Opdrachten kwartaal 3 week 2 – Unittesten en Git submodules

In week 1 heb je testsoftware geschreven met behulp van een eenvoudige macro. In deze les leer je om gebruik te maken van een eenvoudig test-framework. Er zijn vele commerciële en gratis test-frameworks beschikbaar. Wij maken in deze cursus gebruik van GoogleTest. In het eerste en tweede jaar heb je al gebruik gemaakt van Git om de software die je hebt geschreven te beheren. Ook in deze cursus ga je gebruik maken van Git. Je maakt daarbij gebruik van Git submodules om een C module zowel onder WSL als in Code Composer Studio te kunnen gebruiken.

Je leert deze les hoe je:

- C-code systematisch kan testen met behulp van het eenvoudige test-framework GoogleTest;
- een ontwikkelmethode voor software genaamd Test Driven Development (TDD) kan toepassen;
- code die je in meerdere applicaties wilt gebruiken in een Git submodule kan plaatsen;
- zo'n submodule kunt gebruiken onder Windows Subsystem for Linux;
- zo'n submodule kunt gebruiken in Code Composer Studio.

In de introductie van deze les is gedemonstreerd hoe je GoogleTest en de Visual Studio Code extensie C++ TestMate kunt gebruiken. Tevens is uitgelegt hoe de Test Driven Development (TDD)[1] softwareontwikkelmethode toegepast moet worden.

In deze les ga je eerst de in de introductie gedemonstreerde code zelf uitvoeren en daarna ga je deze code uitbreiden volgens de TDD-methode.

3.2.1 Start Visual Studio Code en druk op F1 om het commando venster te openen. Type "WSL" en kies voor "WSL: Connect to WSL in New Window".

Installeer de extensie C++ TestMate[2].

Open een terminal window door op ctrl + ‘ te drukken. Installeer het GoogleTest framework met het volgende commando[3]:

```
sudo pacman -Sy gtest
```

[1] Zie https://en.wikipedia.org/wiki/Test-driven_development.

[2] Zie https://marketplace.visualstudio.com/items?itemName=matepek.vscode-catch2-test-adapter.

[3] Het "[sudo] password for ems" is : ems.

Ga naar het directory `Opdrachten` door in het terminal window in te typen:

```
cd Opdrachten
```

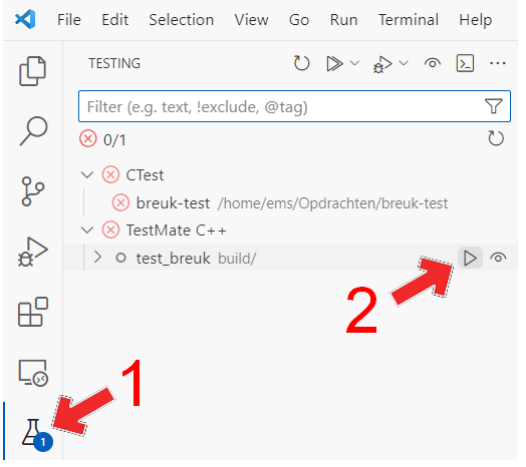
Download het bestand `breuk-test.zip`, pak het uit en verwijder het weer met de volgende commando's:

```
wget https://bitbucket.org/HR_ELEKTRO/ems31/raw/master/Program-↩
 ↪ mas/breuk-test.zip
unzip breuk-test.zip
rm breuk-test.zip
```

Voer vervolgens onderstaand commando uit om dit directory `breuk-test` te openen in Visual Studio Code en CMake (automatisch) uit te voeren.

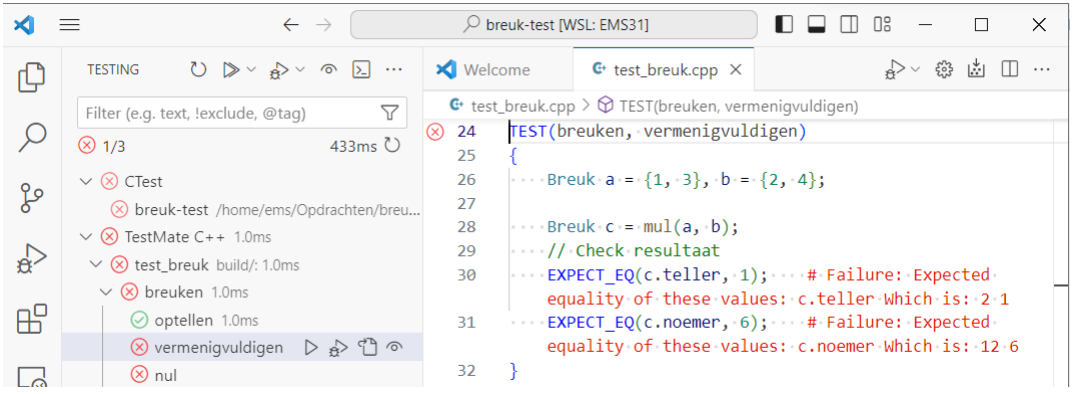
```
code -r breuk-test
```

Build het project door op de Build-knop te klikken of door op F7 te drukken. Je kunt nu de testen uitvoeren door in de statusbalk op de Play-knop te klikken. Maar het is handiger om gebruik te maken van de TestMate-plugin, zie figuur 1.



Figuur 1: Testen uitvoeren met behulp van de TestMate-plugin.

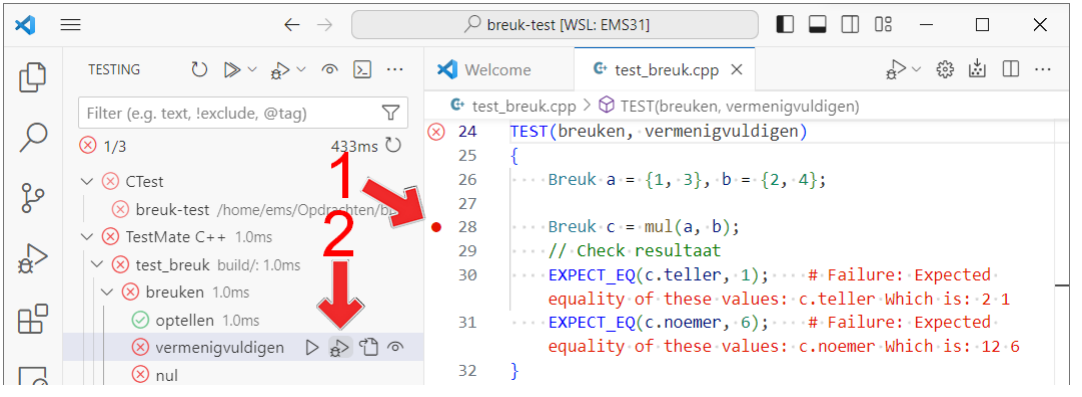
Als je op het >-teken voor `test_breuk` in het TESTING-window klikt, dan zie je het resultaat van de verschillende testen, zie figuur 2.



Figuur 2: Resultaat van de testen.

Als je dubbelklikt op de test `vermenigvuldigen` in het TESTING-window, dan zie je meteen waarom de test voor vermenigvuldigen niet geslaagd is. De test voor het vermenigvuldigen is niet geslaagd omdat de functie `mul` een `Breuk` met de waarde `2/12` teruggeeft terwijl de verwachte waarde `1/6` is.

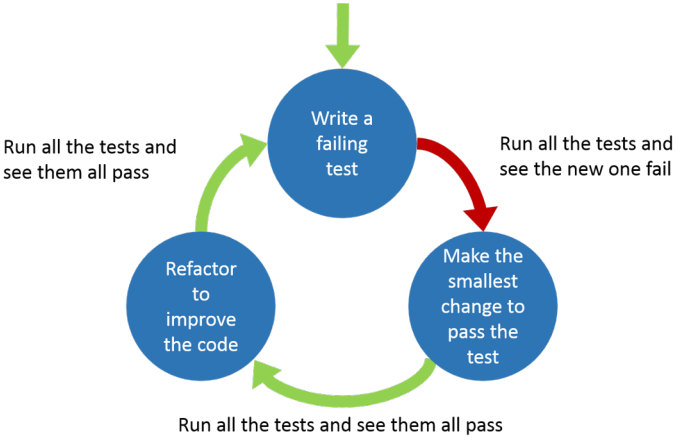
Je kunt de code van deze testfunctie eenvoudig debuggen door een breakpoint te plaatsen op de regel waar de functie `mul` wordt aangeroepen en op het debug-knopje achter de test te drukken, zie figuur 3.



Figuur 3: Debuggen van een test.

Je ziet dat de programmeur die de functie `mul` heeft geschreven vergeten is om het resultaat te normaliseren. Pas de code in `breuk.c` aan zodat het resultaat van de vermenigvuldiging wel genormaliseerd wordt, build het project (druk op F7) en voer de testen opnieuw uit. Als het goed is zijn nu alle testen geslaagd.

Je gaat nu de functionaliteit van module `breuk` uitbreiden met een functie `sub` waarmee je het verschil tussen twee breuken kunt bepalen. Daarbij ga je gebruik maken van de TDD-methode, zie figuur 4. Hieronder wordt stap voor stap uitgelegd hoe je deze methode kunt toepassen.



Figuur 4: Test Driven Development.

