

Developer Manual for Dynamic Parking Signage Project

ELEC 491 Capstone Design Project

Team PL-89

Geoff Goodwin-Wilson

Timothy Cheng

Hiu Lok Cheung

Yuyang He

Jared West

TABLE OF CONTENTS

Glossary	2
1. Summary	3
2. Firmware Development on STM32 and Goodisplay Development Boards	4
2.1 Getting Started	4
2.2 IDE and Debugger	4
2.3 Datasheets	5
2.4 Debugging	5
3. Hardware	5
3.1 General Usage	5
3.1.1 Powering On:	5
3.1.2 Flashing Firmware:	6
3.1.3 Communicating to the device over UART:	7
3.1.4 Hardware Debugging	7
3.1.5 Header functions:	8
3.2 Assembly Instructions	10
3.2.1 Sanding and Touch-up	10
3.2.2 Painting the Enclosure	10
3.2.3 Installing and Soldering the LED Strips	11
3.2.5 Installing LED Diffusers	13
3.2.6 Installing Plexiglass Cover	13
3.2.7 Installing Heat-set Inserts	14
3.2.8 Installing the Display	15
3.2.9 Installing Cable Gland	16
3.2.10 Installing PCB	17
3.2.11 Installing Charge-port and Charging Indicator	18
3.2.12 Solar Cable and Connectors Wiring	19
3.2.13 Gasket Fabrication and Installation	21
3.2.14 Lid and Sign Installation	21
3.3 Known Issues and Resolutions	21
3.3.1 Schematic & PCB Layout	21
3.3.2 Enclosure	22
4. Team Contact Information	24
Appendix A: File Structure Overview	25
Appendix B: Debugging and Troubleshooting Q&A	26

Glossary

E-Ink	<i>Electronic Ink</i>
FPC	Flat Printed Circuit
GCC	<i>GNU Compiler Collection</i>
GPIO	<i>General-Purpose Input/Output</i>
GUI	<i>Graphical User Interface</i>
HTTPS	Hypertext Transfer Protocol Secure
IDE	<i>Integrated Development Environment</i>
IP65	<i>Ingress Protection Standard</i>
KB	<i>Kilo-Byte</i>
MCU	<i>Microcontroller Unit</i>
PCB	<i>Printed Circuit Board</i>
SEEDS	<i>Social Ecological Economic Development Studies</i>
SLS	<i>Selective Laser Sintering</i>
UART	<i>Universal Asynchronous Receiver/Transmitter</i>
UBC	<i>University of British Columbia</i>
UBC PAS	<i>UBC Parking and Access Services</i>

1. Summary

The target audience of this document is for future capstone teams working with UBC PAS on improving the Dynamic Parking Signage project. The purpose of this document is to help future teams understand what tools and resources to use in order to quickly transition into development and reduce time trying to figure things out. We understand the technical challenges for a project of this complexity. We have included resources and tools we used and found useful in our team with the project. This document will serve as an operation guide for the device, including setup procedures, small nuances, hardware assembly and understanding, as well as standard strategies we have used over the course of the project.

As a final note, get used to being uncomfortable, learning, reading lots of documentation, forums and Googling. Good luck!

From Team 089, April 2020

2. Firmware Development on STM32 and Goodisplay Development Boards

2.1 Getting Started

There are three main steps in embedded programming. First, you write the code, which in this capstone project is in C, a fast and powerful yet dangerous (with great power comes great responsibility - pointers, references, magic!) language. This step also includes configuring hardware such as clock speed, peripherals, GPIO, UART using C. Then, you need to compile the code (makefiles) and find a way to upload it into the microcontroller (MCU) flash memory where it finally executes your program. Lastly, you need a way to debug. Often, you will find yourself at 3:00 AM stepping through the instructions one at a time to find a pesky bug.

Back in the day, an embedded programmer would need to use three separate tools to accomplish this task. Today, more MCU manufacturers are creating their own toolchains that help developers prototype rapidly by combining three tools into one. However, none of them are perfect as some limit program size for free users (e.g. maximum 8KB), are expensive (\$1,000/license), or difficult to use as the graphical user interface is not user-friendly and buggy. Generally, manufacturers modify the eclipse (an open-sourced integrated development environment) codebase by adding their own features. Alternatively, one can use unmodified eclipse, visual studio with plugins, or any text editor with a toolchain (to compile the code. e.g. GCC) and STM32CubeProgrammer to load the binary file onto a development board or MCU.

For our project, we used STM32CubeIDE as our development environment as it is free with the 3 aforementioned components built-in. Like IDEs developed by other manufacturers, STM32CubeIDE is also an IDE based on eclipse with STM32CubeMX (code generation tool for configuration i.e. by interacting with a GUI, relevant code is generated for you) and STM32CubeProgrammer (flash on to an MCU). With the ST-Link V2 debugger or development board with builtin ST-Link, one may also debug the program line by line.

2.2 IDE and Debugger

The IDE for programming the microcontroller is STM32CubeIDE. To download go to: <https://www.st.com/en/development-tools/stm32cubeide.html> for the newest version as ST continues to update their software. ST requires an account to be created to download any of their programs. We have provided the relevant credentials in the list of deliverables document.

2.3 Datasheets

During our development process, we used the following ST components (with all of their respective datasheets included in a zipped file in this folder called “datasheets”):

- MCU stm32f103zg (display dev board MCU)
- MCU stm32l496ag (cellular dev board MCU)
- BOARD p-l496-cell02 (cellular dev board)
- BOARD stm32 nucleo-144 board

Further datasheets for the STM32 series are all provided on the website: www.st.com.

2.4 Debugging

Please see appendix B for some common compilation issues. The appendix is in a question and answer format to assist you in debugging in your darkest hours.

As we have struggled in the development process, we adopted a method which is documenting what we did each step of the way during development to track the changes and where we suspect the bugs to be at. See appendix A to find out where these annotations are in the files we have included in the list of deliverables.

3. Hardware

3.1 General Usage

3.1.1 Powering On:

Using the hardware is made very simple with the use of an integrated PCB. The first step is to power the system on. This can be done in multiple different ways. If you have access to 18650 battery cells (should be included in the carry-over supplies), you can insert at least two in series and any amount in parallel which will turn the system on when the long-term shutdown switch is in the “ON” position. If batteries are unavailable, you can power the system through a USB micro-b connector by the large display header, which will provide USB_5V to the 5V rail. There is also an auxiliary power connector which can supply the 5V and 3V3 rails using an external power supply. You may want to disconnect the LED_OUT connector or set the LED_CTRL pin high in firmware to ensure that the LED driver & LEDs are not drawing unnecessary power while debugging.

3.1.3 Communicating to the device over UART:

As long as UART is configured in firmware, you can communicate to the device through the USB Micro-B header since there is a USB to UART converter chip on the PCB. This can be useful for sending and receiving commands from your computer through a terminal (PuTTY or similar), which is helpful for debugging.

