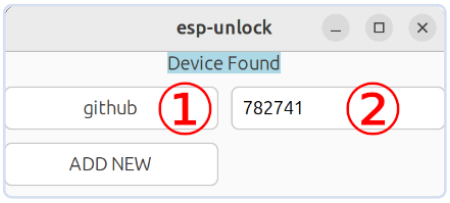


Bild 7. OTP anfordern.

hängt vom jeweiligen Online-Dienst ab, aber in der Regel werden durch die Einrichtung des Geräts oder der Anwendung für die Zwei-Faktor-Authentifizierung geführt. Normalerweise wird ein Schlüssel als QR-Code angezeigt, was bei der Verwendung einer Smartphone-Authentifizierungs-App praktisch ist. Für ESP-Unlock ist jedoch die base32-formatierte Textform erforderlich. GitLab beispielsweise bietet den base32-kodierten Schlüssel auf der Einrichtungsseite für die Zwei-Faktor-Authentifizierung neben dem QR-Code an (**Bild 5**). Sobald der ESP-Unlock an den Host-Computer angeschlossen ist, kann die Zwei-Faktor-Authentifizierung für einen neuen Dienst in der Authenticator-Anwendung mit den folgenden Schritten eingerichtet werden (vergleiche **Bild 6**):

- Klicken Sie auf *ADD NEW*
- Wählen Sie in dem sich öffnenden Fenster einen Service-Namen und geben Sie ihn ein
- Geben Sie den base32-formatierten Schlüssel manuell ein. Beachten Sie, dass alle Leerzeichen entfernt werden müssen, da sonst der Schlüssel auf dem ESP-Unlock nicht korrekt ist.
- Klicken Sie auf *ADD ENTRY*

Wann immer Sie nun ein OTP benötigen, stecken Sie den ESP-Unlock in den USB-Port des Hosts und starten Sie die ESP-Unlock-Authentifizierungsanwendung, die die Namen aller auf dem ESP-Unlock gespeicherten Schlüssel anzeigt und auf Knopfdruck (Schritt 1 in **Bild 7**) das entsprechende OTP anfordert und anzeigt (Schritt 2 in Bild 7).

Wenn Sie dieses Projekt interessant finden, können Sie es sich gerne ansehen oder ausprobieren. Besuchen Sie das *ESP-Unlock-Repository* [3] und das entsprechende *ESP-Unlock Authenticator Application Repository* [5]. Schaltpläne und Platinenlayout finden Sie im Hardware Repository [6], aber die ESP-Unlock-Hardware ist nicht notwendig! Es kann jedes beliebige ESP32-Entwicklungsboard verwendet werden, mit einer leicht veränderten Konfiguration (in einem solchen Fall lesen Sie bitte auch die README.md [3] des *ESP-Unlock Repository*). Jegliche Beiträge zum Projekt, wie Vorschläge, Hinweise, Verbesserungen und Fehlerberichte, sind sehr willkommen!

Übersetzung von Raphael Schermann (230559-02)

Haben Sie Fragen oder Kommentare?

Wenn Sie technische Fragen oder Kommentare zu diesem Artikel haben, können Sie sich gerne an den Autor unter jakob.hasse@mailbox.org oder an die Elektor-Redaktion unter redaktion@elektor.de wenden.

Über den Autor

Jakob Hasse hat einen Abschluss in Informatik und hat verschiedene Interessen. Während seines Studiums arbeitete er am Deutschen Zentrum für Luft- und Raumfahrt im Bereich Robotik. Später entwickelte er eine Leidenschaft für eingebettete Systeme und Open-Source-Software. Nachdem er als Ingenieur für eingebettete Linux-Systeme begonnen hatte, kam Jakob zu Espressif, um an deren FreeRTOS-basiertem Espressif IoT Development Framework zu arbeiten. Außerdem interessiert er sich für IT-Sicherheit, was auch der Hauptantrieb für das ESP-Unlock-Projekt ist.

Passende Produkte

- **ESP32-C3-WROOM-02**
  www.elektor.de/20695
- **Peter Dalmaris, KiCad 6 Like A Pro**
  - 2 Bücher, Paperback, englisch:
    www.elektor.de/20180
  - 2 E-Bücher, PDF, englisch:
    www.elektor.de/20181

WEBLINKS

[1] Standard für ein Zeitbasiertes One Time Password (TOTP): https://ietf.org/rfc/rfc6238.txt
[2] Espressif IoT Development Framework (ESP-IDF): https://github.com/espressif/esp-idf
[3] Das Repository des Projekts: https://github.com/0xjakob/esp-unlock
[4] ESP-IDF-C++: https://github.com/espressif/esp-idf-cxx
[5] Die ESP-Unlock-Authentifizierungsanwendung: https://github.com/0xjakob/esp-unlock-host-gui
[6] Hardware-Repository: https://github.com/0xjakob/esp-unlock-hardware

Dekatron

Ein Stück Geschichte erwacht zum Leben

Von Jeroen Domburg, Espressif

Genau wie in der Musik, in der alte und moderne Stile oft miteinander verschmelzen und außergewöhnliche Ergebnisse erzielen, passiert dies auch in der Elektronik: In diesem Artikel steuert ein hochmodernes ESP32-C3-Modul einen dekadischen Röhrenzähler aus den 1950er Jahren, kurz gesagt ein Dekatron, effektiv an und erweckt dieses Bauteil nach 70 Jahren wieder zum Leben!

Im Gegensatz zu einigen ehrwürdigen Instituten wie Elektor kann Espressif nicht auf eine sehr lange Geschichte zurückblicken. Das ist wahrscheinlich auch gut so, denn es ist zweifelhaft, dass es im Jahr 1961 einen großen Markt für IoT-WLAN-Chips gegeben hätte. Espressif wurde erst 2008 gegründet und brachte gut fünf Jahre später seine ersten Chips heraus: den ESP8089 und den ESP8266. Während der ESP8089 vor allem den OEMs vorbehalten war und beispielsweise Android-Tablets WLAN-Konnektivität verleihen sollte, ist der ESP8266 ein echter IoT-WLAN-Chip zum Selbermachen. Die Tatsache, dass selbst heute noch Leute Projekte mit dem ESP8266 beginnen, zeigt, wie wenig von all dem als „retro" angesehen werden kann. Wenn Sie selbst ein Projekt mit einem ESP8266-Chip starten wollen, empfehlen wir trotz aller Nostalgie, zum Beispiel einen ESP32-C3 zu verwenden, der ungefähr das Gleiche kostet und nach einem halben Jahrzehnt der Innovation ein viel besserer Chip ist.

Ich finde, dass es eine interessantere Geschichte wäre, eine Schnittstelle zwischen dem ganz Alten und dem ganz Neuen zu schaffen. In diesem speziellen Fall werde ich ein mehr als 50 Jahre altes Dekatron mit dem relativ neuen ESP32-C3 verbinden und mit dieser Combo die Nutzung unserer Internet-Bandbreite anzuzeigen.

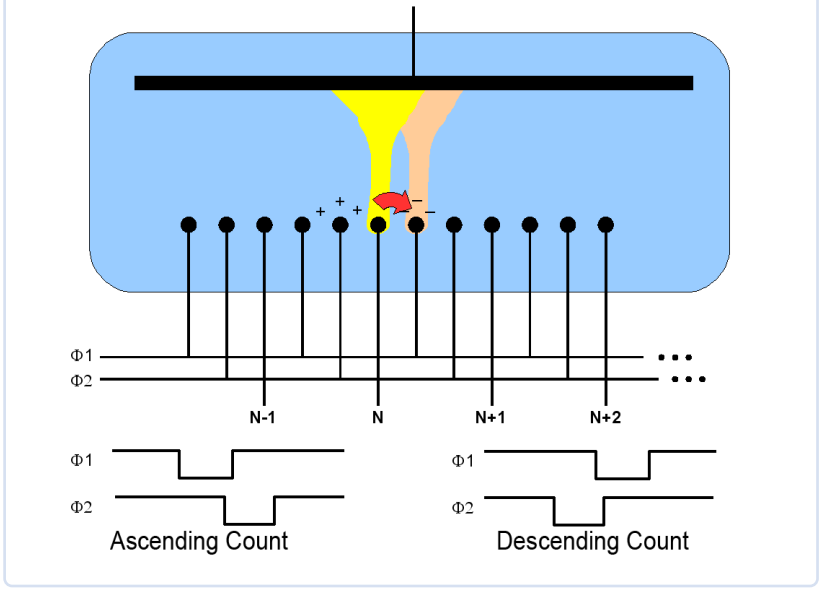


Bild 1. Grundlegendes Funktionsprinzip eines Dekatrons (Quelle: Wikimedia Commons)

Die Dekatron-Röhre

Falls Sie sich mit antiken elektronischen Geräten nicht auskennen: Ein Dekatron ist eine gasgefüllte Glasröhre, im Gegensatz zu einer Vakuumröhre, die in der Regel mit keinerlei Gas gefüllt ist. Bei dem Gas handelt es sich normalerweise um ein inertes Gas wie Neon, aber auch Gase wie Wasserstoff wurden verwendet. Wenn Sie bei „Neon" an die kleinen Neonbirnchen in den Ein- und Ausschaltknöpfen alter Geräte denken, ja, da liegen Sie gar nicht so weit daneben: Sie funktionieren nach einem ähnlichen Prinzip.

Ein Dekatron ist im Grunde ein Zähler mit einer visuellen Anzeige des Zählerstandes. Es funktioniert so: Im oberen Teil des Dekatrons befindet sich eine runde Anode. Um sie herum sind 30 Kathoden in Form von gleichmäßig verteilten Stiften angeordnet, wie man es prinzipiell in der Zeichnung in **Bild 1** und in den Schnappschüssen in **Bild 2** sehen kann. Zehn davon sind sogenannte Anzeige- oder Ausgangskathoden, die anderen zwanzig sind die Glimm- oder Leitkathoden, die in zwei Gruppen unterteilt sind: G1 und G2 (Φ1 und Φ2 in Bild 1). Jede Ausgangskathode hat zwei Führungskathoden, G1 auf der einen und G2 auf der anderen Seite. Alle G1-Kathoden sind parallel geschaltet, ebenso wie alle G2-Kathoden.

Beim Einschalten liegt zwischen der Anode und den Ausgangskathoden eine strombegrenzte Spannung von 400 V an, was höher ist als die Ionisationsspannung des Gases, so dass das Gas an einer dieser Kathoden ionisiert und glüht. Da dadurch die Anodenspannung unter die Ionisierungsspannung sinkt, leuchtet keine andere Elektrode mehr. Das Glühen kann auf die

benachbarten Ausgangskathoden „verschoben" werden, indem G1 und G2 nacheinander geerdet werden: Wenn zum Beispiel G1 geerdet ist und dann G2 geerdet wird, wandert die Ionisierung im Uhrzeigersinn, während sie im umgekehrten Fall nach links wandert.
Wozu diese ganzen Ionisierungstricks? Nun, dadurch sind diese Röhren aus Glas, Metall und Gas in der Lage, die G1/G2-Impulse zu zählen, und zwar bis zehn (daher **Deka**tron). Normalerweise sind einige oder alle Ausgangskathoden zu einzelnen Stiften an der Basis des Dekatrons geführt, so dass alle oder bestimmte Zählerstände erkennbar sind. Auf diese Weise können sie als Impulszähler und Frequenzteiler, aber auch als Speicherelemente wie beim berühmten Harwell-Computer aus dem Jahr 1951 verwendet werden [1]. Man beachte die Gegenwartsform im vorigen Satz: Der Computer ist als Museumsexponat immer noch in Betrieb!
Natürlich wurden Dekatrons in vielerlei Hinsicht obsolet, zunächst durch Zähler aus diskreten Transistoren. Als dann ICs kompliziertere Aufgaben ausführen konnten, übernahmen Chips wie der altehrwürdige Dekadenzähler CD4017 den Staffelstab. Heutzutage schließlich, in der Zeit billiger Mikrocontroller und FPGAs, wird die Funktion des Dekatrons in der Regel durch nur wenige Zeilen Code oder Hardwarebeschreibungssprache erreicht.
Ein Nachteil all dieser neumodischen Zähltechniken ist jedoch, dass sie im Gegensatz zum Dekatron keine visuelle Ausgabe liefern. Sicher, man kann LEDs an den CD4017 anschließen (wie es vermutlich viele Leser in der Vergangenheit getan haben) oder einen Spinner auf das LCD setzen, das man an den schicken neuen Mikrocontroller angeschlossen hat, aber das Neonglühen des Dekatrons hat etwas, das nicht wirklich durch Pixel oder lichtemittierende Halbleiter ersetzt werden kann.

Hoch spannend: die Versorgung

Darin liegt auch ein kleines Problem: Der ESP32-C3 ist ideal geeignet, um diese Pixel oder LEDs zum Leuchten zu bringen, aber nicht so sehr, um mit 70 Jahre alten Glaswaren zusammenzuarbeiten. Die 3,3 V, die der ESP32-C3 an seinen I/O-Pins liefert, schaden dem Dekatron zwar nicht, könnten aber genauso gut auch nicht existieren. Mit anderen Worten, wir müssen nicht nur einen Weg finden, die 400 V zu erzeugen, sondern auch mehrere solcher hohen Spannungen schalten.
Beginnen wir mit den 400 V. Am einfachsten wäre der Einsatz eines Spannungsverdopplers für die Netzspannung. Für dieses Gerät wollte ich jedoch nicht wirklich Netzspannung verwenden, denn die Sicherheitsvorkehrungen wären dann doch ein bisschen übertrieben. Ich habe mich stattdessen für eine moderne USB-C-Verbindung entschieden. Unter all den Laptop-Netzteilen, Handy-Adaptern und den Ladegeräten, die man mit irgendeinem elektronischen Gerät bekommen hat (aber man kann sich beim besten Willen nie mehr erinnern, welches), gibt es genug Netzteile mit diesem Stecker. Außerdem ist USB-PD heute ein gängiger Standard, der es ermöglicht, höhere Spannungen als die normalerweise an einem USB-Anschluss verfügbaren 5 V anzufordern. Das hat sich in diesem Fall als nützlich erwiesen.
Wie erzeugt man nun diese 400 V? Dazu gibt es mehrere Möglichkeiten. Die einfachste davon ist der Einsatz eines Transformators: Man kann die niedrige Gleich- in eine Wechselspannung umwandeln, sie in die Niederspannungswicklung eines Transformators einspeisen und am anderen Ende die Hochspannung abholen. Eine andere Möglichkeit ist ein Sperrwandler: Man lädt das Magnetfeld auf, indem man eine Spannung an die Niederspannungsseite anlegt, nimmt die Spannung weg und lässt das zusammenbrechende Magnetfeld von der Hochspannungsseite auffangen, und schon hat man das Ergebnis. Das Problem bei diesen beiden Lösungen ist, dass Miniaturtransformatoren eher ein Nischenprodukt sind und das Risiko besteht, dass der Entwurf nicht nachgebaut werden kann, wenn der gewählte Transformator nicht mehr auf Lager ist. Ich erwarte zwar nicht, dass diese Schaltung millionenfach hergestellt wird, aber ich möchte, dass sie funktioniert und lange repariert werden kann. Also sind Nicht-Standard-Komponenten nicht wirklich etwas, was ich hier möchte.

Die Lösung

Stattdessen wähle ich eine andere Option: einen Aufwärtswandler. Ein Aufwärtswandler wird normalerweise verwendet, um niedrige Spannungen in *etwas* höhere Spannungen umzuwandeln, zum Beispiel, um eine 5-V-Logik mit zwei AA-Batterien zu betreiben. Sie haben den Nachteil, dass das drastische Erhöhen einer niedrigen Spannung viel problematischer ist - insbesondere muss das Schaltelement, das die niedrige Spannung schaltet, auch der hohen Ausgangsspannung standhalten, und der maximale Faktor, um den Sie Ihre Eingangsspannung erhöhen können, wird durch Dinge wie den Gleichstromwiderstand der verwendeten Induktivität begrenzt. Ein Trick, um dies zu umgehen, ist ein Spannungsverdoppler hinter dem Aufwärtswandler. Auf diese Weise muss der Aufwärtswandler nur die Hälfte der erforderlichen 400 V erzeugen. Das ist keine überragende Idee von mir, sondern stammt aus einer bewährten Dekatron-Aufwärtswandlerschaltung [2]. Der andere Trick besteht darin, mit einer höheren Spannung zu beginnen: In der

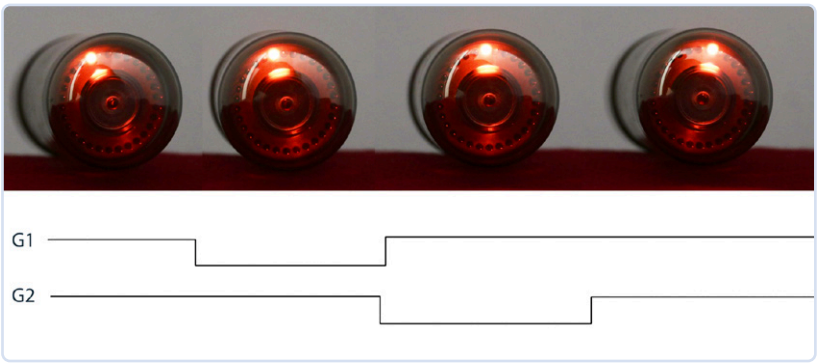
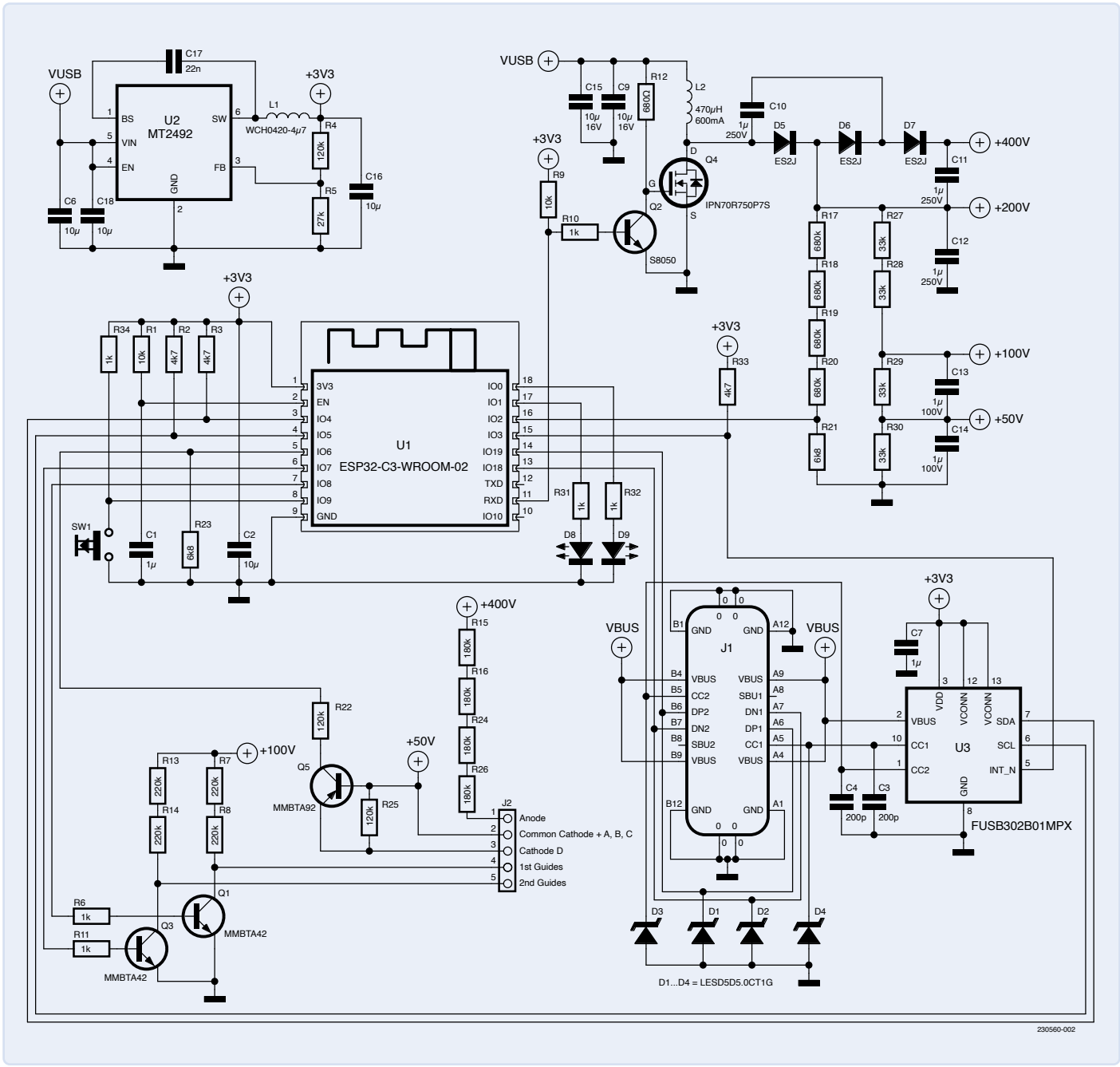


Bild 2. Das Dekatron in Aktion bei einer Zählung, mit den entsprechenden Signalen auf G1 und G2.



▲ *Bild 3. Schaltung dieses Retro-Projekts.*

Regel können PD-kompatible USB-C-Ladegeräte 5 V liefern, aber auch 9 V, 15 V oder 20 V, wenn man es wünscht. Das hat zur Folge, dass der Aufwärtswandler nicht so hart arbeiten muss, um die Eingangsspannung auf den gewünschten Wert zu erhöhen. Eine andere Lösung wäre ein zweiter Aufwärtswandler, der die Versorgungsspannung ein wenig anhebt, bevor sie ganz auf 400 V erhöht wird. Um ehrlich zu sein: Ich habe mich für den USB-PD-Weg entschieden, da ich ohnehin die Fähigkeiten von USB-PD abklopfen wollte.

Die Einzelheiten der Hochspannungseinheit sind oben rechts in **Bild 3** dargestellt. Die Spannung vom USB-Anschluss J1 wird zunächst von zwei Keramik-Kondensatoren stabilisiert. Für die Anhebung der Spannung sorgen zunächst einmal L2 und Q4: Wenn Q4 leitend ist, baut L2 ein Magnetfeld auf, und wenn der Transistor offen ist, wird der Strom aus dem kollabierenden Feld über die Diodenkette D5...D7, die gleichrichtet und die Spannung verdoppelt, in C11 und C12 abgeleitet. Daraus ergeben sich zwei Gleichspannungen, eine von 200 V und eine von 400 V.

Q4 ist eine Besonderheit. Der Schalttransistor muss den hohen Spannungen, die L2 erzeugt, standhalten können. Leider sind MOSFETs, die dazu in der Lage sind, in der Regel keine Logik-Pegel-MOSFETs, so dass Q2 eingesetzt wird, um das 3,3-V-PWM-Signal vom ESP32-C3 so umzuwandeln, dass es den MOSFET schalten kann. Das Tastverhältnis des PWM-Signals wird durch Messung der 200-V-Schiene geregelt. Dazu wird die 200-V-Spannung mit Hilfe von R17 bis R21 so weit verringert, dass der interne ADC des ESP32 sie verarbeiten kann.

Neben den 400 V benötigen wir auch einige niedrigere Spannungen, die aus der der 200-V-Spannung mit einem Widerstandsteiler (R27...R30) erzeugt werden. Beachten Sie, dass im Schaltplan mehrere Widerstände in Reihe geschaltet sind, wo doch ein einziger genügen würde, oder nicht? Der Grund dafür ist, dass ich 0603-Bauteile verwenden wollte, und diese sind nur für eine Spannung

von 75 V oder so geeignet. Wenn man sie in Reihe schaltet, ist der Spannungsabfall an jedem einzelnen Widerstand geringer, was bedeutet, dass keine plötzlichen Lichtbögen entstehen: Hochspannungsüberschläge wären zwar ein toller Indikator für ein tolles Ereignis, aber sie sind nicht reproduzierbar und würden zudem den Raum mit verbrannter Elektronik verpesten.

