Beta Test Plan – Prototroller

Merrick R., Britton M., Caleb O., Evan Z., Yu-yang H.

Alpha Test Results

We have not been able to perform some parts of the alpha testing because we do not have all the hardware yet (very soon!). However, where functionality can be efficiently tested on our prototype board, we have done so. The rest we have refined in test procedures as part of the Beta Test Plan.

Results: Verification of Average Power Draw by Software Measurement (Pico Prototype)

Test Procedures	Expected Results (0 Modules)	Results	Expected Results (*n* Modules)	Results
1. Choose USB port to plug Prototroller into on Windows host	N/A	N/A	N/A	N/A
2. Measure power consumption for this port using USBDeview	Consuming 0mA	Consuming 0mA	Consuming 0mA	Consuming 0mA
3. Plug Prototroller into host	N/A	N/A	Windows does not report "Power Limit Exceeded"	Windows did not report "Power Limit Exceeded"
4. Measure power consumption for port using USBDeview	Consuming <=50 mA	Consuming 100mA	Consuming ≤ 500mA (following $n * 20$mA line)	1-3 Modules: Consuming 100mA

Interpretation: it would seem measuring power draw via software is not completely accurate. For example, with no modules connected, it was reporting 100mA when, in fact, Pico's do not draw that much.

Results: Verification of Module Rescanning (Pico Prototype)

Test Procedures	Expected Results	Results
1. Ensure Prototroller is plugged into host with no connected modules	N/A	N/A
2. Establish a serial connection to the master board with a console such as PuTTY	Output is displayed on serial console	Output is displayed on serial console
3. Press and release the rescan button	Module rescan initiated and all modules identified as disconnected on serial console	Rescanning for active modules message. Modules 0 through 24 are DISCONNECTED

4. Plug-in a module with an invalid ID to a random slot *k*	No change on serial console	No change
5. Press and release the rescan button	Module rescan initiated, no change on serial console (module *k* shown as disconnected).	No change
6. Plug-in a module with a valid ID to another random slot *m*	No change on serial console	No change
7. Press and release the rescan button	Module rescan initiated, only module *m* shown as connected with corresponding ID	Module shown as connected
8. Observe serial console	Data from module in slot *m* is output	Data from module is output
9. Disconnect the module in slot *m*	No output on serial console	Disconnect triggered log messages indicating the new state. "Module 0 appears to have disconnected or is invalid: DISCONNECTED"
10. Press and release the rescan button	Module rescan initiated; all modules shown as disconnected	Rescanning for active modules message. Modules 0 through 24 are DISCONNECTED
11. Disconnect the module in slot *k*	No change on serial console	No change on serial console
12. Repeat 6-9 until every slot has been evaluated	See 6-9 expected results	...

Interpretation: Our design changed from using 25 modules to 20, so the firmware should be updated to reflect this. Testing for the win!

Results: Firmware Behavioral Testing (Debug Firmware)

Master Device		
Test Procedures	**Expected Result**	**Results**
Device Boots and outputs serial (console) data	Visible COM Port when using debug firmware version	Utilizing USBDeview, master device appears as "Master", and is listed under a COM Port.
Device completes initial scan	Console output of connected modules	Console outputs a list of all 25 disconnected modules (no modules connected). Note, console output does not appear until the rescan button is pressed, but the scan is initiated as soon as the master device is connected to power.
Device Rescan Button begins scan	Console output of rescanned modules	Rescanning for active modules message. Modules 0 through 24 are DISCONNECTED.