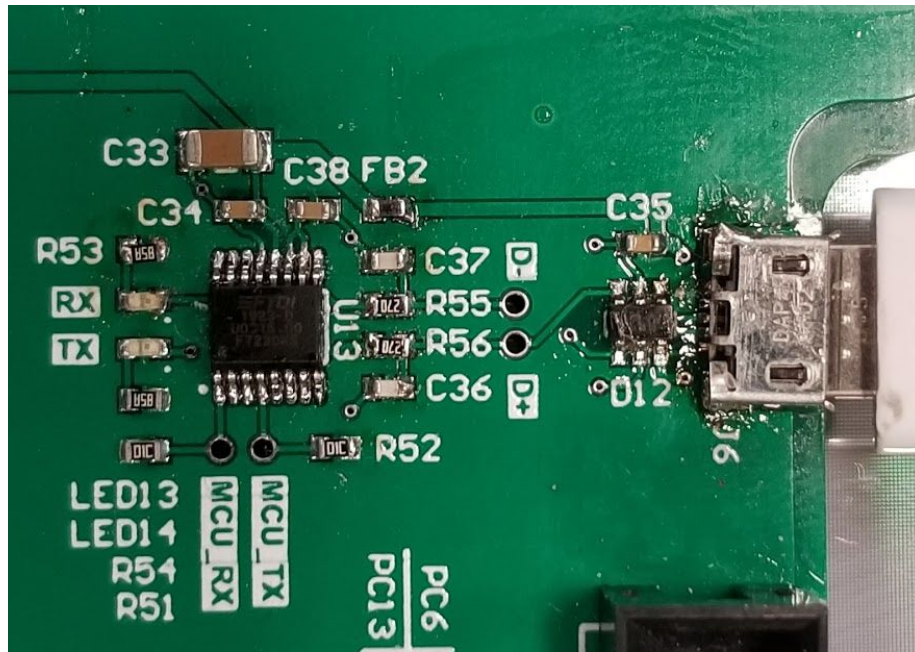


Figure 3. USB to UART Converter Chip

3.1.4 Hardware Debugging

Test points are deliberately added into the hardware in order to probe with the oscilloscope. An oscilloscope probe should fit directly inside each test point, but a ground wire may need to be soldered on to the PCB in order to tap off GND.

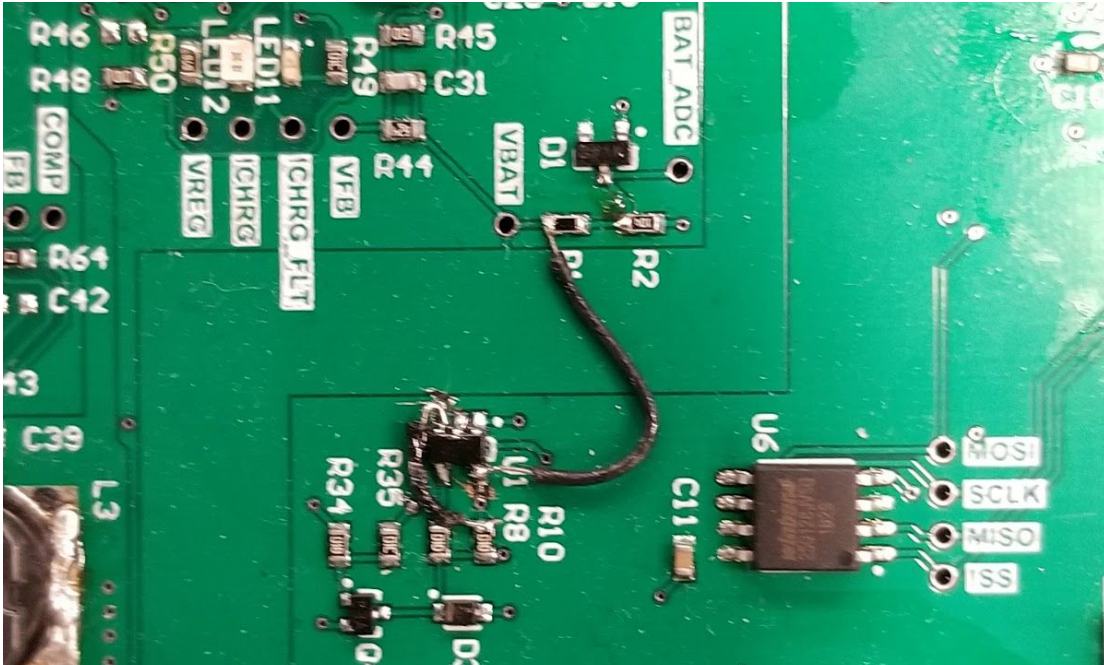


Figure 4. Ground Wire Jumper Connection

3.1.5 Header functions:

There are multiple headers soldered onto the PCB, each with different functionality. In Figure 5, the cellular and display headers are highlighted using red and orange squares respectively. Additional headers are labeled on the PCB.

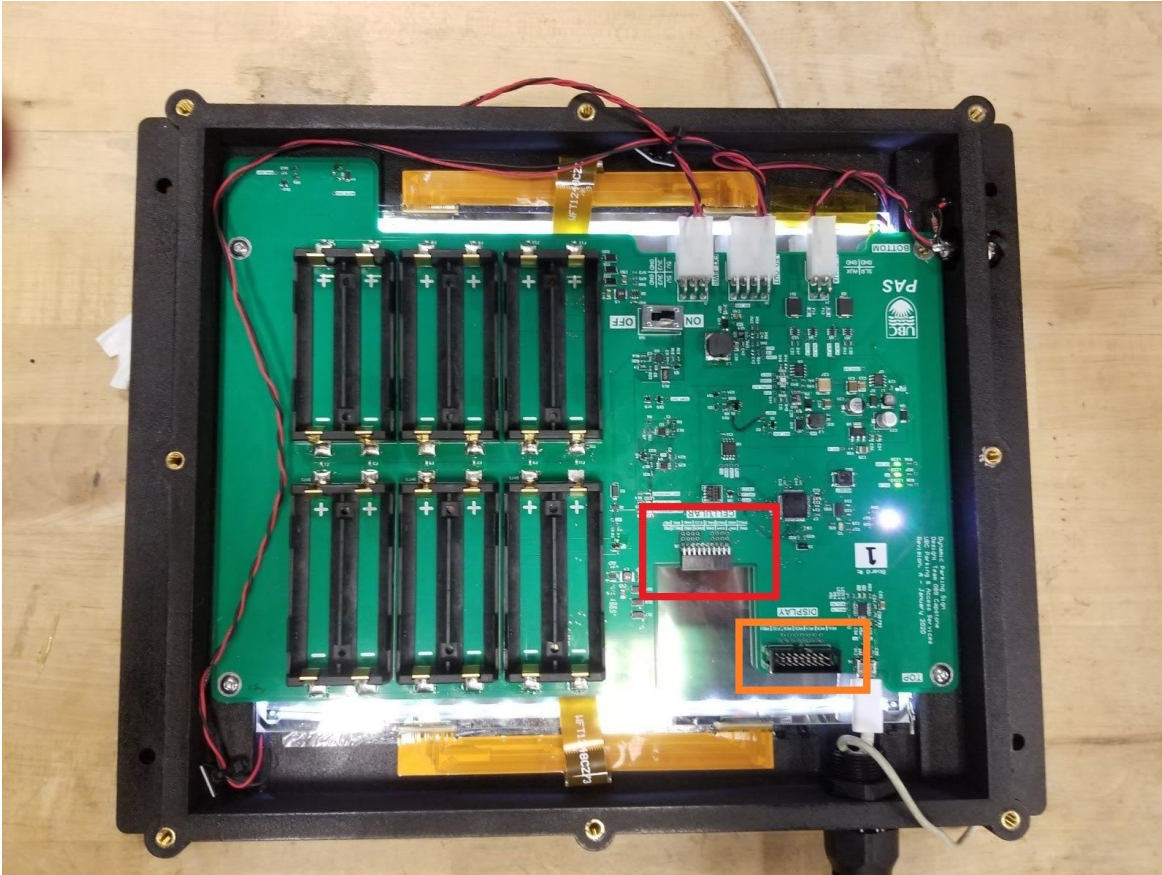


Figure 5. Location for cellular header and display header

Each header has specific functionality as follows:

1. Cellular

As shown in Figure 5, the header indicated by the red rectangle is intended to connect to an ST cellular device which will be included in the carryover package. The device follows the ST-Mod standard, which is used to be compatible with many ST devices, but the hardware configuration on the PCB is designed to only connect with this specific cellular module board.

