

Laborator 4

Tooling

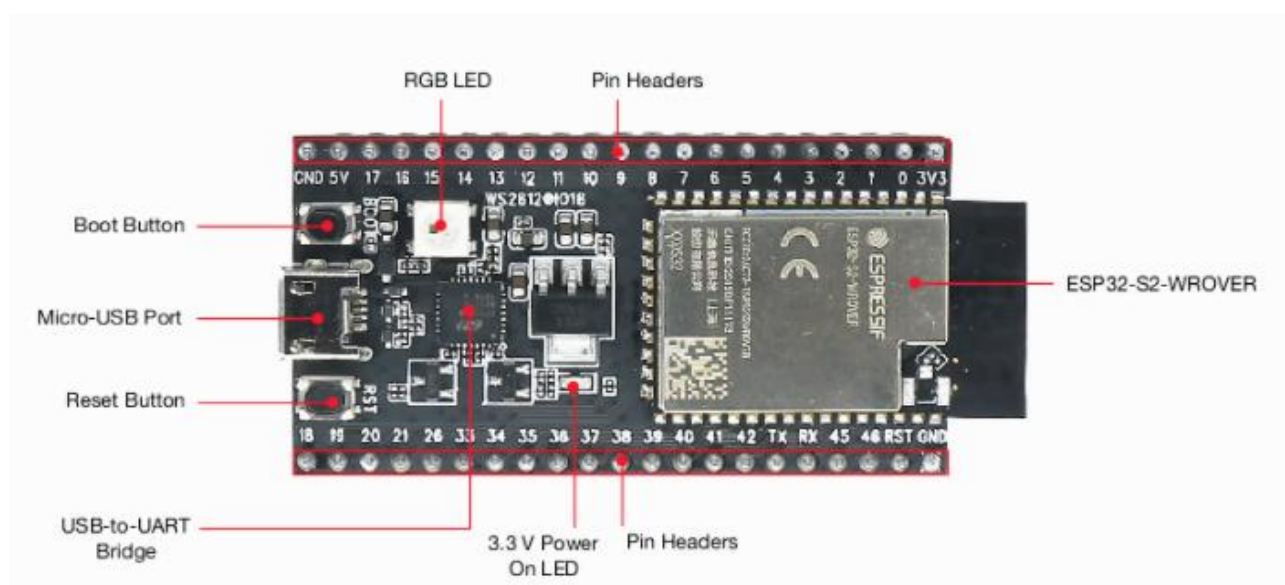
Obiective

- Familiarizarea cu placa de dezvoltare
- Familiarizarea cu mediul de dezvoltare
- Înțelegerea conceptului de “version control”

Cuprins

Obiective	1
Cuprins	1
Prezentarea plăcii de dezvoltare	1
Prezentarea mediului de dezvoltare	2
Crearea unui nou proiect	2

Prezentarea plăcii de dezvoltare



Placa de dezvoltare folosită la acest laborator va fi: **ESP32-S2-Saola-1**; această placă de dezvoltare este proiectată având la bază chipul **ESP32-S2-WROVER**.

ESP32-S2-WROVER este un modul MCU Wi-Fi care integrează ESP32-S2. La baza acestui cip se află un procesor Xtensa® LX7 pe 32 de biți care operează la până la 240 MHz.

Are o antenă PCB, un flash SPI extern de 4 MB și un PSRAM suplimentar de 2 MB.

Prezentarea mediului de dezvoltare

Un mediu de dezvoltare (engl. software development environment, sau integrated development environment - "mediu integrat de dezvoltare") este un set de aplicații care ajută programatorul în scrierea programelor.

Un mediu de dezvoltare combină toți pașii necesari creării unui program:

- editarea codului sursă;
- compilarea;
- depanarea (debug);
- testarea;
- generarea de documentație.

Principalele componente ale unui mediu de dezvoltare sunt editorul de cod sursă și depanatorul. Mediile de dezvoltare apelează compilatoare sau interpretoare, care pot veni în același pachet cu mediul însuși sau pot fi instalate separat de către programator.

