Introduction to STM32 Nucleo

Prof. Bruno Korst - bkf@comm.utoronto.ca

Introduction

This lab will introduce you to the 32-bit Nucleo platform from ST Microelectronics that you will use for the labs in ECE342. You will also learn how to use the different software tools you will need to connect to the Nucleo board. This lab will also teach you how to write good embedded software, which can be very different from other software you have written for other courses.

1. Getting started

The first step is to download and install the software you will need for the labs in ECE342. Specifically, you will need:

1. **Keil μVision** : The main IDE you will use to download programs to and debug the STM32-Nucleo platform.
2. **ST-LINK Drivers**: The drivers needed by the Nucleo platform to connect with your computer.
3. **Hercules SETUP utility**: This program will let you see messages sent from your Nucleo board on a connected computer.

Both of these programs are already installed and available for you to use in the DESL lab room computers. However, we **strongly recommend** you still download and install them on your own computer so you can work on the labs outside of scheduled lab sessions. Let's look at how you can get both of these programs.

1.1 Keil μVision

Keil μVision is a popular IDE used in industry to work with embedded systems. We will be using the Keil **Microcontroller Development Kit (MDK)** software to program the ARM-M4 CPU on the STM32 Nucleo board. MDK uses the Keil μVision Integrated Development Environment (IDE).
Go to this website and fill in the required details. This is only required for Keil to gather statistics on who is using their software. What you fill in does not affect anything; you will still be able to download the software. Once you fill in the details, you will see a page titled **MDK ARM**. At the bottom, you will see *MDR537.EXE*. Download this to your computer and install Keil μVision .

NOTE: Unfortunately, Keil μVision is only available for Windows. If you have a Mac, you may try using the *STMCubeIDE* developed by ST Microelectronics, available here. While we expect it will work very similarly to Keil μVision , we cannot guarantee this and therefore the teaching team might not be able to support you if you run into issues using *STMCubeIDE*.

1.2 ST-Link Drivers

You will connect to the Nucleo board over a USB cable. For this, you will also need to install the USB drivers for the Nucleo board, which you can get from this website. **NOTE:** Make sure to download and install the drivers **BEFORE** connecting the Nucleo board to your computer. Once the drivers are installed, you may need to restart your computer for the drivers to get loaded.

1.3 Hercules SETUP utility

Unlike a regular computer, where you can use `printf` statements to debug your code, the Nucleo platform we will be using does not have a screen for display messages. We will therefore be using the *Hercules* program as our 'screen' to print messages from the board to a computer. These messages will be sent using a *Serial* communication protocol, which you will learn about in class. Download the program from the website here. *Hercules* is a standalone program, that can simply be run without having to be installed first. Open the program and click on the *Serial* tab at the top.
You will see a large window, which is your 'terminal'; messages printed from your Nucleo board will appear here. Before you can communicate, it is necessary to configure *Hercules* with the options on the right of the window. The two options you will change most often are 'Name' to select your PC's communication port (COM), and 'Baud Rate' to set the speed.

2. Running your first program

Start by downloading the sample *blinky* project provided on the course Quercus. *Blinky* is often the first program everyone runs when learning to use a new embedded platform, similar to *hello world* when you learn a new programming language. This very simple program will toggle the blue LED on the board on and off at a set rate. Double-click on *MDK-ARM/blinky.uvproj* to start Keil μVision .

The Keil μVision IDE interface: At first you will only see the *Project explorer* window on the left. This shows you the hierarchy of your entire project. As you can see, even for a very simple program, a large number of files are included in the project. Expand the *Src* folder and open *main.c*.

