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:

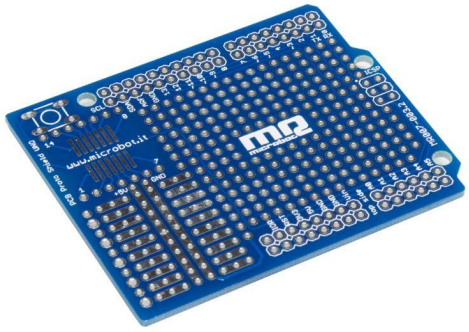

Figure 1: Arduino Prototyping Board

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

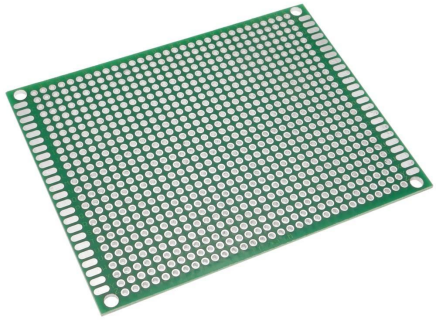

Figure 2: Generic PCB

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.

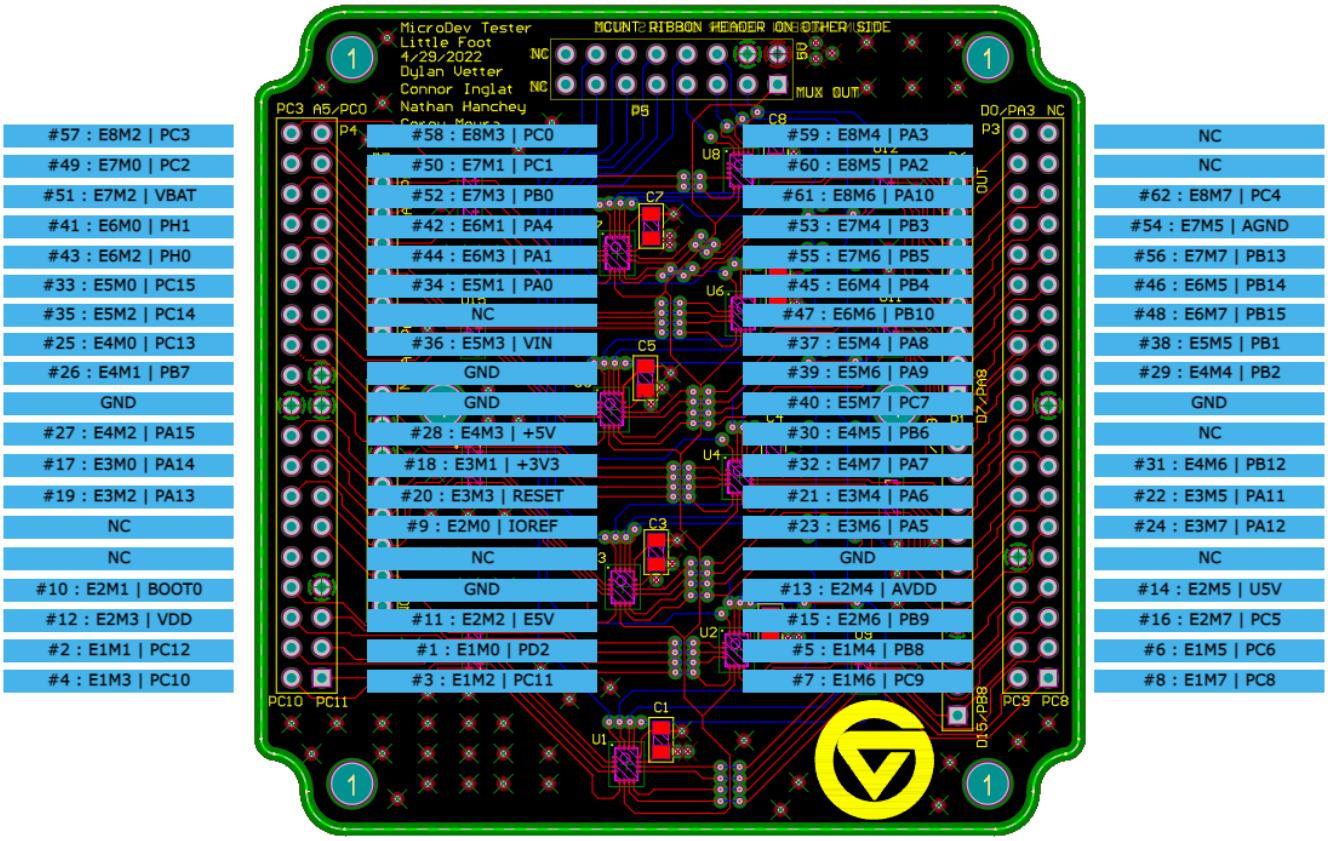

Figure 3: STM Nucleo Pin Mapping

