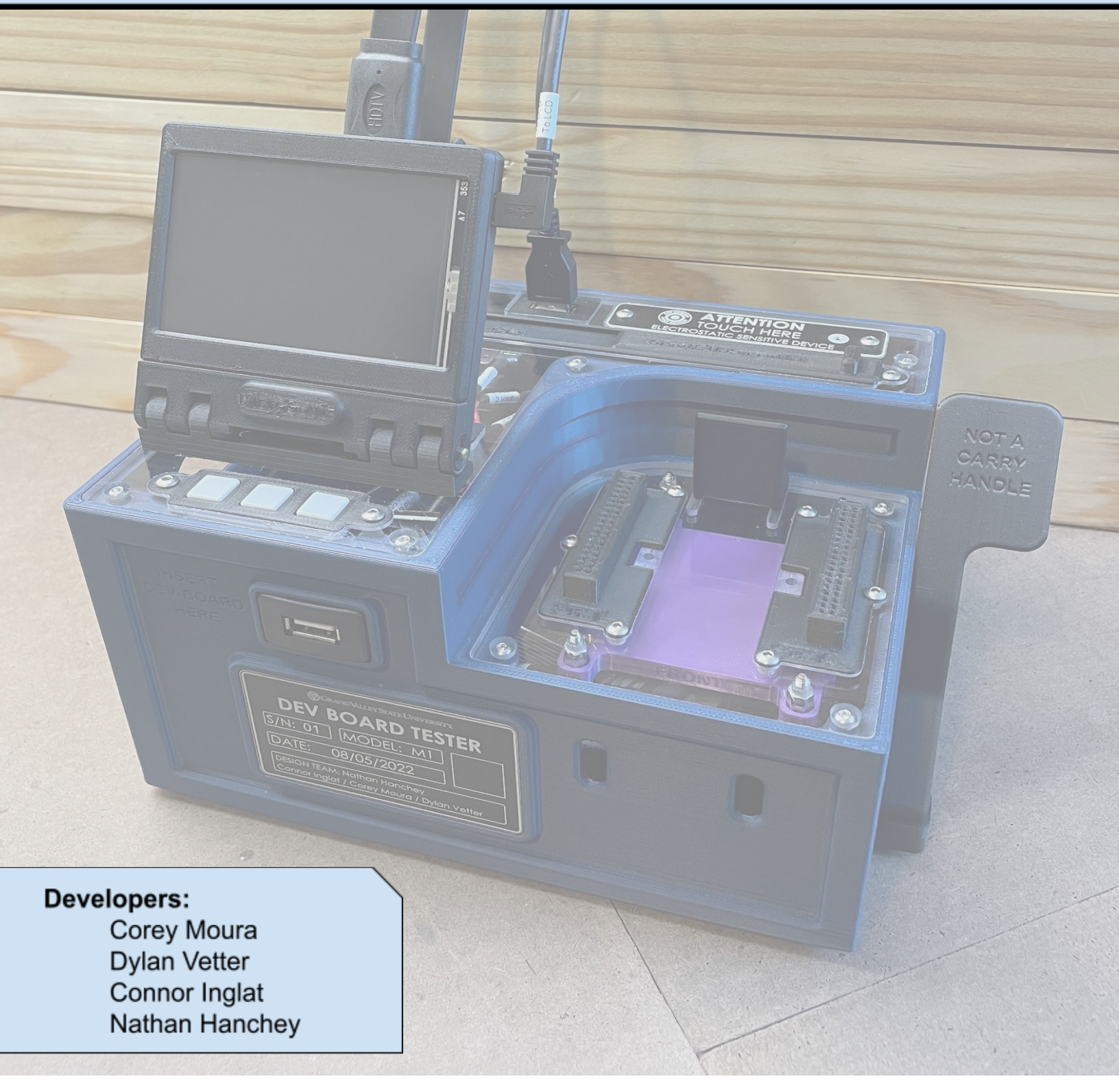
MicroDev User Manual



Developers:

Corey Moura
Dylan Vetter
Connor Inglat
Nathan Hanchey

Table of Contents

Standby Screen 4

Shutdown Screen 5

Detection Screen 6

Software Flash Screen 7

Running Test Screen 8

Test Results Screen 9

Successful Detailed Results Save to USB Drive 10

Failed Save to USB Drive 11

Remove Dev Board Screen **12**

Results Interpretation Guide **13**
- General Test Results 13
- Detailed Test Results 14
- Debugging Detailed Test Results File 14

MicroDev Setup Procedure & Guide **17**
- Raspi OS Installation 17
- Arduino CLI Installation & Application 17
- STM32 CLI Installation & Application 18

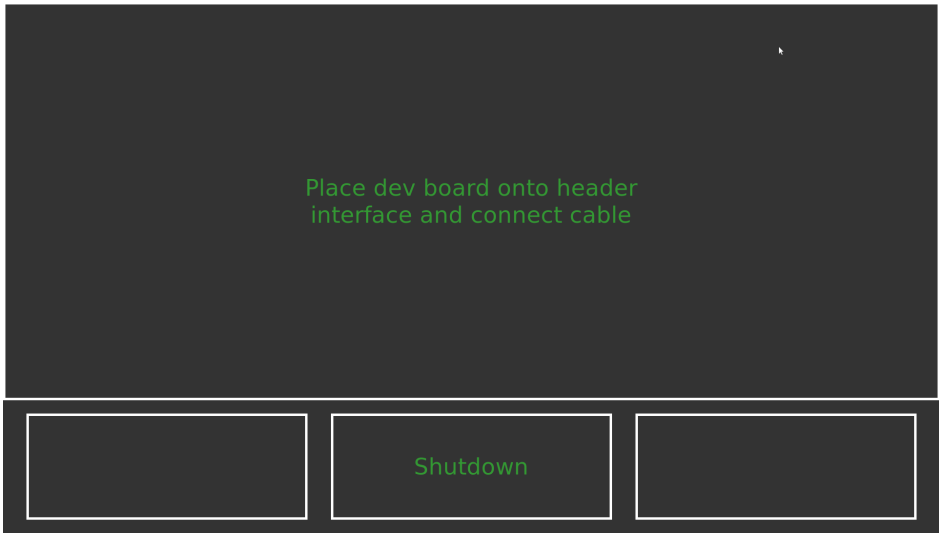
Configuration File Guide **20**

Pin & Power Mode ID to Address Mapping **24**
- STM Nucleo Pin Mapping 24
- Arduino Uno Pin Mapping 26

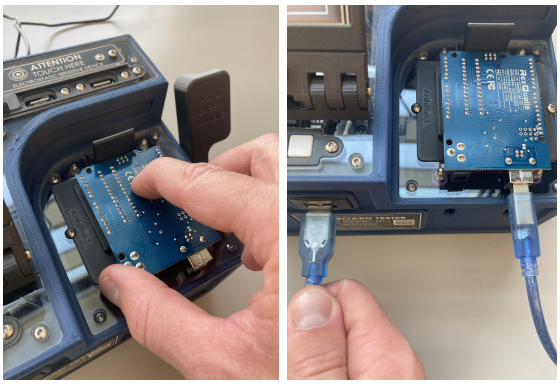
Alternative Subject Board Integration **29**
- New Development Board Footprint 29
- Command Line Interface Alterations 30
- Pin Mapping and Configurations 33
- Subject Board Embedded Code 36

Switching Subject Board Header Interface **37**

Standby Screen

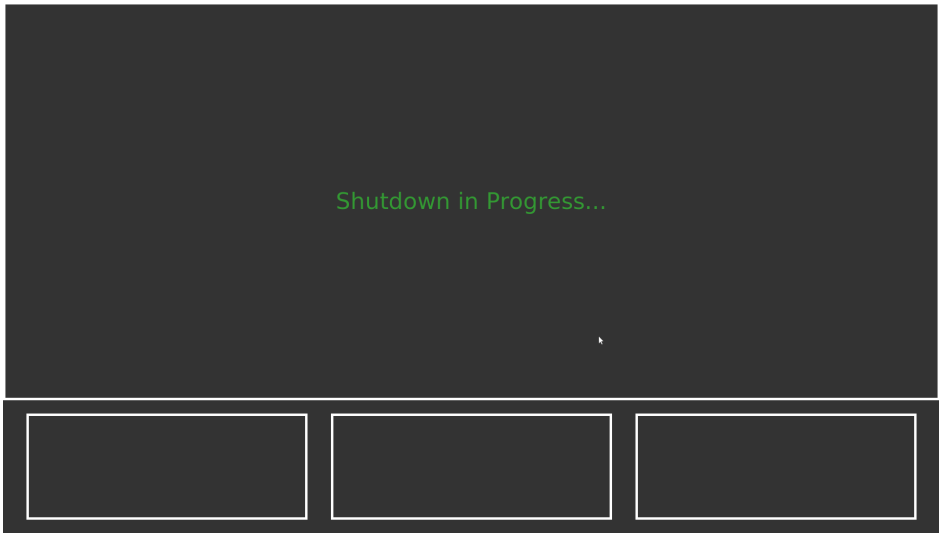


This screen is the first screen that waits for a **subject** Dev Board to be connected. The two microcontroller boards that are compatible and fully supported with the MicroDev Tester are the Arduino Uno and the STM32F446RE Nucleo. Before it continues one must make sure the subject board is firmly inserted into the headers and connected by USB.



It will auto-detect the subject board you have connected if it is supported. The screen also indicates that the middle press button is for shutdown. Be aware this is **not used as a touchscreen**, the press buttons are located beneath the screen.

