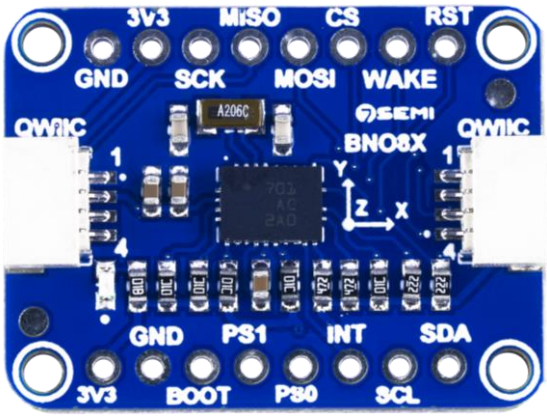
7SEMI

BNO085 9-DOF Absolute Orientation Sensor Breakout I2C Qwiic

Version 2.0

Table of Contents

1.Introduction



The **7Semi BNO085 9-DOF Orientation IMU Fusion Breakout Qwiic** is an advanced sensor module integrating a **triaxial accelerometer, gyroscope, and magnetometer**, making it ideal for **motion tracking, orientation sensing, and VR/AR applications**. The sensor runs **CEVA's SH-2 firmware**, which provides **real-time sensor fusion**, ensuring accurate orientation and motion analysis.

1.1 Features

- 9 Degrees of Freedom (9-DOF): Includes a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer.
- Sensor Fusion Algorithm: Provides accurate 3D orientation, heading, and angular velocity calculations.
- Multiple Output Formats:
  - Quaternion-based rotation vectors
  - Euler angles (roll, pitch, yaw)
  - Gravity vector and linear acceleration
  - Angular velocity
- Qwiic-Compatible: Seamless integration with Qwiic ecosystem via I2C interface.
- Low Power Consumption: Optimized for battery-powered applications.
- Firmware Upgrade Support: Allows DFU (Device Firmware Update) over I2C, SPI, or UART.

