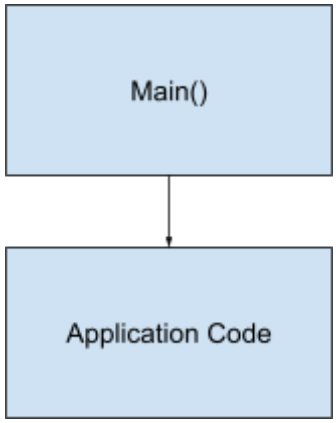
Lab 1 Instructions
Due Date: Sept 6, 2024, at 11:59 PM
Estimated Complexity: Low
Estimated Time to Complete: 2 weeks

(Hypothetical) Engineering Request: Provide a standalone program that will be executed on a core and flash an LED to indicate that the device is active. The LED must flash at a fixed rate for the entire program duration. You will be responsible for writing this program, which will be tested and verified on the STM32F439i discovery board.

Prerequisites:

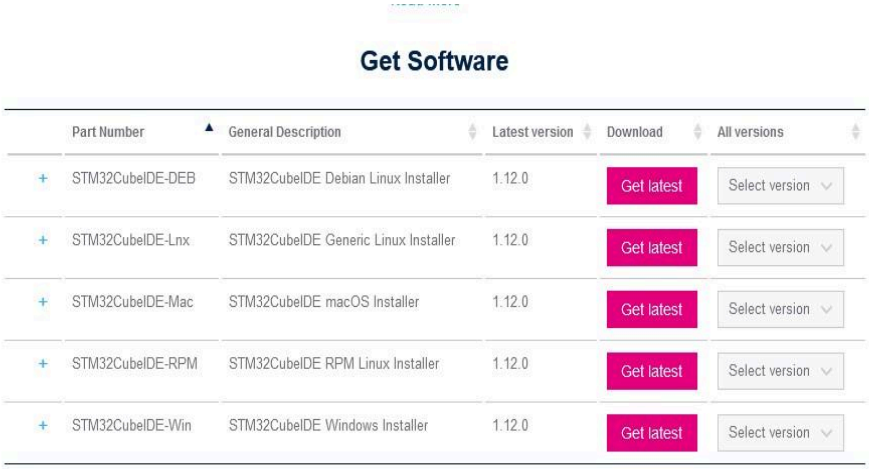
- Know how to operate a computer
- Knowledge of how to follow directions
- A course-approved laptop
- Knowledge of how to run an application installer

Coding Hierarchy for this lab:



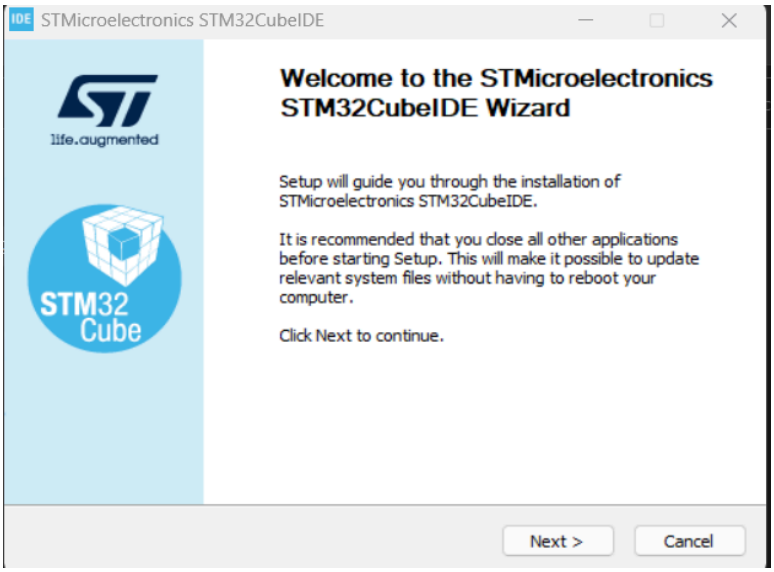

Lab Instruction:

