Exercise 1

In our first exercise we learn how to use the ARM Keil µVision5 Integrated Development Environment and create our first blank project template that can be used in the following exercises. We also test the Nordic Semiconductor's nRF52 development board by flashing a simple demo program to it. If you want to do the exercises on your laptop you have to install the IDE on your computer following the instructions found on the course webpage. All exercises use the evaluation version of Keil which is free of charge with few limitations e.g. a 32 kB program space limitation. This shouldn't be a problem in our exercises. (The course utilizes only Keil and its own C compiler but feel free to try out the open source alternative with GCC ARM compiler and Eclipse IDE with ARM plugins if you're interested.)

Installing the Software Development Kit

The latest SDK from Nordic Semiconductor is 15.1 and is used thorough this course. Note that different SDK versions are usually not compatible with each other. The SDK can be downloaded from here. Unzip the folder wherever you like. Open the directory and observe its contents. The most important here are *components* which include most of the drivers and libraries, and *examples* which includes a variety of useful example projects as well as a blank template project.

Running blinky

Navigate to *<SDK>\examples\peripheral\blinky\pca10040\blank\arm5_no_packs* and open the µVision5 project file in Keil IDE. Connect your nRF board to the computer via USB. Compile and flash the program to the board. You should see the four LEDs flashing in a pattern.

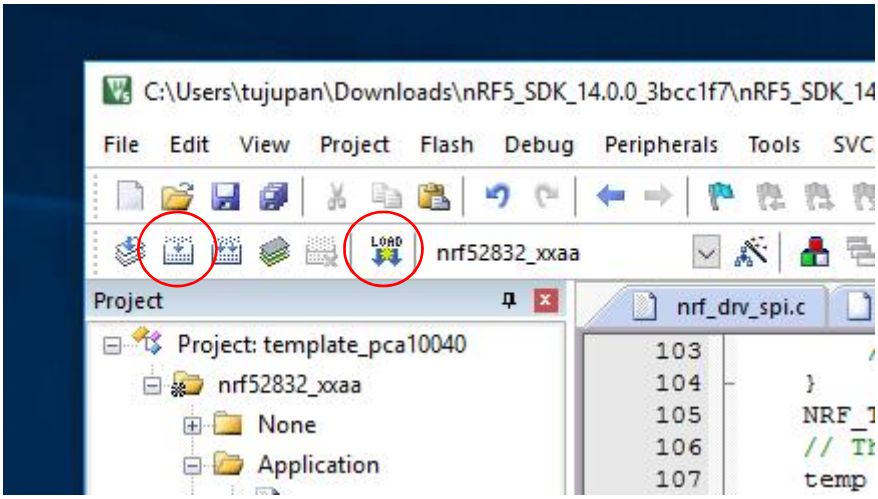

Figure 1: Compile and Flash buttons in Keil µVision5 IDE

Debugging

Using the debugger is a vital part of embedded development. Start the debug view by clicking the debug button. Make sure you have successfully compiled your project.

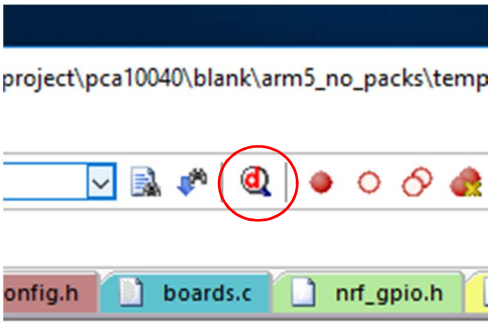

Figure 2: Start debugger

When the debug view has opened, you will see a yellow arrow pointing to your main function. It indicates the line of execution. You can run the code by clicking on the Run button or by pressing F5 or you can run the code line-by-line using the stepping tools.

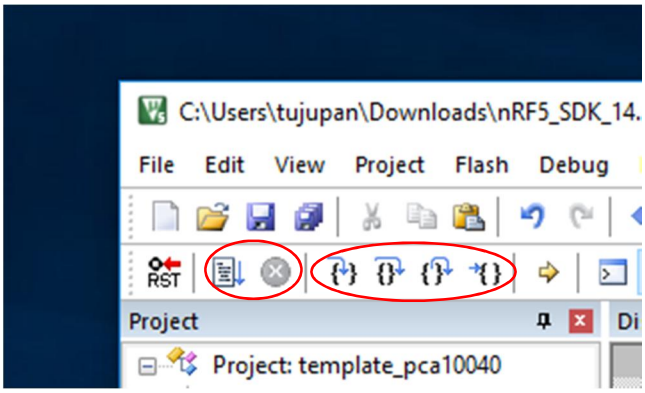

Figure 3: Run/Stop and step-by-step code execution

You can insert a limited amount of breakpoints to the code by clicking on the dark grey areas on the left of your code. Execution will stop on these points until you press Run again. You can also add variables to the watch window by right clicking them and selecting *add 'variable_name' to…* so that you can observe its status during execution. Try to insert a breakpoint so that you can observe how the loop runs by monitoring the 'i' variable.