De obicei un mediu de dezvoltare este specific unui anumit limbaj de programare, însă există la ora actuală și medii de dezvoltare care pot lucra cu mai multe limbaje, de ex. Eclipse sau Microsoft Visual Studio.

Mediul de dezvoltare folosit în cadrul laboratorului este **Visual Studio Code**. Acesta permite editarea și compilarea codului sursă.

Crearea unui nou proiect

Pentru a crea un nou proiect, vom folosi arhiva **ESP3202112021.ZIP**. **Dezarhivarea se va face pe calea "d:\ESP32"**. În această arhivă vom găsi mediul de dezvoltare **Visual Studio Code** împreună cu toate componentele necesare rulării programului de dezvoltare și 3 fișiere cu extensia *.bat:

- CreateBlankProject.bat – pentru crearea unui proiect nou,
- LaunchEnvironment.bat – pentru deschiderea IDE-ului,
- VisualStudioCode.bat – pentru configurarea parametrilor corespunzători plăcii de dezvoltare (COM, Baud Rate, etc.); acesta va apela și **LaunchEnvironment.bat** pentru a deschide IDE-ul.

Fișierul „Makefile” conține detalii despre proiect pentru a putea fi compilat. În folderul „components” o să găsim componentele create de noi după structura de mai sus și în folderul „main” o să găsim, de obicei, punctul de intrare principal al programului, adică funcția „app_main”.

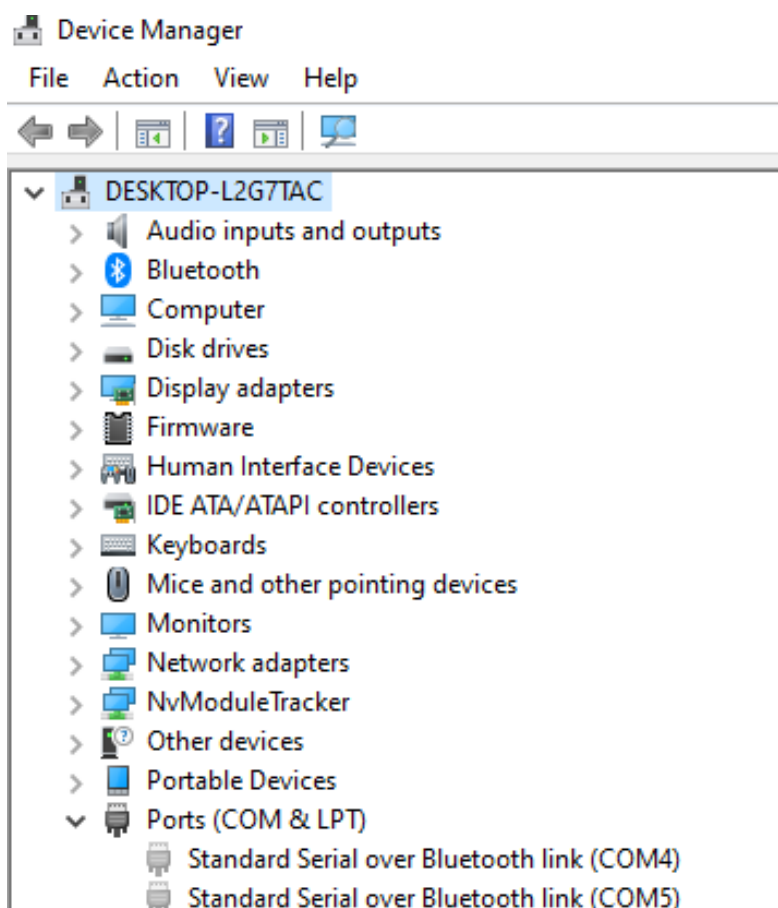
În folderul „build” se găsesc fișierele rezultate în urma compilării.

