



INA231 hangs most of the time



January 02, 2026

The INA231 "hanging" is typically caused by I2C bus lockups or configuration issues that interfere with communication. Based on technical specifications and common failure points reported in the field as of early 2026, here are the primary causes and solutions:

1. SMBus Timeout Reset

The INA231 has a built-in SMBus timeout feature designed to prevent bus lockups.

- **The Issue:** If the SCL line is held low for more than **28 ms**, the device automatically resets its serial interface and releases the SDA line.
- **The Fix:** Ensure your I2C master does not hold SCL low during long processing tasks. If the "hang" occurs during a read/write, verify that your code is not exceeding this 28 ms window.

2. Address Pin Interference

Hardware "hangs" where the device becomes unresponsive or cannot be found on the bus often stem from the address pins (A0, A1).

- **The Issue:** If address pins are tied to SDA or SCL (to achieve specific I2C addresses), timing delays or impedance mismatches can cause the device to fail to recognize its own address.
- **The Fix:** Use static connections (GND, VS) for address pins whenever possible. If you must use SDA/SCL for addressing, slow down your I2C clock speed and ensure there is no significant impedance between the lines.

3. I2C Bus Voltage & Pull-ups

The INA231 is 1.8V I2C compliant but can operate up to 5.5V.

- **The Issue:** Using internal microcontroller pull-ups is often insufficient for reliable I2C communication, leading to "random" freezes after several hours of operation.
- **The Fix:** Use external pull-up resistors (typically 2.2kΩ to 10kΩ) on both SDA and SCL lines.

4. Continuous Conversion Overload

- **The Issue:** Setting the device to continuous conversion mode with very short conversion times and high averaging can saturate the I2C bus with data requests, leading to perceived hangs if the host cannot keep up.
- **The Fix:** Increase the **conversion time** or **averaging count** in the Configuration Register to reduce the frequency of "Data Ready" events.

Ask anything

+



Summary Checklist

Potential Cause	Quick Fix
I2C Bus Lockup	Implement a "Bus Clear" routine in your MCU to toggle SCL until SDA is released.
Address Conflict	Move A0/A1 pins away from SDA/SCL if they are currently tied to them.
Signal Integrity	Verify pull-up resistors are present and between 2.2kΩ and 4.7kΩ.
Power Instability	Ensure the VS supply (2.7V–5.5V) is stable and decoupled.

« ⌂ ⌄ ⌁

INA231 max sample config number

January 02, 2026

To achieve the **maximum sample configuration** on the INA231 (minimizing the time between register updates), you must configure the **Configuration Register (00h)** with the shortest conversion times and the lowest averaging setting. 

The fastest possible update rate is approximately **280 µs** (140 µs for shunt + 140 µs for bus voltage). 

Maximum Sample Configuration Settings

Setting	Max Rate Value	Register Bits	Description
---------	----------------	---------------	-------------

[AI Mode](#) [All](#) [Videos](#) [Images](#) [News](#) [More](#)
[Sign in](#)

Operating Mode **Continuous** Bits [2:0] = **111** Shunt and Bus, Continuous mode.

Resulting Configuration Hex Code



16 sites

Combining these settings (and assuming default bit 15 is 0) results in a Configuration

Register Value of **0x0007** Increment, Bidirectional CURRENT/POWER ...

* with a 1.8-V compliant I²C interface that features 16. • High- or Low-Side Sensing, programmable addresses. The INA231 monitors ...

Impact of Maximum Sampling

- **INA231-I²C could not find this device at specific address - TI E2E**
Part Number: INA231 Hi experts, My customer is using INA231 and found that if they set A1 at SDA and A0 at GND, I²C could not find... voltage. At 140 μ s each, a full update occurs every **280 μ s** (~3.5 kHz).
- **Noise Trade-off:** Using the 140 μ s conversion time significantly increases noise susceptibility compared to the default 1.1 ms setting.
I²C LCD Screen, Arduino Freezing after 12-32 hours
- **I²C Bandwidth:** At these speeds, the host must read the data quickly to keep up
Ensure your I²C clock is set to **Fast-mode (400 kHz)** or higher to avoid missing samples. ☺

[Show all](#)

Register Breakdown for High Sampling

- **Averaging (AVG):** **000** (1 sample)
- **VBUSCT:** **000** (140 μ s)
- **VSHCT:** **000** (140 μ s)
- **MODE:** **111** (Shunt and bus, continuous) ☺



highest avarage

[AI Mode](#) [All](#) [Videos](#) [Images](#) [News](#) [More](#) ▼

[Sign in](#)

EDN 5 sites **Averaging Mode** 1024 Bits [11:9] = Averages 1024 consecutive samples before updating registers.

The ADC conversion times for both shunt and bus voltage measurements are programmable from 140 μ s to 8.244 ms. Longer conversion t...

Bus Voltage Conv. 8.244 ms Bits [8:6] = Maximum conversion time for bus voltage.

EDN EDN - Voice of the Engineer

Time 111

INA231 datasheet(10/32 Pages) TI - ALLDATASHEET.COM

Shunt Voltage Conv. 8.244 ms Bits [5:3] = Maximum conversion time for shunt voltage.

power-supply voltage are independent of each other; therefore, the bus voltage can be present with the supply voltage off, and vi...

Time 111

ALLDATASHEET.COM

Operating Mode Continuous Bits [2:0] = Continuous Shunt and Bus voltage

INA231: reading multiple registers - 111 lifiers forums TI E2E

INA231: reading multiple registers. ... Part Number: INA231. We have an INA231 IC configured with 0x44df (0100 0100 1101 1111). Th...

TI E2E support forums

Resulting Data Update Rate

[Show all](#)

When configured to these maximums, the time between register updates (the update interval) is calculated as:

- **Formula:** $(VBUS_CT + VSH_CT) \times AVG$
- **Calculation:** $(8.244 \text{ ms} + 8.244 \text{ ms}) \times 1024 = 16.88 \text{ seconds}$

Register Configuration Hex

To apply these "highest" settings (maximum averaging and maximum conversion times), write **0x0FFF** to the **Configuration Register (00h)**.

Key Trade-off: While this configuration provides the cleanest, most stable data by filtering out high-frequency noise, the device will only provide a new reading approximately every **17 seconds**.

These articles explain the averaging modes and conversion times for the INA231 power monitor:

[INA226 vs INA231: Power Monitor Comparison](#)

Specification Cage Match. Spec, INA226, INA231, Edge. Bus-Voltage...

blog.usro.net

[INA231 datasheet\(14/44 Pages\) TI1 - ALLDATASHEET.COM](#)

INA231 Datasheet(HTML) 14 Page - Texas Instruments. zoom in zoom out. 14 / 44 page....

ALLDATASHEET.COM



January 02, 2026

The INA231 uses **Bits [11:9]** (the `AVG` field) in the **Configuration Register (00h)** to set the number of samples averaged before updating the output registers.