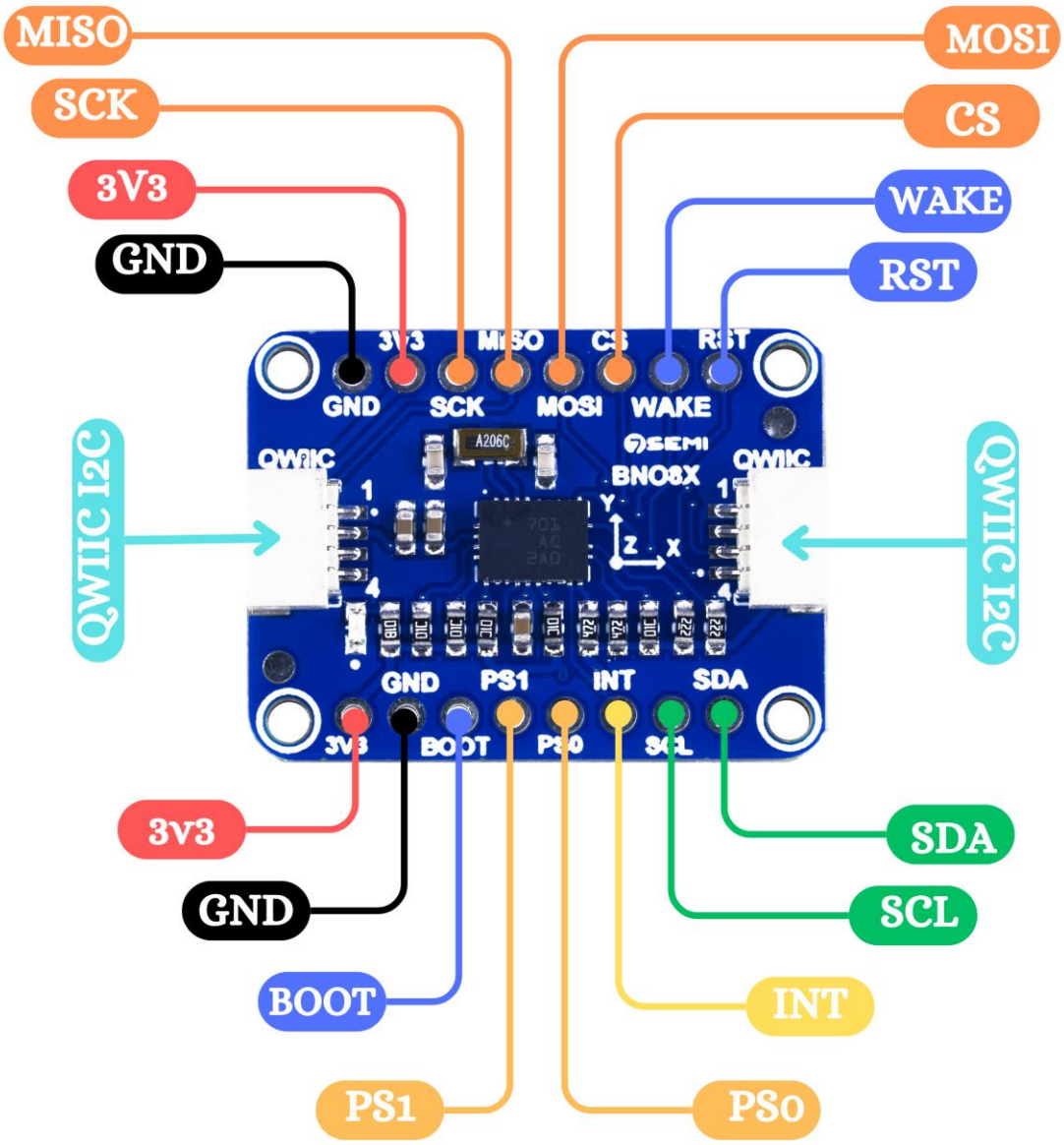
2.Technical Specification

The **Technical Specification** table provides detailed information about **7Semi BNO085 9-DOF Orientation IMU Fusion Breakout Qwiic**, including its operating voltage, current consumption, and electrical characteristics. This data helps users understand the power requirements, communication parameters, and performance capabilities of the sensor. It ensures compatibility with different microcontrollers and embedded systems while providing guidelines for efficient integration into various applications.

BNO085 Specifications

- Accelerometer Range: ±8g
- Gyroscope Range: ±2000°/s
- Magnetometer Range: ±8 Gauss
- Communication Interfaces:
  - I²C (Qwiic)
  - SPI
  - UART
- Operating Voltage: 1.8V - 3.6V
- Power Consumption: Low-power mode available for energy-efficient operation
- Maximum Sensor Data Rates:
  - Gyro Rotation Vector: Up to 1000Hz
  - Rotation Vector & Game Rotation Vector: Up to 400Hz
  - Geomagnetic Rotation Vector: Up to 90Hz
  - Accelerometer: Up to 500Hz
  - Gyroscope: Up to 400Hz
  - Magnetometer: Up to 100Hz
- Firmware Upgrade: Supports Device Firmware Update (DFU) over I²C, SPI, or UART
- Package Dimensions: 3.8mm x 5.2mm x 1.1mm

