WARWICK

SCHOOL OF ENGINEERING

Module Code:
ES2C4

Module Title:
Computer Architecture and Systems

Learning Activity:
Lab. 4 ARM Assembly Programming

Learning Objectives:

- **Introduction to STM32CubeMX and Keil uVision**
- **Write basic ARM assembly programs**

Instructor:
Dr Sam Agbroko sam.agbroko@warwick.ac.uk

1. Introduction

In this lab session, you will practice programming an ARM processor in ARM Assembly Language. The tools you will be using are:

- **Nucleo-L432KC** is an STMicroelectronics development board with the STM32L432KC microcontroller unit (MCU). This MCU is based on the ARM M4 microprocessor.
    - **Datasheet**
    - **Reference manual**
- **STM32CubeMX**
    - **Download**
    - **Documentation**
- **Keil uVision**
    - **Download**
    - **Documentation**

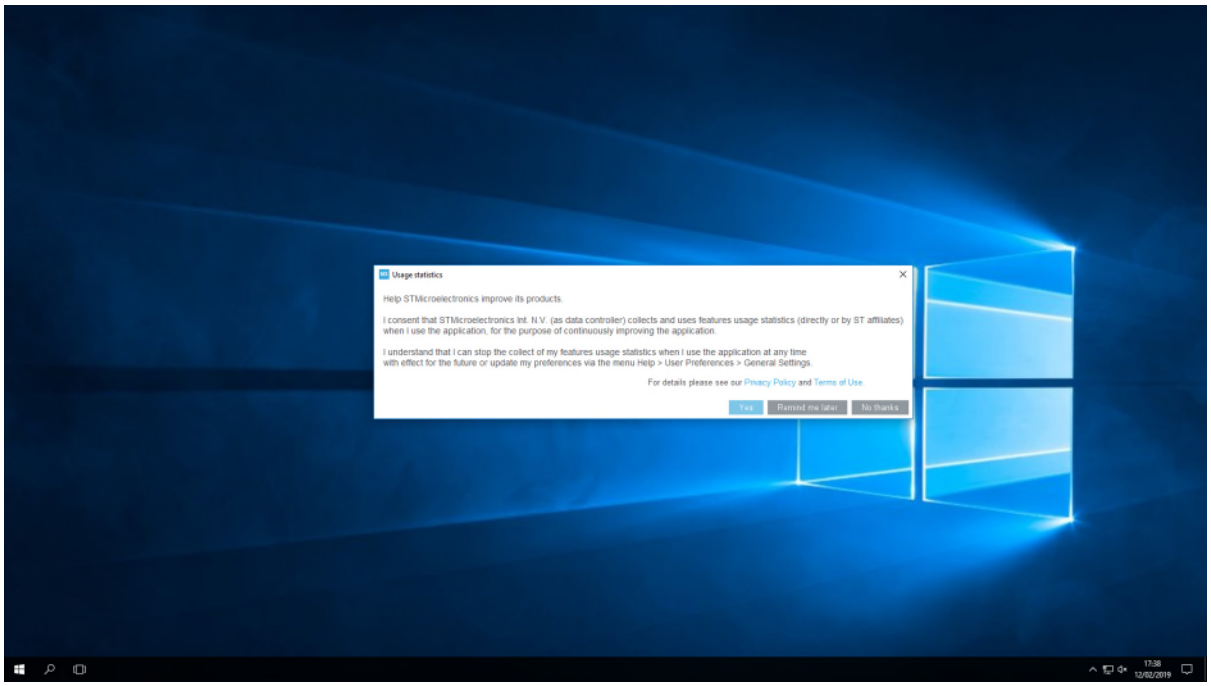
Keil uVision only runs on Windows. If you use a macOS, you could set up a virtual Windows environment on your machine using VMware Fusion.

Section 2 and 3 will guide you through setting up these tools for programming in ARM assembly language. Section 4 guides you through programming in ARM assembly language.