1. Download and Create a Slack account
    a. Use your school email
    b. You should've received a workspace invite via Canvas
    c. Slack will be our primary method of communication, and you can send messages, files, and code snippets directly to the instructional staff through Slack.
2. Download the STM32CubeIDE
    a. Go to https://www.st.com/en/development-tools/stm32cubeide.html
    b. Go to the 'Get Software' section

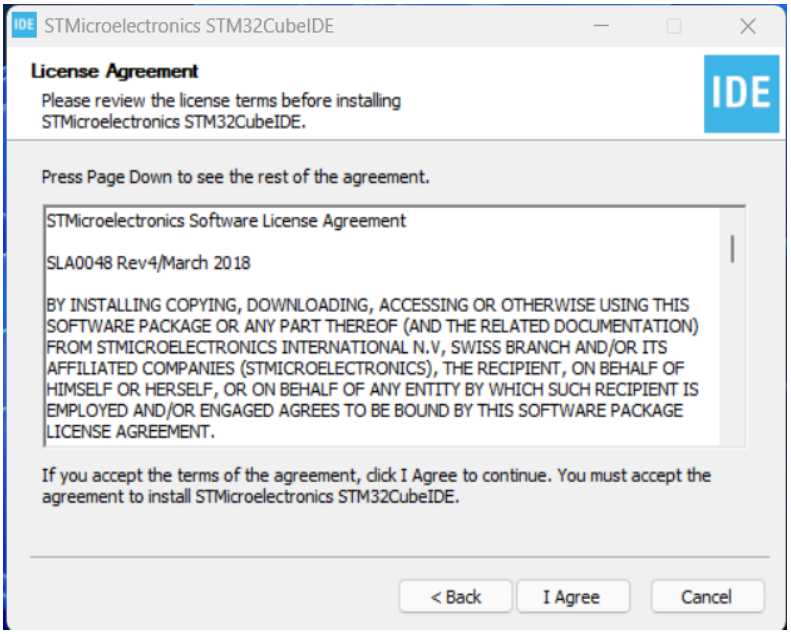



i. Select “Get Lastest”
ii. It should then ask for information to request a download; enter your information (use your school email address)
iii. Within a few minutes, you should get an email from STM. Open it and click the download link
iv. That link should bring you back to the site, where you will need to go back to the ‘Get Software’ section and select ‘Download Latest’
v. The installer file should download

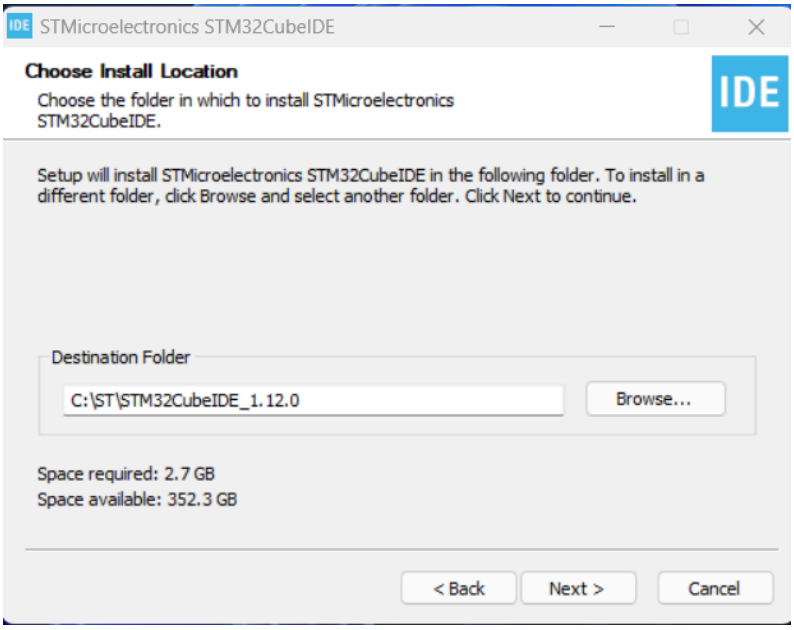
c. Download using the Installer
   i. For Windows:
      1. Double-click the installer. You may be asked if you want to allow this program to make changes to your PC. Select “yes.”
      2. The following window should then come up



