Internet of Things

Laborator 4

Over The Air Programming (OTAP)
(anul universitar 2022-2023)

Universitatea Tehnică ”Gheorghe Asachi” din Iaşi
Facultatea de Automatică şi Calculatoare
Departamentul de Calculatoare

Cuprins

Listă tabele

Listă figuri

1. Scopul laboratorului

- Familiarizarea cu conceptul de actualizare a firmware-ului prin intermediul unei interfeţe de comunicaţie generice.
- Dezvoltarea unei aplicaţii care foloseşte o conexiune Wi-Fi pentru actualizarea codului.

2. OTAP (Over the Air Programming)

2.1 Introducere

După cum îi spune şi numele, o metoda OTAP presupune actualizarea firmware-ului prin intermediul unei interfețe de comunicaţie radio. Principalul avantaj al acestei metode constă în faptul că un sistem poate fi actualizat după ce a fost dat în folosinţă (ex. instalat într-o locaţie greu accesibilă), iar principalul dezavantajul constă în complexitatea suplimentară a logicii de funcţionare a aplicaţiei.

În cadrul unui proces OTAP se disting următoarele etape:

- Preluarea noului cod (fişier) binar prin intermediul canalului de comunicatie (Wi-Fi, Bluetooth, alte tipuri de comunicaţii proprietare)
- Verificarea consistenţei codului binar (pas opţional)
- Scrierea noului cod în memoria Flash
- Modificarea secvenţei de boot pentru a folosi noul cod la următoarea pornire a sistemului

2.2 Implementarea *esp-idf*

esp-idf pune la dispoziţie două componente pentru implementarea unei soluţii OTAP:

- Mecanism de partiţionare al memoriei Flash
- Bibliotecă folosită pentru scrierea memoriei Flash (şi managementul partiţiilor)

Memoria Flash ataşată unui SoC ESP32 poate să conţină maxim 95 de partiţii, acestea putând fi folosite pentru stocarea codului sau a datelor. Tabela de parţii este scrisă la deplasamentul 0x8000 şi are dimensiunea de 3072 de octeţi. La nivelul framework-ului, tabela este descrisă prin intermediul unor fişiere csv cu o structură specifică.

Tabela implicită folosită pentru aplicaţiile dezvoltate în laboratoarele trecute are următoarea structură (şi este descrisă în fişierul `partitions_singleapp.csv` din `$ESP-IDF-DIR$ /components/partition_table`):

```
# Espressif ESP32 Partition Table
# Name, Type, SubType, Offset, Size, Flags
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 1M,
```

Listing 1. Structura `partitions_singleapp.csv`

Partiţia *nvs* este folosită pentru stocarea unor informaţii de iniţializare şi de stare de către bibliotecile mai complexe (ex. Wi-Fi, Bluetooth, lwIP), partiţia *phy_init* este folosită pentru stocarea datelor de calibrare a perifericului radio, iar partiţia *factory* este folosită pentru stocarea aplicaţiei.

În cazul folosirii mecanismului OTA, vor exista două sau mai multe partitii de tip *app*, subtip *ota_x* (unde x ia valori de la 0 la 15) şi o partiţie de tip *data*, subtip *ota* care va conţine informaţiile de boot.

```
# Espressif ESP32 Partition Table
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x4000,
otadata,  data, ota,     0xd000,  0x2000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  0,    0,       0x10000, 1M,
ota_0,    0,    ota_0,   0x110000, 1M,
ota_1,    0,    ota_1,   0x210000, 1M,
```

Listing 2. Structura unei tabele cu două partiţii ota

Info

- Dacă partiţia *otadata* nu contine date (este iniţializată cu octeţi 0xFF), bootloader-ul va încărca aplicaţia implicită din partita *factory*.
- Dacă se doreşte revenirea la aplicaţia implicită (din partiţia *factory*) după ce a fost realizată o actualizare OTAP, conţinutul partiţiei *otadata* trebuie şters.

Dacă partiţia *otadata* indică existenţa a cel puţin o aplicaţie într-o partiţie *ota_x*, bootloader-ul va rula aplicaţia cea mai nouă (vechimea este exprimată prin intermediul unei variabile contor).

Scrierea partiţiilor *ota_x* se realizează alternativ (figura 1), pentru a avea întotdeauna disponibilă în memoria flash şi versiunea anterioară. Astfel, în situaţia în care ultima versiune are bug-uri, se poate reveni la codul anterior pentru a avea în continuare la dispoziţie mecanismul OTAP.

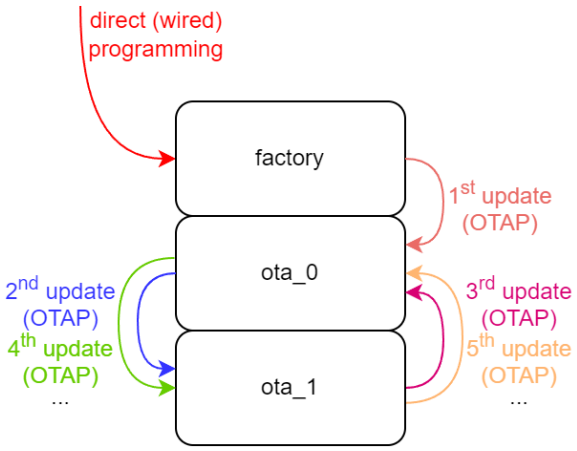

Fig. 1. Secvenţa de utilizare a partiţiilor

2.3 Model de interacţiune

În figura 2 este prezentat un ciclu de interacţiune OTAP. Astfel, pe platformă se încarcă aplicaţia iniţială cu suport OTAP (pas 1), imaginile pentru actualizări fiind ulterior disponibile la nivelul unui server web (pas 2). La generarea unui eveniment de actualizare (ex. apăsare de buton, verificare periodică)(pas 3), platforma ESP32 rulează un client HTTP şi preia de la server noua imagine (pas 4). După scrierea imaginii în următoarea partiţie *ota_x* marcată ca fiind disponibilă, platforma se va resetea şi va rula noul cod (pas 5).