3.Pinouts

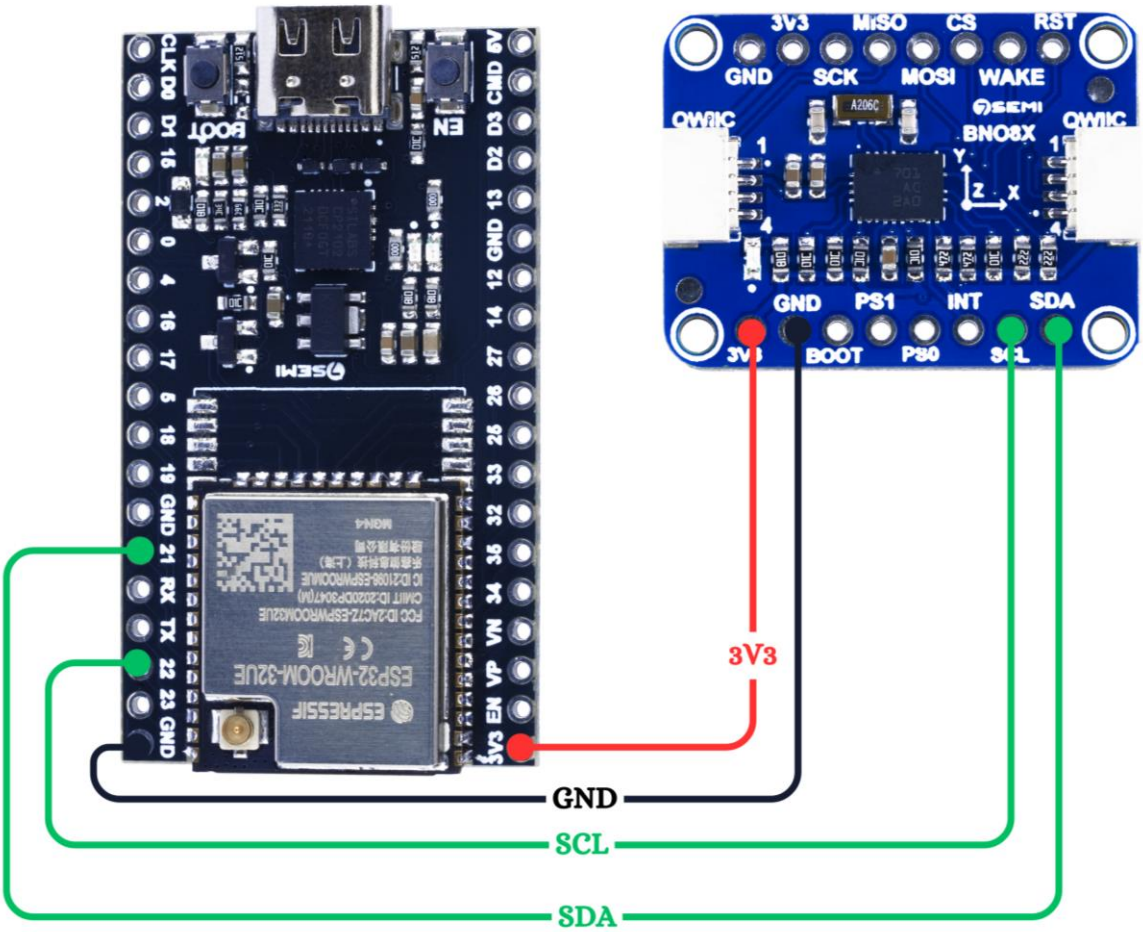

MISO
SCK
3V3
GND
MOSI
CS
WAKE
RST
QWIIC I2C
QWIIC I2C
3V3
GND
BOOT
PS1
INT
PS0
SDA
SCL
GND
SCK
MOSI
WAKE
A206C
BNO8X
QWIIC
QWIIC
GND
PS1
INT
SDA
3V3
BOOT
PS0
SCL

Pin Number	Pin Name	Description
1	3V3	Power Input (3.3V)
2	GND	Ground Connection
3	MISO	SPI Data Output (Master In Slave Out)
4	SCK	SPI Clock
5	MOSI	SPI Data Input (Master Out Slave In)
6	CS	SPI Chip Select
7	WAKE	Wake-up pin for the IMU
8	RST	Reset pin
9	BOOT	Bootloader mode activation
10	PS1	Mode selection pin
11	PS0	Mode selection pin
12	INT	Interrupt pin
13	SCL	I²C Clock Line
14	SDA	I²C Data Line
15	Qwiic	Easy plug-and-play I²C communication

Connection Guidelines

- For I²C communication, connect SCL to the microcontroller's SCL pin and SDA to the SDA pin.
- For SPI communication, use MOSI, MISO, SCK, and CS accordingly.
- The BOOT pin is used for firmware updates or entering bootloader mode.
- The WAKE pin can be used to wake up the sensor from low-power mode.
- The INT pin serves as an interrupt output for motion detection events.