Auf der Seite des Dekatrons werden die 400 V, wie im Schaltplan dargestellt, über einen 720-kΩ-Widerstand in die Anode eingespeist. Diese Widerstandskette, bestehend aus R15, R16, R24 und R26, sorgt ebenfalls für die vom Datenblatt [3] vorgeschlagene Begrenzung des Anodenstroms auf etwa 310 µA. Die G1- und G2-Signale werden durch zwei Hochspannungs-NPN-Transistoren (Q1 und Q3) im Pegel verschoben, und eine der Ausgangskathoden ist an den PNP-Transistor Q5 angeschlossen, der den ESP32-C3 darauf aufmerksam macht, wenn „das Glühen" diese Kathode passiert.

Das ESP32-C3-Modul

Der ESP32-C3 selbst ist in der Mitte des Schaltplans in Bild 3 zu sehen. Da ich ein ESP32-C3-WROOM02-Modul verwende, ist der größte Teil der benötigten Hardware bereits enthalten: Flash, HF-Anpassungsnetzwerke und die Antenne. Das Einzige außerhalb des Moduls sind zwei Anzeige-LEDs, ein Druckknopf und ein RC-Netzwerk, um einen Power-On-Reset zu erzeugen. Wenn Sie (wie ich) KiCad für Ihren Platinenentwurf verwenden: Beachten Sie, dass Espressif ein kostenloses Repository mit Symbolen und Footprints für alle Module und Chips unterhält [4].

Der USB-C-Anschluss (J1) hat zwei Funktionen: Wenn ein USB-PD-Netzteil angeschlossen ist, versorgt er alles mit Strom, aber man kann ihn auch mit dem Computer verbinden. In diesem Fall wird das Dekatron nicht mit Strom versorgt, da die vorhandene 5-V-Versorgung dafür nicht ausreicht, aber er ermöglicht das Programmieren des Codes im Flashspeicher des ESP32-C3. Um die USB-PD-Verhandlung durchzuführen, wird ein USB-C-Controller mit der Bezeichnung FUSB302 verwendet. Der FUSB302 ist über I²C mit dem ESP32-C3 verbunden, so dass der ESP32-C3 mit der Stromversorgungsabteilung kommunizieren kann. Um die an J1 ankommende Spannung in etwas umzuwandeln, mit dem der ESP32-C3 betrieben werden kann, habe ich einen synchronen Abwärtswandler verwendet, nämlich den MT2492 von Aerosemi. Dieser kleine Chip arbeitet mit bis zu 16 V, und die Tatsache, dass er ein Schaltwandler ist, bedeutet, dass er schön kühl bleibt, während er die 3,3-V-Schiene für den ESP32-C3 erzeugt.

Platinenlayout

Bild 4 zeigt den Platinenentwurf. Ich habe die Abmessungen so gewählt, dass die Platine genauso breit ist wie das Dekatron selbst, so dass sich die Elektronik unter der Röhre verstecken kann. Das Entflechten des Platinenlayouts ist etwas anspruchsvoller, als man es gewohnt ist, denn wegen der hohen Spannungen musste ich stets die Kriechstrecken zwischen den Leiterbahnen im Auge behalten. Generell fordert die hohe Spannung physikalisch größere Bauteile. Ich habe es trotzdem geschafft, alles auf einer zweilagigen Platine unterzubringen, sogar mit einer weitgehend intakten Massefläche. Ich entschied mich, die Platine mit einer schönen schwarzen Lötmaske zu versehen, damit sie, wie in **Bild 5** zu sehen, nicht so auffällt und dem Dekatron die Show stiehlt.

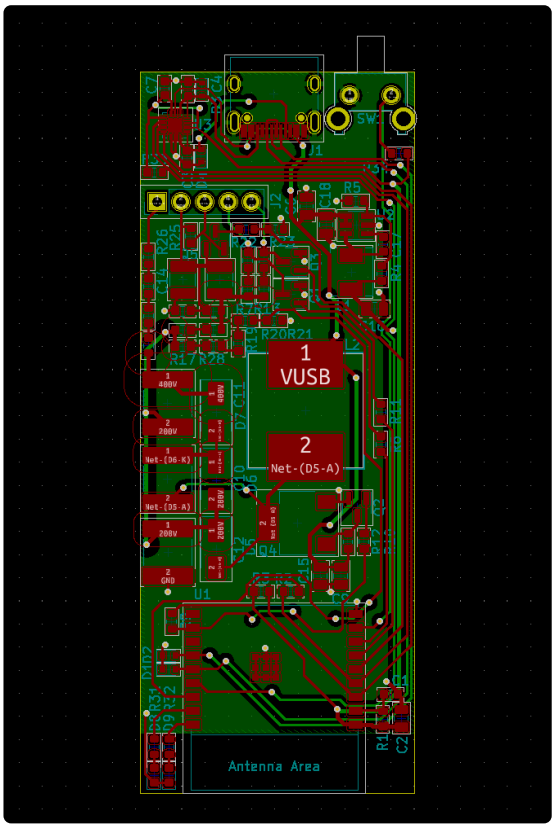


Bild 4. Ein Platinenlayout für das ESP32-C3-basierte Dekatron-Interface.

Software-Implementierung

Nachdem ich die Hardware gebaut hatte, begann ich mit der Programmierung. Die Software muss zunächst einmal dafür sorgen, dass die Hardware mit der richtigen Spannung versorgt wird, und dazu muss sie einen USB-PD-Stack implementieren, der mit dem angeschlossenen Netzteil kommuniziert. Ich habe dafür eine bestehende Bibliothek verwendet, die gut funktioniert [5]. Die Hardware selbst kann einen ziemlich großen Spannungsbereich nutzen. Die Software versucht, 12 V von der Stromversorgung anzufordern, aber wenn das nicht gelingt, akzeptiert sie auch 9 V oder 15 V. Damit das Dekatron auch zu seiner Hochspannung kommt, misst der ESP32-C3 die aktuelle Spannung auf der 200-V-Schiene und passt das PWM-Tastverhältnis für den Aufwärtswandler entsprechend an. Zweitens muss die Software die aktuell

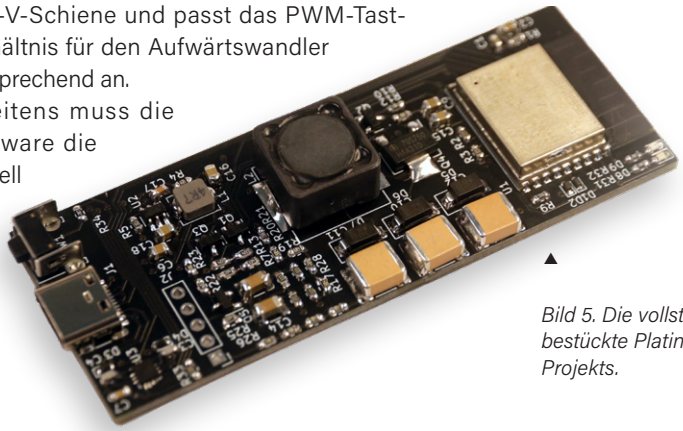


Bild 5. Die vollständig bestückte Platine dieses Projekts.

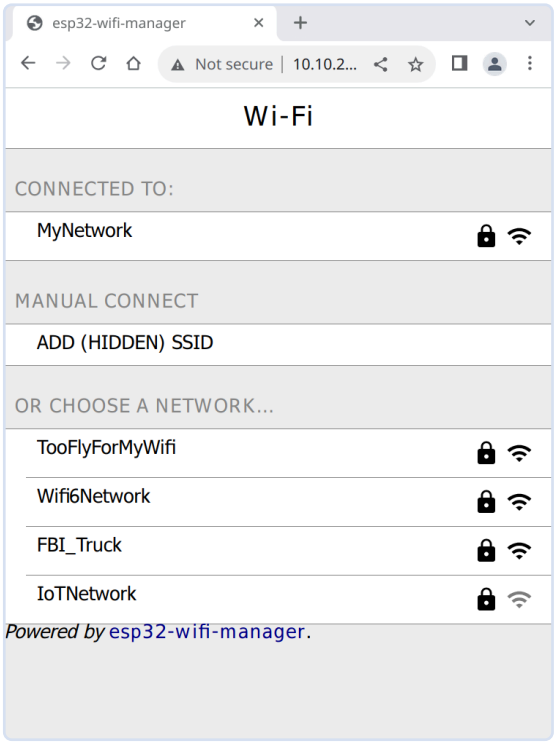


Bild 6. Auswahl des richtigen Zugangspunkts auf der WiFi-Manager-Seite des ESP32.

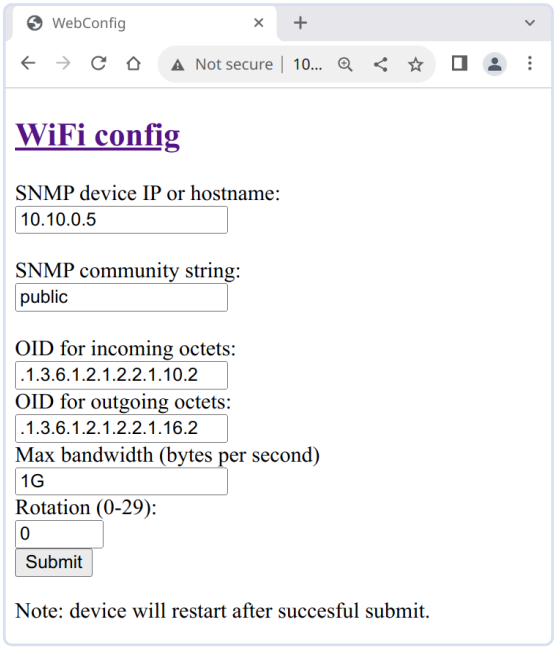


Bild 7. Eingabe der Konfigurationsparameter für die Verbindung.

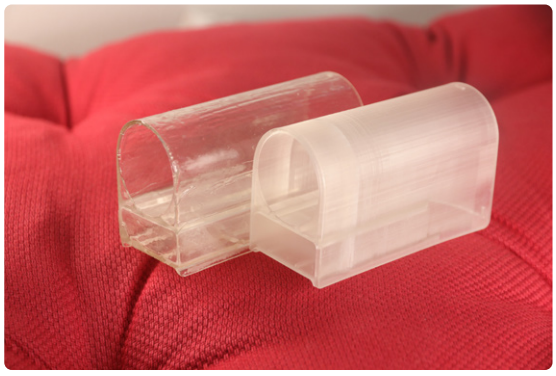


Bild 8. 3D-gedruckte Gehäuse für das Projekt: roh (links) und fertiggestellt mit einem geeigneten Hochglanzlack (rechts).

genutzte Internet-Bandbreite herausfinden. Meine Internetverbindung wird über einen Ethernet-Switch verteilt, und dieser Switch führt eine Statistik über alle Daten, die durch ihn fließen. Diese Daten lassen sich leicht über ein Protokoll namens *Simple Network Management Protocol* (SNMP) abrufen. Dieses Protokoll wird allgemein zur Überwachung von IT-Infrastrukturen wie Routern, Switches, Servern und so weiter verwendet. Der Aufwand ist gering, da nur einige wenige UDP-Pakete erforderlich sind, um die Statistiken für einen Netzwerkanschluss anzufordern, so dass ich dies problemlos mehrmals pro Sekunde tun kann, um einen Echtzeitüberblick über den aktuellen Datenverkehr zu erhalten. Mit diesen Daten wird die Geschwindigkeit der G1- und G2-Impulse eingestellt, das heißt, ein schnellerer Download führt zu einem schnelleren Drehen im Uhrzeigersinn. Wenn jemand stattdessen Daten hochlädt, dreht sich das Dekatron ebenfalls, allerdings gegen den Uhrzeigersinn.

Die letzte Funktion, die die Firmware ausführen muss, ist eine Art Konfigurationsschnittstelle. Ich mag es nicht, Konfigurationen in meiner Firmware fest zu codieren, also brauchte ich eine Möglichkeit, die WLAN-Anmeldedaten sowie die IP- und SNMP-Informationen des Switches einzugeben. Das nutzerfreundliche Eingeben von WLAN-Informationen wird als Provisioning bezeichnet, und ESP-IDF hat dafür eine sehr gute und wenig aufwendige Lösung mit einer Smartphone-App. In diesem Fall habe ich mich jedoch für einen so genannten WiFi-Manager entschieden. Wenn er nicht konfiguriert ist, erstellt er einen WLAN-Access-Point, auf den Sie einen Laptop oder ein Telefon richten können. Sobald die Verbindung hergestellt ist, wird eine Webseite geöffnet, auf der Sie den WLAN-Zugangspunkt auswählen und das Passwort eingeben können (**Bild 6**). Da der Wi-Fi-Manager ohnehin einen Webserver benötigt, lässt er sich leicht erweitern, um auch eine Konfigurationsseite für die SNMP-Parameter hinzuzufügen (**Bild 7**). Um die Fehlersuche zu erleichtern, werden dort auch einige statistische Daten angezeigt, wie zum Beispiel die mit dem Netzteil ausgehandelte Spannung.

Entwurf des Gehäuses

Nachdem Software und Hardware fertig waren, brauchte ich noch eine letzte Sache: ein Gehäuse. In diesem Fall dient ein Gehäuse nicht nur dekorativen Gründen: Auf der Platine sind Spannungen unterwegs, die zwar nicht tödlich sind, aber doch ziemlich wehtun können. Ich habe diese Arbeit „outgesourct“: Ich fragte einen unserer Industriedesigner, ob er mir ein Gehäuse entwerfen und es auf unserem SLA-3D-Drucker drucken könnte. Er konnte, und war so nett, mir den Gefallen zu tun, danke, Kaijie!. Er druckte ein Gehäuse aus transparentem Kunstharz, so dass man das Innere des Dekatrons noch sehen konnte. Zumindest war das die Idee. Wie sich herausstellte, ist das Gehäuse durch die Art des Drucks eher undurchsichtig als durchsichtig. Zum Glück gibt es eine ziemlich einfache Möglichkeit, dies zu beheben: Man beschich-

tet das Objekt einfach mit einer dünnen Schicht Kunstharz und härtet dieses Harz dann mit UV-Licht aus. Das Ergebnis ist in **Bild 8** zu sehen, neben einem unbehandelten Gehäuse. Wahrscheinlich hätte ich die Abdeckung gleichmäßiger gestalten können, aber ehrlich gesagt, stört mich die „handgemachte" Ästhetik des aktuellen Gehäuses nicht, im Gegenteil: Wenn das Dekatron wie in **Bild 9** aufgesetzt ist, lenkt es automatisch die Aufmerksamkeit auf die Vorderseite des Geräts. Das sieht natürlich noch besser aus, wenn das Gerät angeschaltet ist: Auch wenn man nicht weiß, was genau es anzeigt, der leuchtende Punkt, der sich nach rechts und nach links immer weiter dreht, ein echter Blickfang, wie man in **Bild 10** sehen kann.

Bevor wir gehen

Alles in allem habe ich etwas geschaffen, von dem ich hoffe, dass es zumindest teilweise als Retronik bezeichnet werden kann. Vielleicht wird das gesamte Projekt eines Tages als solches bezeichnet werden können. Vielleicht kooperieren Elektor und Espressif bis dahin wieder bei einer Gastausgabe der Zeitschrift, und vielleicht – aber nur vielleicht - darf ich dieses Gerät aus dem Regal nehmen und lyrisch über all die Dinge schwärmen, die zu seiner Existenz führten. Aber bis dahin wird es mir bei meinen nächsten großen Downloads als Anschauungsobjekt nützlich sein.

Wenn Sie ein Dekatron haben und dieses Gerät nachbauen (oder einfach Hardware- oder Software-Teile für eigene Zwecke wiederverwenden) wollen: Das gesamte Projekt ist natürlich Open-Source, und Sie können die Firmware, die Platinenvorlage und die 3D-Designdateien für das Gehäuse auf GitHub [6] verwenden.

RG – 230560-02

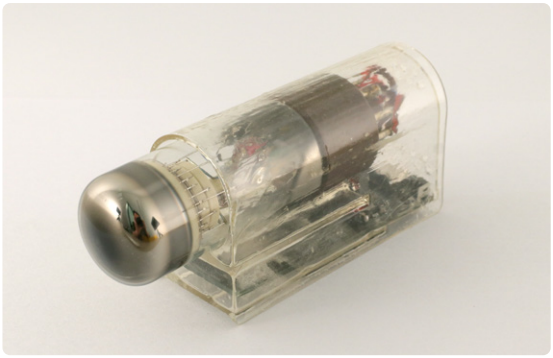


Bild 9. Das komplette Projekt in seiner endgültigen Behausung.

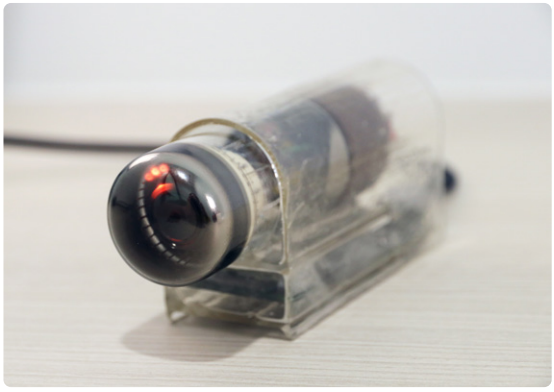


Bild 10. Der Prototyp bei der Arbeit: ein unwiderstehlicher Blickfang!

Haben Sie Fragen oder Kommentare?

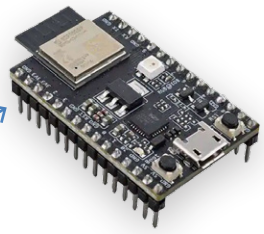
Wenn Sie technische Fragen oder Kommentare zu diesem Artikel haben, wenden Sie sich bitte an den Autor unter jeroen@spritesmods.com oder an die Elektor-Redaktion unter redaktion@elektor.de.

Über den Autor

Jeroen Domburg ist Senior Software and Technical Marketing Manager bei Espressif Systems. Mit mehr als 20 Jahren Embedded-Erfahrung ist er sowohl an Software- als auch an Hardware-Entwicklungen von Espressifs-SoCs beteiligt. In seiner Freizeit tüftelt er gerne an elektronischen Geräten, um sowohl praktisch nützliche als auch weniger nützliche Geräte zu entwickeln.

Passende Produkte

- **Anycubic Photon Mono X - UV-Harz SLA 3D-Drucker**
  www.elektor.de/19831
- **ESP32-C3-DevKitM-1**
  www.elektor.de/20324
- **Dogan Ibrahim, The Complete ESP32 Projects Guide (Elektor, 2019)**
  Buch, Paperback, englisch: www.elektor.de/the-complete-esp32-projects-guide
  E-Buch, PDF, englisch: www.elektor.de/the-complete-esp32-projects-guide-e-book



WEBLINKS

[1] Der Harwell Dekatron Computer (WITCH): https://tnmoc.org/witch
[2] Aufwärtswandler mit Dekatron: https://threeneurons.wordpress.com/dekatron-stuff/dekatron-do-hickie-kit
[3] Datenblatt GC10/4B: http://r-type.org/pdfs/gc10-4b.pdf
[4] KiCad-Bibliotheken für Espressif-Chips und -Module: https://github.com/espressif/kicad-libraries
[5] USB-PD Treiber-Stack für den FUSB302: https://github.com/Ralim/usb-pd
[6] Das Projekt-Repository: https://github.com/Spritetm/elek_dekatron

Revolution in der Heimautomation

Paulus Schoutsen über Home Assistant, ESPHome und mehr

Paulus Schoutsen. (Quelle: P. Schoutsen / Midjourney)

Fragen von Saad Imtiaz (Elektor)

Paulus Schoutsen, der Kopf hinter Home Assistant und ESPHome, spricht über seinen Trip durch die Welten der Heimautomatisierung und des IoT. Erleben Sie, wie die einfach bedienbare Firmware von ESPHome das Zusammenspiel intelligenter Geräte im Heim verändert.

Elektor: Beginnen wir mit Ihrem Hintergrund. Haben Sie sich schon immer für Elektronik und Technik interessiert? Was hat Sie dazu bewogen, an der Universität Twente Informatik zu studieren?

Schoutsen: Ich habe Wirtschaftsinformatik an der Universität Twente studiert. Es gibt dort Programmier- und Datenbankkurse, aber der Schwerpunkt des Studiums liegt auf Informationssystemen: Welche Informationen fließen durch Business-Prozesse und wie kann man sie erfassen, verarbeiten und analysieren. In gewisser Weise war das ein Vorläufer meiner Reise durch die Welt der Heimautomation. Ich habe mich immer sehr für das Programmieren interessiert, aber nicht so sehr für Elektronik. Das hat sich eher zufällig ergeben. Als 2012 das vernetzte Hue-Lichtsystem von Philips auf den Markt kam, habe ich es sofort gekauft. Ich stellte fest, dass es eine lokale API hatte, und schrieb ein Skript, um das Licht von meinem Computer aus zu steuern. Dieses Skript wuchs, und am 17. September 2013 veröffentlichte ich die erste Version auf GitHub: Home Assistant war geboren. Diese quelloffene Smart-Home-Plattform mit Schwerpunkt auf lokaler Steuerung und Datenschutz ist inzwischen das zweitaktivste Open-Source-Projekt auf GitHub [1].

Als Home Assistant zunehmend gedieh, spielte ich mit Elektronik herum, aber es gab ein großes Problem: Die Einbindung von Elektronik ins Internet war teuer. Ein WLAN-Shield für den Arduino kostete 70 $. Doch dann kam mit dem ESP8266 von Espressif der Chip, der alles veränderte.

Der ESP8266 kostete nur 3,50 $. Mit seinem Arduino-kompatiblen Code konnte man eine Verbindung zu einem WLAN herstellen. Dieser preiswerte Chip machte den Selbstbau von Elektronik bezahlbar, die sich in ein Smart Home integrieren ließ.

Während der ESP8266 langsam in der Maker-Szene Fuß fasste, gründete ich Nabu Casa, um die Entwicklung von Home Assistant nachhaltig zu gestalten. Letzteres ist ein wirklich großes Open-Source-Projekt, und ein solches Projekt zu betreiben, erfordert eine Menge an Verwaltung, Prozessen, Struktur und

Über Paulus Schoutsen

Paulus Schoutsen ist der Kopf hinter Home Assistant, einem der größten Open-Source-Projekte auf GitHub. Er gründete die Firma Nabu Casa zur langfristigen Verfügbarkeit und Weiterentwicklung von Home Assistant. Seine Arbeit dreht sich um „Open Home", seiner Vision eines intelligenten Heims, das Privatsphäre, Wahlmöglichkeiten und Nachhaltigkeit bietet.