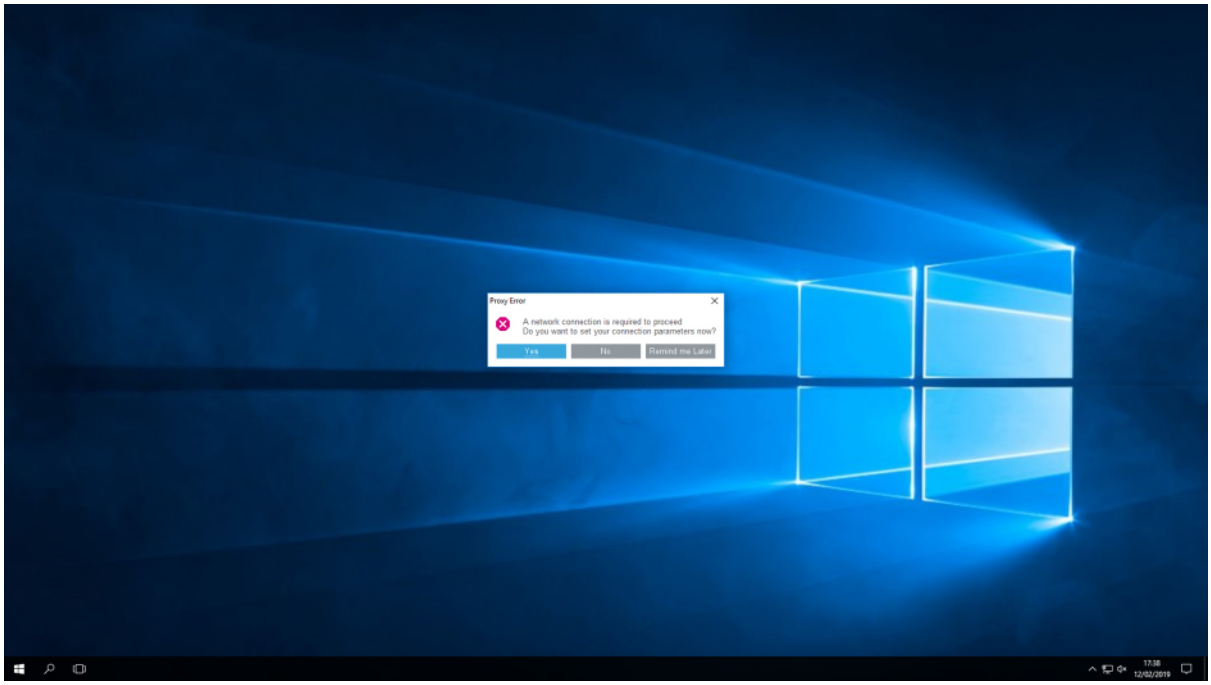
2. STM32CubeMX

STM32CubeMX is a graphical tool that allows easy configuration of STM32 microcontrollers and the generation of the corresponding initialization C code. This section shows you the steps to generate C code for your MCU STM32CubeMX.

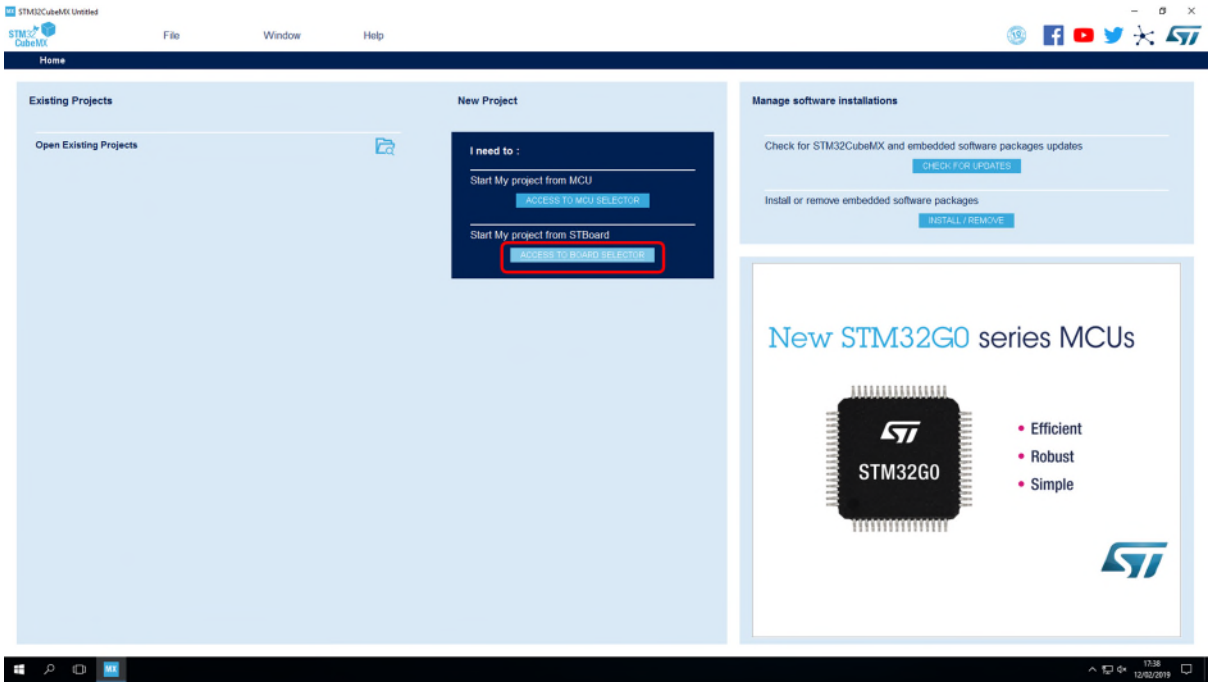
1. Click the Windows button and search for STM32CubeMX and Run the program.
2. Select an option in the dialog box.



