TESTING

This test document is for `V0.2.0` of the Sunscatter Gen II boards. Test instructions are not guaranteed to work with different versions of the hardware.

Electrical Tests

1. Verify 3V3, 5V0 power testpoints report correctly when USB is plugged into the NUCLEO.
2. Verify that 3V3, 5V0, 12V power testpoints report correctly when 12V is applied through the molex connector.

Firmware Tests

Test firmware can be found in the fw/tests folder. The tests should be loaded into MBED and flashed onto the NUCLEO that is plugged into the board prior to starting the test. Check whether the firmware listed in the main.cpp file matches that of this document prior to starting the test.

Tests are designed to build upon each other with minimal revision or changes in structure. I.E., the same functions and routines in sensor tests can be found in the following tests in the list.

1. LED test
    1. Liveliness - verify that Error, Tracking, Heartbeat, CAN_TX/CAN_RX LEDs turn on and blink at 1 Hz.
2. CAN tests
    1. Loopback - HEARTBEAT CAN message can be sent and received in loopback configuration.
    2. With a secondary device - HEARTBEAT CAN message can be sent and received between two devices.
3. PWM tests
    1. Liveliness - verify that the gate driver can be actuated at a known frequency.
    2. Correctness - verify that driving the gate driver to a specific input results in the correct state of each switch.
4. Sensor tests
    1. Liveliness - verify that measurements can be taken from each sensor.

    2. Variance - verify that the measurements taken at known test conditions remain stable with low variance according to requirements.
    3. Accuracy - verify that the measurements taken at known test conditions remain accurate according to requirements.
5. Boost test
    1. Liveliness - verify that the converter boosts the output to an appropriate voltage when hooked up to a source and load.
    2. Performance - verify that the converter boosts at various input/output ratios and meets specific power transfer and efficiency requirements when hooked up to a source and load.
6. PID controller test
    1. Liveliness - verify that the converter settles to a fixed boost ratio when hooked up to a source and a load.
    2. Stability - verify that the converter can accept input and output voltage noise of a known frequency and amplitude profile and remain stable according to requirements.

LED Test Instructions

1. Ensure that the electrical tests pass as described.
2. Ensure that the `fw/tests/led_test/main.cpp` is compatible with the hardware and reports in the code that it is for board revision `v0.2.0`.
3. Flash the program `fw/tests/led_test` onto the STM32 Nucleo board.
4. Plug in USB power.
5. The five onboard LEDs, marked HRTBT, TRACK, ERROR, CANTX, and CANRX should toggle at `1 Hz` simultaneously.

CAN Test Instructions

1. Ensure that the electrical tests pass as described.
2. Ensure that the `fw/tests/can_test/main.cpp` is compatible with the hardware and reports in the code that it is for board revision `v0.2.0`.
3. Set the defines according to the test context.
    1. Loopback test: `__LOOPBACK__` = 1.
    2. Interdevice test: `__LOOPBACK__` = 0.
        1. Device A: `__DEVICE_A__` = 1. (Sends)
        2. Device B: `__DEVICE_B__` = 0. (Receives)
4. Flash the program `fw/tests/can_test` onto the STM32 Nucleo board.
5. Plug in USB power.
6. Loopback test
    1. The five onboard LEDs, marked HRTBT, TRACK, ERROR, CANTX, and CANRX should flash at `1 Hz` simultaneously.
7. Interdevice test
    1. The HRTBT, ERROR LEDs should blink on Device A, and the TRACK LED should blink on Device B, in sync with each other. Based on the timing, they may blink alternatively or at the same time. The LEDs CANTX and CANRX should be on at all times.
    2. If both devices are not connected to each other, the ERROR LED on Device A will no longer turn off after one toggle.

