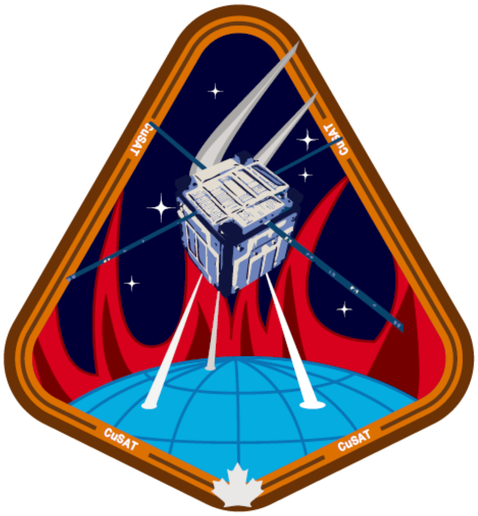
Carleton CuSAT Design Project



Command and Data Handling

Development Environment Setup Guide

CuSAT-CDH-DESG-001
Revision 1

Author:

Tyler Mair

Contents

Foreword

This guide is focused on setup of C&DH teams modified HAL demo code for 2021-2022. We decided to use Eclipse IDE. The prior year 2020-2021 did work to change the build process to use CMAKE and provided no guidance on corresponding IDE setup. Due to this lack of documentation we reverted to using Eclipse and adapting settings from original Hal-demo. We use Eclipse to manage the build settings with 'Gnu Make Builder' with the 'Arm Cross GCC' toolchain. Should a change in IDE or build process be pursued, it must be documented very well so new students can get up and running quickly and not spend months wasting time.

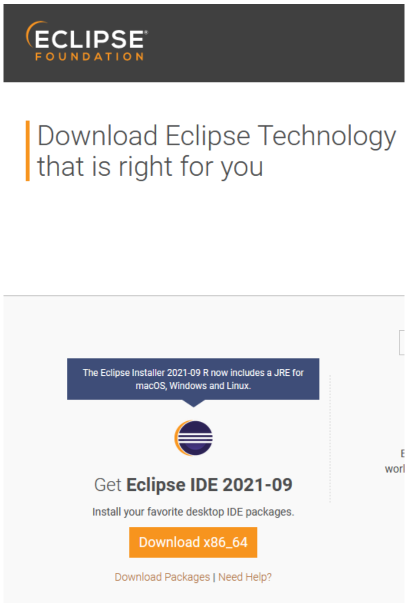
You are expected to know how to use Git, Github, and tools to file manage with Github such as GithubDesktop or Gitbash.

One of the labs outlined in this guide seems to show that the I2C multi-master issue can be worked around and may not be a factor. I suggest attempting to do this lab in the first month. You will learn a lot and be jump started on one of the major issues investigated.

Environment Setup

Download and Install Eclipse – Windows 10 Guide (64 bit)

1. Goto: https://www.eclipse.org/downloads/



Download Eclipse Technology that is right for you

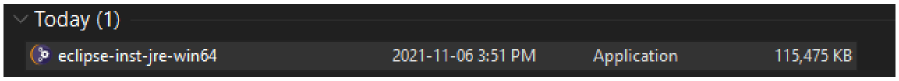
The Eclipse Installer 2021-09 R now includes a JRE for macOS, Windows and Linux.

Get **Eclipse IDE 2021-09**

Install your favorite desktop IDE packages.

Download the installer "Download x86_64", file: "eclipse-inst-jre-win64.exe".

2. The File:



Today (1)

eclipse-inst-jre-win64	2021-11-06 3:51 PM	Application	115,475 KB

3. Install Eclipse IDE for Embedded C/C++ Developers to a location of your choosing and do not modify the VM settings. Below is the particular version we used for reference, you may try more modern versions, it *should* be ok.

Eclipse IDE for Embedded C/C++ Developers (includes Incubating components)

Version: 2021-09 (4.21.0)