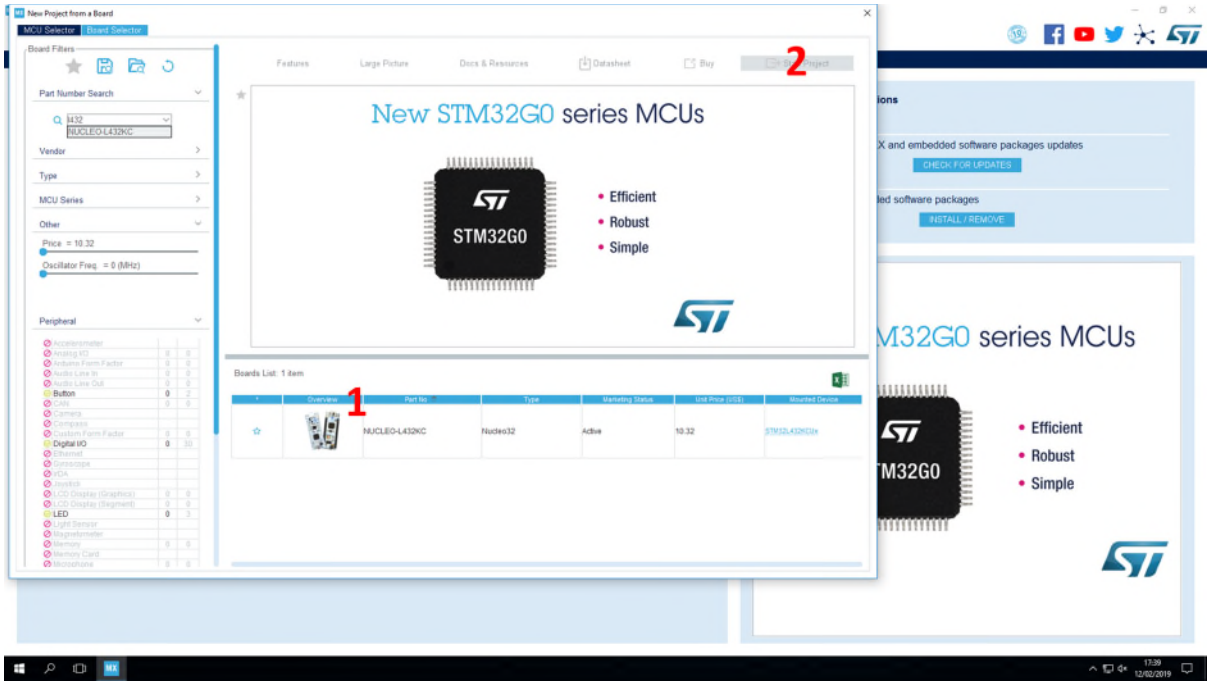
3. Click Yes to allow STM32CubeMX connect to the internet. Click OK in the next dialog box.



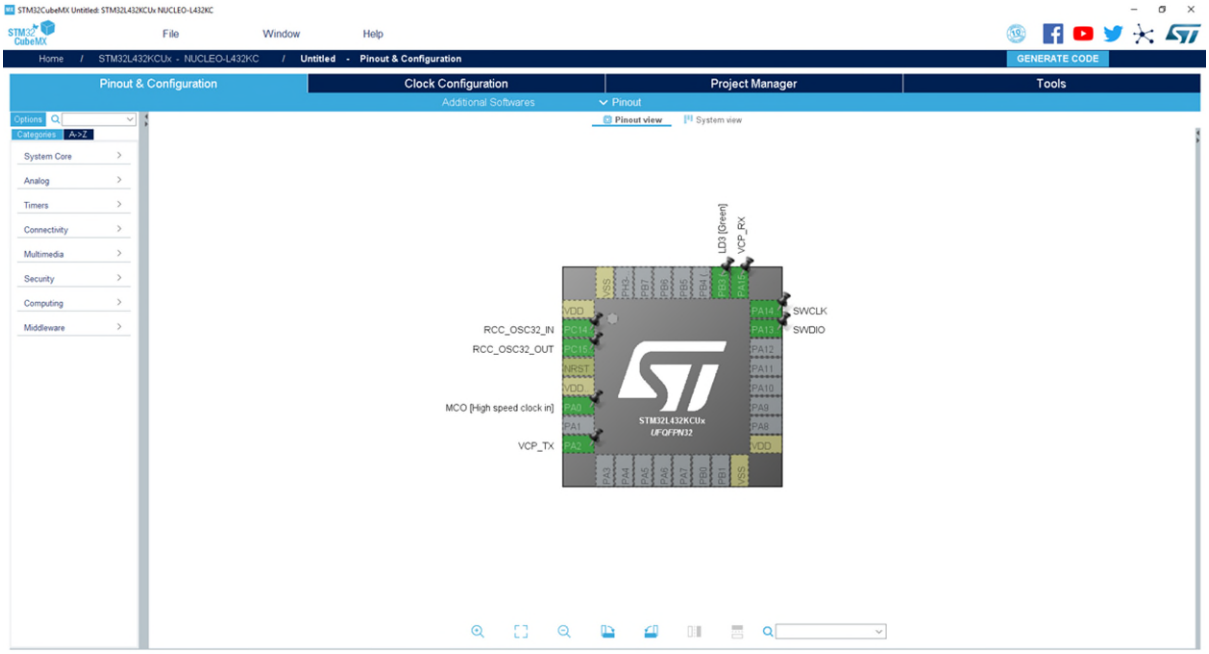
4. Click **ACCESS TO BOARD SELECTOR** to present an option to select the MCU. Next, click **OK** to download support files for the MCU.