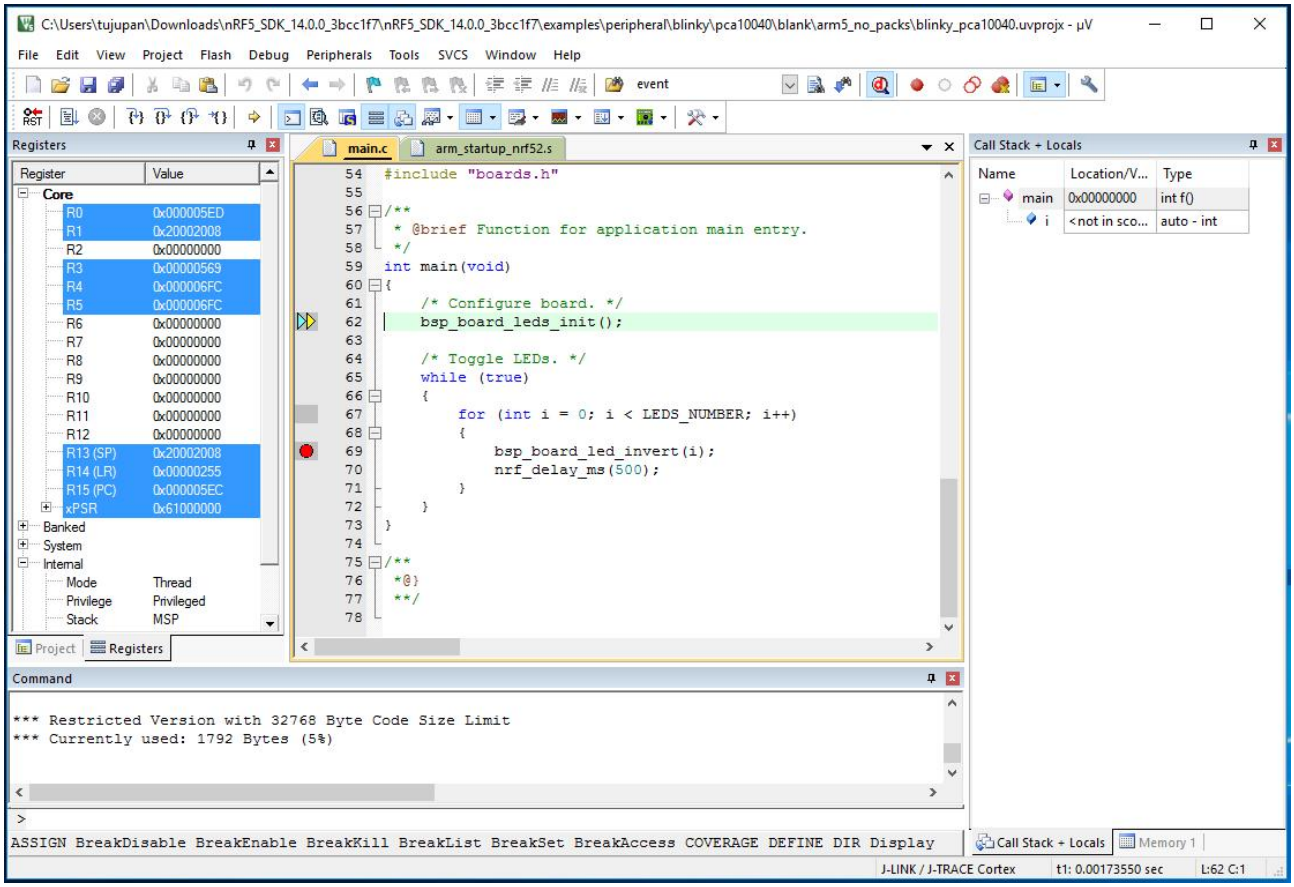

Figure 4: A breakpoint inserted to a for loop and variable 'i' showing in the watch window.