2. Display

You can either connect the display directly from its ribbon headers on the rear of the PCB or through the large ribbon header indicated by the orange rectangle to the GooDisplay display breakout board. Having multiple ways to connect the display is meant to assist with debugging by removing the on-board driving circuitry as a variable.

3. SLR/AUX

Intended as a connection to the solar panel or a 24V, 5.5mm wall adapter. Both are

included in the carry-over package. The pinout is written in silkscreen on the PCB.

4. **LED_OUT**

Connects the LED driver's 12V output directly to the screen illumination LEDs. The four top connections are 12V, and the four bottom connections are GND

5. **EXT_PWR**

Used to power the system from an external power supply. Contains 5V, 3V3 and GND rails. The pinout is written in silkscreen on the PCB.

3.2 Assembly Instructions

The assembly instructions for all of the hardware are listed below in chronological order.

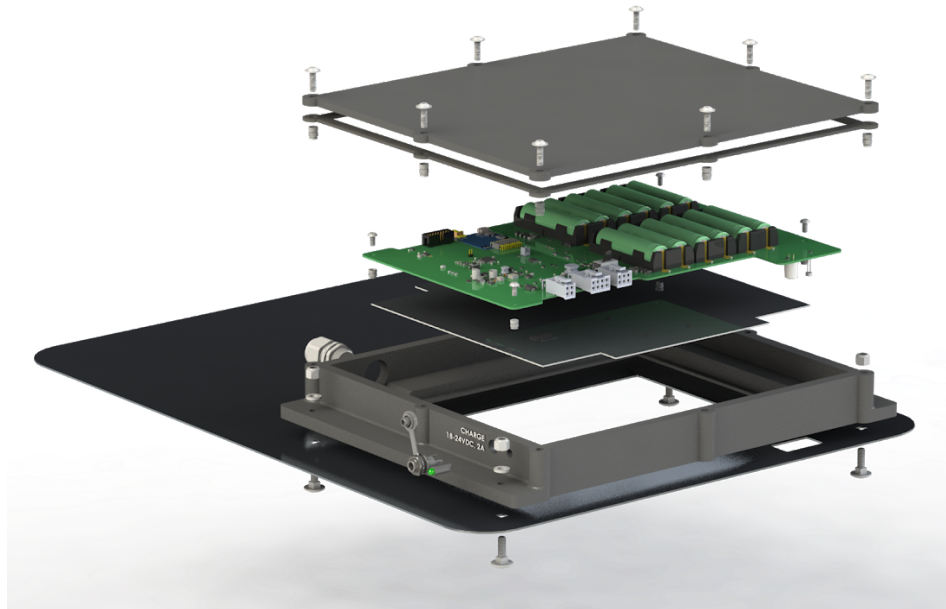


Figure 6. Enclosure exploded view

3.2.1 Sanding and Touch-up

Coat with XTC-3D and sand down all gasket and rubber sealing surfaces (for cable gland, charge port, etc.). Make sure each surface is smooth to ensure proper sealing

3.2.2 Painting the Enclosure

We painted the enclosure with black spray paint since SLS Nylon-12 is generally white by default. Before you paint an enclosure, sand down the gasket surface to ensure tight contact with gasket adhesive.

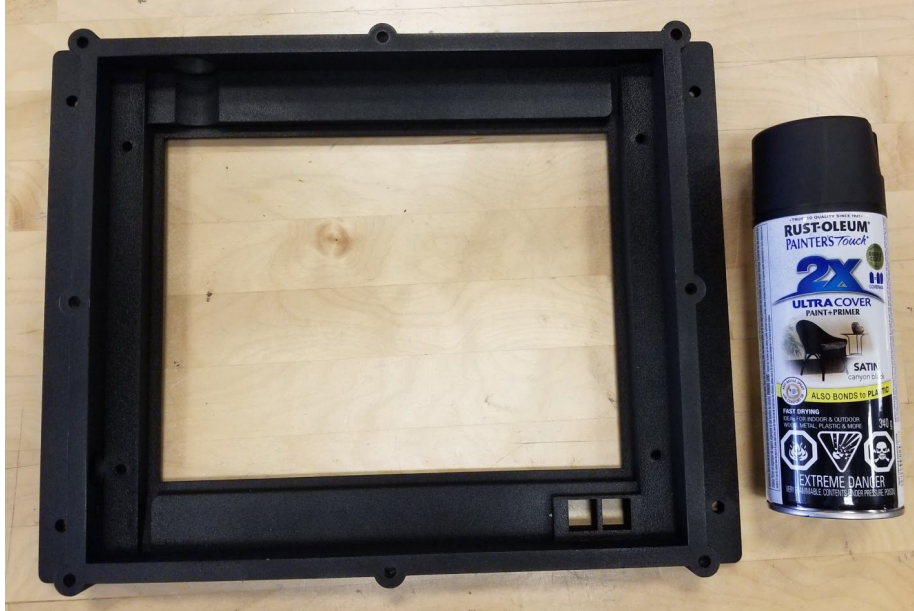


Figure 7. Painted Enclosure

3.2.3 Installing and Soldering the LED Strips

Install the LED strips on the inner sides of the enclosure. The default adhesive on the strips is pretty weak so reinforce it with superglue or similar. Make sure the orientation of the strips is such that the positive terminals are facing upwards.



Figure 8. Enclosure with LED Strip Adhered

Solder the ends of two sets of LED strips together with solid-core wire, and then solder red/black wires for PWR/GND respectively on the ends of each LED strip. Adding more strip segments just connects more LEDs in parallel, so it does not really matter how you wire them up as long as it is convenient. Make sure all the wires are coming out near the PCB connectors to avoid lone wire distances inside the enclosure.

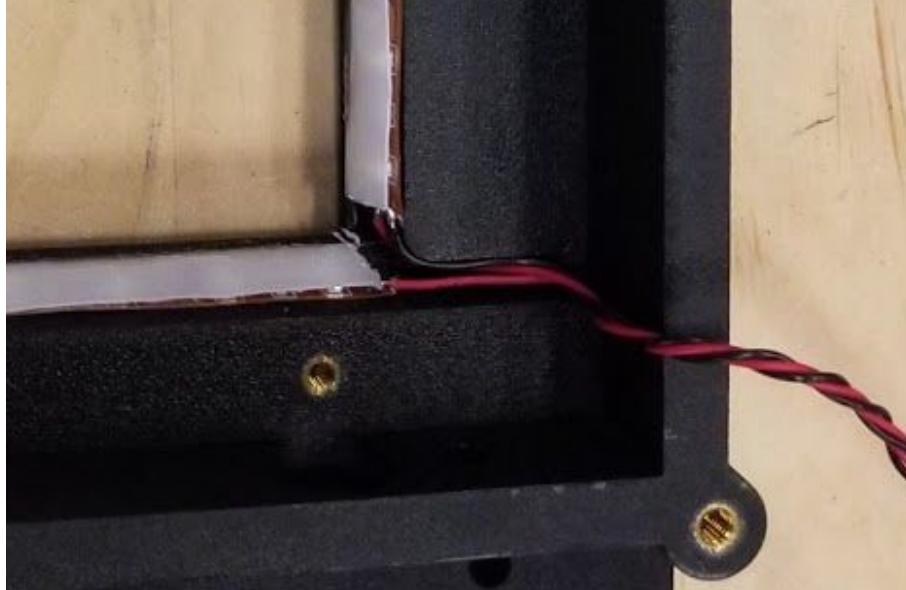


Figure 9. LED Power and Ground Wires



Figure 10. Connections Between LED Strips

3.2.5 Installing LED Diffusers

Cut the LED diffusers to their appropriate size and super-glue them into place. Make sure that the diffusers are not interfering with where the display goes. They should be flush or below the display resting ledge.