4.Hardware Interface



Connection Explanation :

- Connect ESP32 3.3V to BNO085 3V3 for power.
- Connect ESP32 GND to BNO085 GND for a common ground.
- Connect ESP32 GPIO 21 (SDA) to BNO085 SDA for data communication.
- Connect ESP32 GPIO 22 (SCL) to BNO085 SCL for the clock signal.
- This configuration enables communication between the ESP32 and BNO085 using the I²C protocol, by default PS0 & PS1 are connected to ground on the bottom side of the board to set I2C mode.

5.Example code link

We provide example codes to help you get started with the **7Semi BNO085 9-DOF Absolute Orientation Sensor Breakout I2C Qwiic**. These examples demonstrate how to communicate with the sensor and retrieve **absolute orientation, acceleration, gyroscope, and magnetometer data** using the **I²C protocol**. The code is available for two popular platforms: **Arduino and ESP32**.

Code Explanation

```
include <Adafruit_BNO08x.h>


// For SPI mode, we need a CS pin

#define BNO08X_CS 10

#define BNO08X_INT 9


// For SPI mode, we also need a RESET

//#define BNO08X_RESET 5

// but not for I2C or UART

#define BNO08X_RESET -1


Adafruit_BNO08x  bno08x(BNO08X_RESET);

sh2_SensorValue_t sensorValue;


void setup(void) {

  Serial.begin(115200);

  while (!Serial) delay(10);     // will pause Zero, Leonardo, etc
until serial console opens
```

```
  Serial.println("Adafruit BNO08x test!");


  // Try to initialize!
  if (!bno08x.begin_I2C(0x4B)) {
  //if (!bno08x.begin_UART(&Serial1)) {  // Requires a device with >
300 byte UART buffer!
  //if (!bno08x.begin_SPI(BNO08X_CS, BNO08X_INT)) {
    Serial.println("Failed to find BNO08x chip");
    while (1) { delay(10); }
  }
  Serial.println("BNO08x Found!");




  for (int n = 0; n < bno08x.prodIds.numEntries; n++) {
    Serial.print("Part ");
    Serial.print(bno08x.prodIds.entry[n].swPartNumber);
    Serial.print(": Version :");
    Serial.print(bno08x.prodIds.entry[n].swVersionMajor);
    Serial.print(".");
    Serial.print(bno08x.prodIds.entry[n].swVersionMinor);
    Serial.print(".");
    Serial.print(bno08x.prodIds.entry[n].swVersionPatch);
    Serial.print(" Build ");
```

```
    Serial.println(bno08x.prodIds.entry[n].swBuildNumber);
  }

  setReports();

  Serial.println("Reading events");
  delay(100);
}

// Here is where you define the sensor outputs you want to receive
void setReports(void) {
  Serial.println("Setting desired reports");
  if (! bno08x.enableReport(SH2_GAME_ROTATION_VECTOR)) {
    Serial.println("Could not enable game vector");
  }
}

void loop() {
  delay(10);

  if (bno08x.wasReset()) {
    Serial.print("sensor was reset ");
    setReports();
  }
```

```
  if (! bno08x.getSensorEvent(&sensorValue)) {
    return;
  }

  switch (sensorValue.sensorId) {

    case SH2_GAME_ROTATION_VECTOR:
      Serial.print("Game Rotation Vector - r: ");

      Serial.print(sensorValue.un.gameRotationVector.real);
      Serial.print(" i: ");
        Serial.print(sensorValue.un.gameRotationVector.i);
        Serial.print(" j: ");
        Serial.print(sensorValue.un.gameRotationVector.j);
        Serial.print(" k: ");
        Serial.println(sensorValue.un.gameRotationVector.k);
        break;
  }

}
```

1. Library and Pin Definitions

```
#include <Adafruit_BNO08x.h>
#define BNO08X_CS 10
#define BNO08X_INT 9
#define BNO08X_RESET -1
```

- Includes the library required to interface with the sensor.
- Defines SPI-related pins, though the code uses **I2C**, not SPI. The RESET pin is not used (-1).