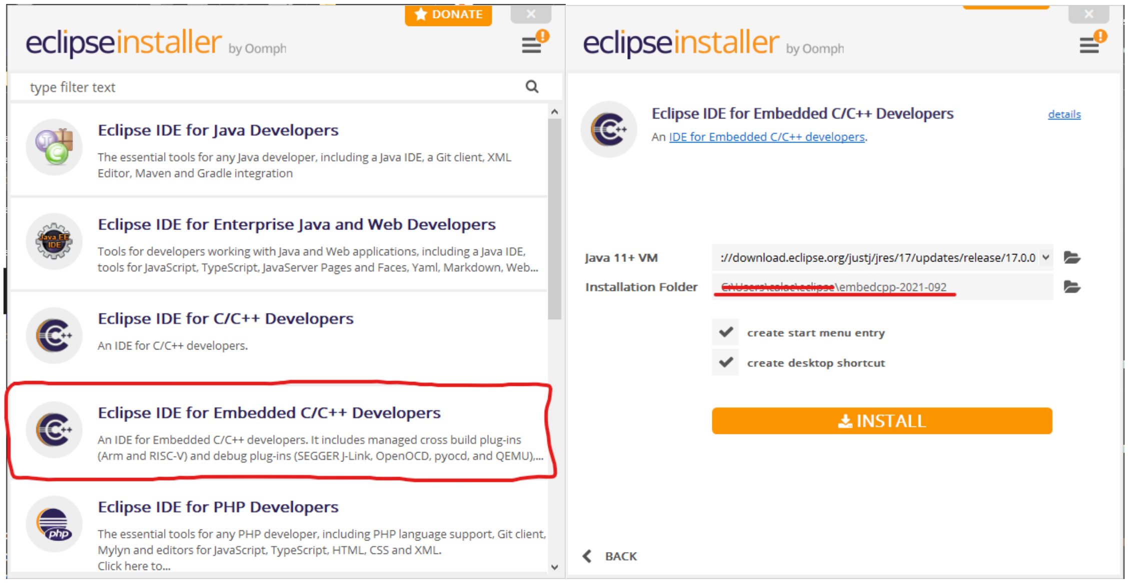
Build id: 20210910-1417

OS: Windows 10, v.10.0, x86_64 / win32

Java vendor: Eclipse Adoptium

Java runtime version: 17+35

Java version: 17



Eclipse IDE for Java Developers

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration

Eclipse IDE for Enterprise Java and Web Developers

Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web...

Eclipse IDE for C/C++ Developers

An IDE for C/C++ developers.

Eclipse IDE for Embedded C/C++ Developers

An IDE for Embedded C/C++ developers. It includes managed cross build plug-ins (Arm and RISC-V) and debug plug-ins (SEGGER J-Link, OpenOCD, pyocd, and QEMU)...

Eclipse IDE for PHP Developers

The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, TypeScript, HTML, CSS and XML.
Click here to...

Eclipse IDE for Embedded C/C++ Developers

An IDE for Embedded C/C++ developers.

Java 11+ VM ://download.eclipse.org/justj/jres/17/updates/release/17.0.0

Installation Folder [redacted]\embedcpp-2021-092

✓ create start menu entry

✓ create desktop shortcut

4. Once eclipse is installed don’t open it yet, we need to install the iOBC SDK elements.

Clone the Repo

1. Clone “[repo_owners_username_here]/MAAE4997B_CDH” to a location of your choosing where you can find it again for project import in Eclipse.
2. I placed the cloned repo into the ISIS folder to keep everything tidy and in one spot.
3. **There is a file under ‘hal-demo” project folder called “.cproject”. This file provides critical setup configuration to Eclipse and fixes several bugs in the original hal-demo IDE configuration that prevented debug testing and compilation. Much effort on my part went into this so you could get up and running quickly. This file should not be ignored in git tracking but also is not expected to change unless certain project ‘C’ properties are changed. This file will be brought into the Eclipse project as part of the “Setup Eclipse and import projects “ step below.**

Install JLink from SEGGER – Adapted from ISIS QuickStart 3.1

1. Find in the repository, the folder “..\MAAE4997B_CDH\Supplementary_Content\Setup_Files\JLinkSetup”
2. You may use the file to install JLink from SEGGER V7.54d for Windows 10 64-bit.
3. You may go to https://www.segger.com/jlink-software.html and download a newer version or different type. Please Document any changes if you get setup with a different version.
4. **NOTE: the version suggested in the ISIS-OBC guide does not exist anymore (Lowest is V5.00), so we had to use new version and fortunately, it worked.**
5. **IMPORTANT: Note the path of installation. You will need to know the path to this: “..\SEGGER\JLink\JLinkGDBServerCL.exe”**

