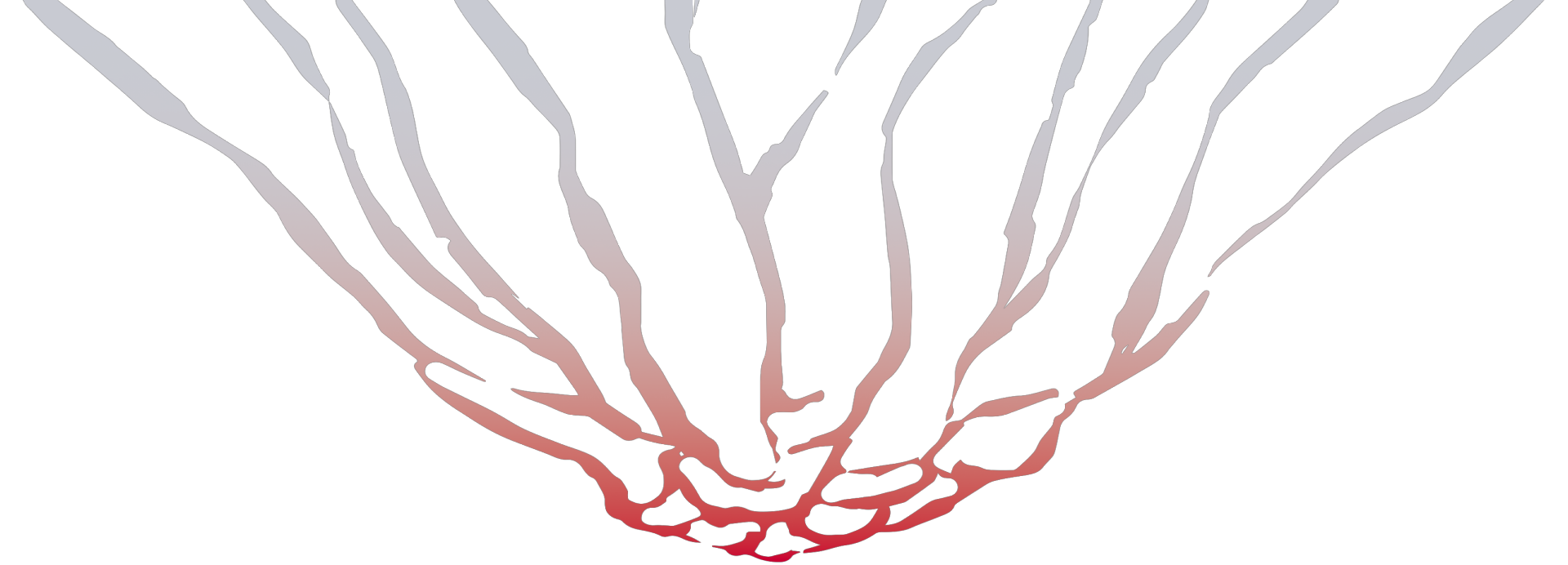


ESPRESSIF

TRAINING

Going In-Depth

Objective

- Hardware Subsystems
- Software Modules

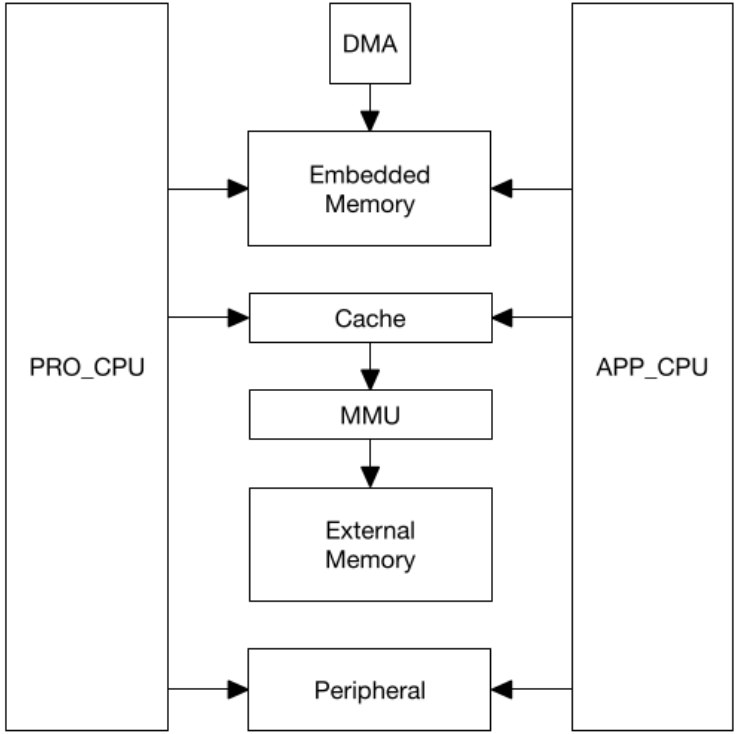
Memory Architecture

Memory Overview

- Memory Subsystem
- Address Mapping
- Embedded Memory
- External Memory
- Memory Speed