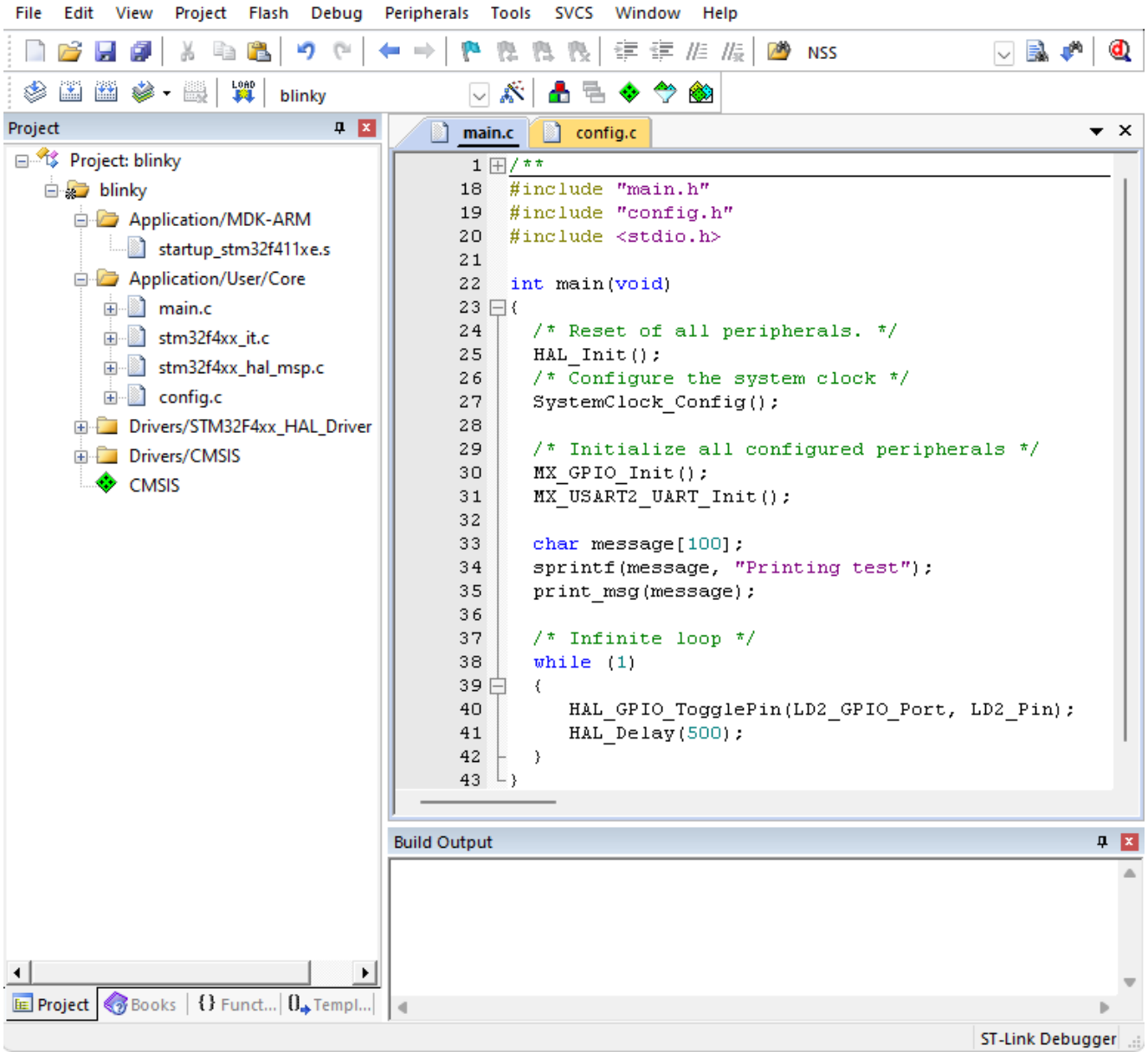

Figure 1. The Keil μVision IDE showing the *blinky* program.

Understanding the provided code: Figure 1 shows the Keil μVision IDE, with `main.c` open. After including the necessary header files, `main.c` has prototypes for functions declared later in the file. In `main()`, there are several functions with initialize different board peripherals. We won't go into the specifics of each of them but to give you an idea of what each of them does (and to give you a preview of what you will learning throughout the course), they are:

1. **HAL_init()** initializes the `Hardware Abstraction Layer` or HAL. The HAL provides easy access to the board's registers and peripherals.
2. **SystemClock_Config()**, provided further down in the code, configures the clock circuitry for the board, to provide the clock that our system uses.
3. **MX_GPIO_Init()** configures the `General Purpose Input/Output` (GPIO) pins on the board. The GPIOs are the main way for us to connect the board to external peripherals so we will be using the GPIOs function a lot this term.
4. **MX_USART_UART_Init()** configures the UART (or Serial) communication protocol that allows us to 'print' from the board to the *Hercules* utility.

All of these functions are defined in `config.c`. There is also code which creates a string, which we can use for printing. We will look at printing in more detail later.