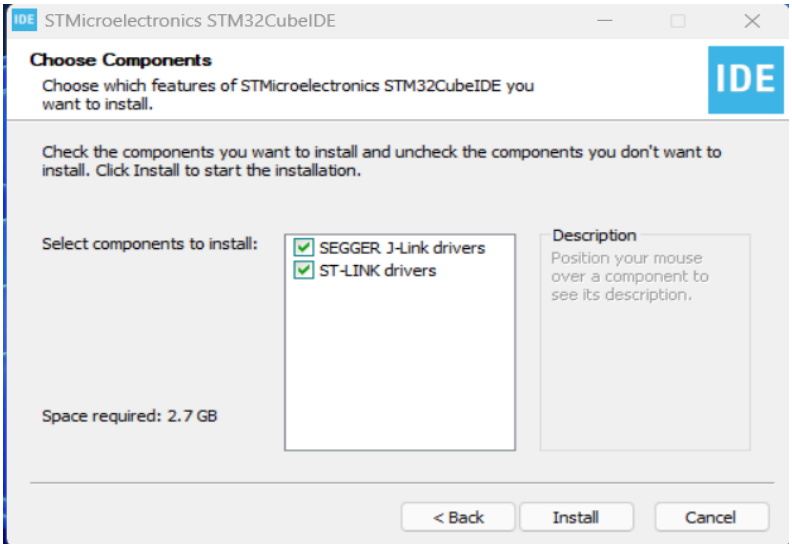

      3. Click Next
      4. Read the license agreement and Click Accept



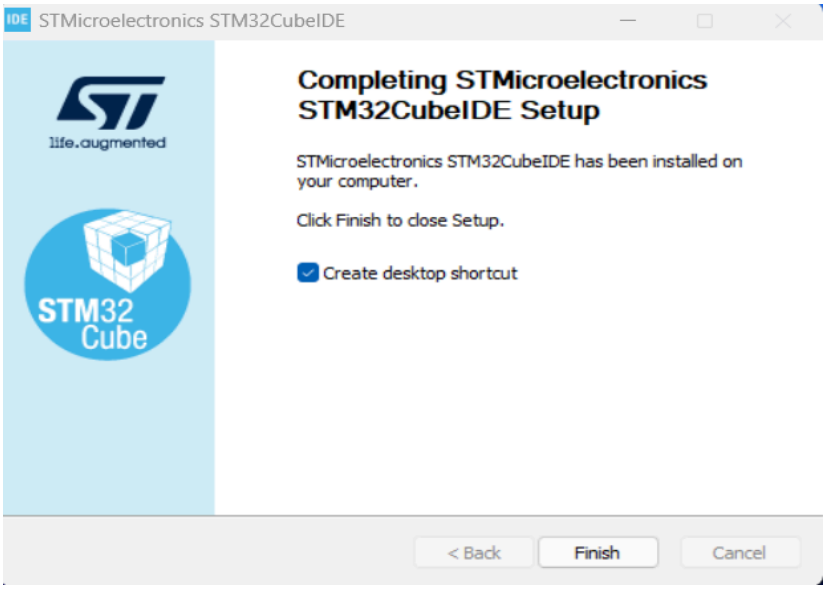

5. Choose the install location; the default is usually sufficient




6. Install the Segger J-Link Drivers and ST-Link Drivers
    a. *These will allow our PC to communicate with relevant components on the board.*
    b. *This option **may not** be shown on Macs, but you will still have them installed I believe*

