Mission possible: DIY smart home hub/controller

Martin Harizanov

Sofia, February 5th 2019

Agenda

- Agenda
- About me
- Goals
- Mission possible: DIY smart home hub/controller
- Q/A

About

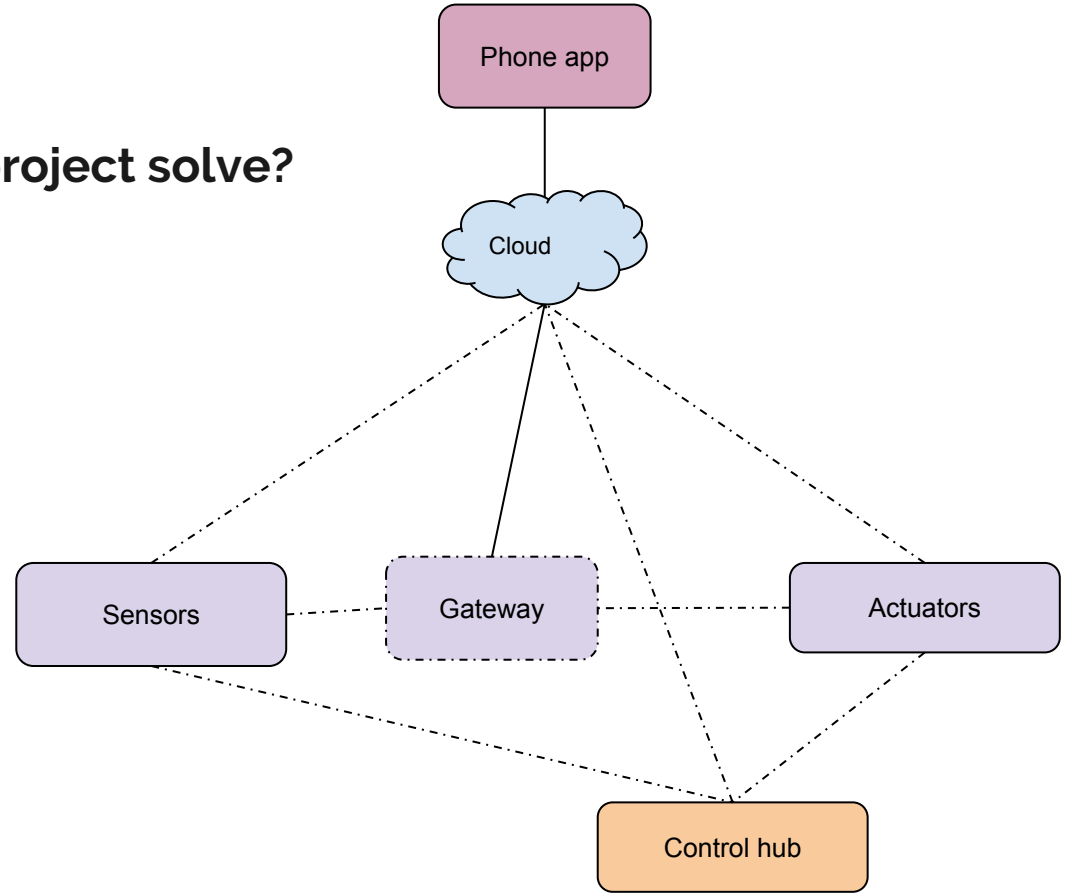
Martin Harizanov

Supervisor, Systems Engineering

Visteon

What problem does this project solve?

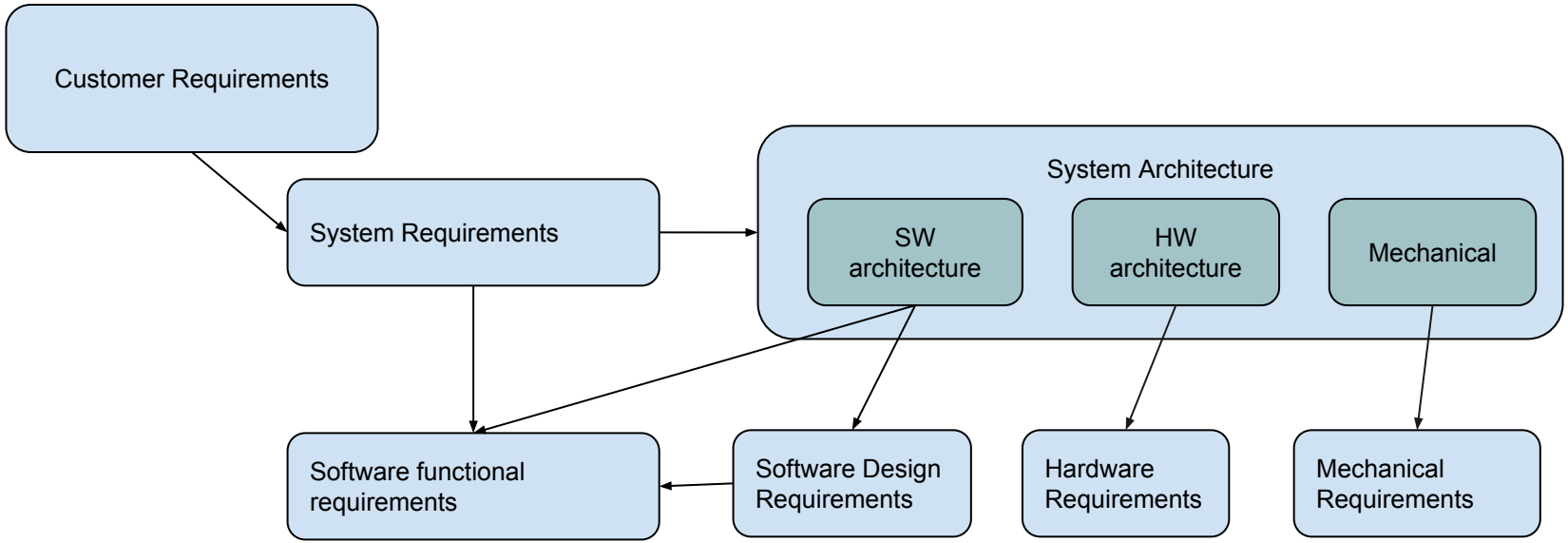
- Home automation control without internet connection
- Shared controls access
- Cost reduction by having control hubs and ‘dumb’ sensors/actuators



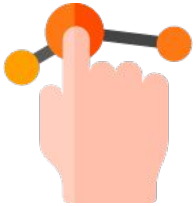
Requirements

- Control hub that integrates with the existing home automation
- Works regardless of the internet connectivity
- Visual and acoustic feedback
- Easy to use
- Reliable
- Low cost

Analysis steps



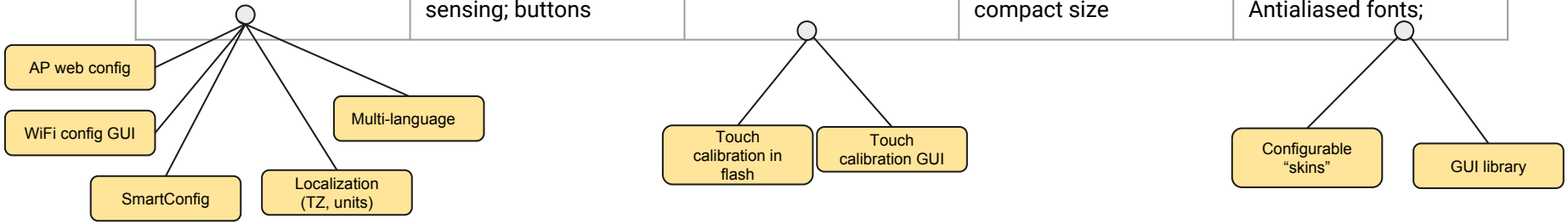
Non-Functional requirements



Usability

Ease with which the user is able to learn, operate, prepare inputs and interpret outputs through interaction with the system

Architecture	Hardware	Software	Mechanics	Design/UX
Localization, easy initial setup, wireless	Color TFT with touch, dimming, ambient light sensing; buttons	Precise touch detection/calibration	Option to hang on wall, magnetic attachment, compact size	Graphical HMI with touch, buttons; Antialiased fonts;



Non-Functional requirements

Reliability

Extent to which the system consistently performs the specified functions without failure

Architecture	Hardware	Software	Mechanics	Design/UX
Watchdog, OTA recovery, safe mode, degraded operation mode, crash recovery, task isolation/prioritization	Thermal, EMI consideration	Power management	Durable moving parts	

Non-Functional requirements

Security

Extent to which the system is safeguarded against deliberate and intrusive faults from internal and external sources

Architecture	Hardware	Software	Mechanics	Design/UX
TLS, RF encryption, mutual authentication, provisioning, access PIN	Encryption chip	Authentication on APIs, DDoS rate limiter, encrypted vFS, NVS	Reflashing port not available without product disassembly	

Non-Functional requirements

Availability

Degree to which users can depend on the system to be up and able to function during normal operating times

Architecture	Hardware	Software	Mechanics	Design/UX
Connections manager, operational state manager , fallback to backup connection. Revert to factory app	HW RTC to ensure time availability regardless of network availability, RF control as alternative to WiFi	Degraded (fail safe) mode		

Non-Functional requirements

Accessibility

Extent to which the system can be used by people with the widest range of capabilities

Architecture	Hardware	Software	Mechanics	Design/UX
	Piezo buzzer for acoustic feedback. RGB LED for visual feedback		Buttons easily pressable	Large contrast icons, color selection, acoustic feedback, visual feedback from distance, lighting sensing and brightness adjustment

..and more requirements

Non-functional

- Integrity
- Safety
- Environmental
- Flexibility
- Scalability
- Maintainability
- Performance
- Storage
- Regulatory
- .. and more

