

Lab Entry – 2026-01-18

Metadata

- Date: 2026-01-18
- Project: Off Grid Solar Battery Charger
- Board / Rev: INA219 Breakout board #2
- Scope: Verify the functionality of the INA219 Breakout board

Objective

Verify that we can write to the INA219 IC. Verify that we can read bus voltage register. Verify that we can read the Current register

Setup

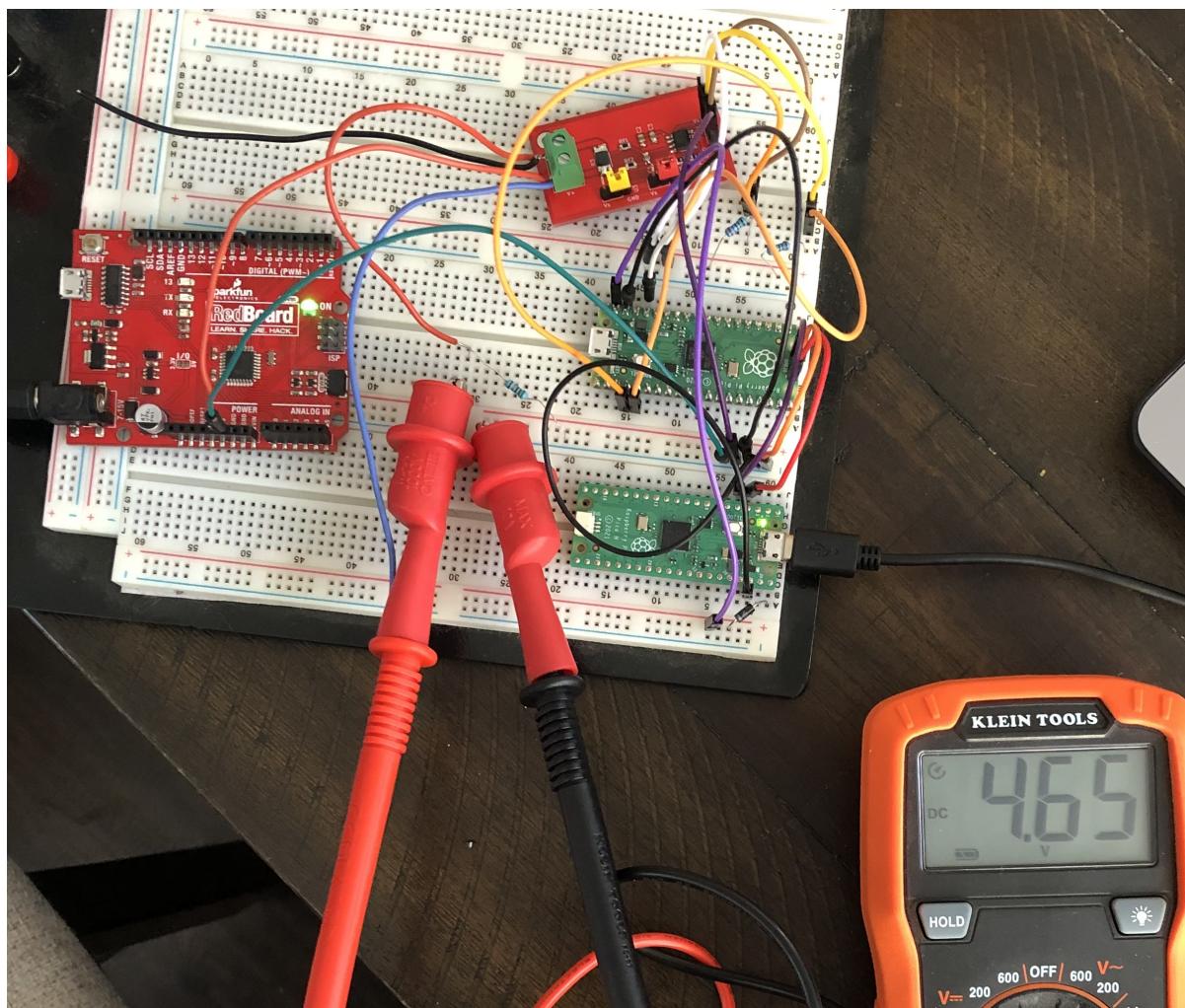


Figure 1: HIL Setup for Bus Voltage Verification

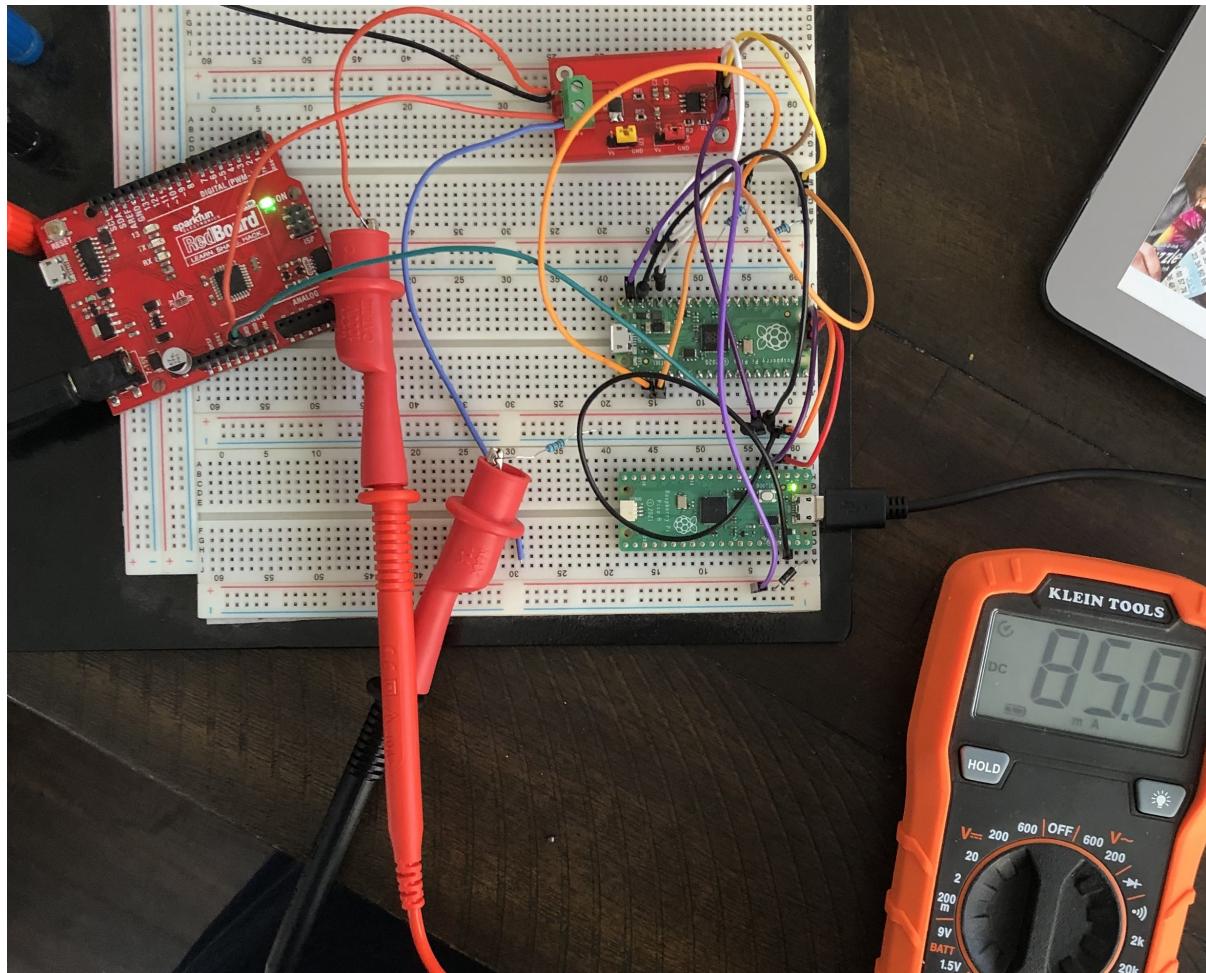


Figure 2: HIL Setup for Current Reg Verification

The screenshot shows a debugger interface with the following details:

- File:** I2C_Test.c
- Watch Tab:**
 - Datarx_Current:** Address [0] = 0x24, Address [1] = 0x4a
 - Datarx_BusVoltage:** Address [0] = 0x24, Address [1] = 0x4a
- Code View:**

```

1  #include <stdio.h>
2
3  #define I2C_PORT i2c1
4  #define INA219_ADDR 0x40
5  #define I2C_SDA 2
6  #define I2C_SCL 3
7
8
9
10
11 uint8_t Datarx_Current[2];
12 uint8_t Datarx_BusVoltage[2];
13
14
15
16
17 int main() {
18     stdio_init_all();
19     i2c_init(I2C_PORT, 100000);
20     gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
21     gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
