SCD40 sensor CO2 reading zero

SCD40 sensor reports CO2 of 0

SCD40 CO2 Reading of 0

A CO2 reading of exactly `0` from the SCD40 is almost always a fault condition, not a real measurement. Here are the likely causes and fixes:

---

1. Sensor Not Yet Initialized / Measurement Not Started

The SCD40 requires you to explicitly start measurements before data is valid.

```python
scd40.start_periodic_measurement()
time.sleep(5)  # Wait at least 5 seconds for first measurement
```

Reading data before calling `start_periodic_measurement()` returns 0 by default.

---

2. Data Ready Flag Not Checked

Reading before data is ready returns stale/zero values. Always poll the ready flag:

```python
if scd40.data_ready:
    co2, temp, humidity = scd40.measurement
    print(f"CO2: {co2} ppm")
```

---

3. I²C Communication Issue

A `0` can result from a failed I²C read that gets silently ignored. Check:

- **Wiring**: SDA, SCL, VDD (2.7–5.5V), GND
- **Pull-ups**: I²C lines need pull-up resistors (typically 10kΩ)
- **Address**: SCD40 default I²C address is `0x62`
- **Scan the bus** to confirm the sensor is visible:

```python
import board, busio
i2c = busio.I2C(board.SCL, board.SDA)
i2c.try_lock()
print(hex(addr) for addr in i2c.scan())  # Should include 0x62
```

---

4. Power Issue

The SCD40 can draw up to **175mA** during measurement. Insufficient power causes erratic or zero readings. Use a dedicated 3.3V or 5V rail, not a GPIO pin.

---

5. Sensor in Wrong Mode

If you called `stop_periodic_measurement()` or are in single-shot mode without triggering a measurement:

```python
scd40.stop_periodic_measurement()  # required before single-shot
time.sleep(0.5)
scd40.measure_single_shot()
time.sleep(5)  # SCD40 needs ~5s
co2, temp, humidity = scd40.measurement
```

---

6. Sensor Needs Reset / Factory Reset

If the sensor was previously misconfigured:

```python
scd40.stop_periodic_measurement()
time.sleep(0.5)
scd40.reinit()  # soft reset
time.sleep(0.5)
scd40.start_periodic_measurement()
```

Quick Diagnostic Checklist

Check	Expected
I²C address visible	`0x62`
`start_periodic_measurement()` called	✅
Wait ≥ 5s before first read	✅
`data_ready` is `True` before reading	✅
VDD voltage	2.7–5.5V
Current supply	≥ 175mA available

What platform/library are you using (CircuitPython, Arduino, ESP-IDF, etc.)? I can give more targeted help.

I don't think any of those ideas is the solution. I am successfully communicating with the SCD40, using Sensirion's 'exampleUsage' program. It reports the S/N of the SCD40, and it returns correct temperature & RH values. Only the CO2 values are zeroes.

That's a very specific failure mode — serial number readable, temp/RH correct, CO2 = 0. That points to something specific about the CO2 channel itself.

Most Likely Cause: Sensor Hasn't Completed Its First CO2 Measurement

The SCD40's CO2 channel (NDIR optical) has a **longer warm-up** than the temp/RH sensor (which is almost instant). Even after `start_periodic_measurement()`, the first few CO2 readings can be `0` or invalid while the optical path stabilizes.

How long have you been waiting? The SCD40 typically needs:

- ~5 seconds for the first measurement cycle
- But up to ~**30–60 seconds** in some conditions before the NDIR CO2 reading is stable and non-zero

Try polling continuously for at least 30 seconds and watching if CO2 ever becomes non-zero.

Second Likely Cause: ASC (Automatic Self-Calibration) Interference

If the sensor was previously used with ASC and has a corrupted baseline, CO2 can read 0. Try **disabling ASC** and doing a forced recalibration:

```c
scd4x_set_automatic_self_calibration(0);  // disable ASC
```

Third Possibility: Sensor is Defective (CO2 Cell Failed)

The SCD40 uses an NDIR (infrared) optical cell for CO2. It's physically separate from the temp/RH element. A failed IR source or detector would give exactly this symptom — everything else works, CO2 is always 0.