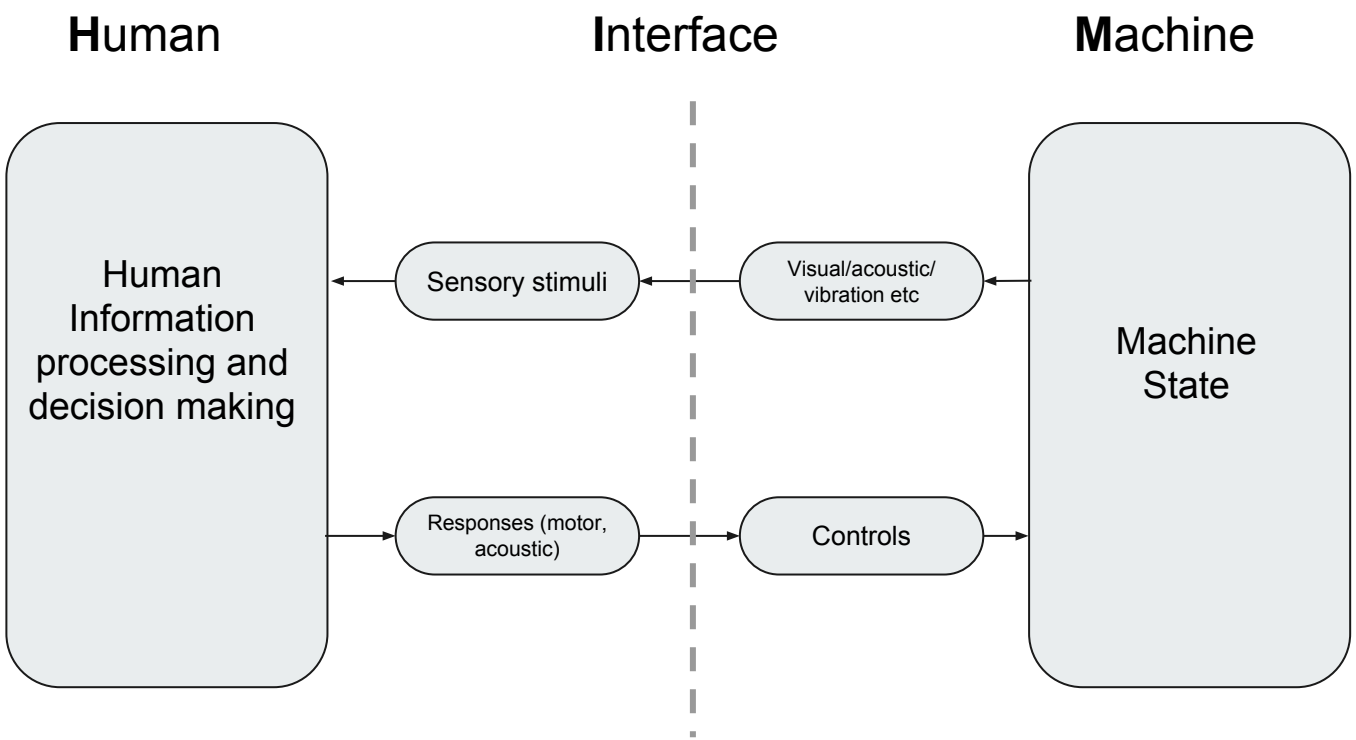
Functional

- Time & Date
- Temperature, Humidity, ambient light
- Configuration
- Connectivity management
- Diagnostics
- Security functions
- Power management
- Input devices

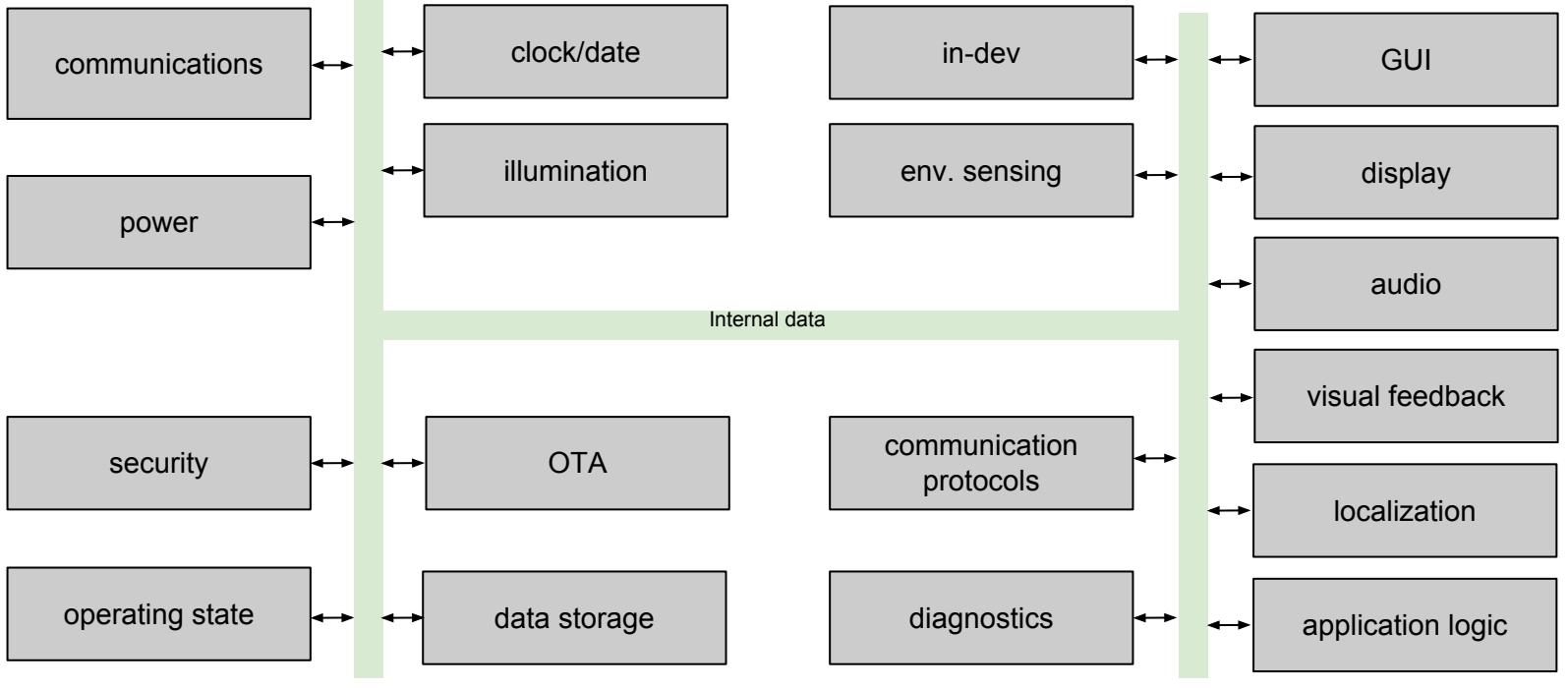
.. and more functional requirements

- Acoustic feedback
- Multi-language
- RF comms
- OTA
- Charts
- Thermostat
- HMI
- API

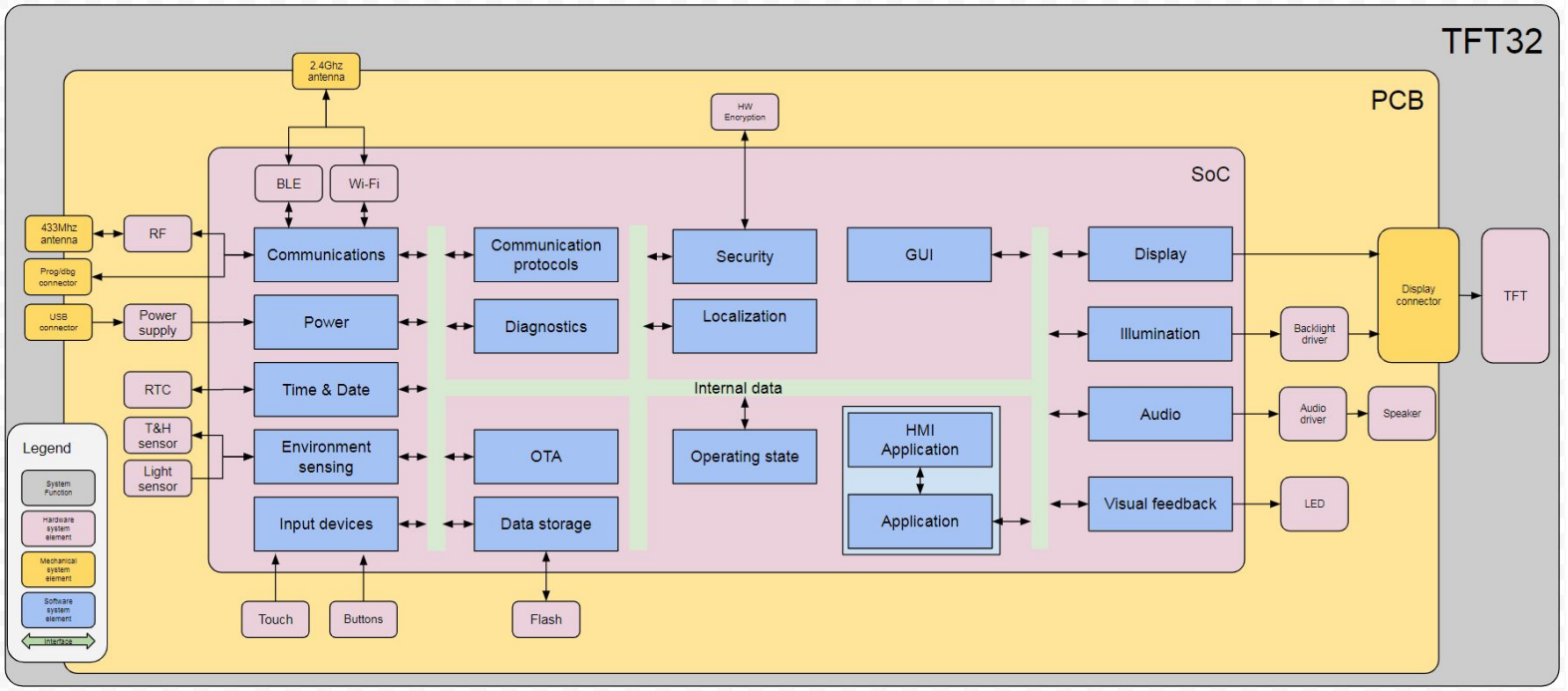
What is HMI?


