

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 Google-Test;
- 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++ TestMate².

Open een terminal window door op **ctrl** + **'** te drukken. Installeer het GoogleTest framework met het volgende commando³:

```
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
```

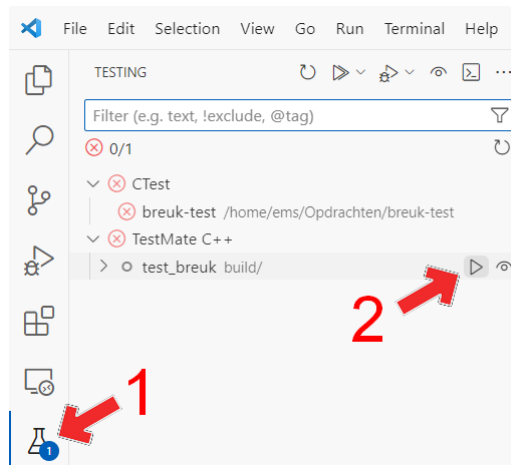
Download het bestand breuk-test.zip, pak het uit en verwijder het weer met de volgende commando's:

```
wget https://github.com/HR-ELEKTRO/ems3/raw/refs/heads/master/↵  
↵ Programmas/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
```

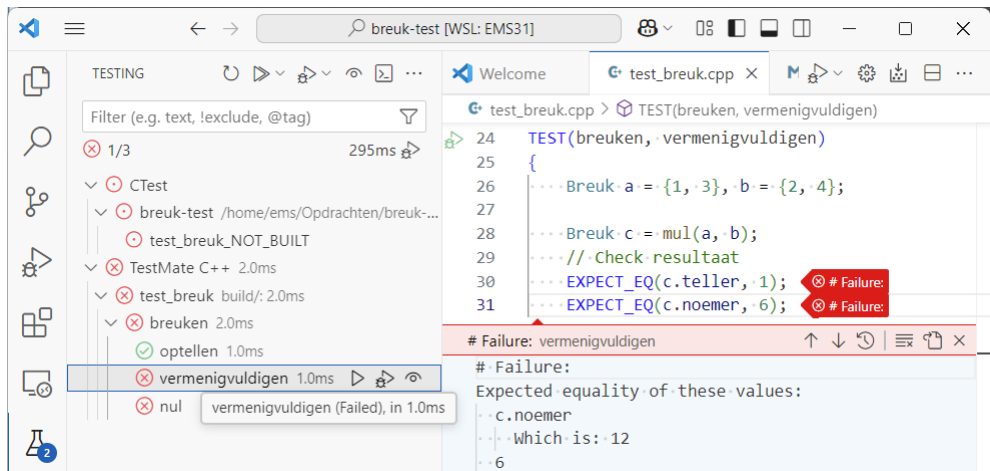
Build het project door op de Build-knop te klikken of door op **F7** te drukken. Je kunt nu de testen uitvoeren door in de statusbalk op de Play-knop te klikken. Maar het is handiger om gebruik te maken van de “C++ TestMate”-plugin⁴, zie [figuur 1](#).



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

Als je op het >-teken voor test_break in het TESTING-window klikt, dan zie je het resultaat van de verschillende testen, zie [figuur 2](#).

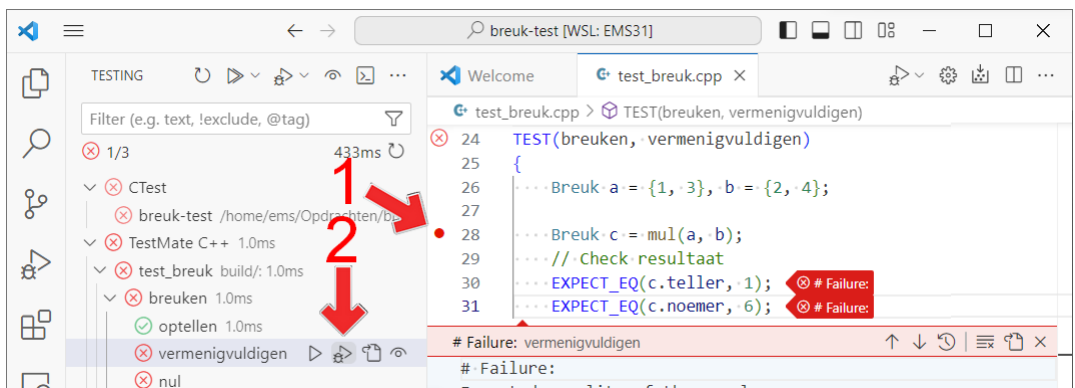
⁴ Zie: <https://marketplace.visualstudio.com/items?itemName=matepek.vscode-catch2-test-adapter>



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](#).

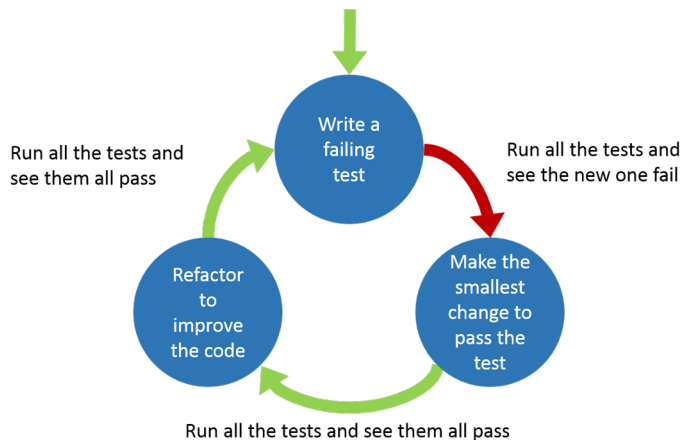


Figuur 3: Debuggen van een test.

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 divide⁵ 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.

⁵ 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>.

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.

Het programma eindigt dan met de melding:

```
test_breuk: /home/ems/Opgaven/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](#).

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TEST RESULTS  TERMINAL  PORTS
$46 | Started PID#15929 - '/home/ems/Opgaven/breuk-test/build/test_breuk'
$46 |
$46 | [ RUN      ] breuken.nul @ ./test_breuk.cpp:59
$46 | | std::cerr:
$46 | test_breuk: /home/ems/Opgaven/breuk-test/breuk.c:21: normaliseer: Assertion
`b.noemer != 0' failed.
$46 |
$46 | | std::cerr
$46 | Stopped PID#15929 - Signal received: SIGABRT - '/home/ems/Opgaven/breuk-test/
/build/test_breuk'
$46 | ✖ Executable run is finished with error.$46 | "/home/ems/Opgaven/breuk-test/
/build/test_breuk"" "--gtest_color=no"" "--gtest_filter=breuken.nul"" "--gtest_also_r
un_disabled_tests""$46 | ! Test has ended unexpectedly: Signal received: SIGABRT

```

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 test⁶ 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.

⁶ Zie eventueel <https://github.com/google/googletest/blob/main/docs/advanced.md#death-tests>.

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_2025-2026_groep_XX_wsl⁷).
- Een repository voor de code die op in Code Composer Studio ontwikkeld wordt (ems31_2025-2026_groep_XX_ccs).
- Een repository voor de modules (bijvoorbeeld buffer) die zowel onder WSL op de pc als in CCS gebruikt worden (ems31_2025-2026_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://github.com/HR-ELEKTRO/ems3/wiki/Software_WSL_git.md. Als je dit al gedaan hebt bij het installeren van WSL, dan kun je deze stap overslaan.