3.2.2

A Voeg in het bestand `test_breuk.cpp` een test toe waarin je de functie `sub` aanroept met de waarden `5/12` en `1/4` en controleer of het resultaat gelijk is aan `1/6`. Als je het project nu build krijg je natuurlijk een compilatiefout:

```
error: 'sub' was not declared in this scope
```

B Voeg in het bestand `breuk.h` de functie `sub` toe:

```
Breuk sub(Breuk a, Breuk b);
```

Voeg in het bestand `breuk.c` de meest eenvoudige implementatie van deze functie toe:

```
Breuk sub(Breuk a, Breuk b)
{
    Breuk res = {1, 6};
    return res;
}
```

Dit lijkt misschien zinloos (want je weet dat deze implementatie onjuist is), maar het is wel een goede manier om te controleren of de test die je in de vorige stap hebt geschreven goed werkt.

Build het project en voer de testen uit. Als het goed is zijn nu alle testen geslaagd.

C Voeg nog een test toe om te testen of `1/4` minus `5/12` gelijk is aan `-1/6`. Je kunt deze test toevoegen aan de al bestaande `TEST` voor aftrekken of je kunt een nieuwe `TEST` aanmaken. Beide manieren zijn goed.

Build het project en voer de testen uit. Als het goed is zal de laatst toegevoegde test nu falen.

D De implementatie van de functie `sub` is natuurlijk niet correct. De volgende stap is om de implementatie van de functie `sub` te verbeteren. Bedenk dat de functie `sub` gebruik kan maken van de functie `add` die al geschreven is. Er geldt namelijk dat: $a - b = a + (-b)$.

Implementeer de functie `sub` opnieuw, maar nu op een correcte manier. Build het project en voer de testen uit. Als het goed is zijn nu alle testen geslaagd.

E Je zou nu tevreden kunnen zijn met de code die je hebt geschreven. Maar misschien is het goed om nog een aantal testen te schrijven. Voeg nog een aantal testen toe (eventueel in de `TEST nul`) om te testen of voor `Breuk` a geldt: $a - a = 0$, $a - 0 = a$ en $0 - a = -a$.

Build het project en voer de testen uit. Zorg er voor dat alle testen slagen.

3.2.3 Breid de functionaliteit van module `breuk` nu uit met een functie `divide`[4] waarmee je $a \div b$ (`Breuk` a gedeeld door `Breuk` b) kunt berekenen. Gebruik de TDD-methode om deze functie te implementeren. Voer de stappen die je hebt uitgevoerd bij opdracht 3.2.2, deelopdrachten A tot en met E opnieuw uit maar nu voor `divide` in plaats van voor `sub`. Bedenk, bij deelopdracht D, dat $\frac{p}{q} \div \frac{r}{s} = \frac{p}{q} \times \frac{s}{r}$. Bedenk bij deelopdracht E dat voor `Breuk` a geldt: $a \div a = 1$, $0 \div a = 0$ en $a \div 0 =$ niet toegestaan. Zie ook de volgende opdracht.

3.2.4 Delen door nul is natuurlijk, ook voor berekeningen met breuken, niet toegestaan. Als je getest hebt wat er gebeurt als je dit toch probeert, dan zul je zien dat de `assert` functie die op de eerste regel van de functie `normaliseer` staat het programma afbreekt, als je het programma uitvoert met de run-knop in de statusbalk.

[4] We kunnen de naam `div` niet gebruiken omdat deze functienaam al gebruikt wordt in `stdlib.h`, zie eventueel https://en.cppreference.com/w/c/numeric/math/div.

Het programma eindigt dan met de melding:

```
test_breuk: /home/ems/Opdrachten/breuk-test/breuk.c:21: ↩
 ↪ normaliseer: Assertion `b.noemer != 0' failed.
Aborted
```

De functie `assert` is een goede manier om het programma netjes af te breken als er een situatie optreedt die niet mag voorkomen (zoals delen door nul).

Als je de testen uitvoert met de run knop in het TESTING-window, zie figuur 3, dan wordt het programma niet afgebroken na de assert, maar gewoon vervolgt waardoor een harde fout optreedt, zie figuur 5.

Figuur 5: Delen door nul.

Met behulp van het GoogleTest framework kun je ook testen of het programma netjes wordt afgebroken als er gedeeld wordt door nul. Voeg hiervoor de volgende test[5] toe:

```
EXPECT_DEATH(divide(b, a), "b.noemer != 0");
```

Build het project en voer de testen uit. Als het goed is zijn nu alle testen geslaagd.

In het tweede jaar heb je al gebruik gemaakt van Git om de software die je hebt geschreven te beheren. In deze cursus ga je ook gebruik maken van Git. Je krijgt daartoe drie Git repositories toegewezen door de docenten:

- Een repository voor de code die op de pc onder Windows Subsystem for Linux ontwikkeld wordt (`ems31_2023-2024_groep_XX_wsl`[6]).

[5] Zie eventueel https://github.com/google/googletest/blob/main/docs/advanced.md#death-tests.

[6] In plaats van `XX` staat het nummer van je groep.

- Een repository voor de code die op in Code Composer Studio ontwikkeld wordt (`ems31_2023-2024_groep_XX_ccs`).
- Een repository voor de modules (bijvoorbeeld buffer) die zowel onder WSL op de pc als in CCS gebruikt worden (`ems31_2023-2024_groep_XX_submodules`). Dit laatste repository kan dan als submodule in de andere twee repositories gebruikt worden.

In de volgende opdrachten ga je de code voor de buffer die maximaal 8 elementen kan bevatten die je vorige week hebt geschreven in een Git submodule plaatsen. Vervolgens ga je deze code testen met behulp van GoogleTest onder WSL. Deze testcode wordt in een aparte Git repository geplaatst.

3.2.5 Open een terminal window in WSL en voer de volgende handleiding uit om Git te configureren: https://bitbucket.org/HR_ELEKTRO/ems31/wiki/Software_WSL_git.md. Als je dit al gedaan hebt bij het installeren van WSL, dan kun je deze stap overslaan.

3.2.6 Open je WSL home directory door het volgende commando in te typen in het terminal window:

```
code -r ~
```

Clone nu het repository `ems31_2023-2024_groep_XX_wsl`[7] in je home directory met behulp van het commando:

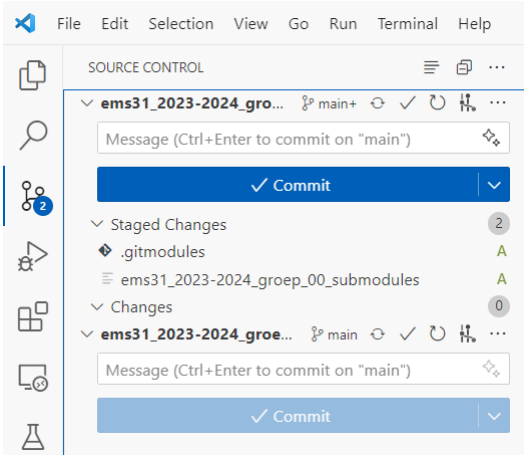
```
git clone git@bitbucket.org:HR_ELE_STUD/ems31_2023-2024_groep_-↩
 ↪ XX_wsl.git
```

Ga naar het gecloonde repository en voeg daar een submodule toe met behulp van de commando's:

```
cd ems31_2023-2024_groep_XX_wsl
git submodule add git@bitbucket.org:HR_ELE_STUD/ems31_2023-20-↩
 ↪ 24_groep_XX_submodules.git
```

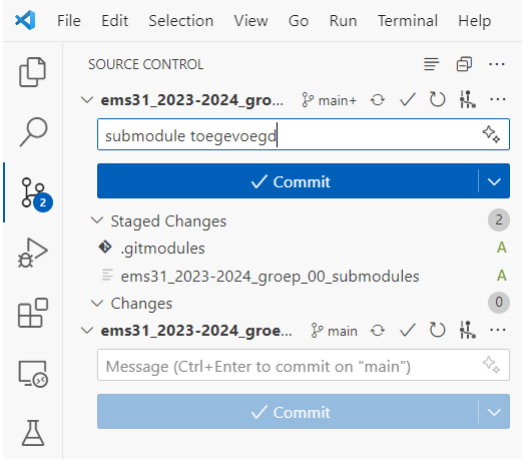
Open nu het Source Control window in Visual Studio Code (druk op ctrl + ⇧ + G), zie figuur 6.

[7] Vul in plaats van `XX` het nummer van je groep in.



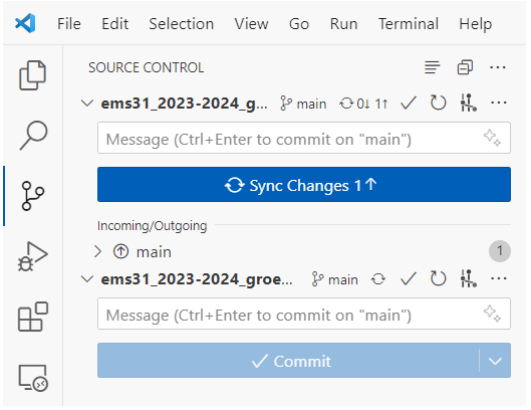
Figuur 6: Source Control window.

Commit nu de wijzigingen die je hebt aangebracht in het repository `ems31_2023-20-24_groep_XX_wsl`, zie figuur 7.



Figuur 7: Commit de wijzigingen.

Push de wijzigingen naar de remote repository, zie figuur 8.

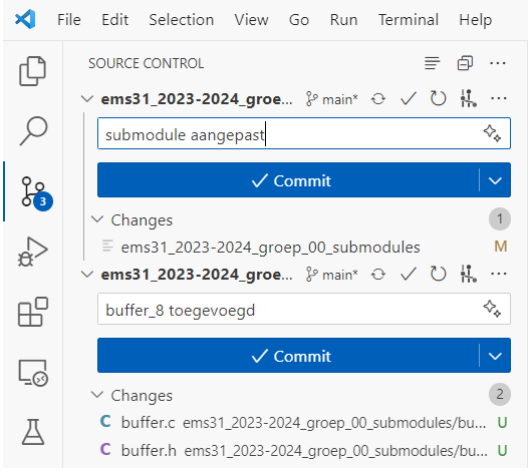


Figuur 8: Push de wijzigingen.