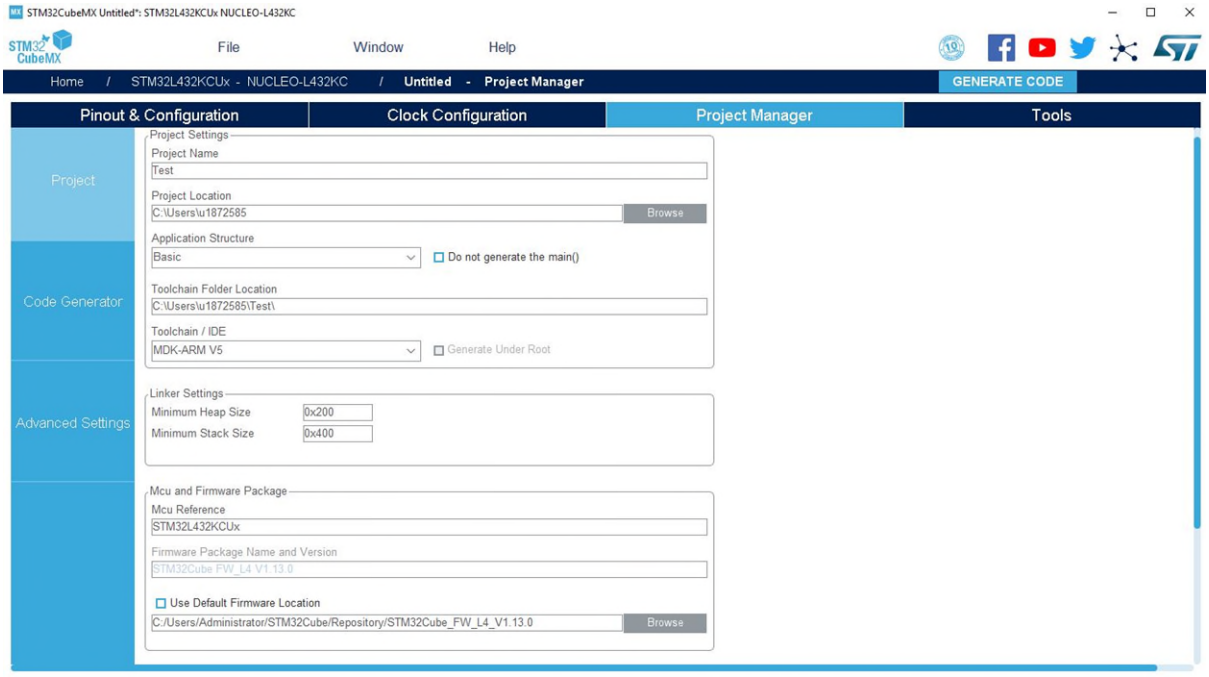
5. On the next dialog, search for the Nucleo-L432KC MCU on the **Part Number Search** textbox on the left panel. Select the MCU board in the **Boards List** (1). Next, click **Start Project** (2) at the top of the dialog. Select **Yes** on the Board Project Options dialog. This allows STM32CubeMX to configure the MCU peripherals to their default mode.



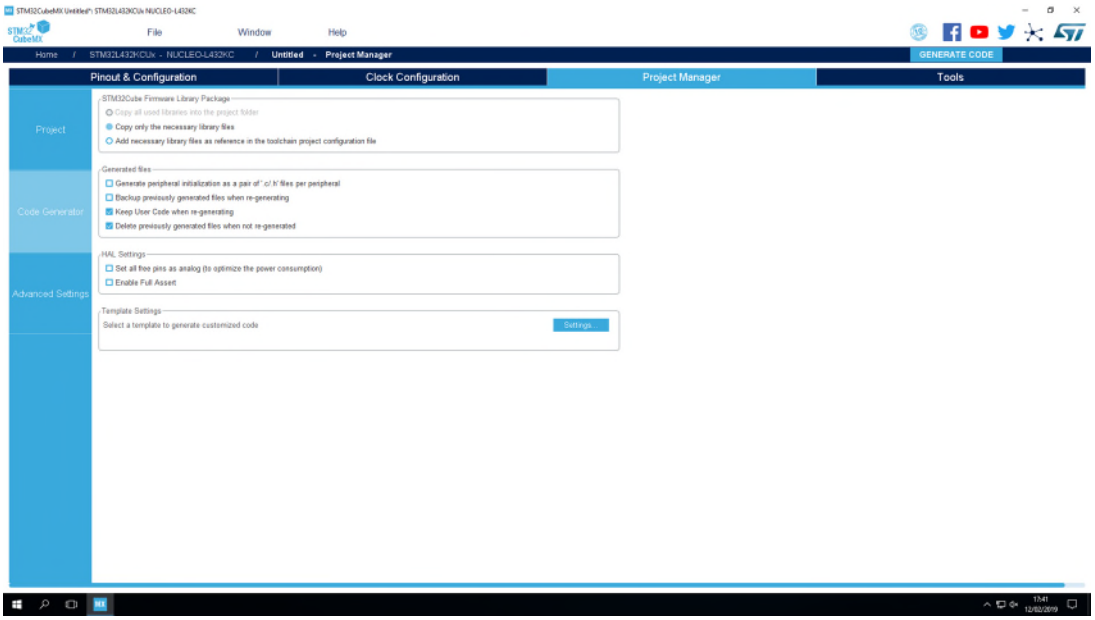
6. The STM32CubeMX is divided into four tabs. The **Pinout & Configuration**, **Clock Configuration**, **Project Manager** and **Tools** tab are located at the top of the software. Notice that the microcontroller in the **Pinout & Configuration** tab already has the pins configured. For now, this configuration is sufficient.



7. Go to the **Project Manager** tab. On this tab, you can define project parameters such as project name, folder etc. Provide a project name and select a project location (use the H drive if you are on a managed machine). From the **Toolchain/IDE** options, select **MDK-ARM**. This tells STM32CubeMX to generate C code for the Keil MDK. If you are using a Warwick managed machine, uncheck the **Use Default Firmware Location** box and click on browse and navigate to the **P:\Course Files\ES2C4\STM32Cube_FW_L4_V1.13.0** folder.

