

HOMEWORK 1: INTRODUCTION TO CubeIDE

A. OBJECTIVE:

1. To familiarize yourself with the structure of ARM microcontrollers.
2. To familiarize the CubeIDE.
3. To compile programs and understanding the interface of CubeIDE.

B. READING:

1. Revise basic C programming and basic assembly programming syntax.
2. CubeIDE Website (<https://www.st.com/en/development-tools/stm32cubeide.html>).
3. Study the CubeIDE User Manual (Available from course Canvas).
4. Study the Cortex-M3 programming Manual (Available from the course Canvas).
 - Chapter 2 – The Cortex-M3 processor.
5. Install the CubeIDE Version 1.9.0. (Available from course Canvas).

C. INSTALLATION OF CubeIDE:

An IDE (Integrated Development Environment) is a software that allows you to design and test your microcontroller-based application. In this course, we will use CubeIDE from STMicroelectronics. STM32CubeIDE is an all-in-one multi-OS development tool, which is part of the STM32Cube software ecosystem. STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors. It is based on the Eclipse®/CDT™ framework and GCC toolchain for the development, and GDB for the debugging. It allows the integration of the hundreds of existing plugins that complete the features of the Eclipse® IDE.

You need to download the software from the course Canvas. It has Windows and Mac Version, please install according to your operating system.

At the beginning of the installation, it will ask you to install drivers for
SEGGER J-Link
ST-LINK

Please install both drivers as we need it to communicate with the ST-LINK Debugger.

After successful installation, you will have the CubeIDE on your Desktop



Figure 1: STM32CubeIDE icon on the Desktop

D. WORKING WITH A PROJECT

Aim:

1. Understanding the CubeIDE working environment by a Project.
2. Simulate and Debug program by running or step tracing the program.
3. View and trace the run-time values of registers, and memory during Simulation or Debug process.

Prerequisite:

1. Watch the **Video 1 – Part I : Development Environment** under Canvas → Modules → MUST Watch Videos to understand the Development Environment.
2. Watch the **Video 2 : Intro to CubeIDE** to understand the interface and basic control of CubeIDE.

Details:

1. Refer to the Video I – Part I: ELEC3300 – Development Environment on Canvas, create a Project in CubeIDE called HW1.
2. Download the main.c of HW1 from the Canvas, replace the main.c in your HW1 Project. The file will be located in: **<Your workspace>/HW1/Core/Src**
3. In main.c, **modify the content of variables and add the code required according to your student ID**. Namely

Array: stdid[8];

Variables: i, j, x, y;

Codes: swapid, oddid, sum, ANDresult, ORresult, XORresult;

Registers: R1, R2.

NO MARK will be awarded if you do not use your student ID to do HW1.

4. On Project Explorer, right click HW1, choose Properties.
In C/C++ Build > Setting, make sure the Optimization level is None (-O0). Click Apply and Close.