Human
Interface
Machine
Human Information processing and decision making
Sensory stimuli
Visual/acoustic/ vibration etc
Machine State
Responses (motor, acoustic)
Controls

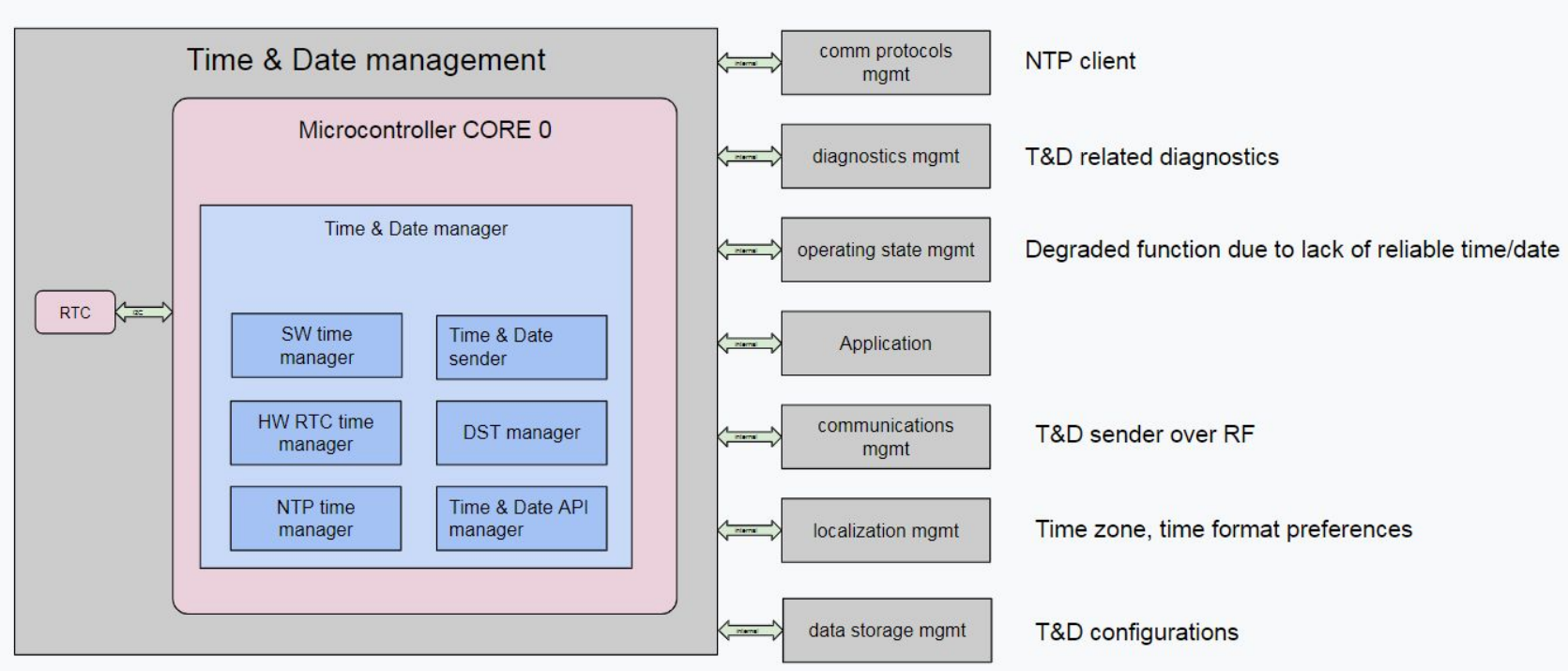
System functions diagram



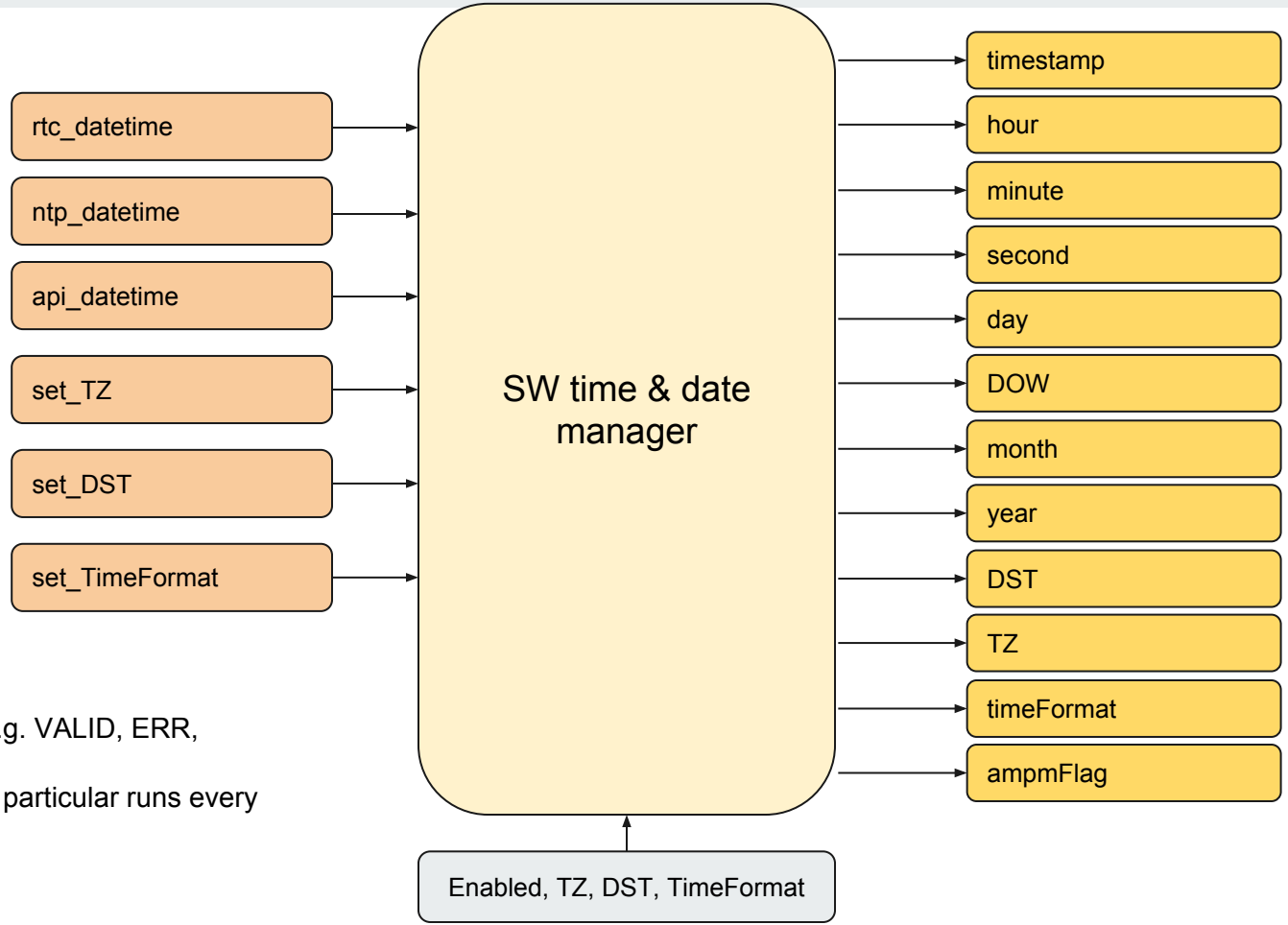
System function allocation diagram



System function block example

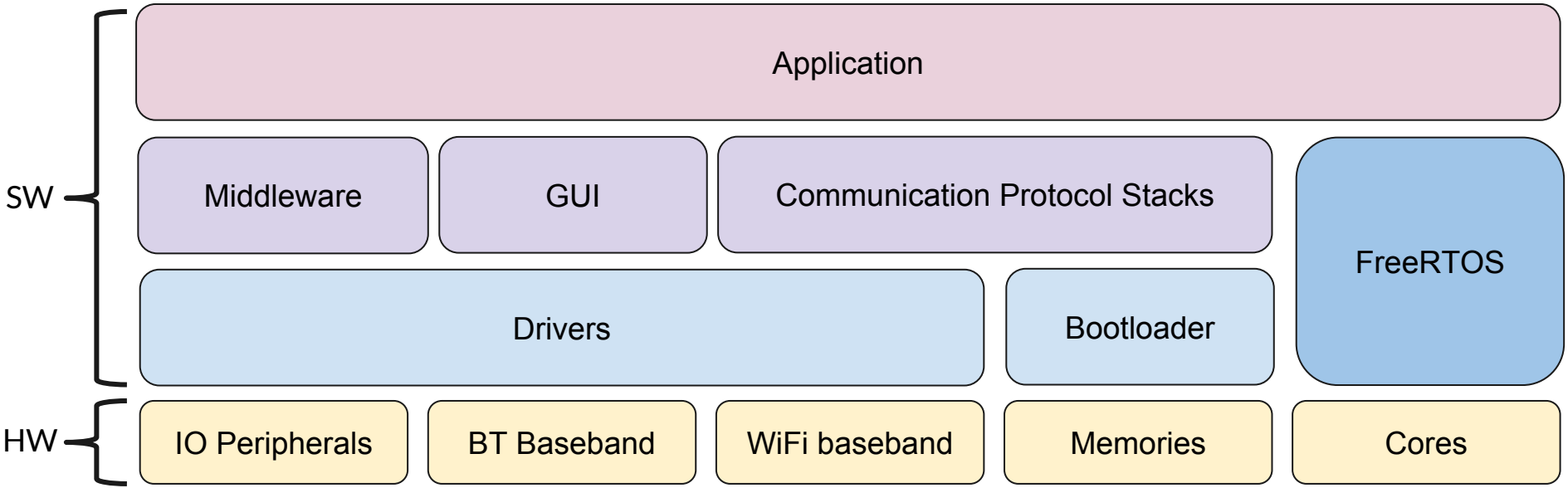


SWCs design



- Signals include STATE e.g. VALID, ERR, TMO, SNA
- SWC has cycle time, this particular runs every 1000ms