Shutdown Screen

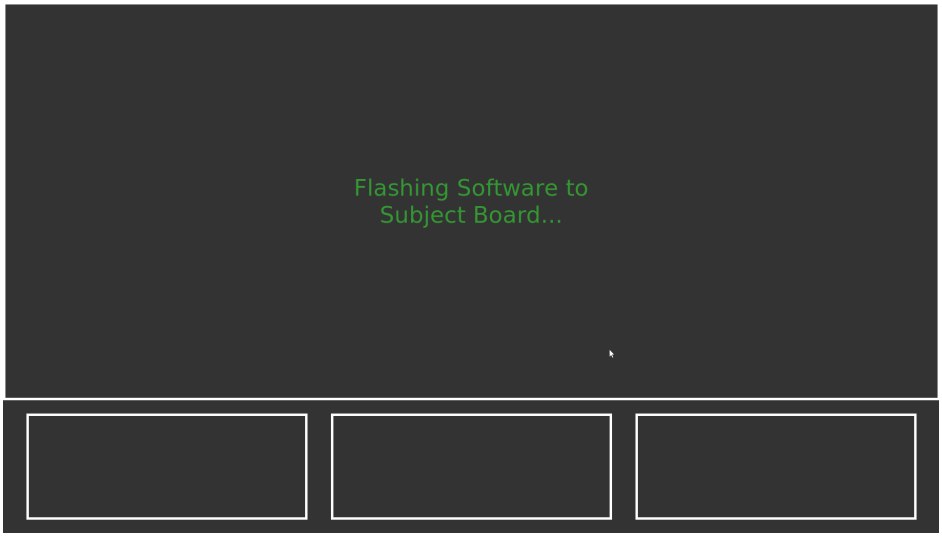


This screen indicates that a shutdown is in progress. Only after shutdown has fully completed should the power be switched off on the back of the enclosure.

Detection Screen

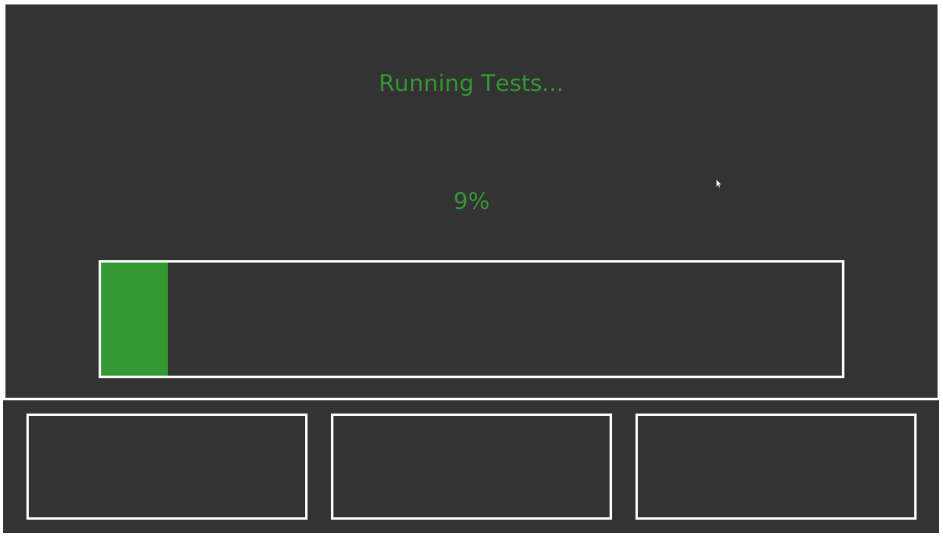
After the subject board is detected, the name of the subject board followed by ‘Detected,’ will be printed to the screen. The middle press button will then begin the tests. If the board is disconnected before the start button is pressed it will return to the Standby Screen.

Software Flash Screen



The subject board is flashed with a testing program before the tests begin. If an error occurs a message will appear indicating that the program flash was unsuccessful and then return to the Standby Screen.

Running Test Screen



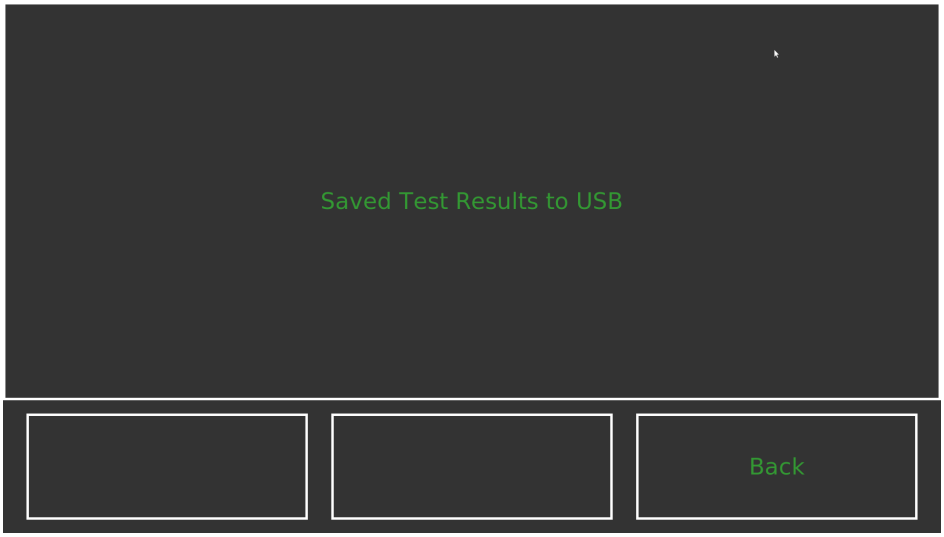
This screen shows the progress of the tests as one waits. There is a moment before the tests begin that it reads 0%. Do not be alarmed if it does not immediately begin testing. It takes about 2 seconds before the first test can begin.

Test Results Screen

This screen shows the results of the tests conducted on the board. In order to see the detailed results of each pin press the middle button that reads details. If a USB drive is connected to the MicroDev Tester, you can save an even more detailed results text file by pressing the left button. Lastly, the right button returns to the standby screen.

For a description of test results interpretation, refer to the **Results Interpretation Guide**.

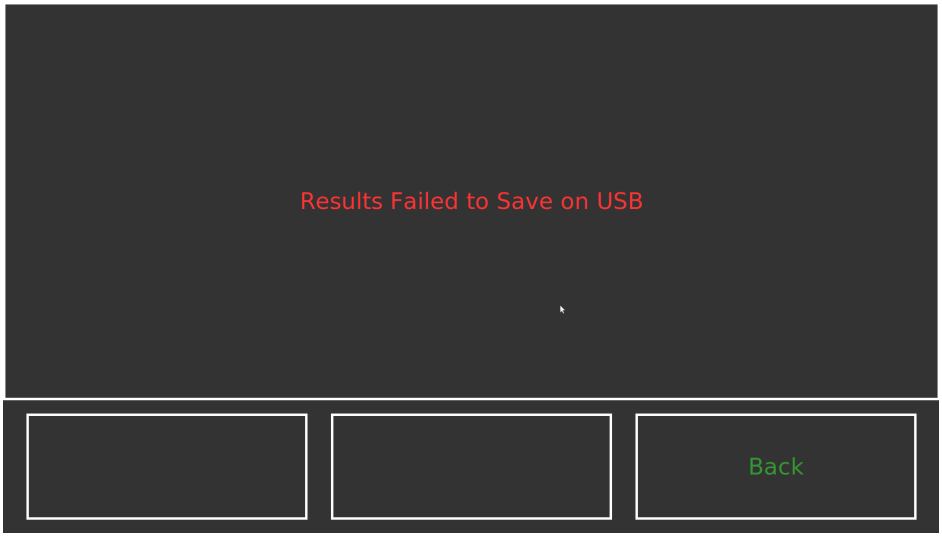
Successful Detailed Results Save to USB Drive



This screen indicates that the USB has successfully saved the detailed results file. During a successful save, 2 files are saved to your main directory on the flash drive. These files are labeled *Debug_MicroDevTest_Results.txt* and *MicroDevTest_Results.txt.*

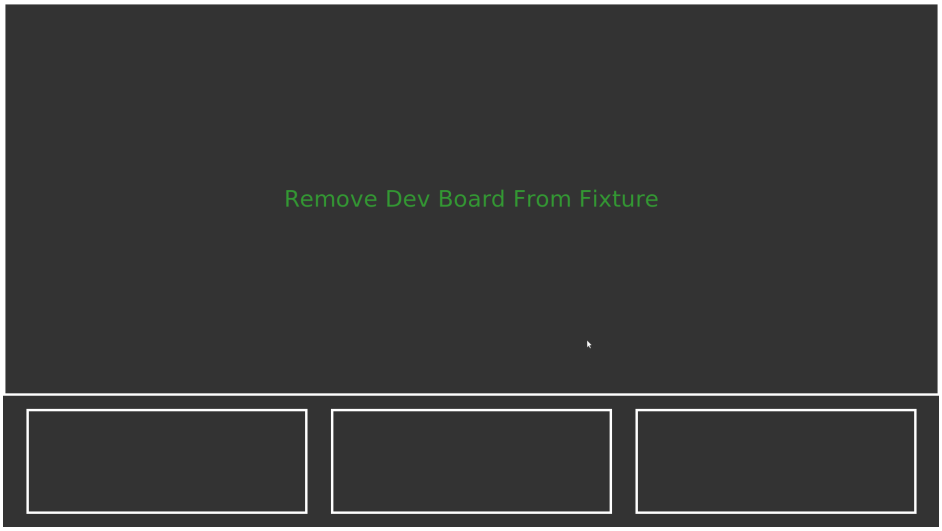
For a description of test results interpretation, refer to the **Results Interpretation Guide**.

Failed Save to USB Drive

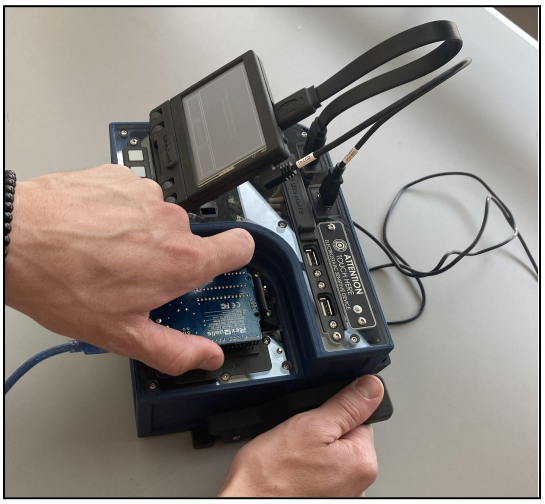


This screen indicates that the file failed to save to the USB or that a USB was not connected to the MicroDev Tester. This may be due to the USB drive not being fully mounted and may work if the save is attempted again.

