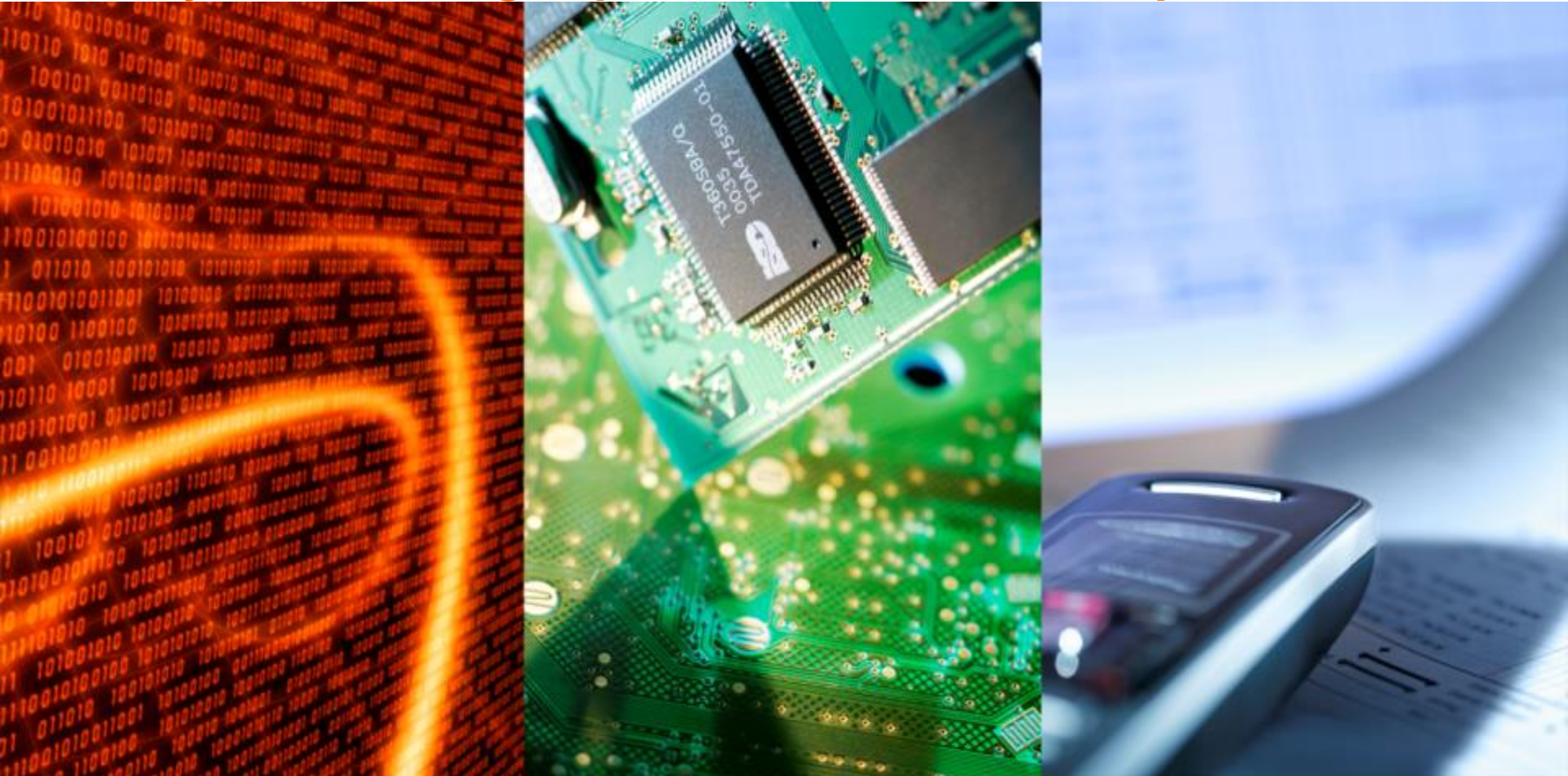


Projekterfahrung Espressif ESP32 – Fallbeispiel "Kevin"



Virtueller Mitbewohner „Kevin“

Kevin simuliert die Anwesenheit von Hausbewohnern durch die Wiedergabe von Geräuschen und Lichteffekten.