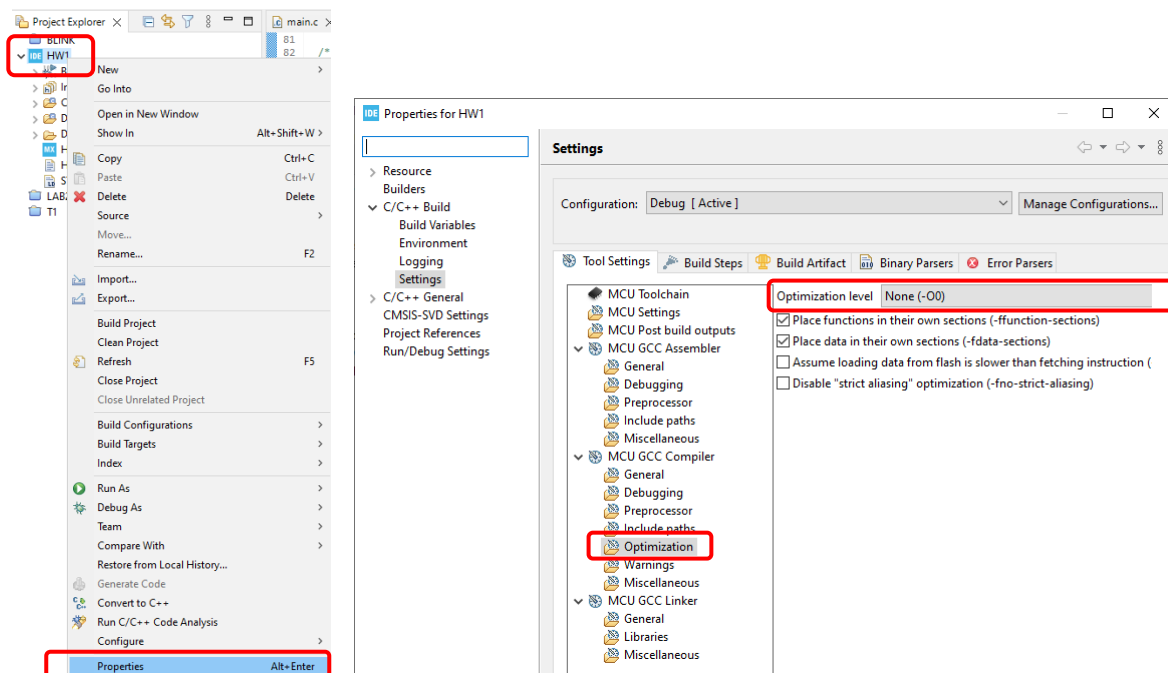


Figure 2: Change Optimization Level.

5. Go to Project → Build All to build your Project

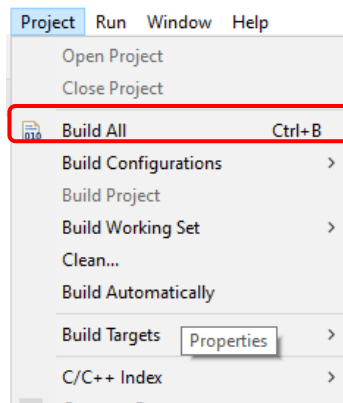


Figure 3: Build the Project

6. You will see at the Console output, it will show the status, make sure you get Build Finished. 0 errors, 0 warnings as shown in Figure 3.

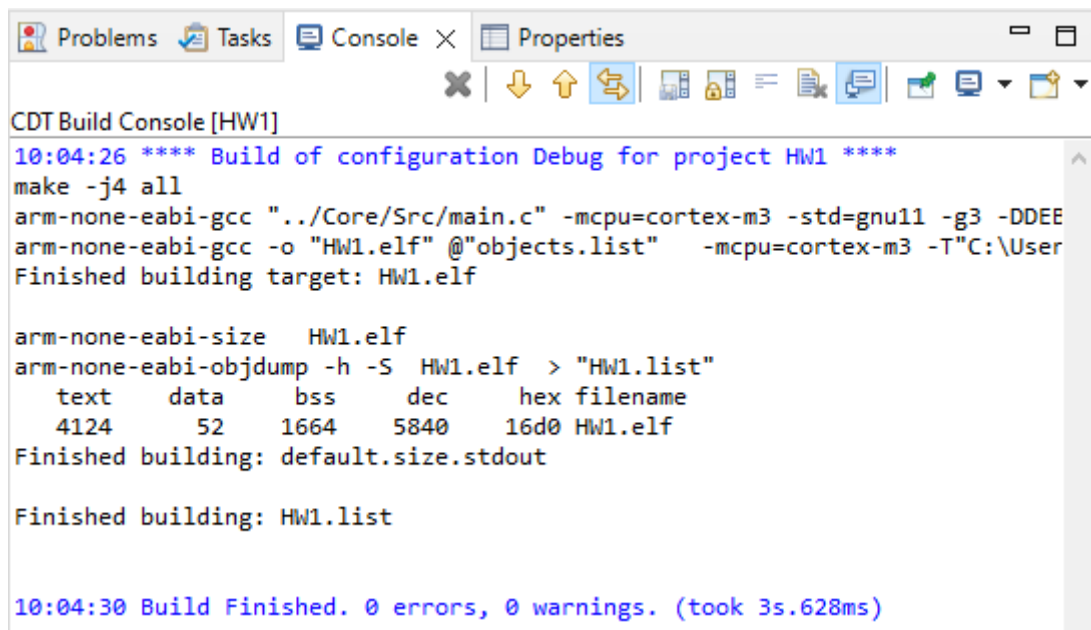


Figure 4: Build the Project

7. Record the Program Size in Hex: _____ (In above example, it is 16d0) on Question 1 of the Worksheet.
8. Now Debug your program by clicking Run → Debug.