PWM Test Instructions

1. Ensure that the electrical tests pass as described.
2. Ensure 8A fuses are added to the PCB.
3. Ensure that the `fw/tests/pwm_test/main.cpp` is compatible with the hardware and reports in the code that it is for board revision `v0.2.0`.
4. Set the defines according to the test context.
    1. Checking liveliness: `PWM_DUTY` = 0.5
    2. Checking correctness:
        1. Observing slew delay: `PWM_DUTY` = 0.5
        2. Low side switch closed, high side switch open: `PWM_DUTY` = ...
        3. High side switch closed, low side switch opwn: `PWM_DUTY` = ...
5. Flash the program `fw/tests/pwm_test` onto the STM32 Nucleo board.
6. Connect a logic analyzer (e.g. Saleae) to the board. Set CH1 to LOW side switch, with probes attached to SW1 and GND. Set CH2 to HIGH side switch, with probes attached to SW2 and VSW. Set CH3 to GATE signal, with probes attached to PWM and GND.
7. Plug in USB power.
8. Liveliness test
    1. The CH3 signal should be toggling at 50% duty cycle, at 50khz frequency.
    2. The high side output and low side output should be toggling at 50% duty cycle, at 50khz frequency.
9. Correctness test
    1. When `PWM_DUTY` = 0.5, the signals at CH1 and CH2 should be alternating. With a high enough sample rate (> 5, 10MSPS), view the signals through the analog feature and zoom into the edge transitions. Both transitions should show a delay of `20ns`, with both signals being low during the transition.
    2. When `PWM_DUTY` = ..., the signal at CH1 should be high, and a multimeter should identify VSW and GND as being shorted.
    3. When `PWM_DUTY` = ..., the signal at CH2 should be high, and a multimeter should identify VSW and VBATT as being shorted.

Sensor Test Instructions

1. Ensure that the electrical tests pass as described.
2. Ensure 8A fuses are added to the PCB.
3. Ensure that the `fw/tests/sensor_test/main.cpp` is compatible with the hardware and reports in the code that it is for board revision `v0.2.0`.
4. Set the defines according to the test context.
    1. `PWM_DUTY` should be set to 1.0 to ...
    2. `PWM_DUTY` should be set to 0.0 to ...
5. Clear the cache (save the initial entry somewhere) in the sensors struct (`L67, L68`) to have a default slope of 1.0 and y-intercept of 0.0 for each sensor.
6. Flash the program `fw/tests/sensor_test` onto the STM32 Nucleo board.
7. Plug in USB power.
8. Sensor output should be recorded on the CLI serial terminal (MBED Studio, PuTTY, etc).
9. Voltage sensor test
    1. Input voltage sensor
        1. Open circuit the LOW SIDE SWITCH by applying `PWM_DUTY` = ...

        2. Apply 0 - 113V on the input terminals (ARR) of the board. Do this at a known range and resolution (e.g., 0 - 110V, in steps of 5V). Wait for an appropriate amount of time per step for the value to settle.
    2. Output voltage sensor
        1. Open circuit the LOW SIDE SWITCH by applying `PWM_DUTY` = ...
        2. Apply 0 - 168 V on the output terminals (BATT) of the board. Do this at a known range and resolution (e.g., 0 - 165V, in steps of 5V). Wait for an appropriate amount of time per step for the value to settle.
10. Current sensor test
    1. Input, output current sensor
    2. Short circuit the HIGH SIDE SWITCH by applying `PWM_DUTY` = ...
    3. Short circuit the output terminals (BATT) of the board.
    4. Apply 0 - 8.25A on the input terminals (ARR) of the board. Do this at a known range and resolution (e.g., 0 - 8A, in steps of 0.25A). Wait for an appropriate amount of time per step for the value to settle.
11. Save the test data in a CSV.
12. Graph the test data on a spreadsheet application (Google Sheets, Microsoft Excel, etc). Identify each step and capture that point (ignore points in between steps). This may be automated by using a clustering algorithm script in Python (left as an exercise to the reader).
13. Take the data points associated with each step and plot them in a scatter graph and determine the Line of Best Fit (LOBF). This should consist of a linear regression with a slope and y-intercept.
14. Derive the slope correction and the Y intercept correction that should be applied to the original transformation (see the calibration function for more detail).
15. Update the cache in the sensors struct with the new values.
16. Rerun the voltage and current sensors test with the loaded cache values, and double check that the output from the terminal makes sense. Repeat the correction if needed to minimize the regression error.

Boost Test Instructions

> WARNING: this test involves LIVE high power electrical systems and CAN be FATAL. All participants should prepare appropriate precautions including high voltage electrical gloves, long sleeved shirts and pants, and closed toed shoes. Participants should NOT touch the device or handle electronics unless properly trained and supervised.

