PCI-2 [Week 4]: Board & Library Familiarization

Objectives

- To be familiar with the B-L4S5I-IOT01A board, the location and function of various sensors, and the STM32L4S5 interfaces used by each sensor.
- To set up the printing configurations in STM32CubeIDE.
- To run the code skeleton and read sensors periodically.
- To get into the habit of **experimenting and exploring things on your own**.

Resources:

Lab2.zip (https://canvas.nus.edu.sg/courses/85090/files/8430770?wrap=1) (https://canvas.nus.edu.sg/courses/85090/files/8430770/download?download_frd=1)

Reference Materials

STM32L4+ Technical reference manual.pdf (https://canvas.nus.edu.sg/courses/85090/files/8295817?wrap=1) (https://canvas.nus.edu.sg/courses/85090/files/8295817/download?download_frd=1)

STM32L4Sxxx Data-sheet.pdf (https://canvas.nus.edu.sg/courses/85090/files/8295818?wrap=1) (https://canvas.nus.edu.sg/courses/85090/files/8295818/download?download_frd=1)

STM32L4+ series Discovery kit.pdf (https://canvas.nus.edu.sg/courses/85090/files/8295819?wrap=1) (https://canvas.nus.edu.sg/courses/85090/files/8295819/download?download_frd=1)

L4S5VI-e03_schematic.pdf (https://canvas.nus.edu.sg/courses/85090/files/8295813?wrap=1) (https://canvas.nus.edu.sg/courses/85090/files/8295813/download?download_frd=1)

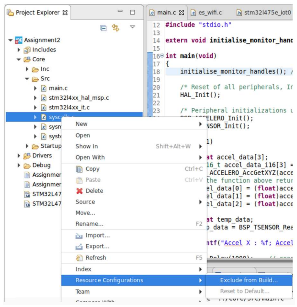
1. Enable printf() in STM32CubeIDE

You will need print() to print some outputs in the console. There are two ways to enable it and their appropriate settings are:

Method A: using semihosting to print in console

source: https://shawnhymel.com/1840/how-to-use-semihosting-with-stm32/ (https://shawnhymel.com/1840/how-to-use-semihosting-with-stm32/)

1. Disable the original *syscalls.c* file. If this is not done, you will get error messages such as "Core/Src/syscalls.c:110: first defined here" when you build. **Right-click** on **syscalls.c** > **Resource Configurations** > **Exclude from Build**.

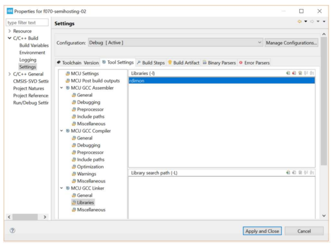


```
#include "stdio.h"

extern void initialise_monitor_hanc

int main(void)
{
    initialise_monitor_handles(); /

    /* Reset of all peripherals, Ir
    HAL_Init();

    /* Peripheral initializations 
```

In the window that pops up, select **Debug** and click **OK**.

2. Go to the **properties**of project and finish the following:

- Go to the **properties** of project, **C/C++ Build** and select the **Tool Settings** tab. Then, select **MCU GCC Linker** > **Libraries**. In the libraries pane, click the **Add… button** and enter **rdimon**. This enables librdimon for us to make system calls with semihosting.

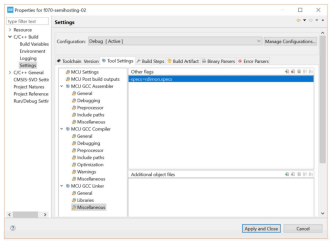


Properties for f070-semihosting-02

Configuration: Debug [Active]

Libraries (-l): rdimon

- Click on **MCU GCC Linker** > **Miscellaneous** while still in the **Tool Settings** tab. Click the **Add… button** and enter **-specs=rdimon.specs** into the dialog box.



Properties for f070-semihosting-02

Configuration: Debug [Active]

Other flags: -specs=rdimon.specs

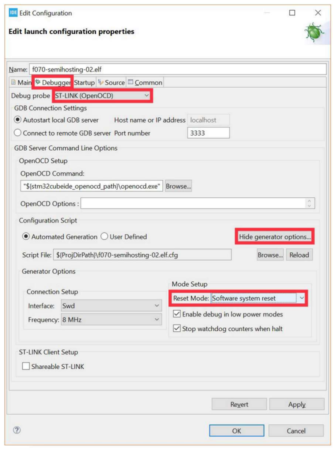
- Click **Apply and Close**.