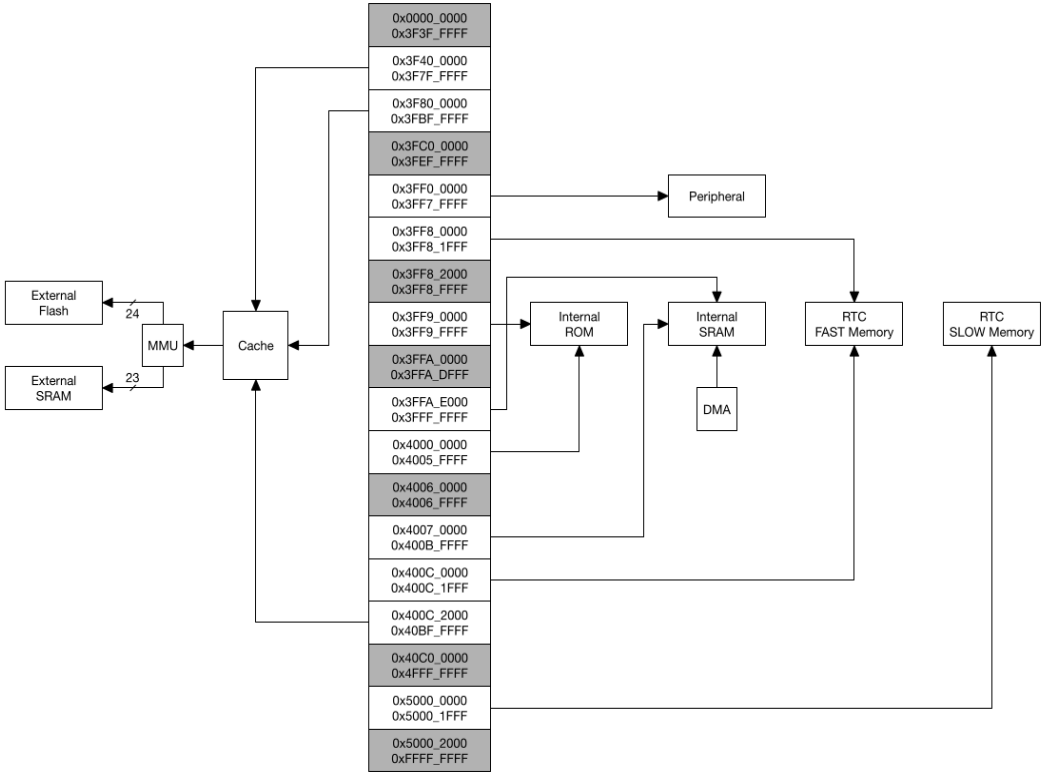
Memory Subsystem

- Symmetric address mapping for both cores
- 4GB address space for both instruction and data bus
- Word-aligned access *only* using instruction bus
- Unaligned access possible using data bus



Address Mapping

Instruction Bus	0x4000_0000 ~ 0x4FFF_FFFF
Data Bus	0x0 ~ 0x4000_0000
Shared I/D	0x5000_0000 ~ 0xFFFF_FFFF



Memory Architecture: Embedded Memory

Embedded Memory

- Internal 448K ROM
  - 1st stage bootloader and ROM libs
- Internal 520K SRAM
  - 192K Instruction SRAM
    - 32K cache memory for *each* core
    - 128K code memory
    - Only 32-bit aligned accesses
  - 328K Data SRAM
    - DMA capable
    - Fastest access

Embedded Memory

- RTC Memory
  - RTC fast memory 8K
    - Only PRO CPU accessible
    - Operates on APB clock (80MHz)
    - Retention purpose
  - RTC slow memory 8K
    - Both CPU can access
    - Operates on fast clock (8MHz)
    - ULP execution and retention purpose

Embedded Memory

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3FF8_0000	0x3FF8_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
	0x3FF8_2000	0x3FF8_FFFF	56 KB	Reserved	-
Data	0x3FF9_0000	0x3FF9_FFFF	64 KB	Internal ROM 1	-
	0x3FFA_0000	0x3FFA_DFFF	56 KB	Reserved	-
Data	0x3FFA_E000	0x3FFD_FFFF	200 KB	Internal SRAM 2	DMA
Data	0x3FFE_0000	0x3FFF_FFFF	128 KB	Internal SRAM 1	DMA
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Instruction	0x4000_0000	0x4000_7FFF	32 KB	Internal ROM 0	Remap
Instruction	0x4000_8000	0x4005_FFFF	352 KB	Internal ROM 0	-
	0x4006_0000	0x4006_FFFF	64 KB	Reserved	-
Instruction	0x4007_0000	0x4007_FFFF	64 KB	Internal SRAM 0	Cache
Instruction	0x4008_0000	0x4009_FFFF	128 KB	Internal SRAM 0	-
Instruction	0x400A_0000	0x400A_FFFF	64 KB	Internal SRAM 1	-
Instruction	0x400B_0000	0x400B_7FFF	32 KB	Internal SRAM 1	Remap
Instruction	0x400B_8000	0x400B_FFFF	32 KB	Internal SRAM 1	-
Instruction	0x400C_0000	0x400C_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data Instruction	0x5000_0000	0x5000_1FFF	8 KB	RTC SLOW Memory	-

Memory Architecture: External Memory

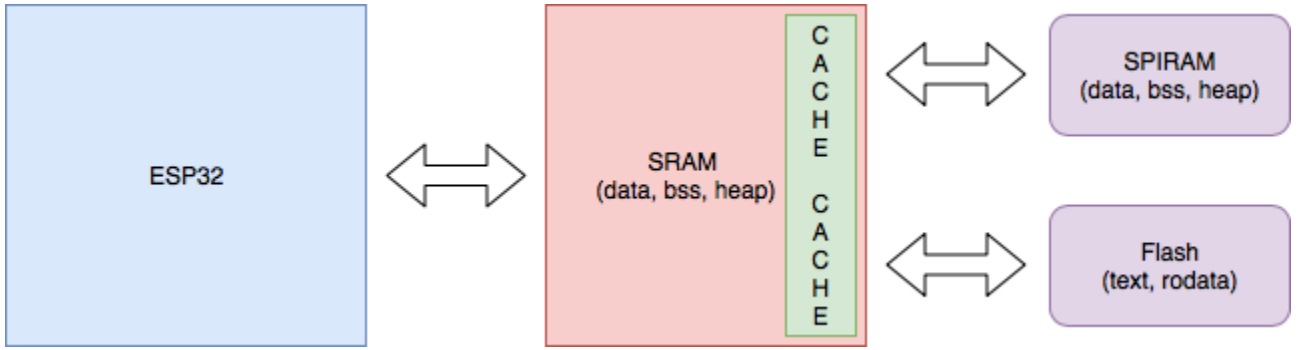
External Memory

- Flash
  - Supports up-to 16M off-chip SPI flash
  - Allows eXecute-In-Place (XIP)
  - Accessed using 32K internal cache
- SPIRAM
  - Supports up-to 8M off-chip SPI SRAM
  - Can be 10x slower compared with internal memory when accessed using cache
  - No DMA capability