Wartung - mehr als jemand in der Freizeit schaffen kann. Die Mitarbeiter dieser Firma arbeiten in Vollzeit und finanzieren sich durch Abonnements zusätzlicher Dienstleistungen für Nutzer. Nabu Casa hat keine Investoren und existiert nur, um seine Nutzer zufrieden zu stellen.
Mit dem Wachstum von Home Assistant definierten wir unsere Vision eines Open Home [2]. Es ist eine Vision für das intelligente Zuhause, die auf drei Werten basiert: Privatsphäre, Wahlmöglichkeiten und Nachhaltigkeit. Dabei erkannten wir folgendes: Wenn ein Smart Home lokal und privat sein soll, sollte dies sowohl für die Smart-Home-Plattform (also Home Assistant) als auch für die damit verbundenen Geräte gelten.
Einer der Hauptentwickler von Home Assistant war Otto Winter. Er interessierte sich für Elektronik, fand aber, dass frühe ESP8266-Projekte zu viel Boilerplate enthielten und eine einfachere Integration mit Home Assistant erforderlich war. Er machte sich daran, dieses Problem zu lösen, und veröffentlichte am 21. Januar 2018 Esphomelib [3]. ESPHome wurde immer beliebter, aber Otto hatte nicht mehr die Zeit, es zu verwalten. Da ESPHome eine wichtige Grundlage für die Entwicklung von Geräten ist, die unseren Open-Home-Vorstellungen entsprechen, wollten wir es unterstützen. Daher hat Nabu Casa zwei Monate später ESPHome übernommen [4]. Heute arbeiten zwei engagierte Vollzeitentwickler an ESPHome, die von den Abonnenten der Home Assistant Cloud von Nabu Casa finanziert werden.

Elektor: Können Sie ESPHome kurz beschreiben? Wie unterscheidet es sich von anderer IoT-Firmware und anderen Hausautomationslösungen?
Schoutsen: ESPHome ist eine Firmware für Boards auf der Basis von ESP8266, ESP32 und Raspberry Pi Pico, die es ermöglicht, Smart-Home-Geräte einfach zu erstellen und zu warten. ESPHome erspart den ganzen Aufwand, der mit dem Schreiben von Software verbunden ist, und ermöglicht es, sich auf die Entwicklung der Hardware zu konzentrieren.
Um ein Beispiel zu geben, wie einfach es ist, etwas mit ESPHome zu erstellen: Nehmen Sie einfach einen Temperatursensor und schließen Sie ihn an Ihren ESP32 an, teilen Sie ESPHome in einer Konfigurationsdatei mit, an welchem Pin der Sensor angeschlossen ist, und drücken Sie auf *Install*. ESPHome generiert dann die notwendige Firmware und installiert sie auf Ihrem ESP32-Gerät. Das Gerät fährt hoch, verbindet sich mit Ihrem WLAN und die gemessene Temperatur ist in Home Assistant verfügbar. Das wars schon alles. Bei der ersten Installation von ESPHome muss man das Board an den PC anschließen. Spätere Aktualisierungen erfolgen OTA. Sie möchten einen zweiten Sensor an dieselbe Karte anschließen? Aktualisieren Sie einfach die Konfigurationsdatei und klicken Sie auf Installieren. Unabhängig vom konkreten Ort im Haus erfolgt die Aktualisierung nahtlos.
Ein zusätzliches Feinheit von ESPHome ist, dass die Geräte auch als Bluetooth-Proxy für Home Assistant fungieren können. Home Assistant kann seine Bluetooth-Reichweite erhöhen, indem ESP32-Geräte mit ESPHome genutzt werden, um auf BLE-Pakete zu warten und direkte Verbindungen zur Steuerung herzustellen. Um dies noch nützlicher zu machen, haben wir auch BTHome [5] eingeführt, ein BLE-Protokoll zum Senden von Daten, die von diesen Bluetooth-Proxys empfangen werden (**Bild 1**).

Elektor: ESPHome hat eine starke Position innerhalb der DIY-Smart-Home-Community. Was glauben Sie, hat zu dieser Popularität geführt?
Schoutsen: Die einfache Nutzung. Wir haben uns sehr darum bemüht, den Umgang mit der Hardware zu vereinfachen. Mit ESPHome muss man nicht mit C++, Compiler-Fehlern oder falsch konfigurierten MQTT-Themen kämpfen. Außerdem lassen sich Konfigurationsdateien ganz einfach mit anderen Benutzern teilen, so dass auch Anfänger sofort funktionierende Geräte erstellen können.
Mit den *ESP Web Tools* haben wir eine Lösung entwickelt, die den Austausch von Konfigurationen erleichtert: Dabei handelt es sich um ein webbasiertes Installationsprogramm für ESP8266- und ESP32-Boards, mit man die Firmware mit einem Klick direkt per Browser installieren kann. Damit können Nutzer fertige Hardware kaufen und ESPHome darauf installieren. Damit ist es sogar möglich, beispielsweise ein *M5Stack Atom Echo Dev Kit* direkt via Browser in einen Sprachassistenten für Home Assistant zu verwandeln [6].

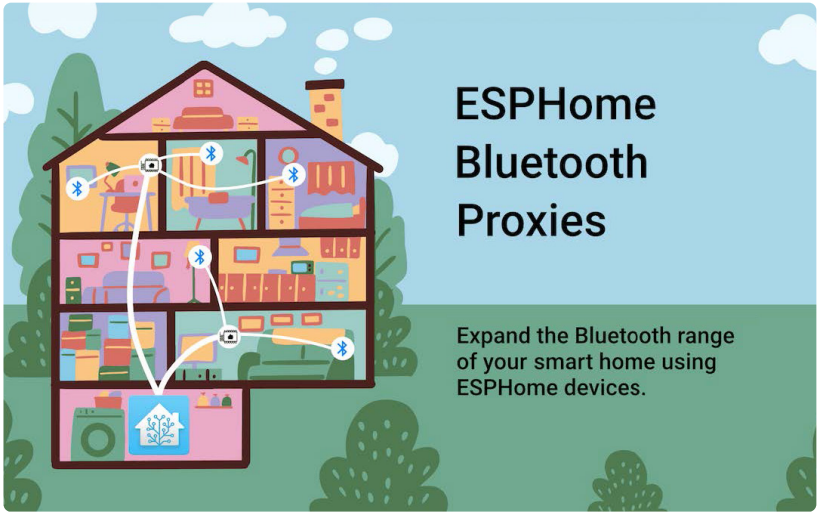
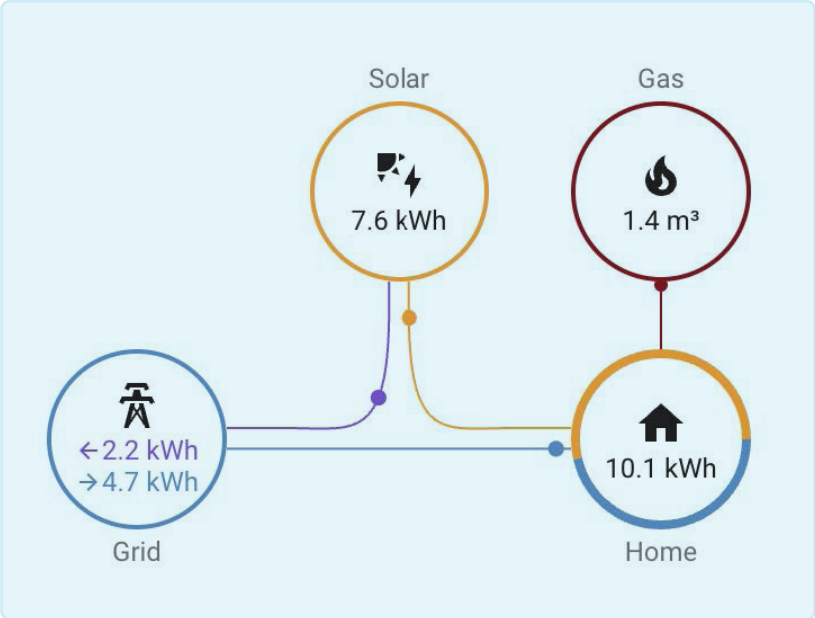


Bild 1: Home Assistant kann die Bluetooth-Reichweite durch die Verwendung von ESPHome-betriebenen Bluetooth-Proxys erweitern. (Quelle: Paulus Schoutsen)



▲

Bild 2: Energie-Dashboard im Home Assistant für einfachen Zugriff auf den Energieverbrauch. Außerdem bietet es Indikatoren zur Bewertung der Netzabhängigkeit. (Quelle: Paulus Schoutsen)

Die *ESP Web Tools* trugen wesentlich zur Verbreitung unserer ESPHome-basierten Projekte bei. Allerdings sind wir der Ansicht, dass diese Technologie mit allen geteilt werden sollte. So werden heute auch die Installer von Tasmota, WLED und viele andere Firmware damit betrieben.

Elektor: Wie kann ESPHome die Integration zahlreicher Sensoren und Geräte in der Heimautomation erleichtern, insbesondere für Personen ohne umfassende Programmierkenntnisse?
Schoutsen: Das Tolle an ESPHome ist, dass man für dessen Einsatz nicht programmieren können muss. Das Problem bei dieser Art Elektronik ist, dass man drei Dinge gleichzeitig lernen muss: Den Bau der Hardware, die Interaktion mit dieser Hardware und die Einbindung dieser Interaktion in ein anderes System. Mit ESPHome muss man sich nur auf den ersten Schritt konzentrieren: das Experimentieren mit der Hardware. Den Rest erledigt ESPHome. Wir ermöglichen es den Nutzern, sich auf den interessanten Teil des Baus ihrer Hardware zu konzentrieren.

Elektor: Die YAML-basierte Einrichtung ist eine der herausragenden Eigenschaften von ESPHome. Können Sie uns erklären, warum Sie sich für diese Methode entschieden haben und welche Vorteile sie für den Benutzer hat?
Schoutsen: Bei der Entwicklung von ESPHome haben wir die Architektur und das Konfigurationsformat von Home Assistant nachgeahmt. Es passt sehr gut zum ESPHome-Format. Jeder angeschlossene Sensor wird durch eine Integration repräsentiert. Diese Integration lässt sich konfigurieren, um die Sensoren abzustimmen, und diese Integration wird bei der nächsten Aktualisierung des Geräts berücksichtigt.

Elektor: Die ESPHome-Community hat dazu beigetragen, die Liste kompatibler Geräte und Komponenten zu erweitern. Wie wichtig ist die Einbindung der Community für den Erfolg des Projekts, und wie gelingt es Ihnen, diese Zusammenarbeit effektiv zu gestalten?
Schoutsen: Für die Open-Source-Heimautomation ist die Community unverzichtbar. Es braucht Zeit, sich mit der Funktionsweise einzelner Sensoren zu beschäftigen, die Treiber herauszufinden und sie in ESPHome einzubinden. Unsere Vollzeit-Maintainer arbeiten mit der Community zusammen, um Beiträge zu überprüfen und Dinge zusammenzuführen.
Interessante Themen für ESPHome waren in diesem Jahr der Sprachassistent, Bluetooth-Proxy und E-Ink-Displays:

- Mit Voice Assistant lässt sich ein ESPHome-Gerät in einen Sprachassistenten verwandeln, der Home Assistant steuert. ESPHome ist für die Aufnahme von Audiodaten zuständig, während Home Assistant die Sprache verarbeitet und auf sie reagiert. Ein Sprachassistent lässt sich via Browser installieren [6].
- Jedes ESP32-Gerät lässt sich in einen Bluetooth-Proxy für Home Assistant verwandeln. Home Assistant nutzt die BLE-Funktion des ESP32, um Daten-Paketen zu lauschen und sich mit Geräten zu verbinden, um sie zu steuern. Ein Bluetooth-Proxy lässt sich auch via Browser installieren [7].
- E-Ink-Displays sind angesagt. Mit der Unterstützung weiterer Displays lassen sich fantastische Dashboards für das Smart Home erstellen.

Elektor: Die Sicherheit von IoT und Heimautomation ist ein großes Thema. Welche Sicherheits- und Datenschutzmaßnahmen ergreift ESPHome zum Schutz von Nutzerdaten und verbundenen Geräten?
Schoutsen: Standardmäßig wird jedes neue ESPHome-Gerät mit dem Noise-Protokoll verschlüsselt. Das ist ein schnelles und effizientes Protokoll, das auch Wireguard und WhatsApp verwenden. Es verhindert das Ausspähen von ESPHome-Daten im Netzwerk.
Bei Verwendung von ESPHome-Geräten zusammen mit Home Assistant ist Ihr intelligentes Zuhause vollständig quelloffen, arbeitet vollständig lokal und ist verschlüsselt. Es werden keine Daten an Dritte weitergegeben, und Ihr Smart Home funktioniert auch ohne Anbindung ans Internet.
Wenn ESPHome ein Sicherheitsupdate enthält, kann es Ihre Konfigurationsdatei neu kompilieren und Ihr Gerät per Funk aktualisieren, damit es mit der neuesten Version läuft. Um die Sicherheit Ihres intelligenten Heims zu gewährleisten, können Sie Ihre ESPHome-Geräte über Home Assistant aktualisieren.

Elektor: Können Sie einige reale Anwendungen oder Projekte benennen, bei denen Home Assistant und ESPHome einen wichtigen oder besonderen Unterschied ausmachen?

ESPHome ist eine Firmware für Boards auf der Basis von ESP8266, ESP32 und Raspberry Pi Pico, mit der Smart-Home-Geräte einfach erstellt und gewartet werden können.

Schoutsen: Durch die Freisetzung von CO_2 in die Atmosphäre erwärmt sich das Klima. Um dem entgegen zu treten, sollten wir unseren CO_2-Fußabdruck so klein wie möglich machen. Unsere Häuser tragen einen erheblichen Teil zum Energieverbrauch bei. 2021 haben wir beim Home Assistant [8] ein Energiemanagement für das Heim eingeführt. Es ermöglicht die Überwachung des Energieverbrauchs und erleichtert die Umstellung auf nachhaltige Energie und erlaubt Kosteneinsparungen (**Bild 2**).
Die wichtigste Information ist die Erfassung des Energiebezugs aus dem Stromnetz. Durch Echtzeitzugriff auf diese Informationen kann man entscheiden, wann man etwa E-Bikes lädt, die Waschmaschine aktiviert oder wann wie heizt.
Diese Daten von Stromzählern zu erhalten, ist einigermaßen standardisiert, aber die Standards unterscheiden sich normalerweise von Land zu Land und von Region zu Region. Gemeinsam mit unserer Community haben wir Open-Source-Geräte auf der Grundlage von ESPHome entwickelt, die verschiedene Standards unterstützen. Der SlimmeLezer von Marcel Zuidwijk [9] beispielsweise lässt sich direkt an die P1-Ports (in den Niederlanden und einigen anderen EU-Ländern) anschließen, um den Energieverbrauch in Echtzeit sowie den kumulierten Strom- und Gasverbrauch abzurufen. Andere Stromzähler nutzen hierfür eine pulsierende LED. Jeder Impuls steht für eine definierte Energieeinheit. Mit dem Home Assistant Glow von Klaas Schoute [10] können diese LED-Impulse gezählt werden, um den genauen Energieverbrauch zu ermitteln. Home Assistant kann so den Verbrauch über einen längeren Zeitraum erfassen (**Bild 3**).

Elektor: Welche Fortschritte oder Erweiterungen können ESPHome-Nutzer erwarten, wenn sich das IoT-Ökosystem weiterentwickelt? Beabsichtigen Sie die Zusammenführung von ESPHome und Home Assistant?
Schoutsen: Einer unserer Werte für das Open Home ist die Wahlfreiheit. Während wir uns darum bemühen, die Integration zwischen Home Assistant und ESPHome zu optimieren, beabsichtigen wir nicht, den Anwendern die Steuerung von ESPHome-Geräten über andere Systeme zu verwehren.

Es gibt einige weitere Möglichkeiten für eine enge Integration. Wenn ein Nutzer zum Beispiel ein Gerät mit ESPHome-Firmware kauft, muss er das ESPHome-Dashboard verwenden, um das Gerät zu übernehmen und Updates zu installieren. Wir wollen dies vereinfachen, so dass die Nutzer sich nicht mit all diesen Dingen beschäftigen müssen, um auf dem neuesten Stand zu bleiben. Dies ist Teil unserer kontinuierlichen Bemühungen, die Anwendung im Rahmen unseres Programms *Works with ESPHome* zu verbessern.
Im Rahmen diese Programms können Hersteller unser Logo nutzen, wenn ihre Geräte eine Reihe von Anforderungen erfüllen, indem sie beispielsweise Funktionen für eine einfachere Inbetriebnahme aktivieren und es ermöglichen, die Konfiguration des Geräts anzupassen.

Elektor: Welche Tipps würden Sie Anfängern geben, die sich für den Einstieg in ESPHome interessieren und mit ihren eigenen Projekten beginnen möchten? Haben Sie irgendwelche Ressourcen oder Best Practices, die Sie weitergeben können?
Schoutsen: Beginnen Sie mit einem einfachen ESP32-Entwicklungsboard und einem Temperatursensor. Bringen Sie es zum Laufen, richten Sie es im Home Assistant ein und beginnen Sie, mit der ESPHome-Gerätekonfiguration zu spielen. Sie werden sehen, dass sich die Änderungen sofort in Home Assistant widerspiegeln, wenn Sie Ihr Gerät aktualisieren.
Der einfachste Weg zur Arbeit mit ESPHome ist die Installation aus dem Add-on-Store von Home Assistant [11]. Die Installation erfolgt mit nur einem Mausklick. Ein Klick auf *Add device* und ein Wizzard

Bild 3: Home Assistant Glow von Klaas Schoute. (Quelle: Paulus Schoutsen)

wird Sie bei der Installation Ihres ersten Geräts unterstützen. Viel Spaß beim Automatisieren!

Elektor: ESPHome ist für seine Benutzerfreundlichkeit bekannt, aber welche Maßnahmen oder Ressourcen würden Sie Neulingen empfehlen, um ihnen den Einstieg in ihr erstes ESPHome-Projekt zu erleichtern?
Schoutsen: Ich würde auf YouTube gehen. Dort gibt es eine große Auswahl an Videos, die den Einstieg erleichtern.

Elektor: Gibt es die Absicht, eine detailliertere Dokumentation oder Kurse anzubieten, die sich in erster Linie an Anfänger richten, um den Zugang zur Heimautomatisierung zu erleichtern?
Schoutsen: Wir prüfen das für 2024 - wir würden gerne ein einfaches Starterkit für ESPHome anbieten.

Elektor: Können Sie etwas über geplante Features oder Upgrades sagen, die ESPHome noch benutzerfreundlicher machen? Gibt es Pläne zur Verbesserung der Benutzeroberfläche?
Schoutsen: Wenn Sie ein ESPHome-Gerät erstellen, betten Sie Ihre SSID und das Passwort in die Firmware ein. Auf diese Weise findet das Gerät automatisch Ihr WLAN und verbindet sich damit. Das ist toll, außer man möchte es mit Freunden teilen oder seine Kreationen verkaufen. Damit das funktioniert, haben wir *Improv Wi-Fi* [12] entwickelt. Dabei handelt es sich um einen Standard zur Einrichtung von Geräten mit BLE oder seriell.
Eine geplante Verbesserung ist die Unterstützung für Improv Wi-Fi via BLE zu Home Assistant. Wenn man ein gekauftes Gerät anschließt, kann Home Assistant durch die Einrichtung führen, das Gerät verbinden und konfigurieren.

Elektor: ESPHome arbeitet bereits mit vielen Sensoren und Geräten zusammen. Werden in Zukunft weitere Sensoren und Komponenten hinzugefügt? Falls ja: Mit welchen Arten von Sensoren kann man rechnen?
Schoutsen: Wir werden niemals aufhören! In letzter Zeit wurden viele mmWave-Sensoren hinzugefügt. Sie bieten neue Möglichkeiten zur bewegungsunabhängiger Erfassung von Anwesenheit in einem Raum. Wir wollen uns auch um billigere Sensoren für die Luftqualität kümmern. Schlechte Raumluft kann die Denkfähigkeit und Ihre Leistung beeinträchtigen. Es wäre toll, solche Sensoren auf der Grundlage von ESPHome zu entwickeln, die uns helfen, gesünder zu leben.

Elektor: Bei der Integration eigener Sensoren oder Geräte kann man durchaus auf Schwierigkeiten stoßen. Wie wollen Sie diesen Prozess noch weiter vereinfachen und es leichter zu machen, ihre eigene Sensoren mit ESPHome zu verbinden?
Schoutsen: Jede ESPHome-Konfiguration erzeugt ein C++-Projekt, das kompiliert und auf dem Gerät installiert wird. Wenn ein Entwickler benutzerdefinierter Sensoren sein eigenes Protokoll erstellt, muss er C++ schreiben, um es in ESPHome zu integrieren. Wir haben Protokolle wie BTHome [5], um Daten etwa Richtung Home Assistant zu senden.

Mit ESPHome kann sich der Anwender auf das Experimentieren mit der Hardware konzentrieren. Um den Rest kümmern sich ESPHome und Home Assistant. Dadurch können Sie sich gleich um die interessanten Teile ihrer Hardware kümmern

Elektor: Wird Home Assistant auch zu einer Cloud-basierten Lösung weiterentwickelt, bei der man ein ESP32-Sensor-Setup verwenden kann, um Daten zu senden und zu visualisieren, damit man Home Assistant nicht auf einem lokalen Server einrichten und ausführen muss?
Schoutsen: Nein. Ihr Smart Home sollte lokal laufen und seinen Daten auch lokal speichern. Mit dem neuen Home Assistant Green [13] wird die Nutzung von Home Assistant weiter vereinfacht. Wir glauben, das ist der einfachste Einstieg für Anfänger in die Welt von Home Assistant. Der Preis liegt bei nur 99 $.

Elektor: Sind ESPHome und Home Assistant an Partnerschaften oder Kooperationen mit Hardwareherstellern oder an anderen Bemühungen zur Verbesserung der Gerätekompatibilität und der Benutzerfreundlichkeit beteiligt?
Schoutsen: Ja. Im Rahmen des Programms *Made for ESPHome* [14] kooperieren wir mit Herstellern, die ESPHome-basierte Geräte verkaufen. Es enthält eine Reihe von Anforderungen, die sicherstellen, dass die Geräte anpassbar und offen sind. Wir haben auch das Programm *Works with Home Assistant* [15] etabliert.

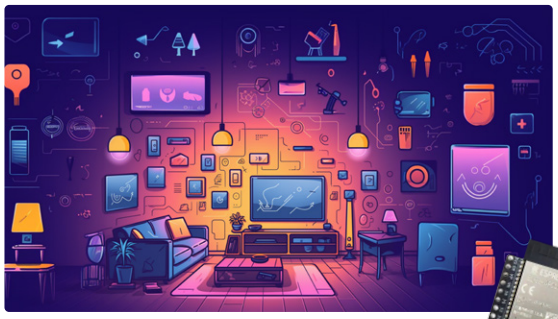
Es stellt sicher, dass wir Geräte auf Kompatibilität testen und mit dem Hersteller zusammenarbeiten, um etwaige Probleme zu beheben.

Elektor: Wie schafft es Home Assistant, neue Funktionen einzuführen und gleichzeitig bestehende zu pflegen, um Zuverlässigkeit und Benutzerfreundlichkeit bei der Weiterentwicklung zu gewährleisten?
Schoutsen: Dazu braucht es eine Menge Leute. Letztes Jahr war Home Assistant das zweitaktivste Open-Source-Projekt auf GitHub. Es haben 13.200 Personen mitgewirkt. All diese Leute arbeiten daran, Home Assistant kompatibel zu halten, wenn sich die Smart-Home-Branche durch die Unterstützung neuer Geräte oder neuer Funktionen in Geräten weiterentwickelt. Die Stärke von Home Assistant ist ihre Community, die es zur besten Plattform macht und die besten Ideen für das Smart Home bietet.

Elektor: Viele Anfänger in Sachen Heimautomation und des IoT befürchten eine steile Lernkurve. Womit lassen sich Hindernisse und Frustrationen am besten überwinden?
Schoutsen: Fangen Sie nicht mit etwas Ausgefallenem an, sondern bleiben zunächst beim Standardvorgehen. Es gibt einige alternative Möglichkeiten zur Installation von Home Assistant, aber bleiben Sie zunächst beim Home Assistant Operating System. Das Gleiche gilt für ESPHome: Bleiben Sie zunächst bei Standard-ESP32-Boards und versuchen Sie nicht, mit weniger verbreiteten Boards wie ESP32-S3 zu experimentieren. Das vereinfacht die Dinge.