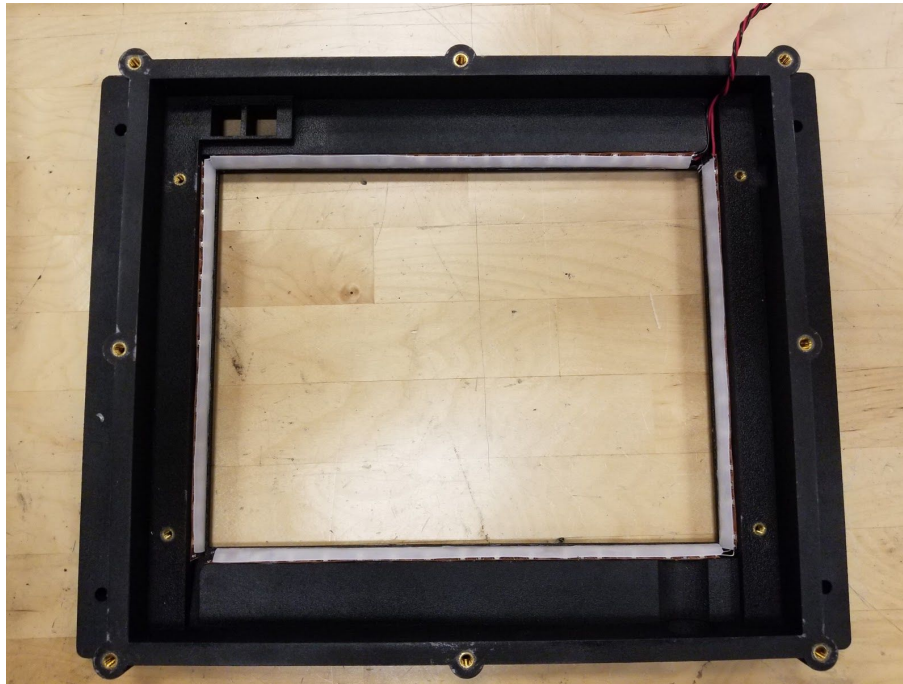


Figure 11. Enclosure with LED Strip and Diffuser Installed

3.2.6 Installing Plexiglass Cover

Apply silicone adhesive to the edges of the screen indent and place the acrylic window on top. It is important that the adhesive has no gaps in coverage to avoid water ingress.



Figure 12. Enclosure with Silicone Adhesive Applied

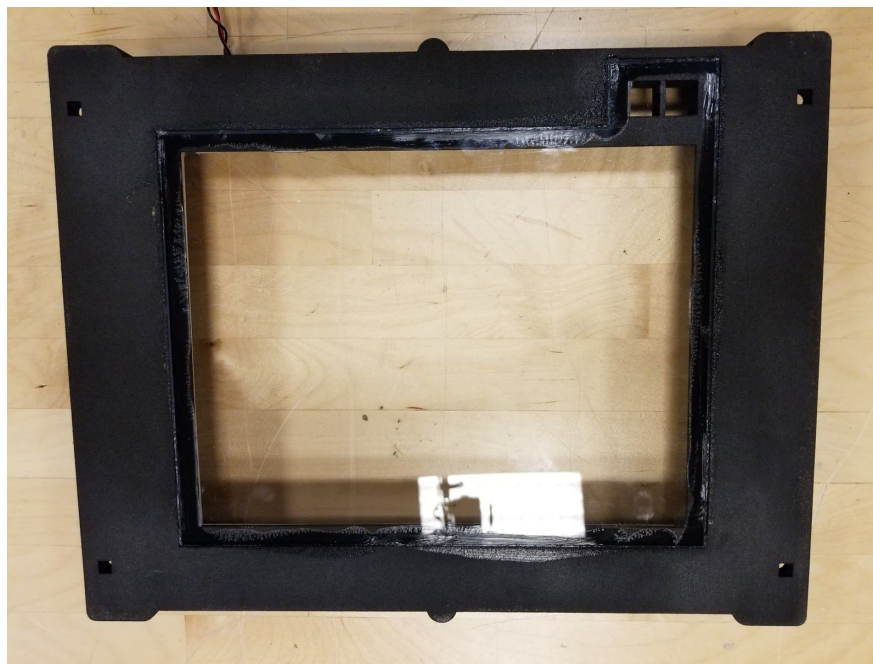


Figure 13. Enclosure with Plexiglass Cover Installed

3.2.7 Installing Heat-set Inserts

Use a wide tip soldering iron to install heat-set threaded inserts. Make sure that each insert is aligned vertically and is flush or below flush with the enclosure surfaces. After all of the inserts are done, clean up the “overflow” nylon from the edges with a sharp knife.



Figure 14. Enclosure with Heat-Set Inserts

3.2.8 Installing the Display

Install some double-sided tape on the top and bottom of where the display sits. Place the display on top and make sure it is aligned in the center (no part of the display is cut off). I recommend taping the display in to start so it is not permanent. Make sure the display is in the correct orientation - this information can be found on the datasheet.

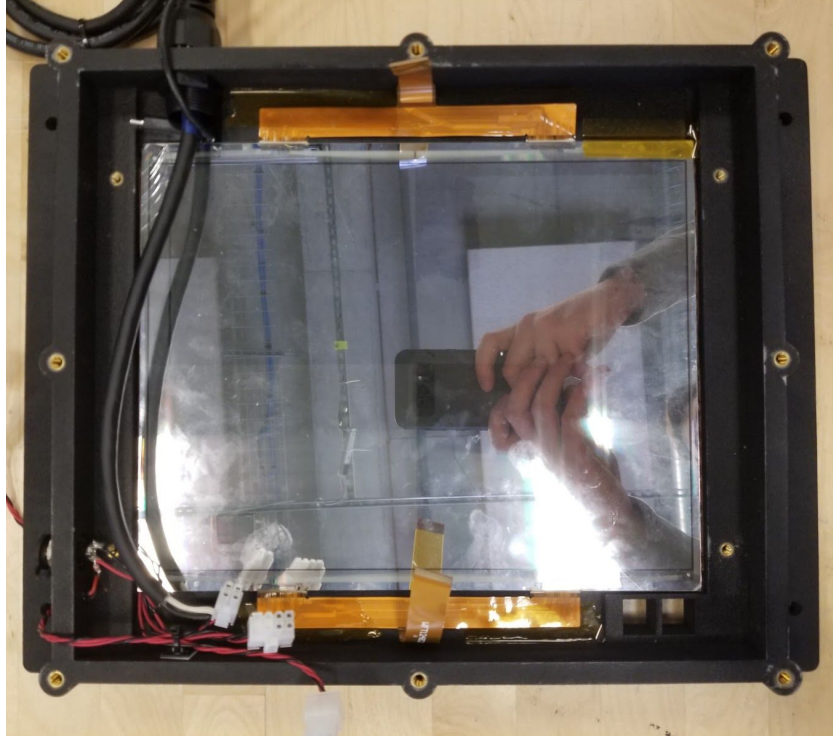


Figure 15. Display Mounted in Enclosure

3.2.9 Installing Cable Gland

Note: This cable-gland has to be installed incorrectly with the current enclosure setup due to the incorrect hole for it in the enclosure. To ensure a waterproof setup, you may need to fill the inside of the cable gland with silicone adhesive after the solar panel cable is installed.