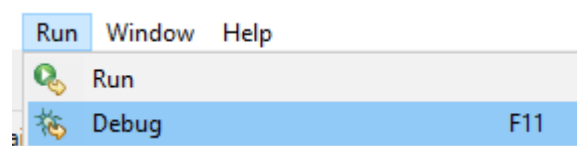


Figure 5: Debug the program

9. You should see now the cursor start at main(void).

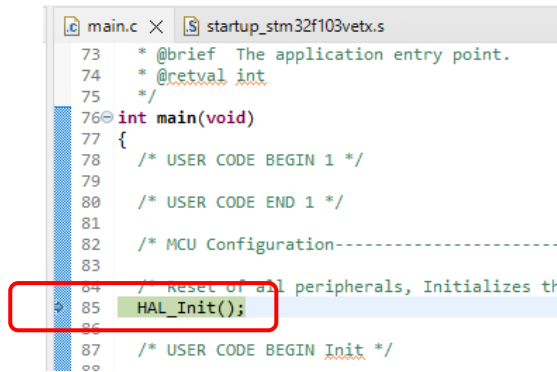


Figure 6: Debug Cursor

10. Below tool bar menu, there are 2 important keys to step by step run your program.





Key	Function
	Step Into (F5)
	Step Over (F6)

Figure 7: Debugging Keys

11. There are many Views to check the status of the program, which are under Windows → Show View.

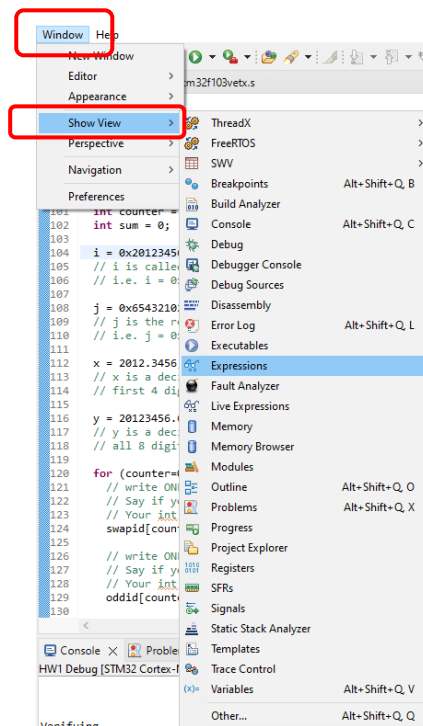


Figure 8: Show View

12. Expressions is one of the View that allow us to check the content of the variables. Choose Show View → Expressions, you will see the Expression View.

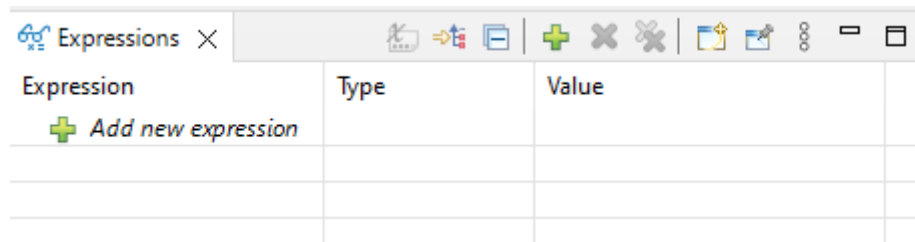


Figure 9: Expressions View

13. Add the expressions according to the Question 2 of the Worksheet.
14. Step Over your program up to the line after the for-loop as below. You can see the cursor is in that instruction. **DO NOT SET BREAK POINT there. OTHERWISE, IT WILL QUIT THE DEBUG**

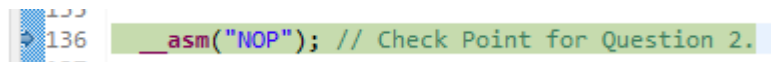


Figure 10: Check Point for Question 2

15. Fill the Question 2 of the Worksheet with your results.
16. You can choose to View the memory by Show View → Memory. At the bottom of screen, you will see a tab Memory.