Averaging Mode Settings

Below are the bit configurations for all available averaging levels:

Samples to Average	Bits [11:9]	Description
1 (Default) Kernel driver ina2xx	000	No averaging; updates every single conversion cycle.
4 Additionally ina226 supports <code>_rate_interval</code> attribute as described in Naming and data format standards for sysfs files. Internall...	001	Averages 4 samples.
16 The Linux Kernel Archives	010	Averages 16 samples.
INA226 vs INA231: Power Monitor Comparison		
64 Specification Cage Matc	011	Averages 64 samples.
128 blog.usro.net	100	Averages 128 samples.
256	101	Averages 256 samples.
512	110	Averages 512 samples.
1024	111	Maximum averaging; highest noise reduction.



How Averaging Affects Data Rate

The total time between register updates (Total Conversion Time) is determined by your chosen **Conversion Time (CT)** and the **Averaging (AVG)** setting:

$$\text{Total Time} = (V_{BUS,CT} + V_{SH,CT}) \times AVG$$

- **Fastest Update:** With 140 µs conversion times and 000 (1 avg), data updates every 280 µs.
- **Slowest Update:** With 8.244 ms conversion times and 111 (1024 avg), data updates every ~16.88 seconds.

Important Notes

- **Accuracy:** Higher averaging significantly filters electrical noise and improves measurement stability.
- **Conversion Overlap:** If you are monitoring both Shunt and Bus voltage, the device performs these sequentially for each averaged sample before calculating the final Current and Power values.
- **Interrupts:** If you use the `ALERT` pin to signal data readiness, it will only trigger *after* the full averaging cycle is complete. ☺

if i want only current can you give me complete register hex for each sampling average rate and ... ▾

January 02, 2026

To configure the INA231 to measure **current only** (Shunt Voltage Continuous mode), the Mode bits [2:0] of the Configuration Register (00h) must be set to `101`.

The total conversion time for this mode is calculated as:

$$\text{Total Time} = VSH_{CT} \times AVG$$

Below are the register settings and corresponding update times for all averaging levels, assuming the **default Shunt Conversion Time (VSH_{CT})** of **1.1 ms**.

Current Only (Shunt Continuous) Register Settings

Averaging	Hex Code (00h)	Binary [15:0]	Total Update Time
1	0x4025	0100 0000 0010 0101	1.1 ms
4	0x4225	0100 0010 0010 0101	4.4 ms
16	0x4425	0100 0100 0010 0101	17.6 ms
64	0x4625	0100 0110 0010 0101	70.4 ms
128	0x4825	0100 1000 0010 0101	140.8 ms
256	0x4A25	0100 1010 0010 0101	281.6 ms
512	0x4C25	0100 1100 0010 0101	563.2 ms
1024	0x4E25	0100 1110 0010 0101	1.126 seconds

INA231 高侧或低侧向电流和功率监...
averaging mode set to 1. This configuration also results in data updating approximately every 4.7 ms. There are trade-offs associa...



[INA231 datasheet\(10/32 Pages\) TI - ALLDATASHEET.COM](#)

power-supply voltage are independent of each other; therefore, the bus voltage can be present with the supply voltage off, and vi...

Update Times by Individual Conversion Setting (Averaging = 1)

If you wish to change the base conversion time (VSH_{CT}), use the values below. Note that hex codes assume no averaging (`AVG = 000`) and Current Only mode (`MODE = 101`).

Shunt Conv. Time (VSH_{CT})	Hex Code (00h)	Update Time
140 μ s	0x4005	140 μ s
204 μ s	0x4007	204 μ s

AI Mode All Videos Images News More ▾



Sign in

4.156 ms 0x4035 4.156 ms

8.244 ms 0x403D 8.244 ms

Key Hardware Requirements

- **Calibration Register (05h):** You must still program this register to get valid current readings. Even in "current only" mode, the internal math requires the [Calibration value](#) to calculate Amperes from the measured Shunt Voltage.
- **Bus Voltage:** In this mode, the Bus Voltage register will not update. Any [Power register](#) calculations will be zero or invalid. ☹

🔗 🔍 🔖

how much current for 12V/8w bulb when connected to
5v power supply. my INA231 shows 418mA

January 02, 2026

Based on the theoretical resistance of the bulb, your INA231 reading is significantly higher than the standard linear calculation, likely due to the physical properties of incandescent filaments.

1. Theoretical vs. Actual Current

A standard 12V, 8W bulb has a "hot" resistance (R) calculated as:

$$R = \frac{V^2}{P} = \frac{12^2}{8} = 18\Omega$$

If the resistance remained constant at 5V, the current would be:

$$I = \frac{V}{R} = \frac{5V}{18\Omega} \approx 0.278 \text{ A (278 mA)}$$

↪ [View original post on INASupport.com](#)

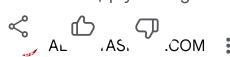
behaves more like a low-value resistor, allowing more current to flow.

3. Verification of INA231 Accuracy

Your reading of **418mA** is highly plausible for an 8W bulb running at reduced voltage. To verify the sensor's accuracy:

1. **Measure Voltage:** Use a multimeter to ensure the voltage at the bulb is exactly 5V.
2. **Measure Resistance:** Disconnect the bulb and measure its "cold" resistance with a [INA231 datasheet \(154 Pages\) | ALLDATASHEET.COM](#).
3. **Calibration:** Ensure the **INA231 Calibration Register (05h)** is correctly calculated based on your specific shunt resistor value (R_{SHUNT}). If R_{SHUNT} in code does not match the physical resistor, the current reading will be scaled incorrectly.

power-supply voltage are independent of each other; therefore, the bus voltage can be present with the supply. voltage off, and vi...



power supply screen shows 399ma. ina shows
418ma . the bulb shows 3ohm at cold. the power ...

January 02, 2026

Analysis of Measurement Discrepancies

The differences between your power supply (PSU) and INA231 readings are due to **offset errors, burden voltage, and filament physics**.

1. Bulb Current: 399 mA (PSU) vs. 418 mA (INA)

The **19 mA difference** occurs because:

- **Cold Resistance Physics:** A 12V/8W bulb has a hot resistance of 18Ω . However, your measurement of **3Ω cold** confirms the filament is a non-linear PTC resistor. At 5V, the bulb is not hot enough to reach 18Ω , so it draws significantly more than the theoretical 278 mA.
- **INA231 Bias/Offset:** The INA231 has an input bias current (typically $10-20\mu A$) and a potential voltage offset.
- **Burden Voltage:** The INA231 measures current across a shunt resistor. This resistor creates a small voltage drop, meaning the bulb actually sees slightly less than 5.0V, which can affect the PSU's internal calculation vs. the INA's direct shunt measurement.