Elektor: Wie werden Firmware- und Bug-Patches gehandhabt? Wird dies zukünftig automatisiert oder rationalisiert, damit man immer über die neuesten und stabilsten Versionen verfügt?
Schoutsen: Sicherheit hat in unserer Vision für das Open Home oberste Priorität. Das gilt auch für Home Assistant und ESPHome. Wir haben es sehr einfach gemacht, alles auf dem neuesten Stand zu halten: Bei Home Assistant kann man alle Teile mit einem einzigen Klick auf der Benutzeroberfläche aktualisieren. Dazu gehört auch die Aktualisierung von ESPHome-Geräten per Funk, wenn eine neue ESPHome-Version herauskommt.

230594-01



Passende Produkte

- **Espressif ESP32-C3-DevKitM-1**
  https://www.elektor.de/20324
- **Espressif ESP32-DevKitC-32E**
  https://www.elektor.de/20518

WEBLINKS

[1] Die besten Open-Source-Projekte: https://octoverse.github.com/2022/state-of-open-source
[2] The Open Home: https://home-assistant.io/blog/2021/12/23/the-open-home
[3] Esphomelib: https://tinyurl.com/29ank85h
[4] ESPHome-Übernahme: https://tinyurl.com/bdcjbtzr
[5] BTHome-Protokoll: https://bthome.io
[6] Voice assistant for Home Assistant: https://tinyurl.com/4kezt9su
[7] Bluetooth-Proxy-Installation: https://esphome.github.io/bluetooth-proxies
[8] Energie-Management mit Home Assistant: https://tinyurl.com/3pc478bz
[9] SlimmeLezer von Marcel Zuidwijk: https://slimmelezer.nl
[10] Home Assistant Glow von Klaas Schoute: https://github.com/klaasnicolaas/home-assistant-glow
[11] Add-on Store in Home Assistant: https://tinyurl.com/5ap2wpm5
[12] Improv Wi-Fi: https://improv-wifi.com
[13] Home Assistant Green: https://home-assistant.io/green
[14] Made for ESPHome: https://esphome.io/guides/made_for_esphome.html
[15] Works with Home Assistant: https://partner.home-assistant.io

Brenne, Firmware, brenne!

Flashen eines ESP32

Von Pedro Minatel, Espressif

Auch wenn Sie neu in der Embedded-Entwicklung sind, werden Sie sicher mit den Begriffen Brennen (Burning), Flashen (Flashing) und Schreiben (Writing) einer Firmware in einen Mikrocontroller oder Flash-Speicher vertraut sein - dies ist ein grundlegender Schritt zur Programmierung einer Anwendung auf einem Gerät. In diesem Artikel gehen wir näher darauf ein, wie dies bei ESP32-Controllern während der Entwicklungs- und Produktionsphase auf effiziente und sichere Weise geschehen kann.

Das Konzept des „Brennens der Firmware“ wurde einst bei einmalig programmierbaren Speichern wie einem programmierbaren Nur-Lese-Speicher (Programmable read-only Memory, PROM) verwendet, bei denen Daten nur einmal in den Speicher geschrieben werden können, so dass der Begriff „Brennen“ die Unumkehrbarkeit impliziert. Die Erklärung dafür ist die Tatsache, dass ein physikalischer Verbindungspfad wie eine Sicherung zwischen zwei Punkten durchgebrannt wird, wodurch der Stromfluss unterbrochen wird und sich dieses „Bit“ von 1 auf 0 ändert.
Im Gegensatz dazu wird eine ESP32-Anwendung in einem einfach wiederprogrammierbaren Flash-Speicher untergebracht, was es Ihnen ermöglicht, die Firmware-Daten tausende Male neu zu schreiben. Obwohl die Begriffe „Flashen“ oder „Schreiben“ für diesen Vorgang angebrachter erscheinen, spricht der Volksmund auch heute noch vom Brennen der Firmware.

Flashen des ESP32

Beim ESP32 ist der Flashvorgang einfacher als bei den meisten anderen Mikrocontrollern, vor allem weil der ESP32 über den UART geflasht werden kann. Sie brauchen kein spezielles Programmier- oder JTAG-Gerät (obwohl der JTAG-Port auf dem ESP32 zum Debuggen vorhanden ist), sondern nur einen externen USB-zu-Seriell-Adapter. Wenn ein moderner ESP32 verwendet wird, besitzt dieser sowohl einen seriellen USB-Anschluss zum Flashen als auch einen USB-JTAG-Anschluss zum internen Debuggen des SoC (System on Chip) durch Softwareimplementierung.

Download-Modus des ESP32

Um die Firmware auf einen ESP32 zu flashen, muss sich dieser im „Download-Modus“ befinden, der von der ROM-Boot-Phase (1st-Stage-Bootloader) gesteuert wird. Andernfalls wird der Boot-Prozess fortgesetzt und versucht, den Flash-Bootloader (2nd-Stage-Bootloader) und dann die Anwendung zu lesen.

Der Download-Modus wird aktiviert, indem man den BOOT-GPIO-Pin (normalerweise GPIO0 oder GPIO9) auf Low zieht und dort hält, während man den SoC zurücksetzt. Sobald der Download-Modus aktiv ist, kann der ESP32 die Firmware über den UART empfangen und im Flash speichern.

Interne serielle USB-Schnittstelle und JTAG

In einigen der neusten SoCs wurden zwei neue und äußerst nützliche Funktionen eingeführt: USB-Seriell und JTAG. Diese verbessern die Entwicklung und reduzieren die Materialkosten, da der externe USB-zu-Seriell-Anschluss entfällt. Außerdem werden einige Pins eingespart (insbesondere für JTAG).
Wenn der UART zum Flashen verwendet wird, müssen zwei GPIO-Pins für den UART (TX und RX) und ein GPIO-Pin zur Aktivierung des Download-Modus verwendet werden. Außerdem muss der Reset-Pin mit der Reset-Schaltung und dem Download-Modus-Pin verbunden sein. Auf der anderen Seite werden bei Verwendung des internen USB-Seriell-Umsetzers nur die GPIOs D+ und D- benötigt, da der automatische Download-Modus intern gehandhabt wird. Unterstützte SoCs sind:

- ESP32-C3
- ESP32-S3 (enthält auch USB Host und Gerät)
- ESP32-C6
- ESP32-H2
- ESP32-P4

Um USB Serial und JTAG auf den unterstützten SoCs zu verwenden, muss der USB-Anschluss an D+ und D- oder eine Testvorrichtung hinzugefügt werden, die direkt über ein USB-Kabel mit dem Computer verbunden sind. Welches die USB-Pins sind, entnehmen Sie bitte dem Datenblatt. Das bedeutet, dass Sie auf jedem ESP32 intern (auf den oben genannten SoCs) den seriellen USB und JTAG

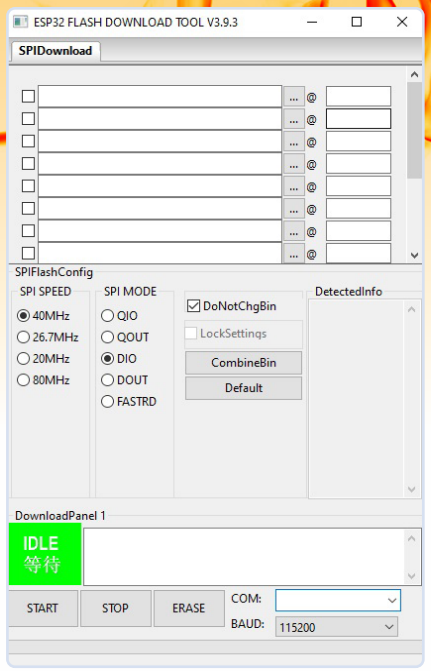


Bild 1. Flash Download Tools ist die offizielle GUI-basierte Flash-Anwendung von Espressif, die verwendet werden kann, ohne dass man ESP-IDF installieren oder ESPTool über die CLI ausführen muss.

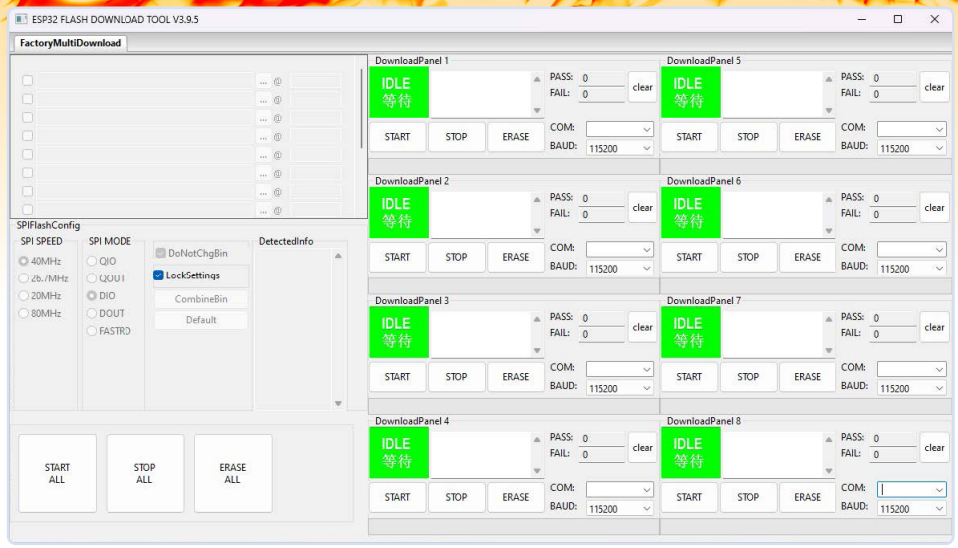


Bild 2. Flash Download Tools im Auslieferungsmodus.

zum Flashen und Debuggen haben, ohne zusätzliche Kosten oder externe Hardware.
Der ESP32-S2 besitzt keinen USB-Seriell-Wandler oder JTAG, aber er hat USB-OTG (Host und Device), so dass Sie mit DFU (Device Firmware Upgrade) anstelle von USB-Seriell flashen können.

Ist das sicher?

Wenn Sicherheitsbedenken bestehen, das Produkt mit einer eingebauten seriellen und JTAG-Schnittstelle an Kunden auszuliefern, können beide Funktionen mit den eFuses DIS_USB_JTAG und DIS_USB_SERIAL_JTAG während der Massenproduktionsphase deaktiviert werden. Diese Aktion ist nicht umkehrbar, also sollte nicht während der Entwicklung damit herumgespielt werden.
Eine weitere interessante eFuse ist die USB_EXCHG_PINS. Diese eFuse tauscht die USB-Pins D+ und D- aus, falls versehentlich Pins in dem Hardware-Design vertauscht wurden.

Flashen wir nun den ESP32!

Es gibt verschiedene Möglichkeiten, den ESP32 zu flashen. Es muss entschieden werden, welcher Weg der bequemere ist, abhängig von den Eigenschaften des Projekts oder der Produktionsphase. Im Folgenden stellen wir alle Möglichkeiten zum Flashen des ESP32 vor.

ESPTool

Das erste, wichtigste und zum Flashen der Firmware meistgenutzte Tool ist das Python-basierte ESPTool [1]. Es ist in das ESP-IDF integriert, so dass es das verwendete Mittel ist, wenn die Firmware über die Befehlszeilenschnittstelle (CLI) oder über die IDE geflasht wird. Um das ESPTool direkt von der CLI aus zu verwenden, müssen einige Parameter gesetzt werden, etwa so:

```
esptool.py -p -b --before default_reset --after hard_reset
--chip write_flash --flash_mode --flash_size --flash_freq
```

Wenn Sie diesem Befehl skeptisch gegenüberstehen, gibt es eine gute Nachricht: Sie können auch mit ESP-IDF erstellte Firmware flashen, indem Sie den Befehl idf.py -p flash verwenden.
Hier sind einige der ESPTool-Funktionen, die man über das reine Flashen von Firmware hinaus nutzen könnte:

- Löschen des gesamten Flash-Speichers oder nur eines Teils des Flash-Speichers
- Lesen und Schreiben aus dem Flash-Speicher und RAM
- Ausführen von Anwendungscode im Flash
- Lesen von MAC-Adressen aus dem OTP-ROM
- Auslesen der Chip-ID aus dem OTP-ROM
- Dump von Headern aus einer Binärdatei (Bootloader oder Anwendung)
- Zusammenführen mehrerer Roh-Binärdateien in eine einzige Datei zum späteren Flashen
- Abrufen einiger sicherheitsrelevanter Daten

Wenn die Firmware während der Massenproduktion geflasht werden muss, kann das ESPTool das Flashen mit Hilfe eines Skripts automatisieren.

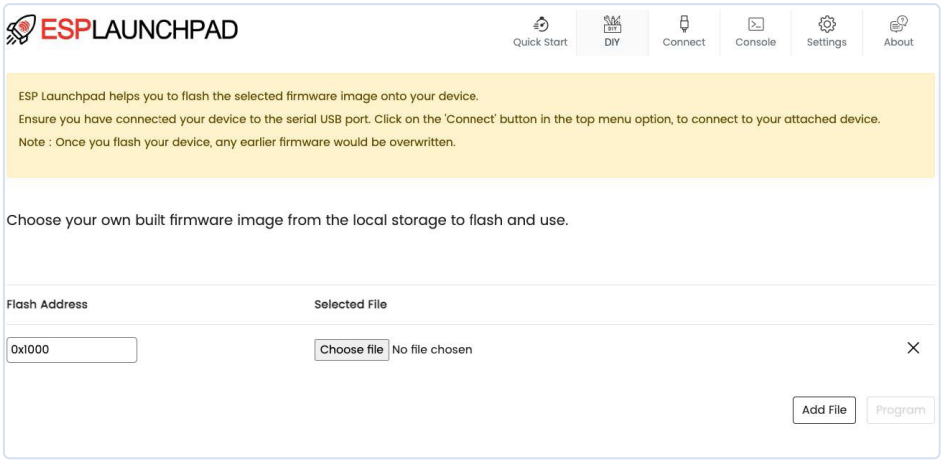


Bild 3. ESP Launchpad, das direkt über den Chrome-Browser aufgerufen wird.

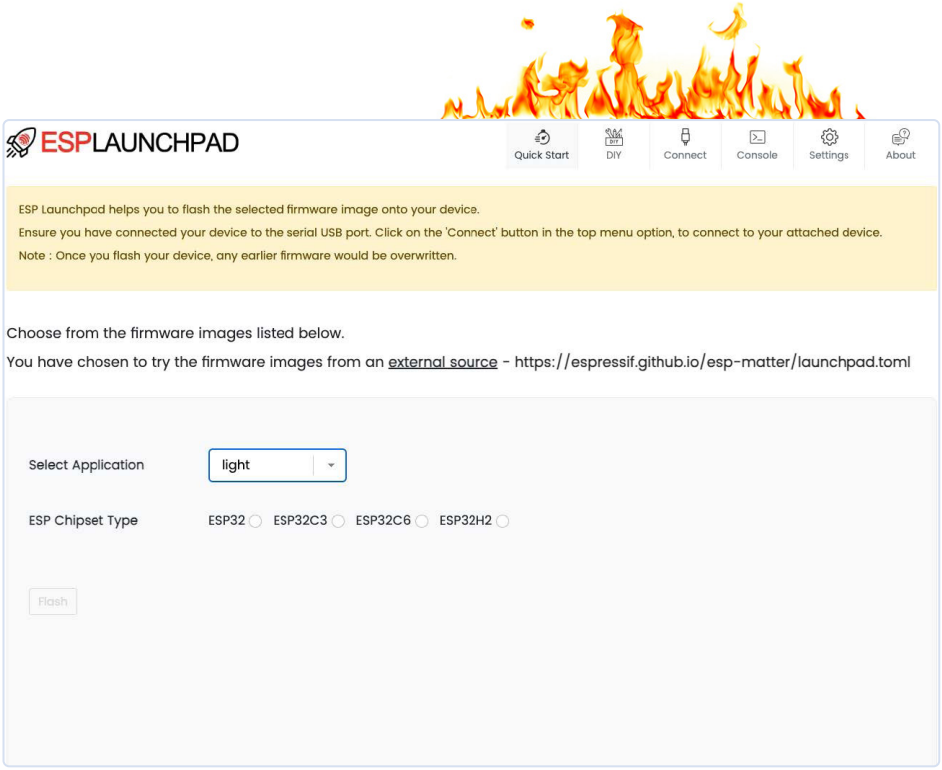


Bild 4. ESP Launchpad kann mit einigen zusätzlichen Konfigurationsdateien angepasst werden.

Flash Download-Tools

Flash Download Tools ist eine grafische Windows-Anwendung, die zum Flashen verwenden werden kann, ohne dass man das ESP-IDF installieren oder ESPTool über die Kommandozeile ausführen muss. Die grafische Benutzeroberfläche ist sehr einfach und leicht zu bedienen und erfordert keine Installation (**Bild 1**).

Um die Flash Download Tools zu verwenden, müssen alle Binärdateien, die geflasht werden sollen, und die jeweiligen Adressen im Flash-Speicher ausgewählt werden.

Flash Download Tools funktioniert auch im Werksmodus. In diesem Modus können mehrere serielle Schnittstellen ausgewählt und die Firmware auf mehrere ESP32 gleichzeitig mit nur einem Klick geflasht werden (**Bild 2**).

ESP-Launchpad

Manchmal möchte man einfach keine Anwendung auf dem Computer installieren oder ausführen und zieht es vor, nur webbasierte Anwendungen zu verwenden. In diesem Fall kann das ESP-Launchpad direkt über den Chrome-Browser aus ausprobiert werden. ESP-Launchpad [2] ist eine auf esptool-js basierende Webanwendung, mit der dank der WebUSB-API (mancher, aber nicht aller Browser) die Firmware direkt ohne Installation geflasht werden kann. Man kann auch die Konsolenausgabe überwachen und das Flash löschen (Löschen aller Daten aus dem Flash).

ESP-Launchpad ist eine Open-Source-Software und kann mit einigen zusätzlichen Dateien angepasst werden (**Bild 3**), um ihm mitzuteilen, wo die Firmware-Datei gespeichert ist und welche

Zielgeräte für diese Datei unterstützt werden (**Bild 4**). Hier braucht man also nur den Link an den Kunden zu senden, der den Flash-Vorgang einfach und intuitiv durchführen kann.

Beispiel für die TOML-Datei

In der TOML-Konfigurationsdatei kann man die URL für die Binärdateien hinzufügen und die Ziele für jede Binärdatei angeben. Es ist wichtig zu beachten, dass die Binärdatei eine einzige Datei sein muss, also die zusammengeführte Version, die alle erforderlichen Binärdateien enthält. Man kann sie mit ESPTool oder Flash Download Tools zusammenführen. Hier ist ein Beispiel für eine TOML:

```
esp_toml_version = 1.0
firmware_images_url = "https://espressif.github.io/
   esp-matter/"
supported_apps = ["light","light_switch"]
[light]
chipsets = ["ESP32","ESP32C3","ESP32C6","ESP32H2"]
image.esp32 = "esp32_light.bin"
image.esp32c3 = "esp32c3_light.bin"
image.esp32c6 = "esp32c6_light.bin"
image.esp32h2 = "esp32h2_light.bin"
ios_app_url = "https://apps.apple.com/app/esp-rainmaker/
   id1497491540"
android_app_url = ""

[light_switch]
chipsets = ["ESP32","ESP32C3","ESP32C6","ESP32H2"]
image.esp32 = "esp32_light_switch.bin"
image.esp32c3 = "esp32c3_light_switch.bin"
image.esp32c6 = "esp32c6_light_switch.bin"
image.esp32h2 = "esp32h2_light_switch.bin"
ios_app_url = "https://apps.apple.com/app/esp-rainmaker/
   id1497491540"
android_app_url = ""
```

Nachdem diese Datei auf einem öffentlichen Server gehostet ist, kann sie mittels URL zusammen mit der ESP-Launchpad-URL [3] verwendet werden.
Eine nützliche Anwendung für ESP-Launchpad ist ein Firmware-Update-Tool, das vom Endkunden nach dem Verkauf verwendet wird, wenn keine Over-the-Air-Updates verfügbar sind.

ESP Serial Flasher

Stellen Sie sich vor, man muss ein ESP32- über ein anderes Gerät oder einen Host-Mikrocontroller flashen. Hierfür kann man sich das Projekt *ESP Serial Flasher* ansehen. In diesem Projekt wird ein Host-Mikrocontroller verwendet, um einen ESP32 zu flashen, wobei es sich auch um einen anderen ESP32 handeln kann (**Bild 5**).
Eine häufige Anwendung für diese Funktion ist, wenn der ESP32 als Funk-Coprozessor verwendet wird und Sie eine neue Firmware-Version über das Host-Gerät flashen möchten. Derzeit unterstützt der *ESP Serial Flasher* folgende Host-Controller:

- ESP32
- STM32
- Raspberry Pi
- Jede MCU mit Zephyr OS

Die Ziel-Mikrocontroller, die Updates über UART unterstützen, sind:

- ESP32
- ESP8266
- ESP32-S2
- ESP32-S3
- ESP32-C3
- ESP32-C2
- ESP32-H2
- ESP32-C6
- ESP32-P4

Over-the-Air-Update

Schließlich ist Over-the-Air-Update eine weitere großartige Lösung für die Aktualisierung der Firmware, insbesondere im Feld. Diese Technik wird häufig verwendet, um Firmware über das Internet oder ein lokales Netzwerk zu aktualisieren, ohne dass die einzelnen Firmware-Updates physisch durchgeführt werden müssen.
Der Hauptvorteil der OTA-Aktualisierung kommt zum Tragen, wenn es Probleme mit der aktuellen Firmware-Version gibt oder eine neue Funktion hinzufügt werden muss. So kann aus der Ferne eine große Menge von Geräten gleichzeitig aktualisiert werden, was Zeit und Geld spart.
Um OTA weiter zu erforschen, sollten Sie sich die Beispiele unter dem ESP-IDF-Projekt auf GitHub oder in der lokalen Installation ansehen.

Bonus: die ESP-USB-Brücke

Dies ist ein weiteres ESP-IDF-Projekt, das bei der Entwicklung helfen kann, insbesondere wenn ein ESP32 ohne USB-Seriell oder JTAG verwendet wird, wie ESP32, ESP32-C2 und ESP32-S2.
Jeder ESP32-S2 oder ESP32-S3 kann in ein Seriell-zu-USB- und JTAG-Gerät verwandeln werden und zum Flashen und Debuggen eines anderen ESP32 verwenden werden. Man muss nur einen beliebigen ESP32-S2 oder ESP32-S3 mit der ESP-USB-Bridge flashen und die GPIOs mit dem Ziel-Mikrocontroller verbinden (**Bild 6**).
Man kann dieses Projekt zusammen mit allen Flashtools wie ESPTool, Flash Download Tools und ESP-Launchpad verwenden.

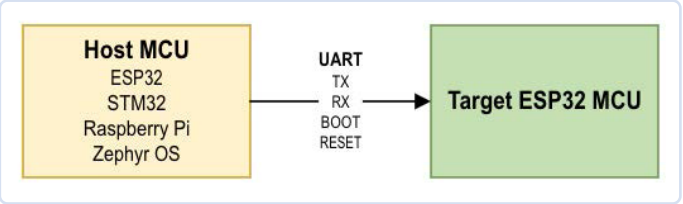


Bild 5. Mit dem Projekt ESP Serial Flasher kann man mit einem Host-Mikrocontroller einen ESP32 flashen, oder sogar einen anderen ESP32 als Host zum Flashen verwenden.