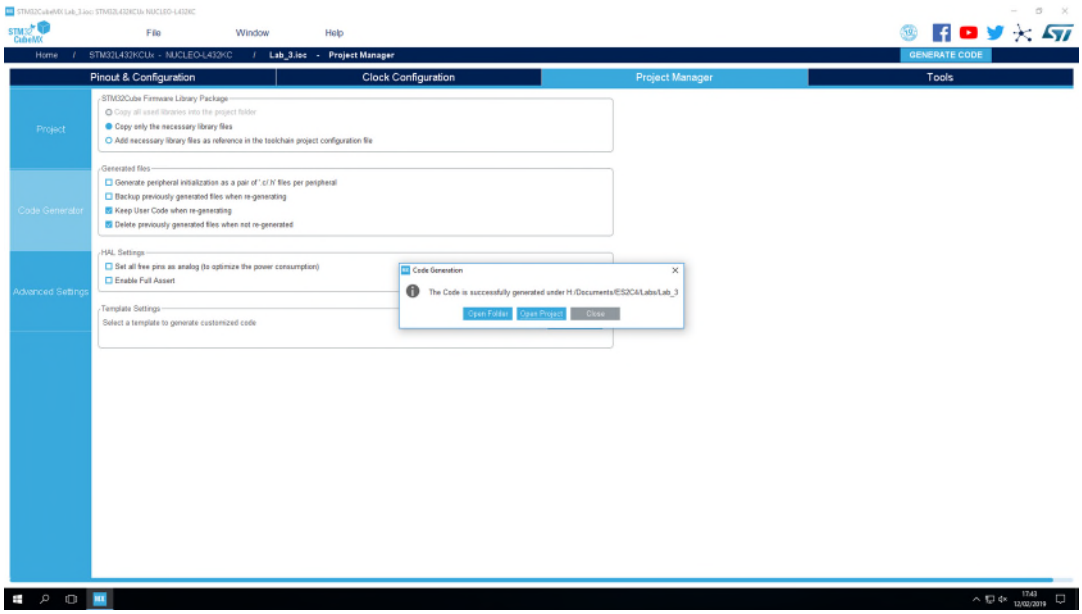


8. On the left panel, select the **Code Generator** page. Ensure that the **Copy only the necessary library files option** is selected.



9. You are now ready to generate your MCU configuration C code. Click Generate Code at the top right of the page. If asked to download the required dependencies, select Yes. After the download is complete, you will be presented with a Code Generation

popup, select Open Project. This will open the project you have just created in Keil MDK.



3. Keil uVision and MDK

The Keil uVision (IDE) and Microcontroller Development Kit (MDK) is the most comprehensive software development solution for Arm-based microcontrollers. It includes all components needed to create, build, and debug embedded applications. When Keil is opened for the first time, the **Pack Installer** window (Figure 1) is automatically opened.

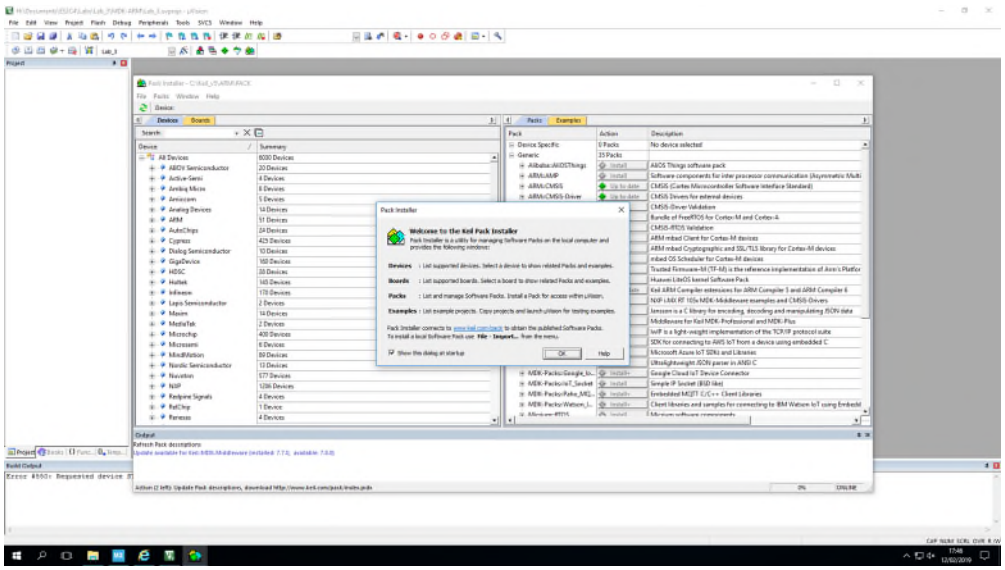


Figure 1: Keil Pack Installer window

This window allows you to download ARM drivers for specific microcontrollers. Click **OK**. If the device pack for your MCU is not installed, a popup will appear asking you to install the device family pack. Click **Install** to allow Keil to download the necessary packs for your MCU. You can monitor the progress of the installation using the status bar at the bottom of the **Pack Installer** window. Once this is process is complete, go ahead and close the **Pack Installer**.

The features of the Keil user interface (UI) in Build mode is annotated in Figure 2. On the Project window to the left of the UI, select the drop-down arrow by the **Application/User** folder and double click the **main.c** file to open it. This file is where our program resides, and it is similar to the **main.c** file you created in the Labs 1,2 and 3. Buttons 1 and 2 highlighted in Figure 2 are used to **Build** and **Rebuild** your program. Button 3 is used to load the program to the MCU. Button 4 is used to enter **Debug** mode. In **Debug** mode, you can step through your instructions and observe how it runs on the MCU.