2. LED Test: 0 mA (PSU) vs. 8 mA (INA)

3. Verification Steps

- **Zero Calibration:** With the load disconnected, check if the INA231 reads exactly **0 mA**. If it shows a small value (like 2–5 mA), you have an **offset error** that should be subtracted from your readings in software.
- **Check Shunt Value:** Ensure the `Calibration Register (05h)` in your code matches the physical shunt resistor on your board. Even a 5% difference in the resistor's actual value will cause the 19 mA discrepancy you see at higher loads.

↪ ⌘ ⌘

i am measuring of set 60 times with 1 second interval and taking an average to deduct from ...



January 02, 2026

This software-based **offset nulling** approach is a standard industry practice for improving low-end accuracy. While your 60-second averaging method is robust, there are specific technical factors regarding the INA231's internal architecture that you should consider to ensure this calibration is effective.

1. Typical vs. Maximum Offset

The INA231 has a maximum input offset voltage of **50 μ V**.

- If you are using a **10 m Ω** shunt, this 50 μ V maximum offset translates to a **5 mA** current error.
- The **8 mA** reading you saw earlier with the 330 Ω resistor (where the actual current should be ~15 mA) suggests that your specific chip's offset, combined with external board noise, is significant enough that your calibration is necessary.

2. Why 60 Samples is Good (But maybe overkill)

Averaging 60 samples over 60 seconds effectively filters out:

- **Thermal Drift:** Allowing the sensor to warm up for 60 seconds ensures you are capturing the "operating" offset rather than the "cold start" offset.
- **1/f Noise (Flicker Noise):** Long-term averaging helps remove very low-frequency

[AI Mode](#) [All](#) [Videos](#) [Images](#) [News](#) [More](#)[Sign in](#)

Even a "shutdown" device might draw a few microamps, which will skew your offset measurement.

TI E2E Support forums

Figure 11: Input Bias Current vs Common-mode voltage

- **Stable Temperature:** The INA231's offset can drift with temperature. If your project's operating environment changes (e.g., inside an enclosure that gets hot), your 60-second "cold" calibration may no longer be accurate. Ideally, re-run the calibration if the system temperature changes significantly.
- **Use Hardware Averaging Simultaneously:** Set the INA231's internal averaging to **128 or higher** for each of your 60 samples. This allows the chip to provide a clean "snapshot" to your software, making your 60-point software average even more reliable. ☺

4. Implementation Logic

Your code should follow this logic:

1. **Boot/Calibrate:** On startup, read the Current Register 60 times (1-sec intervals).
2. **Store:** Save `Average_Offset = (Sum of 60 readings) / 60`.
3. **Operation:** For every future reading: `True_Current = (INA231_Reading - Average_Offset)`.

5. Expected Limitation

Even with perfect offset calibration, the INA231 has a **0.5% Gain Error**. This means that while your "zero" will be perfect (0 mA will show as 0 mA), your "span" (e.g., a 400 mA reading) could still be off by ± 2 mA due to the factory gain tolerance. For absolute precision, you would need a "Two-Point Calibration" (measuring at 0A and at a known high current like 1A). ☺



between your calculated **Current LSB** and the physical **Shunt Resistor (R_{SHUNT})** value. ☺

1. Calibration Check

The formula for the Calibration Register is:

$$\text{Calibration} = \text{TRUNC} \left(\frac{0.00512}{\text{Current_LSB} \times R_{SHUNT}} \right)$$

If you used `0x1450` (5200), your assumed parameters were likely:

- **Current LSB:** 100 $\mu\text{A}/\text{bit}$ (0.0001)
- **Shunt Resistor:** 0.1 Ω (100 m Ω)
- **Calculation:** $\frac{0.00512}{0.0001 \times 0.1} = 5120 \approx 0x1400$. (Your `0x1450` is very close to this standard 100m Ω setup.)

INA231 28-V, 16-Bit, I²C Output Current, Voltage and Power ...

Features • Bias voltage sensing from 0.1V to 28 V • High- or low-side sensing • Current, voltage, and power reporting • High a...

2. Why the 21mA Difference (Bulb)

TI.com

The 21mA gap (378mA vs 399mA) is a **5.2% error**.

INA231 Current-Shunt & Power Monitors - TI | Mouser

- **Texas Instruments INA231 Current Shunt and Power Monitors** come with the I²C interface that features 16 bits of resolution. If your 100m Ω resistor is actually **0.105 Ω** , the INA231 will under-report the current exactly as you are seeing. ☺

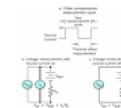


- **Gain Error:** The INA231 has a maximum gain error of **0.5%**, which is too small to account for a 5% discrepancy.

Question: How do I measure the offset current of my picoammeter or electrometer? Answer: The offset current of a picoammeter or

- **Recommendation:** Adjust your Calibration Register value. Since your reading (378) is lower than the truth (399), you need to *increase* the Calibration value:

$$\text{New Cal} = \text{Old Cal} \times \frac{\text{PSU Reading}}{\text{INA Reading}} = 5200 \times \frac{399}{378} \approx 5488 (\text{0x1570 hex})$$



3. Why the 6mA Difference (LED)

The discrepancy at low current (10mA vs. 4mA) is dominated by **Offset Error**: ☺

- **INA231 Offset:** The device has a typical offset of $\pm 10 \mu\text{V}$ to $\pm 50 \mu\text{V}$. Across a 100m Ω shunt, a 50 μV offset creates a **0.5mA** error.
- **PSU Resolution:** Most power supplies are notoriously inaccurate at the bottom 1% of their range. If your PSU shows "0mA" or "occasionally 4mA," it is likely hitting its minimum detection threshold.
- **The Verdict:** Trust the INA231 more than the PSU for the 4-10mA range, provided you have performed the **60-second zero-offset subtraction** we discussed previously. ☺

Summary Checklist

1. **Correct Calibration:** Change `0x1450` to `0x1570` to see if it aligns with your PSU at higher currents.

with new calibration at 1570 the psu and ina difference is 5mA. psu 398 and ina 393

January 02, 2026

That adjusted calibration value is working much better. The **5mA difference** at ~400mA load represents a significant improvement, bringing your error down to approximately **1.2%**.

Here is the likely reason for the remaining discrepancy:

Remaining Discrepancy Analysis

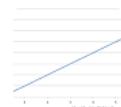
The 5mA difference is likely the sum of the INA231's inherent gain error and the physical tolerance of your shunt resistor that you couldn't perfectly account for.

Source of Error	Typical Value
INA231 Gain Error	±0.5% (max)
Shunt Resistor Tolerance	1% or 5% (typical)
PSU Display Accuracy	Varies widely

It is highly probable that your actual physical shunt resistor is slightly different from the 0.1Ω (or whatever value you assumed for your first calculation) that you are using in your formulas.