Install FTDI Driver - Adapted from ISIS QuickStart 3.2

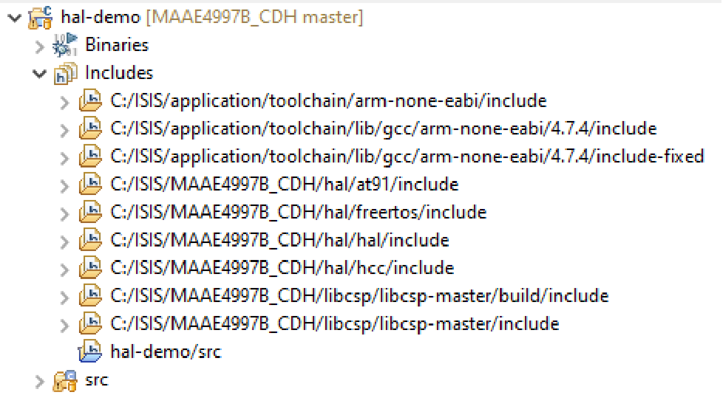
1. Find in the repository the folder “..\MAAE4997B_CDH\Supplementary_Content\Setup_Files\FTDI_Driver”
2. The versions will not change very much so the included file should be ok to use.
3. **File is for Windows (Desktop) – includes support for many versions of windows.**
4. Run the setup file “CDM212364_Setup” to install driver for our debug USB adapter.
5. https://ftdichip.com/drivers/vcp-drivers/ Is the website, we used the ‘setup executable’ link on the right to get the setup wizard. Please select a different one if needed; document any changes.

Install iOBC SDK elements - Adapted from ISIS QuickStart 3.3

1. Acquire the “ISIS-OBC SDK.exe” file (around 512 MB in size) and install the elements included, **do not install Eclipse from here, it is very old, uncheck that as an install option**.
    a. This file is large and can not be stored on the repo and is not on the SVN. ***Consult Bruce - Project Lead, for a copy***.
2. This will install to C:/ISIS/ by default and setup the path environmental variables to point to toolchain elements in this folder. If you change this path, issues may be encountered, I didn’t try a different location. If you attempt it, and it works, or you get it to work, please document this here.

Setup Eclipse and import projects

1. Start up eclipse.
2. Create a workspace in a spot of your choosing, **don't use the default workspace**, pick a name that makes sense. I chose "eclipse-workspace-iOBC-2021" and placed it in the ISIS folder to keep everything tidy and in one spot. You shouldn't have to directly access workspace files so it could be placed wherever is convenient.
3. Close the welcome panel and select import from the left hand project explorer pane.
    a. *You may find import under File menu as well.*
4. From the list select General - Existing Projects into Workspace.
5. Browse for the root directory and locate the repo folder we cloned earlier. Select this folder as root and eclipse will find the projects within.
6. The projects associated with the repo will be listed, select what you want.
    a. At time of writing there is only one project: "hal-demo"
7. All options should be **unchecked** and just finish to import the project.
8. Expand the project and includes folder in the project explorer.



```
hal-demo [MAAE4997B_CDH master]
  Binaries
  Includes
    C:/ISIS/application/toolchain/arm-none-eabi/include
    C:/ISIS/application/toolchain/lib/gcc/arm-none-eabi/4.7.4/include
    C:/ISIS/application/toolchain/lib/gcc/arm-none-eabi/4.7.4/include-fixed
    C:/ISIS/MAAE4997B_CDH/hal/at91/include
    C:/ISIS/MAAE4997B_CDH/hal/freertos/include
    C:/ISIS/MAAE4997B_CDH/hal/hal/include
    C:/ISIS/MAAE4997B_CDH/hal/hcc/include
    C:/ISIS/MAAE4997B_CDH/libcsp/libcsp-master/build/include
    C:/ISIS/MAAE4997B_CDH/libcsp/libcsp-master/include
    hal-demo/src
  src
```

Figure 1 – 2022 CDH modified hal-demo project shown for example

Verify that the /ISIS/application/toochain/ paths are there as well as the hal includes. More may be there for future projects and other libraries.

Expect the toolchain includes to stem from the ISIS folder (from SDK install), and the others are stemming from where the project repo is located. **See troubleshooting "Check Build Variables and Includes paths" section if the includes are not there. Please update image and documentation if this changes.**

NOTE: **libcsp includes are for CSP support (v1.6)**. CSP will allow for compilation of a library, but I was unable to get it to work with the project. Instead, I include the headers and a config file as well as all the source code for CSP. The key reason for this is so we compile hal-demo and csp code together under one version of FreeRTOS instead of two; the later being complex and difficult to reconcile. At some point it may be required to figure out how to use the CSP library with our project instead of the full source code to minimize project size.

9. Under the project folder find and open the file “Jlink.launch”.
10. Change the following line (line 7) to point to the **JLinkGDBServerCL** file:

```
<stringAttribute key="org.eclipse.ui.externaltools.ATTR_LOCATION"
value="C:\Program Files\SEGGER\JLink\JLinkGDBServerCL.exe"/>
```

    NOTE: your path may differ from the one shown in the above example!
11. Under the project folder find and right click on “Jlink.launch” and select “Run As” then select “1 JLink”. A window will pop up since we don’t have the hardware connected, just click cancel. This should be done for each project (if you have other iOBC projects) you would like to run via the debug interface; a Jlink.launch file will be required.