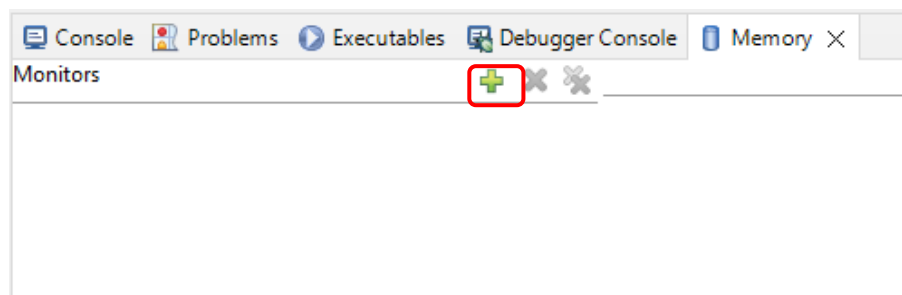


Figure 11: Memory View

17. Press the Green + Button as in Figure 11 to add a memory location, a window pop up. Type 0x20000000 and click OK.

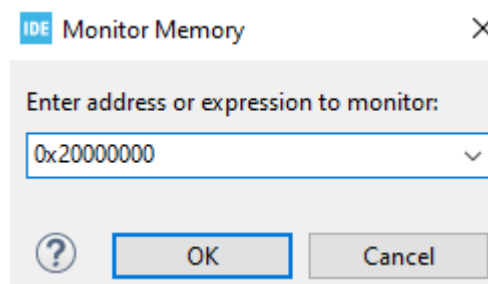


Figure 12: Memory address to Monitor

18. You will see the memory content below. Please note that 4 bytes are grouped together. Color shows the ADDRESS of that BYTE.

Address	0 - 3	4 - 7	8 - B	C - F
20000000	02000000	00000000	01000000	02000000
20000010	03000000	04000000	05000000	06000000
20000020	00A24A04	0F000000	01000000	02BD70B5
20000030	00000000	00000000	00000000	00000000
20000040	00000000	00000000	00000000	56341220
20000050	02214365	0F8BFB44	5119780A	F4307341
20000060	00000000	00000000	00000000	06000000
20000070	05000000	04000000	03000000	02000000
20000080	01000000	00000000	02000000	00000000

Figure 13: Default Memory Renderings in Hex

19. If you refer to the example program, $i = 0x20123456$, so in this view, it showed how the 4 bytes are exactly put in memory in each byte address. With your knowledge from previous courses, what endianness is being used?
20. A Traditional View is easier to see. Click + New Renderings..., then choose Traditional → Add Rendering(s), You will see the following

Address	0x20000000	0x20000001	0x20000002	0x20000003	0x20000004	0x20000005	0x20000006	0x20000007	0x20000008	0x20000009	0x2000000A	0x2000000B	0x2000000C	0x2000000D	0x2000000E	0x2000000F
0x20000000	00000002	00000000	00000001	00000002	00000003	00000004	00000005	00000006	00000007	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E
0x20000010	044AA200	0000000F	00000001	B570BD02	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x20000020	00000000	00000000	00000000	20123456	65432102	44FB8B0F	0A781951	417330F4	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x20000030	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x20000040	00000001	00000000	00000002	00000000	00000000	00000000	00000001	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x20000050	00000000	00000001	00000000	00000000	00000000	4D36466B	1C7619AF	5803683F	19AF7B01	00000000	00000000	00000000	00000000	00000000	00000000	00000000

Figure 14: Traditional Memory Rendering View. A 32-bit number with MSB 0x20, and LSB 0x56

21. Try to understand the difference between the 2 Renderings. (Note that if you mouse over the number in the Traditional Memory View, the byte address is in consistent with the Hex View. So, you might need to refer to the data sheet about the endianness.)
22. With the Traditional Memory View, answer the Question 3 of the answer sheet.

23. You can further check how the C code being compiled to the Assembly code by viewing the Disassembly. Click Windows → Show View → Disassembly.

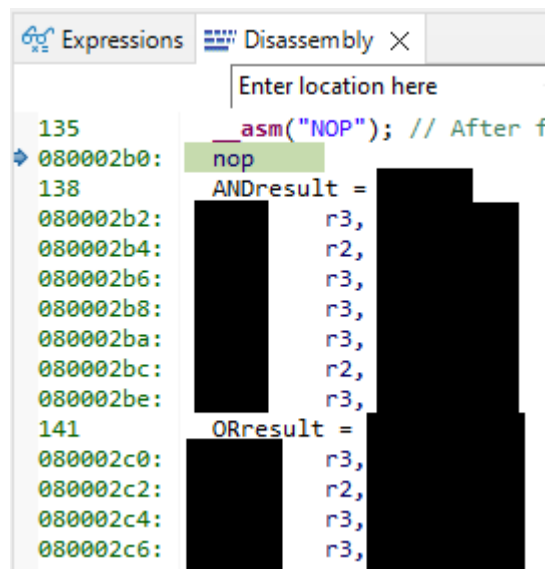


Figure 15: Disassembly View

24. You can click on the Disassembly View and step each assembly instruction.
25. Step Over your program up to the line below, with the Disassembly View and the information from Tradition Memory Rendering View, answer Question 4 and 5 of the Worksheet.