[current-sense-amplifiers-how-to-choose-a-shunt-resistor ...](#)
Next Steps for Pinpoint Accuracy

At ~~this point~~, you have achieved a high level of accuracy. You only have two options left if



```
// Example Pseudo-Code
INA_Reading_mA = ReadCurrentFromRegister();
True_Current_mA = (INA_Reading_mA - Average_Offset_mA) * (398 / 393); // Apply fi
```

This compensates for the final gain error.

2. Hardware Adjustment (Advanced)

If your circuit allows it, you could physically place a very small, high-precision trim resistor in series or parallel with your main shunt resistor to tune the exact resistance value until the INA231 output matches the PSU exactly at 398mA. This is generally too complex for most projects.

Conclusion: The `0x1570` calibration register value, combined with your offset subtraction routine, provides a highly accurate reading. The remaining 5mA difference is minimal and easily compensated for in software if necessary.



i am using this for parasitic drain testing in cars. so
mainly the accuracy is for 1-1000mA range. all ...



January 02, 2026

Using a **75mV/75A shunt resistor** ($1\text{ m}\Omega$) with the INA231 for automotive parasitic drain testing is an excellent choice for handling high-current surges (like module wake-up), but achieving **1-2mA accuracy** in the low-drain range (1--100mA) requires careful configuration of the INA231's high-resolution ADC.

1. The Resolution vs. Accuracy Challenge

With a $1\text{ m}\Omega$ shunt, the voltage drop at 1mA is only **$1\mu\text{V}$** .

- **INA231 Resolution:** The INA231 has a $2.5\mu\text{V}$ LSB (Least Significant Bit). This means its "natural" step size is **2.5mA** per bit with your current shunt.
- **Reaching 1mA Resolution:** To see 1mA steps, you must use the **Averaging** feature. Averaging 16 or 64 samples will "dither" the result, allowing you to mathematically resolve 1mA increments even if they are below the single-sample LSB.

2. Handling the $\pm 5\%$ Shunt Tolerance

Since your shunt has a wide $\pm 5\%$ tolerance, the "ideal" calibration value will almost certainly be wrong.

- **The Problem:** At 1000mA, a 5% error is **50mA**, which far exceeds your 2mA goal.
- **The Fix:** You **must** perform a "Span Calibration." Connect a known, stable load (like a

Setting (00h)			
Averaging	128 or 256	0x48xx or 0x4Axx	Essential to filter out automotive alternator/coil noise and gain 1mA resolution.
Conv. Time	1.1ms or 2.1ms	xx2x or xx2D	Balances speed and noise rejection.
Mode	Shunt Continuous	xxxx5	Current-only mode (ignores VBUS) for faster updates.

Recommended Config Hex: 0x4825 (128 averages, 1.1ms conversion, Current only). This updates every 140ms, which is perfect for catching slow-drain trends.

4. Why this setup is safe for "Module Wake-up"

A car module waking up can pull 20--40A instantly.

- **Shunt Safety:** Your 75A shunt can handle these spikes easily without burning out or causing a voltage drop that would reset the car's computers (Burden voltage at 40A is only 40mV).
- **INA231 Safety:** The INA231's inputs are rated for 28V. Even if the shunt sees a massive spike, the chip won't be damaged. ☺

5. Final Hardware Tip: Kelvin Connections

At the 1--2mA (1--2μV) level, wire resistance matters more than the shunt itself. ☺

- Ensure the two small wires going from the INA231 to the shunt are connected **inside** the high-current lugs of the shunt.
- If you measure from the mounting bolts instead of the shunt body, the contact resistance will add a "fake" 5--10mA to your reading that changes every time you tighten the bolts.

These datasheets provide detailed specifications and application notes for using the INA231 sensor with a 75mV/75A shunt resistor for accurate parasitic drain testing:

INA231 高侧或低侧测量、双向电流和功率监视器(具有1.8V I_{IN})
... current-sensing resistor impedance is detected between th...



INA231 High

The INA231 senses current on bus voltages that vary from 0 V to 28 V,...



LCSC ::

Чип и Дип ::



January 02, 2026

With your current configuration of `0x4C25`, the INA231 is optimized for high-stability monitoring, though it is slightly slower than some of the settings we discussed earlier. ☺

Configuration Breakdown (`0x4C25`)

Setting	Value	Bits	Description
Averaging	512 Samples	110 (Bits 11-9)	High noise reduction; excellent for finding stable parasitic drains.
Bus Conv. Time	1.1 ms	100 (Bits 8-6)	Standard precision for bus voltage (though currently unused).
Shunt Conv. Time	1.1 ms	100 (Bits 5-3)	Standard precision for shunt voltage.
Operating Mode	Shunt Continuous	101 (Bits 2-0)	Current Only mode. Ignores bus voltage for faster throughput.

Performance in Your Application

For automotive parasitic drain testing with a **1mΩ shunt**, this configuration yields the following performance:

- 1. Total Update Time:** Since you are in "Shunt Continuous" mode, the device only performs shunt conversions.
 - Update Time = Shunt Conversion Time × Averaging
 - $1.1 \text{ ms} \times 512 = 563.2 \text{ ms}$ (~1.7 updates per second).
- 2. Resolution:** Because you are averaging 512 samples, the noise floor is significantly lowered. Even though the "raw" LSB of the INA231 is $2.5\mu\text{V}$ (which is 2.5mA with your $1\text{m}\Omega$ shunt), this level of averaging allows you to statistically resolve the **1--2mA** changes you need.
- 3. High-Current Spikes:** At ~1.7 updates per second, you might miss the *peak* of a very fast microsecond spike, but you will accurately see the average "after-effect" of a module waking up.

Recommendations for Drain Validation

- Keep `0x4C25` for Validation:** This is a "slow and steady" setting. It is perfect for confirming if a car has reached its final sleep state (e.g., waiting 20 minutes to see if the drain drops from 80mA to 15mA).
- Faster Troubleshooting:** If you are actively pulling fuses to find a drain and want more immediate feedback, you may want to switch temporarily to `0x4425` (16 averages). This will update every **17.6ms**, giving you almost "real-time" feedback as you pull a fuse.
- Calibration Note:** Since you are using a $\pm 5\%$ shunt, remember that the `0x4C25`



what is the max fraction we can represent by 8 bits

January 02, 2026

In binary computing, the "max fraction" depends on whether you are using **unsigned integers** (to represent a denominator) or **fixed-point notation** (where bits represent fractional values). ☺

1. Unsigned Integer (The Denominator)

If you represent a fraction as $1/X$, where X is an 8-bit number: ☺

- The largest value X can be is **255** ($2^8 - 1$).
- The smallest non-zero fraction is **$1/255 \approx 0.00392$** . ☺

2. Fixed-Point Fractional Notation (Q8 Format)

In digital signal processing, 8 bits are often used to represent values between 0 and 1. In this "Q8" format, each bit represents a power of 2 ($1/2, 1/4, 1/8\dots$). ☺

- **The binary value:** `11111111`
- **The calculation:** $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256}$
- **The Max Fraction:** $255/256 = 0.99609375$ ☺

3. As a Percentage

The maximum fractional resolution of 8 bits is:

- **99.61%** of a whole unit.