Remove Dev Board Screen



This screen is the final screen and indicates that the previous subject board should be removed. First, disconnect the usb from the subject board to the MicroDev device. Then grab the subject board with your left hand and push down on the ejection knob with your right hand. Below is a demonstration of properly removing the subject board.



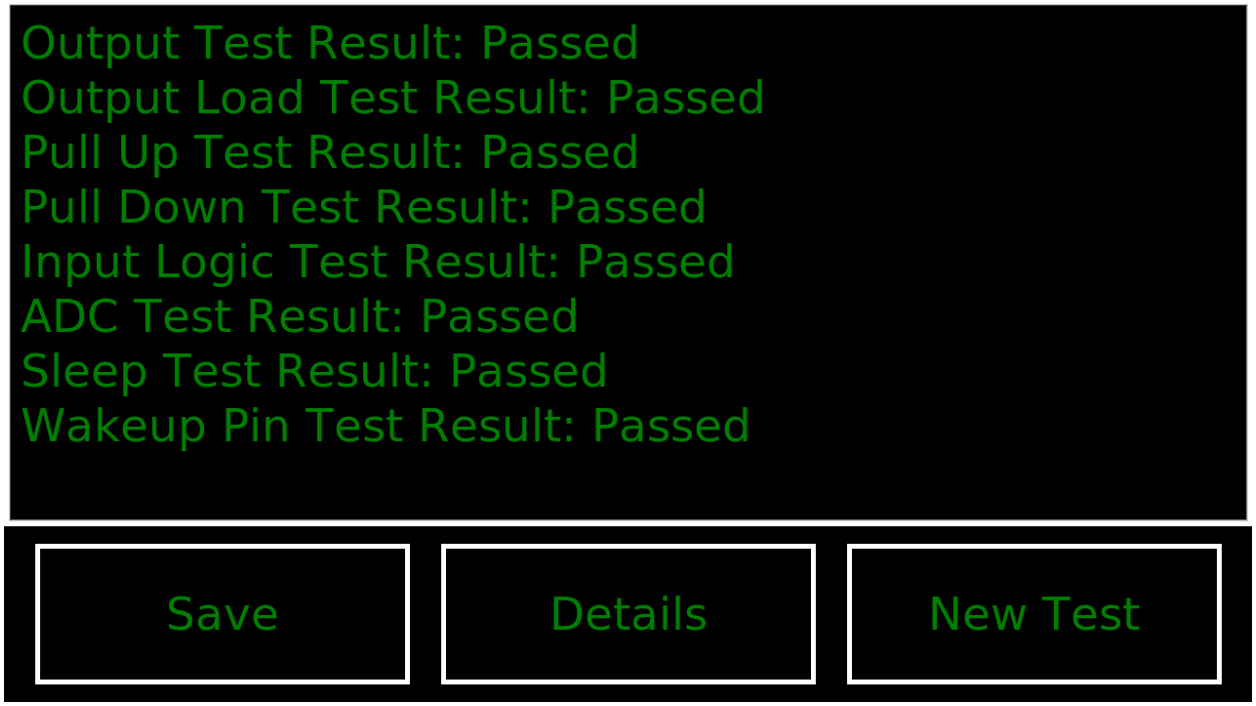
When the previous subject board is removed, the interface will return to the standby screen and wait for a new subject board. If it is not removed after 10 seconds it will return to the standby screen.

Results Interpretation Guide

Description of test results and development board diagnostics.

General Test Results

These results appear on the graphic user interface at the end of a MicroDev test cycle.

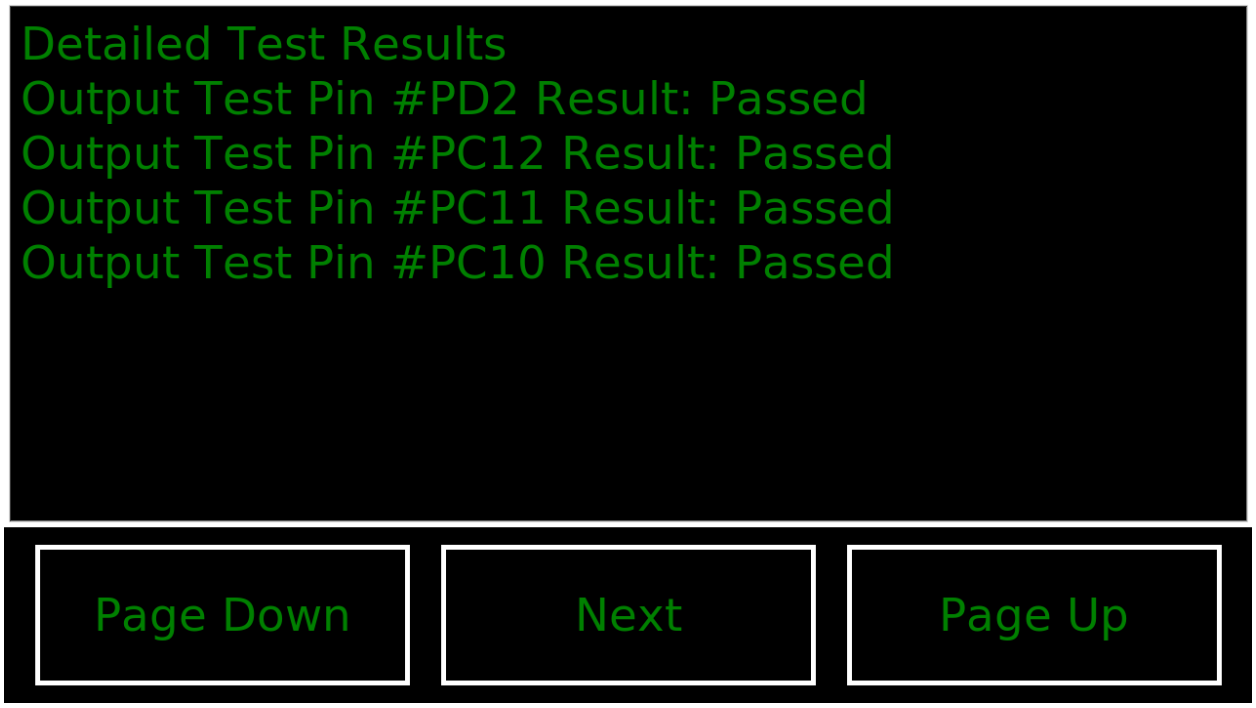


There are eight different tests that may be configured. If every iteration of that test passes then the general results show that test as successful. Although if one pin is faulty, then the whole test displays a failure. This has an advantage of making it easy to tell when the development board is working well but harder to tell where the actual problem is after a flaw is discovered.

Some diagnostics can take place when analyzing if some tests pass and others fail. If the low power modes pass then it can be deduced that it's not the case that much power is being shorted to ground by faulty pins.

Detailed Test Results

The detailed test results show the tests description, the pin name, and then an indication of whether it passes. These results are meant to isolate which pins are faulty or whether any pins were able to pass the configured threshold values.



The detailed test results also appear in a file that can be saved to an external drive labeled **MicroDevTest_Results.txt**.

If there is a faulty GPIO pin there are edge cases where the digital input test (Test #5) shows a false positive, and a passing the test when it should fail. This may occur since the input is floating during this digital input test. These tests will still not show a false negative. By comparing the input results with the input pullup or input pulldown test results it can be verified if the test was a false positive or not.

Debugging Detailed Test Results File

A more thorough set of results can be found after saving the results to a removable flash drive in a file labeled, **Debug_MicroDevTest.txt.**

These results below, for Test #1, show the voltage measurements of a high and low logic output from the subject development board without a resistive load.

```
Test 1: Output without Load
Test Logic High: 3.36 V
Test Logic Low: 0.0 V
Above results for Pin ID PD2
```

These results show the voltage measurements of a high and low logic output from the subject development board with an acceptable resistive load.

```
Test 2: Output w/ Load
Test Logic High: 3.17 V
Test Logic Low: 0.0 V
Above results for Pin ID PD2
```

These results show the input pull up resistance value and confirm that the pull up input is generating a voltage.

```
Test 3: Pull-Up Input
Test Pull Up Resistance Value: 41.0k ohms
Above results for Pin ID PD2
```

These results show the input pull down resistance value and confirm that the pull down input is sinking a voltage.

```
Test 4: Pull-Down Input
Test Pull Down Resistance Value: 29.0k ohms
Above results for Pin ID PD2
```

These results show the state of a digital input to the subject board, 1 indicates a high voltage and 0 a low voltage. The *Logic High* value represents what the subject board returns when the digital signal to a given GPIO is high. The *Logic Low* value represents what the subject board returns when the digital signal to a given GPIO is low.

```
Test 5: Input Logic
Logic High: 1
Logic Low: 0
Above results for Pin ID PD2
```

These results show the voltage measurements of the subject development board after 3.3V, 1.5V and 0V are given to its ADC inputs.

```
Test 6: ADC Check
ADC Return Value: 1: 3.2
ADC Return Value: 2: 1.5
ADC Return Value: 3: 0.0
Above results for Pin ID PC4
```

These results show the before and after current measurements from the entered sleep mode.

```
Test 7: Set Power Mode
Current in mA Before Power Mode: 70.17
Current in mA After Power Mode: 60.59
Above results for Pin ID PD2
```

These results show the before and after current measurements from waking up after the previously entered sleep mode.

```
Test 8: Wakeup From Sleep
Current in mA Before Wakeup: 60.0
Current in mA After Wakeup: 69.8
Above results for Pin ID PD2
```

MicroDev Setup Procedure & Guide

Raspi OS Installation

1. Download Raspbian OS Imager (https://www.raspberrypi.com/software/)
2. Insert and locate SD card on computer
3. Install Recommended Raspberry Pi OS (w/ Software) to SD card.
4. Boot-up Raspi
5. Username: microdev | Password: microdev
6. Connect to Internet
7. Terminal: git clone https://github.com/hancheyn/MicroDev.git
8. Terminal: chmod 777 MicroDev/setup.sh
9. Terminal: ./MicroDev/setup.sh
10. File Manager > Edit > Preferences > Volume Management >
    [uncheck] Show available options for removable media when they are inserted
11. Raspberry Pi Configuration > [disable] Screen Blanking
12. Raspberry Pi Configuration > Interfaces > [enable] I2C

Arduino CLI Installation & Application

Start up raspberry pi using an hdmi monitor, keyboard, and mouse.