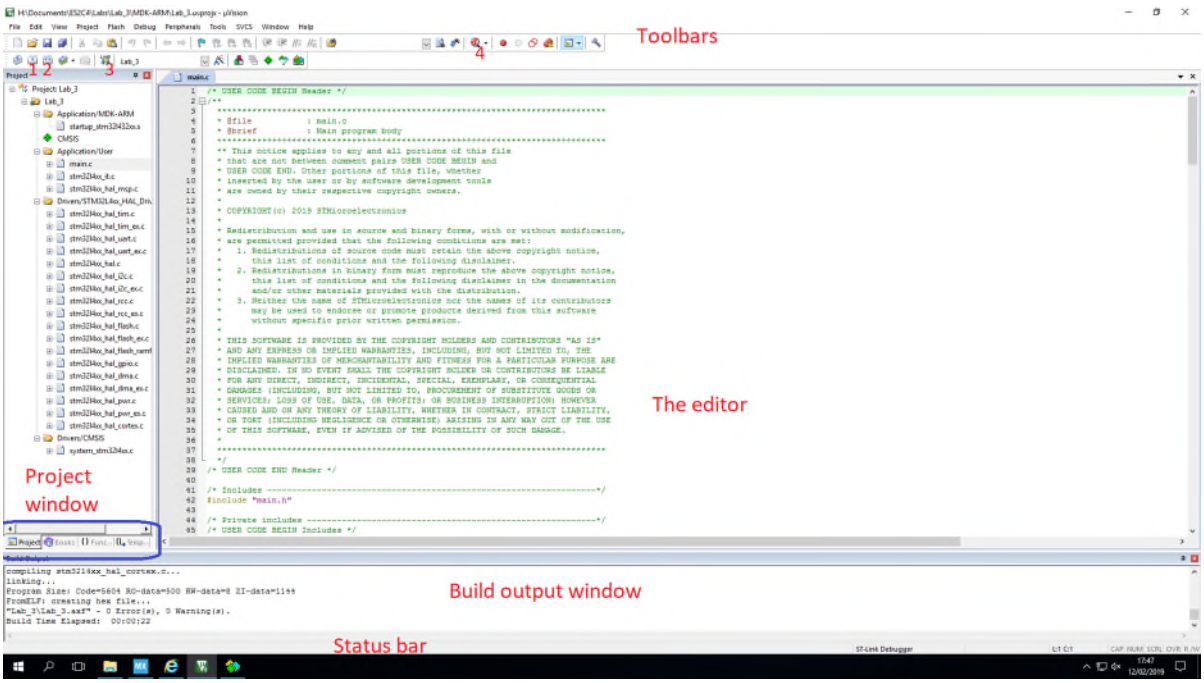


Figure 2: Keil MDK user interface

Keil sorts the files created by STM32CubeMX into four folders in the Project window. These files contain instructions for initialising your MCU. They also contain drivers for managing the functionalities of various peripherals in the MCU. You will learn more about this in the next lab session.

Right-click on the **main.c** file and select **Remove file**. You will create an assembly file to in place of the **main.c** file. Notice that the file extensions must match the programming language, hence .c for C files and .s for assembly files. To create a new assembly file, right-click on the **Application/User** folder and select **Add New Item to Group 'Application/User…'** From the options provided in the popup window, select **Asm File (.s)**. Provide a file name in **Name** textbox and remember to append *.s*. This guide uses *Assembly Language.s* as the file name. Click the **Browse** (**…**) button by the **Location** textbox and select the **Src** folder in your project folder. Click **Add** to add the file to your project.

The new assembly file is created in the **Application/User** folder. Double click on the file to open it in the **Editor**. Enter the following piece of code.

```
        AREA myProg, CODE, READONLY
        ;The AREA directive instructs the assembler to assemble a new code
        or data section.
        ENTRY
        ;The ENTRY directive declares an entry point to a program.
        EXPORT __main
        ;The EXPORT directive is used to give code in other files access
        to the symbols in the current file
__main
        MOV R0, #9 ; R0 = 9
        MOV R1, #3 ; R1 = 3
        END
```

Your editor window should look like Figure 3.

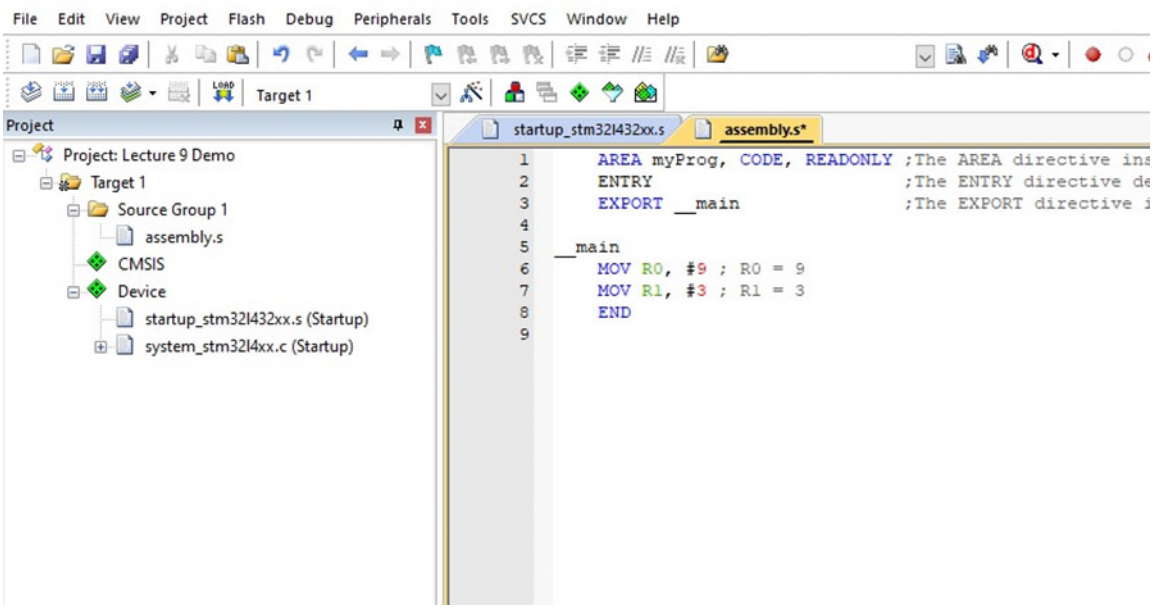


Figure 3: Keil uVision user interface with assembly code

Click on the **Build** button to compile your program. Notice the progress of this step in the **Build output** window below the **Editor**. If this step is completed without any errors, you are ready to run your ARM assembly code in the MCU.