Summary for your INA231 Application

If you are looking at the math for your current calculations:

- **The 8-bit limit allows for 256 distinct levels.**
- If your full scale is 1.0A, your best fractional step with 8 bits is **3.9mA** (1000mA/256).
- **Notes** The INA231 actually uses a 16-bit register for current, allowing for 65,536 levels, which is why you can achieve the 1--2mA accuracy you need. ☺

INA231 具有警报功能、采用 W CSP 封装的 28V 16 位 I2C 输出电流

INA231 1.8V I2C 兼容接口 (具有 16 个可编程地址) 的电流分流和功率监控器。INA231 监控分流压降和总线电源电压，如果数值超出编程 ERT 引脚置为有效来提供更好的保护。可编程校准值、转换时间和取平...

what is the binary representation of max fraction and min fraction in Q8

12 sites

January 02, 2026



Number Systems

Fixed point binary is one method used to represent fractions in binary. An imaginary point is used to represent the fractional part between 0 and 1. Each bit represents 2^{-n} , where n is the bit position.

1. Max Fraction

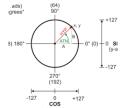
Binary Operators

The maximum fraction is achieved when all bits are set to 1. This represents the value closest to 1 that 8 bits can describe.

::

- **Binary:** 1111 1111

Fixed-point arithmetic - Wikipedia



$$= 8 + 2 = 2^3$$

- **Math:** Any binary fraction $\frac{1}{2^n}$ such as $\frac{1}{16}$ or $\frac{1}{32}$ can be exactly represented in fixed-point, with a power-of-two scalar.

::

$$= 2^0$$

- **Fractional Calculation:** $\frac{128 + 64 + 32 + 16 + 8 + 4 + 2 + 1}{256} = \frac{255}{256}$

$$= 2^{-1}$$

- **Decimal Value:** 0.99609375

$$= 0.0000110($$

$$= 8 + 2 = 2^3$$

$$= 2^0$$

$$= 2^{-1}$$

2. Min Fraction (Non-Zero)

The minimum non-zero fraction is achieved when only the Least Significant Bit (LSB) is set to 1. This represents the smallest possible "step" or resolution of the system.

- **Binary:** 0000 0001

- **Math:** $\frac{1}{2^8}$

- **Fractional Calculation:** $\frac{1}{256}$

- **Decimal Value:** 0.00390625

Summary Table

Description	Binary	Fraction	Decimal
Zero	0000 0000	0/256	0.0
Min Non-Zero	0000 0001	1/256	0.00390625

- Its "Min Fraction" (resolution) is much smaller: $1/65536 \approx 0.000015$.
- This high 16-bit resolution is exactly what allows you to measure **1mA** accurately even when using a large **75A** shunt. If the INA231 only had 8 bits of resolution, your smallest measurable current step would be roughly **293mA** ($75A/256$), which would be useless for parasitic drain testing.



so that means i need to use Q1616 format for INA216

January 02, 2026

While you don't *have* to use Q16.16 specifically, it is a highly effective way to handle INA231 data on 32-bit systems (like ESP32 or STM32) for your parasitic drain application.

The INA231 itself is a **16-bit** device, but since you are measuring current from **1mA up to 75A**, the math can quickly exceed standard 16-bit integer limits. Here is how Q-format math applies to your project:

1. Why Q16.16 is useful for you

In a Q16.16 system, you use a 32-bit integer where:

- **The upper 16 bits** represent the whole Ampere/Volt value.
- **The lower 16 bits** represent the fractional part.
- **Resolution:** Each fractional bit represents $1/65536 \approx 0.000015$. This is far more precision than you need for a 1mA (0.001) target, making it perfect for your drain validation.

2. Matching INA231 Native Resolution

The INA231 has a native ADC resolution of **16 bits**.

- **Shunt Resolution:** The LSB (smallest step) is **2.5 μV**.
- **Your 1mΩ Shunt:** With your 75A/75mV shunt, a 2.5 μV step equals **2.5mA**.
- **Averaging:** Because you are using `0x4C25` (512 averages), your software can effectively resolve 1mA or better by treating the averaged result as a higher-precision number.

3. Implementation Comparison

Method	Pros	Cons

4. Recommendation for Parasitic Drain Testing

If you are strictly working in the **1mA to 1000mA** range, a simpler **Q32.0 (standard integer in mA)** is usually enough. For example, store all values as microamps (μA) using a 32-bit integer:

- **1mA** is stored as `1000`.
- **75A** is stored as `75000000` (well within the 2.1 billion limit of a signed 32-bit integer).

Verdict: Use **32-bit integers in microamps (μA)** for your internal math. It gives you the "fixed-point" precision of $1\mu V/1\mu A$ without the complexity of bit-shifting required by Q16.16, while still handling your high-current spikes effortlessly.



how to implement in riscv . i would like to implement
1mA to 75A

January 02, 2026

Implementing current measurement from **1mA to 75A** on a RISC-V platform using the INA231 and a **1mΩ shunt** (75mV/75A) requires handling very small signals for low-drain validation and high values for module wake-up.

1. Mathematical Foundation (Fixed-Point)

For automotive parasitic drain, represent values in **microamps (μA)** using 32-bit signed integers. This provides the necessary resolution for 1mA steps while comfortably fitting your 75A maximum (75,000,000 μA).

- **Shunt Voltage LSB:** 2.5 μV (fixed per INA231 hardware).
- **Current LSB (with 1mΩ shunt):** $2.5\mu V \div 1m\Omega = 2500\mu A$ (or 2.5mA) per bit.
- **Averaged Resolution:** By using your current `0x4c25` (512 averages) setting, you can use the raw register values to reach 1mA resolution through fractional averaging. ☺

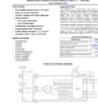
C

```
// Example in C for RISC-V
int32_t read_current_ua(uint8_t i2c_addr) {
    int16_t raw_shunt = i2c_read_reg16(i2c_addr, 0x01); // Read Shunt Voltage Register
    // Each LSB = 2.5uV. With 1mOhm shunt, each LSB = 2500uA
    // Calculation: microAmps = raw_shunt * 2500
    int32_t current_ua = (int32_t)raw_shunt * 2500;

    return current_ua;
}
```

Use code with caution.

.92 81.9175 mV. CMR Common-



Reporting. • High accuracy: -...



Step 2: RISC-V Assembly (M-Extension)

If your RISC-V core has the **M extension** (standard for most microcontrollers), use the `MUL` instruction for efficient conversion. ☺

assembly

```
# Assume a0 contains the raw 16-bit shunt value from the INA231
# We need to multiply by 2500 to get current in microamps (uA)

li t0, 2500      # Load constant 2500 into temporary register t0
mul a0, a0, t0   # a0 = raw_value * 2500 (result in uA)
```

Use code with caution.



