Opdrachten kwartaal 3 week 4 – Trunk-based development met Feature flags

Je bent al bekend met het versiebeheersysteem Git. Zowel Visual Studio Code als Code Composer Studio bieden ondersteuning voor Git, zie https://code.visualstudio.com/docs/sourcecontrol/overview en https://software-dl.ti.com/ccs/esd/documents/sdto_ccs_git-with-ccs.html. Bij het professioneel gebruik van Git wordt vaak een zogenaamde Git workflow gehanteerd. Een workflow die in de praktijk vaak wordt toegepast bij het ontwikkelen van embedded software is de zogenoemde Trunk-based development met Feature flags[1].

Je leert deze les hoe je:

- een FIFO-buffer als een *user defined type* kunt definiëren waardoor je meerdere buffers in een programma kunt gebruiken.
- softwareontwikkeling kan doen volgens de zogenoemde Trunk-based development met Feature flags methode;
- gelijktijdig meerdere nieuwe features voor een embedded applicatie kunt ontwikkelen.

De FIFO-buffer die je als module hebt geïmplementeerd heeft één belangrijke tekortkoming. Je kunt per applicatie slechts één buffer gebruiken. Het zou natuurlijk veel mooier zijn als je dynamisch zoveel buffers zou kunnen aanmaken als je nodig hebt. Zo'n herbruikbaar FIFO-buffer waarvan je er zoveel kunt aanmaken als je wilt, noemen we een *user defined type* (UDT). Om dit mogelijk te maken moet de interface van de buffer aangepast worden.

Een bruikbare interface is gegeven in buffer.h:

```
#ifndef _HR_BroJZ_buffer_
#define _HR_BroJZ_buffer_

#include <stdbool.h>

// interface for buffers with int's

typedef struct buffer_tag *buffer;

// create a new buffer
// returns NULL on failure
```

[1] Zie eventueel: https://developer.harness.io/docs/feature-flags/get-started/trunk-based-development/.

```
buffer new_buffer(void);

// delete a buffer pointed to by pb
void delete_buffer(buffer *pb);

// put value i in buffer b if buffer b is not full
// returns true on success or false otherways
bool buffer_put(buffer b, int i);

// get value from buffer b and writes it to *p if buffer b not empty
// returns true on success or false otherways
bool buffer_get(buffer b, int *p);

// returns true when buffer b is full or false otherways
bool buffer_is_full(buffer b);

// returns true when buffer b is empty or false otherways
bool buffer_is_empty(buffer b);

#endif
```

Merk op dat een buffer geïdentificeerd wordt met behulp van een parameter van het type `buffer`. Dit type is een pointer naar een **`struct`** `buffer_tag` maar dit type is niet gedefinieerd in de file `buffer.h`. De gebruiker van de UDT buffer hoeft dus niet te weten hoe een buffer opgeslagen en geïmplementeerd wordt.

3.4.1 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". Kopieer het directory `buffer_dyn` naar een nieuwe directory `buffers_dyn`. Kopieer het directory `buffer_dyn_test` naar een nieuwe directory `buffers_dyn_test`. Pas de testcode in het bestand `test_buffer_dyn.cpp` aan zodat getest wordt dat er een onbepaald aantal FIFO-buffers aangemaakt en weer verwijderd kan worden. Het benodigde geheugen moet dynamisch worden aangemaakt als een buffer wordt aangemaakt en weer worden vrijgegeven als een buffer wordt verwijderd. In elk FIFO-buffer moet een onbepaald aantal getallen opgeslagen kunnen worden. Het geheugen moet dynamisch worden aangemaakt als een getal in de buffer wordt geplaatst en ook weer netjes vrijgegeven worden als een getal uit de buffer wordt verwijderd. Let er op dat het geheugen ook correct wordt vrijgegeven als een buffer dat nog getallen bevat wordt

verwijderd. Vergeet niet om alle wijzigingen (regelmatig) te commiten en te pushen naar de remote repositories.

3.4.2 Pas nu de code in het bestand `buffer.c` aan zodat er een onbepaald aantal FIFO-buffers aangemaakt en weer verwijderd kan worden. In elk FIFO-buffer moet een onbepaald aantal getallen opgeslagen kunnen worden. Het type `struct buffer_tag` zou bijvoorbeeld als volgt gedefinieerd kunnen worden:

```
struct buffer_tag {
    buffer_element *first;
    buffer_element *last;
};
```

Er kan geen buffer meer aangemaakt worden en geen element meer aan een buffer worden toegevoegd als er geen geheugen meer beschikbaar is.