Layered Architecture

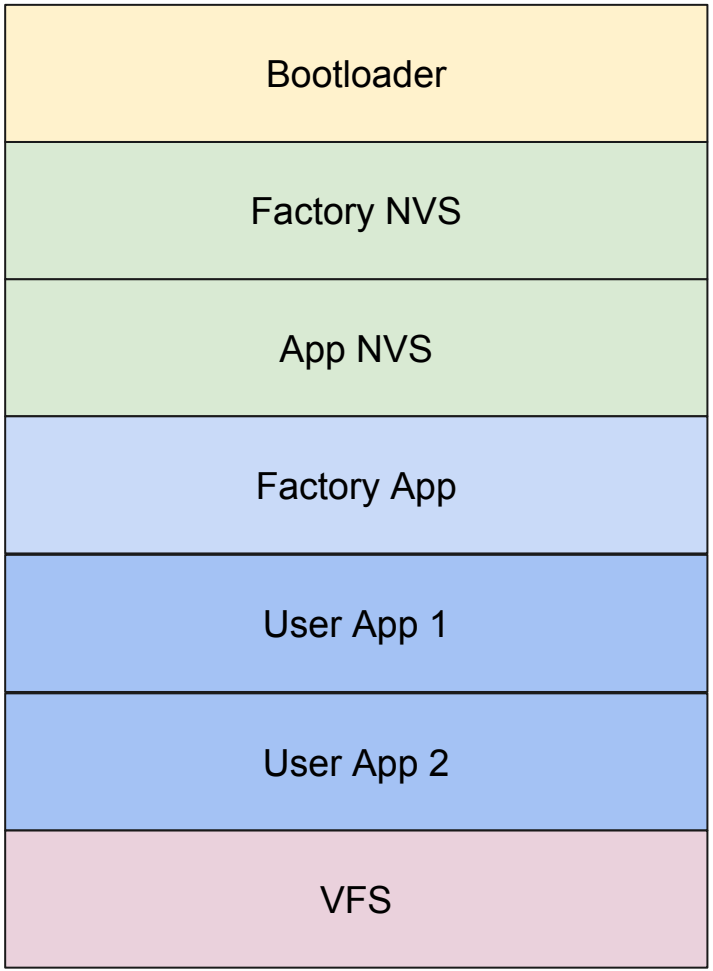


Fact break

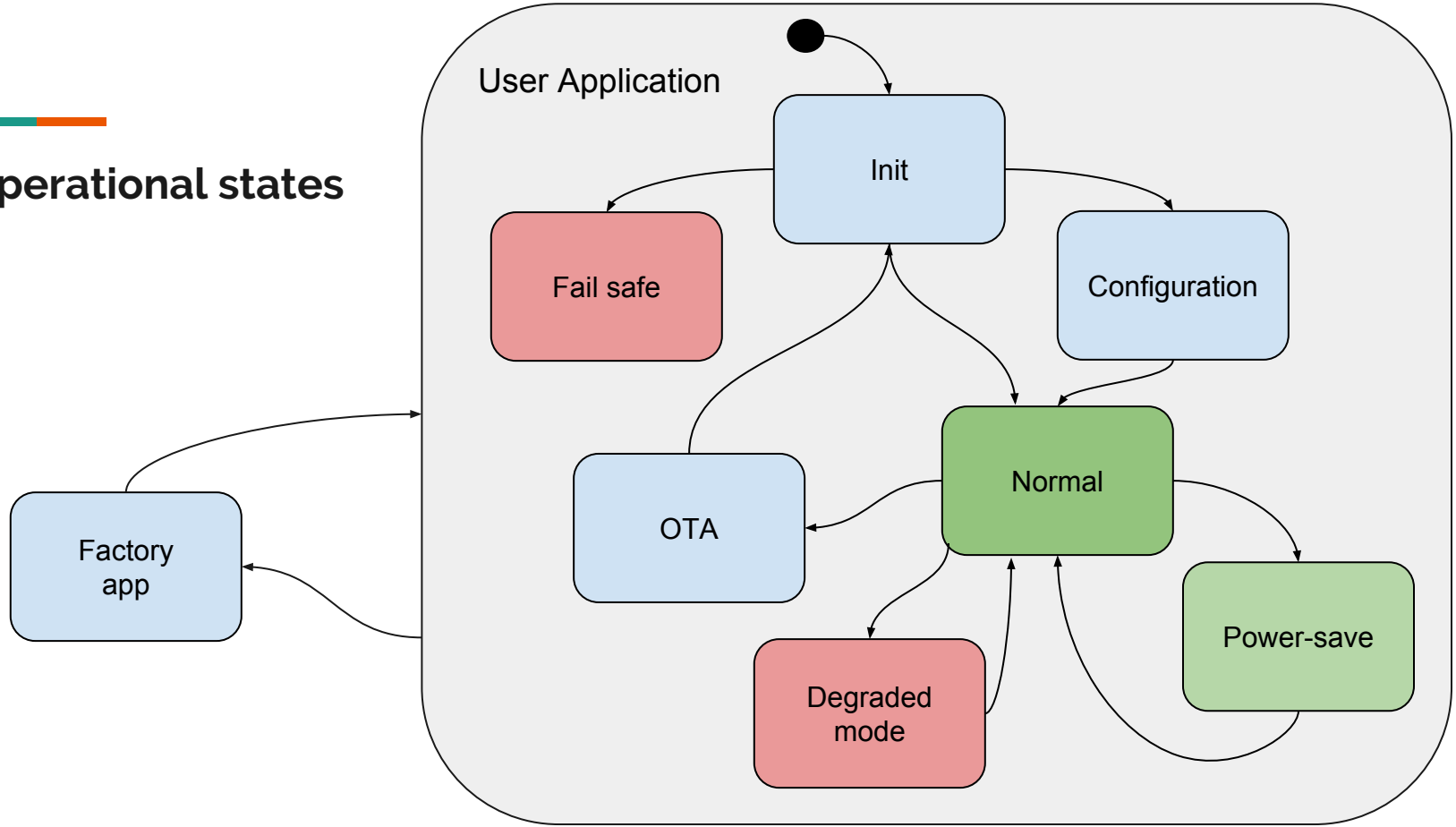
This project has **78,337** lines of code as of today

Partitions

- Flash layout
  - NVS for factory and user apps
  - Factory + user apps
  - VFS (SPIFFS, FAT etc)



Operational states



Power management strategies

- Dynamic Frequency Shifting
- Modem light sleep
- Variable WiFi transmission power
- Screen dimming
- GUI refresh rate reducing
- Peripheral sleep mode, whenever possible
- Overall: interrupt driven functionality vs polling

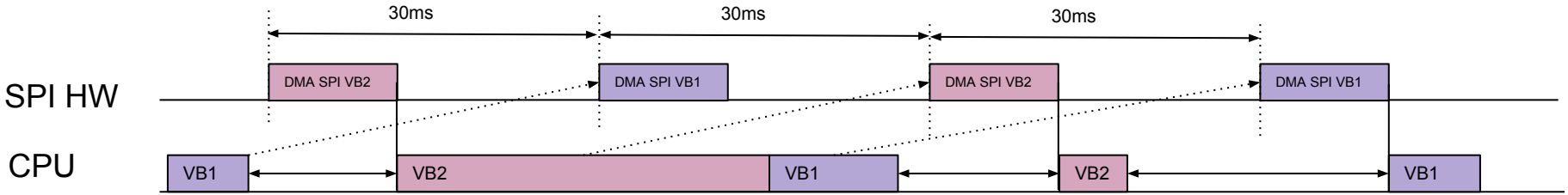
As a result: power consumption is ~60mA while maintaining WiFi connection and being fully operational

CPU freq	current consumption
XTAL(40MHz)	13.32mA
80MHz	22.85mA
160MHz	28.46mA
240MHz	39.95mA

OTA strategies

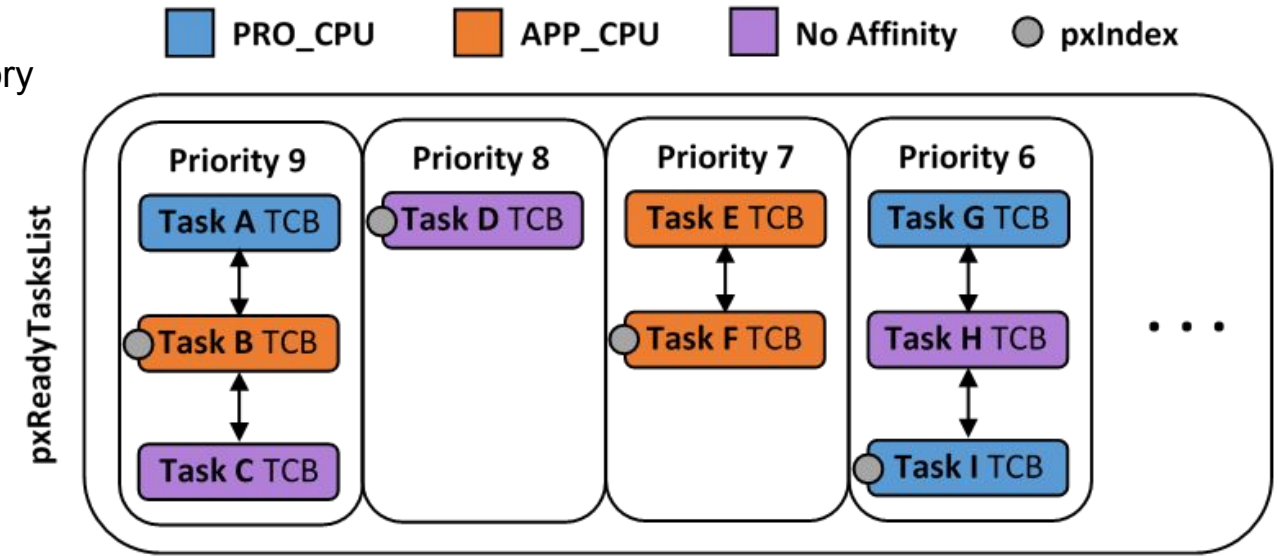
- OTA over web portal
- OTA over HTTPs
- OTA over MQTTs
- Compressed image OTA
- OTA rollback
  - Image verification
  - Tracking restarts due to crash
- Queued OTA
- OTA resume