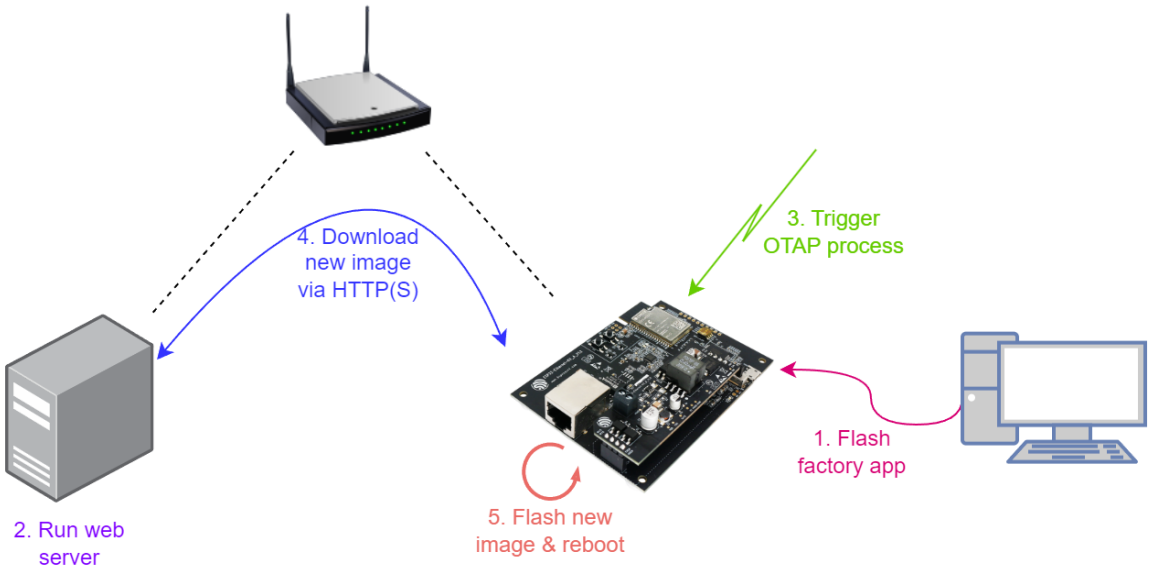

Fig. 2. Modelul de interacţiune de la nivelul unei aplicaţii OTAP via HTTP

3. Sarcini de lucru

1. Urmaţi paşii descrişi în continuare pentru rularea scenariului de bază OTAP:

   - Creaţi un proiect nou şi adăugaţi fişierele sursă disponibile pe platforma Moodle în următorul fel:
     - `main.c` în subfolderul `src`
     - `ca_cert.pem`, `ca_key.pem` şi `server.py` în rădăcina proiectului
   - Pentru folosirea tabelei de partiţii prezentate în listing-ul 2 se va adăuga linia `board_build.partitions = partitions_two_ota.csv` în fişierul *platformio.ini*.
   - Pentru încărcarea certificatului de securitate (ce va fi folosit de serverul web) în memoria flash a ESP32 se va adăuga linia `board_build.embed_txtfiles = ca_cert.pem` în fişierul *platform-io.ini*. De asemenea, pe ultima linie a fişierului *CMakeLists.txt* din subfolderul `src` se va adăuga comanda `target_add_binary_data(${COMPONENT_TARGET} "../ca_cert.pem" TEXT)`
   - Nu uitaţi să adăugaţi în fişierul *platformio.ini* şi restul directivelor indicate în primul laborator!!!
   - Se încarcă codul pe platforma ESP32 ca şi până acum. Se porneşte monitorul serial pentru monitorizarea mesajelor generate de aplicaţie
   - Scriptul *server.py* rulează un server HTTPS care expune fişierul binar rezultat în urma compilării proiectului.

     Se deschide un *Command prompt* sau un nou terminal VSC şi se rulează scriptul (`python server.py`). Acesta foloseşte modulul `flask`. În cazul în care modulul nu este instalat se rulează comanda `pip install flask`.
   - Se apasă butonul conectat la platforma ESP32 şi aplicaţia execută paşii 4 şi 5 prezentaţi mai sus.

2. Aplicaţia anterioară se va extinde pentru a integra un mecansim de preluare a codului pe bază de versionare.

   - În rădăcina proiectului se adaugă scriptul *versioning.py*. În *platformio.ini* se adaugă linia `extra_scripts = pre:versioning.py`.

     Astfel, la fiecare rulare a comenzii **Build**, înainte de începerea procesului de compilare se rulează scriptul *versioning.py*, script care "contorizează" numărul de compilări şi generează un header *version.h*. Acesta conţine trei directive define ce pot fi folosite pentru identificarea versiunii curente.
   - Se va modifica scriptul *server.py*. Serverul HTTP trebuie să expună o resursă suplimentară care va returna versiunea codului prin citirea fişierului *version.h*.

- Se va modifica codul care rulează pe ESP32. În pasul 4 se va prelua versiunea codului disponibil la nivelul serverului HTTP şi codul va fi descărcat doar dacă este mai nou decât cel care rulează pe ESP32. Se va folosi componenta software *esp_http_client*. Documentaţia este disponibilă aici, exemplu de cod este disponibil aici.

Materiale de studiu

- Foaie de catalog şi manual ESP32
- Documentaţie OTA
- Documentaţie tabele de partiţii