Let op! Slechts één groepslid moet de volgende opdracht uitvoeren:

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_2025-2026_groep_XX_wsl⁸ in je home directory met behulp van het commando:

```
git clone git@bitbucket.org:HR_ELE_STUD/ems31_2025-2026_groep_↵  
↵ XX_wsl.git
```

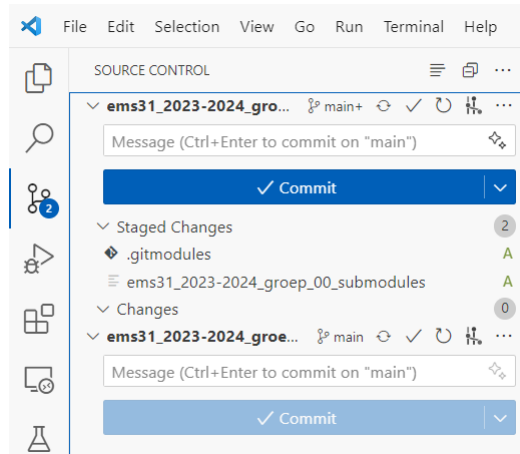
Ga naar het gecloonde repository en voeg daar een submodule toe met behulp van de commando's:

⁷ In plaats van XX staat het nummer van je groep.

⁸ Vul in plaats van XX het nummer van je groep in.

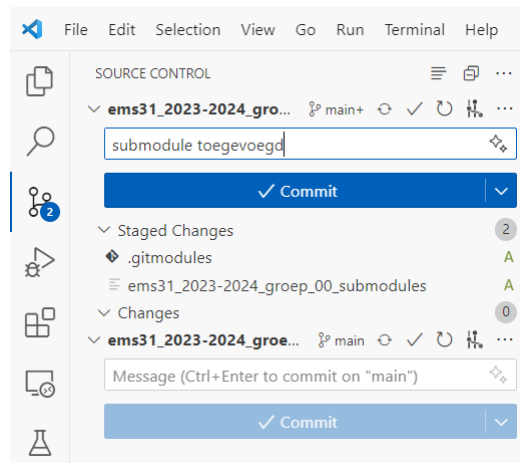
```
cd ems31_2025-2026_groep_XX_wsl
git submodule add git@bitbucket.org:HR_ELE_STUD/ems31_2025-2026↵
↵ _groep_XX_submodules.git
```

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



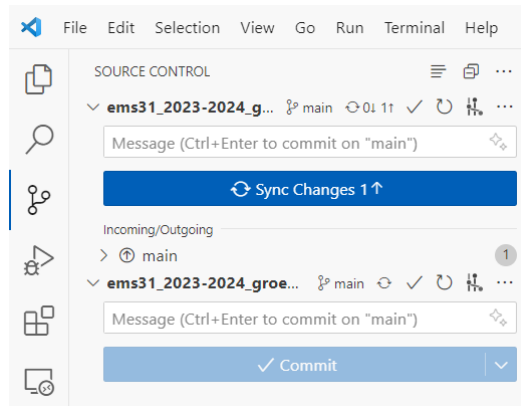
Figuur 6: Source Control window.

Commit nu de wijzigingen die je hebt aangebracht in het repository ems31_2025-2026_groep_XX_wsl, zie [figuur 7](#).



Figuur 7: Commit de wijzigingen.

Push de wijzigingen naar de remote repository, zie [figuur 8](#).

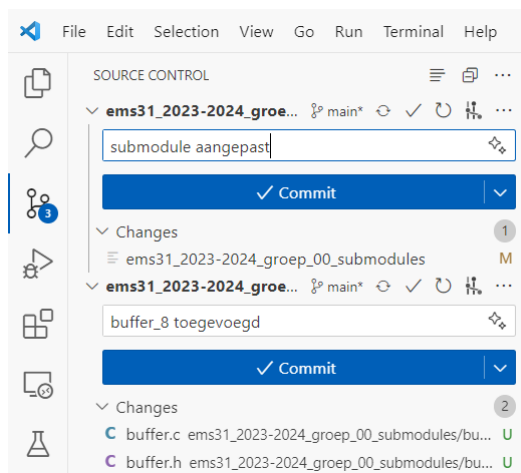


Figuur 8: Push de wijzigingen.

Ga nu naar het directory `ems31_2025-2026_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_2025-2026_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.

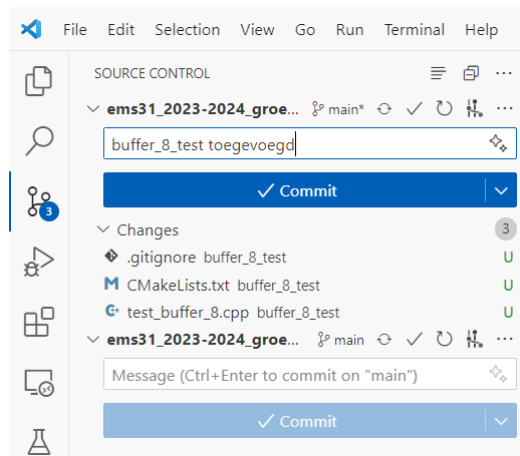
Ga terug naar het directory `ems31_2025-2026_groep_XX_wsl` en voer daar de volgende commando's uit:

```
wget https://github.com/HR-ELEKTRO/ems3/raw/refs/heads/master/↵  
↵ Programmas/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
```

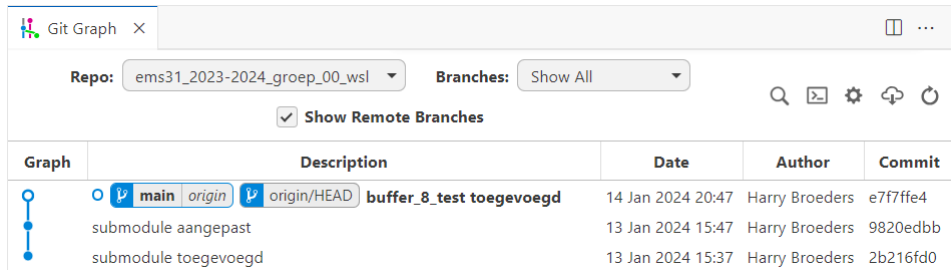
Pas het path naar het bestand `buffer.c` door `XX` te vervangen met je groepsnummer. Open het bestand `test_buffer.c` en pas de regel met de `#include` aan door `XX` te vervangen met je groepsnummer. Sluit beide bestanden en commit de wijzigingen, zie [figuur 10](#).



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 punt⁹):

```
code -r .
```

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

Let op! Het groepslid dat opdracht 3.2.6 niet heeft uitgevoerd moet nu de volgende opdracht uitvoeren:

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

```
code -r ~
```

Clone nu het repository `ems31_2025-2026_groep_XX_wsl`¹⁰ inclusief de submodule in je home directory met behulp van het commando:

```
git clone git@bitbucket.org:HR_ELE_STUD/ems31_2025-2026_groep_ -↵
↵ XX_wsl.git --recurse-submodules
```

Ga naar het gecloonde submodule repository en creëer daar de master branch:

```
cd ems31_2025-2026_groep_XX_wsl
cd ems31_2025-2026_groep_XX_submodules
git checkout master
```

⁹ De punt is een afkorting voor de huidige directory.

¹⁰ Vul in plaats van XX het nummer van je groep in.

Beide groepsleden kunnen nu de wsl repository inclusief de submodule pullen met de commando's:

```
cd ~/ems31_2025-2026_groep_XX_wsl
git pull --recurse-submodules
```

Zie voor verdere uitleg <https://gist.github.com/gitaarik/8735255>. Als je problemen tegenkomt bij het gebruik van submodules, dan kun je de volgende bron raadplegen: <https://gist.github.com/mrnabati/bc59304784e1a3a48dd25f92bf20a420>.