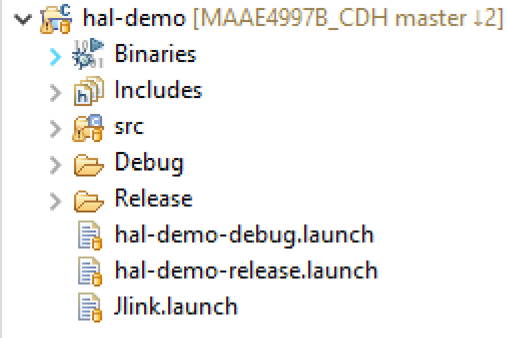


Figure 2 – hal-demo

12. The purpose of this is to add the JLink launch configuration for easy access later on this button. It is possible the repo’s .cproject file included the Jlink launch config here allready, we just want to verify the path works for your particular environment and the launch config can find JLinkGDBServerCL.exe.

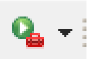


Figure 3 – eclipse external tools button and drop down menu

13. Attempt to build the project with the Hammer icon (if not clickable, try Build All (Ctrl+B), it may become visible after step 19 once you register a launch configuration into Eclipse), beside the hammer icon is a drop down that lets you select the build configuration: debug or release. Try doing both to validate setup has worked.

    Our Team never attempted to upload release code to the iOBC, we verified that it does compile. If you wish to try using release code, it’s on you to follow the ISIS QuickStart guide and work out how to do this; please document this process.

    NOTE: Verify c files in the ‘examples’ folder should have icons with slashes through them, they are excluded from the build since they are for reference only. Many other files are marked excluded in the CSP source code; these exclusions are defined in the .cproject file. The program will not compile if they are included.

Running the code on iOBC Hardware

The following steps require iOBC hardware, JLink Usb Adapter, mounting stand, 3.3 V Ac adapter. If this is your first lab with the hardware, please ensure to have Project Lead or Team Lead support you for proper setup and ESD safety procedures.

14. Follow the **"ISIS-OBC QuickStart Guide v2.2"** [**Step 4**] to hookup the JLink debug link to your computer.
15. Start Putty in the C:/ISIS/applications/PuTTy/ Folder. If you put the ISIS files someplace else, please find the applications folder to locate putty, you may attempt to use a more modern version of putty if you wish. *Document the use of a modern version of PuTTy if you do.*
16. **Follow section 7.1.2 of the iOBC quick start guide** to setup the putty COM port then start putty up, you should see the terminal window is open and showing a cursor. This acts as your debug output stream, i.e. printf(…) or log commands in the code output to the putty console.
17. In Eclipse, launch JLink from the button mentioned in above step 12.
    a. *NOTE: iOBC needs to be powered on or you will see an error, this error is shown in troubleshooting section.*
18. Verify no errors in the console output, the JLink is now armed and waiting for code to upload.
19. To run debug mode, right click "hal-demo-debug.launch" and Debug As and select the option there. This will add the debug launch configuration and can be accessed via the shortcut from here on 🐞▾. Debug will start and the code will run on the iOBC, PuTTY console will display console output. Eclipse will ask to switch to debug mode where breakpoints may be used, you may handle this pop up as you wish.
20. Interact with the program from the Putty window.

GOOD LUCK!

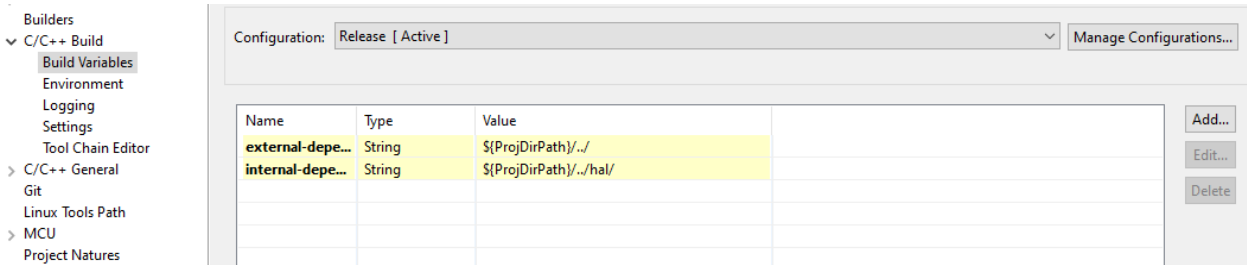
Troubleshooting

Deactivate warnings being treated as errors – If it wont compile due to warnings being errors