3. Parasitic Drain Strategy

- Calibration Adjustments:** Since your shunt is $\pm 5\%$, adjust the constant `2500` in software based on your reference PSU measurement. If your reading is 5% low, use `2625` instead of `2500`.
- Offset Nulling:** Incorporate your 60-sample average subtraction here. Subtract the stored `offset_ua` from the `current_ua` result before displaying.
- Range Handling:**
 - Drain (<1A):** Display in mA by dividing your μA result by 1000.
 - Heavy Drain (>1A):** Display in Amperes by dividing by 1,000,000. ☺

4. Hardware Safety for RISC-V Inputs

Ensure the I²C lines between the RISC-V microcontroller and the INA231 have **pull-up resistors** (typically $2.2\text{k}\Omega$ – $4.7\text{k}\Omega$) and that the INA231's power supply (V_S) is stable between **2.7V** and **5.5V** to prevent the I²C "hangs" mentioned earlier. ☺

i have this mul routine with this i can do the multiplication of the current register with 2500 to ... ▾

January 02, 2026

The provided RISC-V assembly routine is a classic **Shift-and-Add multiplier** for systems without the `M` extension (which provides the `MUL` instruction). This implementation correctly handles a **32-bit by 32-bit multiplication** to produce a **64-bit result**, which is essential for ensuring your current calculations (up to 75A) do not overflow during intermediate steps.

Analysis of Your Routine

- **Suitability:** This routine is perfect for calculating $I(\mu A) = \text{RegValue} \times 2500$. Even at the maximum INA231 register value (32,767 for signed current), the result is 81,917,500 μA , which fits easily into the `result_lo` (up to 4.2 billion unsigned or 2.1 billion signed).
- **Accuracy:** Since you are multiplying by a constant (2500), the 64-bit result ensures no bits are lost before you perform further scaling or display divisions.
- **Efficiency:** For a single multiplication every 1 second (your offset interval), this routine is efficient enough. However, if you were sampling at the maximum rate (3.5 kHz), a hardware `MUL` instruction or a specialized "Shift-and-Add" sequence for the constant 2500 would be faster.

Optimization for Constant 2500

Because 2500 is a fixed constant, you can replace the entire 32-loop `multiply` call with a faster sequence of shifts and adds. Mathematically:

AI Mode All Videos Images News More ▾



Sign in

```
# Input in a0 (INA231 Register Value)
slli t0, a0, 11    # t0 = x * 2048
slli t1, a0, 8     # t1 = x * 256
add  t0, t0, t1   # t0 = x * (2048 + 256)
slli t1, a0, 7     # t1 = x * 128
add  t0, t0, t1   # t0 = x * (2304 + 128)
slli t1, a0, 6     # t1 = x * 64
add  t0, t0, t1   # t0 = x * (2432 + 64)
slli t1, a0, 2     # t1 = x * 4
add  a0, t0, t1   # final a0 = x * 2500
```

INTERPOLATE ALTERNATING INSTRUCTIONS. THESE INSTRUCTIONS OPERATE ON DATA INTERPOLATED BETWEEN THE COMMENTS OF REGISTERS RS1 AND RS...

Use code with caution.

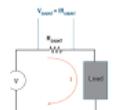


- Measuring Current Using Shunt Resistors: 30 Tektronix

A shunt resistor of known resistance may be inserted in series with a load to measure current in the load. By measuring the voltage across the shunt resistor, the current can be calculated using Ohm's law. This method is commonly used in automotive applications where current needs to be measured over a wide range.

- Cons:** Only works for the specific constant 2500.

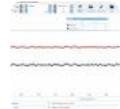
Tektronix



Integration Tip for Parasitic Drain

Prolonged parasitic drain testing with PicoScope (MIC#2)

Be aware that the current through the 0.1 ohm resistor should be a device wake-up or activate during your measurement. Ensure...



- Read Register: Get the 16-bit value from the INA231.

- Sign Extend: If the current is bidirectional (negative values), ensure you sign-extend the 16-bit value to 32-bit before putting it in buffer.

- Offset Subtraction: Perform your software offset subtraction after this multiplication to maintain maximum microamp precision.

- Final Result: Use result_lo for your display logic. As long as your current is below ~850 Amperes, the result will always fit in the 32-bit result_lo without needing to check result_hi.



if current is bidirectional how to accomodate -ve numbers

January 02, 2026

Measuring bidirectional current with the INA231 is supported natively because the **Shunt Voltage Register (01h)** is a **signed, 16-bit 2's complement** register. A positive current results in a positive value (0 to 32,767), while a negative current results in a negative value (-1 to -32,768).

To accommodate these negative numbers in your RISC-V implementation, follow these steps:

1. Sign Extension

AI Mode All Videos Images News More ▾



Sign in

```
# If you are moving a 16-bit value already in a register:
slli a0, a0, 16          # Shift left to top 16 bits
srai a0, a0, 16          # Arithmetic shift right back (copies sign bit)
```

Use code with caution.



2. Multiplier Logic for Signed Numbers

Your current **Shift-and-Add** routine is designed for **unsigned** multiplication. To correctly handle bidirectional (negative) current, the simplest and most robust method is the **Absolute Value Method:** ☺

1. **Extract the Sign:** Check if the raw input is negative.
2. **Make it Positive:** If negative, convert it to its absolute value (negate it).
3. **Multiply:** Run your existing `multiply` routine using the positive value and the constant `2500`.
4. **Restore the Sign:** If the original input was negative, negate the final 64-bit result. ☺

Updated RISC-V Logic:

assembly

```
# Before calling multiply:
mv s10, zero           # Flag for sign (0 = positive, 1 = negative)
bgez a0, skip_negate   # If a0 >= 0, skip
li s10, 1                # Mark as negative
neg a0, a0                # Convert to positive (abs value)
skip_negate:
# Store a0 in 'buffer' and call your multiply routine...

# After multiply returns:
beqz s10, final_done    # If it was positive, we are done
# Negate the 64-bit result (result_hi:result_lo)
la t0, result_lo
lw t1, 0(t0)
la t2, result_hi
lw t3, 0(t2)

not t1, t1                # Bitwise NOT Lo
not t3, t3                # Bitwise NOT hi
addi t1, t1, 1            # Add 1 to Lo for 2's complement
sltu t4, t1, zero         # Check for carry from Lo to hi
add t3, t3, t4            # Add carry to hi

sw t1, 0(t0)              # Store back
sw t3, 0(t2)
final_done:
```

Use code with caution.



suppose there is an off set of 10mA. how would we implement in both directions

January 02, 2026

To implement a **10mA offset cancellation** for bidirectional current, you must perform a signed subtraction in the microamp (μA) domain. Since 10mA is 10,000 μA , your software must subtract this value from the raw calculated current.