3.2.8 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 gebruikt¹¹.

Om software te compileren voor dit bord gebruiken je Code Composer Studio (CCS) versie 12.4, waar je ook bij EMS20 al mee hebt gewerkt. Daarnaast heb je, net zoals bij EMS20, ook de SimpleLink CC32XX SDK versie 6.10 nodig. Een installatiehandleiding vind je op: https://bitbucket.org/HR_ELEKTRO/ems20/wiki/installatie-software.md.

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

¹¹ Indien nodig kun je een CC3220S LaunchPad aanschaffen bij de winkel op Academie Plein.

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-pakketten kunnen op de pc ontvangen en zichtbaar gemaakt worden met het programma `ontvang_temperatures.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.

Let op! Wij adviseren om de software voor de CC3220S LaunchPad te ontwikkelen op de pc van één van de groepsleden¹².

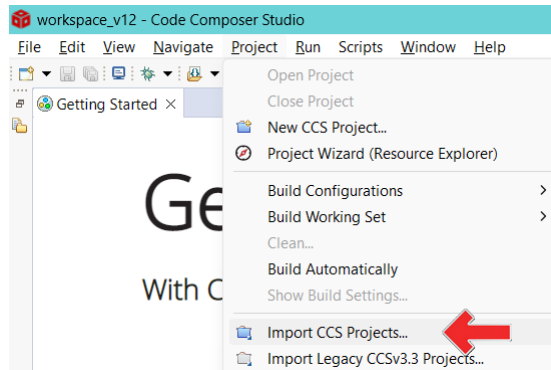
Let op! Wij adviseren om de code in de submodule alleen aan te passen (en meteen te testen) in de WSL omgeving. Deze aangepaste code kun je dan eerst pushen naar de submodule repository en daarna pullen in de CCS omgeving.

3.2.9 A Download het bestand [temperatuur_logger_6_10.zip](#)¹³.

B Start Code Composer Studio en open het menu  `Import CCS Projects...`, zie [figuur 12](#).

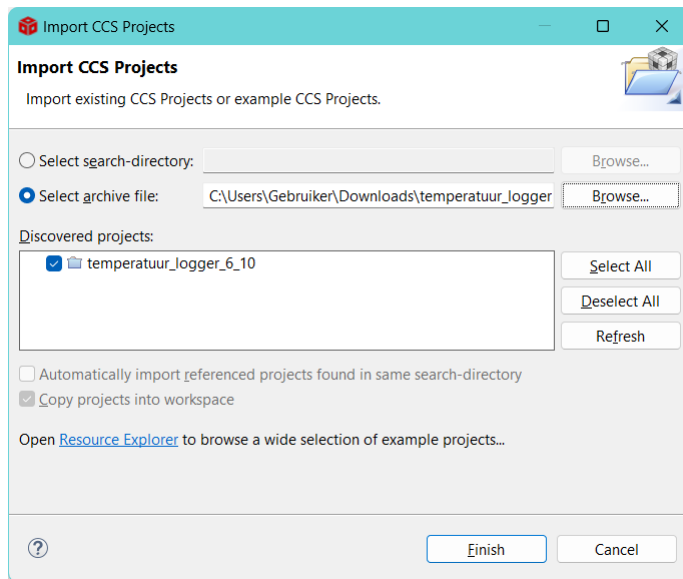
¹² Als je het CCS-project toch wilt overnemen van je groepsgenoot en verder samen wilt ontwikkelen, bedenk dan dat je ook de submodule wilt overnemen en maak gebruik van de uitleg die worden gegeven in de volgende bron: https://software-dl.ti.com/ccs/esd/documents/sdto_ccs_git-with-ccs.html#import-a-version-controlled-ccs-project.

¹³ https://github.com/HR-ELEKTRO/ems3/raw/refs/heads/master/Opdrachten/progs/temperatuur_logger/temperatuur_logger_6_10.zip



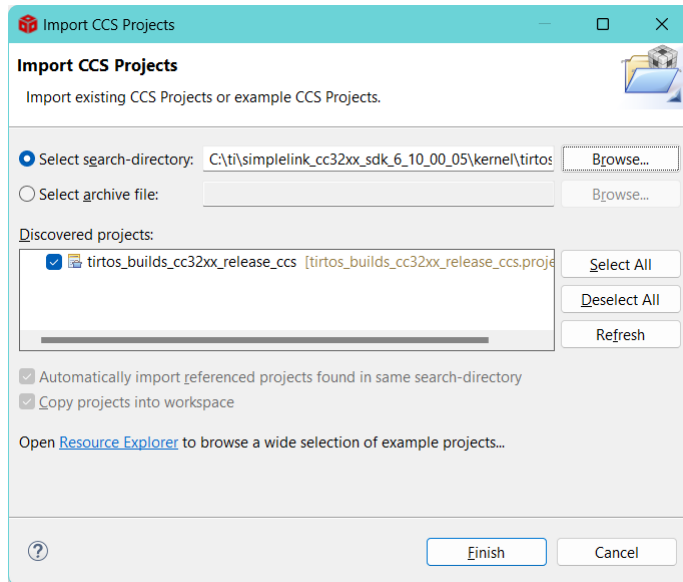
Figuur 12: Open het menu `Project` » `Import CCS Projects...`.

- C** Kies “Select archive file” en selecteer het bestand `temperatuur_logger_6_10.zip` en klik op `Finish`, zie [figuur 13](#).



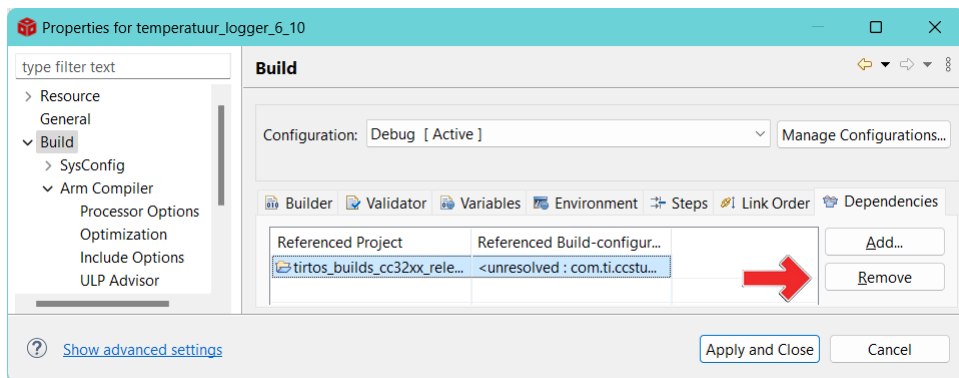
Figuur 13: Select archive file `temperatuur_logger_6_10.zip`

- 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](#).

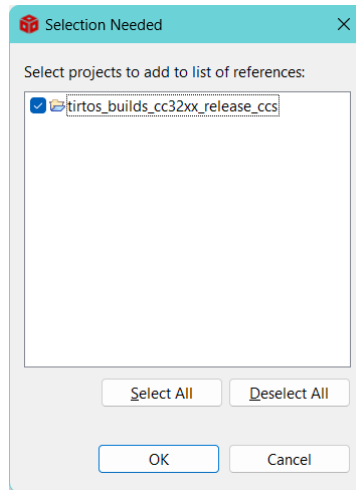


Figuur 14: Importeer het project `tirtos_builds_cc32xx_release_ccs`.