Test: Does CO2 stay at exactly `0` forever, even after several minutes? If yes, and you've ruled out timing, the NDIR cell is likely dead.

What does the output look like — is CO2 *always* `0` on every reading, or just the first few?

I am using a Teensy 4.1 running under the Teensyduino version of the Arduino IDE.

Good context. A few Teensy/Arduino-specific angles to investigate:

Most Likely: I²C Clock Stretching

The SCD40 uses **clock stretching** during CO2 measurement (the NDIR measurement takes time). The Teensy 4.1's I²C peripheral has known issues with clock stretching tolerance.

- Temp/RH returns quickly → no stretching needed → works fine
- CO2 measurement → requires clock stretching → Teensy may time out and return 0 silently

Try slowing down the I²C clock:

```cpp
Wire.begin();
Wire.setClock(10000);  // Try 10kHz instead of default 100kHz
```

This gives the sensor more time to respond before the Teensy gives up.

Check Which Wire Bus You're On

Teensy 4.1 has three I²C buses. Make sure Sensirion's library is using the same `Wire` instance your sensor is wired to (`Wire`, `Wire1`, or `Wire2`).

Sensirion Arduino Library Version

Which version of the Sensirion library are you using? There were bugs in earlier versions where CO2 was read from the wrong byte offset in the response packet — temp/RH would be fine, CO2 zero.

Check in Arduino IDE → **Sketch → Include Library → Manage Libraries** → search `Sensirion I2C SCD4x`. Make sure you're on the latest (1.0.0+).

The clock stretching issue is my strongest suspicion given your exact symptom pattern. Does slowing I²C change anything?

It's been running for several minutes. The clock slowing didn't help. The library version is 1.1.0. Disabling the automatic self calibration didn't help.

Given that CO2 is *always* exactly 0 after minutes of running, with everything else working, let's go deeper.

Check the Raw I²C Response

The SCD40 returns a 9-byte packet for each measurement:

- Bytes 0-1: CO2 (uint16)
- Byte 2: CRC for CO2
- Bytes 3-4: Temperature
- Byte 5: CRC for temp
- Bytes 6-7: Humidity
- Byte 8: CRC for RH

If CO2 bytes are genuinely `0x00 0x00` with a valid CRC of `0x81`, the library accepts it as a legitimate zero reading. **Can you add a raw Wire read** to dump all 9 bytes and their CRCs? That tells us whether the sensor is actually sending zeros or the library is misreading.

Self-Test

The SCD40 has a built-in self-test command. Add this before starting measurements:

```cpp
scd4x.stopPeriodicMeasurement();
delay(500);

uint16_t testResult;
scd4x.performSelfTest(testResult);
Serial.print("Self-test result: 0x");
Serial.println(testResult, HEX);
```

- `0x0000` = passed
- Any other value = sensor fault detected

This directly tells you if the NDIR cell has failed.

Physical Inspection

With the sensor in hand — is there anything blocking the CO2 intake vents on the SCD40 package? The NDIR cell needs air exchange through the openings on the sensor body. If it was mounted flush against a PCB or covered with conformal coating, CO2 can't reach the optical cavity.

What does the self-test return?

> Your Check Which Wire Bus You're On idea seems silly.
> Remember, I am successfully getting the S/N and the temp & RH.

You're right, that was a redundant suggestion — sorry. If the S/N, temp, and RH are all coming through on the same bus, the bus itself is clearly fine.

The self-test result is the most useful next step. What does `performSelfTest()` return?

Self-test result 0. This SCD40 was working a few days ago. It was, and is, mounted on a little Adafruit PCB. I doubt that anything is blocking its air passages, but I blasted it with a little compressed air to dislodge what might be there. How do I do a raw Wire read?