1. Right click on the project in the project explorer and select properties. Or select the project and choose properties under the project menu.
2. Click and expand C/C++ Build on the left side
3. Click and select Settings under C/C++ Build
4. On the right side you should see the tab Tool Settings selected and below that many folders
5. **Important: Select Debug from the Configuration pull down menu at the top, these settings are tied to either Debug or Release configuration make sure you have the one you wish to change selected here.**
6. Click and select the first Warnings folder from the top of the list
7. Verify that the last checkbox on the right side is off, "Generate errors instead of warnings (-Werror)"
8. Select Apply and Close at the bottom of the window to save the change.
9. This will remove the '-Werror' flag used as part of the compilation command eclipse executes when you build the project.

Check Build Variables and Includes paths

1. Right click on the project in the project explorer and select properties. Or select the project and choose properties under the project menu.
2. Click and expand C/C++ Build on the left side
3. Click and select Build Variables under C/C++ Build



Configuration: Release [Active]

Name	Type	Value
external-depe...	String	${ProjDirPath}/../
internal-depe...	String	${ProjDirPath}/../hal/

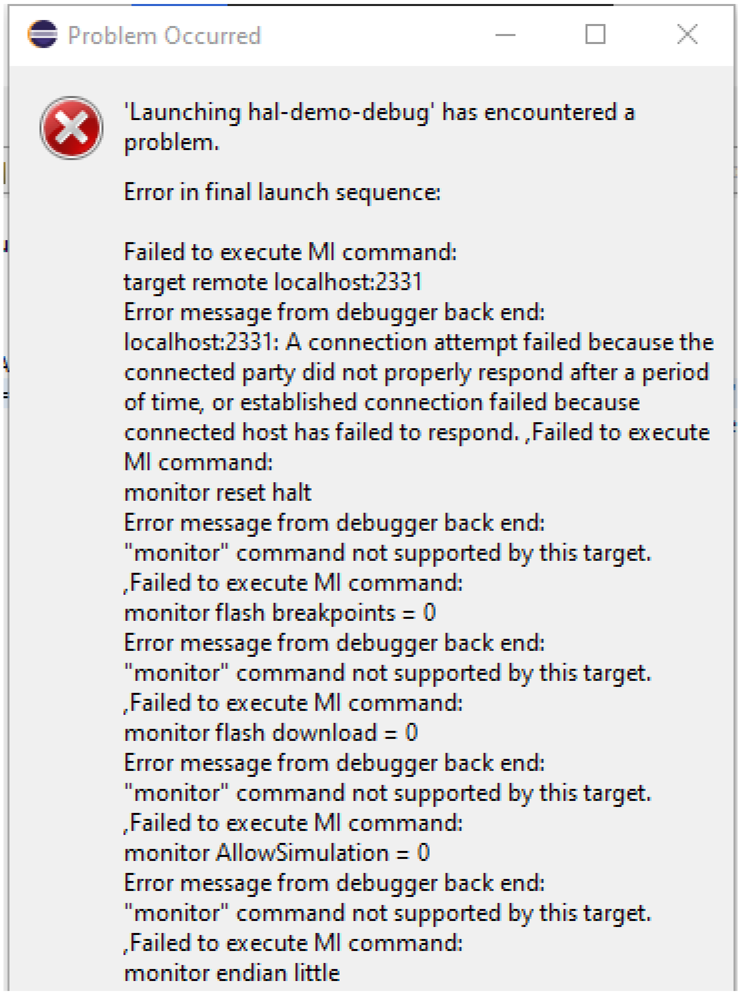
Figure 4 - Build Variables for hal-demo

4. These tags are used in Settings under C/C++ Build category in the includes or Miscellaneous categories for the Compiler and Linker respectfully. Used to define the prepended path information for imports. *Check the Includes folder under Cross ARM C Compiler under Tool Settings, under Settings, under C/C++ Build, for an example of how we define the includes paths.*
5. Notice that we use 'internal-dependencies' for the files located in the 'hal' folder of the repository. 'external-dependencies' is used for other stuff we wish to add in and just looks in the parent folder above the project folder for these.