Download in Add/Remove Software:

1. Raspberry OS Start Menu > Preferences > Add/Remove Software
2. Search “arduino”
3. Check AVR development board IDE
4. Check Command Line Tool for compiling Arduino Sketches
5. Click Apply and Wait for Download

Download in Terminal:

1. Terminal: curl -fsSL
   https://raw.githubusercontent.com/arduino/arduino-cli/master/install.sh | sh
2. Put arduino-cli into a runnable folder so the command will run on its own ($PATH).
3. Terminal Command: arduino-cli core install arduino:avr
4. Terminal Command View Connected Boards: arduino-cli board list
5. Terminal Command Compile .ino example: arduino-cli compile -b arduino:avr:uno Blink.ino -e
6. Terminal Command Upload .ino example: arduino-cli upload -b arduino:avr:uno -p /dev/ttyACM0 Blink.ino

FLASH TO ARDUINO NOTES:
The Arduino CLI does not automatically create a permanent bit file for uploading. In order to do this, one must add the command '-e' while compiling. This looks like the following: *arduino-cli compile -b arduino:avr:uno Blink.ino -e*.This will create a new folder within the arduino project folder that contains a bit file, "[Project name].ino.with_bootloader.bin". This should be in a folder with a file path *./build/arduino.avr.uno/*. In Terminal, the command *arduino-cli upload -b arduino:avr:uno -p /dev/ttyACM0 -i [binary file path]* will upload the file.

STM32 CLI Installation & Application

Start up raspberry pi using an hdmi monitor, keyboard, and mouse.

Download in Add/Remove Software:

1. Raspberry OS Start Menu > Preferences > Add/Remove Software
2. Search "stlink"
3. Check OpenSource ST-Link tools replacement… (x3)
    a. …libstlink1-1.6.1+ds-3
    b. …stlink-gui-1.6.1+ds-3
    c. …stlink-tools-1.6.1+ds-3
4. Click Apply and Wait for Download

FLASH TO STM32 NOTES:
The STMCubeIDE does not automatically create a permanent bit file for uploading. Since it uses a .elf file (located in the debug folder) this must first be converted to a .bin file before flashing can occur. The .elf file can be converted using the following command from terminal *arm-none-eabi-objcopy -O binary F401RE_T.elf main.bin.* In the previous example, F401RE_T.elf becomes a binary file called main.bin. Flashing can then occur using the command *st-flash write main.bin 0x08000000*.

I2C Setup and Configuration

Start up raspberry pi using an hdmi monitor, keyboard, and mouse.

Configure I2C:

1. Raspberry OS Start Menu > Preferences >Raspberry Pi Configuration
2. Click on the Interfaces tab.
3. Switch On I2C and click OK
4. Restart Raspberry Pi
5. Command: sudo i2cdetect -y 1
6. Check that all i2c device addresses are shown while connected to Bigfoot PCB.

Read-Only Configuration

Follow the procedure in the link below to configure a read only file structure for important directories. This configuration also may slow down the execution of testing.
https://learn.adafruit.com/read-only-raspberry-pi

References & Helpful Links:
https://www.raspberrypi.com/software/
https://www.arduino.cc/pro/cli
https://arduino.stackexchange.com/questions/54098/arduino-ide-permision-denied-for-upload-ubuntu
https://www.instructables.com/Start-Developing-STM32-on-Linux/
https://blog.podkalicki.com/how-to-compile-and-burn-the-code-to-stm32-chip-on-linux-ubuntu/
https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial/all
https://raspberry-projects.com/pi/programming-in-python/i2c-programming-in-python/using-the-i2c-interface-2

Configuration File Guide

The configurations are located in specific files and use the pin mapping to identify which pin and multiplexer are used during a given test. The pin map connects a pin identification number, that is used in serial communication, to the multiplexer enable signal and address signals that relate to that pin. The littlefoot PCB uses those enable signals and address signals to isolate a pin to be tested. If this pin mapping is incorrect it can be difficult to debug. The enable signals control the enable pins on each of the eight multiplexers. Only one is enabled at a time. The address signals are a 0 through 7 binary address sent via output signals to control the enabled multiplexer. The disabled multiplexer will not react to the signals even though they are all on the same bus. All code dealing with the configurations is within the Controller.py module.

Use the STM32 Nucleo Pin Map in Figure 1 as a guide to writing a new pin map.

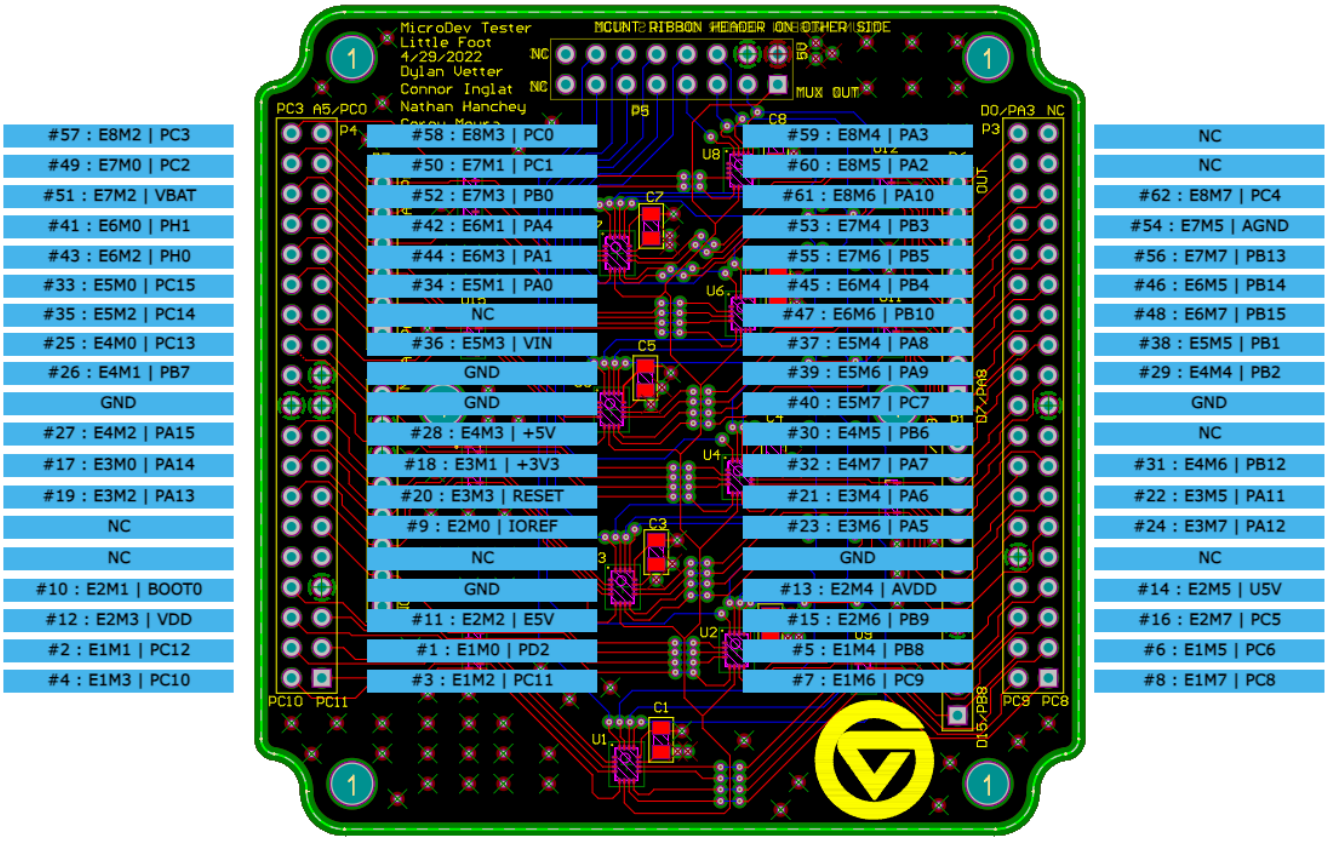


Figure 1: STM Nucleo Pin Mapping

The threshold configuration file for a given subject board is named with the following convention, *[subject]Threshold.config*. The preconfigured STM32 Nucleo file is named

stm32f4Threshold.config and the Arduino Uno is named *unoThreshold.config*. The threshold file is setup as follows:

```
1   TestID,ThresholdValue1,ThresholdValue2,...
2   1,3.5,0.5
3   2,3.5,0.5
4   3,20000,50000,4.0
5   4,20000,50000,0.5
6   5
7   6,5,3.3,1.5,1,0
8   7,6
9   8,6
10  M,4,2.5
```

Code Snippet 1

After the first line, each following line is the test thresholds that will be used as the pass or fail conditions of the test. The first line describes the threshold configuration format, an improperly formatted threshold file will end the testing process. Every line shown above must be in the threshold file just as shown in Code Snippet 1. The second line represents Test #1 and each following line must be the next consecutive Test ID. M, or miscellaneous, goes on line 10. The thresholds values are as follows:

Test #1 | Min High Voltage | Max Low Voltage
Test #2 | Min High Voltage | Max Low Voltage
Test #3 | Min Resistance | Max Resistance | Pull-Up Min Voltage
Test #4 | Min Resistance | Max Resistance | Pull_Down Max Voltage
Test #5 | N/A
Test #6 | Voltages Generated By DAC (user can add more test voltages)
Test #7 | Current Drop mA (initial - final)
Test #8 | Current Drop mA (final - initial)
Misc. | 5V Pin Minimum Voltage | 3V3 Pin Minimum Voltage

The test configuration file for a given subject board is named with the following convention, *[subject]Test.config*. The preconfigured STM32 Nucleo file is named *stm32f4Test.config* and the Arduino Uno is named *unoTest.config*. The test file is setup as follows:

```
1   TestID,PinID,MuxAddress,MuxEnable
2   1,21,4,3
3   1,23,6,3
4   1,30,5,4
5   1,32,7,4
6   1,37,4,5
7   1,39,6,5
8   1,40,7,5
9   1,45,4,6
10  1,47,6,6
11  1,53,4,7
12  1,55,6,7
```

Code Snippet 2

After the first line, each following line is a test that will be conducted. The first line describes the test configuration format, shown in Code Snippet 2. An improperly formatted test file will end the test before finishing. The Test ID and Pin ID that is recognized by the subject development board goes first. Then, the Multiplexer Address and Enable pin comes next, all separated by commas. It is important to remember the pin ID must go with the *address* and *enable value* for that pin. The exception is with tests #7 and #8. Test #7 does not need the address and enable value and Test #8 has a preconfigured wakeup pin depending on the subject board.