Ga nu naar het directory `ems31_2023-2024_groep_XX_submodules`, maak daar een nieuw directory genaamd `buffer_8` aan en kopieer daar de vorige week geschreven bestanden `buffer.h` en `buffer.c` naar toe:

```
cd ems31_2023-2024_groep_XX_submodules
mkdir buffer_8
cp -v ~/Opdrachten/buffer_8/buffer.* buffer_8
```

De `-v` optie (verbose) bij het cp-commando zorgt ervoor dat de bestanden die gekopieerd worden op het scherm getoond worden. Commit de wijzigingen die je hebt aangebracht in beide repositories, zie figuur 9.



Figuur 9: Commit de wijzigingen in beide repositories.

Push de wijzigingen naar de remote repositories.

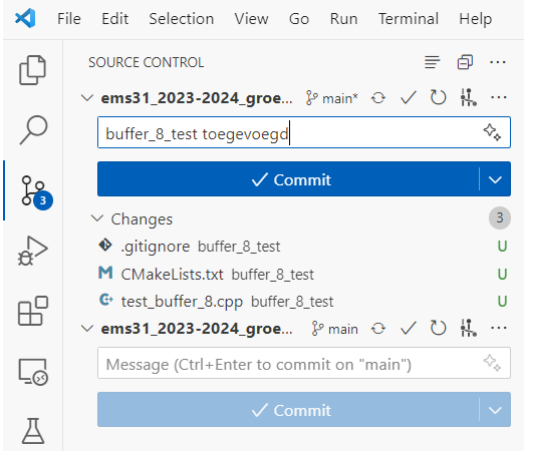
Ga terug naar het directory `ems31_2023-2024_groep_XX_wsl` en voer daar de volgende commando's uit:

```
wget https://bitbucket.org/HR_ELEKTRO/ems31/raw/master/Program-↩
 ↪ mas/buffer_8_test.zip
unzip buffer_8_test.zip
rm buffer_8_test.zip
```

Ga naar het directory `buffer_8_test` en open het bestand `CMakeLists.txt`:

```
cd buffer_8_test
code CMakeLists.txt
```

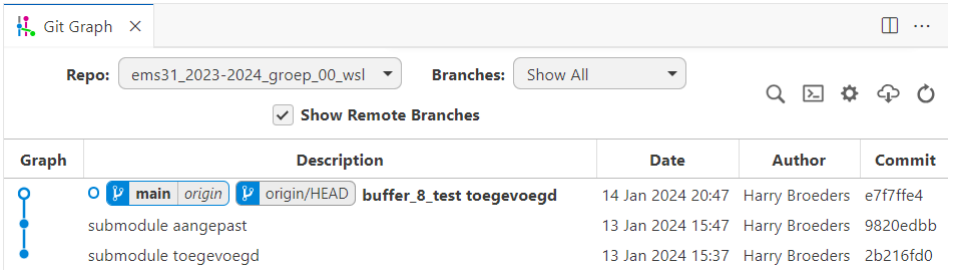
Pas het path naar het bestand `buffer.c` door `XX` te vervangen met je groepsnummer. Open het bestand `test_buffer.c` en pas de regel met de **`#include`** aan door `XX` te vervangen met je groepsnummer. Sluit beide bestanden en commit de wijzigingen, zie figuur 10.



Figuur 10: Commit de wijzigingen.

Push de wijzigingen naar de remote repository.

Met behulp van de Extensie Git Graph kun je de Git geschiedenis van je repository bekijken, zie figuur 11.



Figuur 11: Git Graph.

Open nu het directory `buffer_8_test` in Visual Studio Code zodat CMake uitgevoerd wordt (let op, het commando eindigt met een spatie en een punt[8]):

```
code -r .
```

Build het project en voer de testen uit. Verbeter de code van `buffer_8.c` als niet alle testen slagen.

3.2.7 Breid de module `buffer_8` uit met een functie om het aantal gevulde elementen in de buffer op te vragen:

```
unsigned int number_of_elements_in_buffer();
```

Schrijf *eerst* de testcode en dan pas de implementatie (conform TDD).

Commit en push de wijzigingen naar de remote repositories.

De code voor de buffer die ontwikkeld is op een pc kan nu ook gebruikt worden op een embedded systeem. In het laatste deel van deze les ga je de module `buffer_8` gebruiken in een project dat in Code Composer Studio is ontwikkeld voor een CC3220S LaunchPad. We gaan ervan uit dat je beschikt over dit ontwikkelbord dat je ook al bij EMS20 hebt gebruikt[9].

Om software te compileren voor dit bord gebruiken je Code Composer Studio (CCS), waar je ook bij EMS20 al mee hebt gewerkt. Een installatiehandleiding voor CCS vind je op: https://bitbucket.org/HR_ELEKTRO/ems20/wiki/installatie_software.md. Je kunt het installatiebestand voor CCS versie 12.5 downloaden van https://dr-download.ti.com/software-development/ide-configuration-compiler-or-debugger/MD-J1VdearkvK/12.5.0/CCS12.

[8] De punt is een afkorting voor de huidige directory.

[9] Indien nodig kun je een CC3220S LaunchPad aanschaffen bij de winkel op Academie Plein.

5.0.00007_win64.zip. Daarnaast heb je ook de SimpleLink CC32XX SDK versie 6.10 nodig. Het installatiebestand voor deze SDK kun je downloaden van https://bitbucket.org/HR_ELEKTRO/ems31/downloads/simplelink_cc32xx_sdk_6_10_00_05.exe. De installatie wijst zichzelf.

Als voorbeeld is een applicatie voor de CC3220S LaunchPad beschikbaar genaamd `temperatuur_logger`. Deze applicatie bevat onder andere de volgende twee pthreads:

- een producer die elke seconde de temperatuur (uitgedrukt in tiende graden Celsius) inleest en die wegschrijft in de buffer;
- een consumer die wacht op een binnenkomend UDP-pakket. Als dit pakket de C-string `"temp?"` bevat, dan wordt een temperatuur uit de buffer gelezen en wordt deze waarde (uitgedrukt in tiende graden Celsius), gecodeerd als een C-string, teruggestuurd via UDP naar de afzender. Als op het moment dat dit UDP-pakket binnenkomt geen waarde in de buffer aanwezig is dan wordt een halve seconde gewacht en wordt opnieuw gekeken of er al een waarde in de buffer aanwezig is. Dit wordt net zo lang herhaald totdat een waarde uit de buffer is gelezen. Als het ontvangen UDP-pakket niet de C-string `"temp?"` bevat, dan wordt de C-string `"?"`, teruggestuurd via UDP naar de afzender.

Om UDP-berichten te kunnen versturen van de CC3220S LaunchPad naar een pc is een wifi-router nodig waarmee beide systemen kunnen verbinden. Je kunt gebruik maken van het wifi-netwerk bij je thuis of van een wifi-hotspot op je telefoon. De UDP-paketten kunnen op de pc ontvangen en zichtbaar gemaakt worden met het programma `ontvang_temperaturen.py`. Om dit programma te kunnen draaien gebruik je Thonny, zie https://thonny.org/. De installatie wijst zichzelf.

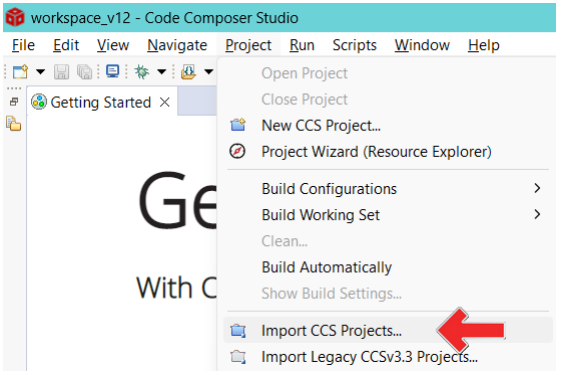
In de volgende opdracht ga je de applicatie `temperatuur_logger` compileren en uitvoeren op de CC3220S LaunchPad.

3.2.8

A Download het bestand temperatuur_logger_6_10.zip[10].

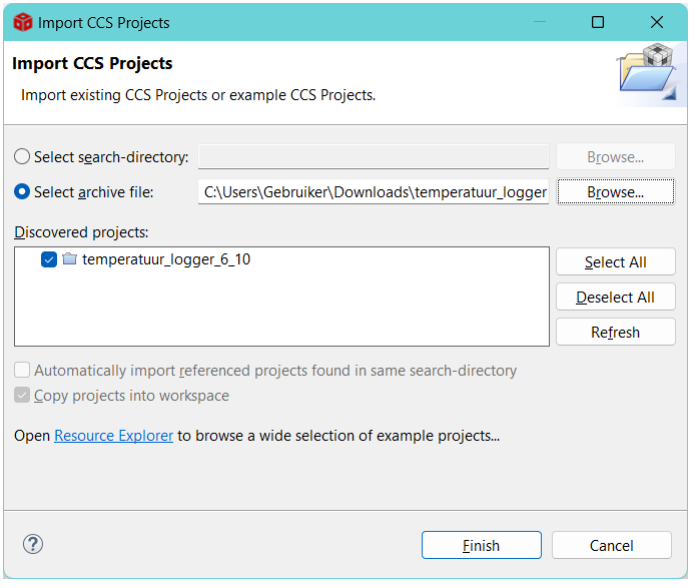
B Start Code Composer Studio en open het menu Project › Import CCS Projects..., zie figuur 12.