Scary Error pop up window – You try to debug without JLink runnning

1. If you try to run the debug launch configuration and JLink is not running, this will occur after a timeout.

2. You have attempted to upload code to iOBC without the JLink connection and Eclipse will give up with an error message.



```
'Launching hal-demo-debug' has encountered a
problem.

Error in final launch sequence:

Failed to execute MI command:
target remote localhost:2331
Error message from debugger back end:
localhost:2331: A connection attempt failed because the
connected party did not properly respond after a period
of time, or established connection failed because
connected host has failed to respond. ,Failed to execute
MI command:
monitor reset halt
Error message from debugger back end:
"monitor" command not supported by this target.
,Failed to execute MI command:
monitor flash breakpoints = 0
Error message from debugger back end:
"monitor" command not supported by this target.
,Failed to execute MI command:
monitor flash download = 0
Error message from debugger back end:
"monitor" command not supported by this target.
,Failed to execute MI command:
monitor AllowSimulation = 0
Error message from debugger back end:
"monitor" command not supported by this target.
,Failed to execute MI command:
monitor endian little
```

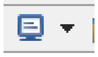
Figure 5 - Try to run program without JLink, error goes right off the screen.

3. Simply close the error and verify that JLink is running in the eclipse console. This image lets you switch consoles in eclipse, one of them should be JLink and if it's running, it will have a stop symbol lit nearby and the console should say waiting for connection with no errors in the logging.
4. Likely, JLink was closed by mistake, simply run Jlink again and then try to run the debug launch.

JLink Red Text Error – The iOBC is not powered

1. I don't have handy the exact text, but Jlink will show red error text in eclipse console if run and the iOBC is off; no power.
2. Power the iOBC and try Jlink again.

JLink Red Text Error – You ran JLink twice

1. If you try to run a second instance of JLink it will fail with an error.  This image lets you switch consoles in eclipse, one of them should be the active JLink, and if it’s running, it will have a stop symbol lit nearby and the console should say waiting for connection with no errors in the logging.
2. Verify that Jlink is running by switching to the appropriate console in eclipse. Close any consoles that are not running with the ‘X’ symbol.

Running Labs

Supported Labs

The 3 labs reviewed in the following guide were verified on April 16th of 2022. Other files in the Hal-Demo Test folder are provided 'as is' for you to experiment with. Only the Files outlined here should work as described here. Note UART lab is not fully verified as we had a hardware issue during the lab that caused the UART to malfunction on our Arduino during testing.

An excellent early lab to attempt is the Mode Switching lab. This is as close to a multi-master solution we got to, it indicates we can use the included HAL drivers and that it is possible to mode switch between slave and master, doing a master command and then when in slave mode receive a slave message. The other master shall constantly attempt to commune with the iOBC and notice that this test throws no errors and works as intended.

This seems to show that the multi-master issue can be worked around and may not be a factor.

Regarding the Hal-Demo main menu functionality:

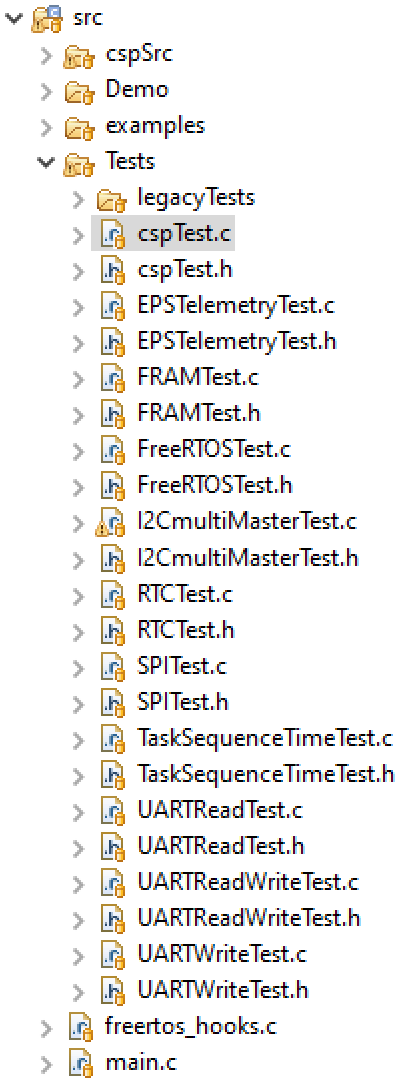
Many of our tests do not allow for returning the main menu to select another test. When a test functions entry function returns false it causes the main menu task to go idle, this is seen by the led's on the iOBC moving in a wave pattern. You must use stop in eclipse to terminate the debug application and then you may re-run it to try another test. You do not need to restart Putty or the Jlink normally.