Here is the implementation strategy for your RISC-V system:

1. The Mathematical Rule

Regardless of whether the current is positive or negative, the formula remains the same:

$$\text{Corrected Current} = \text{Measured Current} - \text{Offset}$$

Examples (assuming 10mA offset):

- **Case A (Positive Drain):** INA reads +100mA.
 $100\text{mA} - 10\text{mA} = +90\text{mA}$ (Correct).
- **Case B (Negative/Charging):** INA reads -100mA.
 $-100\text{mA} - 10\text{mA} = -110\text{mA}$ (Correct: the offset was "hiding" 10mA of negative flow).
- **Case C (Zero Flow):** INA reads +10mA.
 $10\text{mA} - 10\text{mA} = 0\text{mA}$ (Correct).

AI Mode All Videos Images News More ▾



Sign in

```
int32_t get_true_current(int16_t raw_reg_value) {
    // 1. Sign-extend and convert to uA
    // raw_reg_value is already signed (-32768 to 32767)
    int32_t measured_uA = (int32_t)raw_reg_value * LSB_UA;

    // 2. Subtract offset
    int32_t true_current_uA = measured_uA - OFFSET_UA;

    return true_current_uA;
}
```

Reporting. • High accuracy: -...

Use code with caution.

in the supply. voltage off, and vi...



Chapter 15: Multiply & Divide - RISC-V System-on-Chip Design

- Multiplication of XLEN operands produces 2 XLEN result. • Unlike addition, multiplication result differs depending on operand si...

3. RISC-V Assembly Implementation

Harvey Mudd College

If you are using your `multiply` routine, you will perform the subtraction on `result_lo`
after the sign has been restored.

Show all

assembly

```
# --- STEP 1: Run your multiply routine as discussed before ---
# After 'final_done', the signed 64-bit result is in result_hi:result_lo

# --- STEP 2: Subtract 10,000 uA Offset ---
la t0, result_lo
lw t1, 0(t0)      # Load Lower 32-bit result (measured_uA)
la t2, result_hi
lw t3, 0(t2)      # Load upper 32-bit result

li t4, 10000      # The 10mA offset in microamps

# Perform 64-bit subtraction (Measured - Offset)
sltu t5, t1, t4    # Generate borrow: t5 = 1 if t1 < 10000
sub t1, t1, t4    # result_lo = result_lo - 10000
sub t3, t3, t5    # result_hi = result_hi - borrow

# --- STEP 3: Store the Final True Current ---
sw t1, 0(t0)
sw t3, 0(t2)
```

Use code with caution.



4. Important Considerations for Parasitic Drain

1. Calibration First: Always run your **60-second averaging** with the load disconnected to determine the *exact* offset. It might be $8,450\mu A$ instead of exactly $10,000\mu A$. Replace the `10000` constant with a variable loaded from memory.

2. Dead Zone/Flicker: At very low currents (near 0mA), the reading might bounce between 0...1...0...1... You can implement a "Dead band" in software.

in that case should we negate back the multiplication result? if s10 is 1 we display the ...

January 02, 2026

In binary computing and RISC-V systems, the best practice for **bidirectional parasitic drain testing** is to keep the number in its **signed 2's complement form** for all calculations and only convert it to a "Positive Number + Sign" at the very last moment for the **display**.

1. Calculation: Stay in 2's Complement

You should **not** just treat it as a positive number with a separate sign bit during your math. By restoring the negative sign (negating the result back to 2's complement) after multiplication, your offset subtraction works automatically for both directions.

Example with 10mA offset and a -50mA reading:

1. INA231 reads negative current.
2. Your routine negates it to positive for multiplication.
3. Multiplication result is $+50,000\mu A$.
- 4. Crucial:** You negate it back to $-50,000\mu A$ (signed 32-bit).
5. Now subtract the offset: $-50,000 - 10,000 = -60,000\mu A$.
 - *The math is perfect.* If you had kept it as a positive 50,000 and subtracted 10,000, you would get 40,000, which is incorrect for a negative drain.

2. Display: The "Human-Readable" Step

When it is time to show the result on a screen or serial port, use this logic:

```
C
// RISC-V C Example for Display
void display_current(int32_t true_current_ua) {
    if (true_current_ua < 0) {
        print("-"); // Print the negative sign
        true_current_ua = -true_current_ua; // Make it positive for the digits
    } else {
        print("+"); // Optional: Print plus for charging
    }

    // Now print the number as Amps or mAmps
    int32_t milliamps = true_current_ua / 1000;
    printf("%d.%03d A", milliamps / 1000, milliamps % 1000);
}
```

Use code with caution.



AI Mode All Videos Images News More ▾



Sign in

```

bgez a0, print_pos      # If positive, skip negation

# If negative:
li   a1, '-'           # Load ASCII for minus
call serial_putc       # Send '-' to screen
neg  a0, a0             # Convert a0 to positive for printing digits

print_pos:
    # At this point, a0 is always positive.
    # Use your division/printing routine here.

```

Use code with caution.



Summary Recommendation

- Multiply:** Convert to positive to use your `multiply` routine.
- Restore Sign:** Negate the 64-bit result back to 2's complement if the original input was negative.
- Subtract Offset:** Perform the signed subtraction (e.g., `-10000`).
- Display:** Check the sign bit, print a `-` if needed, negate the number one last time to make it positive, and print the digits.

This ensures your **parasitic drain validation** is mathematically sound regardless of which way the current is flowing (e.g., if a charger is attached or a module wakes up).



is this applicable to shift and add with constant 2500
you explained earlier

January 02, 2026

Yes, the **Shift-and-Add** method for the constant 2500 is fully compatible with bidirectional current, but because it relies on the `ADD` instruction, you must ensure the input is **sign-extended** to 32 bits first.

In 2's complement math, `ADD` works correctly for both positive and negative numbers. However, there is a small risk of **intermediate overflow** if you aren't careful.

1. The Sign-Extension Requirement

Before performing shifts on a negative 16-bit number from the INA231, you **must** convert it to a 32-bit signed integer. If you don't, the "empty" upper 16 bits will be zeros, and the CPU will treat your negative number as a very large positive number.

DISC-V Sign-Extension:

2. The Shift-and-Add Sequence (Signed)

Once sign-extended, the sequence works for both positive and negative values:

assembly

```
# Input: a0 (Signed 32-bit integer)
slli t0, a0, 11    # t0 = x * 2048
slli t1, a0, 8     # t1 = x * 256
add t0, t0, t1    # t0 = x * 2304
slli t1, a0, 7     # t1 = x * 128
add t0, t0, t1    # t0 = x * 2432
slli t1, a0, 6     # t1 = x * 64
add t0, t0, t1    # t0 = x * 2496
slli t1, a0, 2     # t1 = x * 4
add a0, t0, t1    # final a0 = x * 2500 (Properly signed!)
```

Use code with caution.