3. Add the following lines in main.c

- **extern void initialise_monitor_handles(void);** before **int main(void)**
- **initialise_monitor_handles();** inside **int main(void)** but before **while(1)** loop
- add **printf(...)** functions in the user code wherever necessary

4. Set debug configuration

- When debugging the project using **Run** > **Debug As** > **STM32 Cortex-M C/C++ Application**, change the following fields as shown in the figures:



Edit Configuration

Edit launch configuration properties

Name: f070-semihosting-02.elf

Debug probe: ST-LINK (OpenOCD)

GDB Connection Settings
- (●) Autostart local GDB server — Host name or IP address: localhost
- ( ) Connect to remote GDB server — Port number: 3333

GDB Server Command Line Options

OpenOCD Setup

OpenOCD Command: "${stm32cubeide_openocd_path}\openocd.exe"

OpenOCD Options :

Configuration Script: (●) Automated Generation ( ) User Defined

Script File: ${ProjDirPath}\f070-semihosting-02.elf.cfg

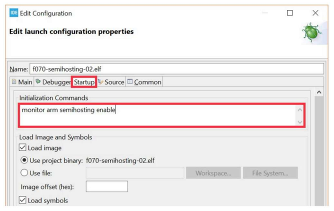
Generator Options

Connection Setup — Interface: Swd; Frequency: 8 MHz

Mode Setup — Reset Mode: Software system reset
- [✓] Enable debug in low power modes
- [✓] Stop watchdog counters when halt

ST-LINK Client Setup
- [ ] Shareable ST-LINK

Course chat


Edit Configuration

Edit launch configuration properties

Name: f070-semihosting-02.elf

Main | Debugger | Startup | Source | Common

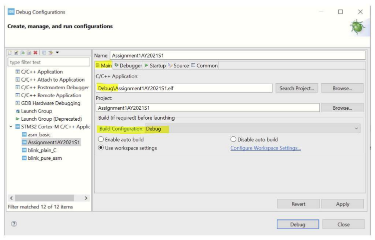
Initialization Commands

```
monitor arm semihosting enable
```

Load Image and Symbols

- [✓] Load image
- (●) Use project binary: f070-semihosting-02.elf
- ( ) Use file:
- Image offset (hex):
- [✓] Load symbols

***** Additional setting in "Main" tab for Macbook users: ***


Debug Configurations

Create, manage, and run configurations

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
- Launch Group
- Launch Group (Deprecated)
- STM32 Cortex-M C/C++ Applic
  - asm_basic
  - Assignment1AY2021S1
  - blink_plain_C
  - blink_pure_asm

Filter matched 12 of 12 items

Name: Assignment1AY2021S1

Main | Debugger | Startup | Source | Common

C/C++ Application: Debug\Assignment1AY2021S1.elf

Project: Assignment1AY2021S1

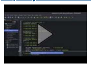
Build (if required) before launching

Build Configuration: Debug

- ( ) Enable auto build
- ( ) Disable auto build
- (●) Use workspace settings
- Configure Workspace Settings...

Method B: using SWV (Serial Wire Viewer) to print in SWV ITM data console (not shown in lab)

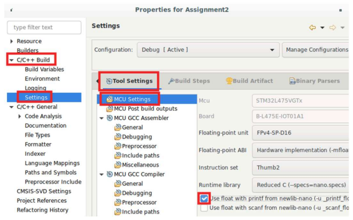
source:
https://youtu.be/sPzQ5CniWtw *(https://youtu.be/sPzQ5CniWtw)*



(https://youtu.be/sPzQ5CniWtw)

2. Printing floating-point numbers

Check the '**Use float with printf...**' option as below in the **Project** > **Properties** > **C/C++ Build** > **Settings** > **Tool Settings** > **MCU Settings**.


Properties for Assignment2

Settings

Configuration: Debug [Active]

Tool Settings | Build Steps | Build Artifact | Binary Parsers

C/C++ Build > Settings > Tool Settings > MCU Settings

Mcu	STM32L475VGTx
Board	B-L475E-IOT01A1
Floating-point unit	FPv4-SP-D16
Floating-point ABI	Hardware implementation (-mfloa
Instruction set	Thumb2
Runtime library	Reduced C (--specs=nano.specs)

- [✓] Use float with printf from newlib-nano (-u _printf_flo
- [ ] Use float with scanf from newlib-nano (-u _scanf_flo

3. Explore the B-L4S5I-IOT01A Board from Data Sheets

Understand the location, purpose, and connections of all the peripherals on the Baseboard. Refer to the **STM32L4+ series Discovery kit.pdf (https://canvas.nus.edu.sg/courses/85090/files/8295819?wrap=1)** (https://canvas.nus.edu.sg/courses/85090/files/8295819/download?download_frd=1) and **STM32L4Sxxx Data-sheet.pdf (https://canvas.nus.edu.sg/courses/85090/files/8295818?wrap=1)** (https://canvas.nus.edu.sg/courses/85090/files/8295818/download?download_frd=1)