The reason for this odd functionality is that many test entry functions will create test tasks and you could go back to the main menu and queue up more different tests. The main menu task never yields the processor until you decide to not run more tests, or a test entry function returns false to the main menu, so the tasks only run when the main menu goes to idle and yields the processor.

You may decide to edit the tests and design one to not use a task. In this case you may want to return true back to the main menu, and this lets you select a new test from the menu; good for re-running your test without a reboot.

Test Files in the hal-demo project

The project has a folder under src called Tests, in this folder are our test files that are referenced by the main menu code located in main.c. The legacyTests folder includes mostly original Hal-demo tests.



```
src
    cspSrc
    Demo
    examples
    Tests
        legacyTests
        cspTest.c
        cspTest.h
        EPSTelemetryTest.c
        EPSTelemetryTest.h
        FRAMTest.c
        FRAMTest.h
        FreeRTOSTest.c
        FreeRTOSTest.h
        I2CmultiMasterTest.c
        I2CmultiMasterTest.h
        RTCTest.c
        RTCTest.h
        SPITest.c
        SPITest.h
        TaskSequenceTimeTest.c
        TaskSequenceTimeTest.h
        UARTReadTest.c
        UARTReadTest.h
        UARTReadWriteTest.c
        UARTReadWriteTest.h
        UARTWriteTest.c
        UARTWriteTest.h
    freertos_hooks.c
    main.c
```

Figure 6 - our hal-demo test files

Examples folder contains CSP example code that should not be expected to compile and should be marked as excluded.

Demo folder has code that was never used or inspected, we don't know what it does.

cspSrc is the source code for CSP v1.6. It has many exclusions to allow for compilation, files not required for our future implementation of CSP.

Mode Switching lab

Prerequisites:

You have completed a test lab and are familiar with the safe setup and operations of the iOBC. You have run our version of hal-demo project on the iOBC in isolation with no attached devices and can run a simple test such as LED test. You have a general knowledge of I2C and TWI. You are familiar with the Multi-master I2C issue. You have inspected the code for the test you are about to run and understand it.

Goal:

Demonstrate an ability to switch I2C modes in a multiple task system with 2 connected I2C devices, one a slave and one a master. This is a key capability of a multi-master capable I2C device.

Components:

- iOBC
- JLink
- Laptop to run the IDE on and observe Putty on.
- Anti-ESD mat with ground line.
- Grounding bracelets
- Breadboard
- Little linking wires
- 2 Arduinos:
    - We used an Arduino Mega and an Uno
    - You need to see console output for your Arduino
    - They need to support I2C connections
    - If you have not provided pull-up resistors for the TWI, ensure the Arduino is providing it. I2C must have pull-ups on the TWI to function. The expected range we can likely use is from 5k to 40k roughly speaking. See the I2C spec, our 2022 Final Report for more information. When we did the test, the Arduino Mega provided internal pull-ups and it just worked.
- Note we run I2C at 100 kHz in this test.

Steps:

1. Follow the normal procedure to setup the iOBC as per your first lab with the project lead.
2. Make sure iOBC is off and hook up the I2C bus to the two Arduinos and iOBC via the breadboard. Follow the available pinout guides and triple check your connections.
3. Find .ino files for Arduino master and slave in the “Supplementary_Content” folder. Run these files on your Arduinos so one is master, and one is slave.
4. You may start up the Arduinos and get them running their code.
    a. The slave waits for a call from iOBC with data “send me your data” and returns “Hello iOBC” as a response.
    b. The master Arduino should attempt to continuously send the message “Hello iOBC”
5. Power the iOBC, start Jlink, run the Debug hal-demo.
6. From Putty main menu select 19 “MultiMaster Test”.
7. Expected Function:

a. The Test starts 2 tasks: A Mode Director that manages the current I2C mode based on a couple flags. This task runs either the master write-read test code or the slave read code. The other test acts to poll a slave device with a random chance (25% of the time); this causes a switch into master mode and once that is done returns to slave mode for at least 2 seconds before random chance kicks back in.

b. The debug output text shall only print when the 25% chance master call occurs to prevent spam. **This will demonstrate receipt of data from the slave and the slave output should show data having been sent to it.**

c. Also, when this occurs, **we will see debug output of the last message received from the master**. This indicates when we are in slave mode, we are picking up the message coming in from the master Arduino. This will show the message with 0xFF (Arduinos string terminator character) characters on the end repeating. Since we set the packet size to 32 and the data being sent is less than this, the HAL driver will duplicate the last character to the end of the read length specified.

d. In practice all data sent over the line will start with a length value so we can use this to ignore any garbage padding on the end of a packet. This garbage padding is simply duplication of the last value received from the I2C data line when we read beyond the size of sent data. This behavior stems from how our HAL driver works.