3.4.3 In het bestand `demo.c` is een demoprogramma gegeven waarin twee FIFO-buffers gebruikt worden. Voer dit programma uit met jouw implementatie van de UDT buffer op de pc. In het bestand `demo_CC3220S.c` is een versie van dit programma gegeven dat onder TI-RTOS kan draaien. Voer dit programma uit met jouw implementatie van de UDT buffer op de CC3220S LaunchPad. Je moet hiervoor een nieuw CCS project aanmaken.

In het vervolg van deze les ga je de temperatuurlogger uit week 3.3 verder uitbreiden. Daarbij ga je de Git workflow "Trunk-based development with feature flags" gebruiken. Code Composer Studio (CCS) heeft echter geen command line interface voor Git.

Er zijn twee manieren om de Git repositories die je in CCS gebruikt te beheren:

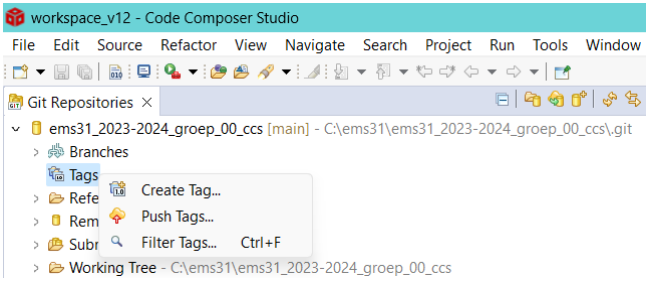
- Gebruik de in CSS ingebouwde versie van EGit[2]. Je maakt in dit geval gebruik van menu-opties en dialoogvensters om de repositories te beheren. In de onderstaande opdrachten zijn enige screenshots toegevoegd om je op weg te helpen.
- Installeer Git for Windows[3] en gebruik de command line interface van Git. Je kunt dat doen door het directory waarin het repository is opgenomen te openen in Visual

[2] Zie evertueel: https://github.com/eclipse-egit/egit/wiki/User-Guide.

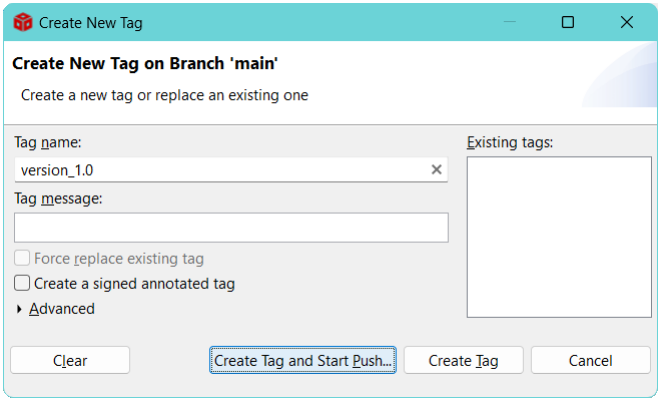
[3] Zie: https://gitforwindows.org/.

Studio Code of in Git BASH. Je maakt in dit geval gebruik van de commando's die ook gebruikt worden in de presentatie van deze week.

3.4.4 Open CCS en tag het project `temperatuur_logger_6_10` uit week 3.3 als versie 1.0. Zie eventueel figuren 1 en 2.

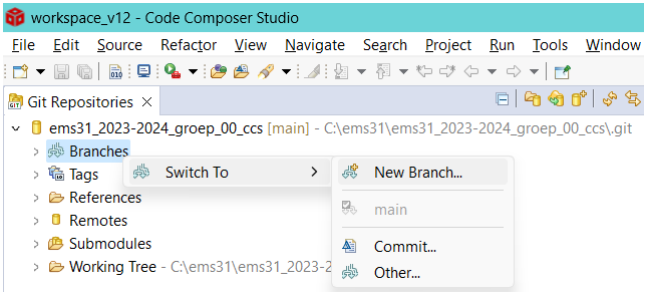


Figuur 1: Toevoegen van een tag in CCS.