Before loading the program unto your MCU, ensure Keil is set up for your MCU. Click on **Flash** button at the top left of the UI above the toolbar. From the drop-down menu, select **Configure Flash Tools..**. On the **Debug** tab, check that **ST-Link Debugger** is selected in the right drop-down field as shown in Figure 4 below. This informs Keil that you will be using

the ST-Link debugger on the Nucleo32-L432KC development board for programming and debugging the MCU.

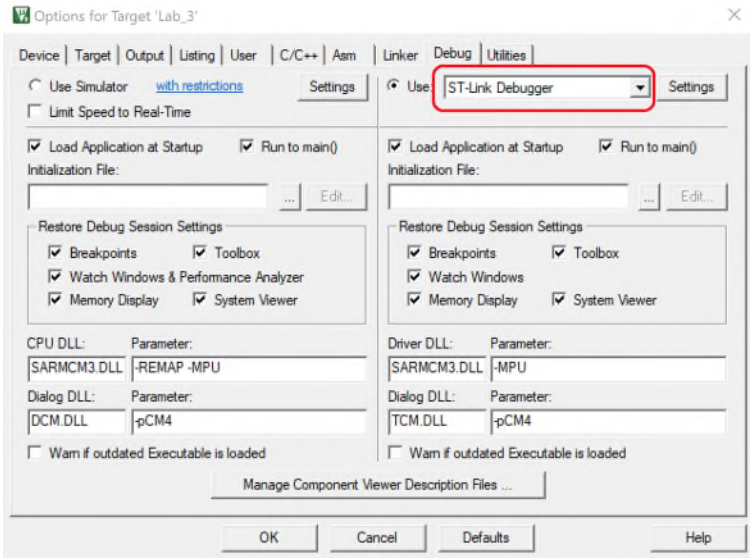


Figure 4: Selecting the ST-Link Debugger

Finally, click on the Settings button to the right of the drop-down menu. On the Flash Download tab, tick the option **Reset and Run** With this option selected, you do not need to press the reset button every time you upload a new code. Click **OK** on both dialog boxes to return to the main UI.

Now place your Nucleo32-L432KC development board on the breadboard and connect your it to the USB port on your computer. Click the **Download** button on the toolbar to load your code onto your MCU. Click the **Start/Stop Debug Session** button on the toolbar to start debugging. The UI changes into debug mode (Figure 5).