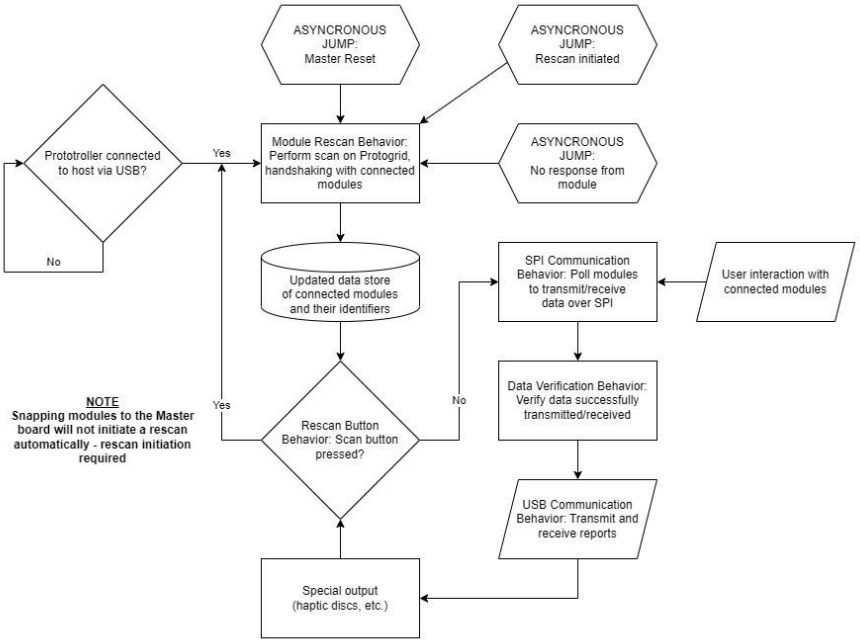
Device receives correct data packets	Console output displays proper data/state of module device	Console receives correct rescan output, displaying button module as connected. The data packets are correct, pressing button outputs a '1', releasing it outputs a '0'.
Device maintains module connection	Console does not display disconnect messages	Console maintains a steady output and displays no disconnect messages if the module is connected and powered.
Module Device		
Test Procedures	**Expected Result**	**Results**
Device Boots and outputs serial (console) data	Visible COM Port when using debug firmware version	COM port was not visible due to it being disabled in fear that print statements would affect our SPI baud rate. After enabling it, the COM port was visible, and we could read serial outputs.
Device reads ADC/GPIO data	Console output of raw data	Joystick module outputs raw data, button module outputs logical high and logical low.
Device calculates computation of data	Console output of computed data	Joystick module outputs raw data, button module outputs logical high and logical low. Master module takes Joystick data and calculates it into normalized data that ranges between (-5, 5).
Device sends data when requested	Console output of successful transmission	Console continually outputs data as master board cycles it via CS.
Device blocks when data not requested	Console does not display additional messages	Console does not output data when not requested, as evidenced by Master device's output being uniform and outputting consistent data in the order of currently connected modules.
Data read is within accurate bounds	Console outputs acceptable data value (tolerance differs based on module type)	The data in master is accurate and module side is more precise and within accurate bounds.

Interpretation: Behavioral testing was a success. We had commented out serial logging module-side to not interfere with SPI (i.e., module readily available to transmit and receive data). This brings up a good point – determining if we want module-side logging at all. Perhaps a debug mode since the logging would only be visible via SWD (no USB-C connector on the module boards) and never during normal use.

Expected Behavior

The core expected behavior has <u>not</u> changed since the Alpha Test Plan. We have already encapsulated the bulk of it, save for a few minor details. These details are related to the fact that we are now utilizing our own custom hardware rather than the RP2040 Pico's utilized in the prototype and alpha builds.