External Memory

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3F40_0000	0x3F7F_FFFF	4 MB	External Flash	Read
Data	0x3F80_0000	0x3FBF_FFFF	4 MB	External SRAM	Read and Write
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Instruction	0x400C_2000	0x40BF_FFFF	11512 KB	External Flash	Read

Simplified Memory Model



Memory Speed

- Embedded Memory
  - ROM/SRAM clocked at CPU frequency, single cycle access possible
  - RTC fast memory clocked at APB clock (80MHz)
  - RTC slow memory from fast clock (8MHz)
  - Upto 400 Mbyte/s for memcpy operation
- DMA uses APB clock to access memory
- External Flash/SPIRAM at 40/80 MHz
  - Dual/Quad mode of operation for flash
  - Upto 32 Mbyte/s for memcpy operation for SPIRAM

Security Architecture

Security Overview

- eFUSE/OTP
- Secure Boot
- Flash Encryption
- Considerations

Security Architecture : eFUSE/OTP

eFUSE/OTP

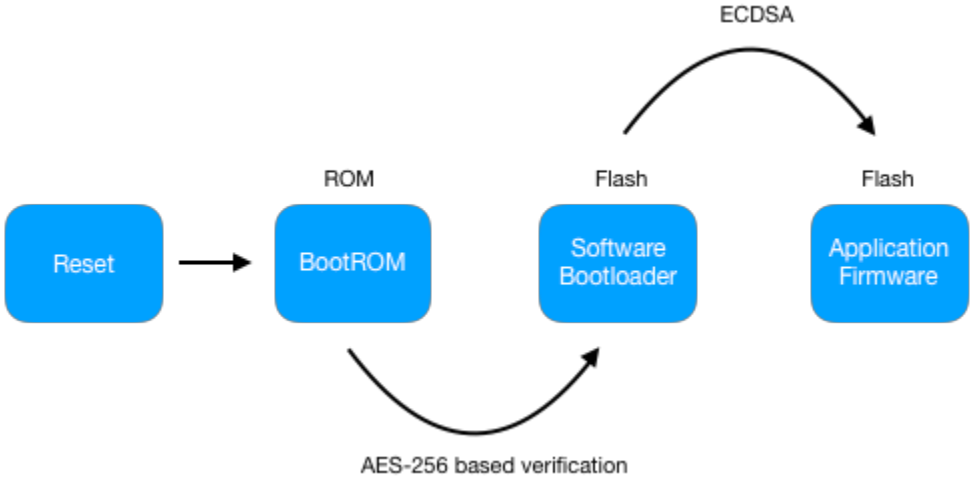
- The eFUSE can be used to program certain information into the SoC
- By default all bits of the eFUSE are 1. Once a bit is flipped to 0, it cannot go back to 1.
- Only software running on the ESP32 can program the eFUSE
- Parts of the eFUSE can even be locked from read-out or writes by the software

eFUSE Structure

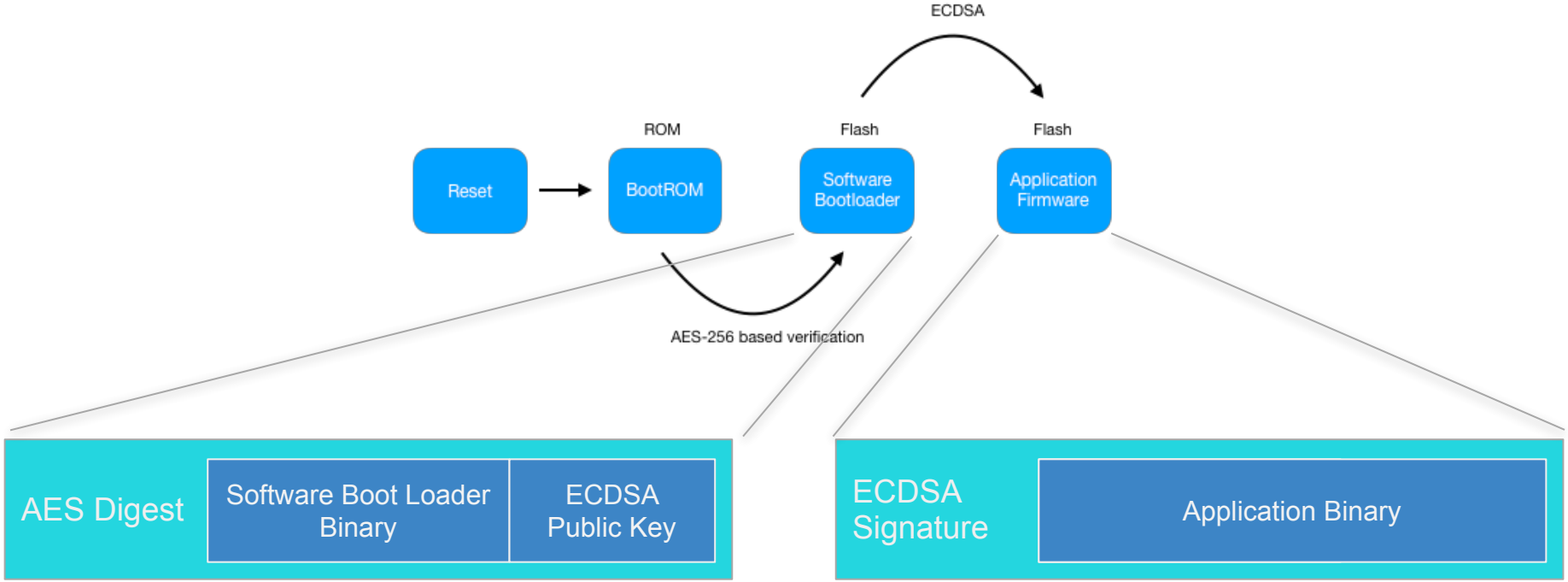
- Application Use: Some part of Block 3 can be used by Applications for their own use
  - Store data that can be read by software, but not by directly reading the flash externally