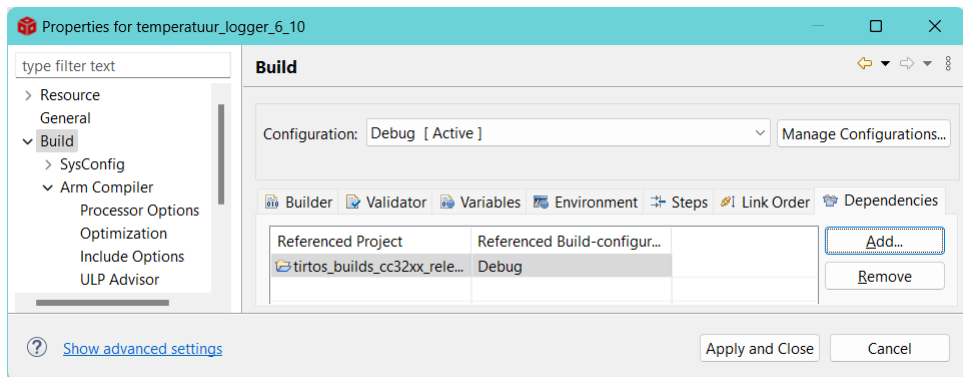
Pas nu de *Properties* van het project `temperatuur_logger_6_10` als volgt aan: ga naar **Build** > **Dependencies**, verwijder het *unresolved* project, zie [figuur 15](#) en voeg nu het geïmporteerde project toe door op **Add..** te drukken en `tirtos_builds_cc32xx_release_ccs` te selecteren, zie [figuren 16](#) en [17](#).



Figuur 15: verwijder het *unresolved* project.



Figuur 16: Selecteer tirtos_builds_cc32xx_release_ccs.

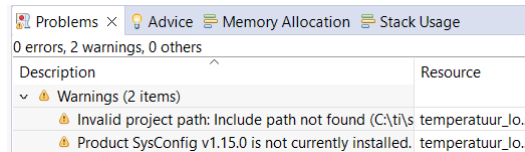


Figuur 17: Klik op `Apply and Close`.

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](#).

```

Console ×
temperatuur_logger_6_10:CIO
[Cortex_M4_0] I2C Initialized!
Connecting to: Broeders.
Detected TMP006 sensor.
Temperature sampled: 177/10 degrees Celsius
Connected to WLAN.
IP Acquired: IP = 192.168.1.96, ← = 192.168.1.254
Waiting for client
Temperature sampled: 177/10 degrees Celsius
Temperature sampled: 177/10 degrees Celsius
Temperature sampled: 176/10 degrees Celsius

```

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](#).

```

Console ×
temperatuur_logger_6_10:CIO
Received: temp?
Temperature sampled: 178/10 degrees Celsius
Send 178
Received UDP packet from: IP = 192.168.1.75, PORT = 52452
Received: temp?
Temperature sampled: 178/10 degrees Celsius
Send 178

```

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

```

Shell ×
Starting ontvang_temperaturen
Temperature received: 177
Temperature received: 178
Temperature received: 178

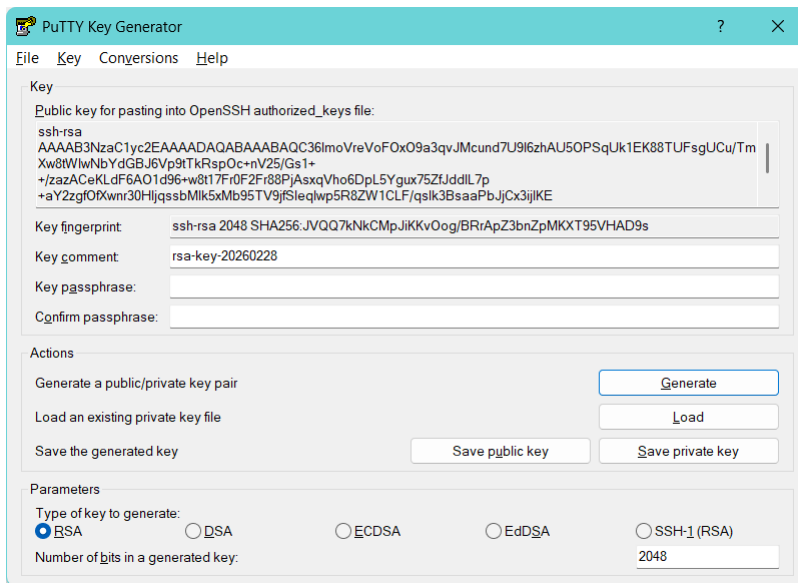
```

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?¹⁴

We gaan het project `temperatuur_logger_6_10` nu opslaan in het Git repository `ems31_2025-2026_groep_XX_ccs`¹⁵.

3.2.10 A Bitbucket (en GitHub, GitLab etc.) ondersteunt authenticatie met een publieke en een privé sleutel. Om een sleutelpaar te genereren gebruiken we de PuTTY package¹⁶. Start het programma PuTTYgen en klik op `Generate`, zie [figuur 22](#). Sla de private key op. Gebruik, als andere personen toegang hebben tot jouw computer, een passphrase, zodat de private key beveiligd is.



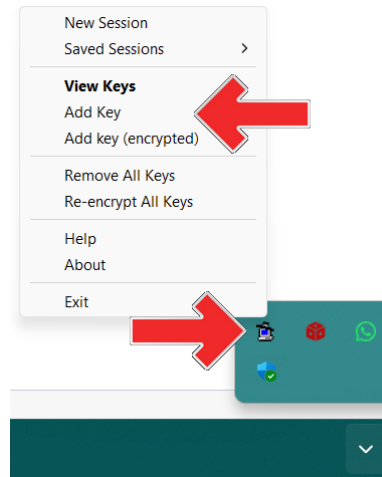
Figuur 22: Het genereren van een sleutelpaar.

B Start het programma Pageant (onderdeel PuTTY package), open het op de taakbalk en laad de private key die je zojuist hebt opgeslagen, zie [figuur 23](#).

¹⁴ 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.

¹⁵ Vervang `XX` door het nummer van je groep.

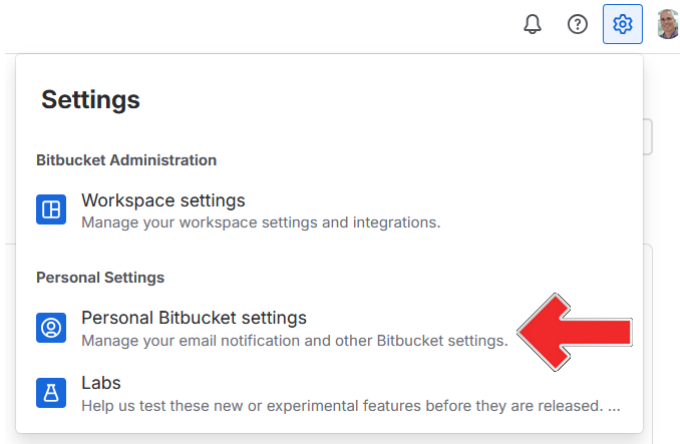
¹⁶ Je kunt de PuTTY package downloaden van <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>.



Figuur 23: Het laden van de private key in Pageant.

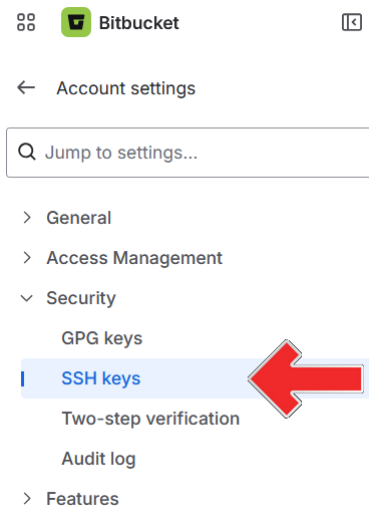
C Kopieer de gegenereerde *public key* en voeg deze, als volgt, toe aan je Bitbucket account:

- Log in op Bitbucket en klik op het tandwiel rechtsboven en kies voor “Personal Bitbucket settings”, zie [figuur 24](#).