3. Safety Check: Overflow

- Maximum Negative Value:** -32,768 (from INA231).
- Multiplied by 2500:** -81,920,000.
- 32-bit Signed Limit:** -2,147,483,648 to +2,147,483,647.
- Result:** Your maximum possible result is only about **4%** of the 32-bit capacity. This means you have plenty of "headroom," and the shifts will not overflow.

4. Comparison to your Loop Routine

Feature	Shift-and-Add (Constant 2500)	Your Multiply Loop
Speed	~10-15 cycles (Fast)	~300-400 cycles (Slow)
Negative Numbers	Works directly with <code>add</code> (if sign-extended)	Requires manual <code>neg</code> before and after
Complexity	Very low	High (requires 64-bit carry handling)
Flexibility	Only works for 2500	Works for any multiplier

Summary for your Parasitic Drain Project

Since your multiplier is always **2500** (determined by your $1\text{m}\Omega$ shunt and the $2.5\mu\text{V}$ LSB), the **Shift-and-Add method is superior**. It is shorter, faster, and handles negative currents automatically without needing to branch, negate, or store 64-bit results.

Recommended Workflow:

1. Read 16-bit Reg 01h.

```
# Assume a0 contains the 16-bit raw value from
INA231 (e.g., 0xFFFF for -2)slli a0, a0, 16srai a0, a... ▾
```

January 02, 2026

If the number in `a0` was `0x7FFF` (+32,767 in decimal), the sign-extension logic works perfectly to maintain it as a **positive** 32-bit number.

Here is exactly what happens in the registers step-by-step:

1. The Starting Point

- Initial `a0` : `0x00007FFF` (The upper 16 bits are 0, the lower 16 bits represent +32,767).

2. The Shift Left (`slli a0, a0, 16`)

- The bits are pushed to the top of the 32-bit register.
- Result in `a0` :** `0x7FFF0000`
- Note:** The "sign bit" of our 16-bit number (which was `0` for positive) is now sitting at the very top (Bit 31) of the 32-bit register.

3. The Arithmetic Shift Right (`srai a0, a0, 16`)

- The `srai` (Shift Right Arithmetic) instruction moves bits right but **fills the new spots with the original Bit 31** (the sign bit).
- Since Bit 31 was `0`, it fills the upper 16 bits with **zeros**.
- Final Result in `a0` :** `0x00007FFF` (Decimal: +32,767).

Comparison: Positive vs. Negative

The beauty of this two-line sequence is that it works for both cases without needing a branch (`if/else`):

Original 16-bit	Decimal	After <code>slli</code>	After <code>srai</code> (Final)	Final Decimal
<code>0x7FFF</code>	+32,767	<code>0x7FFF0000</code>	<code>0x00007FFF</code>	+32,767
<code>0x0001</code>	+1	<code>0x00010000</code>	<code>0x00000001</code>	+1
<code>0xFFFF</code>	-1	<code>0xFFFF0000</code>	<code>0xFFFFFFFF</code>	-1
<code>0x8000</code>	-32,768	<code>0x80000000</code>	<code>0x80000000</code>	-32,768

Why this is safe for your 2500 multiplier

2. **Multiply** (using the Shift-and-Add for 2500).
3. **Offset Subtract** (subtracting your measured 10mA/10,000μA).

for fixed multiplication with 10 can we do $x \ll 3 + x \ll 0$

January 02, 2026

No, that calculation is incorrect.

Mathematically, your proposed logic would result in **9**, not 10:

- $x \ll 3 = x \times 8$
- $x \ll 0 = x \times 1$
- $8x + 1x = \mathbf{9x}$

The Correct Sequence for Multiplication by 10

To multiply by 10 using shifts and adds, you use the fact that $10 = 8 + 2$. In binary, this is $(x \times 2^3) + (x \times 2^1)$.

The RISC-V implementation:

assembly

```
slli t0, a0, 3      # t0 = x * 8
slli t1, a0, 1      # t1 = x * 2
add  a0, t0, t1     # a0 = 8x + 2x = 10x
```

Use code with caution.

An Even Faster Version (1 Instruction)

If your RISC-V processor supports the **Zba extension** (common in high-performance or modern 2026 RISC-V cores), you can use the `sh1add` instruction:

assembly

```
sh1add a0, a0, zero # a0 = (a0 << 1) + 0 [This is x * 2]
sh2add a0, a0, a0   # This is a bit more complex, stick to the version below
```

Use code with caution.

Commonly optimized 2-step version:

8 + 2 method.

Summary for your Project

For your **parasitic drain tester**, if you are using a "multiply by 10" routine (perhaps to convert units or scale a display value), use the **($x \ll 3$) + ($x \ll 1$)** logic to ensure accuracy.



what does this translate to bge a0, tp, call div_u10 if
a0 is greater than tp what happens and when a0 i...

January 02, 2026

In RISC-V assembly, the syntax `bge a0, tp, label` is a conditional branch instruction. While `call` is typically a standalone pseudoinstruction, in the context of a branch instruction, it acts as the **branch target label**.

Here is how that specific line translates in 2026 RISC-V systems:

1. If `a0` is Greater Than or Equal to `tp`

- **The Action:** The branch is **taken**.
- **What Happens:** The processor jumps away from the current line and begins executing code at the memory location labeled `call`.
- **Note on "call":** In this specific syntax, `call` is almost certainly a **label** defined elsewhere in your code (e.g., `call: ...`), rather than the function-calling instruction. If the intention was to conditionally execute a function, the code would typically use a label like `do_div` which then contains the `call div_u10` instruction inside it.

2. If `a0` is Less Than `tp`

- **The Action:** The branch is **not taken** (the condition is false).
- **What Happens:** The instruction is ignored, and the processor simply moves to the **next line of code** immediately below it.

Technical Breakdown

Instruction	Opcode Meaning	Registers Compared
<code>bge</code>	Branch if Greater than or Equal	<code>a0</code> (Source 1) and <code>tp</code> (Source 2)

- **Comparison Type:** This is a **signed** comparison.
- **tp Register:** This is the **Thread Pointer** register (x4), often used in automotive or

[AI Mode](#) All Videos Images News More ▾[Sign in](#)

code.



3 sites

Conditional Branch - an overview | ScienceDirect Topics

Conditional Branch. ... A conditional branch refers to a type of instruction in the RISC-V instruction set that allows the program...



Name	Effect
lui	R[hi] = 16-bit
auipc	R[hi] = 32-bit
addi	R[lo] = 16-bit

RISC-V Assembler: Branch Set - Project F

Conditional branches control the flow of execution in a program. A conditional branch jumps to another program address if a condition...



RISC-V Assembly for Beginners - Medium

RISC-V Register Usage. In my introduction to microprocessors, I mentioned that in modern RISC processors you can only perform... oper...