Security Architecture : Secure Boot

Secure Boot Flow



Secure Boot Flow

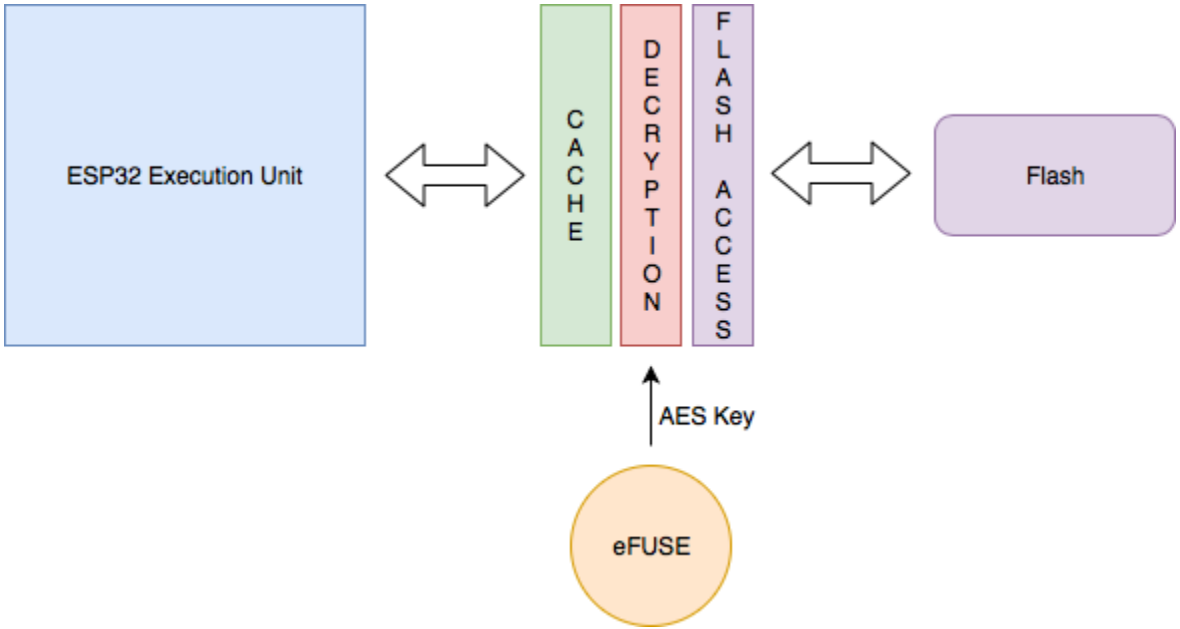


Security Architecture : Flash Encryption

Flash Encryption

- ESP32 typically executes code directly from flash
- ESP32 can be configured such that any data read from flash can be transparently decrypted before reaching the MCU
- The cache controller hardware can be configured to read the decryption key from the eFUSE
- The eFUSE can be protected from software readout

Flash Encryption



Security Architecture : UART Download

UART Download

- Strap pins can trigger UART download mode of ESP32 to be triggered
- This mode can be triggered even when Secure Boot and Flash Encryption are enabled
- Can execute code on MCU, but cannot decrypt flash, or tamper with the firmware

Resources

https://docs.espressif.com/projects/esp-idf/en/latest/security/secure-boot.html

https://docs.espressif.com/projects/esp-idf/en/latest/security/flash-encryption.html

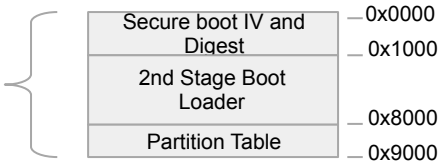
Note:

- Once eFUSE is programmed, the changes cannot be undone.
- Incorrect configuration may lead to an unusable SoC.

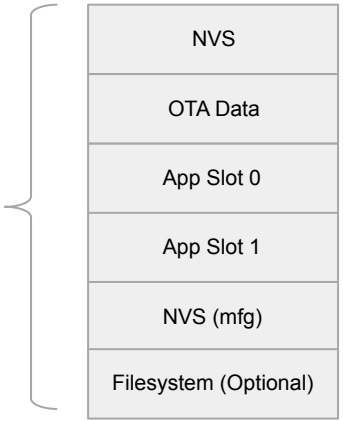
Firmware Anatomy and Boot-up

Flash Layout

Fixed



User Defined (Partition Table)