Prima setare pe care o vom face, va fi in **VisualStudioCode.bat** completând portul "COM" corespunzător plăcii noastre. Pentru asta vom deschide fisierul *.bat folosind NotePad.

```
27 @echo    }, >> settings.json
28 @echo    { >> settings.json
29 @echo        "cwd": "${workspaceFolder}", >> settings.json
30 @echo        "name": "Flash", >> settings.json
31 @echo        "color": "White", >> settings.json
32 @echo        "command": "idf.py -p COM1 flash", >> settings.json
33
```

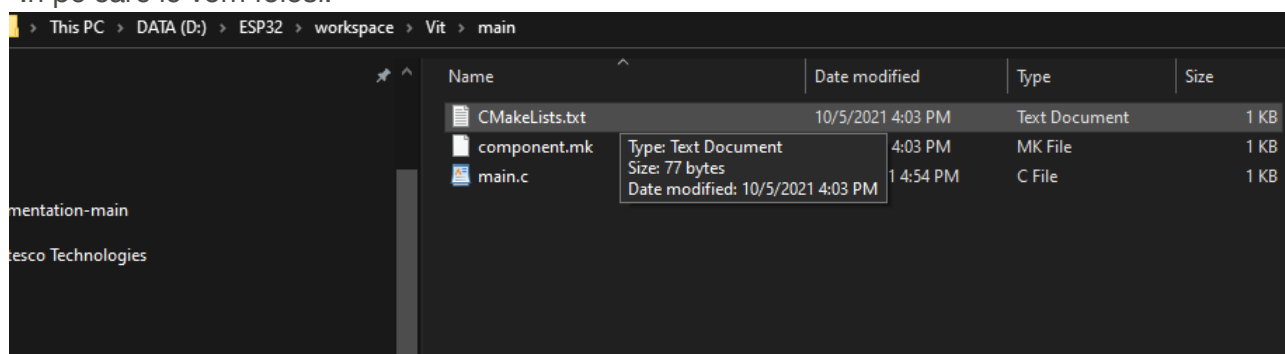
Pentru a gasit portul COM, unde se afla conectata placa vom urma pasii:

1. Deschidem **Device Manager** din **Start**
2. Click pe **View** din bara de meniuri și selectează **Show hidden devices**
3. Apoi selectăm **Ports** din listă.



Pentru crearea unui proiect nou vom rula **CreateBlankProject.bat**. Se va crea un proiect cu numele ales de noi la calea `d:\ESP32\workspace`.

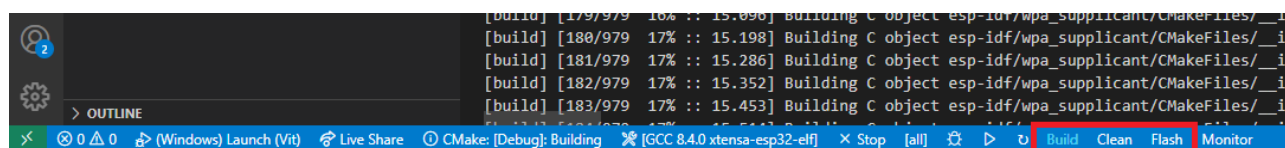
După crearea unui proiect nou, în fisierul **CMakeLists.txt** va trebui să adăugăm toate fișierele *.c și *.h pe care le vom folosi.



În cadrul proiectului nostru, vom utiliza o singură componentă, **VITAL** care conține folderele specifice fiecărui “strat” din cod și fișierele header corespunzătoare acestora. Alături de acestea, mai găsim un fișier **component.mk** în care se găsesc căile spre fișierele sursă pentru a putea fi recunoscute de compilator și o cale spre fișierul folosit ca pagină web: **index.html**.

Pentru a deschide IDE-ul vom rula **VisualStudioCode.bat**.