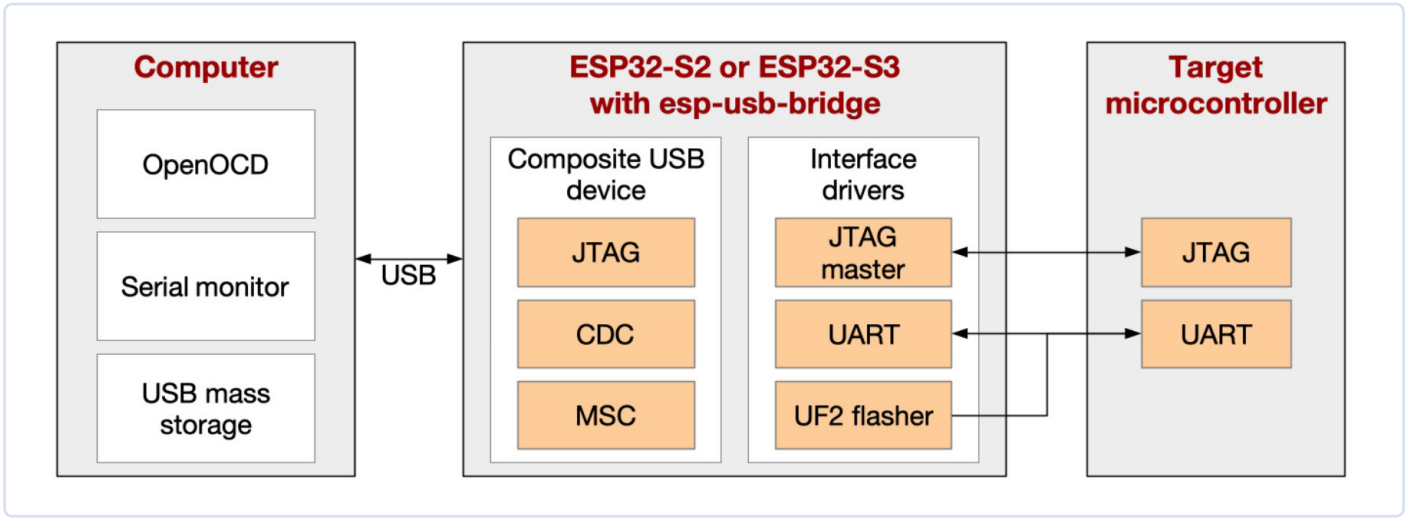


Bild 6. ESP-USB-Bridge ist ein ESP-IDF-Projekt, das mit einem ESP32-S2 oder einen ESP32-S3 eine Verbindung zwischen einem Computer und einem Ziel-Mikrocontroller herstellt.

Diese können auch die USB-MSC (Mass Storage Class) verwenden, indem die Binärdatei im UF2-Format auf dem USB-Massenspeicher abgelegt wird, als der sich die ESP-USB-Bridge beim Anschluss an den Computer outet.

Viele Wege

In der Welt der ESP32-Mikrocontroller gibt es viele Möglichkeiten, neue Firmware auf einen ESP32 zu laden. Auch wenn ESPTool die beste Lösung zu sein scheint, sollten Sie bedenken, dass bestimmte Szenarien fortschrittlichere Tools mit grafischen Benutzeroberflächen (GUIs) oder vereinfachte Flash-Prozesse ermöglichen, die sich an Benutzer verschiedener Plattformen richten, ohne dass eine Softwareinstallation erforderlich ist.

Dieser Artikel soll Ideen zur Verbesserung der Produktentwicklung und -produktion liefern. Außerdem soll er Einzelpersonen die Möglichkeit geben, Firmware einfach zu flashen oder auf die neusten Versionen zu aktualisieren, unabhängig von ihrem technischen Hintergrund oder der von der verwendeten Plattform.

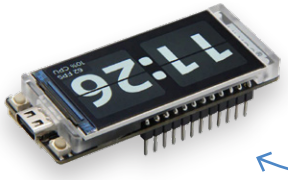
Übersetzung von Raphael Schermann – (230625-02)

Haben Sie Fragen oder Kommentare?

Wenn Sie technische Fragen zu diesem Artikel haben, schicken Sie bitte ein E-Mail an den Autor unter pedro.minatel@espressif.com oder an die Elektor-Redaktion unter redaktion@elektor.de.

Über den Autor

Pedro Minatel berät bei Espressif Entwickler und Maker. Er hat einen Abschluss als Elektroniker und einen Associate-Abschluss in Informationstechnologie. Pedro arbeitet seit 2021 für Espressif, ist aber bereits seit 2014 ein aktives Mitglied der Community, veröffentlicht Artikel und hält Vorträge auf Konferenzen über IoT und die Maker-Community. Derzeit ist er für die jährliche Espressif-Entwicklerkonferenz, die sogenannte DevCon, verantwortlich.



Passende Produkte

- **ESP32-C3-DevKitM-1**
  www.elektor.de/20324
- **LILYGO T-Display-S3 ESP32-S3 Entwicklungsboard (mit Headern)**
  www.elektor.de/20299

WEBLINKS

[1] ESPTool: https://github.com/espressif/esptool
[2] ESP-Launchpad: https://espressif.github.io/esp-launchpad/
[3] ESP-Launchpad WebUSB:
https://espressif.github.io/esp-launchpad/?flashConfigURL=https://espressif.github.io/esp-matter/launchpad.toml

Von Raketen zu Cellos

Praktische Anwendungen und Überlegungen bei der Entwicklung drahtloser Lösungen

Von Sorin Jayaweera, GXB Ventures

Was erhält man, wenn man einen Druckknopf und eine Antenne zusammenbringt? Natürlich einen Raketencomputer!

Der Traum

In diesem Artikel wollen wir Ihnen die wesentlichen Werkzeuge für die Initiierung von Wireless-Projekten an die Hand geben, wobei wir uns auf die Kernhardware, die Strukturen von Kommunikationsnetzwerken und die wichtigsten Überlegungen konzentrieren.

Wir begannen unsere Reise mit dem Versuch, ein Netzwerk preiswerter, kleiner und drahtloser Hilfe-Knöpfe zu bauen, in der Hoffnung, dass wir sie in den Häusern unserer Großfamilie anbringen könnten, um in Notfällen einen einfachen Kontakt mit dem Rest der Familie zu ermöglichen. Abstrakt gesehen benötigt jeder Knopf die Fähigkeit, Daten aufzuzeichnen – nämlich, ob er gedrückt wurde - und die Fähigkeit, diese Information auf irgendeine Weise an etwas zu übertragen, das sie weiterleiten kann. Außerdem mussten wir ein Gateway zum Internet entwickeln, das gleichzeitig das lokale Netzwerk überwachen und Warnmeldungen an entfernte Familienmitglieder übermitteln kann.

Dies ist eine scheinbar einfache Aufgabe, aber sie basiert auf denselben Grundlagen wie viele andere coole Praxisprojekte. Wenn man beispielsweise die gleichen Systeme wie die Taste für ein Kurzstrecken-Kommunikationsnetz verwendet, aber Sensoren zur Aufzeichnung von Umweltdaten hinzufügt, könnte dies die Grundlage für eine effiziente lokale Wetterstation bilden, die Fragen beantworten könnte wie: „Wie hoch ist die aktuelle Temperatur auf dem Dachboden, im Keller und im Hühnerstall?“ Anhand dieser Daten könnten wir entscheiden, ob wir automatisch Lüfter aktivieren, um diese Räume zu kühlen, oder die Lüftungsöffnungen schließen, um sie zu isolieren. Mehrere dieser einfachen und kostengünstigen Stationen zur Messung von Umweltgrößen, die hintereinandergeschaltet auf einem Bauernhof arbeiten, könnten unschätzbare Informationen für eine bessere Pflege von Kulturpflanzen liefern.

Wenn man die Funkreichweite erhöht und relevante Sensoren wie Beschleunigungsmesser und GPS hinzufügt, erhält man einen Tracker für Modellraketen und einen Flugdatenlogger für Bastler. Eine drastische Erhöhung der Reichweite könnte sogar zu einem Telemetrie- und Verfolgungssystem für Höhenballons führen. Umgekehrt wäre eine drastische Verringerung der Latenzzeit bei geringer Reichweite nützlich für die Entwicklung elektronischer Musiksysteme, zum Beispiel für drahtlose kleine Pedale, die mit einem Mikrocontroller im Zentrum der Musikanlage kommunizieren. Mit einigen wenigen gängigen Methoden für Kommunikation, Datenerfassung und Signalverarbeitung lassen sich wirklich viele interessante Dinge realisieren. Was sind also die grundlegenden Anforderungen, die im Mittelpunkt dieser Art von Projekten stehen?

Die Technik

Um die oben genannten Anforderungen zu erfüllen, müssen die Schaltungen kompakt und leicht sein und über eine effiziente Energieverwaltung verfügen. Außerdem sollte jedes Projekt über die Möglichkeit verfügen, drahtlos über ein Netzwerk zu kommunizieren, zum Beispiel

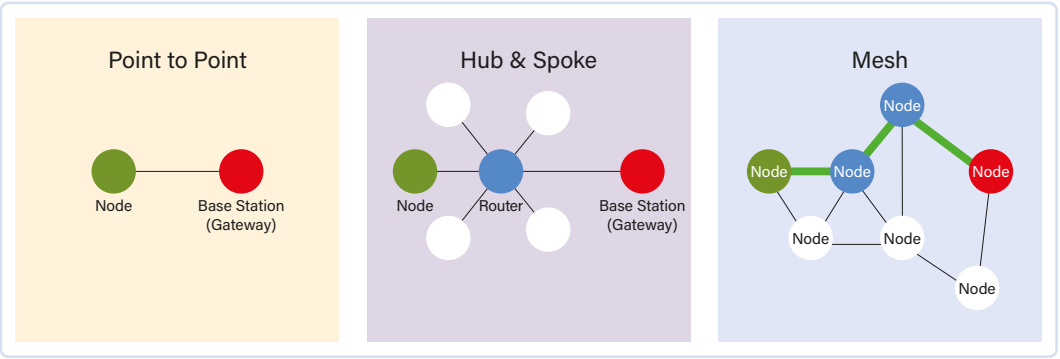


Bild 1. Netzwerktopologien.

über Ad-hoc-Mesh-Netzwerke oder Hub-and-Spoke-Netzwerke (wir werden später noch darauf eingehen, was das bedeutet). Diese Netze können relativ klein sein - zum Beispiel die Hilfetaste - oder weitreichend für die Raketen, aber die Gesamtstruktur dieser Netze ist ähnlich. Für die Hilfetasten und die nachfolgenden Projekte schlossen wir leistungsstarke Minicomputer wie den Raspberry Pi aus - sie haben einen relativ hohen Energiebedarf und vertragen keine „harten" Abschaltungen. Wir haben zunächst Teensy-Controller eingesetzt, sehr leistungsfähige Arduino-kompatible Mikrocontroller für die Signalverarbeitung, die für viele Probleme eine gute Lösung sind. Teensies sind zwar sehr anpassungsfähig, benötigen aber für alles, was sie tun, zusätzliche Peripherie, vor allem für die drahtlose Kommunikation, und das wird mit zunehmender Projektgröße unhandlich. Da drahtlose Kommunikation über große Entfernungen so wichtig ist, haben wir uns nach fertigten integrierten Lösungen umgesehen, um die Komplexität unserer Hilfe-Tasten niedrig zu halten.

Wir haben die LoRa-Boards von Heltec ausprobiert, da diese WLAN-, Bluetooth- und LoRa-Funktionalität besitzen. Die Boards schienen vielversprechend zu sein, aber wir hatten zahlreiche Probleme, sie zum Laufen zu bringen, und empfanden die Community-Foren und den technischen Support als wenig hilfreich. Der mitgelieferte Beispielcode ließ sich nicht richtig kompilieren, ohne dass wir selbst herausfinden mussten, wie wir den Quellcode und die verknüpften Bibliotheken bearbeiten können, und selbst die mitgelieferten Antennen waren für die LoRa-Frequenzbänder, für die sie optimiert sein sollten, falsch abgestimmt. Also suchten wir weiter und entdeckten die verschiedenen Espressif-Chips und die Entwicklungsboards mit diesen Chips. Die Espressif-Chips haben WLAN- und Bluetooth-Funktionalität, was die Komplexität unseres Gesamtsystems reduziert. Mit den auf den Boards integrierten Antennen sind sie ideal für viele lokale IoT-Projekte mit relativ kurzen Entfernungen, da zudem die Reichweite durch die Verwendung mehrerer Chips und die Weiterleitung von Nachrichten erhöht werden kann. Fairerweise muss man sagen, dass nicht alle kommerziellen Boards gut funktionierten - ein ESP-NOW (Wi-Fi) Reichweitentest mit einem ESP-WROOM32-Board und einem ESP32-C3-Board ergab Reichweiten von nur 3 m beziehungsweise 80 m. Aber nachdem wir das Angebot eingegrenzt hatten, hat es uns wirklich Spaß gemacht, den ESP32-C3 in unsere Wireless-Projekte zu integrieren. Das lag vor allem an der Winzigkeit des Chips, der spezifischen Unterstützung für unsere Projekte, der geringen Stromaufnahme und der integrierten Kommunikation mit einer gut gestalteten gedruckten Antenne.

Da wir nun unseren „Hauptcomputer" hatten, mussten wir nur noch die Platine und zusätzliche Sensoren und I/O für jedes Projekt entwerfen. Für die Ruftasten mit kurzer Reichweite beschlossen wir, jede Taste mit einem eigenen C3-Modul auszustatten, das mit Knopfzellenbatterien

Eigenschaften verschiedener Boards

Board	Kosten (USD)	Leistung (mW)	Vorteile	Nachteile
Arduino 	> 15	ca. 400	+ Leicht zu erlernen + Schnelles Prototyping	- Kein Mulitasking - Keine Kommunikation - (Relativ) groß
Teensy 	> 15–40	ca. 800	+ Klein + Sehr schnell + Umfangreiche Audio-Unterstützung + Aktive Community	- Keine integrierte Kommunikation - Hoher Energiebedarf
Raspberry Pi 	> 15–80	ca. 1200	+ „Richtiger" Computer + Multi-Threading + Jede beliebige Programmiersprache + Integrierte Funkverbindung (Bluetooth, WiFi)	- (Relativ) teuer - Verträgt kein „hartes" Abschalten - Höherer Energiebedarf als bei Mikrocontrollern - Größer als Mikrocontroller
ESP32 	> 2–5	ca. 6–300	+ Arduino-kompatibel + Klein (Dev-Boards) und kleiner (Module) + Integrierte Funkverbindung (WiFi, Bluetooth 5 / BLE + Kostengünstig + Optimiert für niedrige Energieaufnahme	- Verbindet sich nur mit 2,4-GHz-WLAN - ESPNow & ESP-WiFi-Mesh nur proprietär

betrieben wird und sich nur einschaltet, wenn die Taste gedrückt wird. Um die Hilfesignale zu empfangen, hatten wir auch ein immer lauschendes Gateway, das sich bei Bedarf mit dem allgemeinen Internet verband und eine Textnachricht an uns schickte. Es handelt sich um eine kleine Version eines „Hub-and-Spoke"-Netzwerks (**Bild 1**), bei dem die immer lauschenden Gateways mit Sensoren verbunden sind, die sich nur bei Bedarf einschalten und senden. Da sich jede Taste nur dann einschalten muss, um ihr eigenes Signal zu senden, wird die Lebensdauer der Batterie erheblich optimiert. Wir haben dieses System implementiert, indem wir nur ESP-NOW von jedem Endknoten zu dem immer zuhörenden Gateway verwenden, das die Nachrichten über Ethernet an das Internet weiterleitet.

Eine weitere sehr nützliche Netzwerktopologie ist das sogenannte Mesh-Netzwerk. In einem Mesh-Netzwerk ist jeder Knoten immer eingeschaltet. Die Nachrichten werden auf dem direktesten Weg zu einem Endziel weitergeleitet. Meshes haben mehr Redundanz, denn wenn ein einzelner Knoten ausfällt, kann der Rest des Systems weiterarbeiten, indem er die Übertragungsroute ändert. Die Kurzstreckenversion eines Mesh-Netzes lässt sich relativ einfach mit der painlessMesh-Bibliothek oder einem ESP-WIFI-MESH-Netz mit Espressif-Boards aufbauen, deren Quellcodes alle leicht online zu finden sind.

Für Projekte mit größerer Reichweite ist WLAN jedoch keine praktikable Lösung. Unser bevorzugter Ansatz ist LoRa - ein Kommunikationsprotokoll mit niedriger Datenrate und geringer Stromaufnahme, das jedoch das Potenzial hat, Reichweiten von mehreren Kilometern zu erzielen. LoRa hat eine sehr niedrige Datenrate und ist nur halbduplex (die Sender können nicht gleichzeitig senden und empfangen), was die Realisierung von Mesh-Netzwerken schwierig macht.

Es gibt mehrere andere potenzielle Kommunikationsmethoden, die im Kasten **Vergleich von Drahtlostechnologien** aufgeführt sind. Bitte um Nachsicht, wir haben nicht jede Lösung getestet, sondern uns hauptsächlich aus dem Internet informiert, bevor wir uns für eine entschieden haben. Für LoRa, ESP-WIFI-MESH und ESP-NOW haben wir aber unsere eigenen Daten angegeben. Andere Lösungen sind schwieriger zu handhaben, eignen sich aber für „offizielle" Projekte.

Für das Raketenverfolgungs-/Datenaufzeichnungssystem haben wir LoRa- statt WLAN-basierter Kommunikation verwendet, da wir Stabilität und große Reichweite benötigten. Als LoRa-Peripheriegerät empfehlen wir entweder den RFM95W von Hope oder den UART-basierten RYLR406 von Reyax. Letzterer ist sehr benutzerfreundlich, verfügt über eine gute Antenne und sendet Daten mit relativ einfachen AT-Befehlen. Es bietet nicht so viel Unterstützung für vorgefertigte Netzwerkstrukturen, eignet sich aber perfekt für einfache Punkt-zu-Punkt-Kommunikation. Alternativ dazu bietet das HopeRF-Board (mit SPI) viel mehr Unterstützung durch die Bibliotheken *radiolib* und *radiohead*, die Implementierungen für (etwas wackelige) Ad-hoc-Mesh-Netzwerke haben.

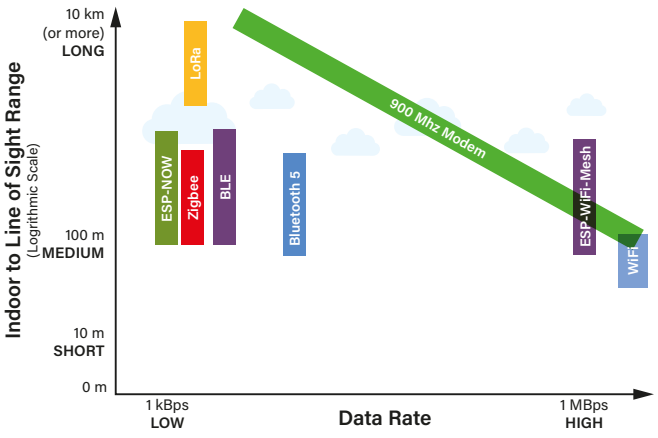
Wir haben eine Mesh-Topologie für die Raketensysteme verwendet, da wir auf diese Weise mehrere Raketen starten konnten und die Raketen in der Luft Sendungen von den gelandeten Raketen empfangen und dann die Standorte aller Raketen an unsere Basisstation übermitteln konnten. Da wir nicht mehr als drei Raketen auf einmal starteten, war die niedrige Datenrate von LoRa und die Vollduplex-Unfähigkeit für unsere Ziele nicht von Nachteil. Für größere Projekte wäre es jedoch gut, LoRaWAN oder andere große Hub-and-Spoke-Netzwerke zu verwenden, die Vollduplex unterstützen.

Wir haben versucht, in diesem zugegebenermaßen kurzen Artikel so viele relevante Einsteigerinformationen wie möglich unterzubringen. Ausführlichere Informationen über Netzwerkstrukturen, die Grundlagen der Verwendung von Espressif-Systemen und so weiter finden Sie im Video zu unserem Vortrag „DevCon23 - From Rockets to Cellos: Real-world Applications of ESP32 Series and Dev Board Variants" auf der Espressif-Website oder auf YouTube [1].

RG — 230627-02

Vergleich von Drahtlostechnologien

Technologie	Datenrate (Kbps)	Reichweite in Innenräumen (Sichtverbindung)
Wi-Fi 2.4 GHz (≠ 5.2 GHz)	9.000	25 m (100 m)
ESP-WiFi-MESH	1.000	50 m (200 m)
Bluetooth 5 (audio)	260	50 m (240 m)
Bluetooth LE (low bit rate)	120	70 m (500 m)
Zigbee 2.4 Ghz	30	70 m (250 m)
LoRa 915 MHz	30	1,2 km (20 km)
ESP-NOW	1	70 m (500 m)
900-MHz-Modem	25 bis 7.000	50 m (45 km)



WEBLINK

[1] „DevCon23 - From Rockets to Cellos: Real-world Applications of ESP32 Series and Dev Board Variants": https://youtu.be/jxPUkmaYp2c

ARDUINO CLOUD

What Arduino Cloud is

Develop from anywhere

- **NO CODE**
  With ready-to-use templates
- **LOW-CODE**
  Automatically generated sketches
- **FULL ARDUINO EXPERIENCE**
  Either offline with the UDE2 or online with the Cloud Editor
- **STORE YOUR SKETCHES ONLINE**
  Use your code in your favourite Arduino development environment

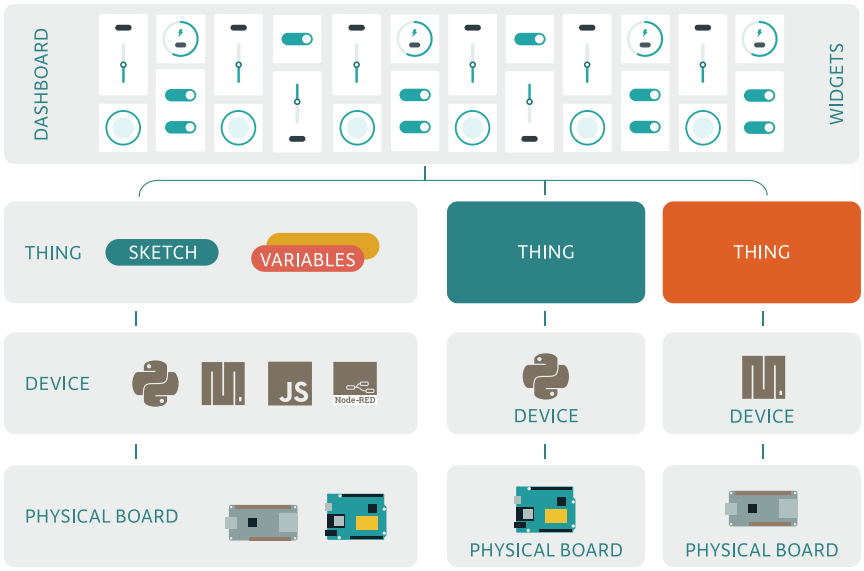
Program/Deploy

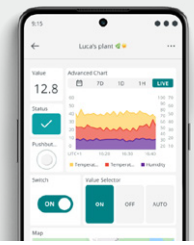
- **CABLE**
  Traditional USB programming
- **OVER-THE-AIR (OTA) UPDATES**
  Deploy your firmware wirelessly to your devices
- **MASS SCALE & AUTOMATION**
  With the Arduino Cloud CLI

Monitor & Control

- **CUSTOM DASHBOARDS**
  Using drag and drop widget
- **INSIGHTFUL WIDGETS**
  Interact with the devices and get real-time and historical data with dozens of widgets
- **MOBILE APP**
  Visualise your data in real-time from your phone with the IoT remote app

How does it work?