DMA SPI and double buffering to speed up GUI rendering



Integrated Development Framework choice

- ESP-IDF using FreeRTOS
- Eclipse
- Github private repository



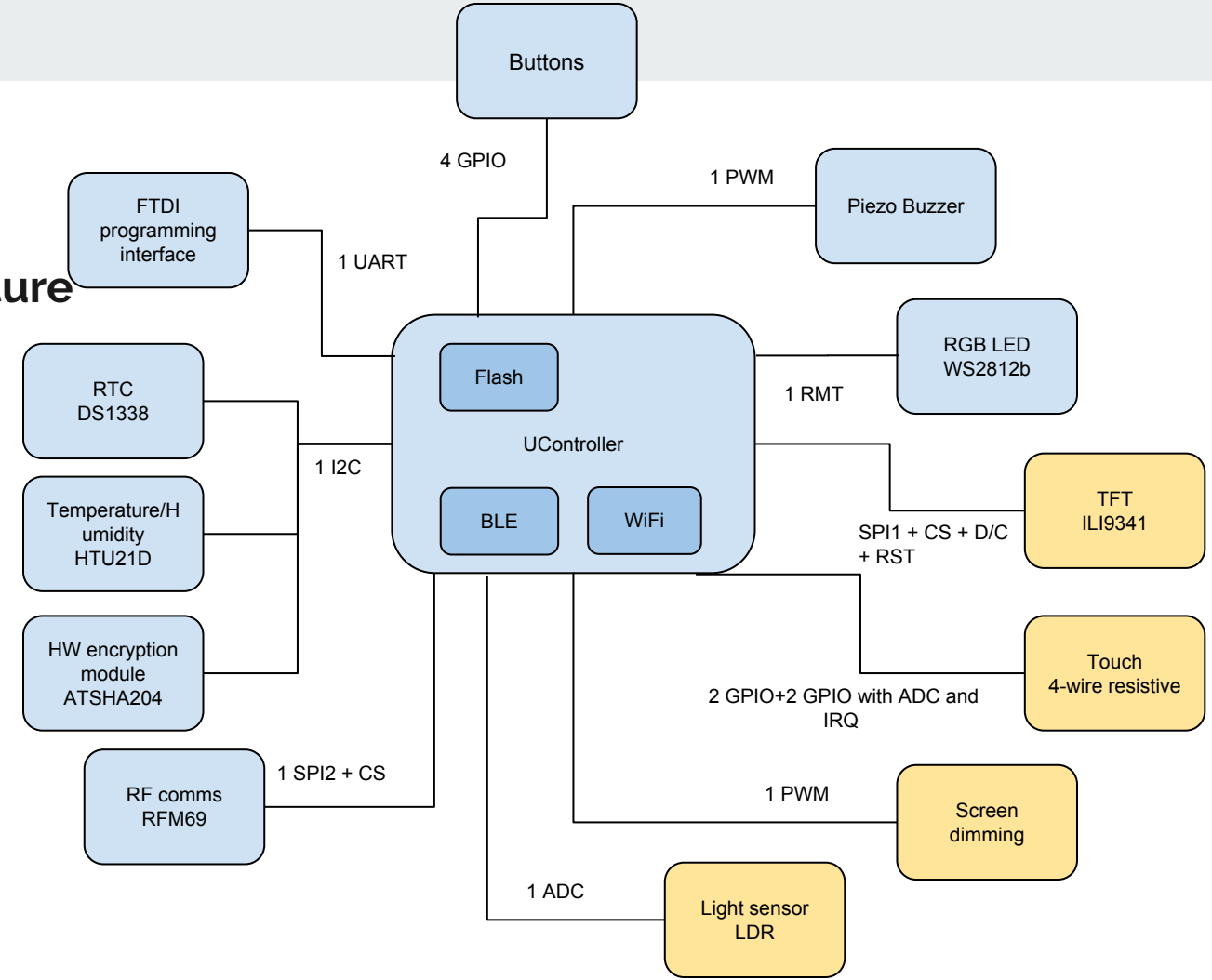
Fact break

This project took 7 months to develop

Physical architecture

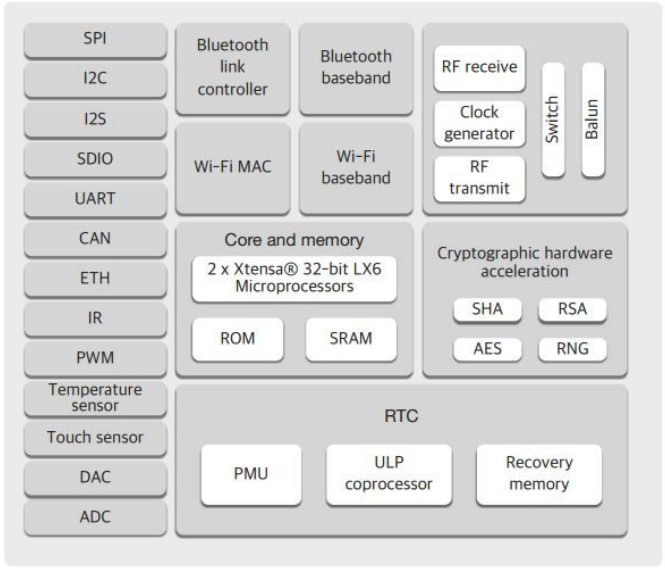
- GPIO 10 total (2 for touch + 3 for TFT CS, D/C and RST + 4 buttons +1 for RF CS)
- PWM 2 total (1 for piezo, 1 for screen dimming)
- RMT 1 total (1 for LED)
- SPI 2 SPI interfaces or 5 pins (1 for TFT CLK, MOSI only and 1 for RF CLS, MOSI, MISO)
- I2C 1 total or 2 pins
- ADC 3 total (1 for LDR and 2 for touch screen)
- UART - 1 total or 2 pins (RX/TX)

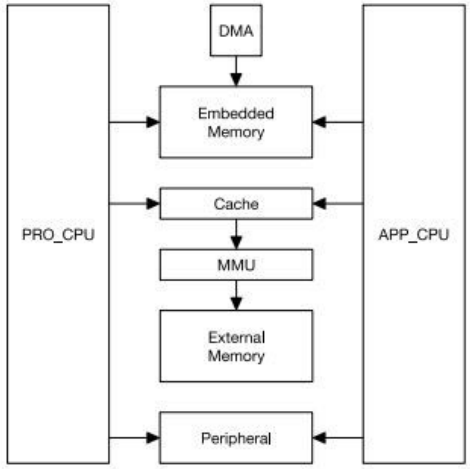
~23 GPIO pins needed



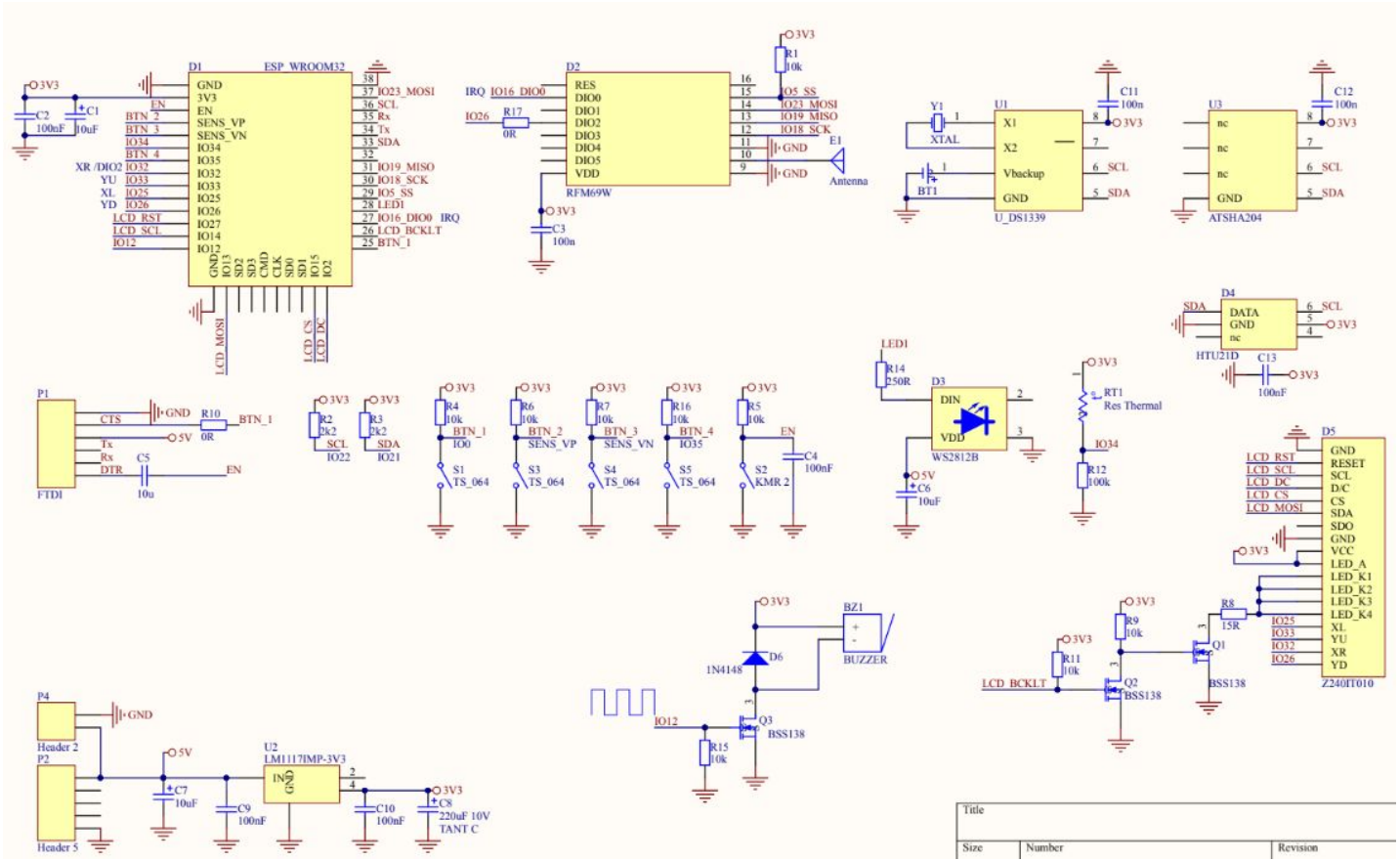
SoC choice

- ESP32 is dual core containing a Protocol CPU (known as CPU 0 or **PRO_CPU**) and an Application CPU (known as CPU 1 or **APP_CPU**). The two cores are identical in practice and share the same memory. This allows the two cores to run tasks interchangeably between them.