Figuur 24: Personal Bitbucket settings aanpassen.

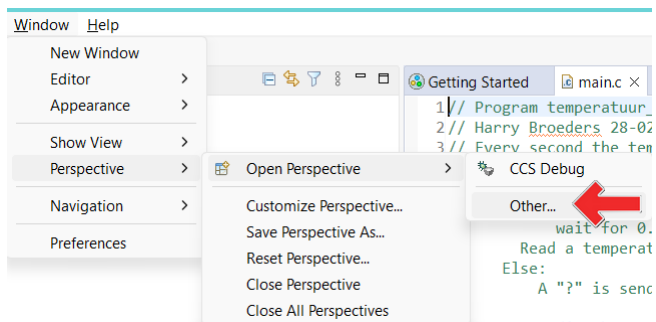
- Kies voor Security > SSH keys en klik op `Add key`, zie [figuur 25](#).



Figuur 25: Key toevoegen.

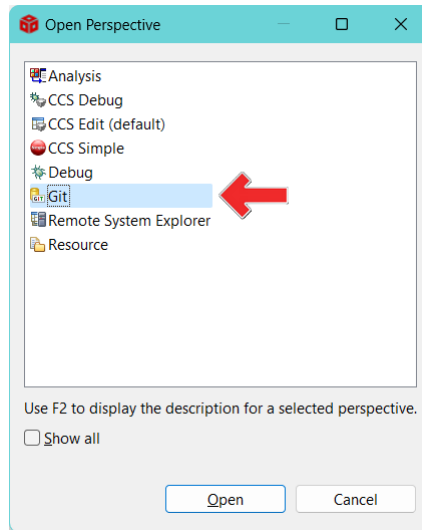
- Plak de public key in het veld “SSH Public Key” en geef de key een naam, bijvoorbeeld “CCS key”, en klik op `Add key`.

D Open in CCS het menu `Window` » `Perspective` » `Open Perspective` » `Other...`, zie [figuur 26](#).



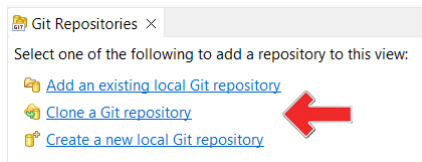
Figuur 26: Open het menu `Window` » `Perspective` » `Open Perspective` » `Other...`.

E Kies “Git” en klik op `Open`, zie [figuur 27](#).



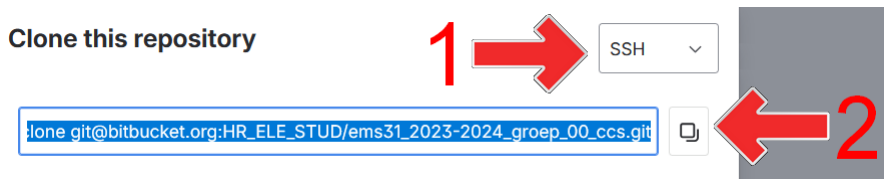
Figuur 27: Open het Git perspective.

F Klik op de link “Clone a Git repository”, zie [figuur 28](#).



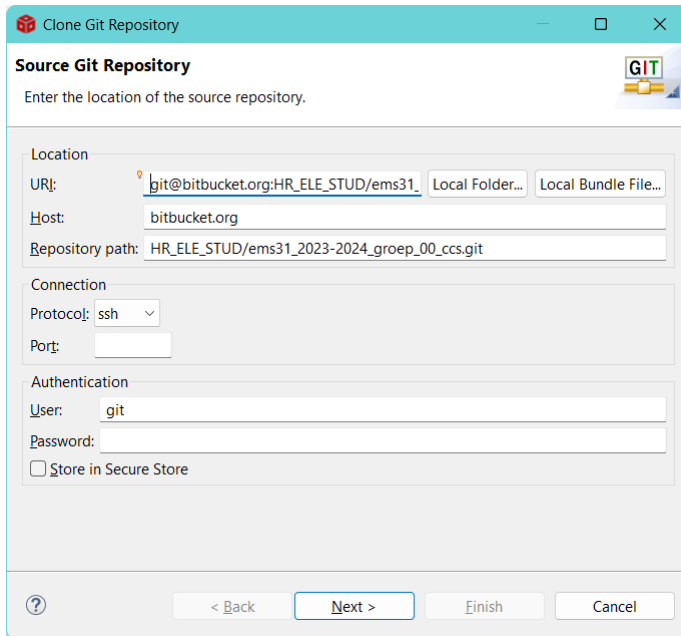
Figuur 28: Clone a Git repository.

G Ga in je internet browser naar <https://bitbucket.org/> en log in. Ga naar het repository `ems31_2025-2026_groep_XX_ccs` en klik op `Clone`. Selecteer “SSH” en kopieer het git clone commando voor dit repository, zie [figuur 29](#).



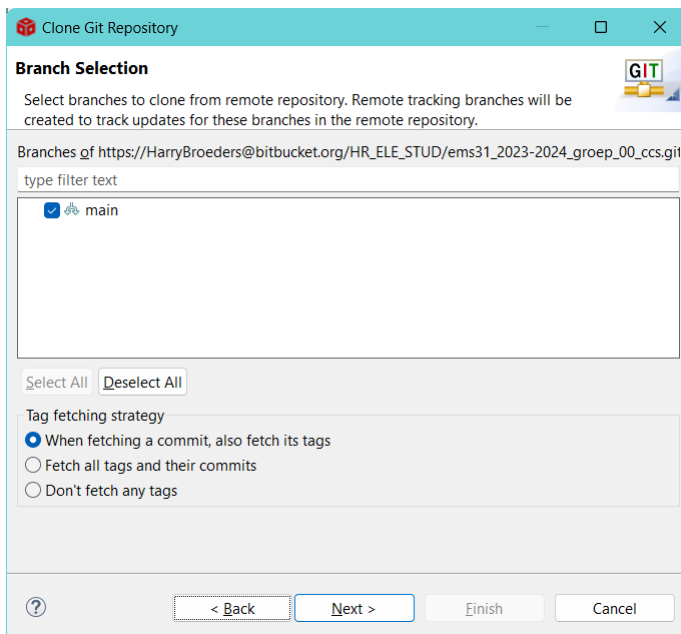
Figuur 29: Kopieer het git clone commando voor dit repository.

H Plak dit in het “URI” veld in CCS, zie [figuur 30](#). Een aantal velden wordt nu automatisch ingevuld. Klik op `Next`.



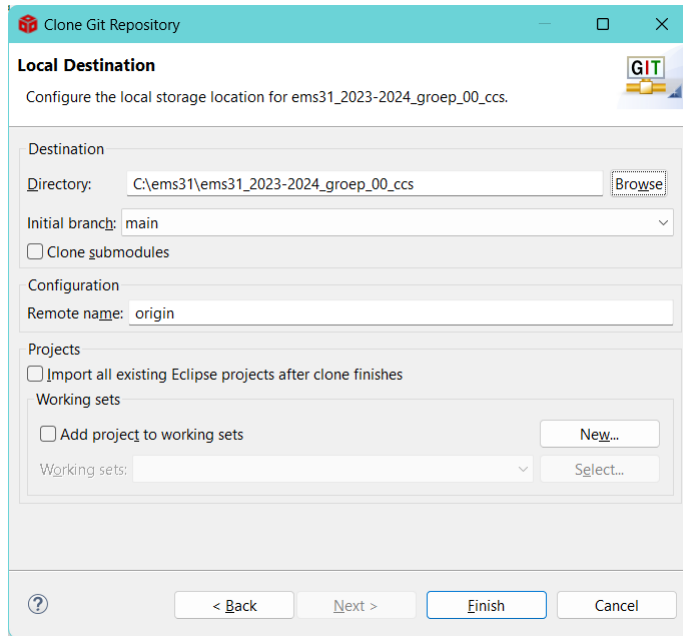
Figuur 30: Plak het git clone commando in het Clone Git Repository window in CCS.