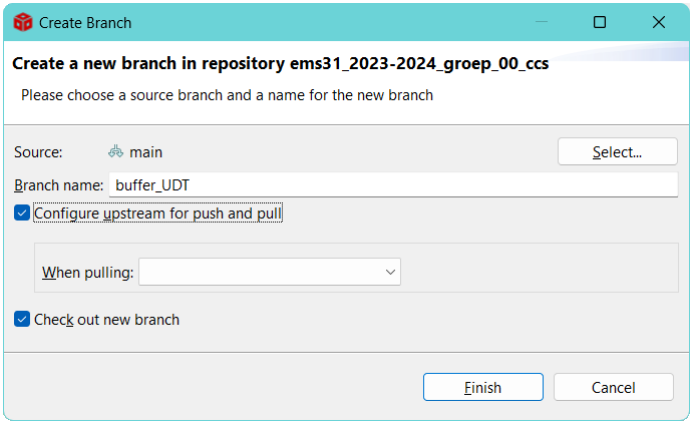
Figuur 2: Zorg dat de nieuwe tag ook naar de remote repository gepusht wordt.

3.4.5 Maak een feature branch aan om de buffer module die in week 3.3 gebruikt werd te vervangen door de buffer UDT die je zojuist hebt ontwikkeld. Zie eventueel figuur 3.



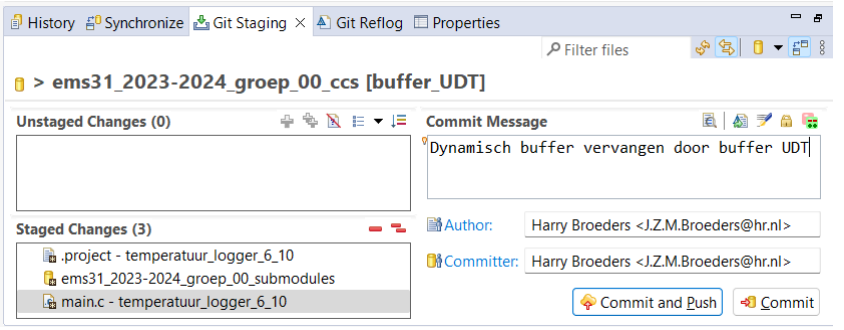
Figuur 3: Aanmaken van een branch in CCS.

Als je “Configure upstream for push and pull” aanvinkt, dan wordt de branch ook naar de remote repository gepusht (bij de volgende push), zie eventueel figuur 4.



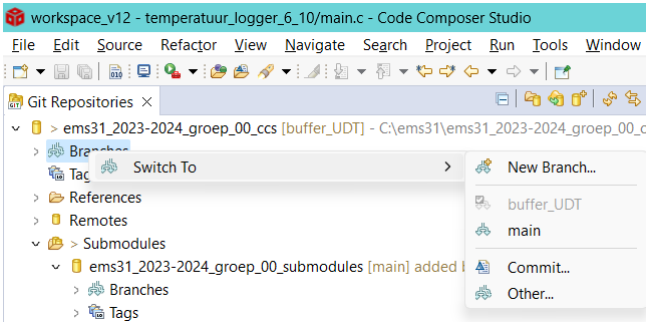
Figuur 4: Zorg dat de nieuwe branch ook naar de remote repository gepusht wordt.

3.4.6 De aanpassingen om de buffer module die in week 3.3 gebruikt te vervangen door de buffer UDT die je zojuist hebt ontwikkeld commit en push je op de zojuist aangemaakte feature branch. Zie eventueel figuur 5.

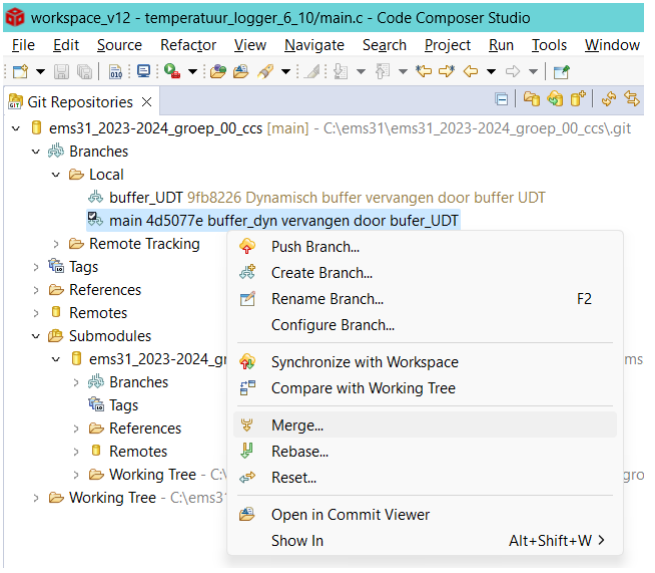


Figuur 5: Commit en push de aanpassingen op de feature branch.