<u>Overall Coarse Expected Behavior</u>



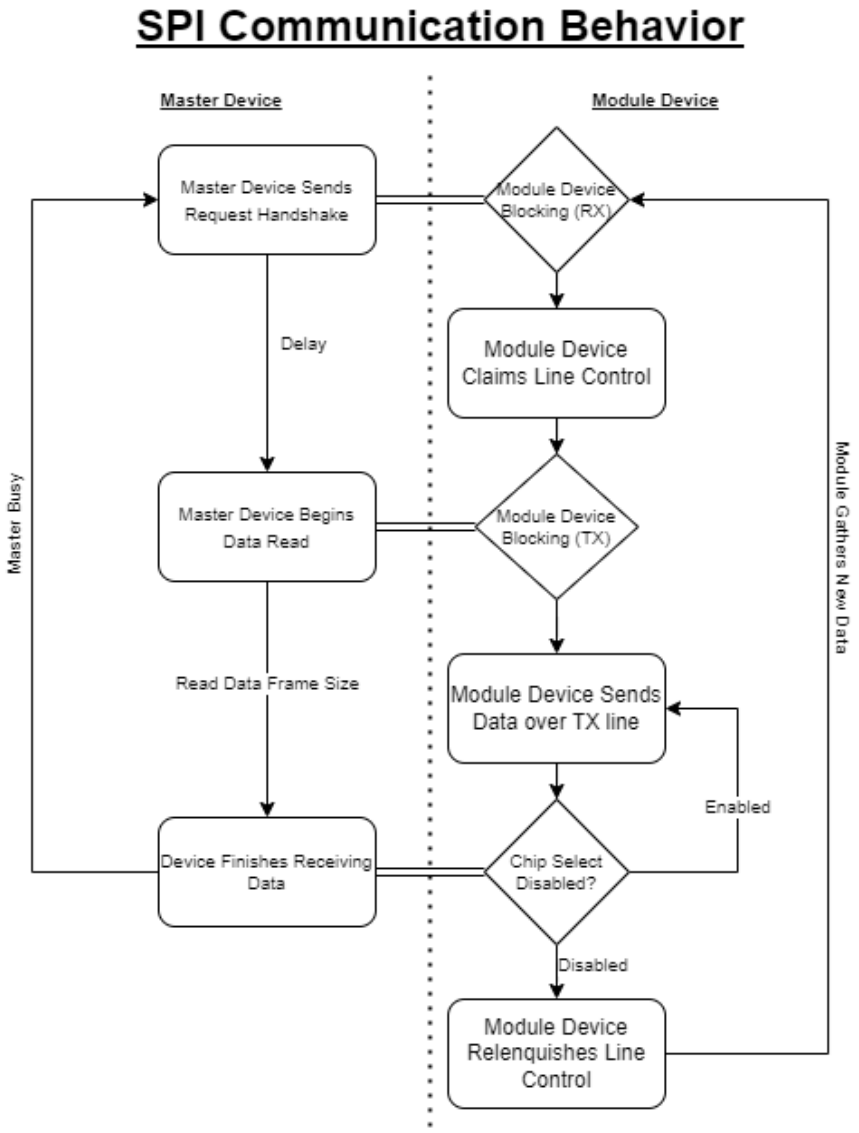
<u>Master Board Flashing Expected Behaviors</u>

<u>Action</u>	**<u>Result</u>**
Do Nothing *(*Powered*)*	N/E - Master RP2040 stays in current mode
Press the "BOOTSEL" button *(*Powered*)*	N/E - Master RP2040 stays in current mode
Press the "RESET" button *(*Powered*)*	Master RP2040 begins executing program code from the beginning of flash
Press and hold the "BOOTSEL" button, press and release the "RESET" button, drag UF2 binary firmware to mount *(*Powered*)*	Master RP2040 mounts to the host in bootloader mode. Master RP2040 de-mounts after firmware is flashed and begins executing program code from the beginning.
Press and hold the "BOOTSEL" button, disconnect USB (if applicable), connect USB, drag UF2 binary firmware to mount	Master RP2040 mounts to the host in bootloader mode. Master RP2040 de-mounts after firmware is flashed and begins executing program code from the beginning.
Alternative: flash master firmware with SWD debug apparatus	Master RP2040 begins executing program code from the beginning