Refer back to **Figure 1** to connect a GPIO pin name to the Pin ID, Multiplexer Address and Multiplexer Enable.

Test #1 |
This test checks that the voltages produced by the subject development board reach a high and low logic level as specified in the configured range of the threshold configurations.

Test #2 |
This test confirms that voltages under a resistive load can reach a high and low logic level as specified in the configured range of the threshold configurations.

Test #3 |
This test confirms the operation of a pullup input and checks if the pull up resistance is within the configured range.

Test #4 |
When applicable, this test confirms the operation of a pulldown input and checks if the pull down resistance is within the configured range.

Test #5 |
This test confirms that the subject board's GPIO pins input can read high and low voltages. Auto-detects board and configures logic levels and does not need configured thresholds.

Test #6 |
This test confirms that the subject board's ADC pins input can read voltages as specified in the configuration file.

Test #7 |
This test puts the board to sleep and checks that the difference in current is greater than the configured threshold in mA. In the current iteration, the wakeup pins are constrained to D2 or D3 on the Arduino board and PA0 on the STM Nucleo.

Test #8 |
Wakes up from power mode using the wakeup pin. In the current iteration, the wakeup pins are constrained to D2 or D3 on the Arduino board and PA0 on the STM Nucleo. The test also checks that the difference in current is greater than the configured threshold in mA.

Test #9 |
Resets the Subject Board Reset Pin

Pin & Power Mode ID to Address Mapping

STM Nucleo Pin Mapping

Below is the mapping of the Littlefoot PCB Multiplexer enable and address to the corresponding pin ID number, seen in Figure 1.

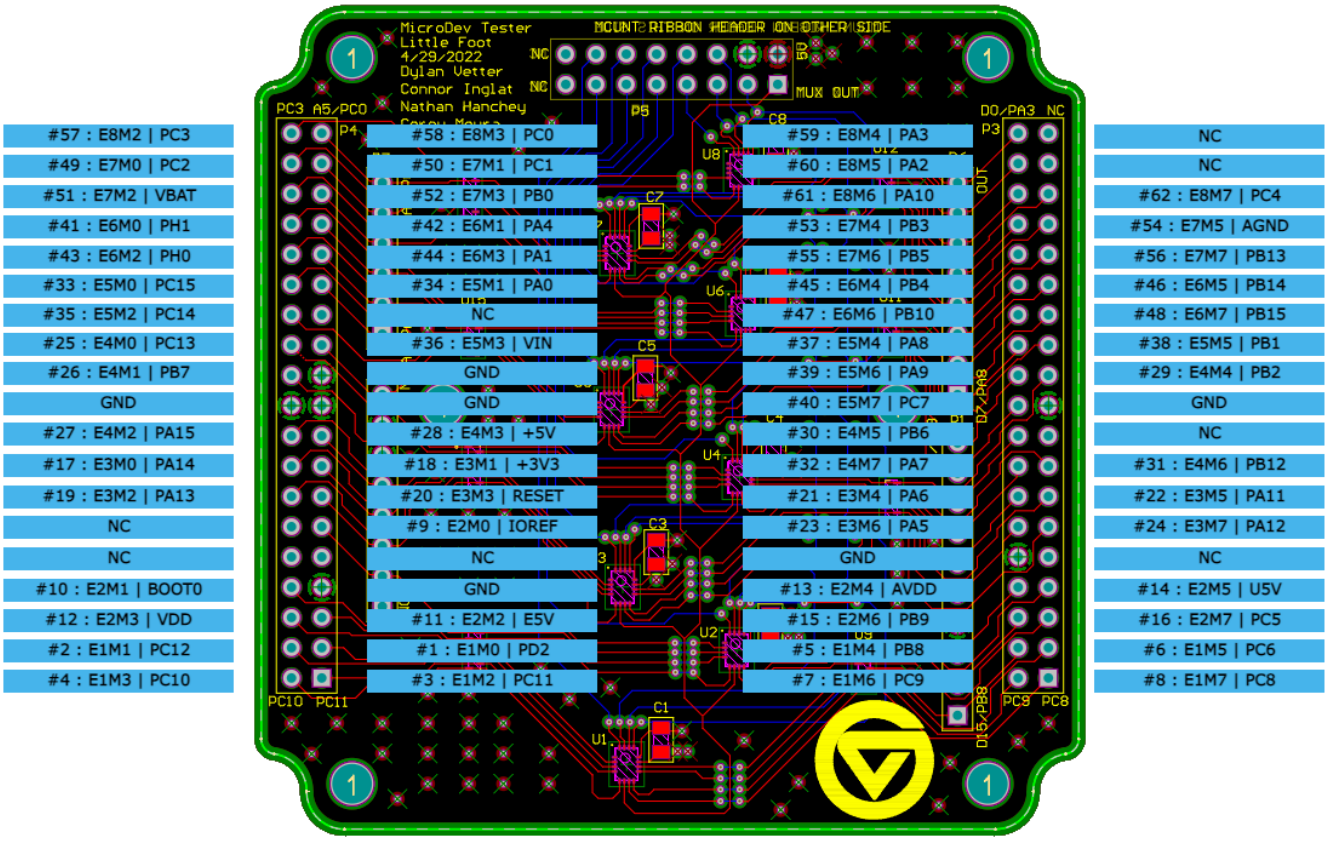


Figure 1: STM32 Nucleo Pin ID Mapping

STM32 GPIO (General Purpose Input Output) Pins

Table 1: STM GPIO Pin Mapping

Pin ID	Mux Address	Enable	Pin Name
1	0	1	PD2
2	1	1	PC12
3	2	1	PC11
4	3	1	PC10
5	4	1	PB8
6	5	1	PC6
7	6	1	PC9
8	7	1	PC8
15	6	2	PB9
16	7	2	PC5
17	0	3	PA14
19	2	3	PA13
21	4	3	PA6
22	5	3	PA11
23	6	3	PA5
24	7	3	PA12
25	0	4	PC13
26	1	4	PB7
27	2	4	PA15
29	4	4	PB2
30	5	4	PB6
31	6	4	PB12
32	7	4	PA7
33	0	5	PC15
34	1	5	PA0
35	2	5	PC14
37	4	5	PA8
38	5	5	PB1

39	6	5	PA9
40	7	5	PC7
41	0	6	PH1
42	1	6	PA4
43	2	6	PH0
44	3	6	PA1
45	4	6	PB4
46	5	6	PB14
47	6	6	PB10
48	7	6	PB15
49	0	7	PC2
50	1	7	PC1
52	3	7	PB0
53	4	7	PB3
55	6	7	PB5
56	7	7	PB13
57	2	8	PC3
58	3	8	PC0
59	4	8	PA3
60	5	8	PA2
61	6	8	PA10
62	7	8	PC4

STM32 Analog Input Pins

Table 2: STM ADC Pin Mapping

Pin ID	Mux Address	Enable	Pin Name
16	7	2	PC5
62	7	8	PC4
38	5	5	PB1
49	0	7	PC2
57	2	8	PC3
34	1	5	PA0
44	3	6	PA1
42	1	6	PA4
52	3	7	PB0
50	1	7	PC1
58	3	8	PC0

STM32 Power Mode IDs

Table 3: STM Power Modes

Power Mode ID	Power Mode Type
1	**Sleep Mode**
2	**Stop Mode**
3	**Standby Mode**

STM32 Wake Up Pins

Pin PA0 (or potentially PA14 if reconfigured)

Arduino Uno Pin Mapping

Below is the mapping of the Littlefoot PCB Multiplexer enable and address to the corresponding pin ID number, seen in Figure 2:

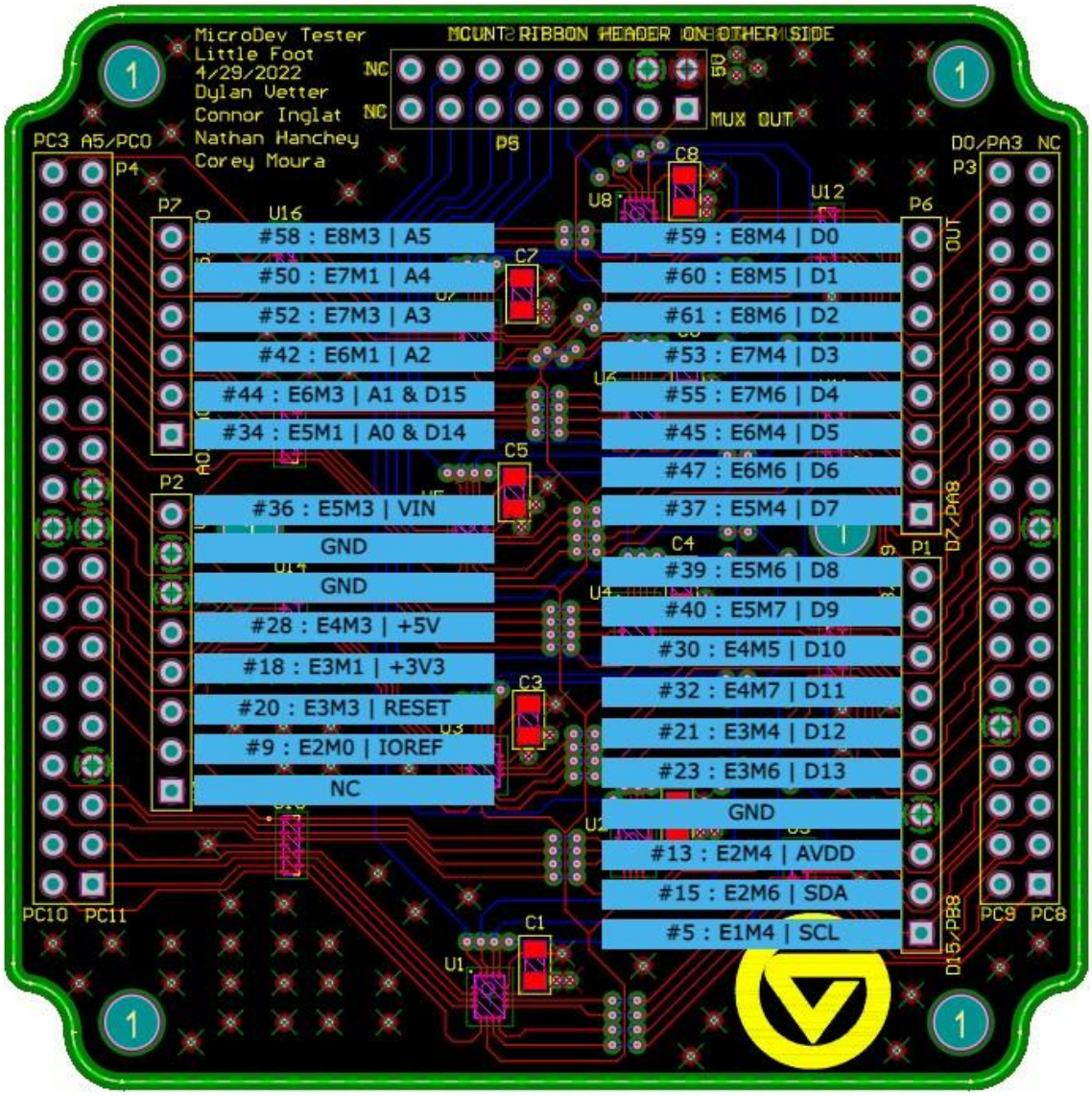


Figure 2: Arduino Uno Pin Mapping

Arduino Uno GPIO (General Purpose Input Output) Pins

Table 4: Arduino GPIO Pin Mapping

Pin ID	Mux Address	Enable	Pin Name
5	4	1	A5/SDA
15	6	2	A4/SCL
21	4	3	D12
23	6	3	D13
30	5	4	D10
32	7	4	D11
37	4	5	D7
39	6	5	D8
40	7	5	D9
45	4	6	D5
47	6	6	D6
53	4	7	D3
55	6	7	D4
59	4	8	D0
60	5	8	D1
61	6	8	D2
34	1	5	A0
44	3	6	A1
42	1	6	A2
52	3	7	A3
50	1	7	A4
58	3	8	A5

Arduino Uno Analog Input Pins

Table 5: Arduino Uno ADC Pin Mapping

Pin ID	Mux Address	Enable	Pin Name
34	1	5	A0
44	3	6	A1
42	1	6	A2
52	3	7	A3
50	1	7	A4
58	3	8	A5

Arduino Uno Power Modes ID

Table 6: Arduino Uno Power Modes

Power Mode ID	Power Mode Type
1	**Sleep Mode Idle**
2	**Sleep Mode ADC**
3	**Sleep Mode Power Save**
4	**Sleep Mode EXT Standby**
5	**Sleep Mode Standby**
6	**Sleep Mode Power Down**

Arduino Uno Wake Up Pins

Pin D2 or D3

Alternative Subject Board Integration

A top-down approach to repurposing the MicroDev code base and hardware for an unsupported subject board. WARNING: Adding or altering code may affect functionality of the MicroDev device as a whole.

New Development Board Footprint

The MicroDev system can be adapted for a development board with less than 62 pins to be under test. In order to connect a new board, it must be made to connect through a usb port for serial communication and an adapter must be created to connect the Littlefoot to the development board footprint.

A simple way to do that would be to use an Arduino Prototyping Board as seen below, and soldering on headers to connect to the alternative development boards design:

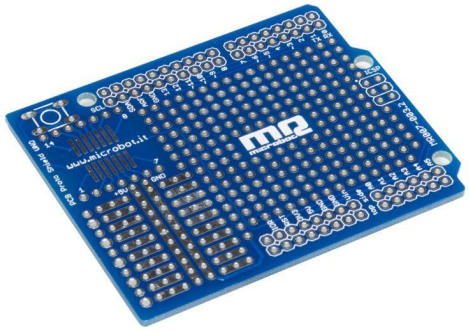


Figure 1: Arduino Prototyping Board

Or, a generic prototyping board to fit over the STM connection to the Littlefoot PCB on the MicroDev device.

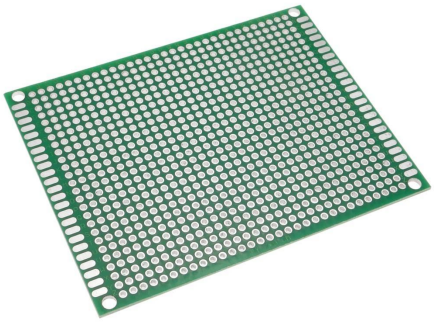


Figure 2: Generic PCB

3.3 Volt and 5 Volt Output Pin ID’s

The 3.3 Volt and 5 Volt output pins are configured to a specific multiplexer address. That means that the new footprint should connect the 3V3 pin and 5V pin of the new development board to the same place as the STM Nucleo and Arduino Uno. This could also be altered in the software by changing the functions that are contained in the Model.py module.

Command Line Interface Alterations

If the board is an alternate STM Nucleo with the same footprint then only step the Board Identification and upload must be added in connection to the same configuration files of the other STM Nucleos. All code for the command line interface is within the Model.py module.

Board Identification and Upload

A new subject board will not be properly identified unless a command line interface compatible with the new development board is installed and configured. This command line interface must be able to identify the subject board through a unique ID, compile the embedded code, and also flash to the development board.

The identification of the connected development board is checked in a loop calling the board_list function. This function is in the Model.py module. The command line interface instruction receives an array of strings back from the subprocess call in Python. Below in Code Snippet 1, is an example of how the information is parsed to identify an Arduino Uno is connected.

```
659  # ------------------------------------------------------------------------
660  # Description: Uses CLI to read for subject board connections
661  # Accepts: NA
662  # Returns: [string] board type
663  # ------------------------------------------------------------------------
664  def board_list():
665
666      # 1) ARDUINO DETECTION
667      res_ardiuno = subprocess.getstatusoutput(f'arduino-cli board list')
668
669      # PARSE DATA FOR ARDUINO
670      ARD = res_ardiuno[1].split("\n")
671      boards = len(ARD)
672
673      # No Board Detection
674      if boards < 2:
675          return "No Boards Detected"
676
677      elif boards == 2:
678          print("Internet Disconnected")
679
680      # Arduino Uno Detection
681      elif boards == 3 or boards == 4:
682          board_type = ARD[1].split(" ")
683          # print(board_type[4])
684          if board_type[4] == "Serial":
685              # print(board_type[8])
686              if board_type[8] == "Uno":
687                  return "Arduino Uno Detected"
688          elif board_type[4] == "Unknown":
689              print(board_type)
690
691      else:
692          return "Overflow"
693
694
```

Code Snippet 1

The conditions for identifying the new subject board should be added after the other boards' conditionals. If nothing else is connected it will pass through those conditions and reach the connected subject board's CLI call. That should be inserted in the lines shown below in Code Snippet 2. The string that is returned after a successful detection has a place in the Controller module that is described in the pin mapping section.

```
695     # 2) STM DETECTION
696     res_stm = subprocess.getstatusoutput(f'st-info --probe')
697     # PARSE DATA FOR STM'S
698     # print(res_stm[1])
699     if len(res_stm) > 1:
700         STM = res_stm[1].split("\n")
701         boards1 = len(STM)
702     else:
703         boards1 = 0
704 
705     # STM UNIQUE IDs
706     # 0x0433: STM32F401xD/E
707     # 0x0431: STM32F411xC/E
708     # 0x0421: STM32F446
709     if boards1 > 6:
710 
711         Chip_ID = STM[5].split(" ")
712         Chip_ID = list(filter(None, Chip_ID))
713 
714         Family = STM[6].split(" ")
715         Family = list(filter(None, Family))
716         Family = str(Family[1]).replace("x", "")
717 
718         print(Chip_ID)
719         print(Family)
720         if Chip_ID[1] == "0x0431":
721             return "STM32F411 Detected"
722 
723         elif Chip_ID[1] == "0x0433":
724             return "STM32F401 Detected"
725 
726         elif Family == "F446" and Chip_ID[1] == "0x0421":
727             return "STM32F446 Detected"
728 
729     #  3) ADD NEW SUBJECT BOARD HERE
730     """
731     The Process for adding a new subject board in software...
732 
733     """
734 
735     # Use class globals for board file path and id info
736     return "No Boards Detected"
```

Code Snippet 2

Pin Mapping and Configurations

Understanding the Pin Map

The pin map connects a pin identification number, that is used in serial communication, to the multiplexer enable signal and address signals that relate to that pin. The littlefoot PCB uses those enable signals and address signals to isolate a pin to be tested. If this pin mapping is incorrect it can be difficult to debug. The enable signals control the enable pins on each of the eight multiplexers. Only one is enabled at a time. The address signals are a 0 through 7 binary address sent via output signals to control the enabled multiplexer. The disabled multiplexer will not react to the signals even though they are all on the same bus. All code dealing with the configurations is within the Controller.py module.

Use the STM32 Nucleo Pin Map in Figure 3 as a guide to writing a new pin map.