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 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 (works on a loop based on length of array)
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 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.

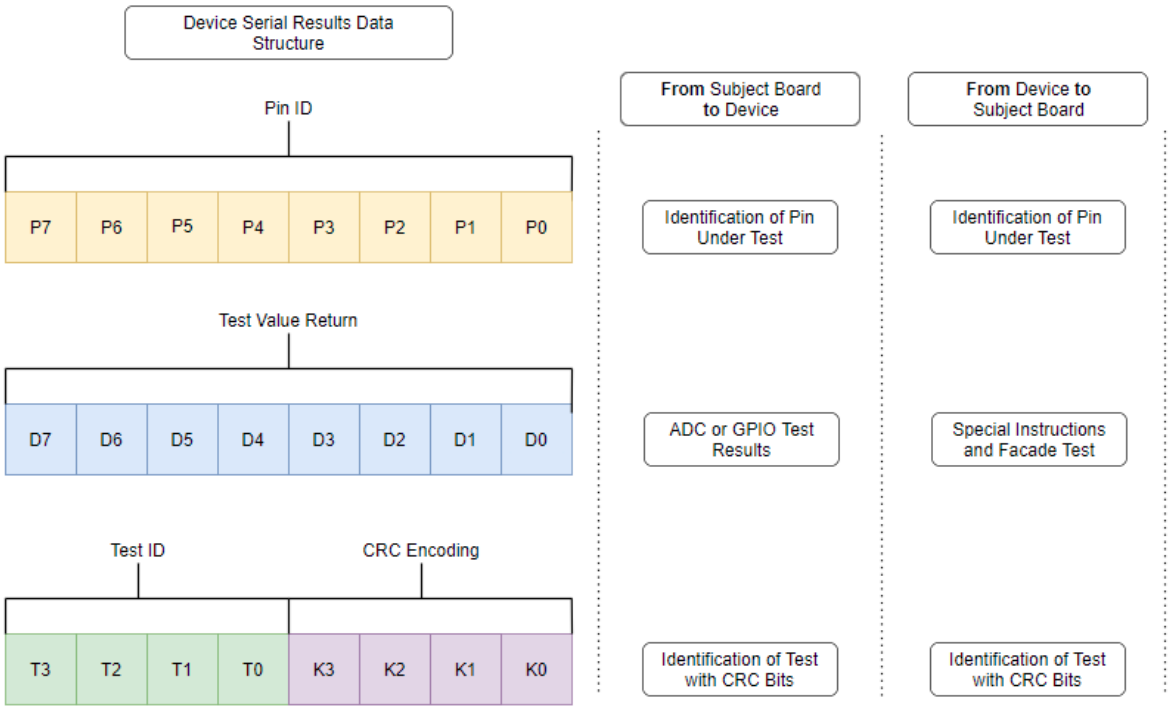

Figure 4: Serial Communication Data Packets

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.