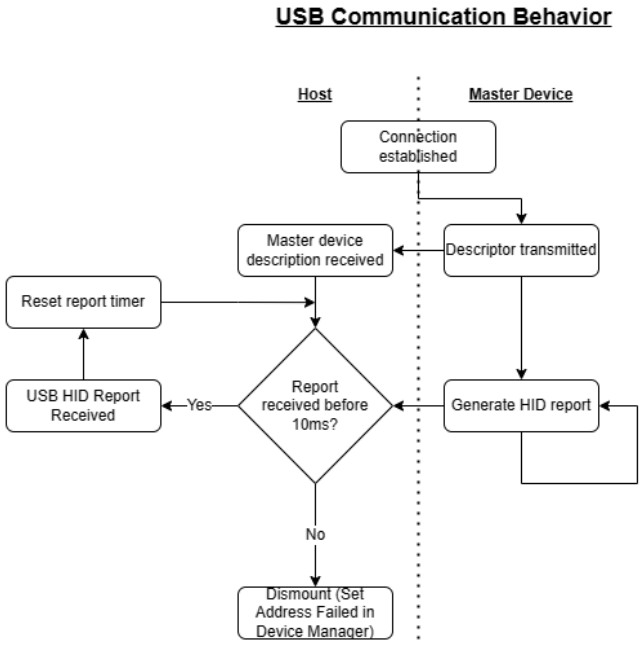
Module Board Flashing Expected Behaviors

Action	**Result**
Do Nothing *(*Powered*)*	N/E – Module RP2040 stays in current mode
Short “BOOTSEL” pad *(*Powered*)*	N/E - Module RP2040 stays in current mode
Pull “RESET” pin low *(*Powered*)*	Module RP2040 begins executing program code from the beginning of flash
Short “BOOTSEL” pad, pull “RESET” pin low, pull “RESET” pin high	Module RP2040 does not mount, because it does not have a USB connection with the host…
Flash *unique module firmware* with SWD debug apparatus	Module RP2040 begins executing program code from the beginning

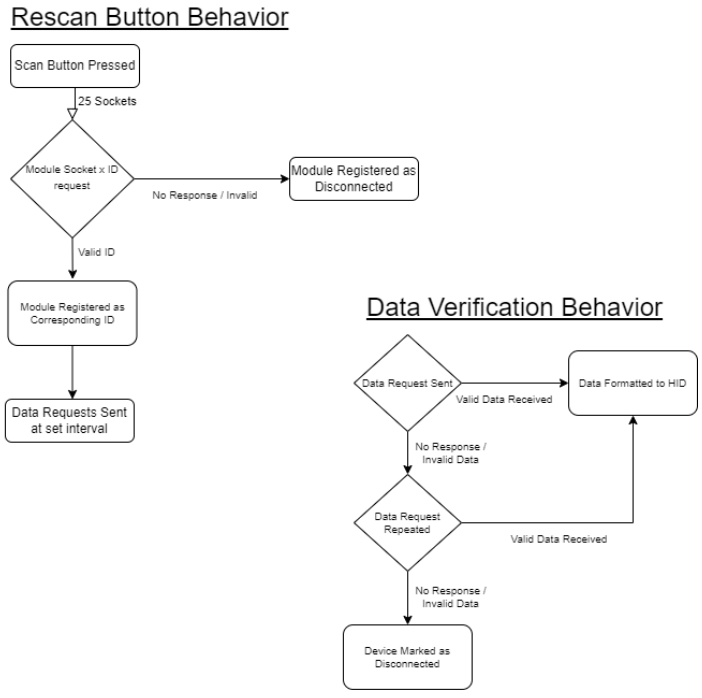
Master ⇔ Module SPI Communication Expected Behaviors



Host ⇔ Master USB Communication / HID Report Expected Behaviors



Module Rescanning / Data Verification Expected Behaviors



Additionally, most logical blocks of source-code are documented with functional purpose, parameters, edge-cases, etc.

Beta Test Procedures

Many test procedures are borrowed from the Alpha Test Plan, since again we physically have not been able to test on our custom hardware. We would like to repeat these on the custom hardware.

Firmware Behavioral Testing (Debug Firmware)

NOTE: An initial test suite for functionality testing master firmware has been created, please see repository for more details.