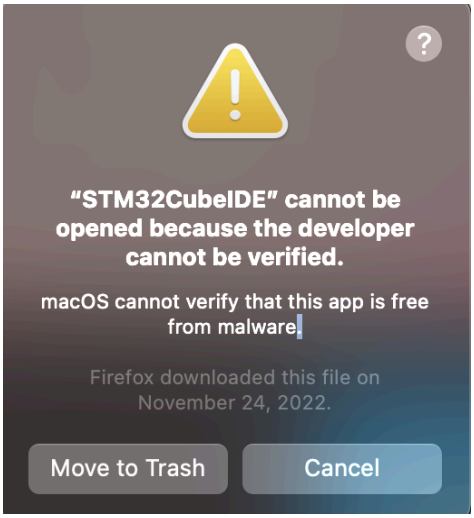



7. Create Desktop shortcut

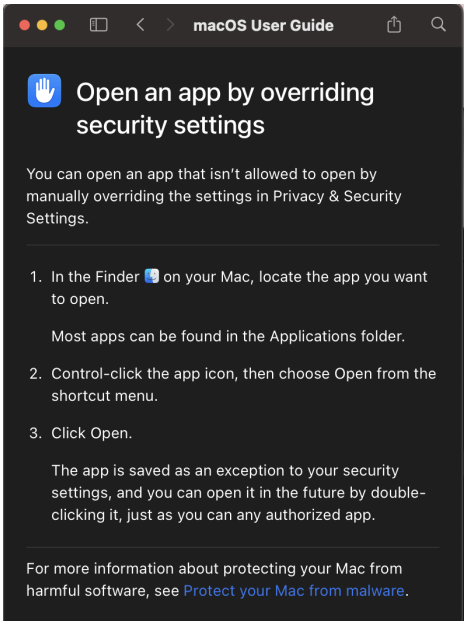



ii. For Mac users:

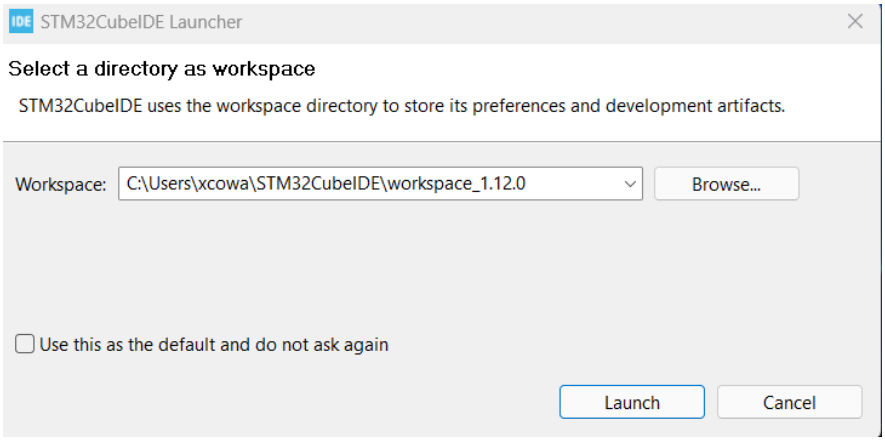
1. The process is pretty much the same for Windows, but the initial download may be a bit different due to the possibility that Apple cannot verify the “author” of the installer.
2. The “error message”:



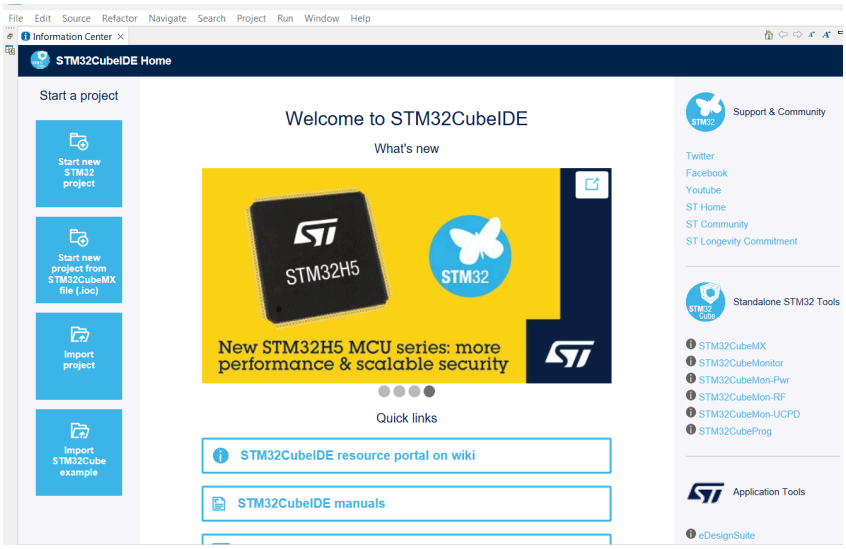

3. The Solution:



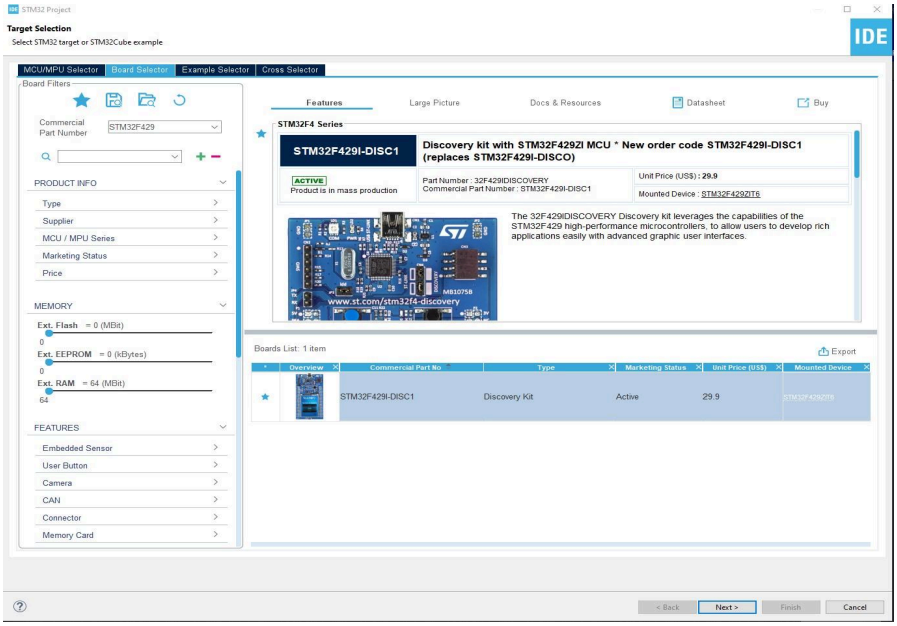

d. Open the STM32CubeIDE
e. It will ask for a "workspace"; choose whichever workspace you would like your projects to be in



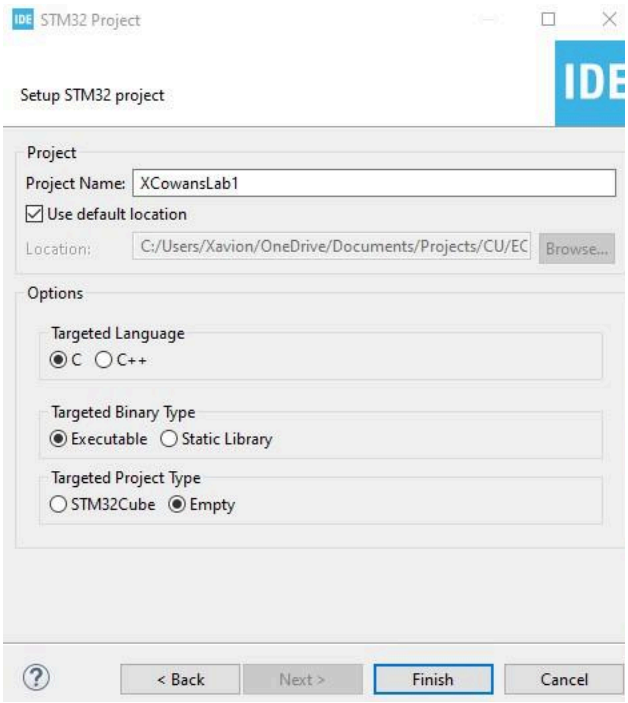

3. Create a new project
    a. You should be at a screen that looks like this:




    b. Select “Start new STM32 project” or go to File>New>STM32 Project
    c. In the Target Selection window:
        i. Go to **board selector**
        ii. type “STM32F429i” in the commercial part ID search box
        iii. Select the STM32F429I-Disc board and select “Next”