> NOTE: The instructions described were with a resistive load in mind, and should be adjusted appropriately with a battery load.

1. Ensure that the electrical tests pass as described.
2. Ensure `8A` fuses are added to the PCB.
3. Ensure that the sensor test has passed and the cache in the sensors struct (`L73, L74`) have been updated with the latest test calibration values.
4. Ensure that the `fw/tests/boost_test/main.cpp` is compatible with the hardware and reports in the code that it is for board revision `v0.2.0`.
5. Determine the test context. Derive the following:

    1. Input voltage
    2. Duty cycle
    3. Load voltage or resistance

    A google sheets is provided to simplify this process. Sheets link. Update the green entries to update the orange entries.

6. For each test context, size and connect the appropriate resistive load to the output terminals. Use a multimeter to observe that the resistance up to the input terminals is as expected (additional, significant resistance through the lines may occur). DO NOT size a resistive load that is rated for the expected power transfer (e.g. 50W power resistor for 100W transfer). DO NOT place the resistive load onto or near flammable objects, as it will get hot at high power loads.

7. For each test context, attach the DC supply to the input and ensure the supply output is OFF. Set up the required output voltage (of the DC supply) to the test context, and set the current limit to slightly higher than the expected current draw, otherwise the system will be current limited.

8. Set the defines according to the test context.

    1. `PWM_DUTY` should be set to the value derived in the previous step.

9. Update the redlines observed in function `event_check_redlines`. If violated, the test will end prematurely without collecting the required success data.

10. Flash the program `fw/tests/sensor_test` onto the STM32 Nucleo board.

11. Plug in USB power.

12. Sensor output should be recorded on the CLI serial terminal (MBED Studio, PuTTY, etc).

13. Turn on the output of the DC power supply and prepare to turn it off in case of catastrophic failure. Check if the voltage and current draw is as expected by the script from step 5.

14. Run the test for an appropriate amount of time needed for the sensor data to stabilize and the data to be collected. Do NOT overheat the resistive load until it is burning to the touch.

15. Collate all the information and run appropriate analysis. See the Power Measurements google sheet Sheet2 for an example of several independent tests and results.

PID Test Instructions

> WARNING: this test involves LIVE high power electrical systems and CAN be FATAL. All participants should prepare appropriate precautions including high voltage electrical gloves, long sleeved shirts and pants, and closed toed shoes. Participants should NOT touch the device or handle electronics unless properly trained and supervised.

> NOTE: The instructions described were with a resistive load in mind, and should be adjusted appropriately with a battery load.

1. Ensure that the electrical tests pass as described.

2. Ensure `8A` fuses are added to the PCB.

3. Ensure that the sensor test has passed and the cache in the sensors struct (`L73, L74`) have been updated with the latest test calibration values.

4. Ensure that the `fw/tests/pid_test/main.cpp` is compatible with the hardware and reports in the code that it is for board revision `v0.2.0`.

5. Determine the test context. Derive the following:

    1. Input voltage
    2. Duty cycle
    3. Load voltage or resistance

    A google sheets is provided to simplify this process. Sheets link. Update the green entries to update the orange entries.

6. For each test context, size and connect the appropriate resistive load to the output terminals. Use a multimeter to observe that the resistance up to the input terminals is as expected (additional, significant resistance through the lines may occur). DO NOT size a resistive load that is rated for the expected power transfer (e.g. 50W power resistor for 100W transfer). DO NOT place the resistive load onto or near flammable objects, as it will get hot at high power loads.

7. For each test context, attach the DC supply to the input and ensure the supply output is OFF. Set up the required output voltage (of the DC supply) to the test context, and set the current limit to slightly higher than the expected current draw, otherwise the system will be current limited.

8. Turn on the output of the DC power supply and prepare to turn it off in case of catastrophic failure. Check if the voltage and current draw is as expected by the script from step 5.