Pentru a putea compila corect proiectul, în meniul din stanga jos a IDE-ului găsim o listă de butoane care conțin comenzile pentru aceste acțiuni:



Butonul numit **Clean** va șterge fișierele compilate și butonul **Build** va face o compilare a acestora. În cazul în care nu avem erori de compilare, putem apăsa butonul **Flash** și codul va fi încărcat pe placa noastră de dezvoltare.

Ce este version control?

Version control presupune organizarea fișierelor proiectului la care lucrăm astfel încât să monitorizăm mai ușor modificările ce apar pe parcurs.

Există astfel două metode: manuală și automată.

- *Metoda manuală* - se crează copii ale unui fișer când începem un bloc nou de modificări. Acest procedeu poate să scape ușor de sub control și nu aduce o bună oraganizare pe termen lung.
- *Metoda automată* - modificările se rescriu în fișier dar se păstrează versiunile vechi pentru motive de erori.

Consola Git Bash

Git este un sistem de version control, open-source, care rulează pe majoritatea platformelor, inclusiv Linux, Windows și OS X.

Care sunt noțiunile care stau la baza Git-ului?

[Git](#) funcționează pe bază de **commituri**. Adică un set de modificări semnificative sau mai puțin semnificative ale unui sau mai multor fișiere, care de obicei au legătură sau funcționează în tandem. Deci fie că am modificat 3 linii sau 100 pe diferite fișiere, acestea se grupează în commituri. De regulă este recomandat să se păstreze o anumită frecvență și complexitate pentru a face commituri, spre exemplu când am implementat o nouă funcționalitate sau a trecut un număr de minute/ore.

O altă noțiune importantă – **branchurile** (ramuri). Ele sunt ramificații de la un proiect inițial cu scopul de a face o nouă copie, când avem de-a face cu o funcționalitate experimentală sau o secvență care nu suntem siguri dacă va funcționa. Astfel, prin a crea un branch, ne este mai ușor de a reveni la ceea ce funcționează în cazul unor erori masive. Dar nu numai, le putem folosi și pentru teste.

Merge – îmbinarea a două branch-uri într-unul. Ex: Funcționalitatea este reușită și trebuie adăugată la proiectul final. Prin merge putem concatena branch-ul principal (master) și cel al funcționalității.

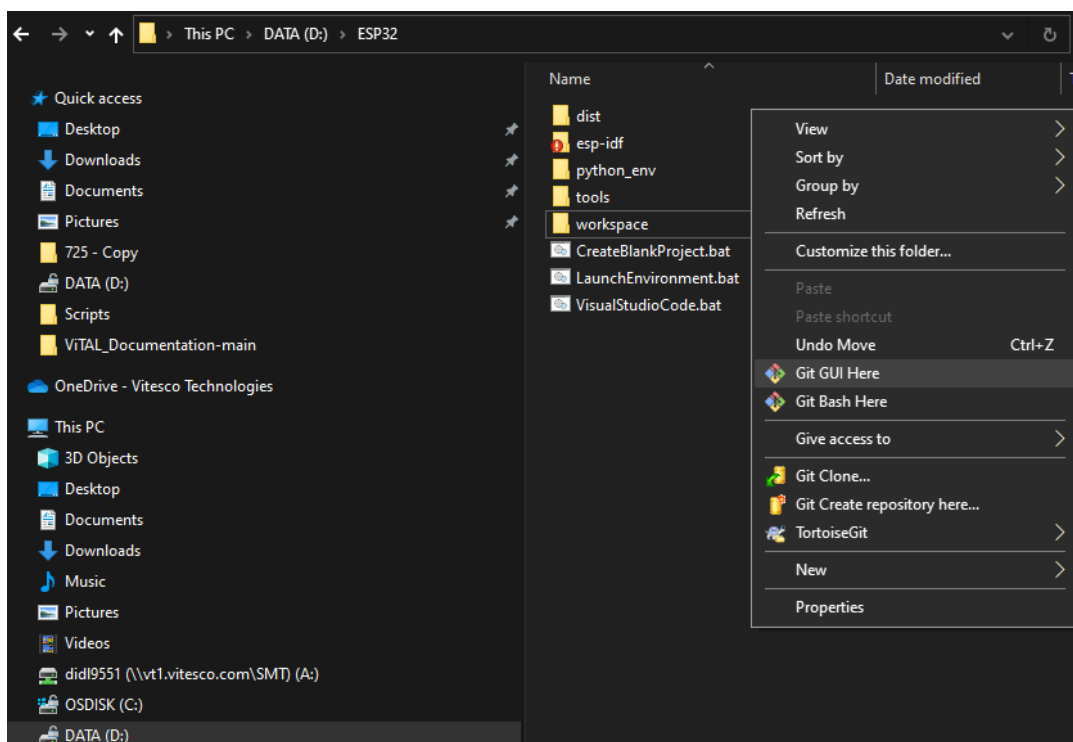
Cum utilizăm Git Bash?