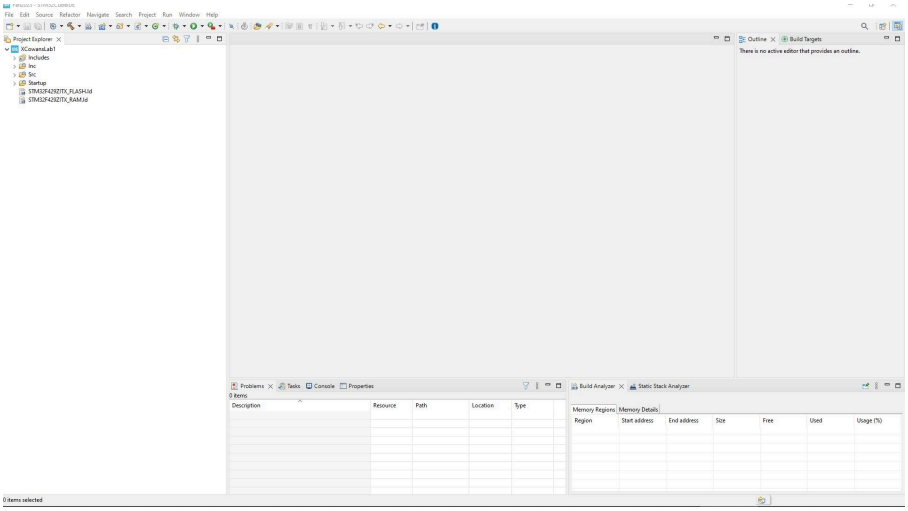



d. In the setup STM32 project:
    i. Name your project ***FirstInitialLastNameLabLabNumber***
        1. For example, XCowansLab1
    ii. Targeted Language should be C
    iii. Targeted Binary Type should be Executable
        1. *This means that the microcontroller can execute this code. Alternatively, code can be 'read-only' and placed into a different section of memory. You will learn about this in later courses.*
    iv. The target Project type should be "Empty"
        1. Failure to select empty will lead you to a path of confusion and despair.
    v. It should look similar to this:




e. Select “Finish”

f. You should now see a screen very similar to this:



g. Open Src/main.c

   i. *This is going to be the entry point of our programs. The CPU will begin execution by beginning execution of int main().*

4. Suppress warning regarding FPU (*Floating Point Unit*) usage in main.c:

   a. Do NOT remove the lines as a method of “suppressing the warnings”

