

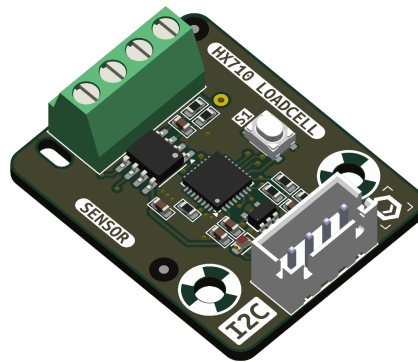
► PDF Export Style (Hidden on web)

VNEHC I2C Loadcell (HX710) Module User Manual - For Beginners

The **MKE-S18 HX710 I2C Loadcell Sensor** consists of a loadcell and a signal processing module using the **HX710**, a high-precision 24-bit ADC. The board integrates a **32-bit ARM Cortex-M0+ microcontroller** that handles reading, noise filtering, and processing data from the HX710, providing the final weight data directly via **I2C**.

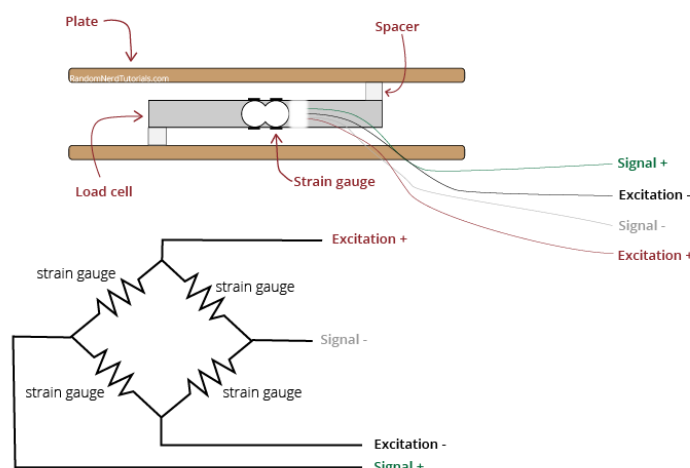
Thanks to this design, main microcontrollers like **Arduino, ESP32, Raspberry Pi, or STM32** can easily read loadcell data using just two pins (**SDA** and **SCL**), freeing up processing resources. The module supports a wide communication voltage range of **3.3V - 5VDC**, making it safely compatible with most popular development boards.