FULL CONTROL IN YOUR HANDS

Use your dashboards on the go, and control projects from anywhere in the world, using the free IoT Cloud Remote app.

GET IT ON Google Play Download on the App Store

Compatible hardware

WITHIN ARDUINO DEVELOPMENT ENVIRONMENTS

ARDUINO

Cloud Applications can be developed using the Arduino Cloud Editor or Arduino IDE 2.

ESP32/ESP8266

+70% Of Arduino Cloud active users use ESP-based boards.

OUTSIDE ARDUINO DEVELOPMENT ENVIRONMENTS

Use your favourite programming environment and language to connect your devices to the Cloud.

Third party platform integration

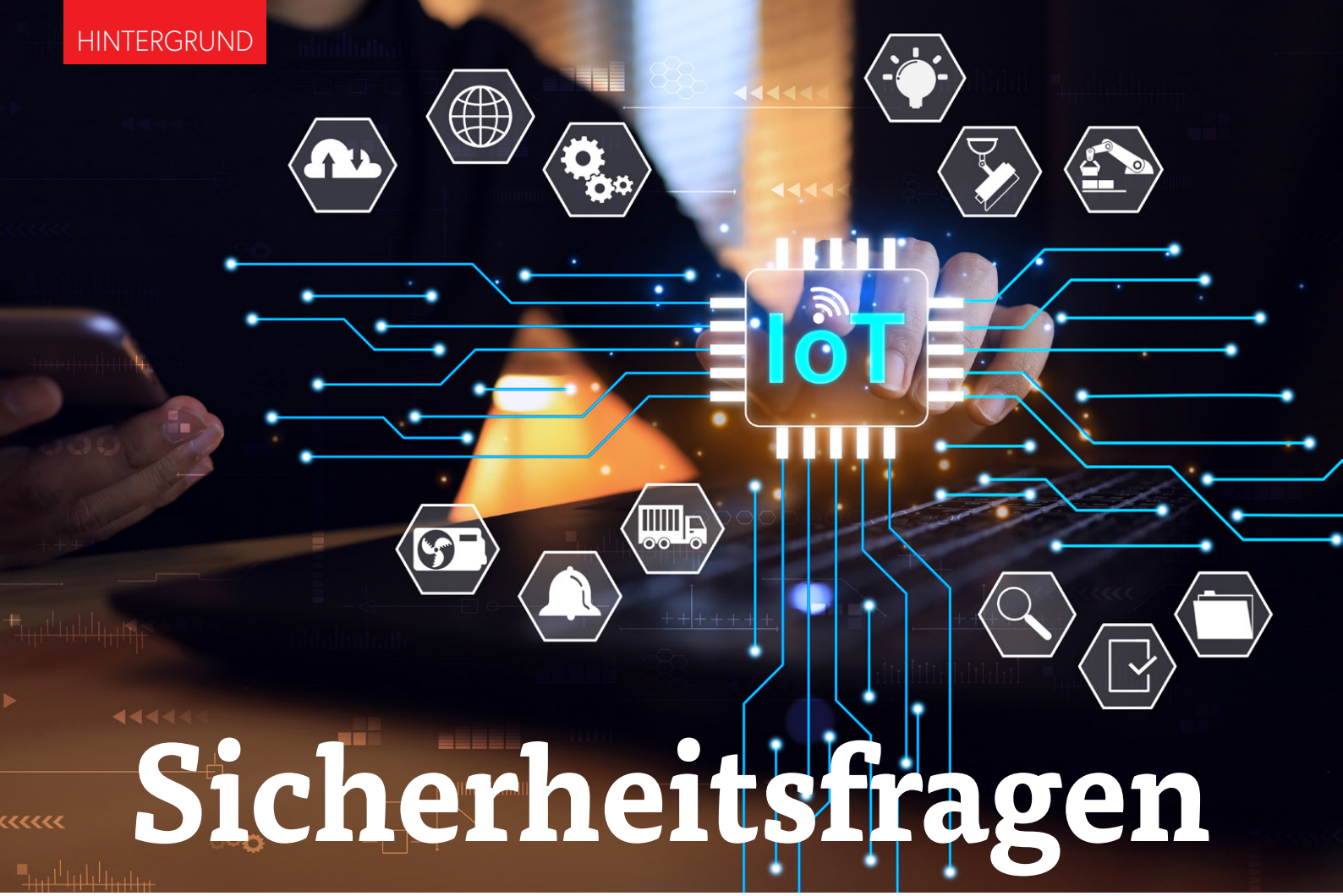
TRIGGER ACTIONS ON THIRD PARTY PLATFORMS

Connect your Arduino Cloud devices to external platforms such as IFTTT, Zapier and Google Services using webhooks and unlock endless possibilities.

Seamlessly integrate your IoT devices with over 2 000 apps, enabling tasks like recieving phone notifications, automating social media updates, streamlining data logging to external files, creating calendar events, or sending e-mail alerts.

Get 30% off on the yearly Maker plan with code ELEKTOR30*

cloud.arduino.cc/elektor



Sicherheitsfragen in der IoT-Fertigung

Wann, wie und warum

Von Aditya Patwardhan, Espressif

Die zunehmende Intelligenz der Geräte „at the Edge“ und vor allem ihre kontinuierliche Weiterentwicklung erfordern ein höheres Sicherheitsniveau und eine ständige dynamische Anpassung durch die Hersteller. Hier sehen wir, wie Secure IoT Manufacturing von Espressif auf diese sich ändernden Anforderungen antwortet.

IoT, das Internet der Dinge, hat sich nahtlos in unser tägliches Leben integriert. Dank des Fortschritts bei hochentwickelten Entwicklungs-Frameworks ist die Erstellung neuer IoT-Produkte deutlich einfacher geworden. Da sich die IoT-Branche mit fortlaufenden Innovationen ständig weiterentwickelt, werden die Anwendungen immer komplexer. Ein wichtiger Aspekt dieser Entwicklung ist, dass sich die Menschen immer mehr Gedanken über die Sicherheit machen. Auf dem Markt gibt es jetzt strengere Sicherheitsstandards mit jeweils eigenen Anforderungen. Dies hat die Herstellung von IoT-Geräten komplizierter gemacht und erfordert eine gründliche Sicherheitsprüfung bei jedem Schritt.

In diesem Artikel gehen wir auf verschiedene Aspekte der sicheren IoT-Herstellung ein. Wir erörtern, welche Probleme bei jedem Schritt auftreten und wie wir bei Espressif es unseren Kunden leicht machen.

Geheimnisse und eindeutige Daten

Die älteren Prozesse der Herstellung MCU-basierter Geräte waren relativ einfach. Nach dem Flashen der Firmware auf die MCU und der Durchführung einer Reihe von Qualitätssicherungsprüfungen war das Gerät bereit für den Einsatz in der Praxis. Die Bedingungen haben sich jedoch erheblich verändert, als das Thema Konnektivität ins Spiel kam. Heutzutage verlangen die meisten IoT-Produkte mehr als nur eine eindeutige MAC-Adresse; sie benötigen individuelle Anmeldedaten, die für jedes Gerät einzigartig sind. Einige davon, die für die sichere Kommunikation mit Remote-Servern benötigt werden, sind geheim, andere sind geheim, weil sie für das sichere Onboarding des Geräts verwendet werden, und wieder andere sind geheim, weil

sie spezifisch für die MCU sind, zum Beispiel, wenn es um die Verschlüsselung der im Flash gespeicherten Daten im Ruhezustand geht. Kryptografie auf der Grundlage der Public Key Infrastructure (PKI) spielt eine wichtige Rolle bei der Gerätesicherheit. In der Regel verfügt ein Gerät über einen öffentlichen Schlüssel, mit dem der Server, mit dem es kommuniziert, direkt oder indirekt authentifiziert werden kann, einen öffentlichen Schlüssel, mit dem die Authentizität einer neuen Firmware, die auf dem Gerät installiert wird, überprüft werden kann, und ein öffentlich-privates Schlüsselpaar, das ein eindeutiges Gerätezertifikat bildet, das beim Cloud-Server oder der zentralen Datenbank registriert wird, um die Authentifizierung des Geräts von der Cloud oder einem Client aus zu ermöglichen. Darüber hinaus muss das Gerät auch sensible Informationen speichern, zum Beispiel die Anmeldedaten für das WLAN. Anhand dieses typischen Falles können Sie im Folgenden die wichtigsten Sicherheitsanforderungen erkennen:

1. Die öffentlichen Schlüssel, die zur Authentifizierung anderer Einheiten verwendet werden, dürfen nicht verfälscht werden.
2. Der private Schlüssel des Geräts muss in einer sicheren Umgebung mit einem Zufallszahlengenerator guter Qualität erzeugt werden.
3. Der private Schlüssel, der die Identität des Geräts darstellt, muss sicher auf dem Gerät gespeichert sein, so dass er nicht in die Hände von Angreifern fallen kann.
4. Das Gerätezertifikat wird von einer Einrichtung signiert, die für die Signierung von Zertifikaten vertrauenswürdig ist.
5. Sensible Informationen wie WLAN-Anmeldedaten müssen in einem verschlüsselten Format im Flash-Speicher gespeichert werden, so dass einfache Flash-Lesevorgänge diese Informationen nicht offenlegen können.

Daraus ergeben sich weitere grundlegende Sicherheitsanforderungen:

1. Das Gerät darf nur die vertrauenswürdige Firmware ausführen und nicht mit bösartigem Code programmiert werden können, der sensible Daten preisgibt.
2. Jedes Gerät muss seinen Flash-Speicher verschlüsseln können; vorzugsweise mit einem eindeutigen Chiffrierungsschlüssel, so dass selbst dann, wenn dieser Schlüssel gefunden wird, nicht die gesamte Geräteflotte gefährdet ist. Vorzugsweise sollte dieser eindeutige Chiffrierungsschlüssel innerhalb des Chips nach dem Zufallsprinzip generiert werden und nicht außerhalb des Chips zugänglich sein.
3. Der Herstellungsprozess sollte einen privaten Schlüssel auf dem Chip generieren, der nicht von außen zugänglich sein darf.
4. Der Herstellungsprozess sollte mit vertrauenswürdiger Hardware erfolgen oder mit einem Cloud-Dienst für die Zertifikatsignierung arbeiten.

Komplexität bei der Herstellung

Obwohl moderne Mikrocontroller wie die ESP32-Serie die Sicherheitsfunktionen bieten, die zur Umsetzung der oben genannten Anforderungen nötig sind, muss man bei der Herstellung dennoch große Sorgfalt walten lassen. Auftragshersteller, bei denen diese Art der Geräteprogrammierung stattfindet, verfügen in der Regel nicht über Wissen über die besten Sicherheitspraktiken, und die Komplexität der sicheren Fertigung kann überwältigend sein. Espressif bietet mehrere Lösungen, um dies zu vereinfachen.

Automatische Secure-Boot-Aktivierung

Secure Boot ist die Funktion, die bewirkt, dass die Hardware die Firmware auf dem Chip bei jedem Startvorgang authentifiziert. Zu diesem Zweck wird der Chip mit einem öffentlichen Schlüssel gesperrt, der die in der signierten Firmware vorhandene Signatur auf Authentizität überprüfen kann. ESP-IDF von Espressif ermöglicht die Aktivierung eines sicheren Bootvorgangs in der zweiten Bootloader-Stufe. Dadurch wird erreicht, dass der Bootloader, wenn er auf dem Chip programmiert und zum ersten Mal ausgeführt wird, den einmal programmierbaren Speicher (One Time Programmable, OTP) mit dem öffentlichen Schlüssel programmiert, der von der Hardware für den sicheren Start verwendet wird. Der Bootloader kann sich auch um die Deaktivierung der Debug- und Programmierschnittstellen kümmern, um den Chip zu sperren. Auf diese Weise muss Ihr Fertiger nicht die Verantwortung für die korrekte Programmierung des OTP übernehmen, um das Gerät zu sichern. Dies alles kann vom Entwickler erledigt werden, der sich mit Sicherheit mit Sicherheit besser auskennt als der Fertiger des Geräts. **Bild 1** veranschaulicht das transitive Vertrauensmodell für sicheres Booten.

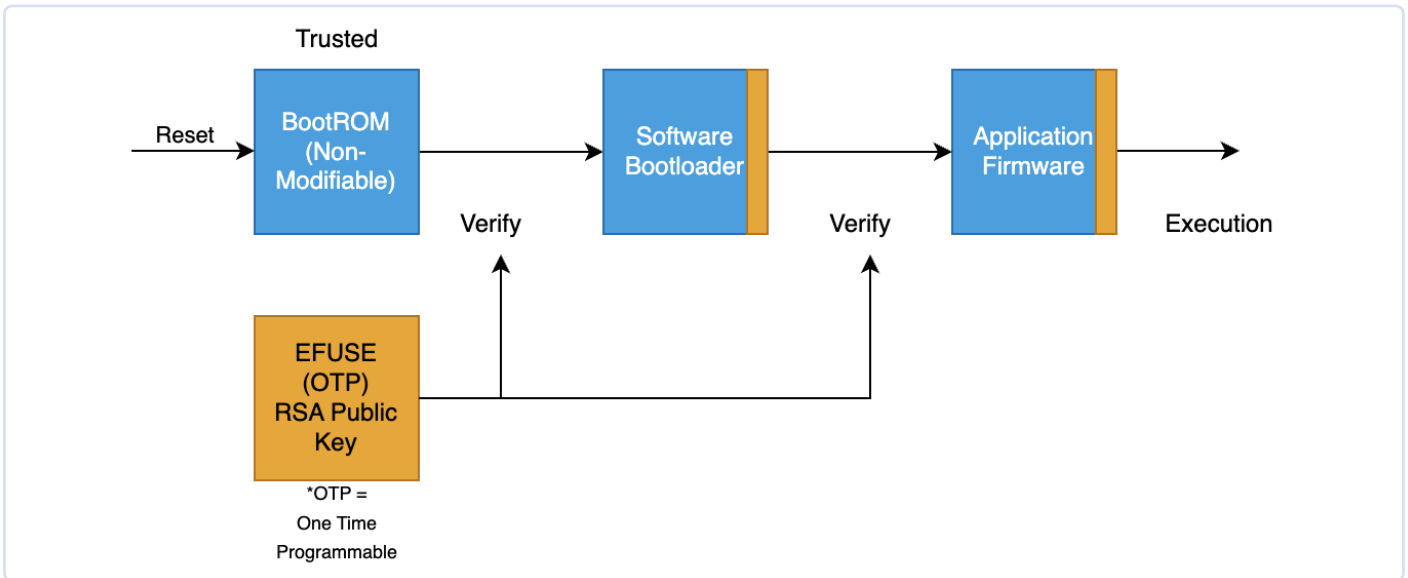


Bild 1. Diagramm des transitiven Vertrauensmodells für Secure Boot.

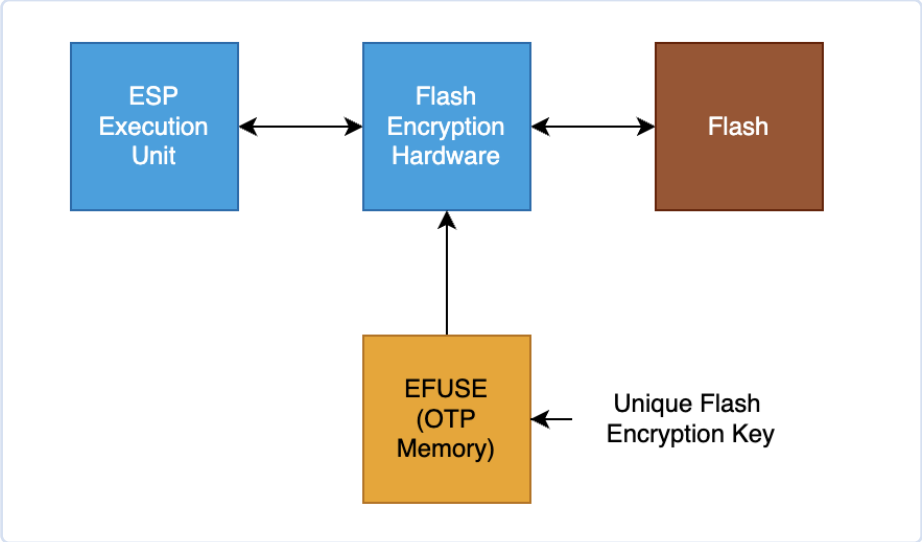


Bild 2. Vereinfachte schematische Darstellung der Flash-Verschlüsselung.

Sobald die Secure-Boot-Funktion aktiviert ist, kann der Chip nicht nur vertrauenswürdige Firmware ausführen, sondern die vertrauenswürdige Firmware kann auch die verschiedenen oben erwähnten öffentlichen Schlüssel enthalten, und diese öffentlichen Schlüssel können nicht gewaltsam geändert werden.

Aktivierung der automatischen Flash-Verschlüsselung

Wir haben die Anforderung diskutiert, einen eindeutigen, symmetrischen Schlüssel im Gerät zu haben, der zur Verschlüsselung des Flash-Speichers verwendet wird. Die Mikrocontroller der ESP32-Serie verfügen über eine Funktion zur Flash-Verschlüsselung, bei der die Hardware einen per Software nicht lesbaren Flash-Chiffrierungsschlüssel im OTP unterstützt. Dieser Schlüssel kann den Flash-Inhalt transparent und dynamisch ver- und entschlüsseln. Der Software-Bootloader von ESP-IDF ermöglicht die Aktivierung dieser Funktion zur Laufzeit, wobei der Software-Bootloader beim ersten Booten des Geräts den Flash-Verschlüsselungsschlüssel im Chip mit seinem „echten" Zufallszahlengenerator (TRNG) generieren kann und ihn mit aktiviertem Ausleseschutz in den OTP-Speicher programmiert. Der Software-Bootloader kann optional auch die Firmware und andere erforderliche Flash-Inhalte verschlüsseln, die beim ersten Booten verschlüsselt werden müssen.

Auf diese Weise ermöglicht die Flash-Verschlüsselung, wenn sie über den Software-Bootloader aktiviert wird, für jedes Gerät die Generierung eines zufälligen Flash-Chiffrierungsschlüssel pro Gerät und eine sichere Flash-Verschlüsselung, die zum Schutz sensibler Gerätedaten wie WLAN-Anmeldeinformationen oder des privaten Schlüssels des Gerätezertifikats verwendet werden kann. **Bild 2** zeigt (vereinfacht) die Flash-Verschlüsselung.

Pre-Provisioning eines Sicherheitszertifikats

Für das Gerätezertifikat haben wir festgestellt, wie wichtig es ist, die Erstellung und Speicherung des privaten Schlüssels zu schützen. Außerdem ist es wichtig, das Gerätezertifikat von einer vertrauenswürdigen Zertifizierungsstelle (CA) signieren zu lassen. Espressif wendet in seinen Fabriken ein sicheres Verfahren zur Bereitstellung von Zertifikaten an, das es den Kunden ermöglicht, vorbereitete Module zu bestellen.

In diesem Prozess gibt es drei zusammen arbeitende Einheiten. Ein *Provisioning Host* in Form eines PCs arbeitet mit einem ESP32-Chip oder -Modul für die Provisionierung. Der Provisioning-Host ist mit einem lokalen oder cloudbasierten *Hardware-Sicherheitsmodul* (HSM) verbunden, das die Zertifizierungsstelle enthält und jedes Zertifikat signieren kann, ohne den für die Signierung verwendeten privaten Schlüssel preiszugeben. Die meisten HSMs sind auch in der Lage, nicht widerlegbare Protokolle zu erstellen, die Aufschluss darüber geben, wie viele Zertifikate von dem HSM signiert wurden. Der Prozess ist in **Bild 3** dargestellt.

Der private Schlüssel des Gerätezertifikats wird auf dem Chip erzeugt und verlässt diesen nie. Der Host gibt Zertifikatsparameter wie Gültigkeitsdauer, gemeinsamer Name und so weiter an und erhält den Certificate Signing Request (CSR). Die CSR wird dann an das lokale oder Cloud-HSM gesendet, um ein signiertes Zertifikat zu erhalten. Dieses Zertifikat wird dann an das Gerät gesendet,

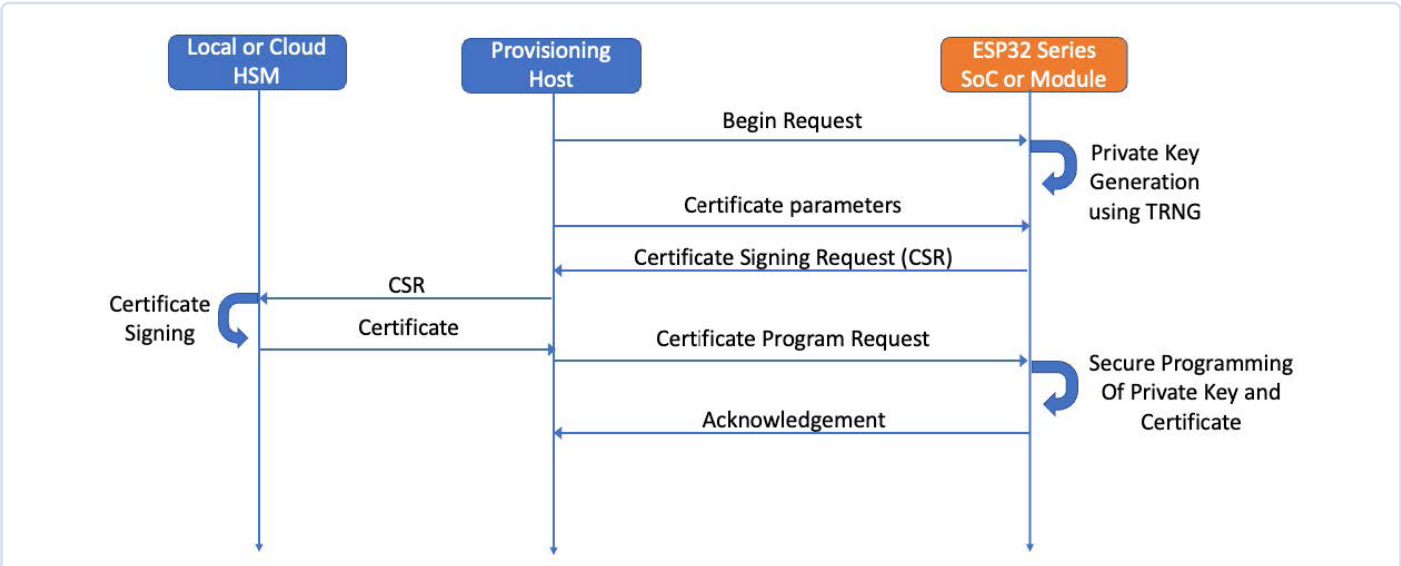


Bild 3. Flussdiagramm des Prozesses der sicheren Zertifikatsprovisionierung.

und das Gerät wird dann mit Secure Boot und Flash-Verschlüsselung gesperrt. Viele der neuen Chips der ESP32-Serie verfügen auch über eine dedizierte Peripherie zur *Digital Signature* (DS), die den privaten Schlüssel des Geräts mit einem speziellen Hardwareblock schützt. Diese DS-Peripherie kann Zertifikatsoperationen direkt mit dem verschlüsselten privaten Schlüssel durchführen, wodurch der private Schlüssel im Klartext auch für die Software unzugänglich ist.

Andere eindeutige Daten pro Gerät

Neben diesen Sicherheitsartefakten muss das Gerät möglicherweise auch über andere eindeutige Daten pro Gerät verfügen, zum Beispiel eine Geräte-ID. Espressif bietet benutzerfreundliche Tools zur Generierung von Binärdateien mit gerätespezifischen Daten aus einer CSV-Datei, die eindeutige gerätespezifische Daten in Tabellenform enthält. Bitte sehen Sie sich dazu [1] an.