Master Device	
Test	**Expected Result**
Device Boots and outputs serial (console) data	Visible COM Port when using debug firmware version
Device completes initial scan	Console output of connected modules, HID packet descriptor matches connected modules, LED Complete indicator code displayed
Device Rescan Button begins scan	Console output of rescanned modules, LED Scanning indicator code displayed
Device receives correct data packets	Console output displays proper data/state of module device, HID Report packets contain data matching serial port
Device maintains module connection	Console does not display disconnect messages
Module Device	
Test	**Expected Result**
Device Boots and outputs serial (console) data	Visible COM Port when using debug firmware version
Device reads ADC/GPIO data	Console output of raw data
Device calculates computation of data	Console output of computed data
Device sends data when requested	Console output of successful transmission
Device blocks when data not requested	Console does not display additional messages
Data read is within accurate bounds	Console outputs acceptable data value (tolerance differs based on module type)

Verification of Average Power Draw by USB Multimeter Tool

Test Procedures	Expected Results (0 Modules)	Expected Results (*n* Modules)
1. Plug in the Prototroller to the host through a USB Multimeter Tool	N/A	N/A
2. Measure power consumption with the tool	Consuming ≤ 100 mA	Consuming n* ~50mA, not exceeding 500mA

Verification of Module Rescanning

Test Procedures	Expected Results
1. Ensure Prototroller is plugged into host with no connected modules	N/A
2. Establish a serial connection to the master board with a console such as PuTTY	Output is displayed on serial console
3. Press and release the rescan button	Module rescan initiated and all modules identified as disconnected on serial console
4. Plug-in a module with an invalid ID to a random slot *k*	No change on serial console
5. Press and release the rescan button	Module rescan initiated, no change on serial console (module *k* shown as disconnected).
6. Plug-in a module with a valid ID to another random slot *m*	No change on serial console
7. Press and release the rescan button	Module rescan initiated, only module *m* shown as connected with corresponding ID
8. Observe serial console	Data from module in slot *m* is output
9. Disconnect the module in slot *m*	No output on serial console
10. Press and release the rescan button	Module rescan initiated; all modules shown as disconnected
11. Disconnect the module in slot *k*	No change on serial console
12. Repeat 6-9 until every slot has been evaluated	See 6-9 expected results

Verification of SPI Communication

Test Procedures	Expected Results
Probe Rx and Tx bus lines for master board with oscilloscope triggers	Handshake is sent, and data is correctly received
Probe CS bus lines for master board with oscilloscope triggers	All modules currently connected are visibly shown on the oscilloscope

Verification of HID Drivers

Test Procedures	Expected Results
Use Device Monitoring Studio to view HID reports	All data packets via reports are correct, and are consistent with module board inputs

Verification of Latency Constraints

Measuring the latency of controllers in general is difficult, but not impossible with the right hardware. In our case, the methodology is to aim a high-speed camera at our host display and a LED connected to a button module mapped to a left-click schema. Then we can compare – in editing software - the delta time between the LED lighting and a UI element on the display changing in response to the left-click. Preferably, use a monitor with 144+Hz refresh rate for a more accurate measurement.

Test Procedures	Expected Results
1. Plug in Prototroller to host with one module connected: button, with left-click schema. Connect Switch => LED => Resistor => +3V3	N/A
2. Press and release the rescan button	N/A
3. Set up mouse on element to be left-clicked (action must modify UI in some way)	N/A
4. Start recording with 240fps, giving smallest possible increment of 4.16ms	N/A
5. Press and release the button	LED ON for as long as the button is pressed.
6. Stop recording	N/A
7. In post-processing software, find the frame the LED lights up and the frame the UI responds.	N/A
8. Convert the difference in frames to a response time: #frames * (1/240)	Response time should be ≤ 20 ms for acceptable result, or ≤ 6ms for nominal result

Verification of Hardware Signals + Circuitry

Test Procedures	Expected Results
Continuity Test between all 7 connection points of master-module connectors	Continuity between each line, no continuity across lines
No shorting across decoupling capacitors	No continuity across capacitor pads
Line value fluctuations across decoupling capacitors	No noise/wobble exceeding 3% of the average constant value of the line (ex: 3Volt ±0.1V), idle or under load
Flash storage interfacing properly executing with RP2040	Program memory is maintained upon loss of power
Oscillator clock speed is within proper limits	Frequency of crystal oscillator signal is within 3% of 12 MHz
Chip Select signals properly activate through decoders	Each numbered chip select trace is held at logical high unless signaled in firmware. Firmware value (1,2,3, etc.) corresponds to proper CS line