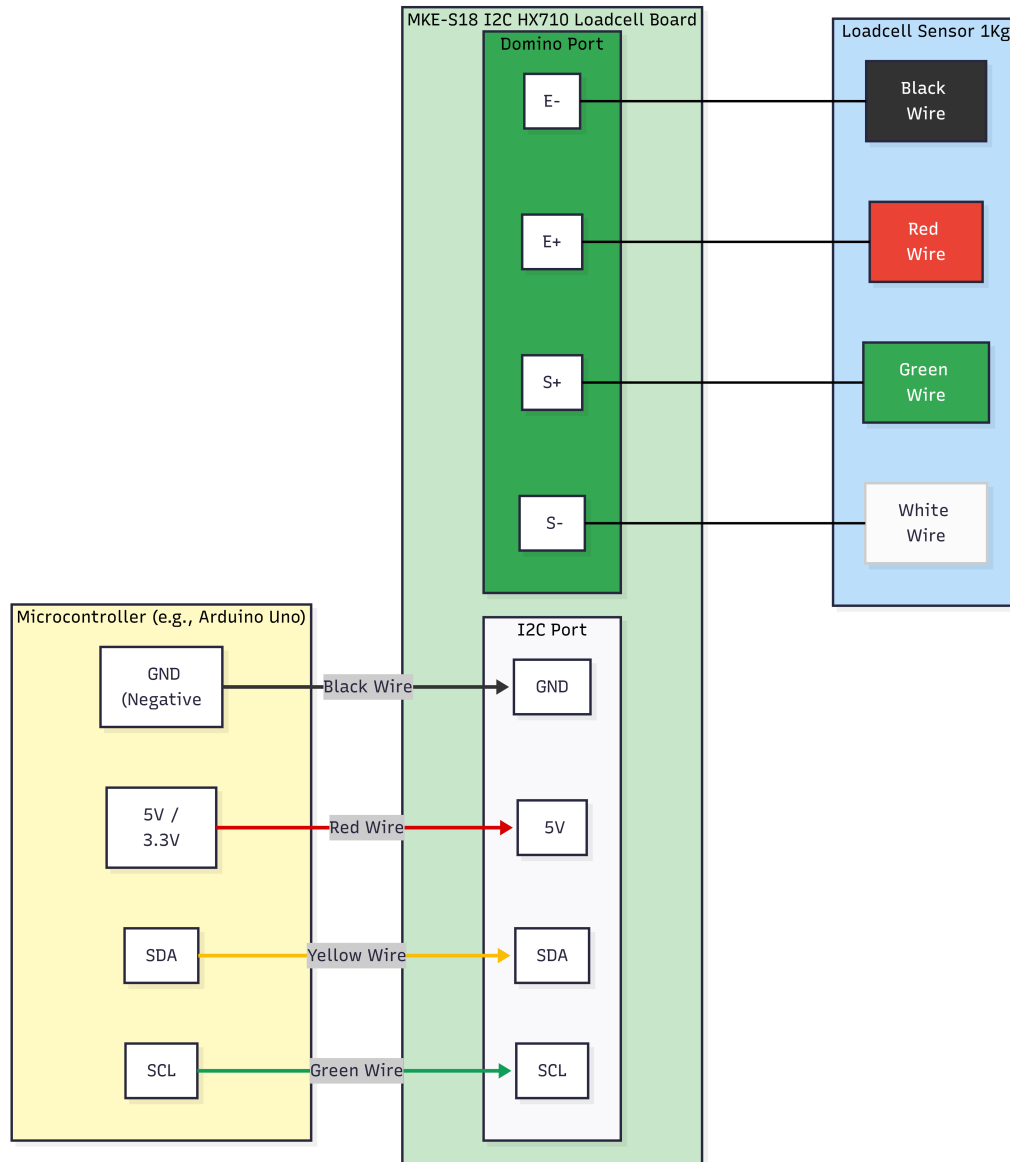
This guide covers how to operate the on-board physical button and quickly program the module with Arduino using the **MKE_I2C_Loadcell** library.



Part 1: Pinout & Hardware Connection

The module communicates with microcontrollers via the I2C protocol (4 wires) and connects to a Loadcell using a 4-pin (or 5-pin) terminal block.





1. Connecting to Microcontroller (I2C Port)

- **GND:** Connect to GND on Arduino/Microcontroller.
- **5V:** Connect to 5V (or 3.3V) on Arduino/Microcontroller.
- **SDA:** Connect to SDA (I2C Data).
- **SCL:** Connect to SCL (I2C Clock).

2. Connecting to Loadcell (Terminal Block)

- **E+ (Red):** Positive power supply for Loadcell (Excitation +).
- **E- (Black):** Negative power supply for Loadcell (Excitation -).
- **S+ (Green):** Positive signal from Loadcell (Signal +).
- **S- (White):** Negative signal from Loadcell (Signal -).

Note: Wire colors may vary depending on the Loadcell manufacturer, please check the datasheet of your specific Loadcell.

Part 2: Basic On-Board Operations

The Loadcell module is equipped with a small built-in physical button (**S1**) for quick operations without writing any code:

1. Tare (Zeroing):

- **Action:** Single click the button.
- **Effect:** The module sets the current weight to **0 grams**. (There is a 1-second delay after clicking to prevent mechanical vibration noise caused by your finger).

2. Factory Reset:

- **Action:** Press and hold the button for more than 3 seconds.
 - **Effect:** Erases all saved configurations and resets the I2C address back to the default **0x0A**.
-

Part 3: Programming with Arduino

Instead of sending complex raw I2C bytes, you just need to install the library and use its pre-written functions.

Step 0: Install the Library

The **MKE_I2C_Loadcell** library is automatically managed through the MakerEdu core library suite called **MKE_ONE**. Installation is very simple:

1. Open the Arduino IDE.
2. Go to **Sketch > Include Library > Manage Libraries...** (or press **Ctrl + Shift + I**).
3. Search for **MKE_ONE**.
4. Find **MKE_ONE** (by MakerEdu.vn) and click **Install** (select *Install All* if asked to install dependencies).

Once completed, the **MKE_I2C_Loadcell** library will be ready to use.