Project template

Next we prepare a blank project template that can be used in exercises to come. Navigate to *<SDK>\examples\peripheral\template_project\pca10040\blank\arm5_no_packs* and open the µVision5 project file. If you want the template to stay intact, you can create a copy of the *template_project* folder. The SDK's template project has a ton of drivers and libraries already configured, and you can disable unnecessary parts yourself to minimize the program size.

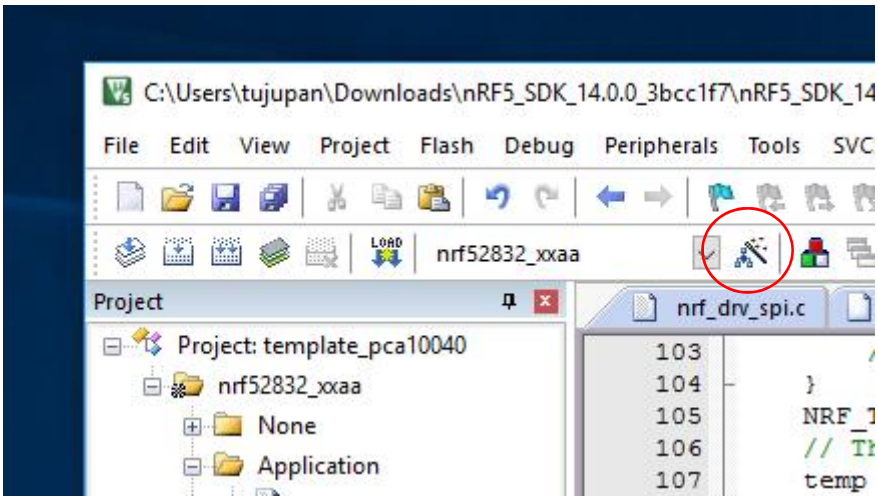

Figure 5: Options for Target

Next thing to do is to make sure that µVision5 has the right configurations for our microcontroller. Click on the Options for Target button in Keil. This opens a configuration window. Now open the nRF52 Product Specification for reference. On Device tab check that the correct processor is in use (Our board has an nRF52832_xxAA microcontroller). On Target tab, check that clock oscillator frequency is correct and refer to the specification sheet for the correct RAM and ROM options. Also make sure that ARM's *microLIB* is in use for better code performance.