[10] https://bitbucket.org/HR_ELEKTRO/ems31/raw/master/Opdrachten/progs/temperatuur_logger/temperatuur_logger_6_10.zip



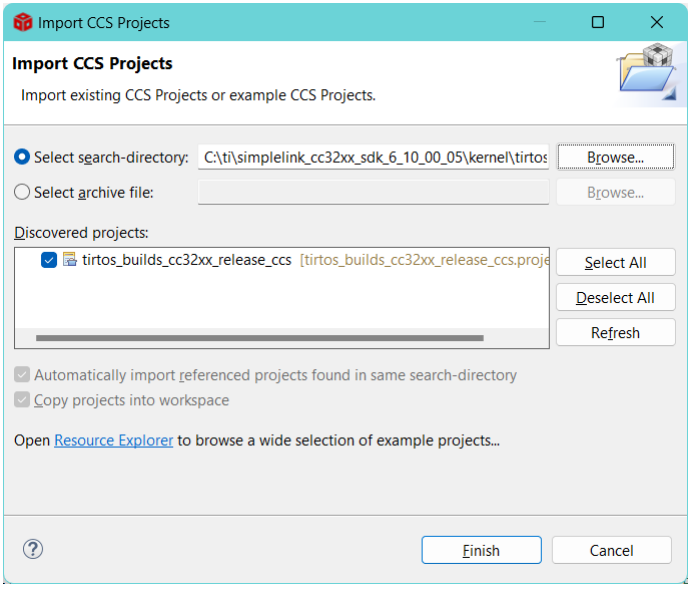
Figuur 12: Open het menu Project > Import CCS Projects...

C Kies "Select archive file" en selecteer het bestand `temperatuur_logger_6_10.zip` en klik op Finish, zie figuur 13.



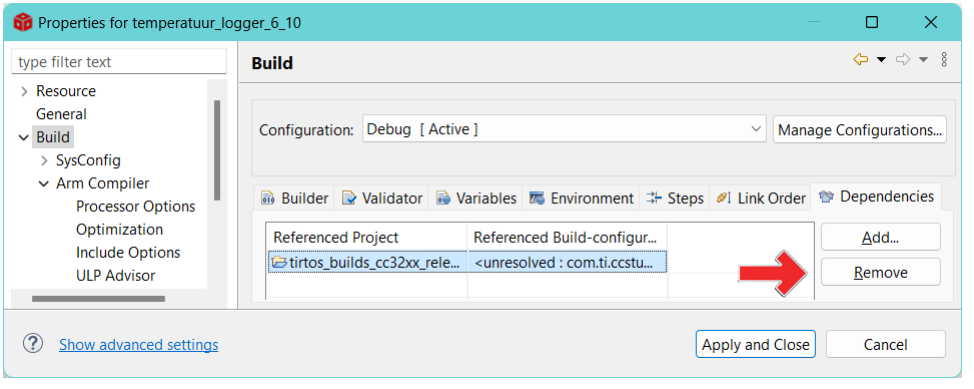
Figuur 13: Select archive file `temperatuur_logger_6_10.zip`

D Als je het project genaamd `tirtos_builds_cc32xx_release_ccs` nog niet in jouw workspace hebt geïmporteerd, dan moet je dit project importeren vanuit de directory `...\ti\simplelink_cc32xx_sdk_...\kernel\tirtos\builds\cc32xx\release\ccs`, zie figuur 14.

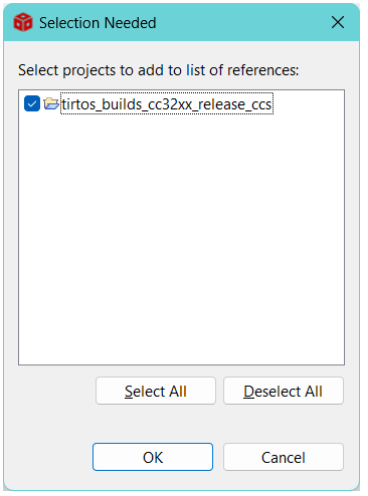


Figuur 14: Importeer het project `tirtos_builds_cc32xx_release_ccs`.

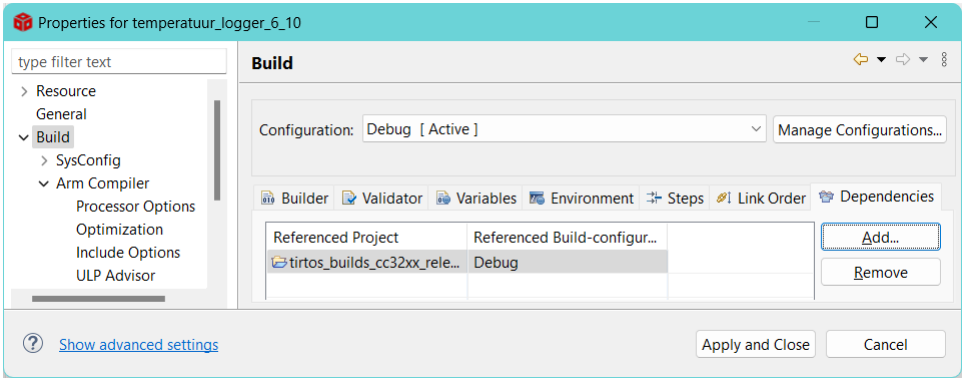
Pas nu de *Properties* van het project `temperatuur_logger` als volgt aan: ga naar Build > Dependencies, verwijder het *unresolved* project, zie figuur 15 en voeg nu het geïmporteerde project toe door op Add.. te drukken en `tirtos_builds_cc32xx_release_ccs` te selecteren, zie figuren 16 en 17.



Figuur 15: verwijder het *unresolved* project.



Figuur 16: Selecteer `tirtos_builds_cc32xx_release_ccs`.

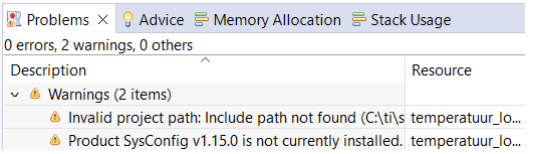


Figuur 17: Klik op Apply and Close.

E Open `main.c` en vul op regel 51 en 53 de ontbrekende informatie in:

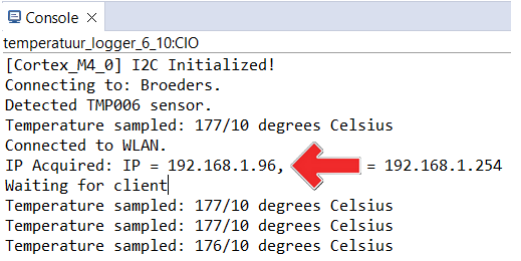
- naam van de wifi-gateway (`SSID_NAME`);
- wachtwoord van de wifi-gateway (`SECURITY_KEY`).

F Build het project `temperatuur_logger_6_10`. Eventuele warnings kun je negeren, zie figuur 18.



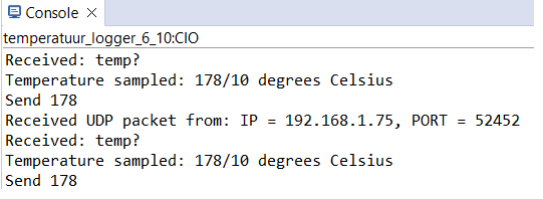
Figuur 18: Eventuele warnings kun je negeren.

G Start de debugger en voer het project uit. Kijk welk IP-adres de CC3220S LaunchPad heeft gekregen, zie figuur 19.

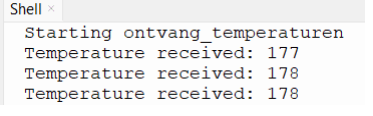


Figuur 19: Noteer het IP-adres dat de CC3220S LaunchPad heeft gekregen.

Open het programma `ontvang_temperaturen.py` in Thonny en vul op regel 7 het IP-adres van de CC3220S LaunchPad in (`IP_temperatuur_logger`). Zorg dat de pc verbonden is met hetzelfde LAN als de CC3220S LaunchPad en start het Python-programma op de pc. Als het goed is zie je nu dat de CC3220S LaunchPad elke seconde een temperatuurwaarde naar de pc stuurt, zie figuren 20 en 21.



Figuur 20: De CC3220S LaunchPad verstuurd elke seconde een temperatuurwaarde.



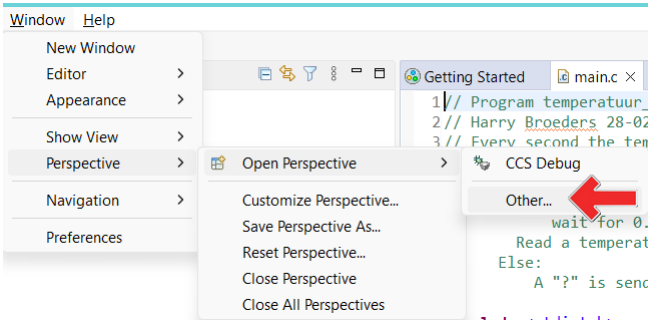
Figuur 21: Deze temperatuurwaarde wordt op de pc ontvangen met behulp van een Python programma.

H Wat gebeurt er als het Python-programma gestopt en weer opnieuw opgestart wordt? Gaat er data verloren?[11]

We gaan het project `temperatuur_logger_6_10` nu opslaan in het Git repository `ems31_20-23-2024_groep_XX_ccs`[12].

3.2.9

A Open het menu Window > Perspective > Open Perspective > Other..., zie figuur 22.

