Tags: #cec32x, #ws, cec32x_ws_38_WS08_intro_to_unit_test_with_Unity

WS 8. Introduction to Unity—A Unit Test Framework

Introduction

We have learned how to use the debugger of Keil to see various variables of the project to make sure the results are correct. We have also learned how to use the simple `printf` function to print out the results to see if the calculations are correct or not. It turns out that there is a better way of doing the test of various functions of a project; this is the so-called unit test. In the ideal case, all the functions are unit-tested. This way, we can run the unit tests when we change the functions to eliminate errors.

In this project, we will use the Unity framework (http://www.throwtheswitch.org/unity) to unit-test a number-swapping function and two number-sorting functions based on it. We will demonstrate the usage of Unity and then you need to develop more test cases using the Unity framework following the given examples.

For this project, you need to take a look at the following docs:

- The short intro on the project page: (http://www.throwtheswitch.org/unity). This is a must read.
- A more comprehensive intro at: https://github.com/ThrowTheSwitch/Unity/blob/master/docs/UnityGettingStartedGuide.md. This is suggested read.
- More docs: https://github.com/ThrowTheSwitch/Unity/tree/master/docs. They are good references.
- Assertion reference: https://github.com/ThrowTheSwitch/Unity/blob/master/docs/UnityAssertionsReference.md.

It turns out that there are some programmers using unit tests to drive/lead the development of the project, and this is called TDD (test-driven development).

Using TDD or not, Unity is an important tool for us.

Tasks for the workshop

Download and unzip the files. Put the unzipped project folder in your project file folder and the three files in the `unity` folder in `C:\pjct_arm\_lib\unity`. (These three files are all we need for the Unity framework.)

When you open the project in Keil, you should be able to see that there are two build targets in the project. See `cec32x_devtool_19_creating_multiple_targets_in_keil.pdf` file in the **Dev tools** section of **Modules** of the class for how to add more targets if you are interested in doing so.

You should be able to build both the **Application** and **Unit test** targets. You should be able to run both. The convention is to have a number of unit tests in the **Unit test** target. When the

functions are tested well, then you can use them in the **Application** target. You can choose the appropriate target in the selection box to the right of the **Download** icon on the toolbar of Keil.

Task 1 (20 points). Read the above online documentation to decide what Unity assertions will be used for the following test:

- Test if two `uint16_t` numbers are the same.
- Test if two `int16_t` numbers are the same.
- Test if two `uint32_t` arrays are the same.
- Test if two `int32_t` arrays are the same.

Note that you need to list these assertions in the report of this workshop to receive the points.

Task 2 (20 points). Add two more test cases to test `swap_c_1` in the function named `test_swap_c_1`. You need to study the given examples in test function `test_swap_c_1` in `testFunctions_all.c` to figure out the pattern of the two test cases and write two more test cases:

- Test if the `swap_c_1` function can swap `16` and `16` correctly.
- Test if the `swap_c_1` function can swap `-16` and `16` correctly. (Yes, we are dealing unsigned numbers; let the compiler handle the conversion of the *negative* number to its TC.)

Task 3 (20 points). Add two more test cases to test `rank_ascending_c` in the function named `test_rank_ascending_c`. You need to study the given examples in test function `test_rank_descending_c` in the `ws_unity_intro_testFunctions_all.c` file to figure out the pattern of the two test cases and write two more test cases:

- Test if the `rank_ascending_c` function can rank an unsigned array `1, 3, 5, 7, 2, 4, 6, 8` correctly.
- Test if the `rank_ascending_c` function can rank an unsigned array `-1, 3, -5, 7, -2, 4, -6, 8` correctly. Note that `-1` is the biggest `uint32_t` number.

Task 4 (20 points). Repeat the test cases of Task 3 for `rank_descending_c`.

Task 5 (10 points). Now, let us add a **fail** test case. Copy the contents of the function named `test_swap_c_1` and save the function name to `test_swap_c_1_fail`. Only keep the first four line of the function. Change the content of `a1` and `a2` to `{1, 3}`, and `{3, 2}`, respectively. You need to run the `test_swap_c_1_fail` function in the `main.c` function just before the `UNITY_END()` function. When you build and run this test, you should be able to see a failed case.

Submission of the work

(10 points)

When submitting your pdf report, please follow the assignment/submission requirements below:

- The assertions of Task 1. (The contents are evaluated as Task 1, which means its 20 points are not part of the points here.)
- Code snippets of all your new code.
- The screenshots of the `printf` results of the final running of the **Unit test** target.