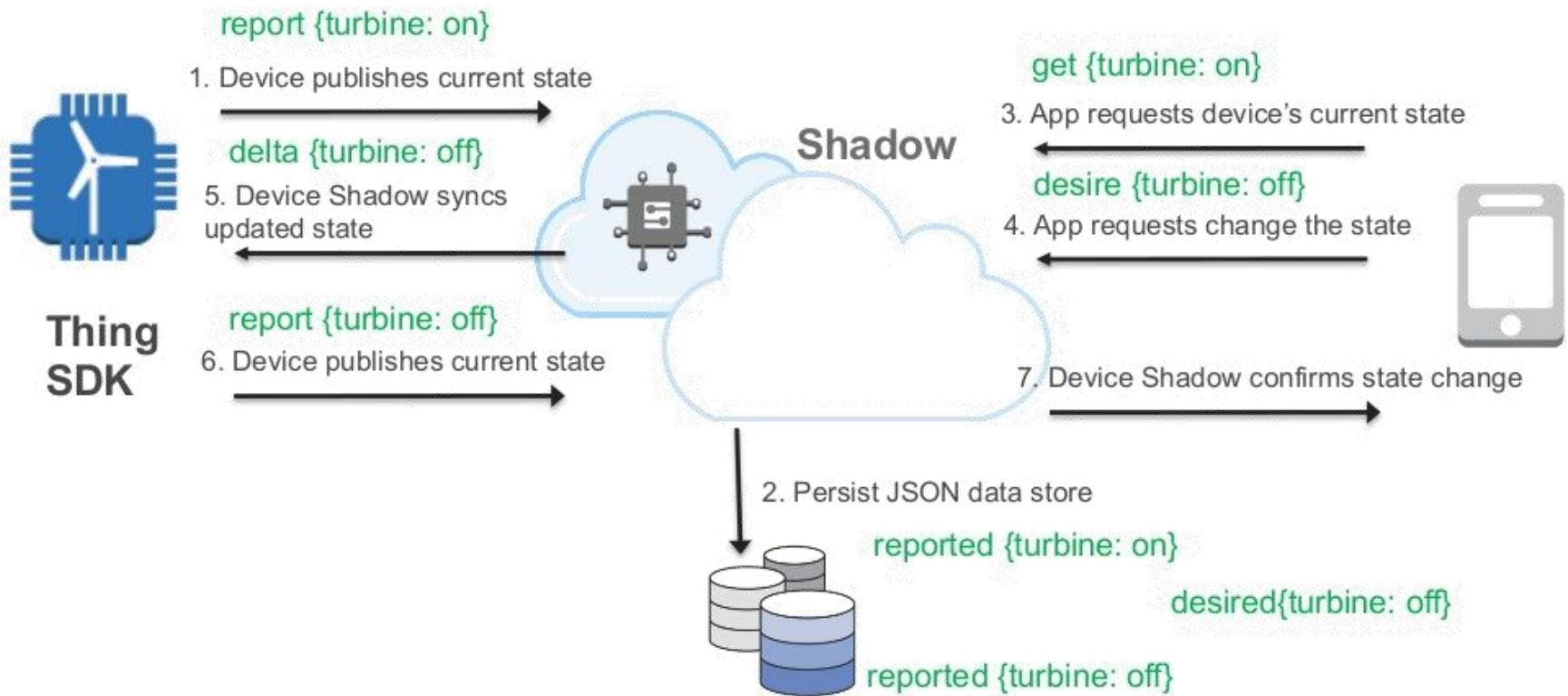


© Mitipi AG, Zürich

Projektpartner Mitipi

- WLAN für Konfiguration via AWS Cloud
- BLE für Inbetriebnahme
- Audio Player
- LEDs für Fernseher-Simulation und Statusanzeige
- Bedienknöpfe
- Verschlüsselung
- Datenspeicher für Audio und LED-Sequenzen

Device Shadow Service for AWS IoT



Welche HW-Plattform eignet sich am Besten?