Git Bash permite utilizarea funcționalităților Git prin apelarea comenzilor din consolă.

Pasul 1

Pentru inițializarea Git-ului într-un director, mai întâi trebuie să ajungem în directorul respectiv. Acest lucru se poate face în două moduri:

Navigarea folosind Windows Explorer până la directorul necesar și deschiderea acolo a consolei GitBash (click-dreapta, Git Bash Here).



Navigarea folosind consola Git Bash

În primul rând, se deschide aplicația Git Bash.

Pentru navigație se utilizează comanda `cd` urmată de locație

Ex: `cd D:/ESP32/workspace/Demo`

Când ne aflăm într-un director, dacă vrem să înaintăm în alt folder, apelăm comanda:

`cd <denumirea>`

Pentru a ne întoarce la folderul părinte, apelăm comanda

`cd ..`

Pentru a șterge comenzile anterioare (doar vizual) **`Ctrl-L`**

Pasul 2

Pentru crearea unui repository Git se utilizează comanda:

`git init`

care va crea un repository Git gol în directorul în care ne aflăm

Pasul 3

Pentru adăugarea unor fișiere noi, se utilizează comanda

`git add nume.format`

ex: `git add file.txt`

sau este posibilă utilizarea de *wildcard* (selectarea mai multor fișiere cu același format) prin comanda:

git add *.format


ex: git add *.c *.h (va adăuga toate fișierele cu extensia .c și .h)

Pentru a adăuga toate fișierele din folder se apelează comanda: **git add.**

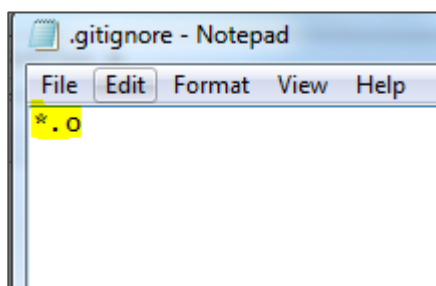
Obs: Uneori avem fișiere de o anumită extensie pe care nu dorim să le adăugăm deloc, spre exemplu fișierele obiect, cu extensia .o

Atunci, aflându-ne în folderul proiectului apelăm comanda: **touch.gitignore**

Această comandă crează un fișier cu extensia .gitignore

 .git	10/26/2018 4:44 PM	File folder
 Proiect	10/26/2018 4:09 PM	File folder
 .gitignore	10/29/2018 5:46 PM	Text Document

În acest fișier putem include extensia fișierelor pe care dorim să le ignorăm:



Acum, apelând comanda: **git status** (afișează starea curentă a proiectului) vom vedea că fișierele .o nu mai apar

Pasul 4

Pentru a da **commit** se apelează comanda:

git commit -m "Mesaj asociat acestui commit"

Utilizarea Branch-urilor

Pentru crearea unui **branch** nou se apelează comanda:

git branch <branch nou>

Pentru a afișa toate branch-urile existente:

git branch

Pentru a trece pe alt branch:

git checkout <numele>

Pentru a șterge un branch:

git branch -d <numele>

Observație:

Dacă ne aflăm pe un branch și adăugăm un fișier în „staging”(comanda: **git add**), iar apoi, fără a da commit, trecem pe alt branch, vom observa că acest fișier încă persistă și ar putea crea probleme mai apoi.