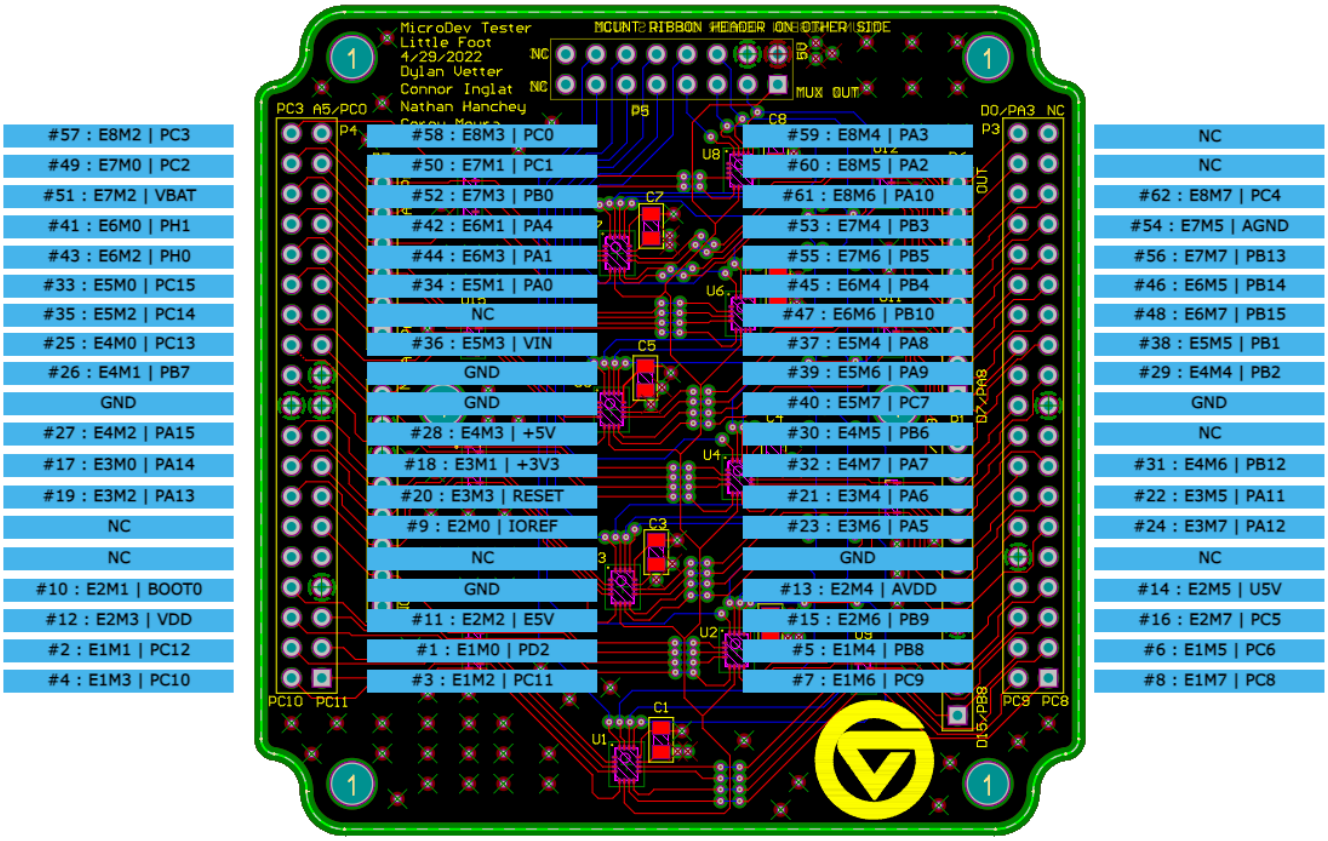


Figure 3: STM Nucleo Pin Mapping

The Pin IDs must be consistent between the test configuration file and the embedded C code receiving the pin ID value.

```
"""
Arduino Uno Mapping Key
"""
arduino_pinmap = {1: 'None', 2: 'None', 3: 'None', 4: 'None', 5: 'SCL',
                  6: 'None', 7: 'None', 8: 'None', 9: 'IOREF', 10: 'None', 11: 'None',
                  12: 'VDD', 13: 'AVDD', 14: 'None', 15: 'SDA', 16: 'None', 17: 'None',
                  18: '3V3', 19: 'None', 20: 'RESET', 21: 'D12', 22: 'None',
                  23: 'D13', 24: 'None', 25: 'None', 26: 'None', 27: 'None', 28: '5V',
                  29: 'None', 30: 'D10', 31: 'None', 32: 'D11', 33: 'None', 34: 'A0',
                  35: 'None', 36: 'VIN', 37: 'D7', 38: 'None', 39: 'D8', 40: 'D9',
                  41: 'None', 42: 'A2', 43: 'None', 44: 'A1', 45: 'D5', 46: 'None',
                  47: 'D6', 48: 'None', 49: 'None', 50: 'A4', 51: 'VBAT',  52: 'A3',
                  53: 'D3', 54: 'None', 55: 'D4', 56: 'None', 57: 'None', 58: 'A5',
                  59: 'D0', 60: 'D1', 61: 'D2', 62: 'None'}

"""
New Subject Config.
Add a Subject Mapping Key Below
"""
configurable_board = "Name Here"
configurable_filename = "fileTest.config"
configurableThreshold_filename = "fileThreshold.config"
configurable_logic = 3.3
configurable_pinmap = {1: 'None', 2: 'None', 3: 'None', 4: 'None', 5: 'None',
               6: 'None', 7: 'None', 8: 'None', 9: 'None', 10: 'None', 11: 'None',
               12: 'None', 13: 'None', 14: 'None', 15: 'None', 16: 'None', 17: 'None',
               18: '3V3', 19: 'None', 20: 'None', 21: 'None', 22: 'None',
               23: 'None', 24: 'None', 25: 'None', 26: 'None', 27: 'None', 28: '5V',
               29: 'None', 30: 'None', 31: 'None', 32: 'None', 33: 'None', 34: 'None',
               35: 'None', 36: 'None', 37: 'None', 38: 'None', 39: 'None', 40: 'None',
               41: 'None', 42: 'None', 43: 'None', 44: 'None', 45: 'None', 46: 'None',
               47: 'None', 48: 'None', 49: 'None', 50: 'None', 51: 'None',  52: 'None',
               53: 'None', 54: 'None', 55: 'None', 56: 'None', 57: 'None', 58: 'None',
               59: 'None', 60: 'None', 61: 'None', 62: 'None'}
```

Code Snippet 3

The string *configurable_board,* seen in Code Snippet 3, **must be the same string** that is returned from the board_list function added in Model.py. The high logic level of the new subject development board is to be written as the *configurable_logic*. The string for the threshold filename, *configurableThreshold_filename*, and test filenames, *configurable_filename*, must exist in the project Python folder.

The threshold configuration file for a given subject board is named with the following convention, *[subject]Threshold.config*. The threshold file is setup as follows:

```
1   TestID,ThresholdValue1,ThresholdValue2,...
2   1,3.5,0.5
3   2,3.5,0.5
4   3,20000,50000,4.0
5   4,20000,50000,0.5
6   5
7   6,5,3.3,1.5,1,0
8   7,6
9   8,6
10  M,4,2.5
```

Code Snippet 4

After the first line, each following line is the test thresholds that will be used as the pass or fail conditions of the test. The first line describes the threshold configuration format, an improperly formatted threshold file will end the testing process. Every line shown above must be in the threshold file just as shown in Code Snippet 4. The second line represents Test #1 and each following line must be the next consecutive Test ID. M, or miscellaneous, goes on line 10. The thresholds values are as follows:

Test #1 | Min High Voltage | Max Low Voltage
Test #2 | Min High Voltage | Max Low Voltage
Test #3 | Min Resistance | Max Resistance | Pull-Up Min Voltage
Test #4 | Min Resistance | Max Resistance | Pull_Down Max Voltage
Test #5 | N/A
Test #6 | Voltages Generated By DAC (user can add more test voltages)
Test #7 | Current Drop mA (initial - final)
Test #8 | Current Drop mA (final - initial)
Misc. | 5V Pin Minimum Voltage | 3V3 Pin Minimum Voltage

The test configuration file for a given subject board is named with the following convention, *[subject]Test.config*. The test file is setup as follows:

```
1   TestID,PinID,MuxAddress,MuxEnable
2   1,21,4,3
3   1,23,6,3
4   1,30,5,4
5   1,32,7,4
6   1,37,4,5
7   1,39,6,5
8   1,40,7,5
9   1,45,4,6
10  1,47,6,6
11  1,53,4,7
12  1,55,6,7
```

Code Snippet 5

After the first line, each following line is a test that will be conducted. The first line describes the test configuration format, shown in Code Snippet 5. An improperly formatted test file will end the test before finishing. The Test ID and Pin ID that is recognized by the subject development board goes first. Then, the Multiplexer Address and Enable pin comes next, all separated by commas. It is important to remember the pin ID must go with the *address* and *enable value* for that pin. The exception is with tests #7 and #8. Test #7 does not need the address and enable value and Test #8 has a preconfigured wakeup pin depending on the subject board.

Subject Board Embedded Code

Code Model for Serial Communication Data Packets

There are a few functions that must be altered in order to use the MicroDev code base for a new development board. The *main* must also be altered for initializations unique to the development board and for receiving serial data. The additional functions that must be replaced or written are as follows:

Main.c

```
int command_read(unsigned char data[]);
int command_write(unsigned int pin, unsigned int result, unsigned int test);
int analogRead(int pin);
int digitalRead(int pin);
void analogWrite(int pin, int value);
void digitalWrite(int pin, int logic);
void pinMode(int pin, int mode);
```

```
void configure_sleep_mode(unsigned int sleepmode, unsigned int interruptPin);
void wakeUp();

Setup.c
struct pin pin_set(uint32_t pin, uint32_t clock, GPIO_TypeDef * gpio,  uint8_t pin_id);
void init_pins(struct pin pins[]);
```

Within the Development folder of the MicroDev directory, there are C files to get a jumpstart with developing the new embedded code for the MicroDev system.

The following model demonstrates how to wait for serial communication on the subject board. Additionally, a method of establishing a pin mapping in the embedded code is referenced. Below is a pseudocode example of the STM32 code used to wait for serial communication for the next test.

```
INITIALIZE pin map
MAIN LOOP

IF serial message received
        Read message
        IF message passes decoding
                IF normal test
                        Run Test
                        Write serial message with results
                ELSE IF facade test
                        Write serial message with pin register value
```

Pin Configurations and Setup

Use default configurations, auto-generated code, or the hardware abstraction layer to assist with configuring pins or power modes, if applicable.