Über den Autor

Aditya Patwardhan ist Software-Ingenieur bei Espressif und verfügt über mehr als vier Jahre Berufserfahrung. Seine Interessengebiete umfassen Systeme, Sicherheit, maschinelles Lernen und die spannende Welt des IoT. Aditya hält sich leidenschaftlich gern über neue Entwicklungen im Sicherheitsbereich auf dem Laufenden und nutzt diese Entwicklungen, um robuste und sichere IoT-Anwendungen zu entwickeln.

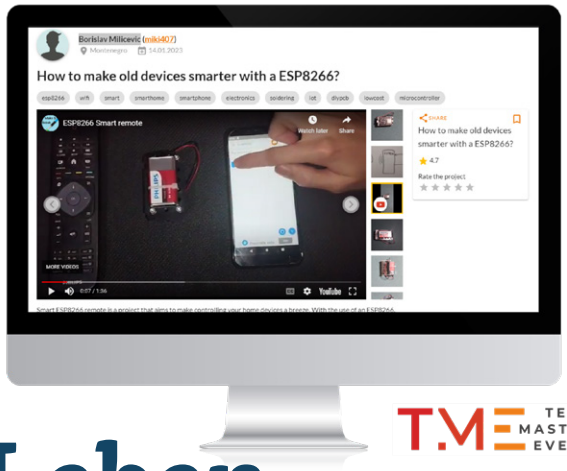
Alles kommt zusammen

Sicherheitsanforderungen und die Programmierung eindeutiger gerätespezifischer Daten machen die Herstellung von IoT-Geräten kompliziert. Espressif bietet eine große Flexibilität im Fertigungsprozess, die es den Kunden ermöglicht, Sicherheitsfunktionen wie Secure Boot und Flash-Verschlüsselung in der Espressif-Fertigungslinie zu aktivieren, ohne die Sicherheit zu beeinträchtigen. Der Pre-Provisioning-Prozess für Module von Espressif ermöglicht die sichere Programmierung von Gerätezertifikaten. Espressif ist auch eine von der *Connectivity Standards Alliance* (CSA) zugelassene *Product Attestation Authority* und kann Chips und Module mit den *Device Attestation Certificates* (DAC) vorbereiten, die für die Herstellung von Matter-konformen Geräten erforderlich sind. Darüber hinaus unterstützt Espressif auch bei der kundenspezifischen Firmware-Programmierung und der Vorprogrammierung einzigartiger Daten, was die Herstellung von IoT-Geräten erheblich vereinfacht.

RG -- 230638-02

WEBLINK

[1] Fertigungshilfsmittel: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/storage/mass_mfg.html



Für ein einfacheres und komfortableres Leben

Ein Amateurprojekt basierend auf dem ESP8266-Modul von Espressif

Ein Betrag von Transfer Multisort Elektronik Sp. Z.o.o.

Mit jedem Jahr wird das Konzept des SmartHome immer populärer und die Verfügbarkeit von hilfreichen Lösungen, unseren Lebensraum effizienter zu verwalten, wächst. Darüber hinaus sind einige Produkte auf dem Markt, die mit älteren Geräten kompatibel sind, so dass wir diese zusammen mit den neuesten technologischen Errungenschaften nutzen können. Die Fernsteuerung von Haushaltsgeräten und die Automatisierung verschiedener Prozesse helfen, die Energieeffizienz zu verbessern, die Umwelt zu schützen, unseren Komfort zu erhöhen und Geld zu sparen. Das Projekt Smart ESP8266 remote, das für einen Wettbewerb von TechMasterEvent entwickelt wurde, vereint all diese Vorteile.

Espressif ist ein Hersteller integrierter SoC-Schaltungen und drahtloser Übertragungsmodule, von denen viele bei TME erhältlich sind. Dank ihrer kompakten Größe und ihres geringen Energiebedarfs können Espressif-Produkte sowohl in der Konsum- als auch in der Industrieelektronik erfolgreich eingesetzt werden.

Im Folgenden sollen Sie ein Gerät auf der Basis des ESP8266-Moduls kennenlernen. Es handelt sich um ein Amateurprojekt, das von einem Teilnehmer eines TechMasterEvent-Wettbewerbs entwickelt wurde. Die Teilnehmer sollten ein Elektronikprojekt entwerfen, das „das Leben einfacher macht". Sie finden das ESP8266-Modul sowie verschiedene andere Komponenten, die beim Bau Ihrer IoT-Projekte nützlich sein können (Einplatinencomputer, Kommunikations- und Speichermodule, Displays und vieles mehr) unter [1].

ESP8266, IR-LED und IR-Empfänger

Die intelligente ESP8266-Fernbedienung ist ein Projekt, das die Steuerung von Haushaltsgeräten zum Kinderspiel machen soll. Durch die Verwendung eines ESP8266, einer IR-LED und eines IR-Empfängers macht dieses Projekt mehrere Fernbedienungen für verschiedene Geräte wie Klimaanlagen oder Fernseher überflüssig. Das Projekt lässt sich mit einer Telefon-App verbinden, so dass die Benutzer ganz einfach Befehle vom Handy an ihre Geräte senden und sogar die von ihren aktuellen Fernbedienungen gesendeten Signale zur späteren Verwendung speichern können.

Die smarte ESP8266-Fernbedienung ist nicht nur bequem und einfach zu bedienen, sondern auch eine großartige Lösung für ältere Geräte, die möglicherweise nicht mit herkömmlicher Smart-Home-Technologie kompatibel sind.

Da die smarte ESP8266-Fernbedienung Signale von herkömmlichen Fernbedienungen lesen und speichern kann, ermöglicht sie die Steuerung älterer Geräte, die möglicherweise nicht mit dem Internet oder anderen Smart-Home-Systemen verbunden werden können. Dies macht sie zu einer kostengünstigen Alternative zur Aufrüstung Ihrer Geräte oder zum Neukauf teurer Smart-Home-Geräte. Die IR-LED und der IR-Empfänger werden zum Senden beziehungsweise Empfangen von IR-Signalen verwendet, die die Haushaltsgeräte steuern. Das Projekt kann Signale von herkömmlichen Fernbedienungen lesen und speichern, so dass der Benutzer ältere Geräte steuern kann, die möglicherweise nicht die Fähigkeit haben, sich mit dem Internet oder anderen Smart-Home-Systemen zu verbinden. Zusätzlich zu den Hardwarekomponenten benötigt das Smart-ESP8266-Fernbedienungsprojekt auch Software, die Sie unter [2] finden.

Die smarte ESP8266-Fernbedienung bietet mehrere Vorteile wie Bequemlichkeit und Benutzerfreundlichkeit, Kosteneffizienz und Flexibilität. Sie macht den Kauf teurer Smart-Home-Geräte oder die Aufrüstung älterer Geräte überflüssig und ist damit eine kostengünstige Alternative. Das Projekt ist außerdem so flexibel, dass es leicht modifiziert oder angepasst werden kann, um mit verschiedenen Geräten und unterschiedlichen IR-Protokollen zu arbeiten, was es zu einer vielseitigen Lösung für die Steuerung verschiedener Geräte mit einer einzigen App macht.

RG – 230656-02

WEBLINKS

[1] TME-Shop: http://www.tme.eu/de/
[2] Quellcode für dieses Projekt: https://t1p.de/9hlzn

MACNICA
ATD EUROPE

Your official authorized distributor
in Europe for Espressif Systems

ESPRESSIF

Macnica ATD Europe
+49 (0)89 899 143-11
sales.mae@macnica.com
www.macnica-atd-europe.com

Blynk

Wie man IoT-Apps ohne Software-Expertise erstellt

Mit der Blynk IoT-Plattform und Espressif-Hardware

Ein Beitrag von Blynk Inc.

Stellen Sie sich vor, Sie könnten eine mobile App entwickeln, ohne eine Zeile Code zu schreiben, sie individuell gestalten und innerhalb eines Monats in den App-Stores veröffentlichen. Eine professionelle IoT-Software ohne Software-Ingenieure starten? Mit Blynk IoT ist dies innerhalb eines Monats und nicht in Jahren möglich!

Die Blynk-Firmware-Bibliothek unterstützt:
- ESP32
- ESP32-S2
- ESP32-S3
- ESP32-C3
- ESP8266
- und Andere

Was ist in der Blynk IoT-Plattform enthalten?

Blynk ist eine Low-Code-IoT-Softwareplattform mit Cloud, Firmware-Bibliotheken, einem nativen mobilen No-Code-App-Builder und einer Webkonsole zur Verwaltung. Sie erhalten WLAN-Gerätebereitstellung, Datenvisualisierung, Automatisierung, Benachrichtigungen, OTA-Updates und ein robustes Benutzer- und Gerätemanagementsystem als integrierte Standardfunktionen [1].

App-Builder von Blynk für iOS und Android

Der App-Builder ermöglicht die Erstellung von schnellen Prototypen und voll funktionsfähigen Apps ohne Programmierkenntnisse. Im Konstruktionsmodus können Sie aus über 50 anpassbaren UI-Elementen wählen und diese per Drag-and-Drop auf die Arbeitsfläche ziehen, um eine benutzerdefinierte Benutzeroberfläche für Ihr vernetztes Produkt zu erstellen.

Web-Dashboard-Builder

Es bietet eine ähnliche Architektur zur Erstellung von historischen und Echtzeit-Datenvisualisierungen sowie zur Steuerung und Überwachung von Geräten mit vorgefertigten UI-Elementen.

Bild 1. No-Code-Schnittstellen erstellt mit Blynk.

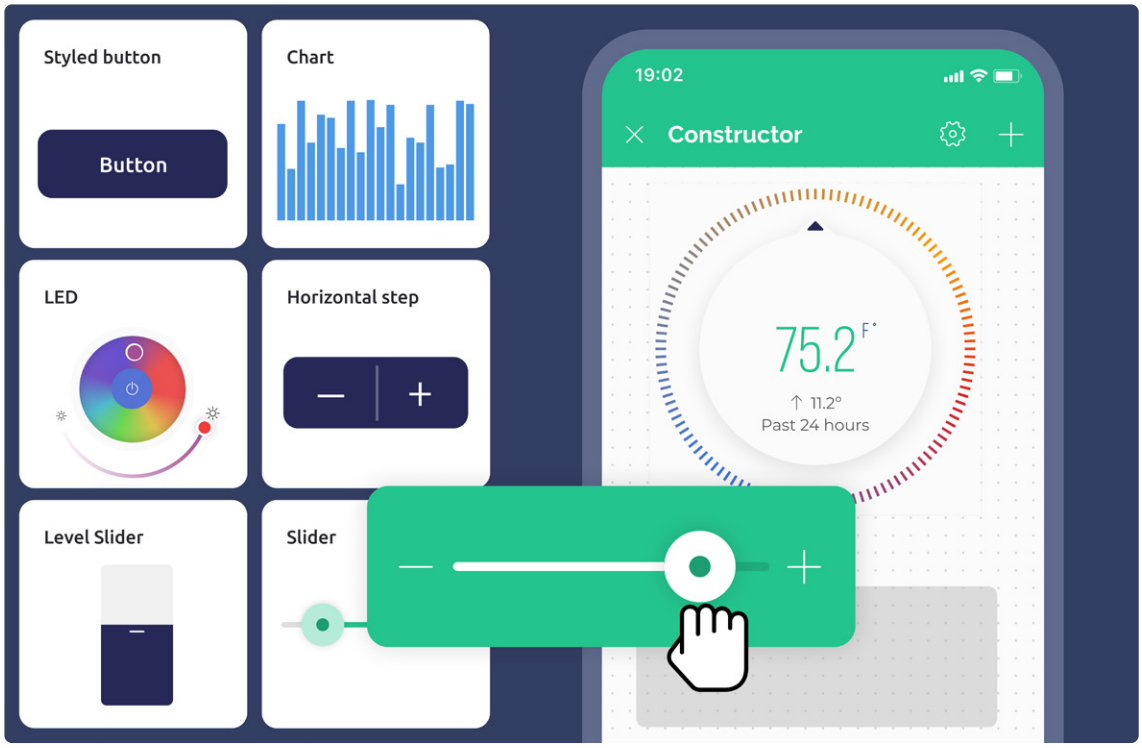


Bild 2. Blynk Drag-n-Drop App Builder.

Erweitertes Benutzermanagementsystem

Es hält alles strukturiert, auch im Unternehmensmaßstab. Sie können eine mehrstufige Organisationsstruktur erstellen und Geräte sowie Benutzerrollen und -berechtigungen verwalten.

Integriertes Geräte-Lebenszyklusmanagement

Diese Funktion deckt alle Anforderungen im Zusammenhang mit Token-Management und WLAN-Bereitstellung ab. Sie bietet zuverlässige und sichere OTA-Firmware-Updates über eine einfache Schnittstelle.

No-Code-Automatisierungsszenarien

Diese können basierend auf Datum, Tageszeit, Benutzeraktionen oder Gerätezustand eingestellt werden. Sie können Benutzer über wichtige Ereignisse auf der Hardware über Pushes, In-Apps, E-Mails oder SMS informieren.

Wie verbinden Sie Ihr ESP mit Blynk? Wie groß ist der Integrationsaufwand?

Je nach Ihrer Hardware-Konfiguration können Sie sich für *Blynk.Edgent* [3] bei einer Single-MCU-Konfiguration oder für *Blynk.NCP* [4][5] bei einer Dual-MCU-Konfiguration entscheiden. Beide Wege verfügen über alle integrierten Blynk-IoT-Funktionen und eine gesicherte Verbindung zur Blynk Cloud.

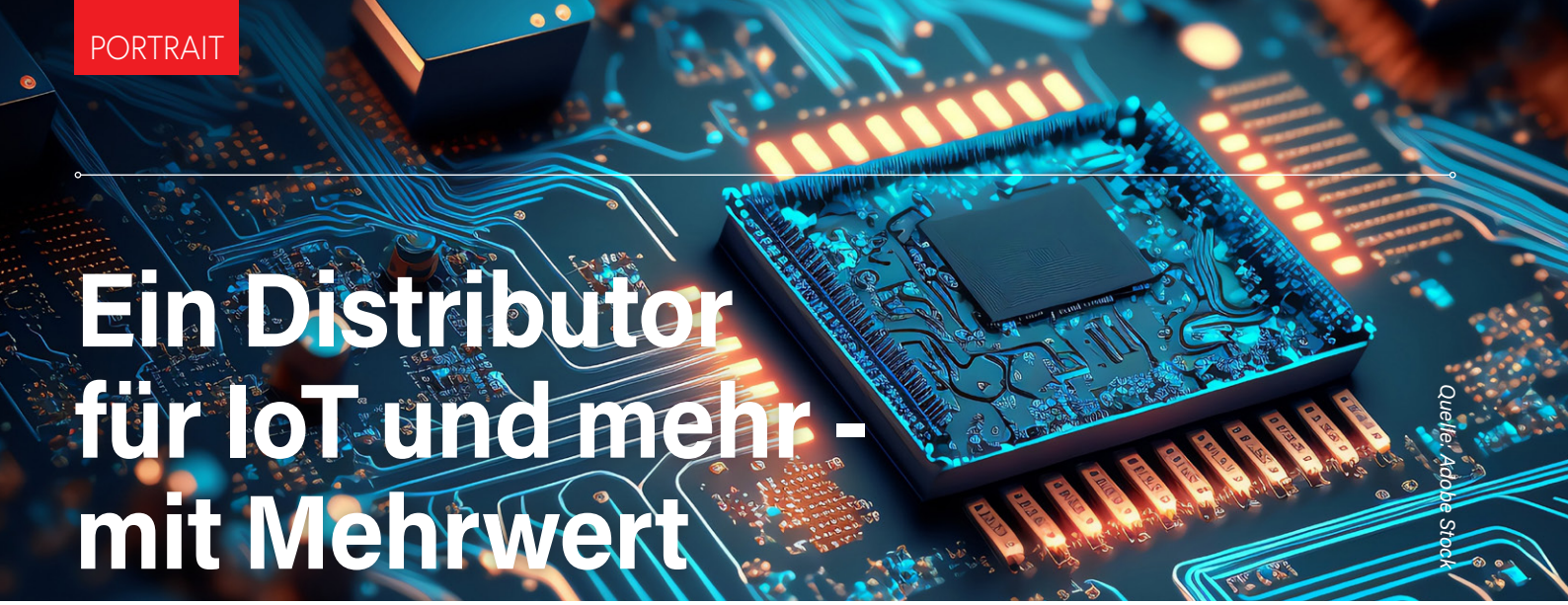
Mit den von Blynk bereitgestellten Code-Beispielen ist der Aufwand minimal. Bei der Dual-MCU-Konfiguration verwenden Sie eine fertiges Binärdatei für den NCP und eine leichte Bibliothek für den primären MCU, der über die UART-Schnittstelle mit dem NCP kommuniziert.

Ihre Reise von der Gerätekonfiguration bis zur vollständigen IoT-Infrastruktur und App-Einführung kann nur wenige Wochen dauern [6].

230659-02

WEBLINKS

[1] Offizielle Webseite: https://bit.ly/blynk-io
[2] Blynk.Console - Erstellen Sie Ihr kostenloses Konto: https://bit.ly/web-cloud
[3] Blynk.Edgent Dokumentation: https://bit.ly/docs-edgent
[4] Blynk.NCP Dokumentation: https://bit.ly/docs-ncp
[5] Was ist Blynk.NCP: https://bit.ly/info-ncp
[6] Fertiges Wetterstationsprojekt zum Ausprobieren: https://bit.ly/blueprint-weather



Ein Distributor für IoT und mehr - mit Mehrwert

Quelle: Adobe Stock

Ein Beitrag von Steliau Technology

Steliau Technology ist ein innovatives Unternehmen, das sich auf elektronische Lösungen spezialisiert hat. Das Unternehmen zeichnet sich durch sein Fachwissen im Ingenieurwesen und seine Leidenschaft für Innovation aus. Steliau Technology bietet über seinen Partner Espressif wichtige elektronische Komponenten für drahtlose Konnektivität und IoT an, wie z.B. ESP32-C5, ESP32-C6, ESP32-P4, ESP32-S6 und viele weitere Produkte.

Steliau Technology [1] wurde 2018 gegründet und versteht sich als Mehrwertvermarkter von elektronischen Lösungen. Mensch-Maschine-Schnittstellen, Touchscreens und -Lösungen, Konnektivität und IoT - all diese Fachgebiete werden von Steliau weitgehend beherrscht und verleihen dem Unternehmen einen guten Ruf auf dem Elektronikmarkt.

Steliau Technology ist für seine strategischen Partnerschaften mit führenden Elektronikunternehmen bekannt, durch die es seine Position in den Bereichen IoT und Konnektivität stärken kann.

Espressif und Steliau Technology pflegen eine historische Partnerschaft und als erster Vertriebspartner eine über Jahre hinweg gefestigte Beziehung. Als offizieller Vertriebspartner für Frankreich und Italien ist Steliau Technology die zentrale Anlaufstelle für Espressif-Lösungen.

Steliau Technology ist bestens gerüstet, um einen umfassenden Support für die gesamte Espressif-Produktpalette anzubieten, sowohl was die Hardware als auch eingebettete Software betrifft. Das Team verfügt über ein umfassendes technisches Fachwissen, das alle Aspekte der Konnektivität abdeckt, vom Hardwaredesign bis zur Softwareprogrammierung. Das bedeutet, dass Steliau Technology in der Lage ist, den Kunden umfassende Unterstützung zu bieten und so robuste und leistungsfähige Konnektivitätslösungen zu gewährleisten.

Diese starke Partnerschaft garantiert unseren Kunden einen privilegierten Zugang zu den besten Konnektivitätslösungen auf dem Markt, zum Beispiel die neuesten Espressif -Generationen: ESP32-C5, ESP32-C6, ESP32-P4, ESP32-S6.

Steliau Technology liefert über seinen Partner Espressif elektronische Komponenten, die für die drahtlose Vernetzung und das IoT in einer Vielzahl von Märkten und Branchen unerlässlich sind, und trägt so zur ständigen Weiterentwicklung der Technologie bei.

IoT-Lösungen und mehr

Im Bereich des Internet of Things (IoT) werden oft WLAN- und Bluetooth-Mikrocontroller von Espressif eingesetzt.

ESPRESSIF By STELIAU TECHNOLOGY

Diese Komponenten sind entscheidend für Smart-Home-Anwendungen wie vernetzte Thermostate, Sicherheitskameras und Lichtsteuerungsgeräte. Sie spielen eine wichtige Rolle bei der Interoperabilität von vernetzten Produkten im Haus mit kompatiblen MATTER-Lösungen (insbesondere über Wi-Fi und Thread). Die Lösungen von Espressif werden auch in der Industrie zur Fernüberwachung, Datenerfassung und Steuerung von Maschinen eingesetzt. Darüber hinaus sind die Produkte von Espressif im Gesundheitssektor vertreten, wo sie vernetzte medizinische Geräte und Geräte zur Überwachung der körperlichen Fitness mit Strom versorgen.

Als globaler Elektronikpartner kann Steliau Lösungen anbieten, die die Produkte von Espressif für die Steuerung von Touchscreens integrieren. Steliau Technology ist in der Lage, integrierte Lösungen zu entwickeln, bei denen die Espressif-Produkte den Touchscreen steuern, und hat bereits mehrere Erfolge erzielt, insbesondere bei Bildschirmgrößen von 7 Zoll. Unsere Fähigkeit, eine umfassende Lösung anzubieten, wird durch einen speziellen technischen Support verstärkt, der all diese Bereiche abdeckt.

230661-02

WEBLINK

[1] Steliau Technology: https://steliau-technology.com/en

Haben Sie Fragen?

Steliau steht für alle Anfragen zur Verfügung. Bitte wenden Sie sich für Anfragen zu Espressif-Lösungen an remi.krief@steliau-technology.com.

The Next Era of **Microcontrollers**

ESPRESSIF

High Performance MCU

With RISC-V Dual-Core Upto 400MHz

AI Acceleation

High-Speed **MEMORY**

Powerful **IMAGE & VOICE** Processing Capabilities

HMI Capabilities

- MIPI-CSI with ISP
- MIPI-DSI - 1080P
- Capacitive Touch
- H.264 Encoding - 1080P@30fps
- Pixel Processing Accellerator

Best-in-Class Security

- Cryptographic Accelerators
- Secure Boot, Flash Encryption
- Private Key protection
- Access Controls

ESP32-P4

Connectivity

- USB2.0 High Speed
- Ethernet
- SPI
- SDIO3.0
- UART
- I2C, I2S

.....


IP Camera


Touch Panel


Video Door bell


Robotic Control


Industrial Robot

www.espressif.com

Learn More About **ESP SoCs**

Schnelle und einfache IoT-Entwicklung mit M5Stack

Ein Beitrag von M5Stack

M5STACK, der weltbekannte Anbieter der modularen, auf ESP32 basierenden IoT-Entwicklungsplattform hat Hunderte von Controllern, Sensoren, Aktoren und Kommunikationsmodulen in modularer Bauweise im Angebot, die sich über Standardschnittstellen verbinden lassen. Durch das Kombinieren von Modulen mit unterschiedlichen Funktionen können Anwender die Produktprüfung und -entwicklung beschleunigen.