Programmierbare Funkmodul
z.B. Murata Type 1LD

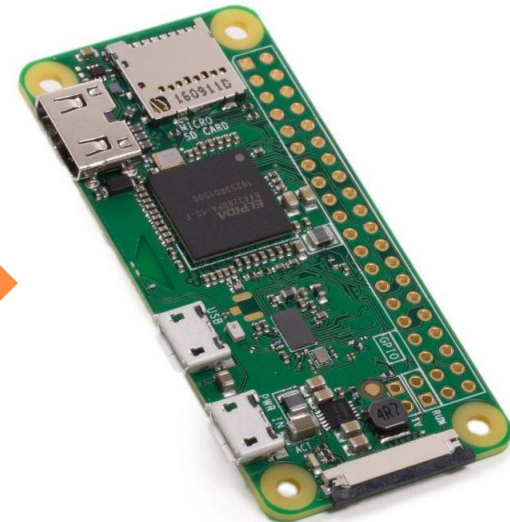


Single Packaging



Hardware/Software turnkey solution significantly reduce development cost!

Linux-basierter SBC
z.B. Raspberry Pi Zero W

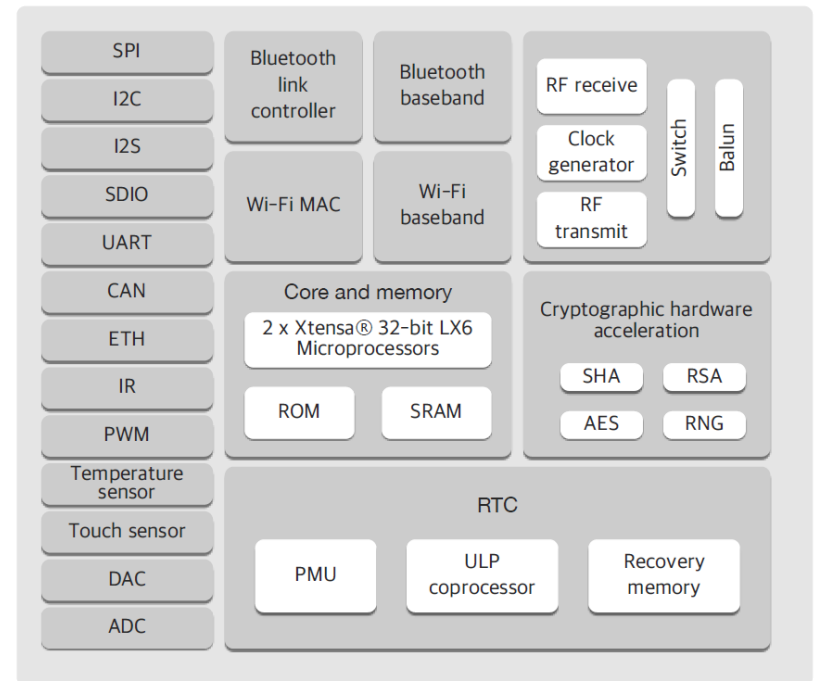


- Proprietäres SDK
- Meist länger Verfügbar
- Preis > \$15

- Standard POSIX-API
- Manchmal nur kurz Verfügbar
- Preis > \$10

ESP32 Chip

- Xtensa dual-core CPU (240 MHz)
- 448 KiB ROM
- 520 KiB SRAM (+16 KiB RTC SRAM)
- WiFi 802.11 b/g/n (2.4 GHz)
- Bluetooth 4.2 mit BLE
- QSPI für externes Flash und RAM (mit AES-HW für Verschlüsselung)
- AES/SHA/RSA HW-Beschleuniger
- SD-Card / SDIO
- Timer / RTC / Watchdog
- UART / I2C / I2S / SPI
- 12 bit ADC / 8 bit DAC
- 32 GPIO



Integrierter Speicher

- Das ROM beinhaltet den ROM-Loader und Low Level SW für den Funkteil
- Das SRAM ist aufgeteilt in Data RAM (DRAM) und Instruction RAM (IRAM)
- Das IRAM (192 KiB) wird gebraucht für:
 - Cache (64 KiB)
 - ISRs und System Level SW (128 KiB)
- Nur das DRAM (328 KiB) steht für Variablen zur Verfügung
- Das RTC SRAM liegt in einer separaten Power Domain, wird somit auch noch mit Strom versorgt, wenn der Rest des ESP32 ausgeschaltet ist.