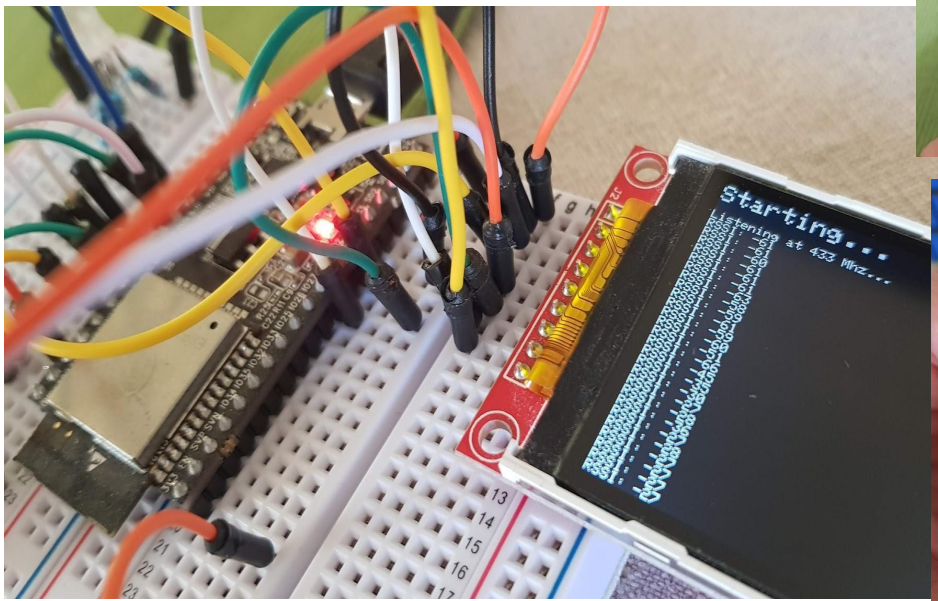


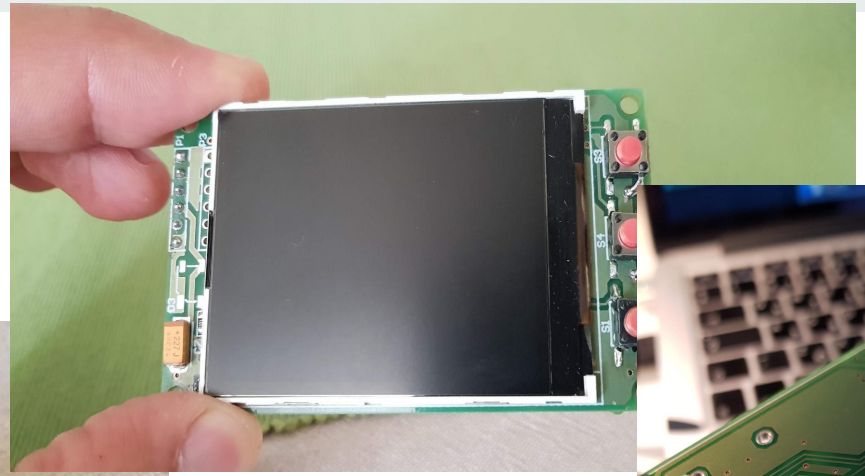


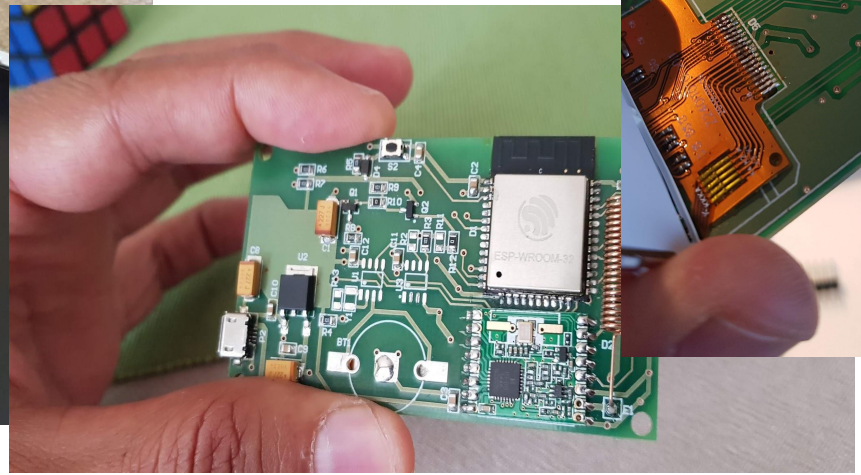
HW design


D1
ESP_WROOM32
GND
3V3
EN
SENS_VP
SENS_VN
IO34
IO35
IO32
IO33
IO25
IO26
IO27
IO14
IO12
IO23_MOSI
SCL
Rx
Tx
SDA
IO19_MISO
IO18_SCK
IO5_SS
LED1
IO16_DIO0
LCD_BCKLT
BTN_1
D2
RFM69W
RES
DIO0
DIO1
DIO2
DIO3
DIO4
DIO5
VDD
IRQ
Antenna
U1
U_DS1339
Vbackup
XTAL
BT1
U3
ATSHA204
D4
HTU21D
DATA
D3
WS2812B
DIN
VDD
LED1
RT1
Res Thermal
P1
FTDI
CTS
DTR
P4
Header 2
Header 5
U2
LM1117IMP-3V3
D5
Z240IT010
LCD_RST
LCD_SCL
LCD_DC
LCD_CS
LCD_MOSI
LED_A
LED_K1
LED_K2
LED_K3
LED_K4
XL
YU
XR
YD
BZ1
BUZZER
D6
1N4148
Q1
Q2
Q3
BSS138
S1
S2
S3
S4
S5
TS_064
KMR 2
Title
Size
Number
Revision