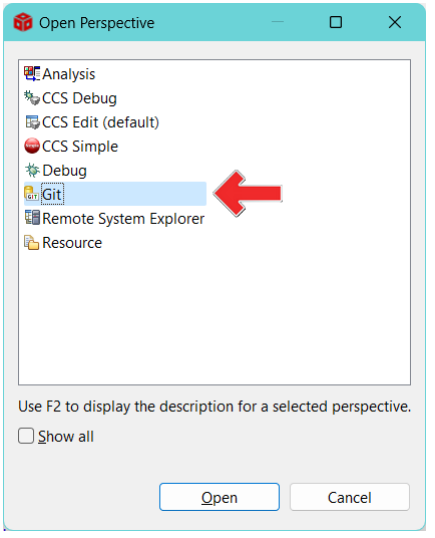


Figuur 22: Open het menu Window > Perspective > Open Perspective > Other...

B Kies “Git” en klik op Open, zie figuur 23.

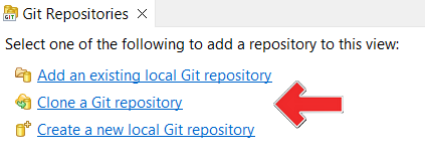
[11] Als je in `buffer.c` kijkt, dan zie je dat er slechts één waarde in dit buffer opgeslagen kan worden. Het is dus niet zo vreemd dat er data verloren gaat.

[12] Vervang `XX` door het nummer van je groep.



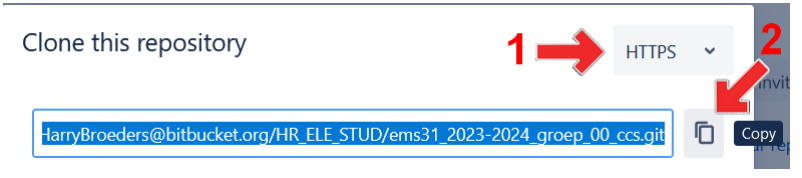
Figuur 23: Open het Git perspective.

C Klik op de link “Clone a Git repository”, zie figuur 24.



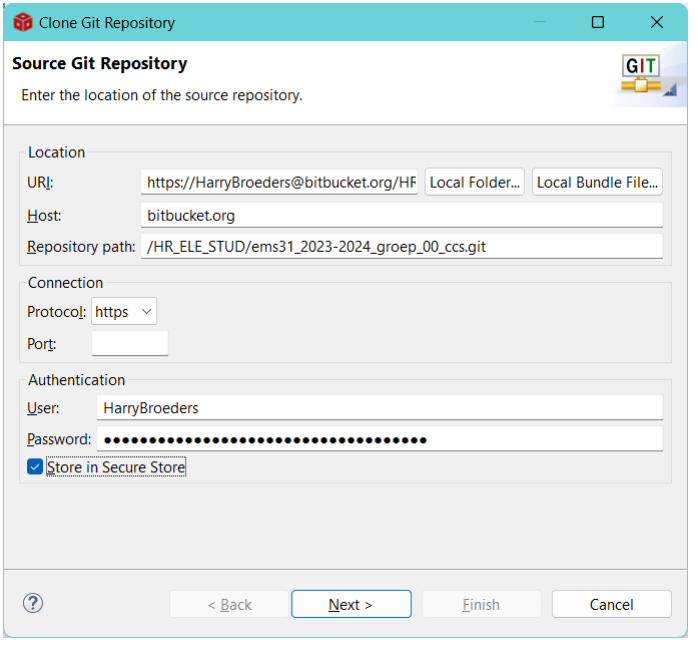
Figuur 24: Clone a Git repository.

D Ga in je internet browser naar `https://bitbucket.org/` en log in. Ga naar het repository `ems31_2023-2024_groep_XX_ccs` en klik op Clone. Selecteer “HTTPS” en kopieer het git clone commando voor dit repository, zie figuur 25.



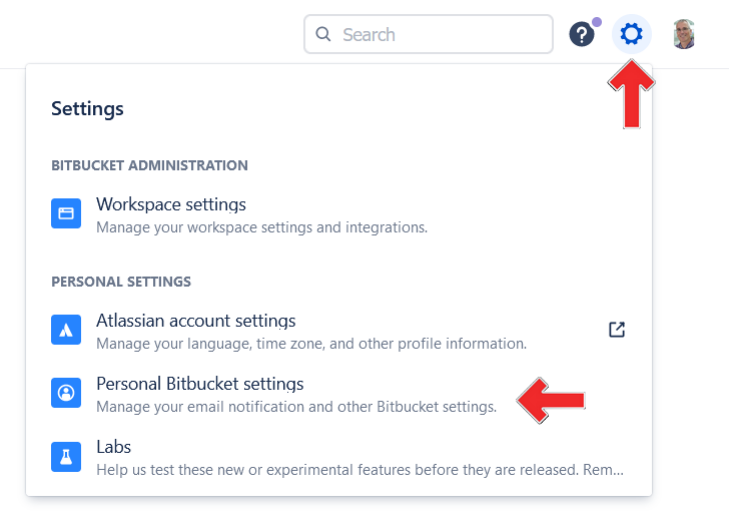
Figuur 25: Kopieer het git clone commando voor dit repository.

E Plak dit in het “URI” veld in CCS, zie figuur 26. Bijna alle velden (behalve “Password”) worden nu automatisch ingevuld.



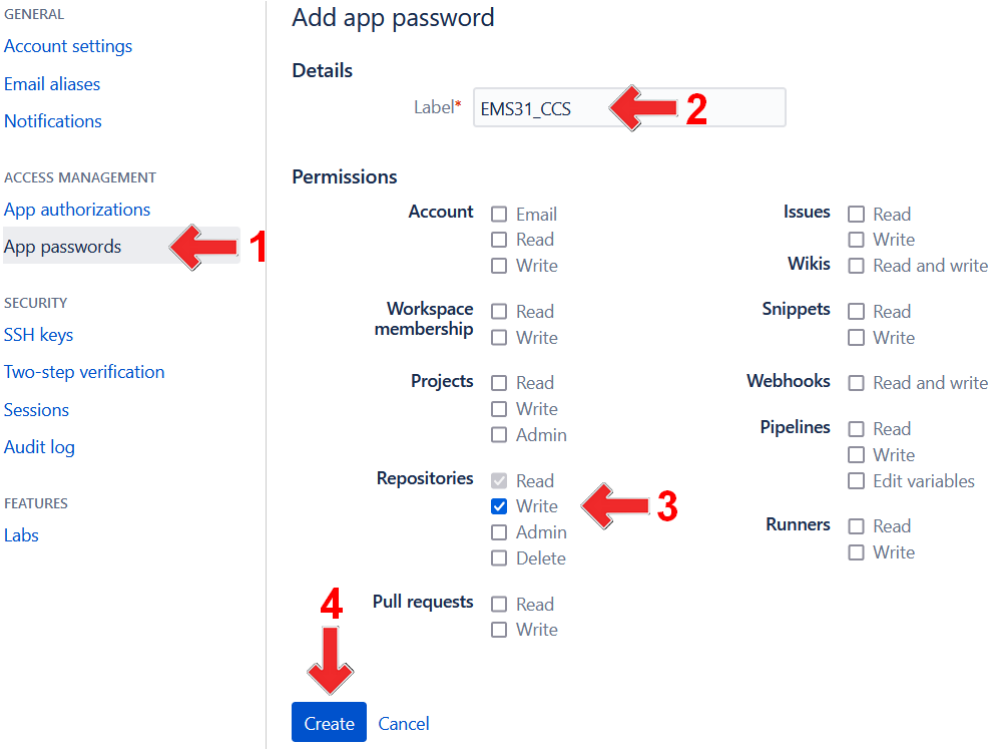
Figuur 26: Plak het git clone commando in het Clone Git Repository window in CCS.

F Om het Git repository te kunnen clonen moet je eerst een app password aanmaken. Ga weer naar je internet browser en klik op het tandwiel rechtsboven en kies voor "Personal Bitbucket settings", zie figuur 27.



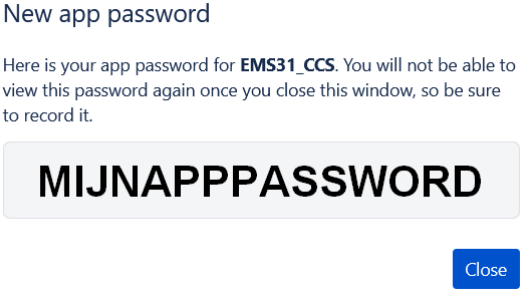
Figuur 27: Personal Bitbucket settings aanpassen.

G Klik op App passwords, vul een label in, selecteer “Write” permissions voor repositories en klik op Create, zie figuur 28.



Figuur 28: Creëer een app password.

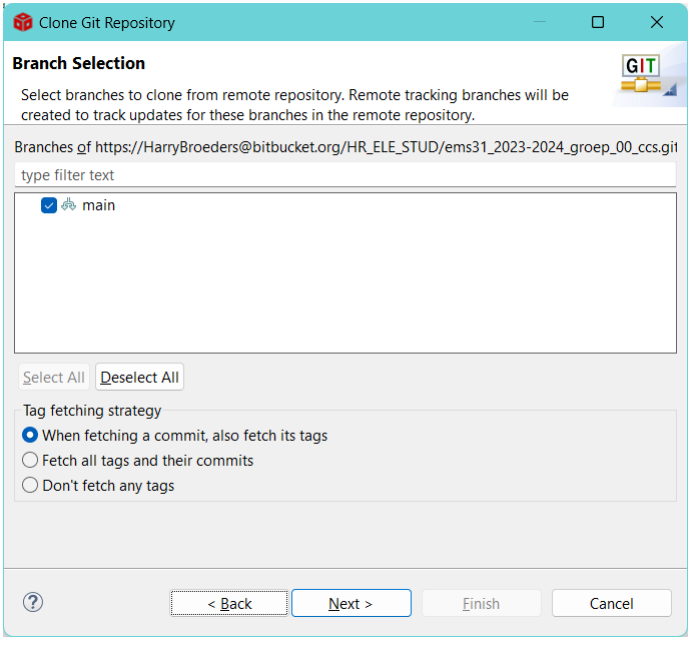
H Het password verschijnt nu in een nieuw window, zie figuur 29. Kopieer dit password.