22     gpio_pull_up(4);
23     gpio_pull_up(5);
24
25     uint8_t reg[3] = {
26         0x05, //write to the calibration reg
27         0x0E, // calibration MSB
28         0x50 // calibration LSB
29     };
2
30     //update calibration register
31     int r = i2c_write_blocking(I2C_PORT, INA219_ADDR, &reg[0], 3, false);
32
33     // if successfully updated calibration reg get the measured current
34     if(r ==3){
35         uint8_t Current_Reg_Addr = 0x04;
36         r = i2c_write_blocking(I2C_PORT, INA219_ADDR, &Current_Reg_Addr, 1, true);
37
38         if(r==1){ //successfully updated pointer to current reg read word
39             r = i2c_read_blocking(I2C_PORT, INA219_ADDR, &Datarx_Current[0], 2, false);
40         }
41     }
42
43     r =0; //reset r
44     reg [0] = 0x02; //update pointer to bus voltage register
45     r = i2c_write_blocking(I2C_PORT, INA219_ADDR, &reg[0], 1, true);
46
47     if (r == 1){ // if pointer successfully updated read word
48         r = i2c_read_blocking(I2C_PORT, INA219_ADDR, &Datarx_BusVoltage[0], 2, false);
49     }else{
50         printf("INA219 did NOT ACK at 0x40\n");
51     }
52     while (1) tight_loop_contents();
53
54     return 1;
55 }
```