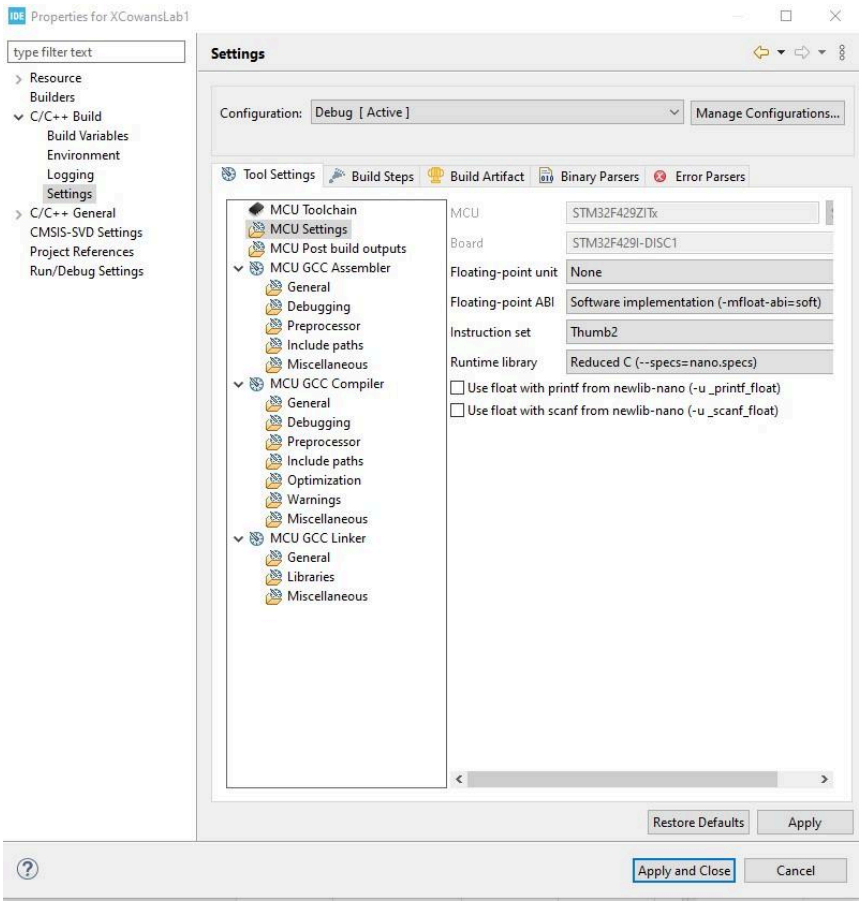
      i. While that technically “solves” the problem, it creates potential issues later down the road if/when we decide to use the FPU or become reliant on it

   b. Go to Project>Properties>C/C++ Build > Settings > MCU Settings

      i. Change the Floating-point unit to “None”

ii. Change Floating-point ABI to “Software implementation”

1. *ABI is a specification on which rules to follow.*
2. *Changing this to “Software implementation” says floating point ABI will be supported by software, and the compiler does need to generate FPU (Floating point unit instructions)*
3. *Here is a link that explains that and more: https://embeddedartistry.com/blog/2017/10/11/demystifying-arm-floating-point-compiler-options/*




iii. Click “Apply and Close”

*iv. You should now see the grayed-out code in main.c, this means that that section of code will not be compiled. Review compile switches in C if needed*

5. Add the Application Code
   a. Create the ApplicationCode.* fileset (A header and source file)
      i. Note, this fileset needs to be named exactly like this! Failure to comply with the naming scheme will result in you forfeiting all points related to code standards
   b. Add code to the header file
      i. Create the following function **prototypes**. If an input or output argument is not specified, you can assume there is none for that specific argument

ii. Note, these functions do not have to be named exactly how I name them. You are free to change the name if you'd like
    1. AppDelay
        a. Brief: This will act as a function to do a software delay
        b. Input Arguments: A uint32_t to determine the amount of tie to delay
    2. EnablePeripheralsAndClocks
        a. Brief: Configures and enables peripherals and clocks
    3. TurnOnRedLED
        a. Brief: Turn on the red LED
    4. TurnOffRedLED
        a. Brief: Turn off the red LED
    5. ToggleRedLED
        a. Brief: Toggle the red LED, that is, if the LED is on, turn it off. If the LED is on, turn it off

c. Add code to the source file
    i. Create the following macros and have them evaluated to their appropriate value, you will have to use the reference manual for this
        1. The RCC enable direct address for GPIOG
            a. Hint: this will be at address 0x40023830
                i. 0x40023800 is the RCC base address, 0x30 is the AHB1ENR offset
            b. So it will look something like this:

```
#define GPIOG_RCC_ENABLE_ADDRESS  0x40023830
```

        2. The GPIOG AHB1ENR bit offset
            a. Hint: This will evaluate to 6
            b. So:

```
#define GPIOG_AHB1ENR_BIT_OFFSET    6
```