Figuur 29: Het aangemaakte app password.

I Kopieer dit wachtwoord naar het “Password” veld in CCS, zie figuur 26.

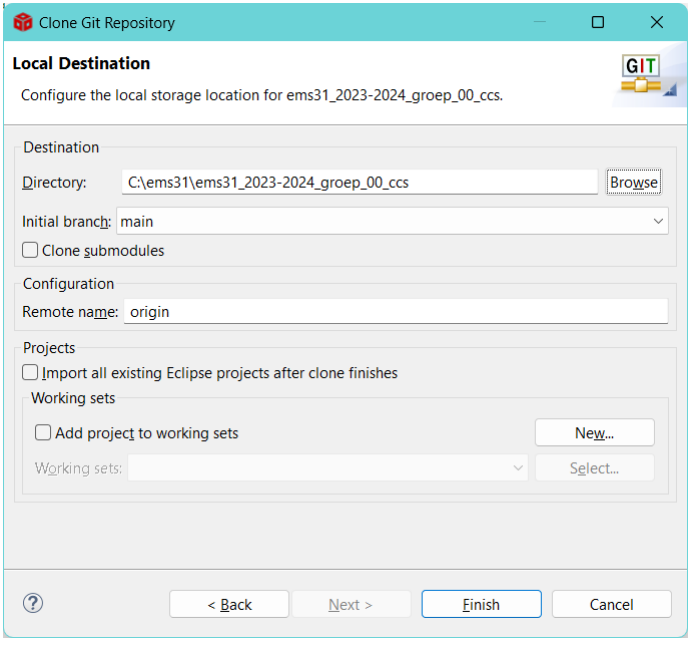
J Vink de optie “Store in Secure Store” aan en klik op Next. Selecteer de branch “main” en klik op Next, zie figuur 30.



Figuur 30: Selecteer de main branch en klik op Next.

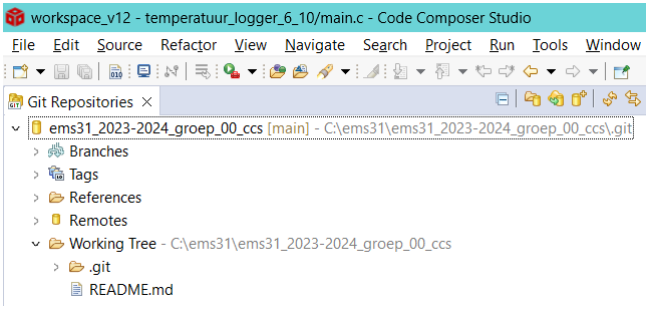
K Selecteer de directory waarin je het repository wilt clonen[13], bijvoorbeeld `C:\ems31` en klik op Finish, zie figuur 31.

[13] Gebruik hiervoor *niet* de workspace directory van Code Composer Studio.



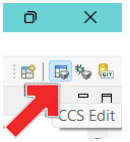
Figuur 31: Selecteer de directory waarin je het repository wilt clonen.

L Als het goed is, zie je nu onder “Git Repositories” het geclonede repository, zie figuur 32. Onder “Working Tree” zie je de geclonede bestanden (in dit geval alleen `README.md`.



Figuur 32: Het geclonede repository.

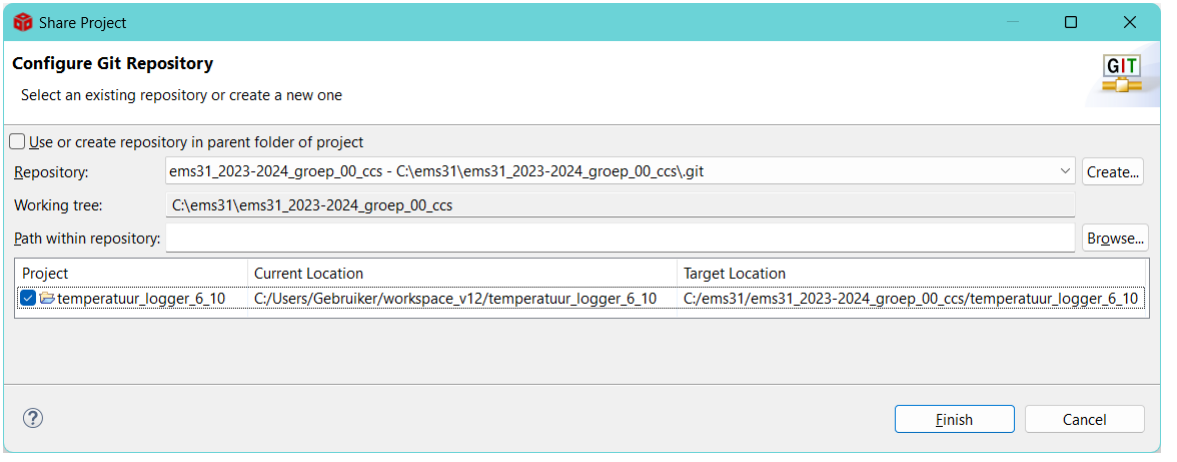
M Schakel met het knopje rechtsboven in CCS over naar het CCS Edit perspective, zie figuur 33.



Figuur 33: Schakel over naar het CCS Edit perspective.

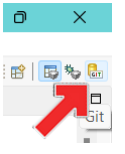
N Klik met de rechtermuisknop op het project `temperatuur_logger_6_10` en kies voor Team > Share Project....

O Selecteer het repository `ems31_2023-2024_groep_XX_ccs`, zie figuur 34. Je ziet dat het project, als je op Finish klikt, gekopieerd wordt vanuit het workspace directory naar de plaats waar je het repository hebt gecloned.



Figuur 34: Plaats het project in het repository.

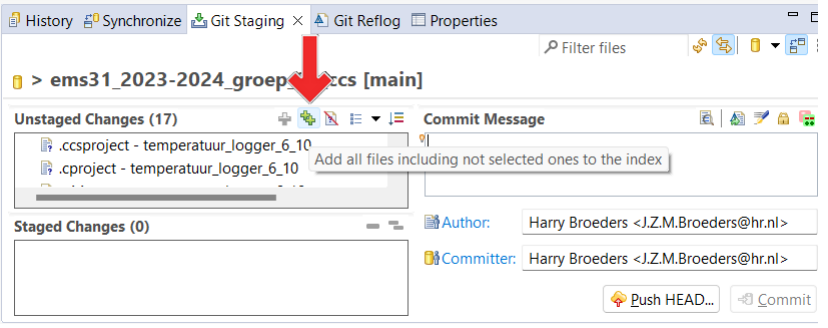
P Schakel met het knopje rechtsboven in CCS over naar het Git perspective, zie figuur 35.



Figuur 35: Schakel over naar het Git perspective.

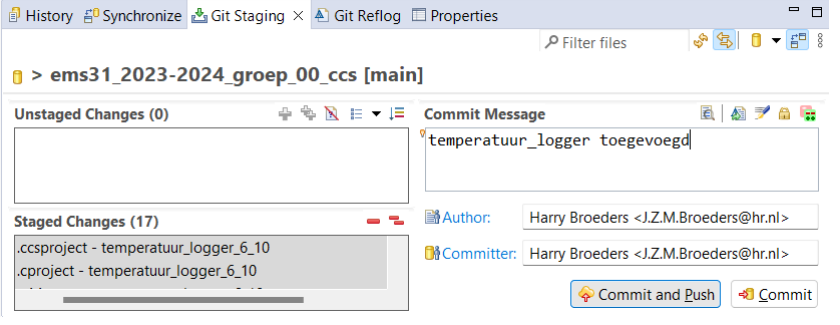
Q Je ziet onder het tabblad “Git Staging” in het veld “Unstaged Changes” de bestanden die je hebt toegevoegd in het directory waarin je het repository hebt gecloned.

Klik op het knopje met de dubbele groene plusjes om deze bestanden te “stagen” (klaar te zetten voor een commit), zie figuur 36.



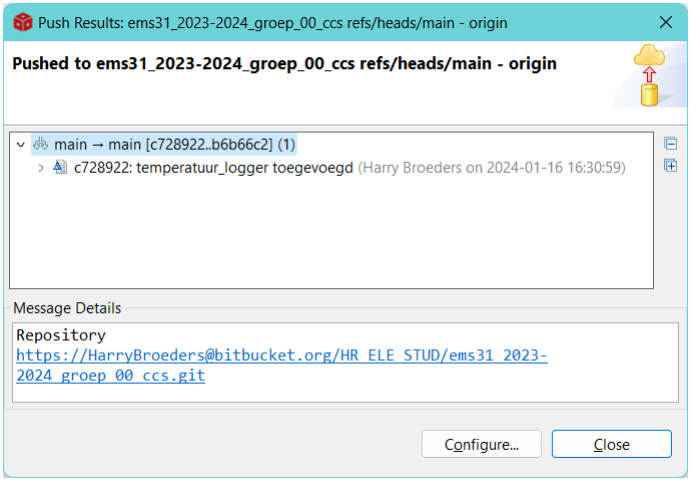
Figuur 36: “Stage” de toegevoegde bestanden.