Figure 16. Installed Cable Gland

3.2.10 Installing PCB

Place the board in the correct orientation so the sensors are inside their pockets. Connect display ribbon cables to the headers on the bottom of the board. This can be tricky due to space constraints. Install each fastener except for the one by the charging port. It will interfere with the charge port terminals (this is fixed in the next enclosure revision).

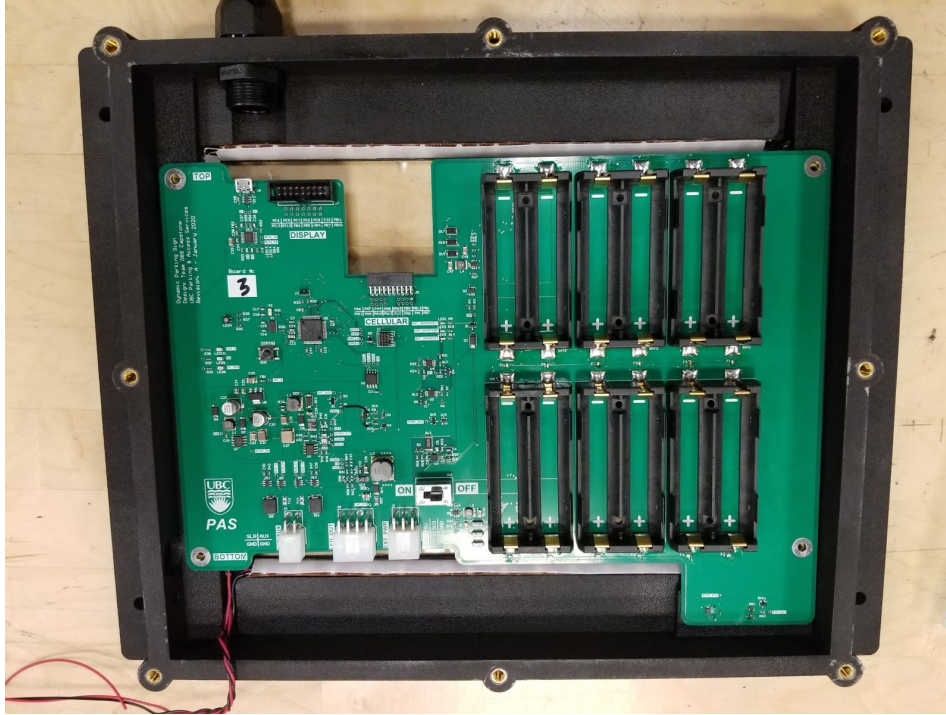


Figure 17. Enclosure with Installed PCB

3.2.11 Installing Charge-port and Charging Indicator

Cut off the S/C terminal (labeled) on the charge port. Insert the charge port and 5mm LED into their designated holes. Epoxy the back of the LED to ensure no water ingress. This is not necessary for the charge port, it is already IP65 rated. Make sure the charge port rubber gasket is on the inside of the enclosure.

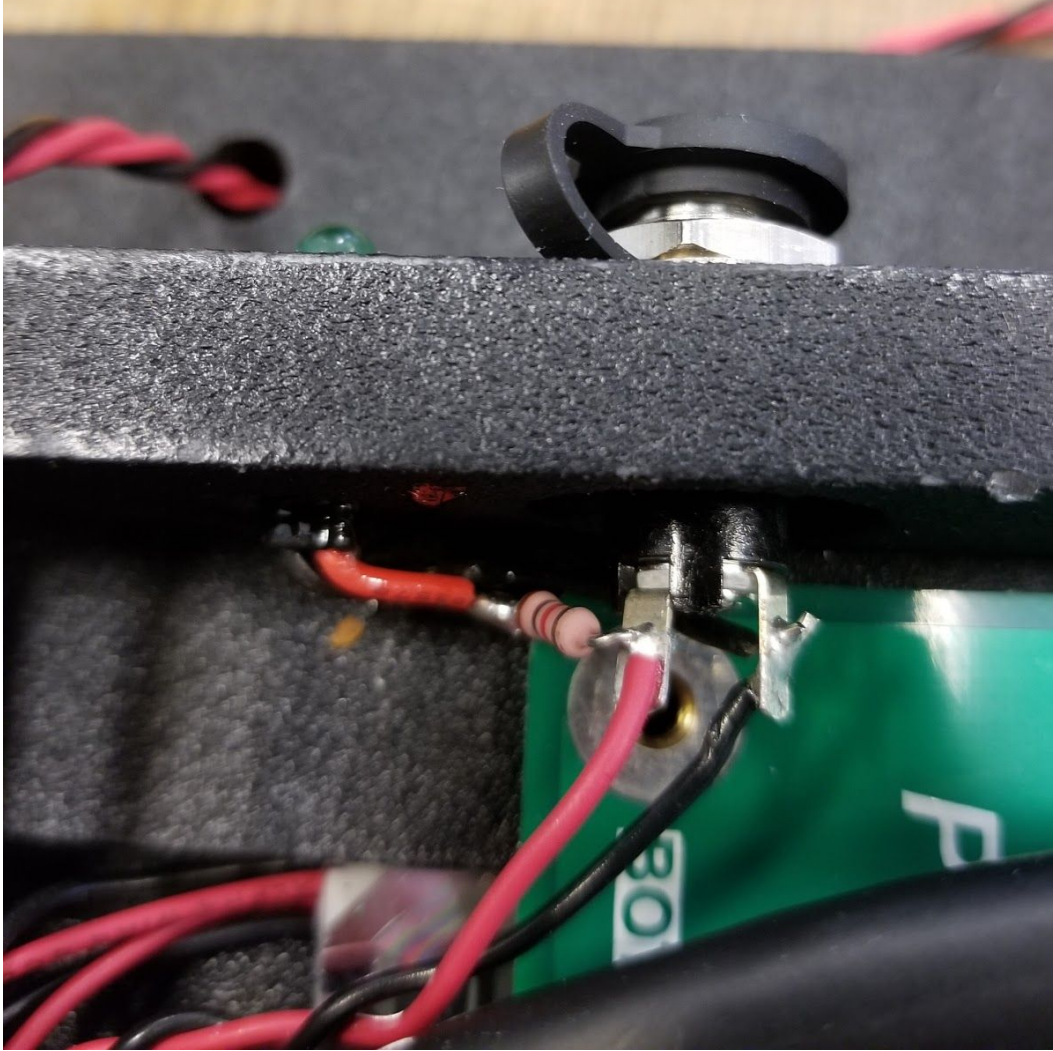


Figure 18. Charge Indicator Connection

3.2.12 Solar Cable and Connectors Wiring

Crimp the solar cable and insert the contacts into the correct places in the SLR/AUX Molex connector. Run the cable out of the enclosure through the cable gland. Take appropriate measures to waterproof the cable gland.