        3. The GPIOG MODER direct address
        4. The GPIOG ODR direct address
        5. The GPIOG MODER offset
        6. The LED_ODR_BIT offset
        7. A macro that will contain the length of your first name. This will be a number.
    ii. In the AppDelay function:
        1. Create and initialize an array with your name
            a. Use the macro you created above for the size of the array
        2. Create another array with the same size, and name it something like "destinationArray"
        3. Create two for loops. One of them will be contained in the other
            a. The outside for loop will count to the time given by the input parameter

b. The inside loop will iterate through your name array and the destination array, putting the letter from your name array into the destination array
    i. This is awful practice, but this is how we will use delay functions for now

iii. In the EnablePeripheralsAndClocks
1. Create two pointers of size uint32_t, and have them point to the GPIOG RCC enable address and the GPIOG mode address respectively
    a. For example, one of the lines will look like this:

```
uint32_t * clkPtr = (uint32_t *) GPIOG_RCC_ENABLE_ADDRESS;
```

    *b. We need to cast the address as a pointer so we can access it properly*
2. Utilizing the pointer for the mode address, clear the bits responsible for GPIOG moder configuration
    a. This will look something like this

```
*modePtr &= ~(0x3 << GPIOG_MODER_OFFSET);
```

3. Utilizing that same pointer, set the appropriate moder bits for GPIOG
4. Utilizing the clock pointer, set the bit responsible for enabling the bus that GPIOG is on

iv. In the TurnOnRedLED function
1. Create a pointer of size uint32_t and have it point to the GPIOG ODR address
    a. Using that pointer and a macro you created above, set the appropriate bit in the ODR register

v. In the TrunOffRedLED function
1. Similar process to turning it one, but instead of setting a bit in the ODR register, you will clear it

vi. In the ToggleRedLED function
1. Create a pointer to the GPIOG ODR address
2. Create an if statement that will determine if the ODR bit responsible for the LED is set. If it is, turn off the LED; otherwise, turn it on.

6. Integrate code into main.c
    a. Properly include the applicationCode fileset in main.c
    b. Create a macro for the default delay, you can set this to however long you want, I used 25000
    c. In the main function
        i. Call your function that is responsible for enabling the clock and peripherals
        ii. In the infinite for-loop, put a call to your delay function, the macro created above should be the input argument
        iii. Toggle the Red LED

7. Build your code
8. Download/Flash your code to the board and debug your program!
    a. *You will likely need to set breakpoints, view registers, and step through code. Please verify you know how to do that.*
    b. Hint: The error should be *around* the configuring the GPIO peripheral...
9. Verify your code doesn't have warnings by doing a clean build
    a. To address the unused warning in ApplicationCode.c use the [[maybe_unused]] attribute in c on the line where the variable is declared
        i. The IDE may still say there is a warning on this line, but the compiler won't. This is due to the linting utilizing a different version of C than the compiler
10. Export your project and turn it into Canvas by the Due Date

Acceptance Criteria:

- The code compiles
- The LED flashes at a fixed rate
- You have joined the Slack channel with a work/school-appropriate picture of yourself
- Use of the generated HAL code is prohibited for this assignment, if you use the HAL, you will forfeit ALL points for this assignment!

Grading Rubric:

This lab will be worth 150 points. The breakdown of grading will be given below.

- Code Compilation (15 points)
    - Full credit - Code Compiles with 0 errors and 0 code warnings
    - Partial Credit - Code contains warnings (-10 points for each warning)
    - No credit - Code does not compile (Student has at least one error)
- Code Standards and Hierarchy (20 points)
    - Proper layering of files, for each violation, 10 points will get subtracted from the 20 points possible
- Code functionality (20 points)
    - Does the student have a flashing LED (30 points)?
- Class Involvement (5 points)
    - Did the student join the Slack with an appropriate picture of them?
- Project Exporting (10 points)
    - Was the project exported right with the appropriate naming?
- Lab Attendance (5 points)
    - Did the student attend lab sessions appropriately and consistently
- Acknowledging the Lab Contract in Canvas (25 Points)
    - Did the student read and acknowledge the lab contract?
- Interview Grading (50 points)
    - Does the student understand a basic concept utilized in their code? (15 Points)
    - Does the student understand a semi-complex concept utilized in their code? (25 points)

- Does the student understand or partially understand a semi-advanced concept utilized in their code? (10 points)