ESP32 Module



- ESP32 (dual-core)
- CE- und FCC-Zulassung
- Integrierte Antenne
- 4 MiB QSPI Flash
- \$3.50 Einzelstückpreis



- ESP32 (dual-core)
- CE- und FCC-Zulassung
- Integrierte Antenne
- 4 MiB QSPI Flash
- 4 MiB QSPI RAM
- \$4.30 Einzelstückpreis

Espressif IoT Development Framework

Die Basis-Software wird auf GitHub bereitgestellt

ESP-IDF is the official development framework for the [ESP32](#) chip.

Developing With the ESP-IDF

Setting Up ESP-IDF

See setup guides for detailed instructions to set up the ESP-IDF:

- [Windows Setup Guide](#)
- [Mac OS Setup Guide](#)
- [Linux Setup Guide](#)

Finding a Project

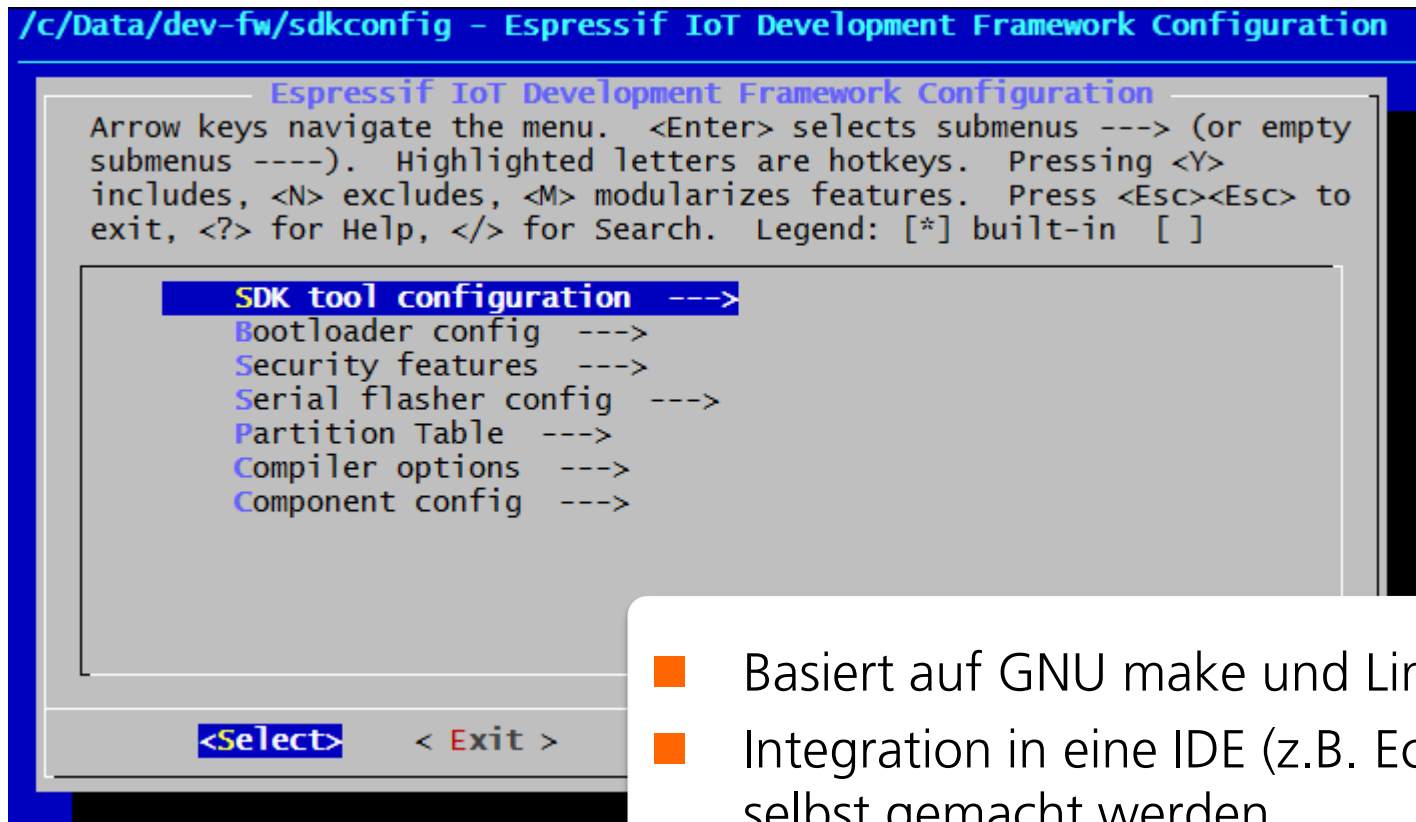
As well as the [esp-idf-template](#) project mentioned in the setup guide, ESP-IDF comes with some example projects in the [examples](#) directory.