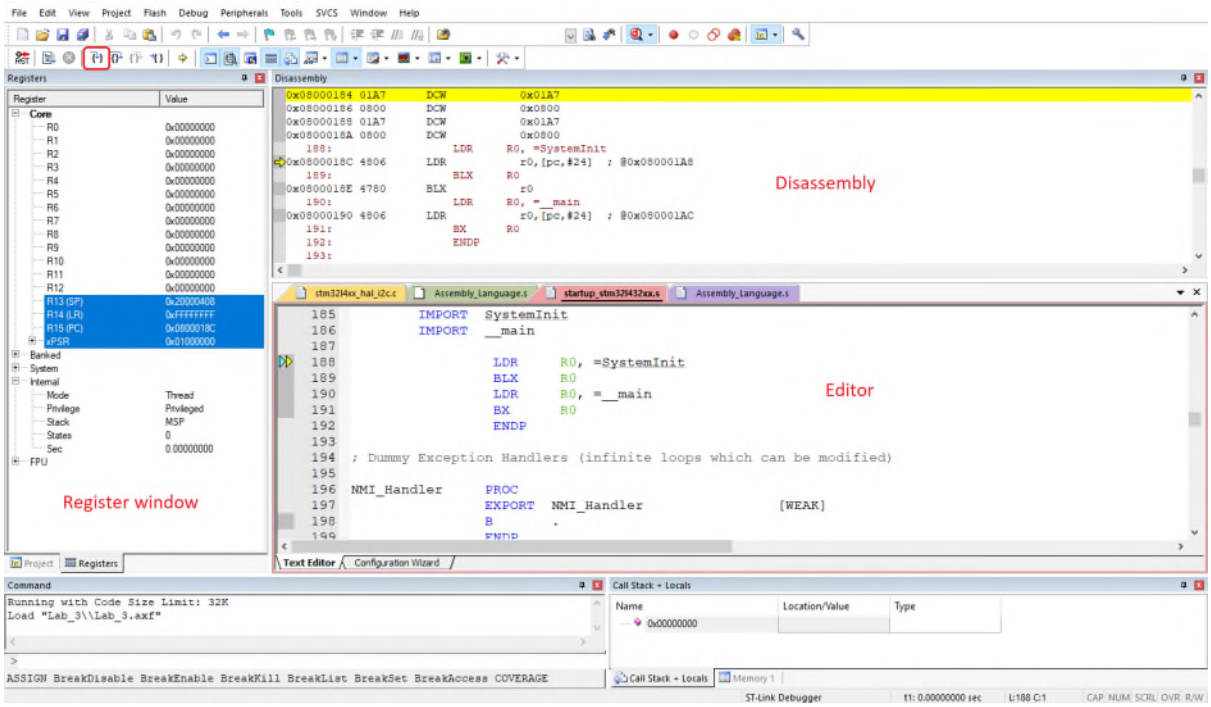


Figure 5: Keil Debug mode

In Debug mode, the ARM registers are displayed on the Register window. The **Step** button has been highlighted on the toolbar. This button is used to step through each line in your code. The Disassembly window is now visible above the editor and displays the assembly instructions and memory addresses where the instructions are stored. Notice that the Startup file is opened automatically in the editor. This file contains instructions for setting up the MCU for your use. Go to your assembly file, place the cursor just before the first `MOV` instruction, right click and select **Run to Cursor Line**. Notice that the yellow arrow now points to your move instruction in the Disassembly window and the address for this instruction has been loaded into the Program Counter (PC) in register 15. Click the **Step** button on the toolbar to execute this line of code. Notice that R0 now holds the value 9.

For the remaining tasks, you only need to change the code between the `main` and `END` words.

4. Tasks

The following assembly codes have been provided for you to explore the ARM Instruction Set Architecture as discussed in the lectures. C code is also provided to show you the high-level implementation. **Do not enter C code in assembly file!** As you **step** through each line of code, observe the values that are written to the registers. Remember to **Build** your program after each edit and upload it to the MCU.

4.1. Basic operations

The code below demonstrates basic addition and subtraction in ARM assembly language using registers.

```
          AREA myProg, CODE, READONLY
          ENTRY
          EXPORT __main
__main
          ;R1 = 4, R2 = 7
          MOV R1,  #4    ;      R1 = 4
          MOV R2,  #7    ;      R2 = 7
          ADD R3, R1, R2 ; R3 = R1 + R2
          SUB R4, R1, R2 ; R3 = R1 - R2
          END
```

4.2. Logical instructions

The code below demonstrates logical AND (AND), OR (ORR) and Move Not (MVN) logical instructions in ARM assembly language. For these instructions, the first operand must be a register.

```
          AREA myProg, CODE, READONLY
          ENTRY
          EXPORT __main
__main
          ;R1 = 0xFF, R2 = 0xF0
          MOV R1, #0xFF ; R1 = 0xFF
          MOV R2, #0xF0 ; R2 = 0xF0
          AND R3, R1, R2
          ORR R4, R1, R2
          MVN R6, R2
          ; Add an instruction to set the 3rd bit in R1 to 0
          and store the result in R7.
          END
```

4.3. Shift instructions

The code below demonstrates the shift instructions which are used to shift the value in a register right or left.

```
          AREA myProg, CODE, READONLY
          ENTRY
          EXPORT __main
__main
          ;R1 = 0xFF, R2 = 0xF0
          MOV R1, #0xFF ; R1 = 0xFF
          MOV R2, #0xF0 ; R2 = 0xF0
          LSL R3, R1, #3
          LSR R4, R1, #4
          ROR R5, R2, #2
          ; Add an instruction to set R6 to 0x08000000
          END
```

4.4. if/else instructions

This section demonstrates writing of equivalent C code in ARM assembly language.

C Code

```
          if (3 == 7)
            f = 3 + 7
          else
            f = 7 - 3
```

ARM Assembly Code

```
          AREA myProg, CODE, READONLY
          ENTRY
          EXPORT__main
_main
          ;R0 = f, R2 = 3, R3 = 7
          MOV R2,  #3    ;     R2 = 3
          MOV R3,  #7    ;     R3 = 7
          CMP R2, R3
          ADDEQ R4, R3, R3
          SUBNE R5, R3, R2
          ;Expand the Current Program Status Register (xPSR) so these
          registers
          are visible
          END
```

4.5. For loop instructions

C Code

```
//sum of numbers between 1 and 9.
int sum = 0;
for (int i = 8; i > 1; i = i - 1)
   sum = sum + i;
```

ARM Assembly Code

```
          AREA myProg, CODE, READONLY
          ENTRY
          EXPORT __main
__main
          MOV R0, #8          ;R0 = i
          MOV R1, #0          ;R1 = sum
FOR
          CMP R0, #0          ;R0 > 1?
          BLE DONE            ;if (R0 > 1) exit loop
          ADD R1, R1, R0      ;sum = sum + 1 loop body
          SUBS R0, R0, #1     ;i = i -1 loop operation
          B FOR               ;Repeat loop
DONE
          ;Write a program to add from 1 to 9 using a while loop
          END
```

4.6. Function call

Observe the addresses of instructions in the Disassembly window. Also, take note of the addresses loaded into the Link Register (R14 LR) and Program Counter (R15 PC). Record the address of the branch instruction and the addresses in the LR and PC before executing the branch instruction. Record these addresses again before and after returning from the callee to the caller function.

```
          AREA myProg, CODE, READONLY
          ENTRY
          EXPORT__main
__main
          ;Call a function to add 5 to an argument and return the result
          MOV R1, #9          ;place the argument in R1
          BL FUNC             ;call function.
          B CONTINUE          ;program continues after the function
FUNC
          ADD R0, R1, #5 ;carry out sum and place result in R0
          MOV PC, LR      ;return to the caller
CONTINUE
```

Can you explain how the instruction MOV PC, LR is used to return from the callee to the caller function?

Write a C program to carry out the function in the ARM assembly code above.