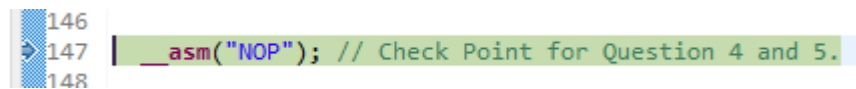


Figure 15: Check Point for Question 4 and 5

26. You can see all the contents of the Registers by clicking Windows → Show View → Registers

Name	Value	Description
General Registers		
r0	0x0	
r1	0x2000008c	
r2	0x20000068	
r3	0x45511554	
r4	0x200000b0	
r5	0x0	

Figure 16: Register View

27. There is a Program Status Register called xpsr for storing the flags. On Register View, if you click on xpsr, you will see the content in Hex, Decimal .. etc.

1010 0101	xpsr	0x21000000
1010 0101	primask	0x0
1010 0101	basepri	0x0
1010 0101	faultmask	0x0
1010 0101	control	0x0
1010 0101	misp	0x2000fff0
1010 0101	psp	0x0

<

Name : xpsr
 Hex:0x21000000
 Decimal:553648128
 Octal:04100000000
 Binary:100001000000000000000000000000
 Default:553648128

Figure 19: xpsr Register

28. Refer to the Tutorial 1A, about Program Status Register, try to associate the 32-bit Hex value for the flags N, Z, C and V. (**Note: As the leading 0's of binary format are omitted, it is easy to make error if looking at binary.**)
29. Step Over your program up to Check Point for Question 6, answer Question 6 based on the Hex value of xpsr.

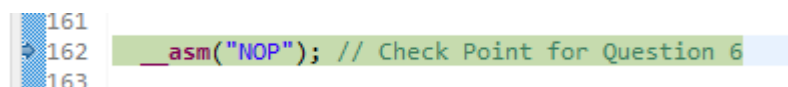


Figure 20: Check Point for Question 6

30. Step Over your program up to Check Point for Question 7, answer Question 7 based on the Hex value of xpsr.
31. Step Over your program up to Check Point for Question 8, answer Question 8 based on the Hex value of xpsr.
32. Take a screen cap of your Traditional Memory View at Check Point for Question 8 as your answer to Question 9.

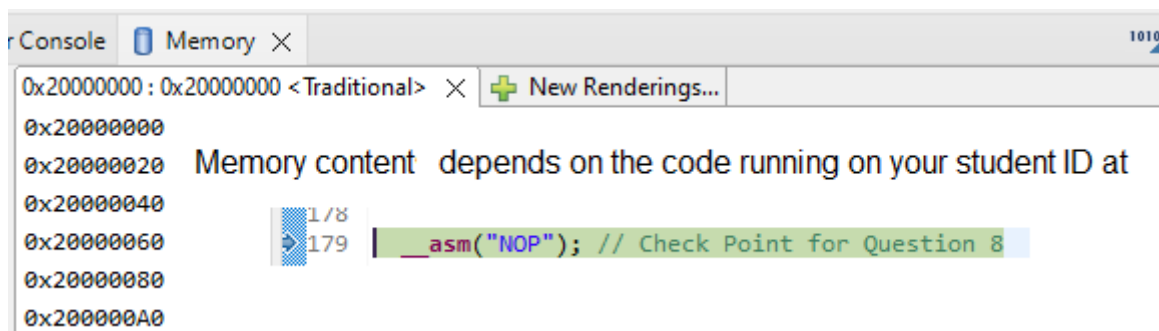


Figure 21: Traditional Memory View at Check Point for Question 8

33. Terminate your Debug by clicking the Red Square

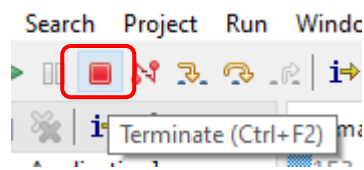


Figure 22: Terminate the Debug

34. Refer to Page 2, Figure 2, change your Optimization to Optimize most (-O3). Then choose Project → Build All to build your Project.

35. In the Console Window, record the Program Size in Hex: _____, and answer Question 9 of the Worksheet.