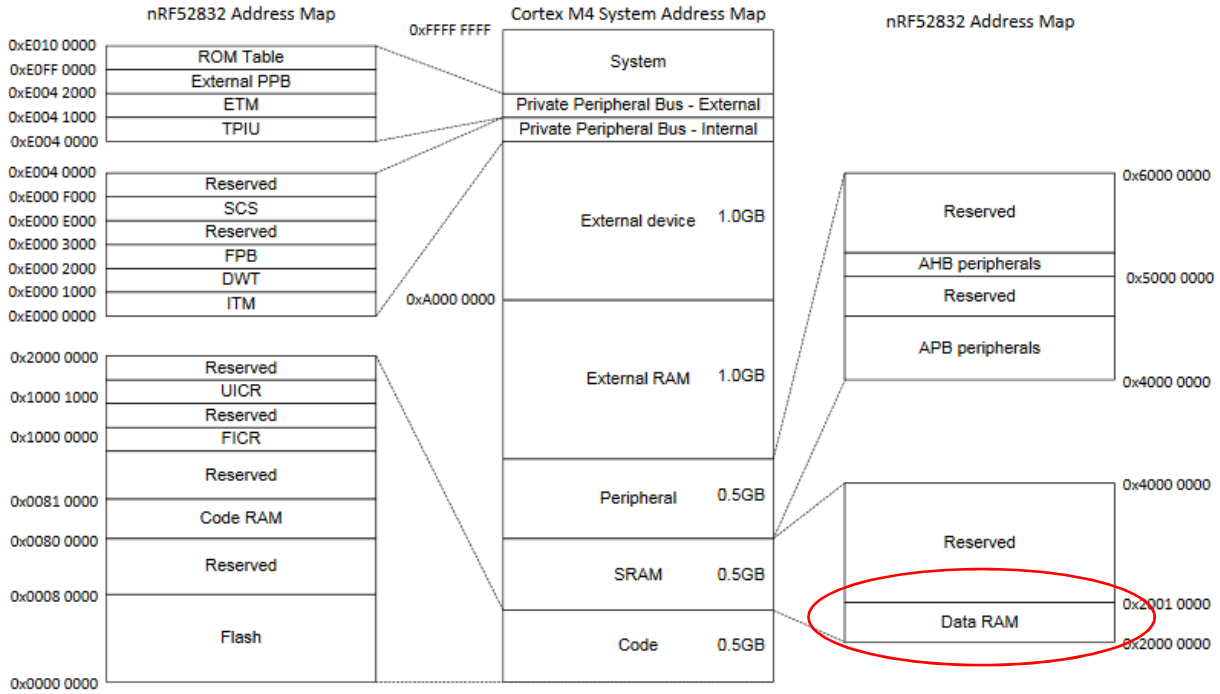

Figure 6: nRF52 memory map

Next click on the C/C++ tab. There are a couple of important features here. Optimization level can be set from –O0 to –O3, former being the least optimized and latter being highly optimized. This affects radically to the size of the program but also debugging capabilities. Include paths includes all the necessary library and driver paths assigned to the preprocessor, so each *#include* statement should have a corresponding path here. An important fact to realize is that we use a cross compiler which means our target platform uses different architecture than our development platform.

Under Debug tab we choose the right debugger hardware. Our evaluation board has an on-board *Segger J-LINK* debugger IC. Click on Settings and you should see the serial number matching the one on the board. Also check that we use a software debug device with a max frequency of 5 MHz instead of standard JTAG. Under SW Device you should see our processor's debug module (*ARM CoreSight*).

