#cec32xNotes, cec32x_lab_12_Lab1-Basic-Debugging-with-Keil

Lab 1. Basic Debugging with Keil MDK-ARM

Introduction

In this lab, we practice basic debugging techniques in the Keil MDK-ARM simulator by writing and debugging C functions to generate Fibonacci numbers and perform other tasks.

There are three types of basic debugging techniques we will be practicing in this lab:

- Displaying the results using the **Debug (printf) Viewer** window.
- Displaying the local variables in the **Watch** window without printing.
- Displaying the global variables in the **Memory** window without printing.

Programming and debugging always proceed in an iterative way. You need to write a few lines of code, then test it using some debugging techniques. When there are no compiler errors and the results are the same as what you expect, move on to write some more code and debug it, until the entire project is finished.

Again, we need to go between the **Edit** mode to the **Debug** mode frequently, which can be easily done by clicking the **Stop/Start Debug Session** button (or pressing Ctrl + F5).

You may want to get started from the running code of Workshop 1.

Note that when we use a HW kit, the **Debug (printf) Viewer** will not be available, and we will use other approaches to print out the results. The **Watch** and **Memory** windows still work when running the program on the HW kit.

Lab Tasks

Programming Tasks

(45 points)

The programming tasks are given below, but as indicated above, you need to use an iterative approach to proceed:

- Define the following global variables which are all `uint32_t`:
    - an an array with 10 elements named `fibonacci_array`,
    - a variable to save the Euclidean norm of the above `fibonacci_array`, named `euclidean_norm`, and
    - a variable to save the maximum value that a `uint32_t` integer can hold, named `max_of_uint32_t`.
- Write the following functions:

- `generate_fibo_array` to generate the first `N` numbers of a Fibonacci sequence, starting from `1, 1`, using the algorithm given at https://en.wikipedia.org/wiki/Fibonacci_number. The input and output are given below:
    - Input:
        - The address of `fibonacci_array`, the array used to save the Fibonacci sequence. **Note that this has to be a pointer**.
        - `N`, the number of elements in the Fibonacci sequence we need to generate, including the first two `1`s.
    - Output: `void`—we change the values of `fibonacci_array` directly in the function and hence there is no return value needed.
- `calculate_eucl_norm` to calculate the Euclidean norm of `fibonacci_array`. In this function, we need to use a local `float` variable, named `norm`. The input and output are given below:
    - Input: the same as those of `generate_fibo_array`.
    - Output: an `int32_t` variable representing the integer part of the euclidean norm. Note that we use this type of return to simplify the checking of the contents in memory later.
- `print_array` to print out the first `N` numbers of a `uint32_t` array in the **Debug (printf) Viewer** window. The input and output are, respectively:
    - Input: the same as those of `generate_fibo_array`.
    - Output: `void`.

- Write the main function to do the following:
    - Define a string `my_team` to save the names of you and your team member. Print the name using `printf("Results of Lab1 from %s.\n", my_team);`.
    - Print out the addresses of `fibonacci_array` and `euclidean_norm` in hexadecimal format. Note that you can use `0x%p` to format the printing of the address of a variable.
    - Assign `0xFFFFFFFF` to `max_of_uint32_t` and then print the address and value of it. Note that you can use `%u` to print an unsigned integer.
    - Call `generate_fibo_array` to generate the first `10` numbers of `fibonacci_array`.
    - Call `calculate_eucl_norm` to calculate the Euclidean norm of the above array with 10 numbers. The result is returned to `euclidean_norm`.
    - Call `print_array` to print out the first `12` numbers of `fibonacci_array`. (Yes, this is intentional. You should be able to see that the last two numbers are not Fibonacci numbers.)

Debugging Tasks

(45 points)

Now, perform the following three approaches of debugging in the **Debug** mode:

- Display the results in the **Debug (printf) Viewer** window. This is done in the `main` function and in `print_array`. Take screenshots that contain the printouts to be used in the report. This is **Report artifact 1**. Check `cec32x_devtool_22_running_debugging_a_program_in_Keil_sim.pdf` if you need a refresh of how to print in the Keil simulator.
- Display local variables `my_team` and `norm` in a **Watch** window. To open a **Watch** window, click the triangle beside the **Watch Windows** button and select **Watch 1**. Before running the program, select and drag the name of the above two variables to `<Enter Expression>` in the **Watch 1** window. To display `my_team`, you should set a breakpoint on the line immediately after the definition of `my_team` to halt the program there to display it, a local variable. Note that the local variables can only be displayed in *scope*. To display `norm`, you should set a breakpoint in function `calculate_eucl_norm` just before the `return` statement to halt the program there and display `norm`. Take screenshots that contain the above two local variables and save them in your lab report. This is **Report artifact 2**.
- Display global variables in the **Memory 1** window after running the entire program. Display the memory starting at the address `0x2000 0000` using the **Decimal** and **Unsigned Int** format. (These sections can be made in the context menu which can be opened by right-clicking in the memory window.) Take screenshots that contain all global variables. This is **Report artifact 3**.
- Reverse the order of definitions of the three global variables in the source and obtain **Report artifact 4** in the same way as obtaining **Report artifact 3**. (Can you see which address corresponds to which variable?)

Report

(10 points)

The focus of this lab is the programming and the debugging. Yet, we need to see the report to assess your work. The 10 points for the report is for the format only. You will lose the points of the lab tasks if your results are not correct or the screenshots are missing.

The following are the requirements for the report:

- Include **Report artifact 1** and indicate the values of `fibonacci_array[4]`, `euclidean_norm` and `max_of_uint32_t` from it.
- Include **Report artifact 2** and determine the values of `norm` and `my_team` from it.
- Include **Report artifact 3** and clearly identify the values of `fibonacci_array[4]`, `euclidean_norm`, and `max_of_uint32_t` from it.

Compare these values with those from Report artifact 1.

- Include **Report artifact 4** and comment on the change of the addresses of these variables in the memory.

NOTE. You are suggested to team up with another student work on the lab, but **you need to submit your own lab report for this lab**. This is a good way to accumulate basic debugging skills. (For some other labs, you can submit a single copy of lab report and code for each team, as indicated in the Lab info.)