Enthält:

- Build System
- GCC Cross Tool Chain
- Programming Tool
- SW-Komponenten
- Code-Beispiele

Build System

```
/c/Data/dev-fw/sdkconfig - Espressif IoT Development Framework Configuration
```



```
Espressif IoT Development Framework Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

  SDK tool configuration --->
  Bootloader config --->
  Security features --->
  Serial flasher config --->
  Partition Table --->
  Compiler options --->
  Component config --->

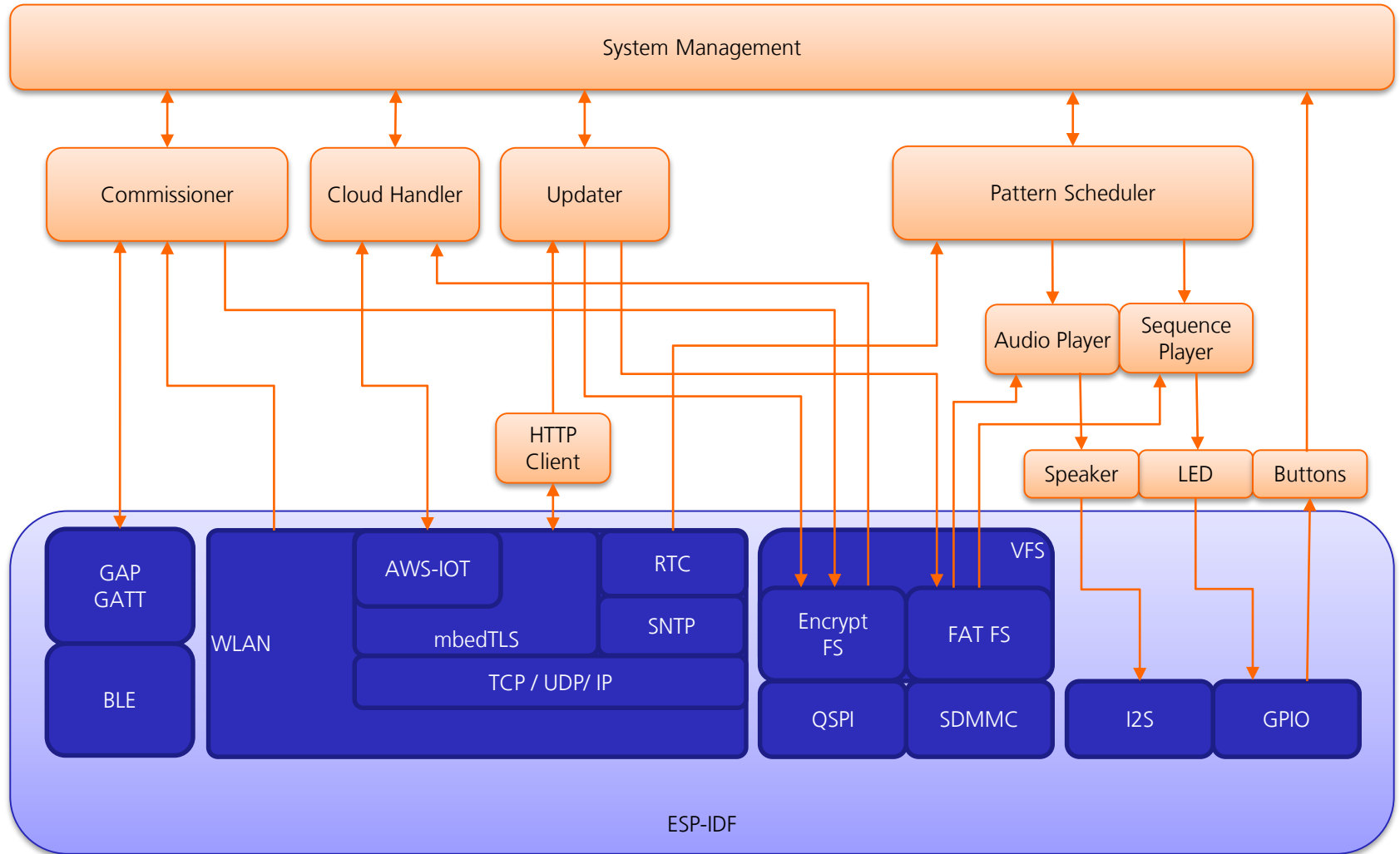
<Select> <Exit >
```