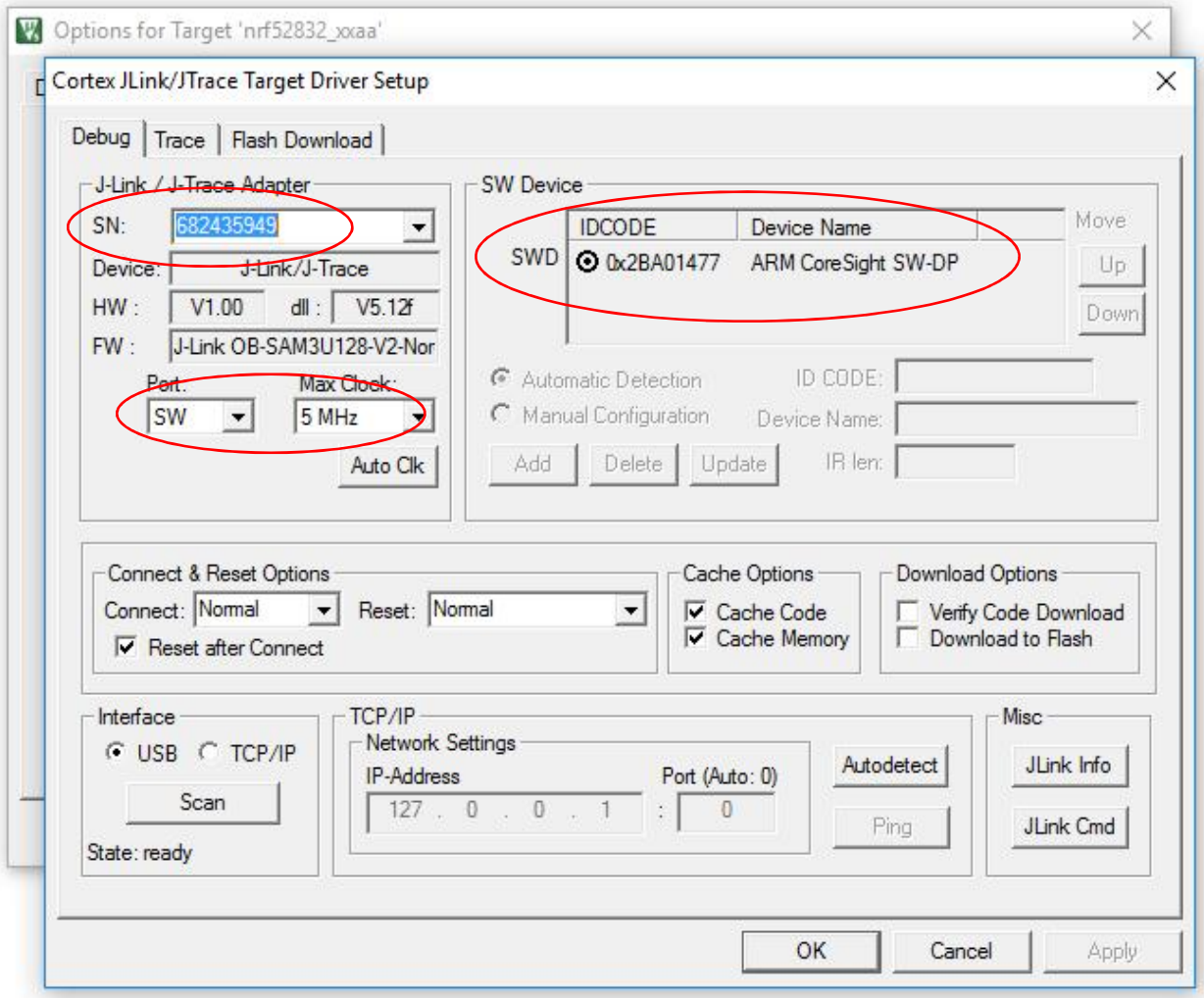

Figure 7: Debugger options

Then open *sdk_config.h* header file and observe how each library and driver must be enabled here in order to use them in your code. Remember to disable the modules not in use to reduce program size.

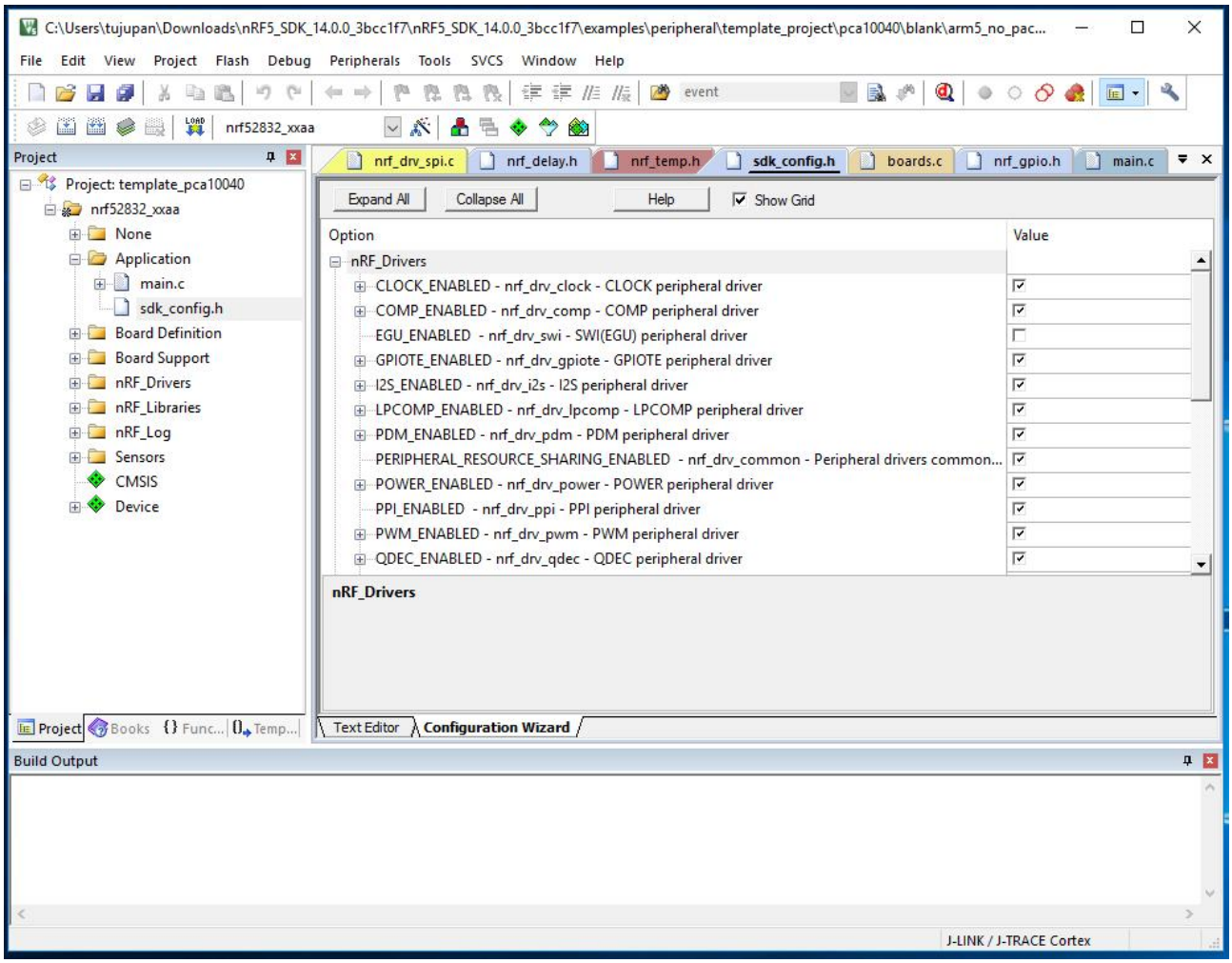

Figure 8: sdk_config.h browsed in Configuration Wizard view

If you want to add your own libraries and drivers click on Project -> Manage -> Project Items and add the .c files to an existing folder or create a new one. Note that you also have to declare these to the preprocessor in your code and in Options for Target as shown above.