Good thinking on the compressed air. Here's how to do a raw Wire read to see exactly what bytes the sensor is sending:

cpp

```cpp
#include <Wire.h>

#define SCD40_ADDR 0x62
#define CMD_READ_MEASUREMENT 0xEC05

void readRawMeasurement() {
    // Send the read measurement command
    Wire.beginTransmission(SCD40_ADDR);
    Wire.write(0xEC);
    Wire.write(0x05);
    Wire.endTransmission();

    delay(1);  // Small delay before reading

    // Request 9 bytes
    Wire.requestFrom(SCD40_ADDR, 9);

    Serial.print("Raw bytes: ");
    uint8_t data[9];
```

```
    for (int i = 0; i < 9; i++) {
        data[i] = Wire.read();
        Serial.print("0x");
        if (data[i] < 0x10) Serial.print("0");
        Serial.print(data[i], HEX);
        Serial.print(" ");
    }
    Serial.println();

    // Decode CO2 manually
    uint16_t co2_raw = (data[0] << 8) | data[1];
    uint8_t co2_crc = data[2];
    Serial.print("CO2 raw word: 0x");
    Serial.println(co2_raw, HEX);
    Serial.print("CO2 CRC byte: 0x");
    Serial.println(co2_crc, HEX);
}
```

Call this **after** `start_periodic_measurement()` and at least 5 seconds of waiting, inside your loop.

The key things to look for:

- Are bytes 0-1 actually `0x00 0x00`?
- Is the CRC byte (byte 2) `0x81`? (that's the valid CRC for `0x0000`)
- Or is the CRC something else, suggesting corrupted data?

If CO2 bytes are `0x00 0x00` with CRC `0x81`, the sensor itself is genuinely reporting zero — pointing to a failed NDIR cell. If the CRC doesn't match, something is going wrong in transmission.

I put the call to readRawMeasurement into the loop, but left the code that tries to display the CO2/Temp/RH. The output is then

Looks like the output got cut off — you started to paste it but it didn't come through. Please share what the serial monitor showed!

```
Self-test result: 0x0
serial number: 0x7532DF073B09
Raw bytes: 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
CO2 raw word: 0xFFFF
```

Show more

`0xFF` on every byte is very telling — that's what you get when `Wire.requestFrom()` returns no data and `Wire.read()` returns 0xFF (the default for an empty buffer).

The problem is **timing in your raw read function**. The `readRawMeasurement()` function is sending the read command and immediately trying to read back — but it's racing with or interfering with the Sensirion library's own communication. When both try to talk to the sensor in the same loop iteration, the raw read gets nothing.

Try adding a **data ready check** before issuing the raw read command:

cpp

```cpp
void readRawMeasurement() {
    // First check data ready (command 0xE4B8)
    Wire.beginTransmission(SCD40_ADDR);
    Wire.write(0xE4);
    Wire.write(0xB8);
    Wire.endTransmission();
    delay(1);

    Wire.requestFrom(SCD40_ADDR, 3);
    uint8_t dr0 = Wire.read();
    uint8_t dr1 = Wire.read();
    uint8_t dr2 = Wire.read();  // CRC
    Serial.print("Data ready bytes: 0x");
    Serial.print(dr0, HEX); Serial.print(" 0x");
    Serial.print(dr1, HEX); Serial.print(" 0x");
    Serial.println(dr2, HEX);

    bool ready = (dr1 & 0x07FF) != 0;  // lower 11 bits of word
    Serial.print("Data ready: ");
```

```
    Serial.println(ready ? "YES" : "NO");

    if (!ready) return;

    // Now read measurement (command 0xEC05)
    Wire.beginTransmission(SCD40_ADDR);
    Wire.write(0xEC);
    Wire.write(0x05);
    Wire.endTransmission();
    delay(1);

    Wire.requestFrom(SCD40_ADDR, 9);
    // ... rest of read as before
}
```

What does the data ready response show?

```
Self-test result: 0x0
serial number: 0x7532DF073B09
Raw bytes: 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
CO2 raw word: 0xFFFF
CO2 CRC byte: 0xFF
Raw bytes: 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
CO2 raw word: 0xFFFF
CO2 CRC byte: 0xFF
```

Still all `0xFF` — so even the data ready command is getting no response. Yet the Sensirion library successfully reads temp/RH from the same sensor on the same bus.

The difference must be in **how the Sensirion library issues the read command** versus your raw code. Specifically, the SCD40 requires a **stop condition followed by a delay** between sending the command and clocking out the response.