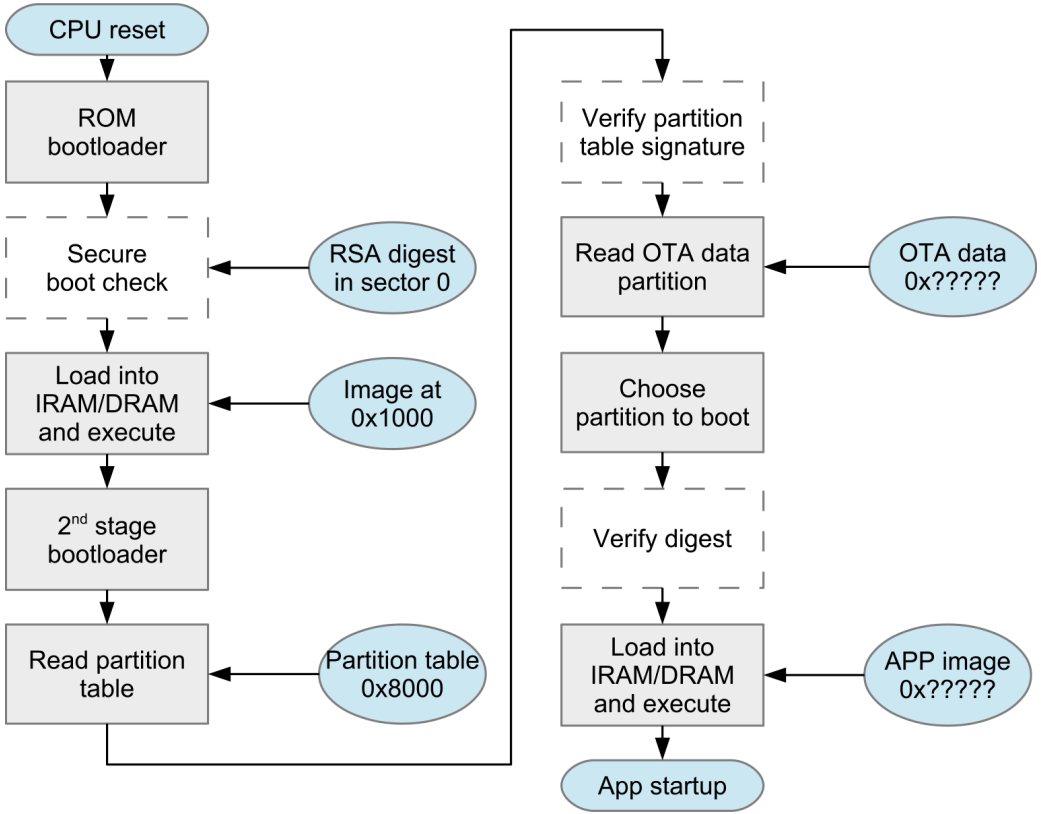


Name	Type	Sub-type	Offset	Size	Flags
nvs	data	nvs	0x9000	0x4000	
otadata	data	ota	0xd000	0x2000	
ota_1	app	ota_1	0x10000	0x180000	
ota_2	app	ota_2	0x19000	0x180000	
mfg	data	nvs	0x31000	0x4000	

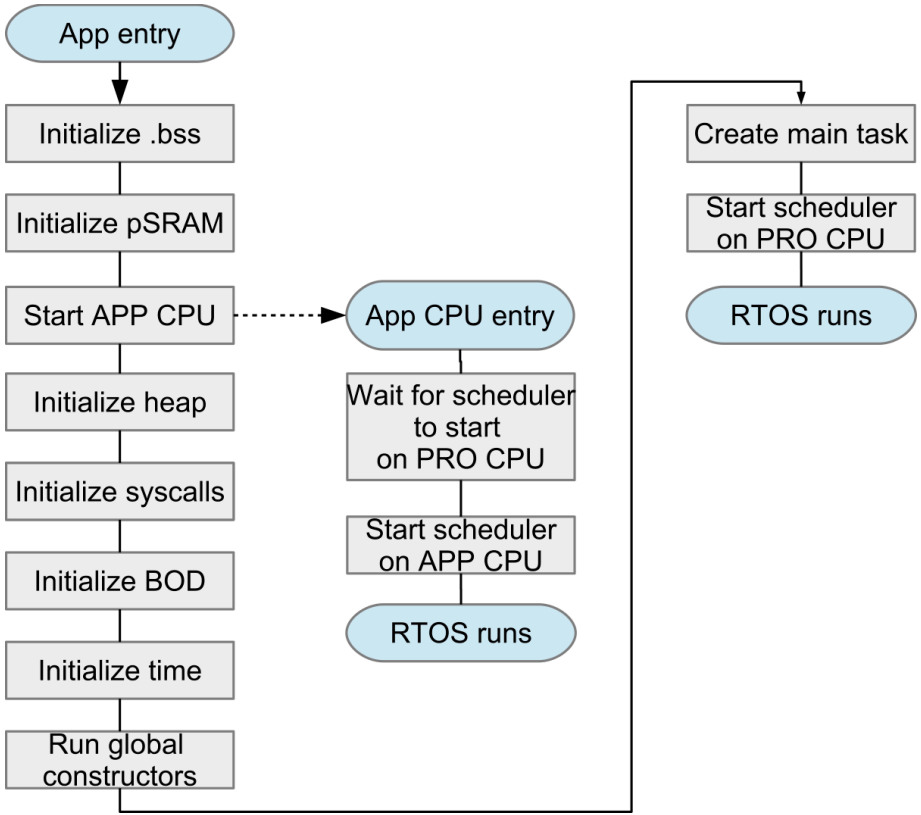
Flash Components

- NVS
  - Non-volatile storage for configuration data
- OTA Data
  - Used internally by the OTA module to identify the latest active partition
- App Slot 0 and 1
  - Active-Passive partition for OTA updates
- NVS (Mfg) (Optional)
  - NVS partition for manufacturing data