- Basiert auf GNU make und Linux kconfig
- Integration in eine IDE (z.B. Eclipse) muss selbst gemacht werden
- Debugging ist via JTAG-Interface und GDB/OpenOCD möglich

SW-Komponenten

- WiFi Subsystem (closed source, nur als Binärbibliothek)
- Bluetooth Subsystem (Physical Layer ist closed source)
- lwIP TCP/IP Subsystem
- mbedTLS SSL/TLS Subsystem
- FreeRTOS (adaptiert für Dual-Core)
- AWS-IOT Device SDK von Amazon
- (Secure) Bootloader
- Over-the-Air Firmware Update (OTA)
- SPIFFS
- FatFs
- Wear Levelling für QSPI-Flash (Flash translation layer, z.B. für FatFs)
- Treiber für die diversen Peripherien

SW-Architektur



QSPI-Flash-Partition

#	Name,	Type,	SubType,	Offset,	Size,	Flags
#	boot,	boot,	,	0,	0x9000,	encrypted
	nvs,	data,	nvs,	0x9000,	0x5000	
	otadata,	data,	ota,	,	0x2000,	encrypted
	ota0,	app,	ota_0,	0x10000,	1920K,	encrypted
	ota1,	app,	ota_1,	,	1920K,	encrypted
	intfs,	data,	fat,	,	128K,	encrypted
	coredump,	data,	coredump,	,	64K	

- Boot-Partition enthält (Secure) Bootloader
- NVS-Partition enthält persistente Konfigurationsdaten
- OTA0/1-Partitionen enthalten den Anwendungscode (aus der einen Partition läuft die Anwendung, in die andere Partition schreibt der OTA-Update die aktualisierte Anwendung)
- INTFS enthält eine verschlüsselte FAT-Partition

Probleme

- ESP-IDF ist noch nicht ausgereift
- AWS-IOT Device SDK ist eine Katastrophe
- mbedTLS ist Speicherhungrig
- Speicherverbrauch generell

Ausgereiftheit

- ESP-IDF ist nach wie vor in Entwicklung, obwohl bereits V3.0 released
- Die APIs sind noch nicht stabil, d.h. der Wechsel auf die neuste ESP-IDF kann Anpassungen an der Anwendung nach sich ziehen
- Bluedroid war anfänglich unvollständig und fehlerhaft, ist aber unterdessen brauchbar
- Fehler im I2S-Treiber haben wir mit eigenem Hack geflickt
- Momentan funktioniert das Verschlüsseln einer FatFs-Partition im QSPI-Flash noch nicht