Step 1: Include Library

First, include the library and create a Loadcell object:

```
#include "MKE_I2C_Loadcell.h"

// Create a variable to represent the Loadcell module
MKE_I2C_Loadcell scale;
```

Step 2: Initialize (Setup)

In the **setup()** function, start the I2C bus and call **begin()**:

```

void setup() {
  Serial.begin(115200);
  Wire.begin(); // Mandatory I2C initialization

  // Start communication with the Loadcell module
  if (!scale.begin()) {
    Serial.println(F("Loadcell not found! Please check connections."));
    while(1) delay(100); // Stop the program if failed
  }

  Serial.println(F("Loadcell connected successfully!"));

  // You can call tare() once here to zero the scale on startup
  // scale.tare();
}

```

Step 3: Read Data (Loop)

To read the current weight in Grams, simply call `getGram()`:

```

void loop() {
  // Get weight value (grams)
  float weightGram = scale.getGram();

  Serial.print(F("Weight: "));
  Serial.print(weightGram, 1);
  Serial.println(F(" g"));

  delay(500);
}

```

Other Useful Functions

- `scale.tare()`: Call this to software Tare (zero the scale).
- `scale.setFilterLevel(2)`: Adjust sensitivity/smoothness. `0`: Extremely fast response but fluctuating values, `3`: Very stable reading and anti-vibration but slower response. Default is `2`.
- `scale.getKilogram()`: Get the weight in Kilograms (Kg).

When to Calibrate?

When first using the module or after mounting it to a new mechanical frame, the Gram reading might be incorrect. You only need to calibrate it once. The module will automatically calculate the Scale factor and **permanently save it to the EEPROM**. You do not need to run this code again on subsequent reboots!

Quick Calibration Code Example (e.g., using a 500g weight):

```
#include <Wire.h>
#include "MKE_I2C_Loadcell.h"

MKE_I2C_Loadcell scale;

// Customize your reference weight (Unit: Grams)
const float REFERENCE_WEIGHT_GRAM = 500.0;

void setup() {
  Serial.begin(115200);
  Wire.begin();

  if (!scale.begin()) {
    Serial.println(F("Loadcell not found! Please check connections."));
    while (1) delay(100);
  }
  Serial.println(F("Loadcell connected successfully!"));

  Serial.println(F("\n--- INSTRUCTIONS ---"));
  Serial.println(F("1. Remove all items from the scale, send 't' to Tare (Zero)"));
  Serial.println(F("2. Place a calibration weight (e.g., 500g) on the scale, send 'c' to Calibrate"));
  Serial.println(F("-----\n"));
}

void loop() {
  if (Serial.available()) {
    char cmd = Serial.read();

    // TARE (ZERO)
    if (cmd == 't' || cmd == 'T') {
      Serial.println(F("\n[CMD] Taring..."));
      scale.tare();
      Serial.println(F("-> Tare successful. Weight reset to 0.));
    }
    // CALIBRATE
    else if (cmd == 'c' || cmd == 'C') {
      Serial.print(F("\n[CMD] Calibrating with reference weight: "));
      Serial.print(REFERENCE_WEIGHT_GRAM);
      Serial.println(F("g ..."));

      scale.calibrate(REFERENCE_WEIGHT_GRAM);
      delay(500); // Wait for the module to save to EEPROM

      Serial.print(F("-> Calibration successful! New Scale Factor: "));
      Serial.println(scale.getScale(), 6);
    }
  }

  // Print weight continuously
```

```
float weight = scale.getGram();  
Serial.print(F("Current Weight: "));  
Serial.print(weight, 1);  
Serial.println(F(" g"));  
  
delay(300);  
}
```

Using Multiple Loadcells Simultaneously

To connect multiple I2C Loadcell modules to the same SDA/SCL bus, each module must have a unique I2C address (default is `0x0A`).