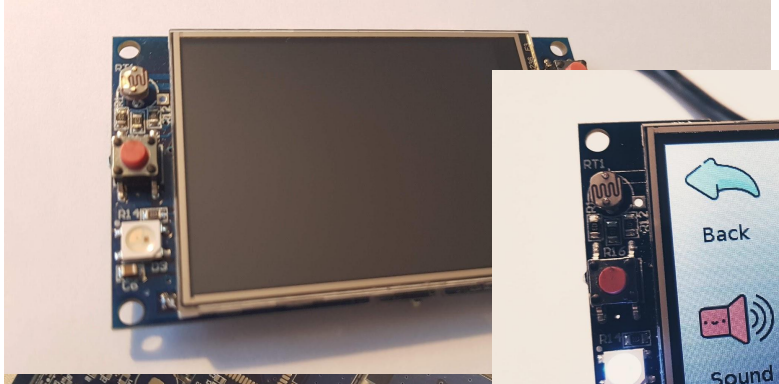
HW prototypes

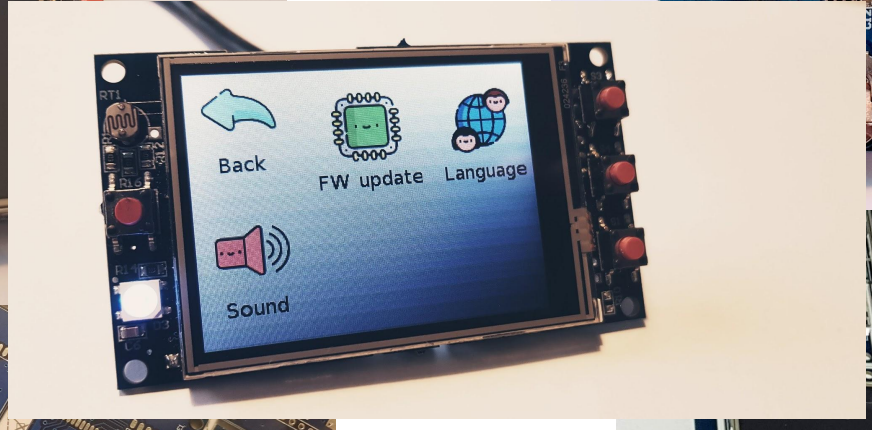


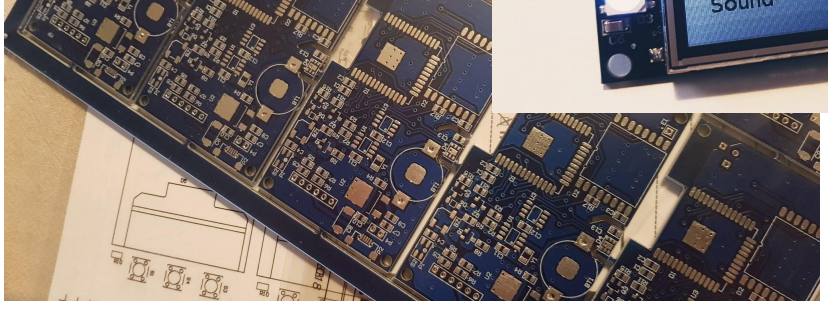


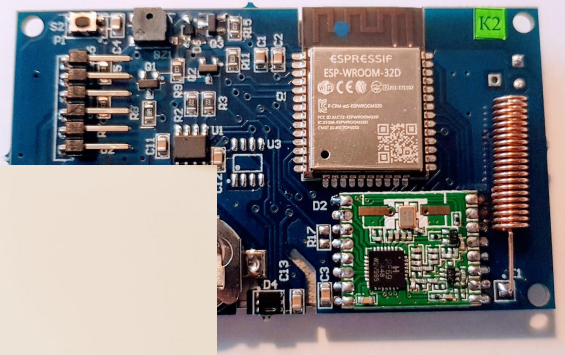


Final version … after six PCB revisions

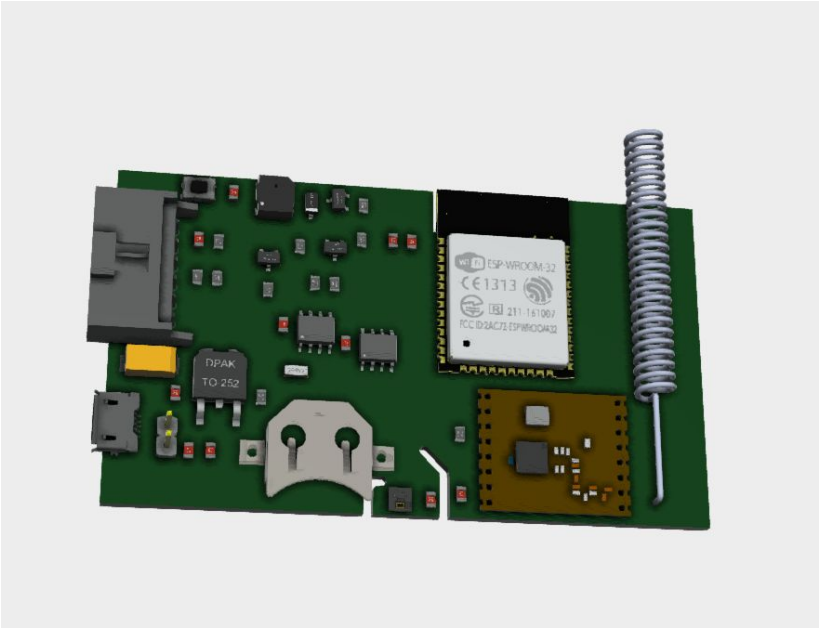


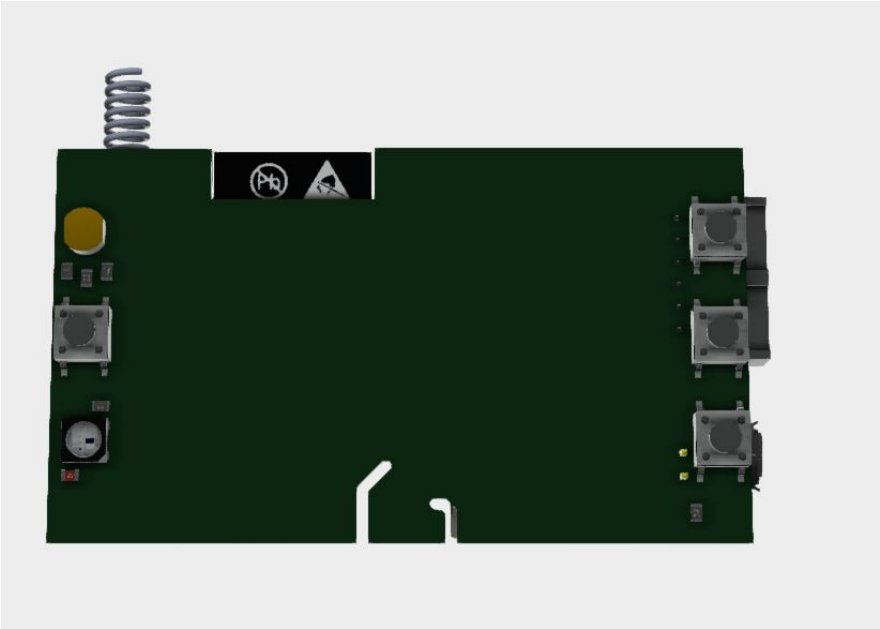




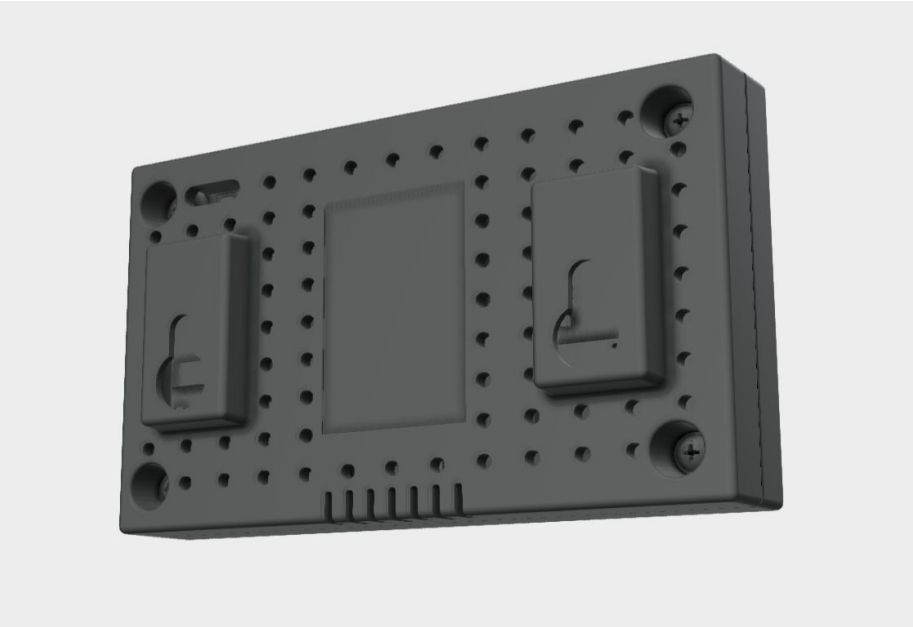


Mechanics - 3D PCB model for enclosure design



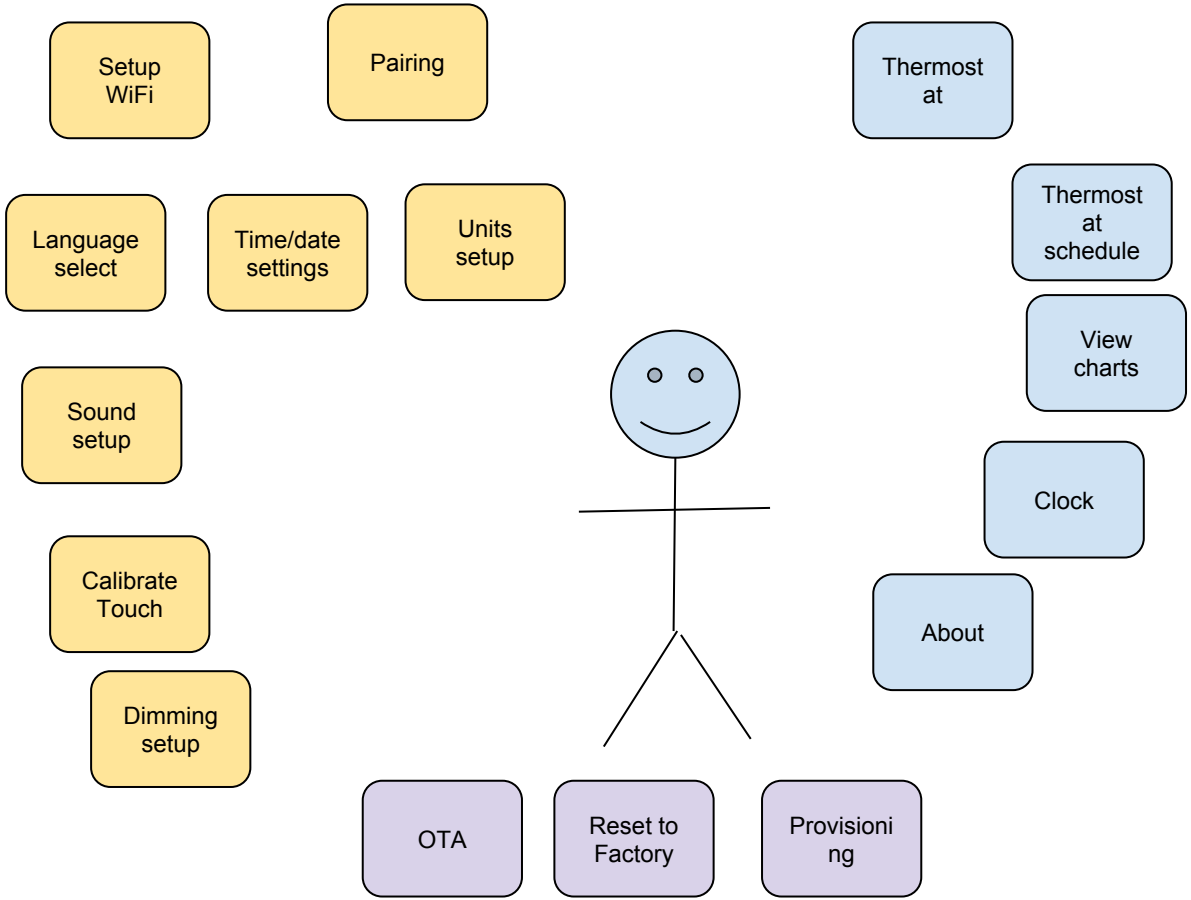


Mechanics - 3D printable enclosure

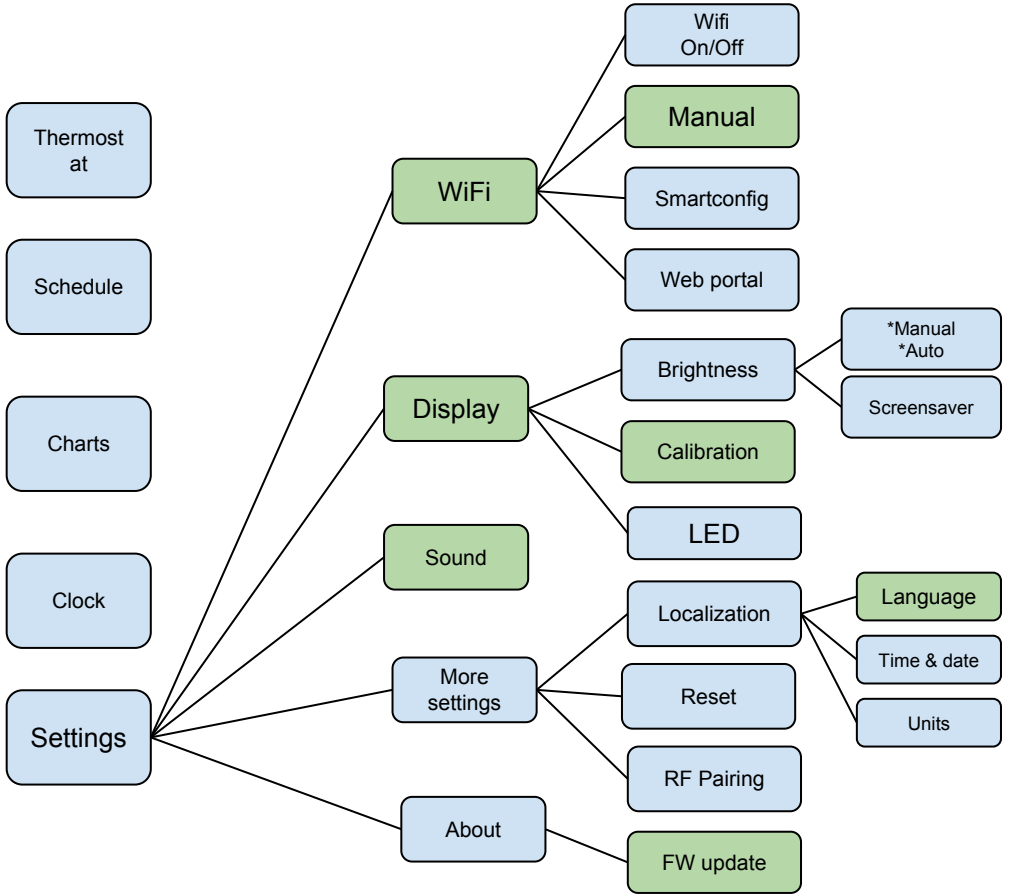


Mechanics - 3D printable enclosure

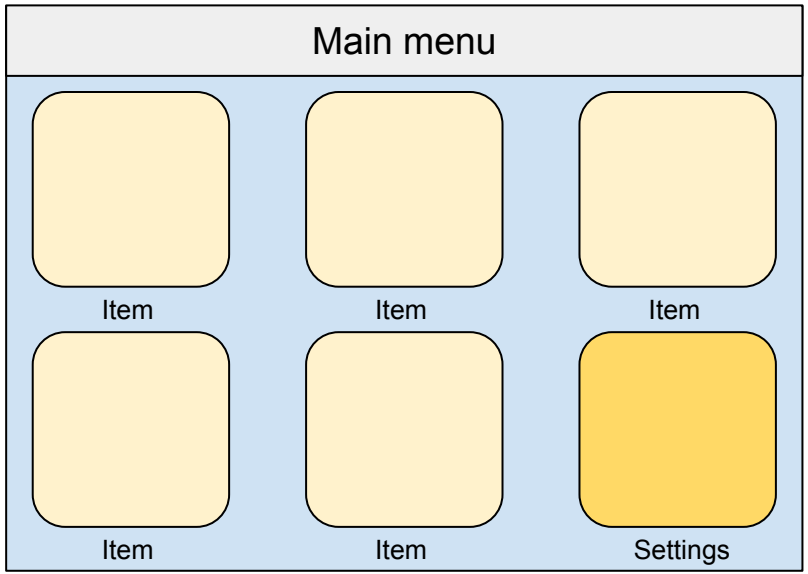
HMI Use cases

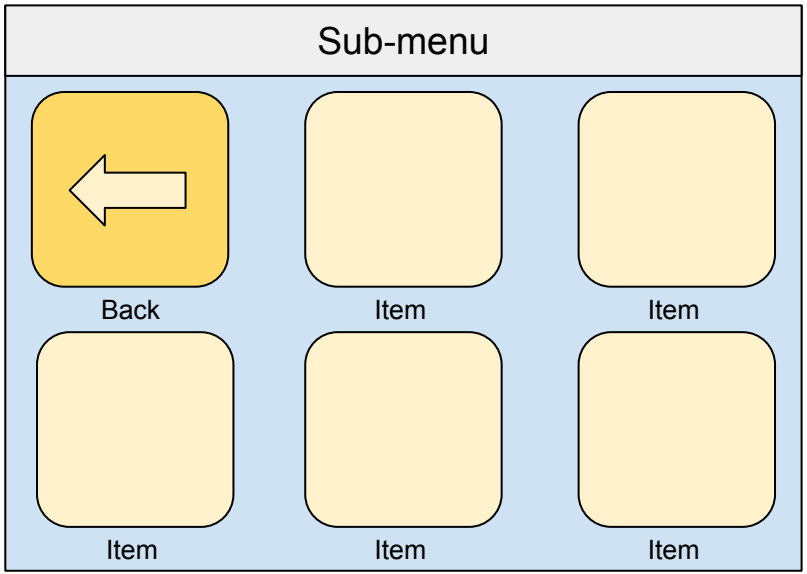


Menu structure

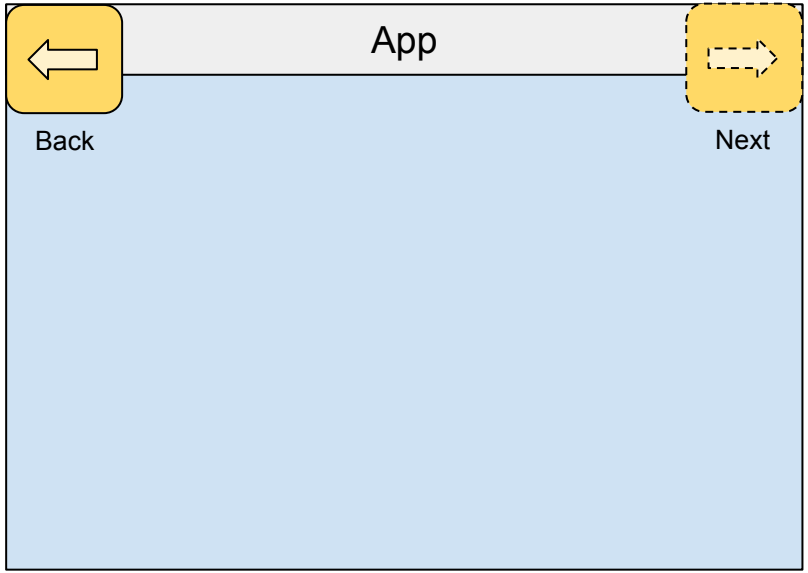


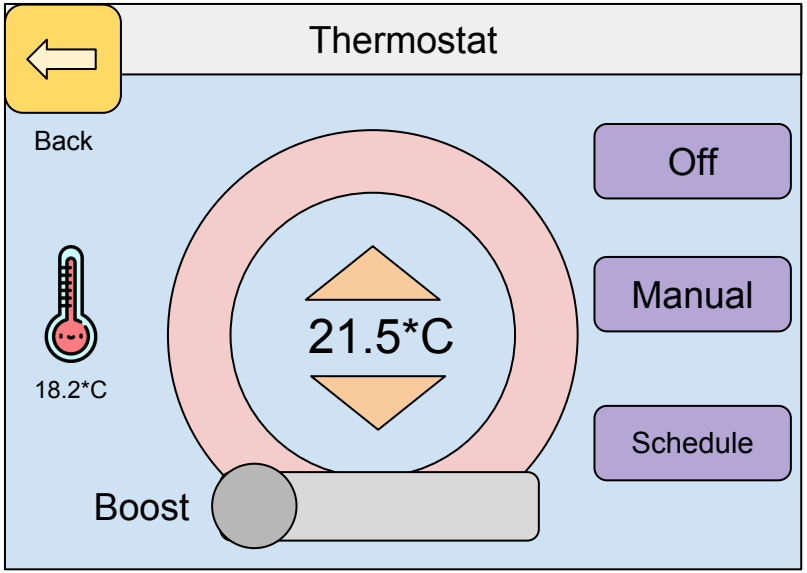
Menu design