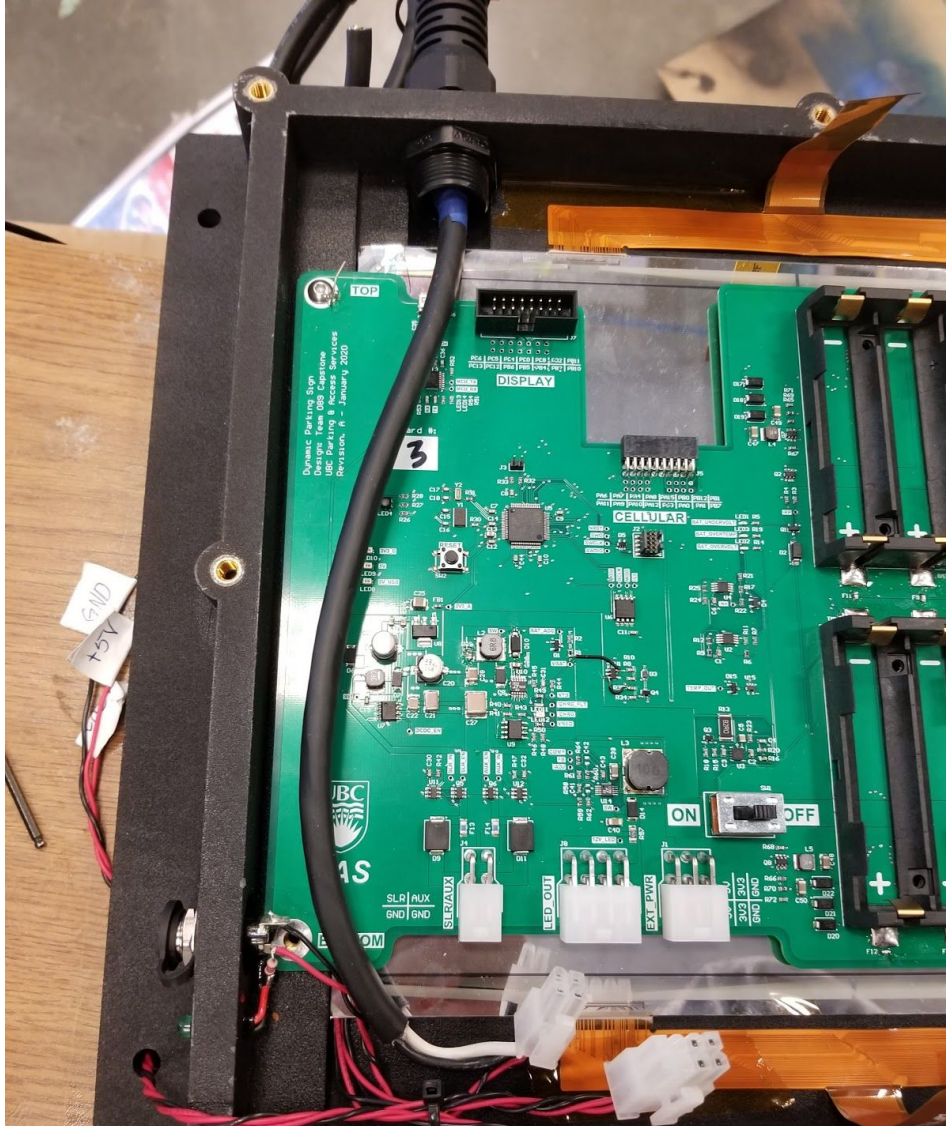


Figure 19. Solar Panel Cable Installed

Crimp the ends of the wires and insert them into their respective connectors. Make sure all the wires are cut to the shortest length. All wires should be twisted to ensure neatness.

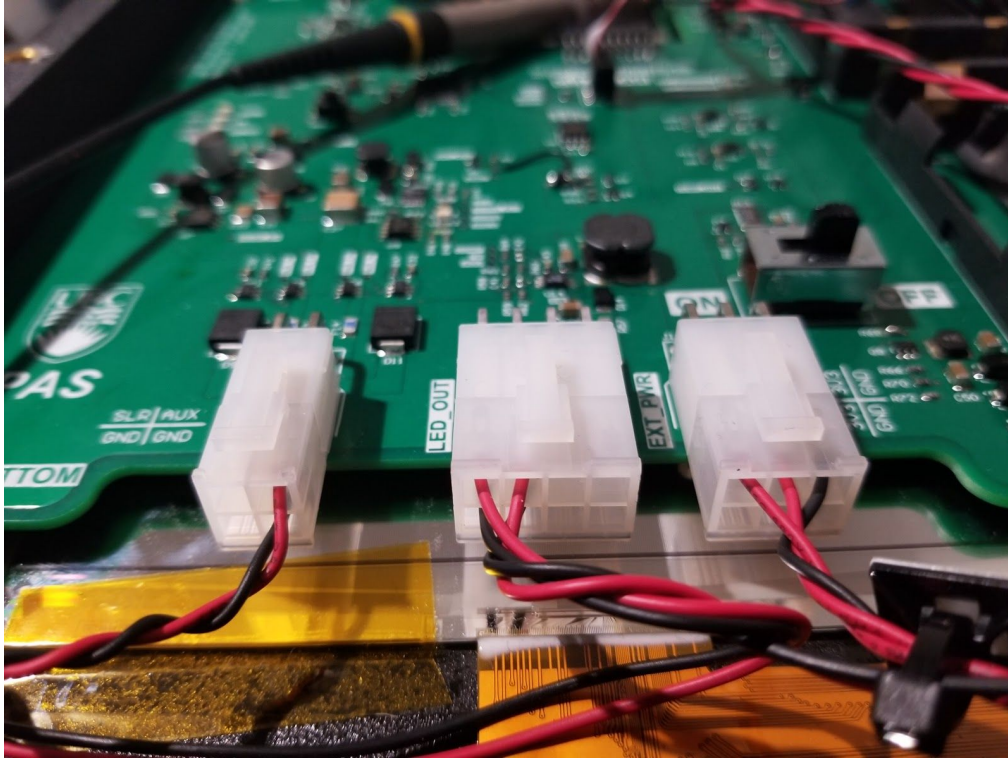


Figure 20. Wiring Connections to PCB

3.2.13 Gasket Fabrication and Installation

Print out the .dxf file for the gasket surface and cut out the shape on a piece of closed-cell foam material. Adhere the gasket on the lid-to-enclosure gasket surface.

3.2.14 Lid and Sign Installation

The lid and sign can be fastened to the threaded inserts and carriage bolts/lock nuts/washers respectively. Ensure the lid properly compresses the gasket (50%, even compression).

3.3 Known Issues and Resolutions

There are a number of known issues with each part of the system, some of which are fixed in the current design files, others that still need to be implemented. This section should serve as a good to-do list for the new hardware team for Revision B of the design.