Look at how the Sensirion library does it — it likely uses `endTransmission(true)` and a specific delay. Try increasing your delay between send and read:

```cpp
Wire.beginTransmission(SCD40_ADDR);
Wire.write(0xEC);
Wire.write(0x05);
Wire.endTransmission(true);  // explicit stop condition
delay(100);                   // try a longer delay
Wire.requestFrom(SCD40_ADDR, 9);
```

But more importantly — **look at the Sensirion library source** to see exactly how it sends the read measurement command. In Arduino IDE:

- Find the library folder (usually `Arduino/libraries/Sensirion_I2C_SCD4x/`)
- Open `SensirionI2CScd4x.cpp`
- Find the `readMeasurement` function
- Check what delay it uses between `endTransmission` and `requestFrom`

What does the library source show there?

Ignore much of my last comment. I miscoded readRawMeasurement. Now that it's correct, the output is

The output got cut off again — please paste what the serial monitor shows!

```
Self-test result: 0x0
serial number: 0x7532DF073B09
Raw bytes: 0x00 0x00 0x81 0x68 0x69 0xD0 0x5B 0x64 0x03
CO2 raw word: 0x0
```

Show more

This is now very clear. The sensor is genuinely sending `0x00 0x00` with a valid CRC of `0x81` for CO2 — the library is reading it correctly. The temp/RH bytes are changing normally across readings. **The NDIR CO2 cell itself is outputting zero.**

This confirms a **hardware failure of the CO2 measurement cell**. The sensor is not broken in a way that affects I²C communication or the temp/RH subsystem — just the NDIR optical path.

Since you said it was working a few days ago, possible causes:

- **Electrostatic discharge** — the NDIR detector is sensitive to ESD
- **Overvoltage spike** on VDD, even briefly
- **Condensation** inside the optical cavity if it was exposed to high humidity
- Simply **early failure** — it happens, especially with NDIR sensors

At this point the sensor needs to be replaced. The good news is the Adafruit SCD40 breakout is inexpensive (~$15). Since you have the breakout board already, just swap the sensor itself if you're comfortable with soldering, or replace the whole breakout.

Before giving up on it — do you have any way to test it at a **higher CO2 concentration**, like breathing directly on it? A truly failed cell would still read 0. If it's something like a contaminated optical path, a high enough concentration might show a flicker above zero.

```
Self-test result: 0x0
serial number: 0x7532DF073B09
Raw bytes: 0x00 0x00 0x81 0x68 0x69 0xD0 0x5B 0x64 0x03
CO2 raw word: 0x0
```

Show more

Same conclusion — the sensor is definitively sending `0x0000` with valid CRC `0x81` for CO2 every single time, while temp/RH vary normally. The hardware CO2 cell has failed.

One last thing worth trying before replacing it — a **factory reset**, which clears all persisted calibration data including the CO2 baseline:

```cpp
scd4x.stopPeriodicMeasurement();
delay(500);
scd4x.performFactoryReset();
delay(1200);  // datasheet specifies 1200ms after factory reset
scd4x.startPeriodicMeasurement();
```

A corrupted stored baseline could theoretically cause this, and factory reset is the only way to rule it out. The self-test passing with 0x0 makes this unlikely, but it costs nothing to try.

If that doesn't change anything, the sensor needs replacing. Given it was working a few days ago with no obvious physical damage, I'd contact Adafruit support — they're generally good about warranty replacements for sensors that fail early without obvious misuse.

The Heavy Breathing test had the expect results on temp & RH, but the CO2 stayed 0. I'm sure you're right; I've got a dead sensor. Thanks.

You're welcome. The heavy breathing test was the perfect final confirmation — temp and RH spiked just as expected, and CO2 stayed flatlined. The NDIR cell is definitely dead.

Good luck with the replacement, and at least you now have a solid raw diagnostic routine in your toolkit for next time.

I've forgotten how to save a transcript of this conversation. I remember that it's simple.

You can save it a few ways from claude.ai:

- **Copy the conversation** — there may be a copy button in the menu
- **Print to PDF** — use your browser's File → Print → Save as PDF

But honestly the simplest way is just to use your browser: **File → Save Page As** and save it as a complete webpage or PDF.

If you remember it being even simpler than that, it might be the **share** button — look for a share/export icon near the top of the conversation. That may have changed since you last used it though. Do any of those ring a bell?