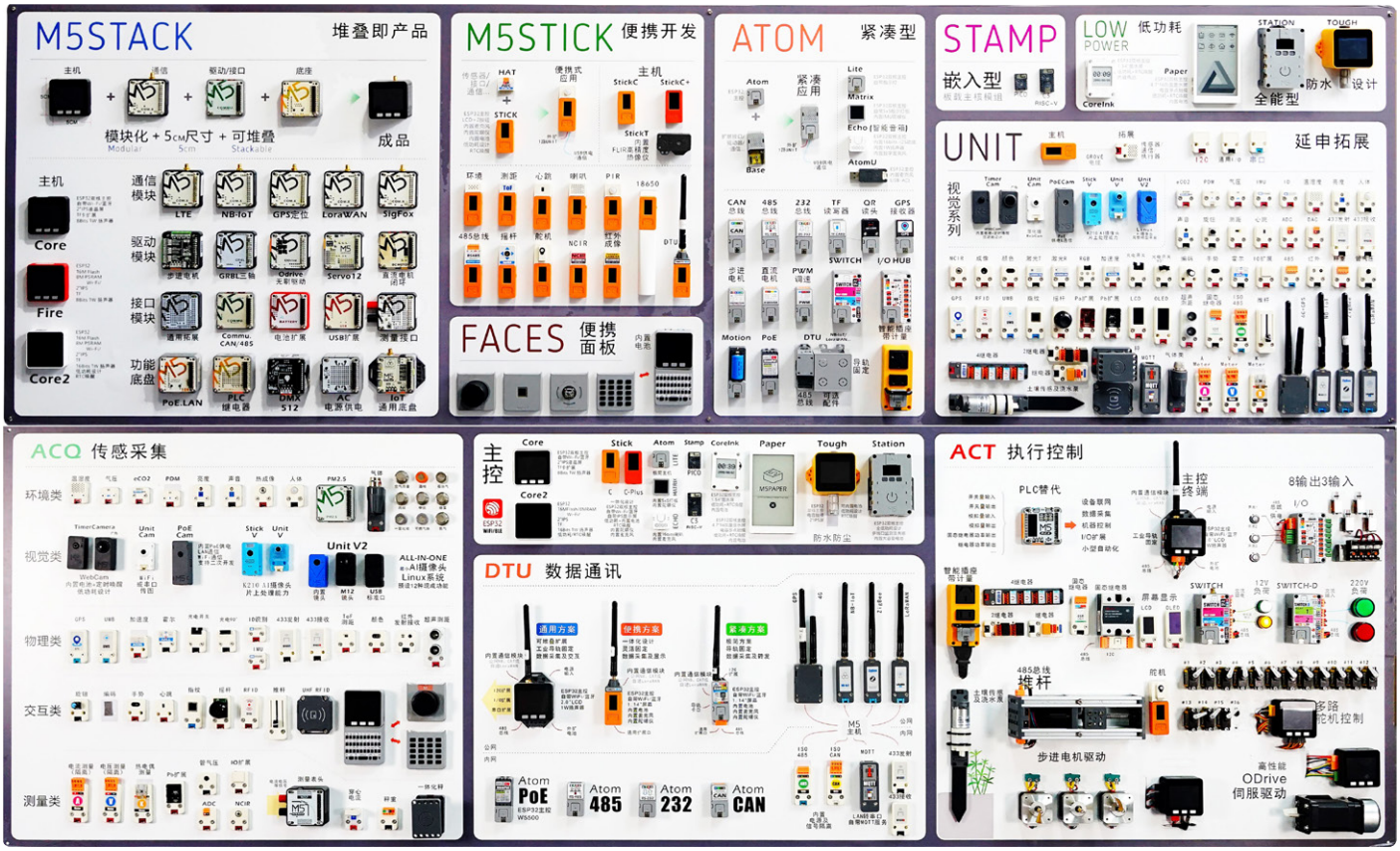


Bild 1. M5Stack-Ökosystem-Familie.

Bild 2. M5Dial ist für das Smart Home geeignet.

Die M5Stack-Module (**Bild 1**) [1] lassen sich mit der grafischen Low-Code-Programmier-IDE UIFlow „zusammenstecken", so dass Sie damit experimentieren können, um das beste Erlebnis für das Prototyping von IoT-Projekten zu erhalten. Das gilt sowohl für Einsteiger als auch für professionelle Entwickler.
Mit stapelbaren Hardware-Modulen und einer benutzerfreundlichen grafischen Programmierplattform bietet M5Stack Kunden aus den Bereichen Industrielles IoT, Gebäudeautomatisierung, Smarter Einzelhandel, Smarte Landwirtschaft und MINT eine effiziente sowie zuverlässige IoT-Entwicklungserfahrung, die schnell und einfach ist.

Neu: Der M5Dial

Das kürzlich vorgestellte M5Dial [2] eignet sich hervorragend für das Smart Home. Als vielseitiges Embedded-Entwicklungsboard vereint das M5Dial verschiedene Funktionen und Sensoren, die für die Smart-Home-Steuerung notwendig sind (**Bild 2**).

Sind Sie ein Fan von ESP32, dann ist M5Stack ein Muss!

Der Hauptcontroller des M5Dial ist der M5StampS3, ein Mikrocontroller, der auf dem ESP32-S3-Chip basiert. Er ist für seine hohe Leistung und seinen geringen Stromverbedarf bekannt. Das Gerät unterstützt WLAN- und Bluetooth-Verbindungen sowie mehrere Peripherieschnittstellen wie SPI, I²C, UART, ADC und mehr. Der M5StampS3 hat zudem 8 MByte integrierten Flash-Speicher, der ausreichend Platz für Nutzer bietet.
Der M5Dial verfügt über einen runden 1,28-Zoll-TFT-Touchscreen, einen Drehknopf, ein RFID-Erkennungsmodul, eine RTC-Schaltung, einen Summer, physische Tasten sowie andere Funktionen, womit Benutzer verschiedene Projekte leicht umsetzen können.

Das herausragende Merkmal von M5Dial ist der Drehknopf, der die Position und Richtung des Knopfes genau aufzeichnet und den Nutzern ein verbessertes interaktives Erlebnis bietet. Mit dem Drehknopf können Nutzer Einstellungen wie Lautstärke, Helligkeit und Menüs anpassen oder Haushaltsgeräte wie Licht, Klimaanlage, Vorhänge und so weiter steuern. Auf dem eingebauten Bildschirm des Geräts lassen sich zudem verschiedene interaktive Farben und Effekte darstellen. Dank seiner kompakten Größe von 45 mm × 45 mm × 32,2 mm und dem geringen Gewicht von 46,6 g lässt sich M5Dial einfach benutzen.
Ob zur Steuerung von Haushaltsgeräten im Smart Home oder zur Überwachung und Steuerung von Systemen in der Industrieautomation, M5Dial lässt sich einfach integrieren und bietet intelligente Steuerungs- sowie interaktive Funktionen.

230662-02

WEBLINKS

[1] Der Innovator der modularen IoT-Entwicklungsplattform | M5Stack: https://m5stack.com
[2] Intelligenter Drehknopf ESP32-S3 mit rundem 1,28"-Touchscreen:
https://shop.m5stack.com/products/m5stack-dial-esp32-s3-smart-rotary-knob-w-1-28-round-touch-screen

Erstellung einer intelligenten Benutzeroberfläche auf ESP32

Ein Beitrag von Slint

Smartphones haben das Nutzererlebnis von touch-basierten Steuerungen neu definiert. Die Erstellung einer intelligenten Benutzeroberfläche erfordert den Einsatz moderner Werkzeuge. In diesem Artikel teilen wir Tipps und präsentieren Slint, ein Toolkit für interaktive Benutzeroberflächen, die Nutzererwartungen erfüllen und übertreffen.

Bild 1. Logos von C++ und Rust.

Über Slint - Ein Toolkit der nächsten Generation zur Erstellung nativer, grafischer Oberflächen in C++, Rust und JavaScript mit einer breiten plattformübergreifender Unterstützung wie Bare-Metal, RTOSs und Linux und 10 K+ GitHub-Sternen.

Wählen Sie Ihre Programmiersprache – C/C++ oder Rust

In der Embedded-Programmierung sind C/C++ schon lange die beliebtesten Programmiersprachen. Aber Rust, bekannt für seine Speichersicherheit und Leistung, wird bei Embedded-Entwicklern immer beliebter.

Slint, das einzige Toolkit mit nativen APIs für C++ und Rust (**Bild 1**), bietet Entwicklern die Wahl: Schreiben der Geschäftslogik in einer der beiden Sprachen. Darüber hinaus bietet es einen Übergangspfad für diejenigen, die ihren Code von C/C++ auf Rust umstellen möchten.

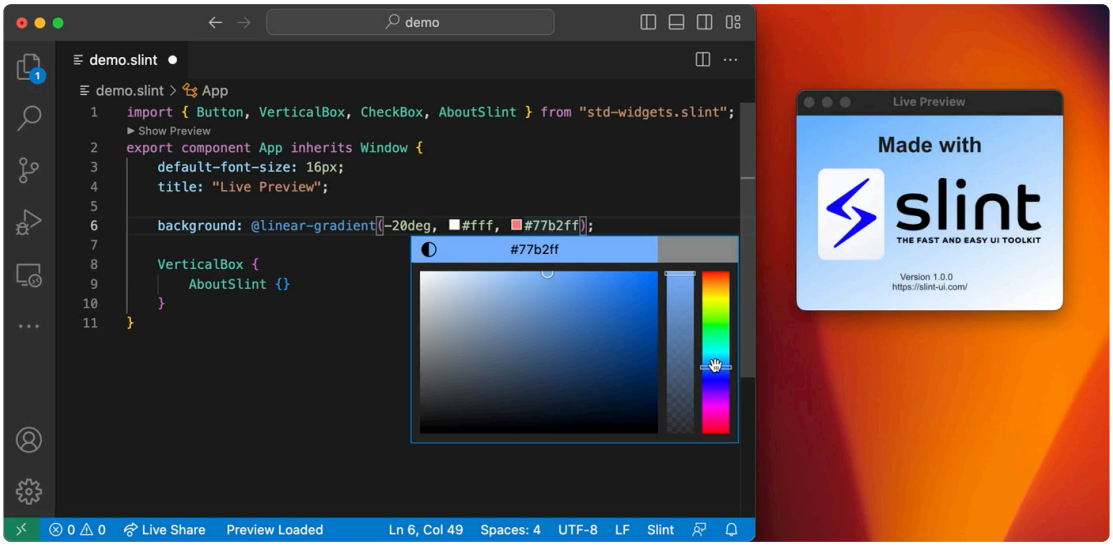


Bild 2. Schnelle Iteration mit der Live-Preview von Slint.

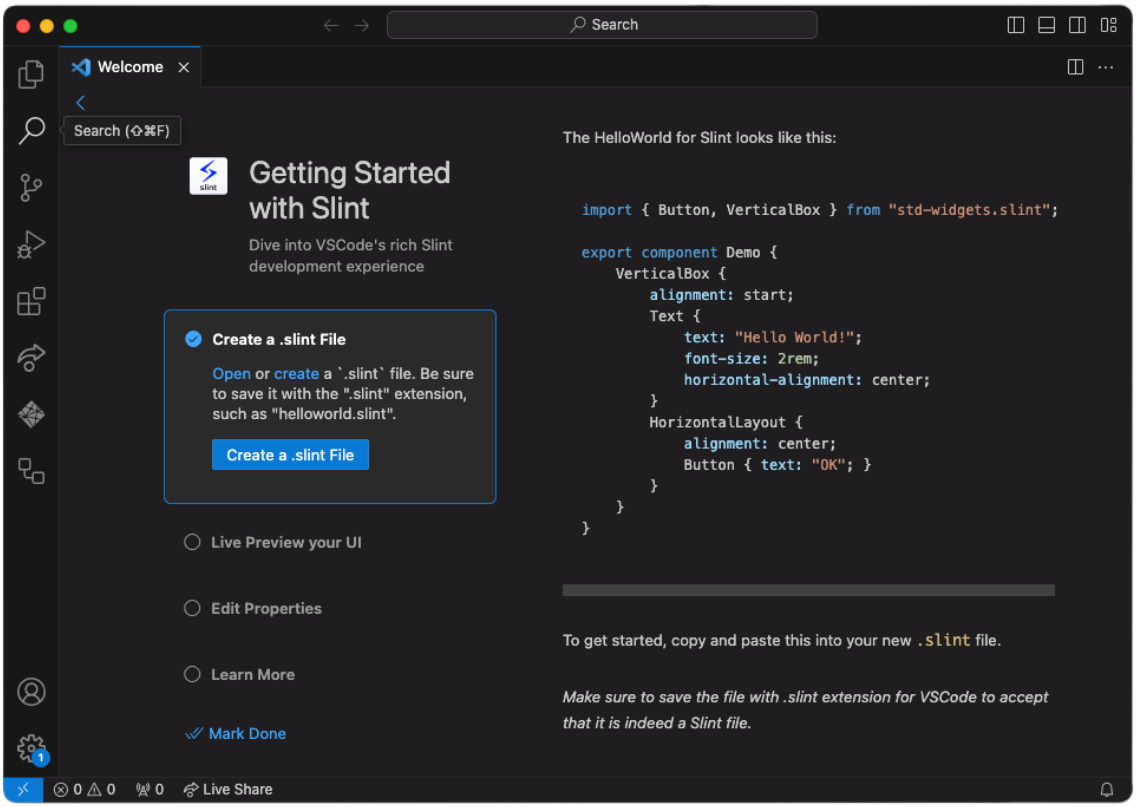


Bild 3.
Starten Sie mit Slint.

Trennen Sie das Benutzerinterface von der Geschäftslogik

MVC- und MVVM-Entwurfsmustern fördern die Trennung von Benutzerinterface und Geschäftslogik, um die Effizienz und Codequalität zu steigern.
In Slint ist die Benutzeroberfläche in einer HTML/CSS-ähnlichen Sprache definiert, die eine klare Trennung zwischen Präsentation und Geschäftslogik gewährleistet. Die Live-Preview von Slint ermöglicht die Verfeinerung des UI-Designs durch schnelle Iterationen (**Bild 2**).

Genießen Sie eine gute Entwicklererfahrung

Die heutige Komplexität in der Softwareentwicklung erfordert eine gute Entwicklererfahrung: Entwickler können mit Zuversicht arbeiten, eine größere Wirkung erzielen und sich zufrieden fühlen. Verwenden Sie weiterhin Ihre Lieblings-IDE. Wählen Sie zwischen Slint's VSCode-Extension (**Bild 3**) und dem allgemeinen Sprachserver (LSP): Nutzen Sie Codevervollständigung, Syntaxhervorhebung, Fehlerdiagnose, Live-Preview und mehr. Slint bietet eine ESP-IDF-Komponente, die die Integration mit Espressifs IoT-Development-Framework (IDF) vereinfacht.

Bieten Sie ein außergewöhnliches Nutzererlebnis

Die Leistung der Benutzeroberfläche ist entscheidend für ein außergewöhnliches Nutzererlebnis. Genießen Sie die Flexibilität im Hardware-Design mit Slints Fähigkeiten zur zeilenweisen und Framebuffer-Darstellung auf der ESP32-Plattform, um einen vielseitigen Ansatz für die Geräteentwicklung sicherzustellen (**Bild 4**).

Beginnen Sie mit Slint auf ESP32, besuchen Sie [1].

RG – 230670-02

Bild 4. Eine Slint-Demo auf ESP32.

WEBLINK

[1] Slint auf dem ESP32: https://slint.dev/esp32

Holen Sie sich die neue ESPRESSIF Hardware!

Es gibt nichts, was uns mehr begeistert, als neue Hardware in die Hände zu bekommen, und so war diese Zusammenarbeit mit Espressif ein wahrer Genuss! Möchten Sie sich davon überzeugen? Elektor hat das Lager erweitert, um alle Produkte, die in dieser Ausgabe vorgestellt werden, unterbringen zu können!

ESP32-S3-Box-3

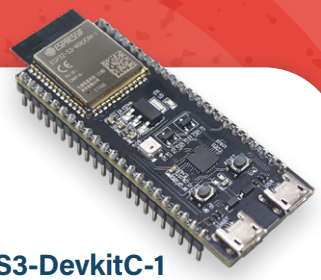
Die ESP32-S3-BOX-3 ist ein vollständig quelloffenes AIoT-Entwicklungskit, das auf dem leistungsstarken ESP32-S3 AI SoC basiert und das Feld der traditionellen Entwicklungsboards revolutionieren soll. Die ESP32-S3-BOX-3 ist mit einer Vielzahl von Add-ons ausgestattet, die es Entwicklern ermöglichen, die Funktionalität des Kits einfach anzupassen und zu erweitern.

www.elektor.de/20627

ESP32-S3-Eye

Der ESP32-S3-EYE ist ein kleines KI-Entwicklungsboard. Es basiert auf dem ESP32-S3 SoC und ESP-WHO, dem KI-Entwicklungsframework von Espressif. Es verfügt über eine 2-Megapixel-Kamera, ein LCD-Display und ein Mikrofon, die für die Bilderkennung und Audioverarbeitung verwendet werden.

www.elektor.de/20626



ESP32-S3-DevkitC-1

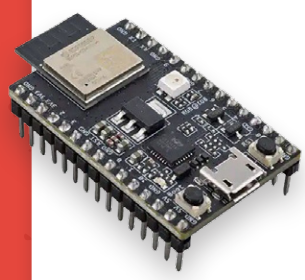
Das ESP32-S3-DevKitC-1 ist ein Entwicklungsboard der Einstiegsklasse, das mit ESP32-S3-WROOM-1, ESP32-S3-WROOM-1U oder ESP32-S3-WROOM-2 plus einem universellen Wi-Fi + Bluetooth Low Energy MCU-Modul ausgestattet ist, das vollständige Wi-Fi- und Bluetooth LE-Funktionen integriert.

www.elektor.de/20697

ESP32-Cam-CH340

Das ESP32-Cam-CH340-Entwicklungsboard kann in verschiedenen Internet-of-Things-Anwendungen eingesetzt werden, z. B. in intelligenten Heimgeräten, drahtloser Industriesteuerung, drahtloser Überwachung, drahtloser QR-Identifikation, drahtlosen Ortungssystemen und anderen Internet-of-Things-Anwendungen.

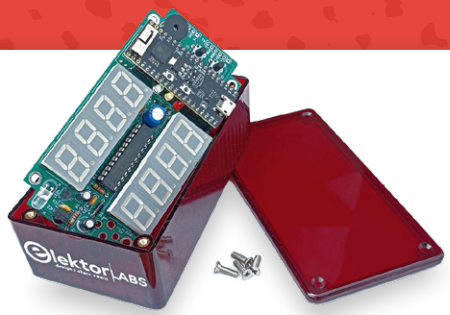
www.elektor.de/19333



ESP32-C3-DevKitM-1

ESP32-C3-DevKitM-1 ist ein Entwicklungsboard der Einstiegsklasse, das auf dem ESP32-C3-MINI-1 basiert, einem Modul, das aufgrund seiner geringen Größe benannt wurde. Dieses Board verfügt über vollständige Wi-Fi- und Bluetooth LE-Funktionen. Die meisten E/A-Pins des ESP32-C3-MINI-1-Moduls sind auf die Stiftleisten auf beiden Seiten des Boards aufgeteilt, um die Anbindung zu erleichtern. Entwickler können Peripheriegeräte entweder mit Jumper-Drähten anschließen oder ESP32-C3-DevKitM-1 auf einem Breadboard montieren.

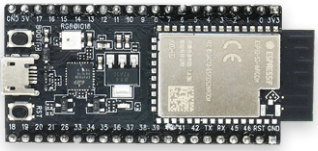
www.elektor.de/20324



Elektor Cloc 2.0 Kit

Cloc ist ein einfach zu bauender ESP32-basierter Wecker, der sich mit einem Zeitserver verbindet und Radio und TV steuert. Er hat ein doppeltes 7-Segment-Retro-Display mit variabler Helligkeit. Ein Display zeigt die aktuelle Zeit an, das andere die Alarmzeit.

www.elektor.de/20438



ESP32-S2-Saola-1M

ESP32-S2-Saola-1M ist ein kleines ESP32-S2-basiertes Entwicklungsboard. Die meisten E/A-Pins sind auf die Stiftleisten auf beiden Seiten aufgeteilt, um den Anschluss zu erleichtern. Entwickler können Peripheriegeräte entweder mit Jumperdrähten anschließen oder ESP32-S2-Saola-1M auf einem Breadboard montieren. ESP32-S2-Saola-1M ist mit dem ESP32-S2-WROOM-Modul ausgestattet, einem leistungsstarken, generischen Wi-Fi-MCU-Modul, das über eine Vielzahl von Peripheriegeräten verfügt.

www.elektor.de/19694

Arduino Nano ESP32

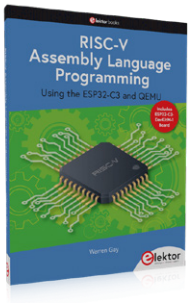
Das Arduino Nano ESP32 ist ein Nano-Formfaktor-Board, das auf dem ESP32-S3 (eingebettet im NORA-W106-10B von u-blox) basiert. Es ist das erste Arduino-Board, das vollständig auf einem ESP32 basiert. Es bietet Wi-Fi, Bluetooth LE, Debugging über natives USB in der Arduino-IDE sowie einen geringen Stromverbrauch.

www.elektor.de/20562

MakePython ESP32 Development Kit

Dieses MakePython ESP32 Kit ist ein unverzichtbares Entwicklungskit für die ESP32 MicroPython-Programmierung. Neben dem MakePython ESP32-Entwicklungsboard enthält dieses Kit die grundlegenden elektronischen Komponenten und Module, die Sie benötigen, um mit dem Programmieren zu beginnen. Mit den 46 Projekten im beiliegenden Buch können Sie einfache Elektronik-Projekte mit MicroPython auf ESP32-Basis erstellen und eigene IoT-Projekte einrichten.

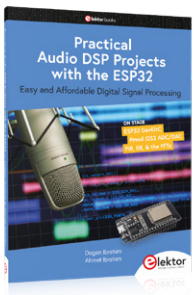
www.elektor.de/20452



RISC-V Assembly Language Programming using ESP32-C3 and QEMU (+ GRATIS ESP32 RISC-V Board)

Die Verfügbarkeit des Espressif ESP32-C3 Chips bietet eine Möglichkeit, praktische Erfahrungen mit RISC-V zu sammeln. Der quelloffene QEMU-Emulator bietet eine 64-Bit-Erfahrung mit RISC-V unter Linux. Dies sind nur zwei Möglichkeiten für Studenten und Enthusiasten gleichermaßen, RISC-V in diesem Buch zu erkunden. Die Projekte in diesem Buch sind auf das Nötigste reduziert, um die Assembler-Konzepte klar und einfach zu halten.

www.elektor.de/20296



Practical Audio DSP Projects with the ESP32

Das Ziel dieses Buches ist es, die Grundprinzipien der digitalen Signalverarbeitung (DSP) zu vermitteln und sie aus praktischer Sicht mit einem Minimum an Mathematik vorzustellen. Es werden nur die Grundlagen der Theorie zeitdiskreter Systeme vermittelt, die ausreichen, um DSP-Anwendungen in Echtzeit zu implementieren. Die praktischen Implementierungen werden in Echtzeit unter Verwendung des sehr beliebten ESP32 DevKitC Mikrocontroller-Entwicklungsboards beschrieben.

www.elektor.de/20558

MicroPython für Mikrocontroller

Leistungsfähige Controller wie der ESP32 der Firma Espressif Systems bieten eine hervorragende Performance sowie Wi-Fi- und Bluetooth-Funktionalität zu einem günstigen Preis. Mit diesen Eigenschaften wurde die Maker-Szene im Sturm erobert. Im Vergleich zu anderen Controllern weist der ESP32 einen deutlich größeren Flash und SRAM-Speicher, sowie eine wesentlich höhere CPU-Geschwindigkeit auf. Aufgrund dieser Leistungsmerkmale eignet sich der Chip nicht nur für klassische C-Anwendungen, sondern insbesondere auch für die Programmierung mit MicroPython. Dieses Buch führt in die Anwendung der modernen Ein-Chip-Systeme ein.

www.elektor.de/19412