The pins must also be identified on the subject development board to run each test. One way of doing that is to create a structure in the embedded code that has the important register values and use the index as the pin ID value, as seen in Code Snippet 7.

```
struct pin pin_set(uint32_t pin, uint32_t clock, GPIO_TypeDef * gpio,  uint8_t pin_id) {

        struct pin P;
        P.pin = pin;
        P.pin_id = pin_id;

        P.clock = clock;
```

```
        P.GPIO = gpio;

        return P;
}

// Initialize pin struct array
void init_pins(struct pin pins[]) {

        // Pin 0 Example | PA5
        pins[0] = pin_set(0x05, 0x01, GPIOA, 0);

        // Pin 1 | PD2
        pins[1] = pin_set(0x02, 0x08, GPIOD, 1);

        // Pin 2 | PC12
        pins[2] = pin_set(0x0C, 0x04, GPIOC, 2);

        // Pin 3 | PC11
        pins[3] = pin_set(0x0B, 0x04, GPIOC, 3);

        ...
}
```

Code Snippet 7

Test Inputs and Outputs

The following figure, Figure 4, describes what is generally communicated through serial communication.

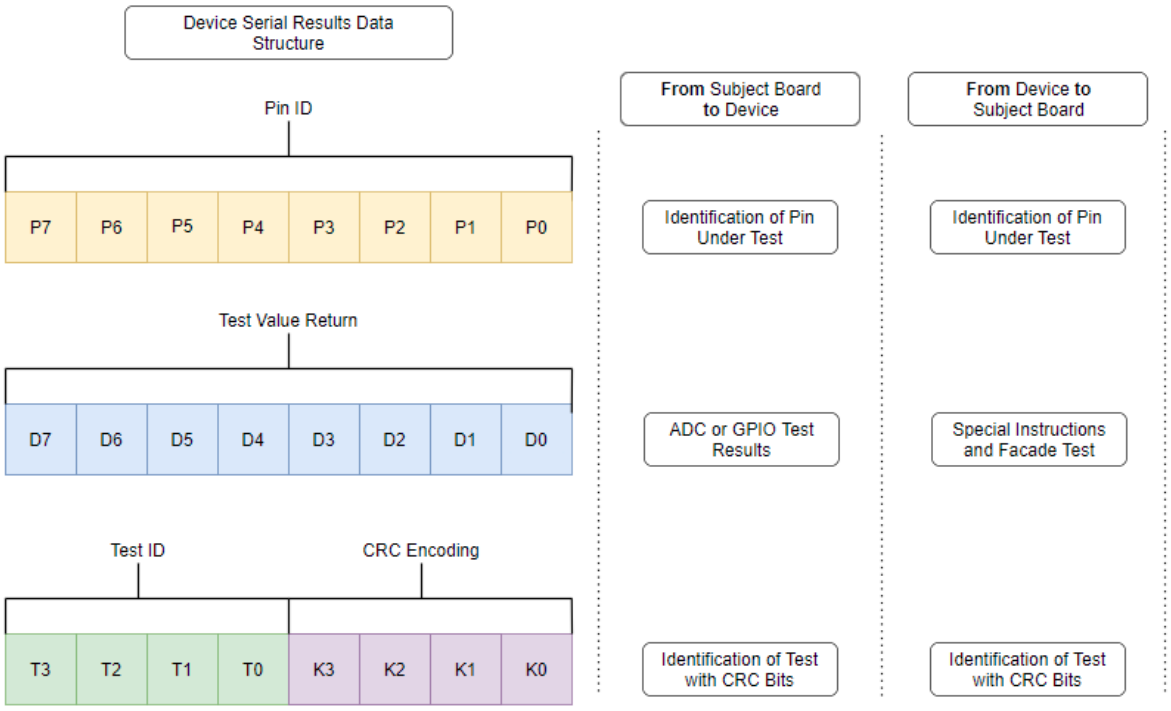


Figure 4: Serial Communication Data Packets

Specifically, each test has its own Test ID value and some of them return a measurement in an 8-bit number. A description of this is below.

Output Test #1 - Sets Voltage with no returned measurement

Output Test Under Load #2 - Sets Voltage with no returned measurement

Input Pull Up Test #3 - Sets Voltage with no returned measurement

Input Pull Down Test #4 - Sets Voltage with no returned measurement

Input Logic Test #5 - Sets Input and Returns a 1 or 0 for High or Low digital signal.

Analog to Digital Converter Test #6 - Sets ADC input and return value shifted to 8 bits.

Power Mode Test #7 - Sets Power Mode and returns nothing

Wakeup Test #8 - Wakes up for low power mode

Testing and Verifying New Code

Refer back to software verification in the design document.

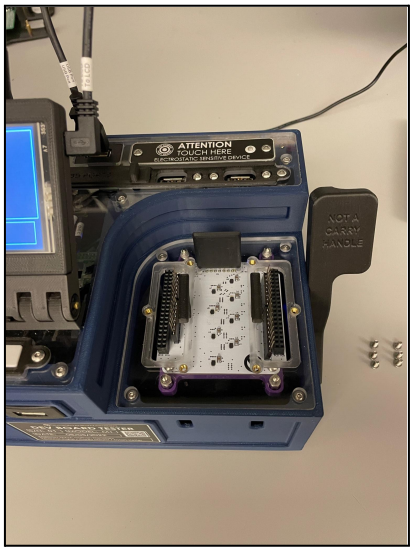
Switching Subject Board Header Interface

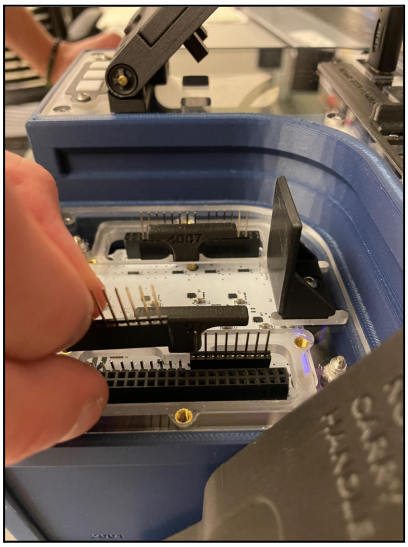
The subject board header interface can be changed by unscrewing the header covers and swapping them with the cover located on the back of the test fixture. Below outlines the steps to complete this.

Step 1: Locate the 6 screws holding down the header covers.

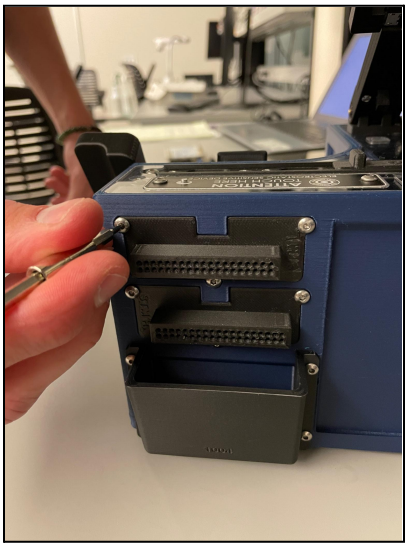
Step 2: Remove the screws using the allen wrench located in the rear compartment and remove the header covers by pressing the handle and rotating the header cover up and over the header extensions.

Step 3: The header extensions should now be exposed. Remove the current header extensions from the PCB.



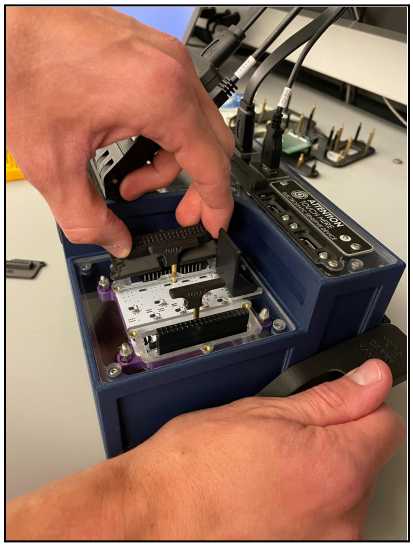


Step 4: Locate the header covers stored on the back of the fixture. Remove the retaining screws.

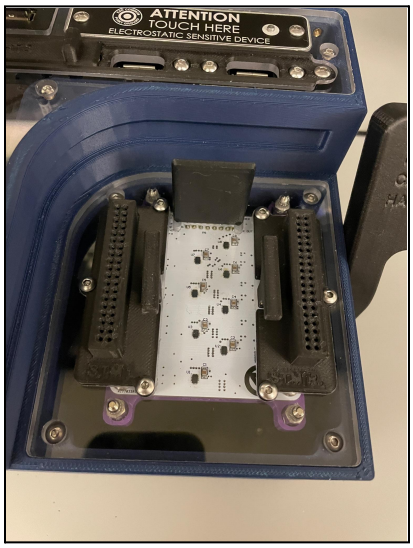


Step 5: Place the new configurations header extensions onto the pcb interface.

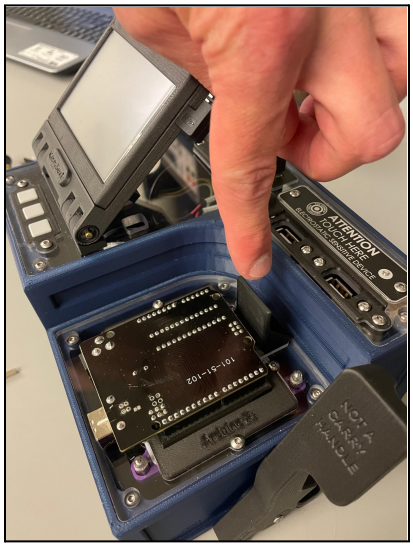
Step 6: Place the header covers over the header extensions by pushing the crank handle down and rotating the header cover into place.



Step 7: Secure the new header configuration in place by using the (6) screws and secure the unused header cover configuration onto the back of the testing fixture.



Step 8: Loosen the rear alignment guide and slide it backwards and place the subject board onto the interface.



Step 9: Slide the alignment guide forward until it touches the subject board. Tighten the screw and place the allen wrench into the rear storage compartment.

The testing fixture is now ready to use.