3.3.1 Schematic & PCB Layout

Unfinished:

1. The solar charging MPPT chip needs a ground pad on the footprint. Currently, the solder mask is scraped off the PCB for a ground pad

2. C5 needs to be a 0805 footprint instead of 0603
3. USB Micro-B connector footprint is wrong. Missing two thru-holes by the connector pins.
4. Change 5V regulator Digi-Key part number to LM22670MRE-5.0/NOPBCT-ND. It is currently the adjustable option, -5.0 is a fixed 5V output version
5. The under-voltage comparator has 500mV of hysteresis instead of the intended 200mV. Change hysteresis resistor divider to fix this
6. Fault AND gate footprint VCC and VSS are flipped
7. All vias should be tented to prevent shorts
8. The charging controller should have a diode-OR controller from VBAT to 24V_IN. The 5V buck converter should have 24V_IN input instead of VBAT input. This configuration is shown on the example schematic of the charging controller datasheet
9. Display ribbon headers should be moved ~0.5cm leftwards to properly align with display ribbon cables
10. Indicator LEDs should be depopulated for actual use to reduce operating current. More measures should be taken to reduce the quiescent current in the battery management circuit.
11. Current leaks through the cell balancing chip when a single cell is inserted. Look for a way to solve this
12. The LED_OUT connector should be 4 pin instead of 8 pin - 2x +12V and 2x GND.
13. Adjust the number of batteries in parallel for experimental energy consumption to meet the 2-week requirement
14. The MCU pin text designation on the display header is mislabelled on Sensors & IO sheet for PB13, PB14, and PB15
15. Address PCB testing issues in the validation document
16. Use precision resistors (0.1%) on under-voltage and over-temperature comparator threshold dividers
17. Consider making custom breakout boards mapping the display ribbon cables to standard ribbon headers. This will make the design much easier to manufacture. The hardest part right now is connecting the display to the back of the PCB. This will also eliminate all components on the back of the PCB

3.3.2 Enclosure

Finished:

1. Charge port and indicator LED should be moved in order to avoid conflicts with PCB fastener
2. Enclosure hole for cable gland has incorrect dimensions

Unfinished:

1. Move hole for cable gland so it does not interfere with the top of the enclosure

2. Add slot on top of enclosure and holes to mount metal drip-edge to prevent water ingress
3. All gasket and rubber seal surfaces should be coated with XTC-3D and sanded to ensure smooth contact surface
4. Add a ledge near the cable gland cutout for the display to sit on. From lack of support, one of the displays cracked on that corner
5. Investigate using an LED backlight for this type of display. This will reduce vertical enclosure footprint by removing the need for LED strips
6. Accommodate size and mounting for custom display ribbon header adapter boards (if you choose to design them)

4. Team Contact Information

Name	Phone Number	Email
Geoff Goodwin-Wilson	604-690-1008	goodwinwilsongeoff@gmail.com
Timothy Cheng	604-783-2421	timothycheng222@alumni.ubc.ca
Jared West	250-961-3039	jaredwest1994@gmail.com
Yuyang He	778-681-0610	yuyanghe97@gmail.com
Larry Cheung	778-798-7588	hiulok.cheung@alumni.ubc.ca

Appendix A: File Structure Overview

In this section, we will guide you through on how to use the zipped folder we have included in the list of deliverables. We found many resources useful and believe you may too. Once you go into the folder 'files' that we have included in our deliverables, you will see 4 folders (see Figure 21).



Figure 21. Repository Folders

In components, there are 8 folders (see Figure 22). This repository was downloaded from our google drive. In this folder, there are manufacturer manuals, schematics, and debugging notes that we used during our capstone project. For example, in the AWS server folder, you will find debugging notes we made through hours of debugging the given AWS code for the cellular development board. It took us 40+ hours to find a memory bug by Amazon that allocated the size of a pointer when it should have allocated memory for the structure that the pointer points to. We hope our debugging notes will aid your team in your development process and save you from wasting time on the same issues we've encountered.











 AWS Server	4/8/2020 7:24 PM	File folder	
 Cellular Board	4/8/2020 7:24 PM	File folder	
 E-ink Display	4/8/2020 7:24 PM	File folder	
 Enclosure	4/8/2020 7:24 PM	File folder	
 MCU	4/8/2020 7:24 PM	File folder	
 Nucleo Board	4/8/2020 7:24 PM	File folder	
 PlacePod	4/8/2020 7:24 PM	File folder	
 Solar Panel	4/8/2020 7:24 PM	File folder	
 Parts Bought	4/8/2020 7:24 PM	Microsoft Excel W...	8 KB
 Power Budget	4/8/2020 10:12 PM	Microsoft Excel W...	15 KB

Figure 22. Folders Within Components Folder

In the Github folder (see Figure 23), there are 3 zip files. Make sure you unzip them first before navigating, otherwise it would be very slow as the computer processes the unzipped files for you to use in real time.



 496firmware-master	4/8/2020 10:47 PM	Compressed (zipp...	581,168 KB
 capstonewebapp	4/8/2020 10:46 PM	Compressed (zipp...	178 KB
 dynamicparking-master	4/8/2020 10:44 PM	Compressed (zipp...	46,455 KB

Figure 23. Zipped Source Code Folders

Appendix B: Debugging and Troubleshooting Q&A

In this appendix, we include some issues that we encountered during setup for development. It took us quite a bit of time to figure them out and wish to minimize that time for your team as you set up for development.

1. **Compilation error:** “cannot find STM32_Debugger_CLI.exe”

Reason: This error occurs when the installation of STM32_Debugger_CLI.exe is installed into the wrong directory. This executable is 64-bits but on default is installed in:

C:\Program Files (x86)\STMicroelectronics\STM32Cube\STM32CubeProgrammer

but the path that scripts expect in the provided demo code is:

C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer

Solution 1: In “postbuild.sh” line 87, change the path to the actual directory of STM32_Debugger_CLI.exe, which is the path with (x86).

Solution 2: uninstall STM32CubeProgrammer and reinstall in C:\Program Files\...\...

2. “Break at address “0x8000----” with no debug information available, or outside of program code.” If you see this message - seen in Figure 24 - while in debugging mode in STM32CubeIDE, it will be impossible to debug as the first breakpoint is supposed to be in main().

Break at address "0x800c714" with no debug information available, or outside of program code.

View Disassembly...

Configure when this editor is shown Preferences...

Figure 24. Break at Address Error Message

Possible Reason: An incorrect version of cubeIDE is installed. This issue occurred when team members used version 1.1.0 but went away when version 1.2.0 was installed.

3. When installing STM32CubeIDE on macOS, you may see "STM32CubeIDE is damaged and can't be opened".

Reason: From researching around the web, we found that it appears that when macOS adds some extended attributes to the .dmg when a download is downloaded directly on the Mac.

Solution: download the Mac version on a pc, move the .dmg file to the intended Mac and run installation. No issues should occur after. To reading further, visit <https://community.st.com/s/question/0D50X0000BUh6nK/not-able-to-open-stm32cubeide-110-on-macos-mohave-10146?s1oid=00Db0000000YtG6&t=1574078933520>

4. Where is my binary (or BIN) file? I want to flash it directly to the chip without debugging after building and compiling my code.

Solution: Right-click on your project, go to Properties - you will see an image like Figure 25. Then C/C++ Build -> Settings -> Tool settings -> MCU Post-build outputs and check "Convert to binary file (-O binary).