Respectiv folosim comandă **git stash** pentru a trece aceste schimbări într-o fază în care ele nu vor influența munca noastră de pe alt branch. Când suntem gata să continuăm munca pe branch-ul inițial, apelăm comanda: **git stash apply** , pentru a recupera acele modificări nefinisate.

Până acum am vorbit doar de manipularea locală a proiectului, dar pentru ca mai multe persoane să aibă acces la proiect, acesta trebuie urcat pe un server apelând comanda:

git push specificând:

1)Unde vrem să dăm upload (link-ul)

2)Care branch

Pentru a nu fi nevoiți să introducem de fiecare dată link-ul, putem să îl memorăm:

git remote add <RepoName> <url>

ex: *git remote add myRepo https://github.com/ViTAL_Laboratory/ViTAL_lab.git*

Acum, când introducem **myRepo**, acesta va fi înlocuit cu link-ul de mai sus.

Pentru a da push branch-ului *master* apelăm comanda:

git push myRepo master

Pentru a descărca un proiect aflat pe server(integral sau parțial) folosim:

a) git clone <RepoName>

Obținem o copie integrală a proiectului (inclusiv folderul .git care conține toată istoria proiectului)

b) git fetch <RepoName>

Selectează și descarcă doar modificările făcute la proiect (diferențele dintre local și remote)

c) git pull <RepoName>

Este asemănător cu *fetch*, doar că în același timp realizează un *merge* între versiunea descarcată și cea locală

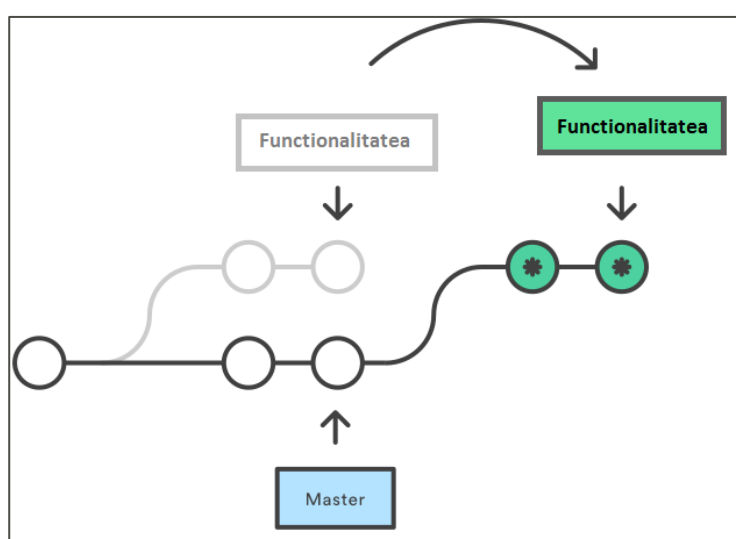
Uneori avem nevoie să anulăm anumite commit-uri (spre exemplu când nu am reușit să implementăm o funcționalitate).

Pentru aceasta se folosește comanda: **git reset**

Resetul poate fi de trei tipuri:

- 1) Soft reset – este analogic cu un **amend**. Modificările făcute sunt trecute în staged
- 2) Mixed reset – modificările făcute trec în unstaged
- 3) Hard reset – modificările făcute sunt înlăturate definitiv

O altă modalitate de a obține rezultatul unui **merge** este **rebase**.



Printr-o serie de commit-uri efectuate automat se obține o liniarizare a istoricului proiectului.

Pentru aceasta folosim comanda **git rebase** specificând:

- a) branch-ul pe care ne aflăm.
- b) branch-ul funcționalității

Ex: git rebase Master Funcționalitatea