R Type een “Commit Message” in en en klik op Commit and Push om de bestanden in het repository te committen en het repository te pushen naar bitbucket, zie figuur 37.



Figuur 37: Commit de bestanden en push het repository.

S Ter bevestiging verschijnt het window dat is weergegeven in figuur 38.



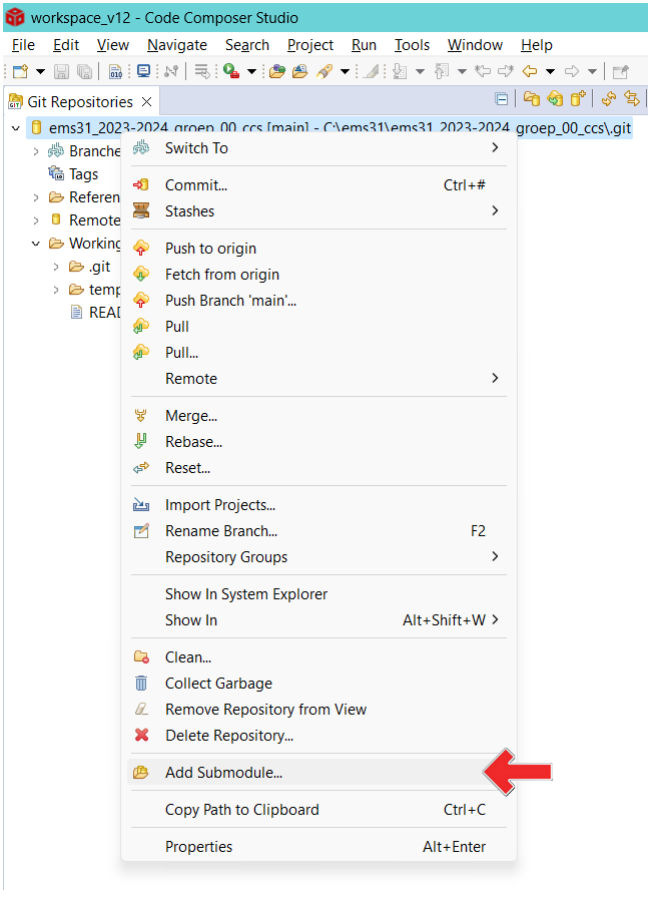
Figuur 38: De bestanden zijn gecommit en het repository is gepusht.

We gaan nu de Git submodule `ems31_2023-2024_groep_XX_submodules`[14]. met daarin de implementatie van een buffer waarin 8 elementen opgeslagen kunnen worden, die je in week 1 hebt geschreven en getest met behulp van WSL, toevoegen aan dit project.

3.2.10 A

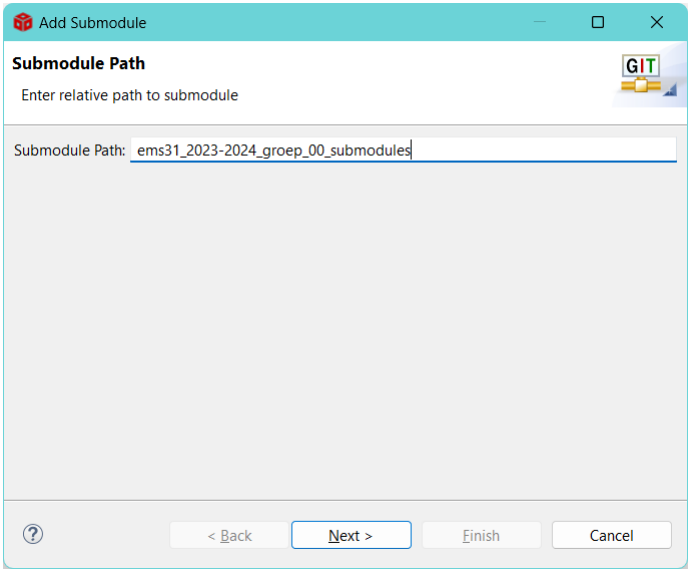
Klik met de rechtermuisknop op het repository `ems31_2023-2024_groep_XX_ccs` en kies voor Add Submodule..., zie figuur 39.

[14] Vervang `XX` door het nummer van je groep.



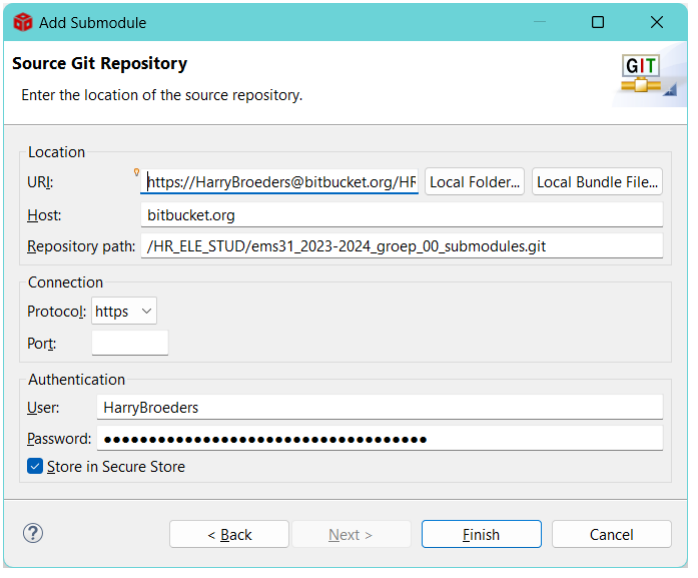
Figuur 39: Voeg een Git submodule toe.

B Voer een directory naam in waarin de Git submodule geplaatst moet worden, bijvoorbeeld `ems31_2023-2024_groep_XX_submodules`, en klik op Next, zie figuur 40.



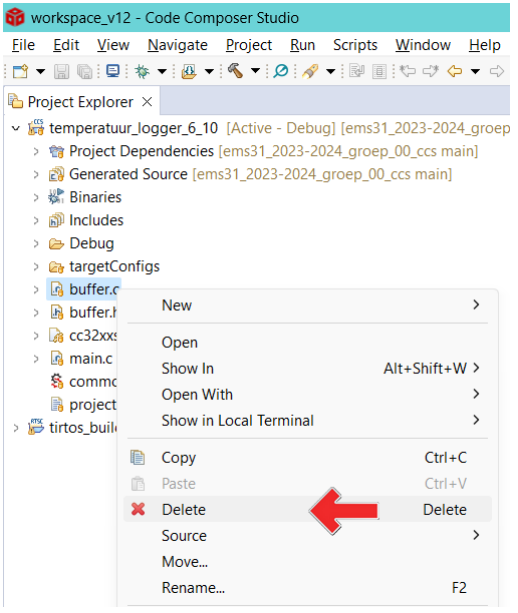
Figuur 40: Voer de directory naam in waarin de Git submodule geplaatst moet worden.

C Ga in je internet browser naar `https://bitbucket.org/` en log in. Ga naar het repository `ems31_2023-2024_groep_XX_submodules` en klik op Clone. Selecteer "HTTPS" en kopieer het git clone commando voor dit repository. Plak dit in het "URI" veld in CCS, zie figuur 41. Klik op Finish.



Figuur 41: Het toevoegen van de Git submodule.

D Schakel met het knopje rechtsboven in CCS over naar het CCS Edit perspective, zie figuur 33. Verwijder de bestanden `buffer.h` en `buffer.c` uit het project `temperatuur_logger_6_10`, zie figuur 42.



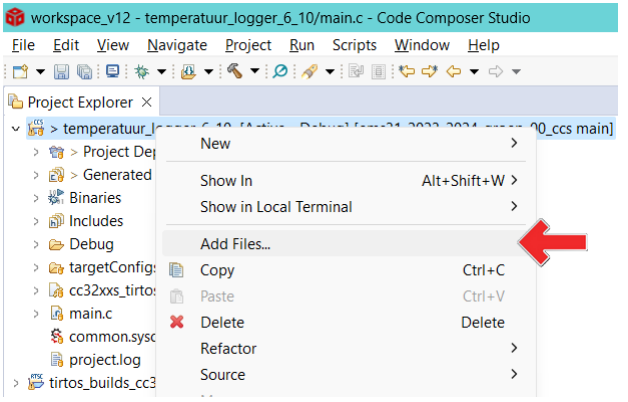
Figuur 42: Verwijder `buffer.h` en `buffer.c`.

E Open het bestand `main.c` en include het bestand `buffer.h` uit de Git submodule, zie figuur 43.

```
30
31 #include "../ems31_2023-2024_groep_00_submodules/buffer_8/buffer.h"
32 #include "ti_drivers_config.h"
```

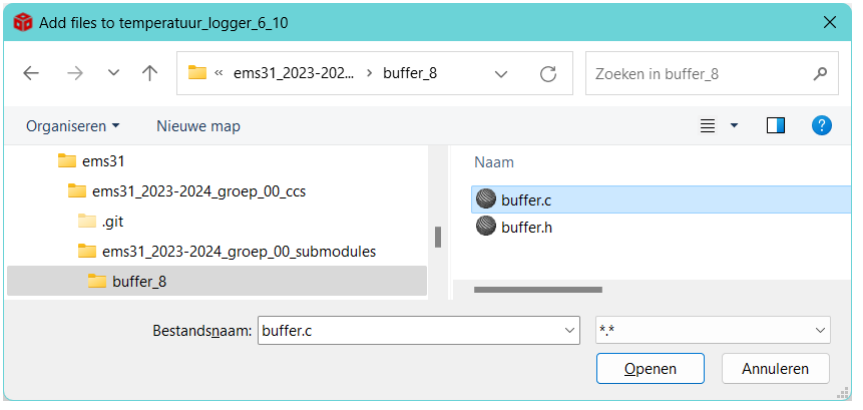
Figuur 43: Include `buffer.h` uit de Git submodule.