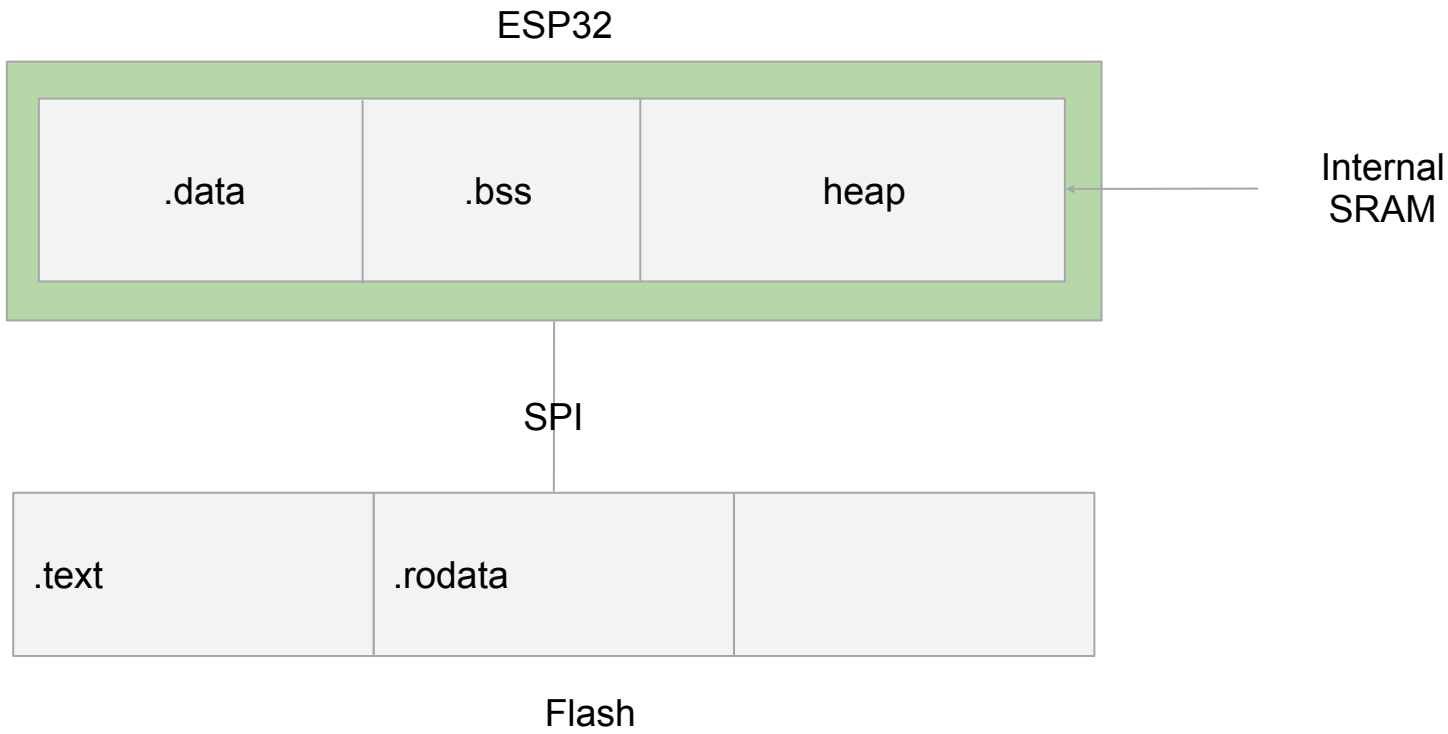
Bootloader Execution Flow



Application Startup Flow



Memory Layout during execution

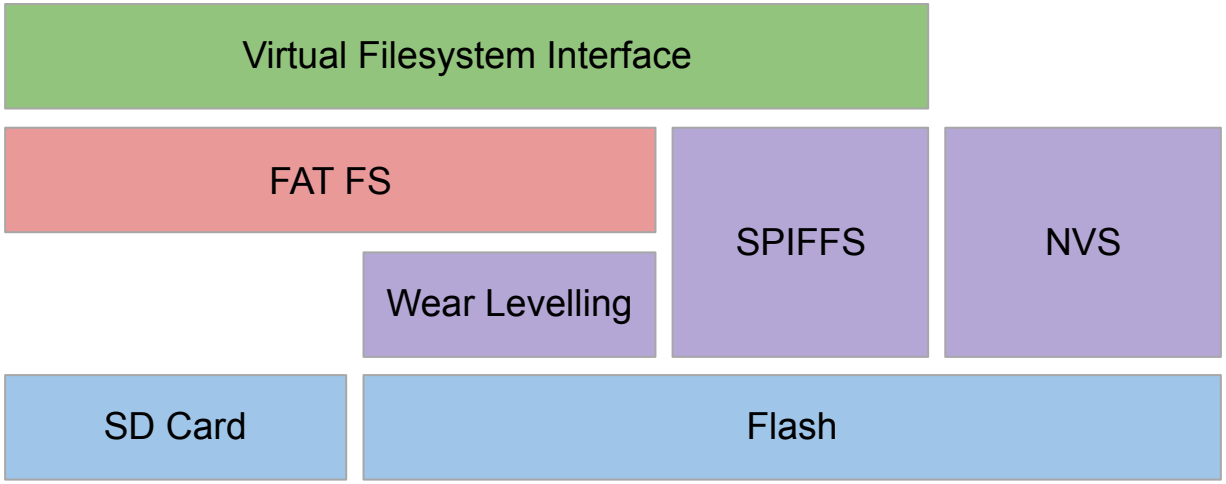


Storage

Storage Overview

- NVS
- Virtual File System Interface
- FAT
- SPIFFS
- SD Card
- Wear Levelling

Storage Options



NVS

- Key-Value store in flash
- Power loss resilient
- Wear-levelling: Log based structure
- Support namespaces
- Used for: Manufacturing settings, User’s configuration, Maintaining state across resets

FATFS

- ChaN’s FATFS library R0.12b (to be updated to R. 013)
  - http://elm-chan.org/fsw/ff/00index_e.html
- Works on SD Card or Wear-Levelling Driver

SPIFFS

- SPIFFS is a popular filesystem used in many ESP8266 based projects
  - https://github.com/pellepl/spiffs
- Flat filesystem
- Not compatible with flash encryption
- Supports filesystem image sizes from 64KB upwards

Networking

Networking Overview

- Application Protocols:
  - HTTP (Server, Client),
  - HTTP2 Client,
  - MQTT Client
  - mDNS/DNS-SD Bonjour
  - COAP (Server, Client)
- TLS (Transport Layer Security)
- LWIP
- Wi-Fi
- Bluetooth
- Ethernet

LWIP

Light Weight IP (http://www.nongnu.org/lwip/2_1_x/index.html)

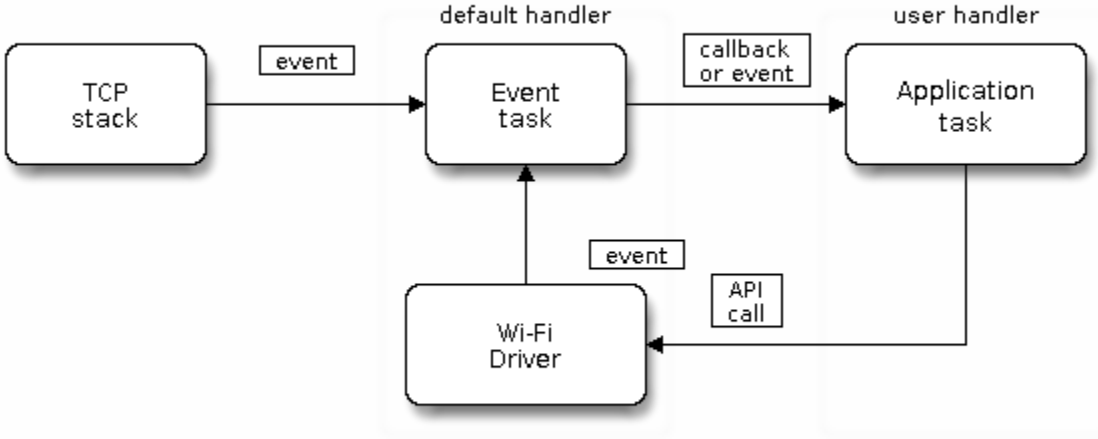
- Modified with ESP’s patches
- TCP/UDP, IPv4/v6, DHCP, ICMP, IGMP
- BSD Socket API

Networking: Wi-Fi

Wi-Fi Features

- 802.11b/g/n
- Modes: Station, SoftAP, Promiscuous, Simultaneous (STA & SoftAP)
- Security: WPA/WPA2/WPA2-Enterprise, WPS
- 802.11-compliant power management
- Adaptive rate fallback algorithm
- Antenna diversity
- Up to 20 Mbps TCP and 30Mbps UDP OTA throughput

Wi-Fi Programming Model



An application can register its own event callback function.