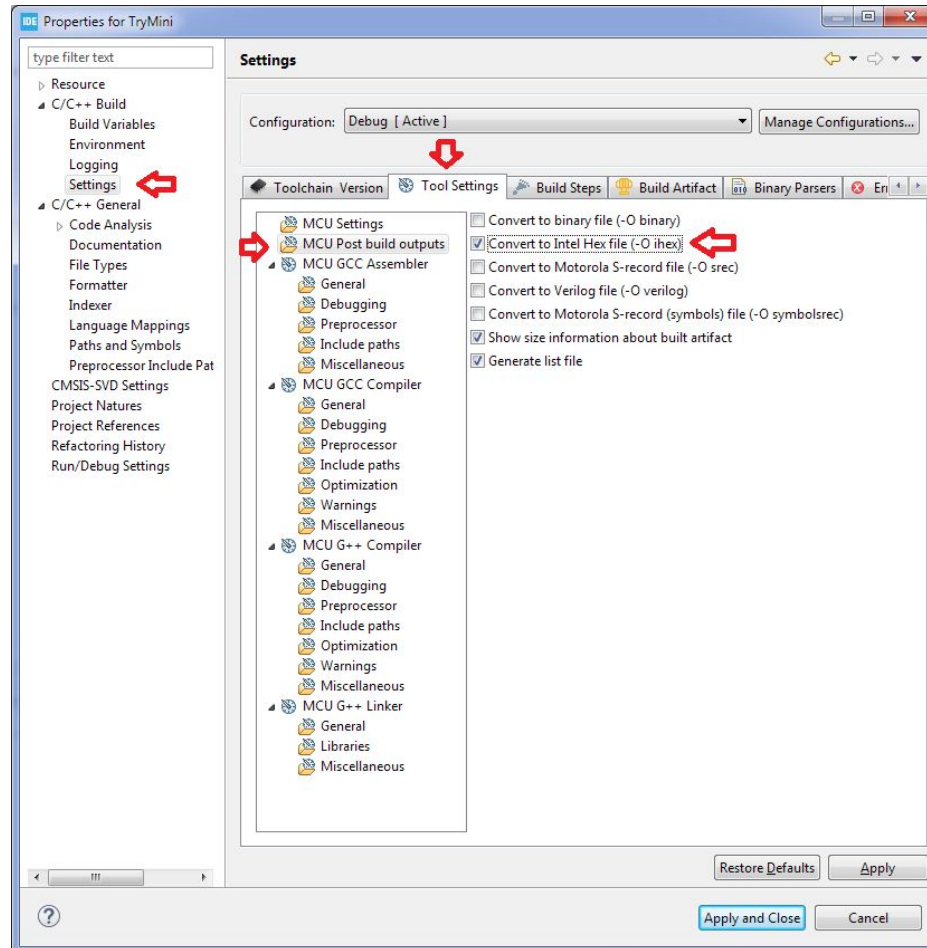


Figure 25. Properties for TryMini

- Nothing is happening with the Goodisplay e-ink display even though you compiled and loaded it on the F103 development board without any issues.

Possible Reason 1: You hooked up the cable for the display wrong.

Solution: Verify that the E-Ink display in Figure 26 is facing up. The DESPI-C1248 - seen in Figure 26 - the board is placed on top of the display. Make sure to connect the driver and display cables exactly as it is shown in Figure 27. The connector for the cables on the display should be facing upward. Connector P1 should be connected to the main FPC with the serial WFT1248BZ23 or the right cable when the white logo is at the bottom of the driver board (see Figure 26). Connector P2 should be connected to the ancillary FPC with the serial WFT1248BZ24 (see Figure 29). The serial of the FPCs can be found on the top side when the display is facing up (see Figure 30).

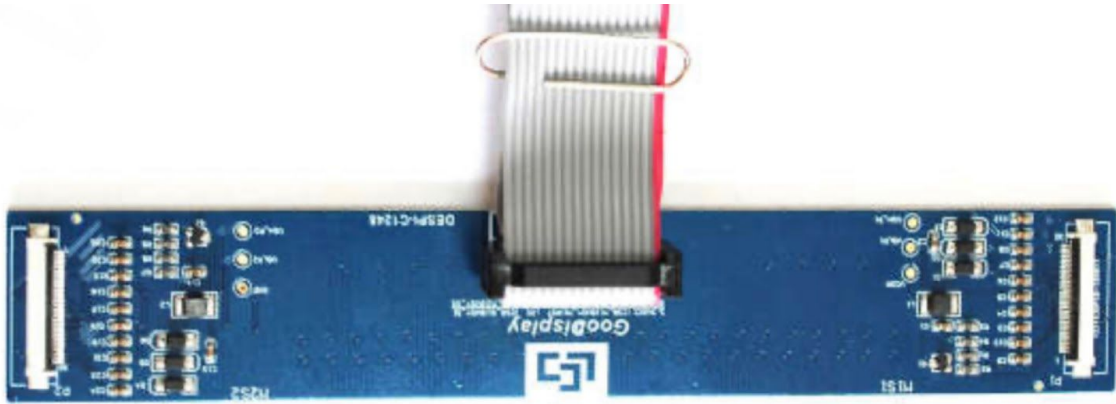


Figure 26. DESPI-C1248 Pinboard for E-Ink Display

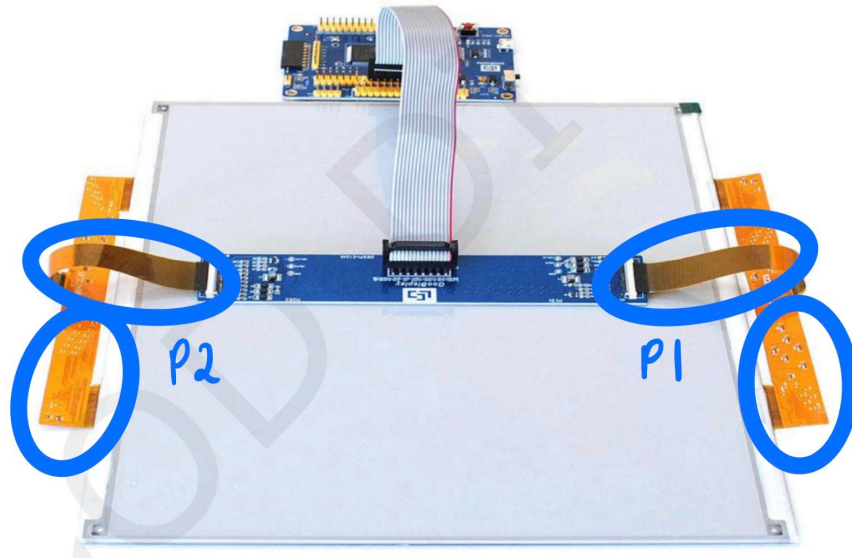


Figure 27. Display Connections with STM F103 Development Board



Figure 28. Main FPC with Connector P1



Figure 29. Ancillary FPC with Connector P2



Figure 30. Serial Location on FPC

Possible Reason 2: JTAG cable is not connected correctly.

Solution: Make sure cable is connected in the same orientation as Figure 27 with the red cable furthest from broken pins (circled in blue in Figure 31). Four pins are removed by the manufacturer on purpose and are not used.



Figure 31. Connector Locations (Blue - Pins Removed)

6. I cannot compile the project on a specific computer even though I had it working on another computer. I am sure the code is exactly the same as before.

Possible Reason: You are trying to run the AWS source code that can toggle the LED light. We encountered an obscure bug where if your directory path is too long, the project will not compile.

Solution: Delete the project from the STM32CubeIDE. Get a fresh copy of the file i.e. one you have never opened in STM32CubeIDE. Move the copy to the shortest possible directory path. We moved it to right under the C: or E: drive and compiled it and it worked.