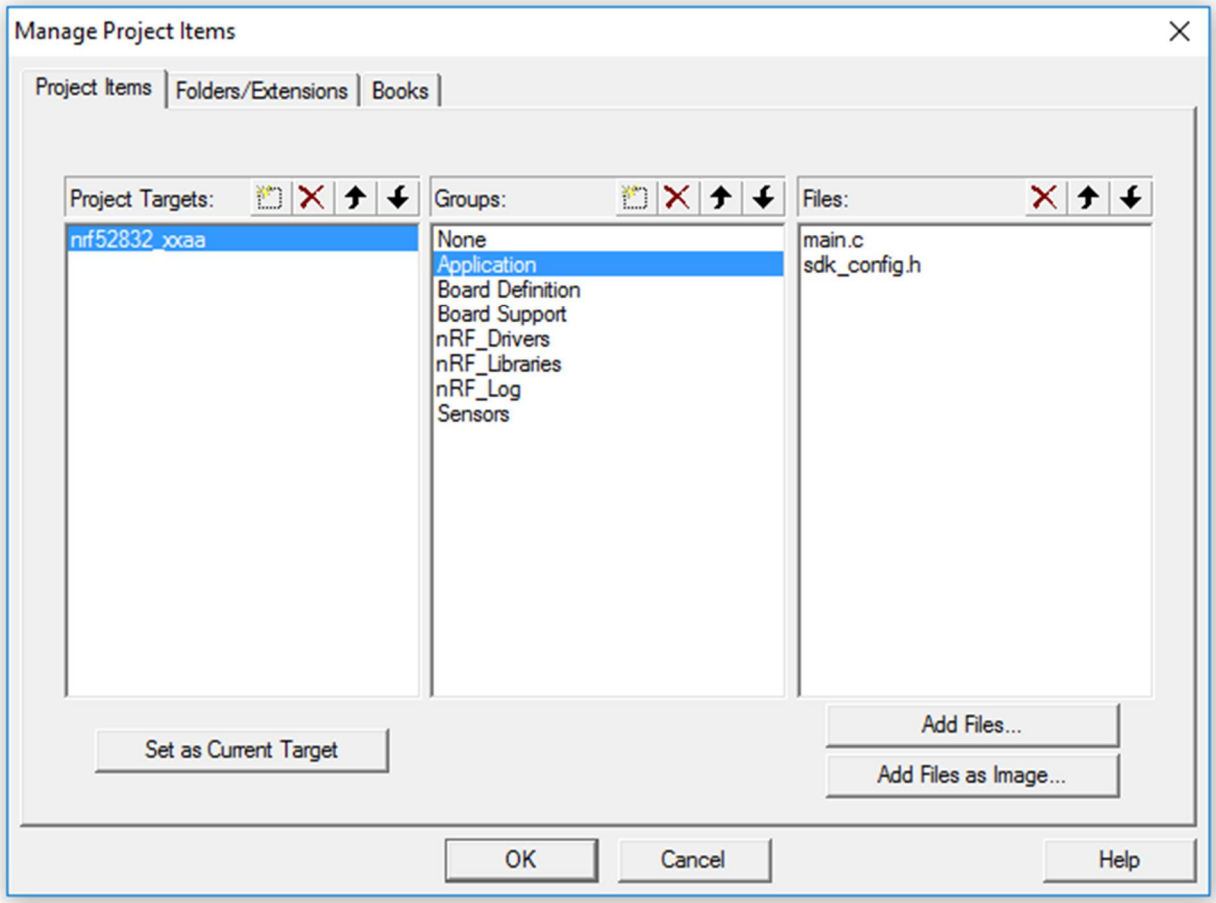

Figure 9: Project item view

Compiling

Try to compile the code and check how large the resulting .hex file is. Note that the template project should be larger than the previous blinky app because of its extra libraries.