Be able to answer questions such as

- Where is the temperature sensor mounted on the board?
- Which interface on STM32L4S5 is used by the temperature sensor for data transfer? I2C1 / I2C2 / SPI1 / UART2?
- Which STM32L4S5 pin(s) are used by the temperature sensor for data transfer and what is the alternate function number (you will need to refer to **STM32L4Sxxx Data-sheet.pdf (https://canvas.nus.edu.sg/courses/85090/files/8295818?wrap=1)** (https://canvas.nus.edu.sg/courses/85090/files/8295818/download?download_frd=1) too for the latter)?
- How about the STM32L4S5 pins used by the accelerometer? Are some of the STM32L4S5 pins used by the two sensors the same? If yes, how can it be possible, won't there be a clash?
- You can find more information (than what is provided in **STM32L4+ series Discovery kit.pdf (https://canvas.nus.edu.sg/courses/85090/files/8295819?wrap=1)** (https://canvas.nus.edu.sg/courses/85090/files/8295819/download?download_frd=1) about the temperature sensor itself by Googling for the device part number (e.g. HTS221 for the temperature sensor) + datasheet. You need the datasheet of a device when
  1. What kind of packaging does HTS221 come in?
  2. How many pins does it have?
  3. you wish to know more about the features and specifications of the device.
  4. you are making the connections between the device (e.g. HTS221) and the microcontroller (.e.g. STM32L4S5) yourself and/or you are designing a hardware board yourself rather than using a ready-made board like B-L4S5I-IOT01A.
  5. you do not have ready to use driver functions/libraries to interface it with the specific microcontroller you are using.
  6. you want to have finer control over the device than what is provided by library functions.

4. Run, Test and Understand the Code Skeleton

You will be building on the **Lab2.zip (https://canvas.nus.edu.sg/courses/85090/files/8430770?wrap=1)** (https://canvas.nus.edu.sg/courses/85090/files/8430770/download?download_frd=1) project for this lab.

- Identify the x-, y-, and z-axes of the board and observe how the acceleration values change as you change the orientation of the board.
- How can you bring about a change in the temperature sensor reading?
- Make use of debugging features such as single-stepping, running until breakpoints ('Resume' button), stepping over, stepping into, and stepping out of functions ('Step Return' button), using the 'Terminate and Relaunch' button, inspecting the values of variables etc. Learn how to switch between perspectives (Window > Perspective > Open Perspective > Debug or C/C++).

5. Demo Program Modification

Modify the main.c of the project to do the following. The experience gained during the lab is crucial. Most of the code can be reused for your assignment, with appropriate modifications.

- Read the accelerometer and print the value once every 1 second.
- Read the temperature sensor and print the value once every 1.5 seconds.

Hint: You can use a delay, which is the highest common factor of the reading/writing frequencies as the parameter for HAL_delay() inside the main while loop. This, along with a counter variable, will allow you to achieve the desired reading/writing frequency.

6. Tips on using the IDE

- Learn the various shortcuts of the IDE, which will allow you to do certain repetitive tasks much faster than what is possible with mouseclicks. E.g. F5 for Step Over, F6 for Step Into, F7 for Step Return, F8 for Resume, Ctrl+F2 to terminate.
- Ctrl+Space after typing the first few characters of anything (variable names, function names, etc) gives you autocomplete options. Use arrow keys and Enter to select the correct completion if there are multiple options.
- There is even an instruction stepping mode (button next to Step Return) where you can step through the generated assembly instructions instead of the C code. Not too useful normally, but allows you to see the correspondence between C and assembly.
- Good code is code which is readable, comprehensible, and extensible. If you can't read/comprehend, you can't debug the code, you can't improve the code, you can't collaborate in a team. Hence the importance of following good coding styles/practices.
  - **Write detailed comments**. This will be very useful later for you yourself or other people who read your code.
  - **Use descriptive variable names** for variable etc. rather than short generic names like i, j, k. This will help your code be more readable and comprehensible.
  - **Indent your code properly**. Indent with tabs, not with spaces. This also makes your code easier to read.
- **Use constants/variables** for values that appear at multiple places rather than hard-coding values.
- Note that the example below is just an example, not specific to STM32.

Poor Coding Style

```
void motorWrite()
        {                               // Poor position for opening brace.
    //
    int j = 100;            // Non-descriptive variable name
  pwmWrite(5, j);           /*      Indented with spaces, and not aligned with the previous line.
                                                  Hard-coded values.
                                                  No comment to indicate what the line does.*/
                            }                // Poor position for closing brace.
```

Good Coding Style

```
const int leftMotor = 5; // pin 5 used to control left motor of the robot

void motorWrite(){
   //
   int leftMotorSpeed = 100;
   pwmWrite(leftMotor, leftMotorSpeed); // to write the analog value 100 to pin 5.
}
```