AWS-IOT Device SDK

- Nach Amazon soll das «Embedded C SDK» besonders Ressourcen schonend sein, braucht aber über 40 KiB RAM
- Es ist schlecht designed (verwendet viele statisch allozierte globale Variablen)
- Wir mussten die Implementation anpassen und grosse statisch allozierte Felder auf den Heap legen
- Das API ist Callback-basiert aber trotzdem blockierend, unabhängiges Senden und Empfangen ist nicht möglich
- Amazon bietet auch ein «C++ SDK» an das noch mehr Ressourcen braucht! Wir haben es nicht ausprobiert.

mbedTLS

- mbedTLS alloziert für jede Verbindung zwei 16 KiB Frame-Buffer für die Sende- und die Empfangsrichtung
- Während dem Verbindungsaufbau wird zusätzlich ca. 30 KiB RAM gebraucht für die Authentifizierung und Schlüsselaushandlung
- Unsere Anwendung muss min. zwei SSL-Verbindungen parallel handhaben können (zur AWS-Cloud und für den OTA-Update zu einem HTTPS-Server)
- Das ergibt 94 KiB Heap-Speicher nur für die SSL-Verbindungen

Speicherverbrauch Debug-Konfiguration

```

DRAM .data size: 16760 bytes
DRAM .bss size: 71320 bytes
Used static DRAM: 88080 bytes ( 174064 available, 33.6% used)
Used static IRAM: 122468 bytes ( 8604 available, 93.4% used)
    Flash code: 836042 bytes
    Flash rodata: 191516 bytes
Total image size:~1166786 bytes (.bin may be padded larger)

```

	DRAM	IRAM	Flash	Total
WiFi:	21584	29950	177533	229067
Bluedroid:	18900	12107	175217	206224
StdLibrary:	2370	16478	123117	141965
mbedTLS:	356	0	133502	133858
lwIP:	4221	0	113177	117398
System:	6087	29794	81321	117202
Driver:	712	15221	88976	104909
Application:	14996	0	82191	97187
AWS-IOT:	13470	0	18943	32413
FreeRTOS:	4924	17668	2164	24756
FileSystem:	154	0	23820	23974

Speicherverbrauch Release-Konfiguration

```

DRAM .data size:    13568 bytes
DRAM .bss  size:    70672 bytes
Used static DRAM:   84240 bytes ( 177904 available, 32.1% used)
Used static IRAM:  103396 bytes (  27676 available, 78.9% used)
    Flash code:    686310 bytes
    Flash rodata:   77492 bytes
Total image size:~ 880766 bytes (.bin may be padded larger)

```

	DRAM	IRAM	Flash	Total
WiFi:	21587	29966	176088	227641
Bluedroid:	18900	12066	146111	177077
StdLibrary:	1389	13559	116338	131286
mbedTLS:	356	0	122878	123234
lwIP:	4221	0	79161	83382
Application:	14434	0	33903	48337
System:	3878	18982	23287	46147
Driver:	660	13445	28727	42832
AWS-IOT:	13467	0	12733	26200
FreeRTOS:	4924	14210	1666	20800
FileSystem:	154	0	20132	20286

Fazit

- Der ESP32 ist ein interessanter Chip mit vielen Möglichkeiten
- Er ist preislich sehr attraktiv
- Support findet man im Forum und auf GitHub
- Entwicklung des ESP-IDF immer noch stark im Fluss
- Ziemlich Ressourcen-hungrig da ESP-IDF viele SW-Komponenten aus dem Linux-Ökosystem verwendet
- AWS-IOT-Komponente sollte ersetzt werden (vielleicht Mongoose OS?)

Links

- Mitipi
<https://www.mitipi.com>
- Espressif ESP32
<https://www.espressif.com/en/products/hardware/esp32/overview>
- GitHub ESP-IDF
<https://github.com/espressif/esp-idf>
- ESP-IDF Dokumentation
<http://esp-idf.readthedocs.io/en/latest>
- AWS-IOT
<https://aws.amazon.com/iot>
- mbedTLS
<https://tls.mbed.org/>

Wir sind Ihre Lösung.

Arendi AG
Eichtalstrasse 55
8634 Hombrechtikon
Schweiz

www.arendi.ch