Typical Wi-Fi Programming Sequence

- Initialize Wi-Fi stack using *esp_wifi_init* API
  - Allocates internal structures, buffers
  - Registers an event handler and starts wifi task
- Set mode using *esp_wifi_set_mode* API
  - STA, AP, STA + AP
- Set mode specific configuration using *esp_wifi_set_config* API
  - SSID, Password, Beacon Interval, Channel etc
- Start WiFi using *esp_wifi_start* API as per previous config
- Perform mode specific operations and handle events
  - For example, start STA scan using *esp_wifi_scan_start* and handle *SYSTEM_EVENT_SCAN_DONE*
- Stop and deinit wifi using *esp_wifi_stop* & *esp_wifi_deinit* APIs

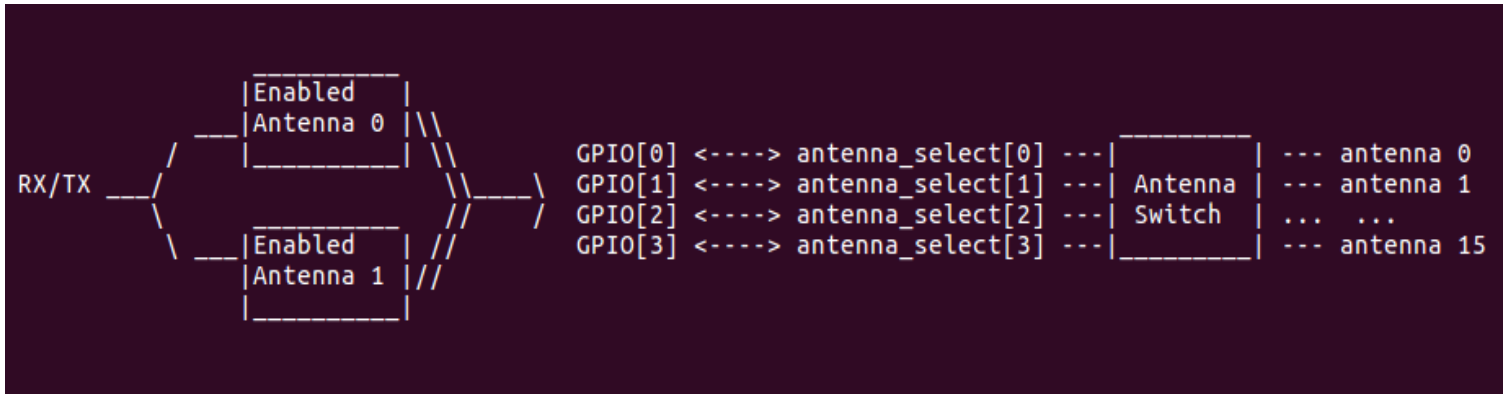
Wi-Fi Protocol Modes

- Call *esp_wifi_set_protocol* to set any of the following Wi-Fi protocol modes
  - 802.11 B
  - 802.11 BG
  - 802.11 BGN
  - 802.11 BGNLR
  - 802.11 LR
- 802.11 LR
  - This mode is an Espressif-patented mode which can achieve a one-kilometer line of sight range.

Wi-Fi Channel & Tx Power Management

- Call *esp_wifi_set_country* to set the country code which limits the channel range. The following policies can be configured.
  - *WIFI_COUNTRY_POLICY_AUTO*
    - When connected, the country info of the AP to which the station is connected is used.
    - When not connected, the configured country info is used.
  - *WIFI_COUNTRY_POLICY_MANUAL*
    - Configured country info is used regardless of the connection state
- Max Tx power can be configured by
  - *"menuconfig: component => PHY => Max Wi-Fi TX Power"* to set the default
  - Call *esp_wifi_set_max_tx_power* in the application if the default value is not desirable.
- Note - The APIs do not validate the per-country rules. It is up to the user to fill in all fields according to local regulations.

Wi-Fi Multiple Antennas



- Up to 16 antennas through external antenna switch controlled by 4 address pins
  - antenna_select[0:3].
- Use **esp_wifi_set_ant_gpio** to configure which GPIOs are connected to antenna_selects.
- Only one or two antennas can be simultaneously enabled for RX/TX.
- Use **esp_wifi_set_ant**
  - To configure which antennas are enabled.
  - To configure antenna selection mode (which enabled antenna is used for Tx and Rx)
    - WIFI_ANT_MODE_ANT0, WIFI_ANT_MODE_ANT1, WIFI_ANT_MODE_AUTO