I Selecteer de branch “main” en klik op **Next**, zie [figuur 31](#).



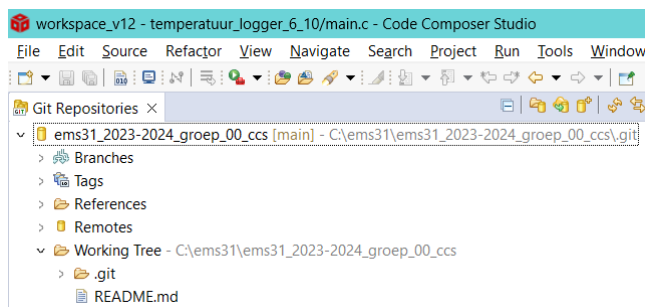
Figuur 31: Selecteer de main branch en klik op **Next**.

J Selecteer de directory waarin je het repository wilt clonen¹⁷, bijvoorbeeld `C:\ems31` en klik op **Finish**, zie [figuur 32](#).



Figuur 32: Selecteer de directory waarin je het repository wilt clonen.

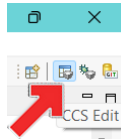
K Als het goed is, zie je nu onder “Git Repositories” het gecloneerde repository, zie [figuur 33](#). Onder “Working Tree” zie je de gecloneerde bestanden (in dit geval alleen `README.md`).



Figuur 33: Het gecloneerde repository.

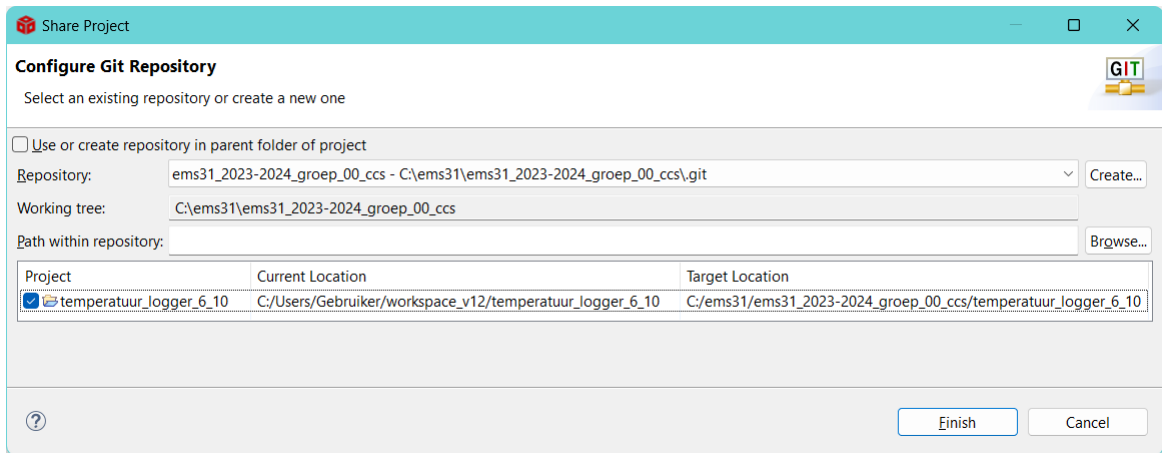
¹⁷ Gebruik hiervoor *niet* de workspace directory van Code Composer Studio.

- L** Schakel met het knopje rechtsboven in CCS over naar het CCS Edit perspective, zie [figuur 34](#).



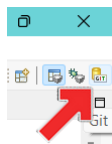
Figuur 34: Schakel over naar het CCS Edit perspective.

- M** Klik met de rechtermuisknop op het project `temperatuur_logger_6_10` en kies voor `Team` > `Share Project...`.
- N** Selecteer het repository `ems31_2025-2026_groep_XX_ccs`, zie [figuur 35](#). 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.



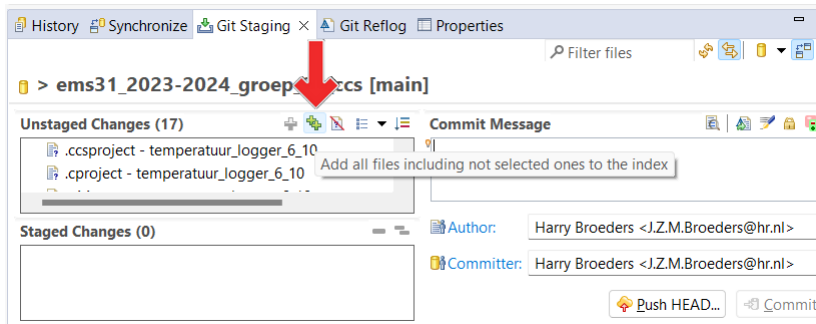
Figuur 35: Plaats het project in het repository.

- O** Schakel met het knopje rechtsboven in CCS over naar het Git perspective, zie [figuur 36](#).



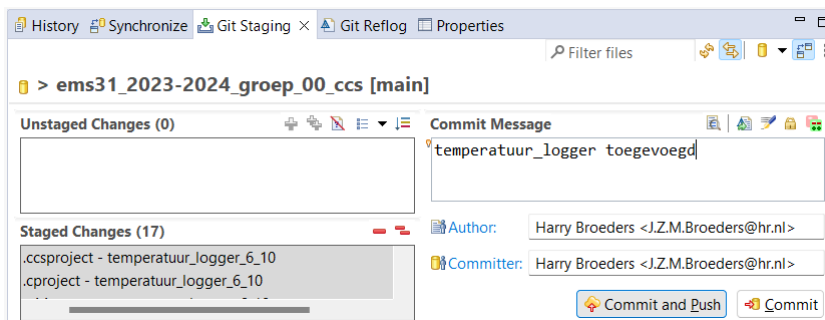
Figuur 36: Schakel over naar het Git perspective.

P 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 37](#).



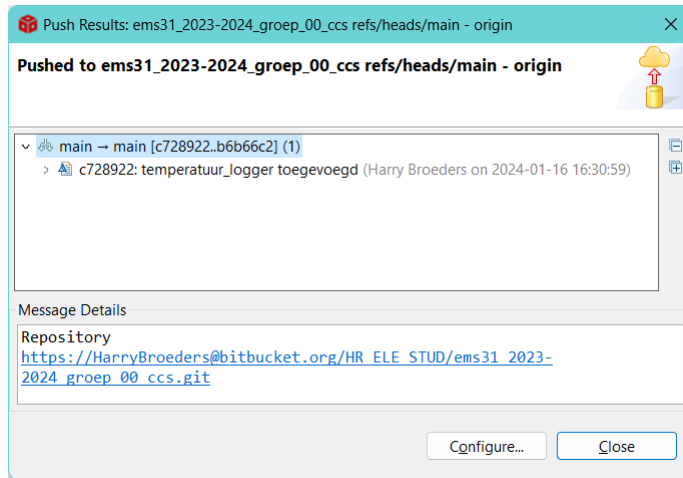
Figuur 37: “Stage” de toegevoegde bestanden.

Q Type een “Commit Message” in en klik op [Commit and Push](#) om de bestanden in het repository te committen en het repository te pushen naar bitbucket, zie [figuur 38](#).



