Exercise 3: Debugging

Lab IoT

Philipp H. Kindt

Assistant Professorship for Pervasive Computing Systems (PCS)
TU Chemnitz

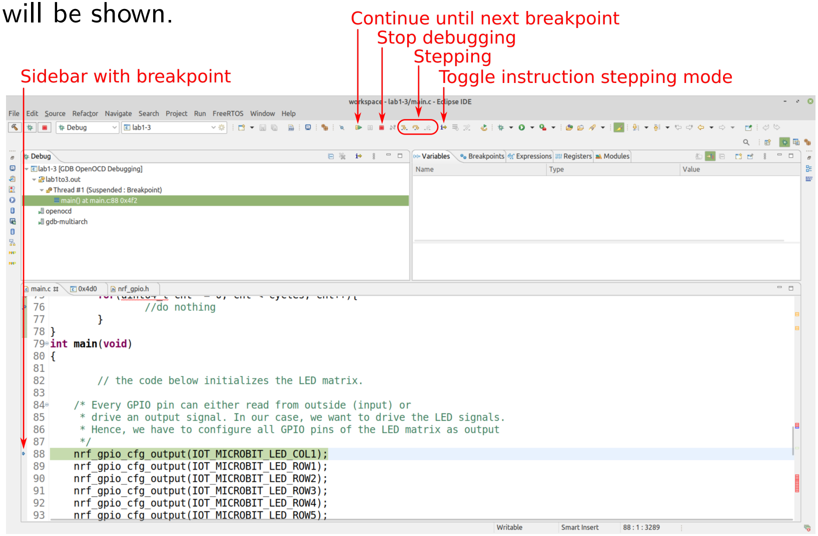
November 11, 2021

Goals

- The goal of this exercise is to make yourself familiar with debugging embedded systems in Eclipse and beyond.
- You will learn how to step through code line-by-line, how to examine variable values during runtime and how to modify them.
- It will involve setting up the debug configuration, setting breakpoints, stepping through the code, etc.
- The code you will examine in the debugger will already be given - no changes needed.
- Your task will be to “look inside” the code execution on the hardware using using different software tools for debugging.

The Debugging Screen

Once you start a debugging session in Eclipse, the following screen will be shown.



Continue until next breakpoint
Stop debugging
Stepping
Toggle instruction stepping mode
Sidebar with breakpoint
workspace - lab1-3/main.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run FreeRTOS Window Help
Debug
lab1-3
Debug
lab1-3 [GDB OpenOCD Debugging]
lab1to3.out
Thread #1 (Suspended : Breakpoint)
main() at main.c:88 0x4f2
openocd
gdb-multiarch
Variables
Breakpoints
Expressions
Registers
Modules
Name
Type
Value
main.c
0x4d0
nrf_gpio.h
Writable
Smart Insert
88 : 1 : 3289

```
    for(uint64_t cnt = 0; cnt < cycles; cnt++){
            //do nothing
        }
}
int main(void)
{

        // the code below initializes the LED matrix.

    /* Every GPIO pin can either read from outside (input) or
     * drive an output signal. In our case, we want to drive the LED signals.
     * Hence, we have to configure all GPIO pins of the LED matrix as output
     */
    nrf_gpio_cfg_output(IOT_MICROBIT_LED_COL1);
    nrf_gpio_cfg_output(IOT_MICROBIT_LED_ROW1);
    nrf_gpio_cfg_output(IOT_MICROBIT_LED_ROW2);
    nrf_gpio_cfg_output(IOT_MICROBIT_LED_ROW3);
    nrf_gpio_cfg_output(IOT_MICROBIT_LED_ROW4);
    nrf_gpio_cfg_output(IOT_MICROBIT_LED_ROW5);
```

The Debugging Screen (2)

- **Continue button:** The code is executed until the next breakpoint occurs.
- **Stop debugging button:** Stop the debugging session.
- **Stepping:** Execute the currently highlighted code. 3 flavors:
  1. Step Into: If the highlighted code is a function call, stop before the first instruction of the function.
  2. Step Over: If the highlighted code is a function call, stop before the first instruction after the control flow has returned from the function.
  3. Step Return: Stop before the first instruction after returning from the **current** function.
- **Sidebar:** Shows all breakpoints. Right click -> "Toggle Breakpoint" can be used to create a new breakpoint.
- **Toggle instruction stepping:** When the *instruction stepping mode* is active, each debugging step (e.g., "Step Into") concerns one assembly instruction in spite of one line of C-code.

Task Instructions

Let us start a debugging session. Towards this, please carry out the following steps.

- Click "Run -> Debug Configurations"
- Select "Lab1-3" and make yourself familiar with the options
- Click "Debug"
- Eclipse will now flash the program to the board, run it and stop at the first line of code in `main()`
- Use the "Step Into" button to step into the function `nrf_gpio_cfg_output()`
- Which value does the function parameter `pin_number` have?
- Make yourself familiar with the different stepping modes.
- Create a new breakpoint at line 76 and run your code (using the green arrow) until this breakpoint.
- How can you reduce the number of loop iterations in the `delay_cycles()` function? (Hint: manipulate `cnt`).

Underneath the Hood: Debugging Using the Console

- The eclipse debugger is just a graphical front-end to the GNU Debugger (GDB)[1].
- GDB itself relies on the Open On-Chip Debugger[2] to communicate with the Micro:bit board (more precisely, with the CMSIS-DAP).
- We now debug the the same code using these tools directly, i.e., without using Eclipse.

[1] `https://www.gnu.org/software/gdb/`

[2] `https://openocd.org/`

Debugging using OpenOCD + GDB

- Open a terminal.
- Run openocd using the following command: `openocd -f interface/cmsis-dap.cfg -f target/nrf52.cfg`
- openocd has now connected to the Micro:bit and listens on TCP port 3333 to communicate with other programs, e.g., GDB.
- The next step is running GDB. For this purpose, open another terminal and go to the `exercises/lab1-3` folder.
- Run GDB by typing *gdb-multiarch*.
- GDB now awaits commands to control the debugging.

Debugging using OpenOCD + GDB (2)

- To connect to open-OCD, type `target remote localhost:3333`.
- To tell GDB which file to debug, type `file build/lab1to3.out`.
- To load the program onto the processor, type `load`.
- Set a breakpoint at line 88 (i.e., the first line in main) using `break main.c:88`.
- Run the program till the breakpoint by typing `continue`.
- Step thorough it using `step`.
- `bt` will give you additional information about the functions currently being executed.
- `info frame` will show you the entire stack belonging to the current function.

Debugging using OpenOCD + GDB (3)

- After stepping through a few instructions, type `continue` to continue the program execution.
- Type `break` to interrupt the program execution again. It will (most likely) stop in `delay_cycles()`.
- How can you again set cycles to a lower value? Hint: type `help` to get a list of supported commands.
- `q` will terminate GDB. CTRL+C will cause a termination of openocd.
- Open again the eclipse IDE. Click “Run -> "Debug Configurations”. Go through the debug configurations. You should now understand (almost) all options under the “debugger” tab.