- **Step 1:** Connect only 1 Loadcell module to your Arduino and open the `07_Change_I2C_Address` example sketch to assign a new address (e.g., `0x0B`).
- **Step 2:** Repeat for other modules if necessary (`0x0C`, `0x0D`...).
- **Step 3:** Write your code to declare instances and call the `begin` function using the corresponding address:

Example Code for Reading 2 Loadcells:

```
#include <Wire.h>
#include "MKE_I2C_Loadcell.h"

// Declare 2 instances for 2 Loadcell modules
MKE_I2C_Loadcell scale1; // Default address is 0x0A
MKE_I2C_Loadcell scale2; // We will use 0x0B

void setup() {
    Serial.begin(115200);
    Wire.begin();

    Serial.println(F("Initializing Loadcells..."));

    // Initialize scale 1 (Default address 0x0A)
    if (!scale1.begin(0x0A)) {
        Serial.println(F("Loadcell 1 (0x0A) not found!"));
    } else {
        Serial.println(F("Loadcell 1 (0x0A) connected successfully!"));
    }

    // Initialize scale 2 (Address changed to 0x0B)
    if (!scale2.begin(0x0B)) {
        Serial.println(F("Loadcell 2 (0x0B) not found! Did you change its address?"));
    } else {
        Serial.println(F("Loadcell 2 (0x0B) connected successfully!"));
    }
}

void loop() {
    // Read weight from both scales
    float weight1 = scale1.getGram();
    float weight2 = scale2.getGram();

    // Print to Serial Monitor
    Serial.print(F("Scale 1 (0x0A): "));
    Serial.print(weight1, 1);
    Serial.print(F(" g \t| "));

    Serial.print(F("Scale 2 (0x0B): "));
    Serial.print(weight2, 1);
    Serial.println(F(" g"));

    delay(500);
}
```

Part 4: Basic I2C Protocol (For Other Platforms)

If you are not using Arduino (e.g., STM32, Raspberry Pi, ESP-IDF) and want to write your own driver, you can communicate directly via I2C. The default I2C address of the module is **0x0A**.

I2C Transaction Sequence (Write & Read)

Every command (whether reading or writing) must follow this strict 3-step sequence to allow the module enough time to process data:

1. **Send Request (Write 6 bytes):** The I2C Master writes a 6-byte packet to the Loadcell module:
 - **Byte 1:** The I2C address of the module (e.g., **0x0A**).
 - **Byte 2:** The **Mode ID** command (See table below).
 - **Byte 3 to 6:** 32-bit Payload (transmitted as Big Endian - highest byte first). If the command doesn't require sending data (read-only), fill these 4 bytes with **0x00**.
2. **Wait for Processing:** The Master must delay for at least **5ms**.
3. **Read Result (Read 4 bytes):** The Master requests 4 bytes from the Loadcell module. The returned data is always 32-bit (Big Endian).
 - *Note: For the Get Weight command (0x34), the 4 bytes returned represent a standard IEEE 754 32-bit Float. You must cast this byte array back to a Float type in your code.*
 - **C/C++ Code Example to convert 4 bytes to Float:**

```
// Assuming 4 bytes received from I2C (Big Endian) are stored in a
buffer
uint8_t buffer[4] = {0x43, 0x7A, 0x00, 0x00}; // Represents 250.0

// Step 1: Combine 4 bytes into a 32-bit integer
uint32_t raw_uint32 = ((uint32_t)buffer[0] << 24) |
                      ((uint32_t)buffer[1] << 16) |
                      ((uint32_t)buffer[2] << 8)  |
                      ((uint32_t)buffer[3]);

// Step 2: Cast the memory to Float (using memcpy is the safest
method)
float weight_gram;
memcpy(&weight_gram, &raw_uint32, sizeof(float));

// Result: weight_gram = 250.0
```

Basic I2C Commands (Extended)

Mode ID (Dec)	Mode ID (Hex)	Function	Payload / Return
10	0x0A	Get I2C Address	Returns uint32_t
2	0x02	Get Module ID	Returns uint32_t
4	0x04	Get Firmware Version	Returns uint32_t
50	0x32	Get Raw ADC Value	Returns int32_t
51	0x33	Tare (Zeroing) command	Write 0x00 (4 bytes)
52	0x34	Get Weight (Grams)	Returns 32-bit Float
53	0x35	Calibrate	Write weight 32-bit Float (Saves to EEPROM)
54	0x36	Get Scale Factor	Returns 32-bit Float
55	0x37	Set Scale Factor	Write 32-bit Float (Saves to EEPROM)
58	0x3A	Get Filter Level	Returns 0 to 3
59	0x3B	Set Filter Level	Write 0 to 3 (4 bytes) (Saves to EEPROM)