Figuur 38: Commit de bestanden en push het repository.

R Ter bevestiging verschijnt het window dat is weergegeven in [figuur 39](#).

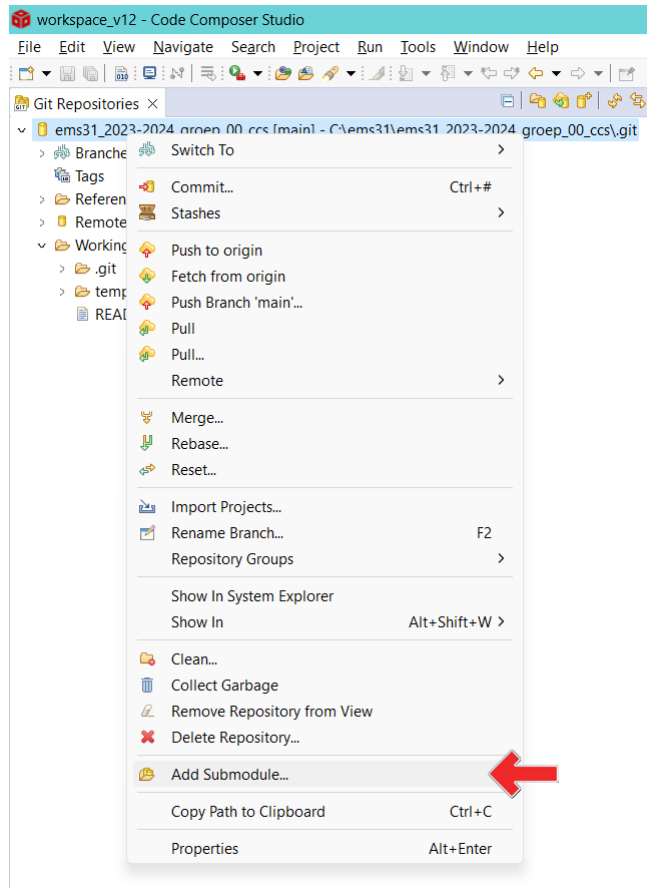


Figuur 39: De bestanden zijn gecommit en het repository is gepusht.

We gaan nu de Git submodule `ems31_2025-2026_groep_XX_submodules`¹⁸ 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.

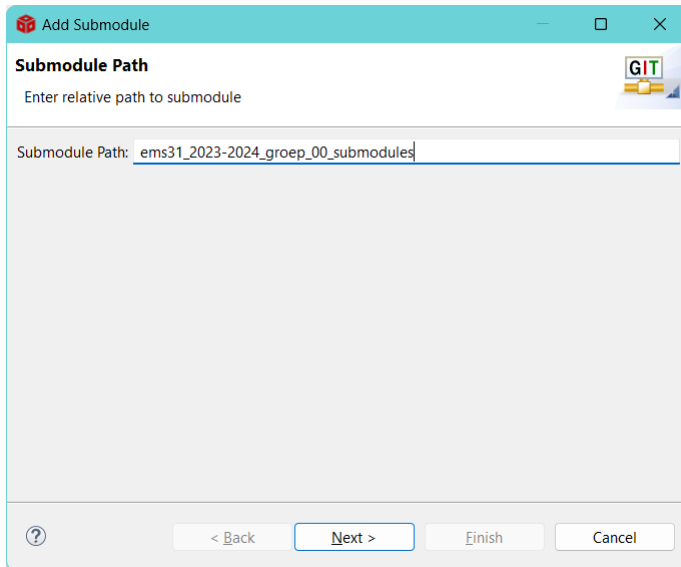
3.2.11 A Klik met de rechtermuisknop op het repository `ems31_2025-2026_groep_XX_ccs` en kies voor `Add Submodule...`, zie [figuur 40](#).

¹⁸ Vervang XX door het nummer van je groep.



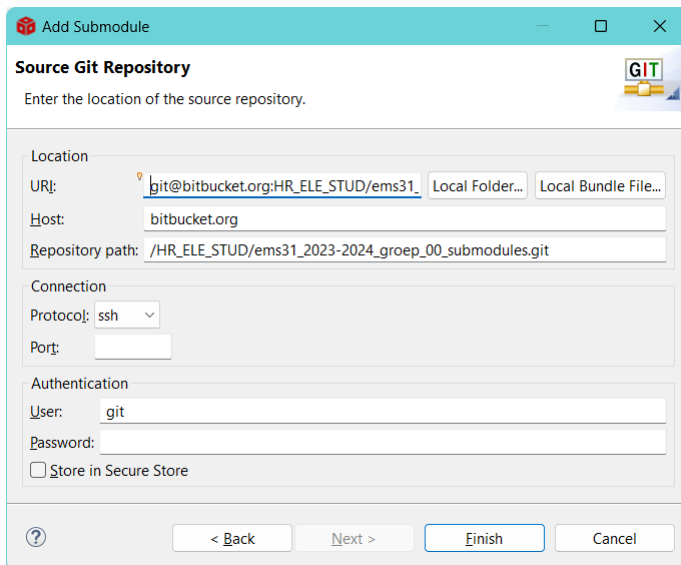
Figuur 40: Voeg een Git submodule toe.

- B** Voer een directory naam in waarin de Git submodule geplaatst moet worden, bijvoorbeeld `ems31_2025-2026_groep_XX_submodules`, en klik op `Next`, zie [figuur 41](#).



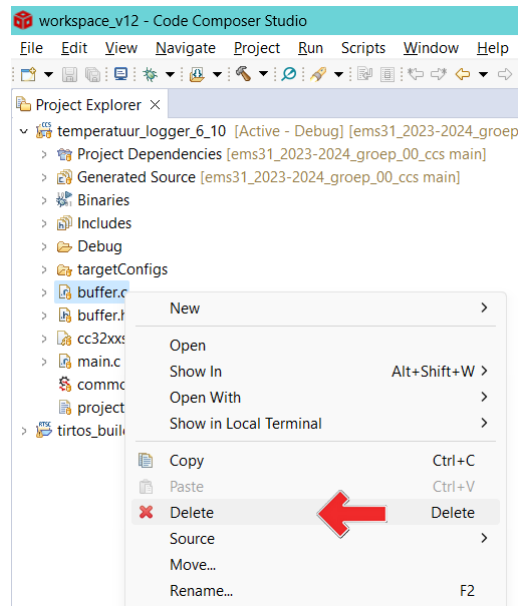
Figuur 41: Voer de directory naam in waarin de Git submodule geplaatst moet worden.

- C** Ga in je internet browser naar <https://bitbucket.org/> en log in. Ga naar het repository `ems31_2025-2026_groep_XX_submodules` en klik op `Clone`. Selecteer “SSH” en kopieer het git clone commando voor dit repository. Plak dit in het “URI” veld in CCS, zie [figuur 42](#). Klik op `Finish`.



Figuur 42: Het toevoegen van de Git submodule.

- D** Schakel met het knopje rechtsboven in CCS over naar het CCS Edit perspective, zie [figuur 34](#). Verwijder de bestanden `buffer.h` en `buffer.c` uit het project `temperatuur_logger_6_10`, zie [figuur 43](#).