Main menu
Item
Item
Item
Item
Item
Settings

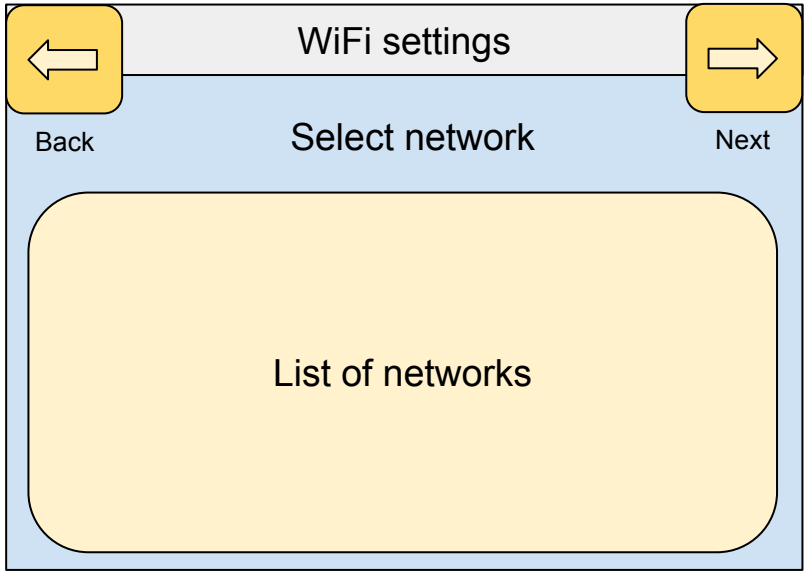

Sub-menu
Back
Item
Item
Item
Item
Item

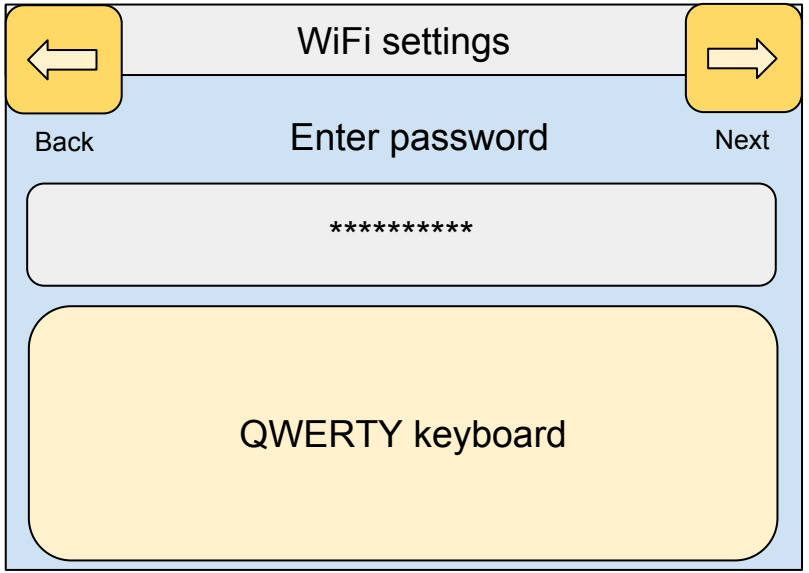
Menu design


App
Back
Next


Thermostat
Back
Off
Manual
Schedule
21.5*C
18.2*C
Boost

Menu design


WiFi settings
Back
Select network
Next
List of networks

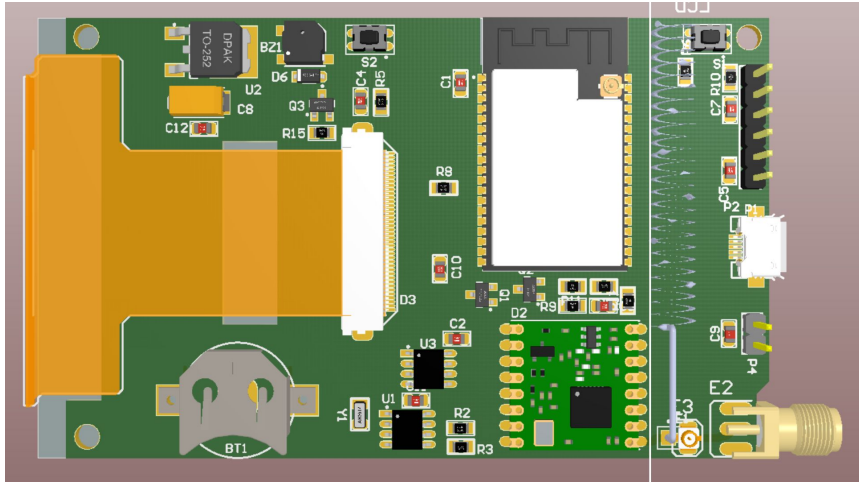

WiFi settings
Back
Enter password
Next
****
QWERTY keyboard

Challenges

- Driver conversion to FreeRTOS
- Temperature skew
- Enclosure design
- DMA, double video buffers
- Overall R&D process, as this is my first project using ESP32

Future plans

- Larger 3.5” TFT
- LoRa support
- No buttons, thin frame
- More powerful ESP32 package with PSRAM support
- Improved firmware functions



Questions?