Electronic Connectivity System source code organization

1. Project source tree overview

The ECS project has the following structure:

```
ECS/
     common/
     esp32_src/
          esp_app/
          esp_ota/
          components/
               app_aws_iot/
               app_flash/
               app_settings/
               app_sntp/
               BLE/
               http_server/
               json_utils/
               led_blinker/
     scripts/
     stm32_src/
     idf_update.sh
     Makefile
```

1.1 common/

Contains common files and sources.

Files presented is the certificates used by ESP32 and host scripts.

Sources is the SPI protocol definitions and CRC8 implementation both used by STM32 and ESP32.

1.2 esp32_src/

1.2.1 esp32_src/esp_app

Contains ESP32 startup and ESP32 device logic sources. All the other ESP32 component APIs are used here.

1.2.2 esp32_src/esp_ota

Not used in this project. Contains OTA for ESP32 firmware.

1.2.3 esp32_src/components

A different components that is used in device startup and business logics.

1.2.3.1 esp32_src/components/app_aws_iot

AWS IoT and MQTT – related functions.

1.2.3.2 esp32_src/components/app_flash

Contains functions, related to flash memory. Almost not used and to be replaced by “app_settings”.

1.2.3.3 esp32_src/components/app_settings

Contains functions for settings JSON file handling. Settings file is flashing directly into ESP32 by esp32_flash_settings.sh script.

1.2.3.4 esp32_src/components/app_sntp

A few SNTP routines.

1.2.3.5 esp32_src/components/BLE

BLE routines. Contains BLE initialization, GAP and GATT service handlers and user BLE task.

1.2.3.6 esp32_src/components/http_server

HTTP server implementation needed for WiFi credentials setup

1.2.3.7 esp32_src/components/json_utils

Contains 1 function – jsmn_get_key_index().

1.2.3.8 esp32_src/components/led_blinker

Contains a task for LED blinking in a different modes (started, connected and http_server).

1.3 stm32_src/

Contains STM32 sources.

The main sources are SPI protocol state machine (src/spi_protocol.c) and SPI protocol handlers (src/spi_protocol_handlers.c).

State machine contains the behaviour implementation based on a received byte. Handlers contains the functions that’s invoking by state machine which is independent by nature of SPI protocol and SPI at all (temperature read, RFID transmit, discrete inputs read etc.)

1.4 scripts/

1.4.1 esp32_erase.sh
Erase all flash memory in ESP32.
Example: ./esp32_erase.sh /dev/ttyUSB0

1.4.2 esp32_flash_all.sh
Flash app, settings and certificates into ESP32.
Example: ./esp32_flash_all.sh /dev/ttyUSB0

1.4.3 esp32_flash_app.sh
Flash app only.
Example: ./esp32_flash_app.sh /dev/ttyUSB0

1.4.4 esp32_flash_aws_certs.sh
Flash AWS certificates only.
Example: ./esp32_flash_aws_certs.sh /dev/ttyUSB0
Note: for now we're using only certs/3c81… certs for all devices. Later this script will be changed with certificates path argument.

1.4.5 esp32_flash_ota_loader.sh
Flash OTA loader
Example: ./esp32_flash_ota_loader.sh /dev/ttyUSB0

1.4.6 esp32_flash_settings.sh
Flash settings JSON file
Example: ./esp32_flash_settings.sh settings.json /dev/ttyUSB0

1.4.7 stm32_flash.sh
Flash STM32 firmware.
Example: ./stm32_flash.sh

1.4.8 host_pub_to_topic.sh
Publish into specified MQTT topic. Currently, MQTT endpoint is hardcoded into script (to avoid typos when testing) and certificates used is taking from common/certs_3c81… directory. Useful to check host connection to endpoint.
Example: ./host_pub_to_topic.sh "test_topic" "payload"

1.4.9 host_subscribe_to_topic.sh
Subscripe to specified MQTT topic. Currently, MQTT endpoint is hardcoded into script (to avoid typos when testing) and certificates used is taking from common/certs_3c81… directory. Useful to check MQTT connectivity and watch the MQTT traffic from device.
Example: ./host_subscribe_to_topic.sh "test_topic"