F Klik met de rechtermuisknop op het project `temperatuur_logger_6_10` en kies voor Add Files..., zie figuur 44.



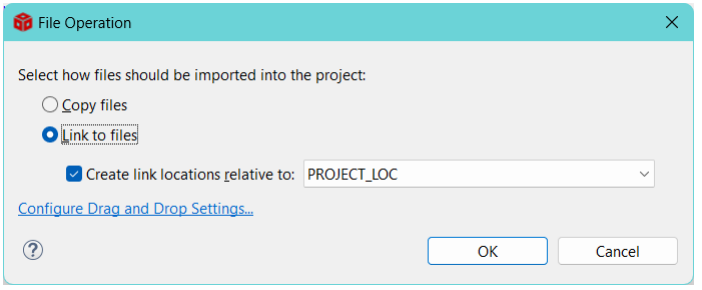
Figuur 44: Voeg bestanden toe aan het project.

G Selecteer het bestand `buffer.c` uit de Git submodule en klik op Openen, zie figuur 45.



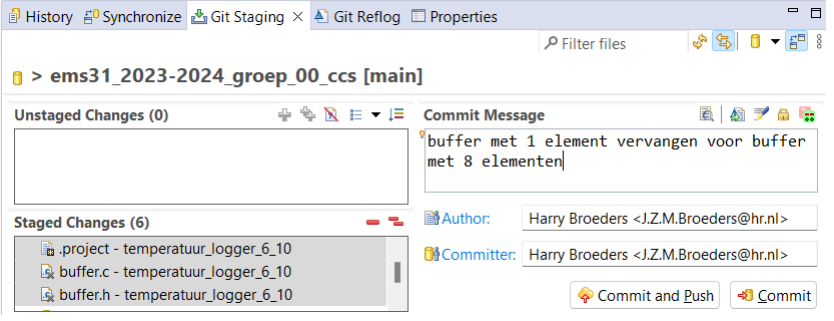
Figuur 45: Selecteer `buffer.c` uit de Git submodule.

H Selecteer “Link to files” en klik op OK, zie figuur 46. We willen een link naar het bestand in het project opnemen omdat we het bestand niet willen kopiëren naar het project, maar we willen het bestand gebruiken dat in de Git submodule staat.



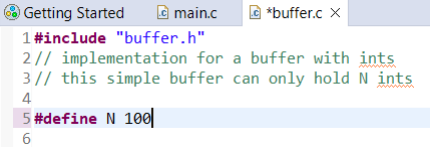
Figuur 46: Neem een link naar het bestand in het project op.

I Schakel met het knopje rechtsboven in CCS over naar het Git perspective, zie figuur 35. Commit de gewijzigde bestanden en push het repository, zie figuur 47.



Figuur 47: Commit en push de wijzigingen.

J Schakel met het knopje rechtsboven in CCS over naar het CCS Edit perspective, zie figuur 33. Open het bestand `buffer.c` en pas de implementatie van de buffer aan zodat 100 in plaats van 8 elementen in de buffer opgeslagen kunnen worden. Als het goed is, dan kun je dit doen door één regel in je code aan te passen, zie figuur 48.



Figuur 48: We willen 100 integers in de buffer op kunnen slaan.

K Build het project `temperatuur_logger_6_10`. Start de debugger en voer het project uit. Start ook het Python-programma `ontvang_temperaturen.py` in Thonny op de pc. Als het goed is, dan zie je nu dat de CC3220S LaunchPad elke seconde

een temperatuurwaarde naar de pc stuurt. Als je het Python programma nu stopt en je vinger op de temperatuursnsor houdt, dan zie je dat de temperatuurwaarde op de pc omhoog gaat, zie figuur 49.

```
Received: temp?
Temperature sampled: 184/10 degrees Celsius
Send 184
Temperature sampled: 184/10 degrees Celsius
Temperature sampled: 184/10 degrees Celsius
Temperature sampled: 184/10 degrees Celsius
Temperature sampled: 184/10 degrees Celsius
Temperature sampled: 184/10 degrees Celsius
Temperature sampled: 184/10 degrees Celsius
Temperature sampled: 184/10 degrees Celsius
Temperature sampled: 184/10 degrees Celsius
Temperature sampled: 184/10 degrees Celsius
Temperature sampled: 184/10 degrees Celsius
Temperature sampled: 184/10 degrees Celsius
Temperature sampled: 185/10 degrees Celsius
Temperature sampled: 185/10 degrees Celsius
Temperature sampled: 185/10 degrees Celsius
Temperature sampled: 190/10 degrees Celsius
Temperature sampled: 195/10 degrees Celsius
Temperature sampled: 200/10 degrees Celsius
Temperature sampled: 204/10 degrees Celsius
Temperature sampled: 205/10 degrees Celsius
Temperature sampled: 208/10 degrees Celsius
Temperature sampled: 210/10 degrees Celsius
Temperature sampled: 210/10 degrees Celsius
Temperature sampled: 212/10 degrees Celsius
Temperature sampled: 212/10 degrees Celsius
Temperature sampled: 213/10 degrees Celsius
Temperature sampled: 214/10 degrees Celsius
```

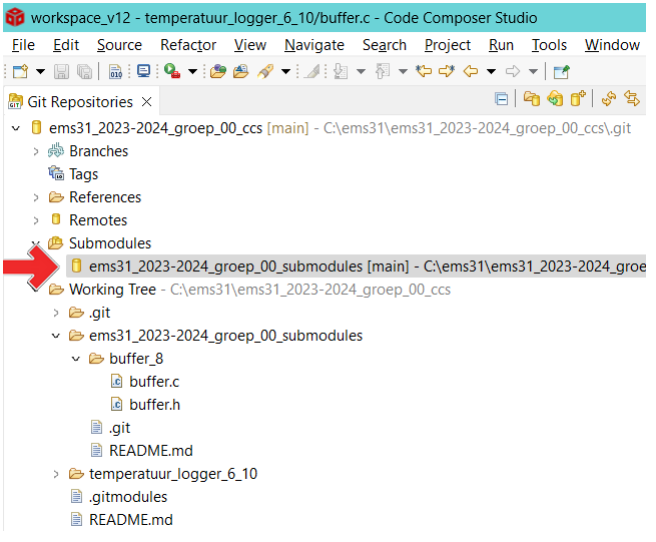
Figuur 49: Elke seconde wordt de temperatuur gemeten.

L Als je het Python programma nu, *na ongeveer een minuut*, weer opstart, dan zie je dat er geen temperatuurwaarden verloren zijn gegaan en dat alle gemeten temperatuurwaarden alsnog verschijnen op de pc, zie figuur 50. Deze waarden zijn tussentijds opgeslagen in de buffer op de CC3220S LaunchPad. Hoelang kun je het Python programma stoppen voordat er wel temperatuurwaarden verloren gaan?

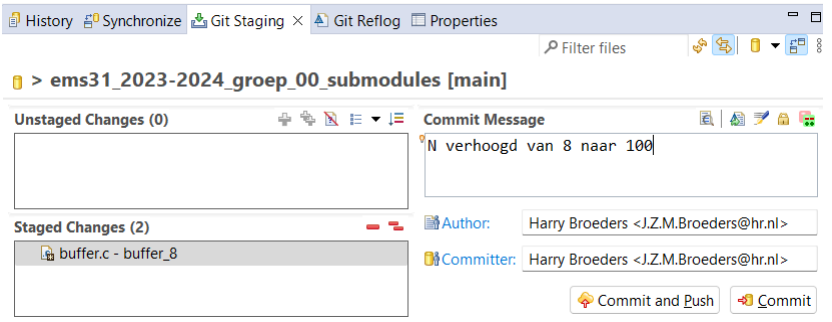
```
Shell
>>> %Run ontvang_temperaturen.py
 Starting ontvang_temperaturen
 Temperature received: 184
 Temperature received: 184
 Temperature received: 184
 Temperature received: 184
 Temperature received: 184
 Temperature received: 184
 Temperature received: 184
 Temperature received: 184
 Temperature received: 184
 Temperature received: 184
 Temperature received: 184
 Temperature received: 184
 Temperature received: 184
 Temperature received: 185
 Temperature received: 185
 Temperature received: 185
 Temperature received: 190
 Temperature received: 195
 Temperature received: 200
 Temperature received: 204
 Temperature received: 205
 Temperature received: 208
 Temperature received: 210
 Temperature received: 210
 Temperature received: 212
 Temperature received: 212
 Temperature received: 213
 Temperature received: 214
```

Figuur 50: De in de buffer opgeslagen temperatuurwaarden worden op de pc getoond.

M Schakel met het knopje rechtsboven in CCS over naar het Git perspective, zie figuur 35. Selecteer de Git submodule `ems31_2023-2024_groep_XX_submodules`, zie figuur 51 en commit en push de verandering, zie figuur 52.



Figuur 51: Selecteer de Git submodule.



Figuur 52: Commit en push de verandering in de Git submodule.

Het is lastig om te bepalen hoe groot de buffer moet zijn om te voorkomen dat er data verloren gaat. Als we de buffer te klein kiezen gaat er misschien onnodig data verloren, maar als we de buffer te groot kiezen dan past het misschien niet in het RAM van de CC3220S LaunchPad.

Volgende week ga je een buffer implementeren die “dynamisch geheugen” gebruikt. Hierdoor wordt het geheugen pas gereserveerd als het nodig is. Het buffer loopt dan pas over als het RAM van de CC3220S LaunchPad volledig gevuld is.