Code inside an infinite loop: You will see that `main` ends with a `while(1)` loop. Code running on embedded systems typically uses such an infinite loop. As the programs we will run do not have an OS, there is nowhere for the *main()* function to return to. So, any code you want to run continuously should be placed inside this *while(1)* loop. For the *blinky* program, you will see two lines inside the loop.

Toggling the LED: The first line toggles the blue LED, which is labelled *LD2* on the board itself. As the LED is outside the actual chip, it is connected to our system via a GPIO. So, we use the *HAL_GPIO_TogglePin()* function to toggle the LED; if it is on, we turn it off and vice-versa. The toggle function takes two parameters: 1) The port that the GPIO is part of and 2) The specific GPIO pin connected to the LED. Both of these are provided as *#defines* in the *main.h* file.

Adding a delay: Since the board is running at several megahertz, if we toggled the LED every clock cycle, we would just see it being on all the time. To slow down the rate of toggling, we need to add a small delay, which we do using the *HAL_Delay()* function. This function takes in a time (in milliseconds) and causes the CPU to wait for that long before continuing.

Running on a board: To compile the `blinky` project, select *Project* > *Build Project* or press the *F*7 key. You will see the compilation output in the *Build Output* window, at the bottom of the IDE. This is where you check if you had any compile errors that you need to fix. Once this is done, you can connect your Nucleo board to your computer using the provided USB cable. To download your code to the board, select *Flash* > *Download* or press the *F*8 key. Your code should now be downloaded onto the Flash memory of the Nucleo board.
You should now see the blue LED on the board flashing at a fixed speed. Try changing the value inside the *HAL_Delay()* to change the rate of flashing. You can try running other experiments to better understand this code.

We will occasionally suggest some ideas for your to try, which will be presented in boxes like the one shown below:

> **Try it yourself:** Try flashing different Morse code patterns on the LED, similar to Lab 5 in ECE241. You can find the full Morse Code alphabet here. How can you store the sequence of dots and dashes for each letter? Can you flash a word or even a whole sentence?

2.1 Printing from the board

Now that you have run your first program, let's see how you can send information from your board to your computer. We will use the `Universal Asynchronous Receive Transmit (UART)` peripheral for printing. `config.c` already configures `UART2` for you. You will learn more about how the UART peripheral works in future labs. UART uses the 'serial' or 'COM' port to receive data sent from the Nucleo board. Many decades ago, computers often came with COM ports but modern computers do not. The Nucleo board, therefore sends Serial data over USB.
However, your computer still 'sees' this as a separate COM port. Since computers support many COM ports, you will need to find out which COM port the Nucleo board is connected to. Download the `check-com-port` app from the Lab 0 page on Quercus and run it on your computer. This should tell you which COM port the Nucleo board is connected to.
Next, open the `Hercules` program and click on the `Serial` page at the top. Configure the Serial port as follows:

1. **Name**: Select the COM port you saw above.
2. **Baud**: 38400
3. **Data Size**: 8
4. **Parity**: None

5. **Handshake**: None
6. **Mode**: Free

Matching these settings **exactly** between your Nucleo board and the Hercules program is essential. If the settings do not match, you will see garbage data when trying to print. Configure the Hercules program as indicated and click 'open'.
On the Nucleo side, you will have to add code to perform printing. In 'config.c' you will find a simple function for printing over UART, called 'print_msg'. This function takes a null-terminated string (in the form of a character array) and prints each character one by one. While you can directly print simple messages, we want to be able to print other variables (e.g., floats), the same way we do with `printf`. For this, we can instead use the `sprintf` function, which operates in the same way as `printf`, but 'prints' the message to a character array instead of the terminal (i.e., stdout). To use this function to print, you must do the following:

1. Create a array of type `char`. You can make this any reasonable size, say 100.
2. Include `stdio.h` at the top, so we can use the `sprintf` function.
3. Use `sprintf` to write to the character array.
4. Call the `print_msg` function by passing in the character array.

Exploring the project: The Keil IDE provides you with a lot of tools you can use to help write, test and debug your programs. For example, to take a look at *main.h*, you can just click on the file in the *Outline* window shown on the right.

You can also try running the program in 'Debug' mode, which allows you to set breakpoints and view register and memory values after each instruction. However, keep in mind that 'Debug' mode only affects the CPU and does not affect the behaviour of peripherals. You should be careful when using 'Debug' mode as you may not be able to reproduce a failure you see in the regular 'Run' mode, when using 'Debug' mode.