Figure 3: Firmware Setup for Bus Voltage Verification

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure with files like `I2C_Test.c`, `launch.json`, and `i2c.h`.
- Variables:** Local variable `reg` is defined as `[3]`. Global variable `Datarx_Current` is shown with values `0 = 0x0` and `1 = 0x57`.
- Watch:** The variable `Datarx_BusVoltage` is selected, showing its value as `[2]`.
- Code Editor:** The `I2C_Test.c` file contains C code for initializing the I2C port, setting up GPIO functions, and reading from the INA219 sensor. The current line of code is highlighted at line 52.

```

4
5
6 #define I2C_PORT i2c1
7 #define INA219_ADDR 0x40
8 #define I2C_SDA 2
9 #define I2C_SCL 3
10
11
12 uint8_t Datarx_Current[2];
13 uint8_t Datarx_BusVoltage[2];
14
15
16
17 int main() {
18     stdio_init_all();
19     i2c_init(I2C_PORT, 100000);
20     gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
21     gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
22     gpio_pull_up(4);
23     gpio_pull_up(5);
24
25     uint8_t reg[3] = {
26         0x05, //write to the calibration reg
27         0x0E, // calibration MSB
28         0x50 // calibration LSB
29     };
29
30     //update calibration register
31     int r = i2c_write_blocking(I2C_PORT, INA219_ADDR, &reg[0], 3, false);
32
33     // if successfully updated calibration reg get the measured current
34     if(r == 3){
35         uint8_t Current_Reg_Addr = 0x04;
36         r = i2c_write_blocking(I2C_PORT, INA219_ADDR, &Current_Reg_Addr, 1, true);
37
38         if(r == 1){ //successfully updated pointer to current reg read word
39             r = i2c_read_blocking(I2C_PORT, INA219_ADDR, &Datarx_Current[0], 2, false);
40         }
41     }
42
43     r = 0; //reset r
44     reg[0] = 0x02; //update pointer to bus voltage register
45     r = i2c_write_blocking(I2C_PORT, INA219_ADDR, &reg[0], 1, true);
46
47     if (r == 1){ //if pointer successfully updated read word
48         r = i2c_read_blocking(I2C_PORT, INA219_ADDR, &Datarx_BusVoltage[0], 2, false);
49     }else{
50         printf("INA219 did NOT ACK at 0x40\n");
51     }
52     while (1) tight_loop_contents();
53
54 }

```

Figure 4: Firmware Setup for Current Reg Verification

Measurements

Looking at Figure 1 we see that the multimeter measured 4.65 V on the bus. Looking at Figure 3 we can see that the values stored in `Datarx_BusVoltage` is 0x244A. Converting this into a voltage reading as defined in the datasheet we get:

$$(0x244A \gg 3) * 4 = 4644 \text{ mV}$$

Looking at Figure 2 we see that the multimeter reads a system current of 85.8 mA. Looking at Figure 4 we see that `Datarx_Current` holds 0x0057 which is 87 mA.

Observations

The multimeter readings and the INA219 IC reading were within reason. The current reading is a little more off due to the fact that when the current reading was made in the software vs when I took the picture of the system current, the numbers had already changed on the multimeter.

Conclusions / Next Steps

Both INA219 Breakout boards have now passed the HIL test. My next step is to wrap up the control part of this project by now working on the gate drive circuit and PWM output from Raspberry Pi Pico.

Step by step plan for the next two weeks:

- Build the Gate Drive circuit and supply it with the regulated 5V output.
- Develop software on the Raspberry pi pcio to output a pwm signal at 100 KHz and be able to adjust the duty cycle.
- Hook up the gate drive circuit to the pico w pwm output. Use mulimeter to verify the voltage output is higher and changes with duty cycle than with just the pico (no gate drive).