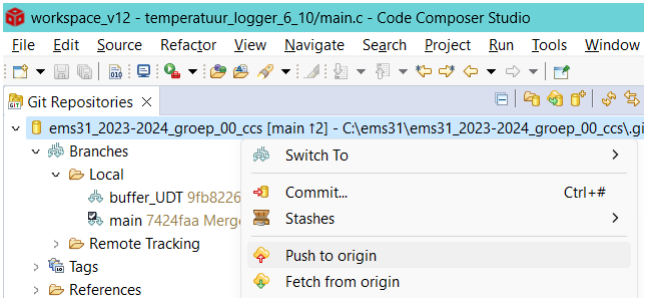
Als de feature correct werkt, dan kan deze code in de main branch opgenomen worden Zie eventueel figuren 6 tot en met 8.



Figuur 6: Switchen naar een andere branch in CCS.



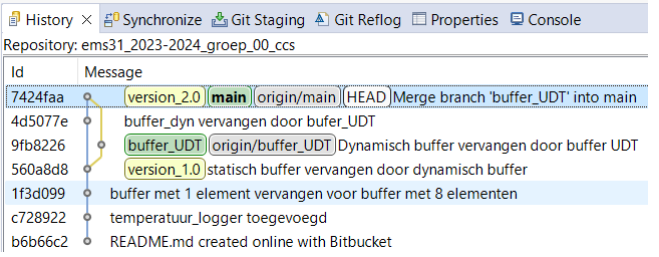
Figuur 7: Merge in CCS.



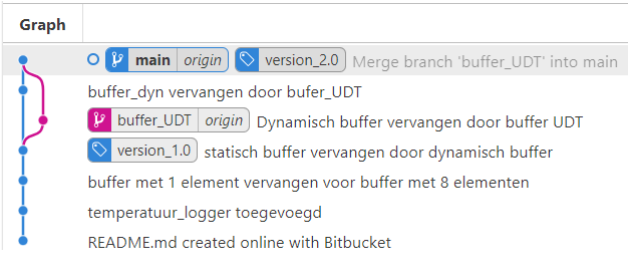
Figuur 8: Push in CCS.

3.4.7 Tag deze versie als versie 2.0.

Als je opdrachten 3.4.4 tot en met 3.4.7 correct hebt uitgevoerd dan ziet je Git history er ongeveer zo uit als gegeven is in figuren 9 en 10.



Figuur 9: Git history in CCS.



Figuur 10: Git history in Visual Studio Code.

De temperatuurlogger kan nog op minstens 3 punten verbeterd worden:

- Het is nu onduidelijk op welk tijdstip een temperatuur gemeten is. Zorg ervoor dat aan elke gemeten temperatuur een timestamp wordt gekoppeld. De timestamp moet het tijdstip weergeven dat de meting uitgevoerd is, uitgedrukt in seconden sinds de start van het programma. Dit kan geïmplementeerd worden met behulp van twee functies uit TI-RTOS `Seconds_set` en `Seconds_get`. Deze functies zijn gedeclareerd in `<ti/sysbios/hal/Seconds.h>`.
- De berichten die met UDP over wifi worden verstuurd kunnen verloren gaan. Het programma `ontvang_temperaturen.py` moet nu na elk ontvangen UDP pakket een acknowledge terugsturen. Als binnen 0,5 s geen acknowledge ontvangen wordt, dan wordt het betreffende UDP-pakket nogmaals verstuurd. Het volgende pakket wordt pas verstuurd als het vorige pakket is acknowledged. Pakketten worden dus altijd in de juiste volgorde verstuurd.
- Als de verbinding verbroken wordt en de buffer het gehele beschikbare geheugen heeft gevuld, dan worden nieuwe temperatuurmetingen weggegooid. Dit zorgt ervoor dat als de verbinding weer hersteld wordt, de meest recente meetgegevens niet beschikbaar zijn. Het is veel logischer om zodra de buffer vol is, de oudste meeting te verwijderen en de laatste meting toe te voegen in de buffer.

3.4.8 Implementeer twee van de bovengenoemde drie verbeterpunten in twee aparte feature branches. Deze verbeterpunten kunnen door twee verschillende personen tegelijkertijd worden geïmplementeerd. Maak gebruik van de Git workflow “Trunk-based development with feature flags”.

3.4.9 Zorg ervoor dat de twee features in samenhang nog steeds correct werken. Tag de commit met beide features enabled tot slot in de master branch als versie 3.0.