Results:

This test is designed to show we can I2C mode switch with HAL Drivers and as long as the incoming message is repeating, we can also pick it up once the iOBC switches back from its master mode business into slave mode again.

Note that the test code for the master mode call is blocking in nature so it must finish, time out, or error out, before iOBC can switch back to slave mode. Our demo test will always switch back to slave after a master mode call regardless of outcome.

It should also run error free; all our driver test calls can return error codes; we should not see any. However, we have not tested extensively and have not varied the test parameters to push it to the limits. We have not quantified any metrics, nor have we simulated a more complex schedule for iOBC to represent realistic delays in tasks scheduling; open questions and potential future work to do.

How this can be designed and implemented into CSP so we can recognise CSP Packets from the AX100 is an open question (our report does cover some aspects however) and a challenging task. I imagine some sort of in-between driver wrapper CSP can interact with that handles mode switching; it would simplify to one driver API for CSP to use. I didn’t get too deep into figuring out the CSP integration so its on you! Good Luck.

Task Sequence Time Test Lab

Prerequisites:

You have completed a test lab and are familiar with the safe setup and operations of the iOBC. You have run our version of hal-demo project on the iOBC in isolation with no attached devices and can run a simple test such as LED test.

Goal:

Showcase task timing and parsing of time codes

Components:

- iOBC
- JLink
- Laptop to run the IDE on and observe Putty on.
- Anti-ESD mat with ground line.
- Grounding bracelets

Steps:

1. Follow the normal procedure to setup the iOBC as per your first lab with the project lead.
2. Power the iOBC, start Jlink, run the Debug hal-demo.
3. From Putty main menu select 27 “Task Sequence Time Test”.
4. Expected Function:
   a. Runs a series of parsing tests
   b. Please wait 30 seconds
   c. Task 1 should run and output a message
   d. At 50 seconds it runs again and outputs a message

Results:

The results are self evident, please read through the code and Final Report to understand more about what is occurring under the hood.

UART Read Test Lab

Prerequisites:

You have completed a test lab and are familiar with the safe setup and operations of the iOBC. You have run our version of hal-demo project on the iOBC in isolation with no attached devices and can run a simple test such as LED test. You are familiar with how UART works and how to hook it up. You are familiar with Arduino coding.

Note:

This Test may not work out of the box and may require modification of the provided Arduino Code and or iOBC code on the fly. UART has been a troublesome bus to work with and we even had a hardware short circuit occur with our Arduino Mega during verification that caused it to read its own UART register over and over forever.

We could not get the UART Write Read test to work again, we only Verified the Read test. Due to the hardware malfunction, we could not continue work on the code and it is left to pass on to you.

Our UART team member told us we need to unplug UART for Arduinos when ever we made a change to the code, to dump the buffers and sort of reset it. Make sure to be read up on UART and Arduinos and understand how it works. I worry this indiscriminate pulling of powered UART lines was risky and I suggest fully powering down the Arduino instead.

Goal:

Showcase UART functionality

Components:

- iOBC
- JLink
- Laptop to run the IDE on and observe Putty on.
- Anti-ESD mat with ground line.
- Grounding bracelets
- Arduino acting as a UART source of data.
    - Ideally has console output as well to see data from iOBC

Steps:

1. Follow the normal procedure to setup the iOBC as per your first lab with the project lead.
2. You may start up the Arduino and get it running code “UART_Test_Read.ino”.
    a. You want the Arduino to constantly output “123a”
3. Power the iOBC, start Jlink, run the Debug hal-demo.
4. From Putty main menu select 21 “UART Read Test”.
5. Expected Function:
    a. Loops endlessly
    b. Should see 123aa on the debug input (see why in results)
    c. Also debug output indicating a successful read

Results:

The iOBC is setup to read in 5 bytes into a 5 long buffer. This causes the string to keep its order and not get scrambled. However, since the actual data is only 4 long, we get the effect we saw before from HAL drivers where it duplicates the last value read into our buffer; hence the duplicate ‘a’.

We noticed message scrambling when the test task had a delay, so we set vTaskDelay(0) to prevent delay. In practice this needs to be understood more completely. We are under the impression that if we start a UART task, it must complete without switching tasks or we risk malformed/misaligned data. Worthy of future study to make sure UART data comes through correctly in a variety of circumstances.

Please read through the code and Final Report to understand more about what is occurring under the hood with UART tests and labs during the term.

It is on you to try out UART Read and Write test and see if you can get it working. The Arduino Code is provided as a starting point. Good Luck.