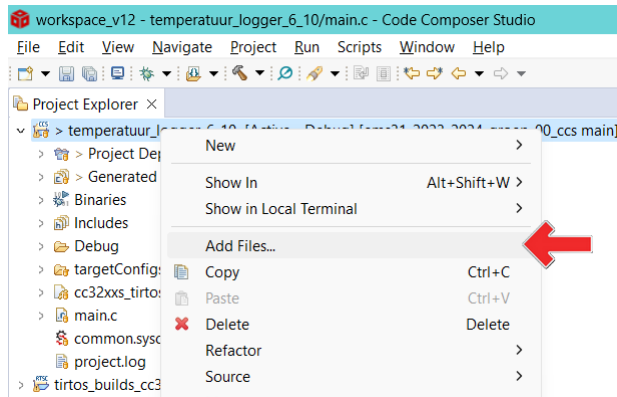
Figuur 43: Verwijder `buffer.h` en `buffer.c`.

- E** Open het bestand `main.c` en include het bestand `buffer.h` uit de Git submodule, zie [figuur 44](#).

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

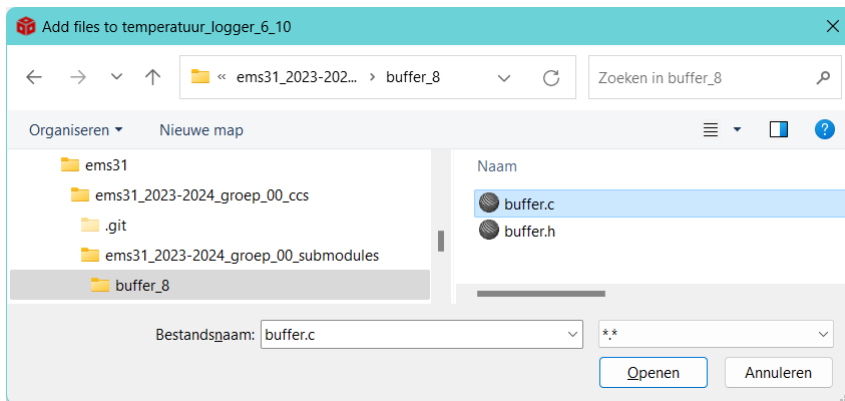
Figuur 44: Include `buffer.h` uit de Git submodule.

- F** Klik met de rechtermuisknop op het project `temperatuur_logger_6_10` en kies voor `Add Files...`, zie [figuur 45](#).



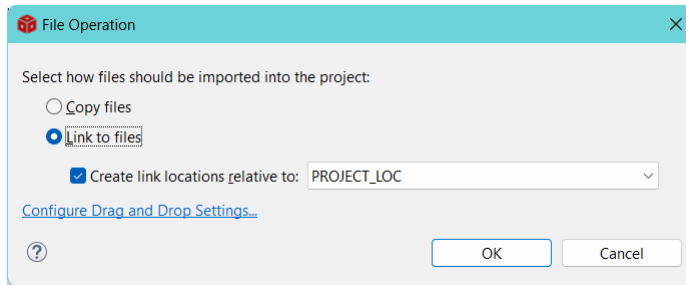
Figuur 45: Voeg bestanden toe aan het project.

G Selecteer het bestand `buffer.c` uit de Git submodule en klik op **Openen**, zie [figuur 46](#).



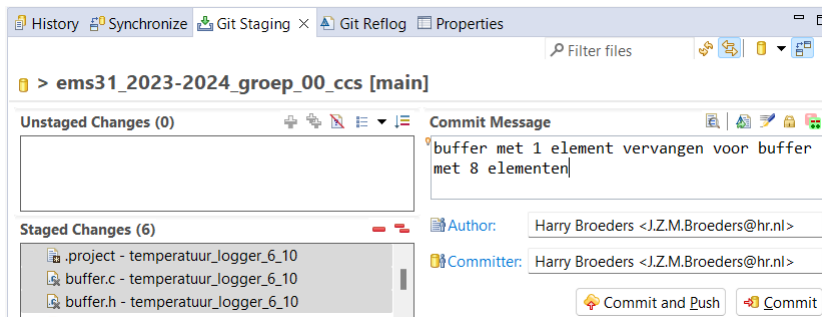
Figuur 46: Selecteer `buffer.c` uit de Git submodule.

H Selecteer “Link to files” en klik op **OK**, zie [figuur 47](#). 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 47: 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 36](#). Commit de gewijzigde bestanden en push het repository, zie [figuur 48](#).



Figuur 48: Commit en push de wijzigingen.

- J** Schakel met het knopje rechtsboven in CCS over naar het CCS Edit perspective, zie [figuur 34](#). 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 49](#).

```
Getting Started | main.c | *buffer.c ×
1 #include "buffer.h"
2 // implementation for a buffer with ints
3 // this simple buffer can only hold N ints
4
5 #define N 100
6
```

Figuur 49: 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 50](#).

```
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: 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 50: Elke seconde wordt de temperatuur gemeten.

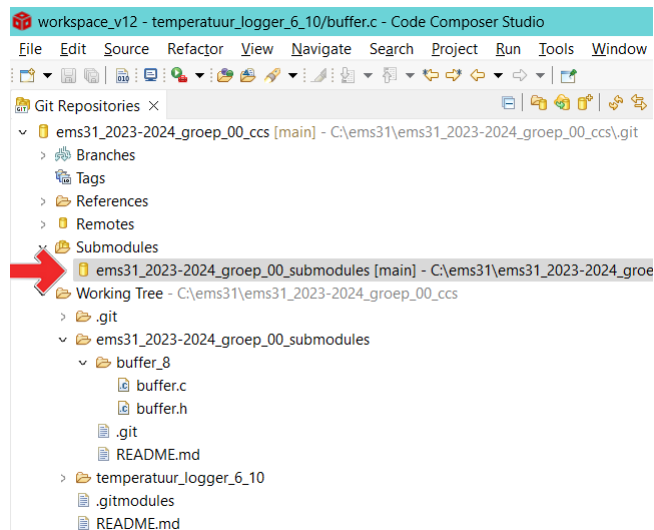
L Als je het Python programma nu, *na ongeveer een minuut*, weer opstart, dan zie je dat er geen temperatuurwaarden verloren zijn gegaan en dat alle gemeten temperatuurwaarden alsnog verschijnen op de pc, zie [figuur 51](#). Deze waarden zijn tussentijds opgeslagen in de buffer op de CC3220S LaunchPad. Hoelang kun je het Python programma stoppen voordat er wel temperatuurwaarden verloren gaan?

```
Shell <
>>> %Run ontvang_temperatures.py

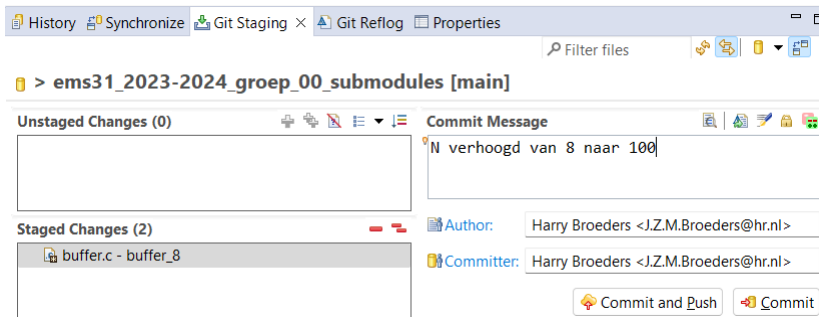
Starting ontvang_temperatures
Temperature received: 184
Temperature received: 184
Temperature received: 184
Temperature received: 184
Temperature received: 184
Temperature received: 184
Temperature received: 184
Temperature received: 184
Temperature received: 184
Temperature received: 184
Temperature 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 51: 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 36](#). Selecteer de Git submodule `ems31_2025-2026_groep_XX_submodules`, zie [figuur 52](#) en commit en push de verandering, zie [figuur 53](#).



Figuur 52: Selecteer de Git submodule.



Figuur 53: 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.