Wi-Fi Power Saving Modes

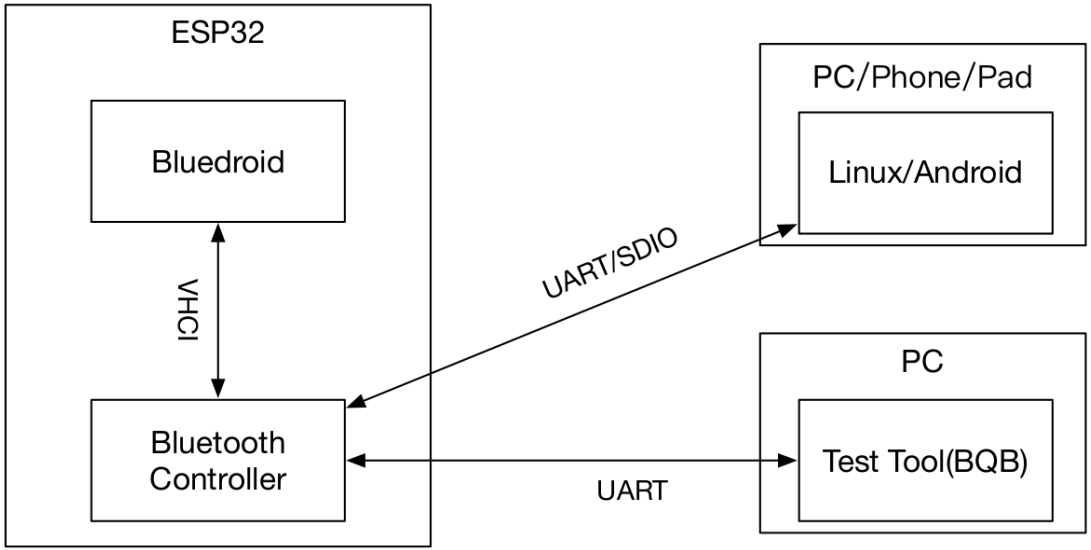
- Modem-sleep mode
  - Refers to the legacy power-saving mode in the IEEE 802.11 protocol
  - Works in STA only mode
  - When connected to AP STA switches between ACTIVE and SLEEP states
  - In SLEEP state, RF, PHY and BB are turned off in order to reduce power consumption
  - Connection is maintained
  - Current reduces from average ~120mA to ~40mA

- Types of Modem-sleep (chosen using *esp_wifi_set_ps*)
  - WIFI_PS_MIN_MODEM
    - STA wakes up every DTIM
  - WIFI_PS_MAX_MODEM
    - STA wakes up every LISTEN_INTERVEL configured by *esp_wifi_set_config*

Networking: Bluetooth

Overview

- ESP32 supports dual-mode Bluetooth
- Bluetooth Host and Controller Architecture



Architecture

- ESP IDF uses significantly modified Bluedroid as the Bluetooth Host (Classic BT + BLE)
- Layers
  - BTU - Bluetooth Adaptation Layer
  - BTC - Bluetooth Control Layer
  - HCI - Host Controller Interface
- Design Principles
  - Minimize the load on user tasks
  - Streamline the structure by handling over Bluetooth related tasks to the BTC layer

Classic Bluetooth

- To exchange large data over a small range
- Applications
  - Wireless Headsets and Speakers
  - Wireless Keyboards and Printers
  - Files transfer
- Supported Profiles
  - GAP, A2DP, AVRCP (CT), SPP, HFP Client

Bluetooth Low Energy

- Smaller and highly optimized version of classic Bluetooth
- Also known as Bluetooth Smart
- Applications
  - Gateways and Home Appliances
  - Wi-Fi network configuration - Blufi
- Supported Profiles
  - GAP, GATT, HID (Device), SPP-like, iBeacon and Eddystone

Security in Bluetooth

- Legacy Pairing (v2.0 devices and before)
  - Static Pin Code
- Secure Simple Pairing (v2.1 devices)
  - Just Works
  - Passkey Entry
  - Numeric Comparison

Cont'd...

- LE Legacy Pairing (v4.0, 4.1 and 4.2 devices)
  - Just Works
  - Passkey Entry

- LE Secure Connections (v4.2 devices)
  - Just Works
  - Passkey Entry
  - Numeric Comparison

Bluetooth Mesh

- Enables many-to-many device communications
- Operates on BLE and is compatible with core specification version 4.0 or higher
- Supported Features
  - Node and Provisioner Roles
  - PB-ADV and PB-GATT Bearers
  - Relay and Proxy support
  - Foundation Models
    - Configuration Server and Client Models
    - Health Server and Client Models

Cont'd...

- Client Models
  - Generic Client Models
  - Light Client Models
  - Sensor Client Models
  - Time and Scenes Client Models
- Applications
  - Commercial Lighting
  - Sensor Network Solutions
  - Generic IoT solutions with thousands of devices communicating with each other

Application Protocols

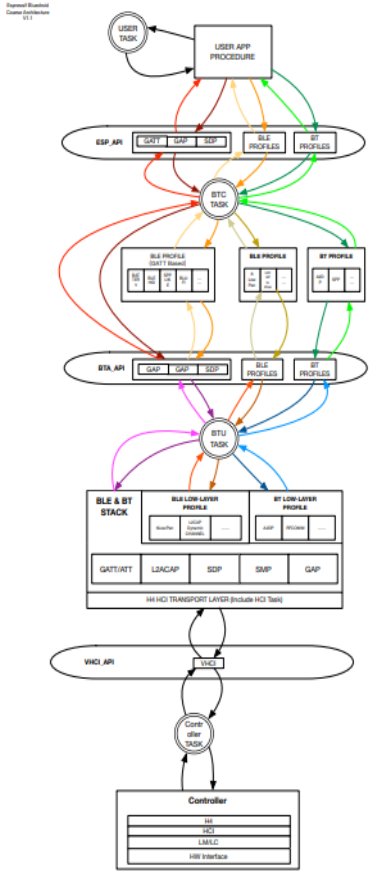
- Application Protocols:
  - HTTP (Server, Client),
  - HTTP2 Client,
  - MQTT Client
  - mDNS/DNS-SD Bonjour
  - COAP (Server, Client)

- TLS (Transport Layer Security)

Thank You!

Backup

ESP32 Bluedroid Architecture



Demo - Classic Bluetooth

- A2DP Sink
  - Advanced Audio Distribution Profile
  - How multimedia audio can be streamed over a Bluetooth connection
  - Can be used in conjunction with AVRCP for remote control on devices

Demo - Bluetooth Low Energy

- BLE Peripheral
  - BLE Advertisement
  - GATT Service - Light (custom)
  - Custom Characteristic - State (custom)
  - Control the light using BLE

Bluetooth 5.0

- Data transfer speeds upto 2Mbps
- Extended range
- Increased maximum Tx Power
- Increased broadcast message capacity