9. Set the defines according to the test context.

    1. `PWM_DUTY_START` should be set to some reasonable starting value within bounds.
    2. `SINK_TARGET` should be set to the value derived in the previous step.
    3. `NOISE_ON` should be set to 1 if running the stability test.
    4. `SOURCE_NOISE_AMPLITUDE` should be set depending on the test requirements.
    5. `SINK_NOISE_AMPLITUDE` should be set depending on the test requirements.

10. Update the redlines observed in function `event_check_redlines`. If violated, the test will end prematurely without collecting the required success data.

11. Flash the program `fw/tests/sensor_test` onto the STM32 Nucleo board.

12. Plug in USB power.

13. Sensor output should be recorded on the CLI serial terminal (MBED Studio, PuTTY, etc).

14. Run the test for an appropriate amount of time needed for the sensor data to stabilize and the data to be collected. Do NOT overheat the resistive load until it is burning to the touch. Observe that the duty cycle and output voltage converge to an expected value as derived in step 5. It should remain stable in both the liveliness and stabilty tests, although the latter might oscillate slightly larger and longer.

15. Adjust the input voltage to various points and observe that the duty cycle and output voltage converge to an expected value as derived in step 5. Be prepared to shut off the output if the device goes out of control or the redlines check fails.

16. Collate all the information and run appropriate analysis. Observed should be a time series graph of the input voltage, output voltage, input current, output current, duty cycle, and additionally power transfer and loss metrics. It should be observed that the output voltage and duty cycle converge back to the `SINK_TARGET` regardless of the input voltage. Check for abnormalities in expected power loss over the test period. Observe for stability variations at different operating points. Analyze the convergence speed at different step sizes.

Main Program Test

> WARNING: this test involves LIVE high power electrical systems and CAN be FATAL. All participants should prepare appropriate precautions including high voltage electrical gloves, long sleeved shirts and pants, and closed toed shoes. Participants should NOT touch the device or handle electronics unless properly trained and supervised.

> NOTE: The instructions described were with a photovoltaic array source battery load in mind. Do not use otherwise.

1. Ensure that the electrical tests and firmware tests pass as described.
2. Ensure `8A` fuses are added to the PCB.
3. Ensure that the sensor test has passed and the cache in the sensors struct (`L113, L114`) have been updated with the latest test calibration values.
4. Ensure that the `fw/src/main.cpp` is compatible with the hardware and reports in the code that it is for board revision `v0.2.0`.
5. Set the defines according to the test context.
    1. `__DEBUG__` - 0 if CAN controlled, 1 if auto start and reporting through USB.
    2. `PWM_DUTY_START` - Starting boost duty cycle.
    3. `HEARTBEAT_FREQ` - Frequency of heartbeat LED and liveliness report.
    4. `REDLINE_FREQ` - Frequency of redline checks.
    5. `MEASURE_FREQ` - Frequency of sensor measurements.
    6. `PID_FREQ` - Frequency of PID controller.
    7. `MPPT_FREQ` - Frequency of MPPT algorithm.
    8. `FILTER_WIDTH` - Window size of sensor filters.
    9. `PID_P_COEFF`, `PID_I_COEFF`, `PID_D_COEFF` - PID control coefficiencts.
    10. `MIN_INP_VOLT` ... `MAX_DUTY` - Redline check bounds.
6. Update the redlines observed in function `event_check_redlines`. If violated, the test will end prematurely without collecting the required success data.
7. Flash the program `fw/tests/sensor_test` onto the STM32 Nucleo board.
8. Plug in 12V/USB power.
9. Keep the solar array contactor disconnected and battery contactor disconnected. These are LIVE high power electronics and electrical gloves must be used at all times! Keep the battery precharge circuit ON.
10. Attach the solar array leads to the XT60 connectors for the array input, with proper care taken that the positive lead is hooked to the positive terminal and vice versa for negative.

11. Attach the battery array leads to the XT60 connectors for the array input, with proper care taken that the positive lead is hooked to the positive terminal and vice versa for negative.
12. Connect the contactors and monitor the data output through CAN/USB for proper and gradual operation.
13. Disable the battery precharge circuit.
14. Monitor system vitals until the end of the test or a redline occurs.
15. Shut down operation IN ORDER by (1) enabling the precharge circuit, (2) disconnecting the battery contactor, and (3) disconnecting the array contactor.