2. Setup Function Overview

```
void setup(void) {
  Serial.begin(115200);
  while (!Serial) delay(10);
```

- Starts serial communication for logging via USB.
- Waits until the serial monitor opens.

```
if (!bno08x.begin_I2C()) {
    Serial.println("Failed to find BNO08x chip");
    while (1) { delay(10); }
  }
```

- Initializes the sensor via I2C (begin_I2C()).
- If initialization fails, it prints an error and halts.

```
for (int n = 0; n < bno08x.prodIds.numEntries; n++) {
    // Logs firmware version information from the sensor
  }
```

- Loops through the sensor's product IDs and prints out firmware version info.

setReports();

- Calls a function to enable specific sensor output (Game Rotation Vector).

3. setReports Function

```
   void setReports(void) {
     if (! bno08x.enableReport(SH2_GAME_ROTATION_VECTOR)) {
       Serial.println("Could not enable game vector");
     }
}
```

- Enables the **Game Rotation Vector** report, which provides orientation as a quaternion (r, i, j, k).

4. Loop Function

```
  void loop() {
  delay(10);
  if (bno08x.wasReset()) {
    setReports(); // Re-enable reports if the sensor resets
  }

  if (! bno08x.getSensorEvent(&sensorValue)) {
    return; // No new data
  }

  switch (sensorValue.sensorId) {
    case SH2_GAME_ROTATION_VECTOR:
      // Print the quaternion values
  }
}
```

- Checks if the sensor was reset, and re-enables reports if so.
- Reads new sensor data.
- If the data is from SH2_GAME_ROTATION_VECTOR, it prints the quaternion components to the serial monitor.

5. Output

- You'll see logs like:

- Game Rotation Vector - r: 0.98 i: 0.01 j: -0.05 k: 0.12

6. ESP32 Example Code

This example targets ESP32 boards and showcases:

- Configuring the ESP32 I²C interface to communicate with the **BNO085** sensor Board.
- Reading absolute orientation, acceleration, gyroscope, and magnetometer data.
- Printing the sensor data to the Serial Monitor.

How to Access the Code?

Download Link for Arduino and ESP32 Example: Click Here

5.1 Sample Serial Output Arduino

Sample output Image of Arduino:

```
Game Rotation Vector - r: 0.84 i: 0.37 j: -0.40 k: 0.05
Game Rotation Vector - r: 0.84 i: 0.37 j: -0.40 k: 0.05
Game Rotation Vector - r: 0.84 i: 0.37 j: -0.40 k: 0.05
Game Rotation Vector - r: 0.84 i: 0.37 j: -0.40 k: 0.05
Game Rotation Vector - r: 0.83 i: 0.37 j: -0.41 k: 0.05
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.41 k: 0.05
Game Rotation Vector - r: 0.84 i: 0.36 j: -0.41 k: 0.05
Game Rotation Vector - r: 0.84 i: 0.36 j: -0.41 k: 0.05
Game Rotation Vector - r: 0.84 i: 0.36 j: -0.41 k: 0.05
Game Rotation Vector - r: 0.84 i: 0.36 j: -0.41 k: 0.05
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.41 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.41 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.41 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.41 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.41 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.41 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.42 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.42 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.42 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.42 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.37 j: -0.42 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.37 j: -0.42 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.37 j: -0.41 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.37 j: -0.41 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.37 j: -0.41 k: 0.05
Game Rotation Vector - r: 0.83 i: 0.37 j: -0.41 k: 0.05
Game Rotation Vector - r: 0.83 i: 0.37 j: -0.41 k: 0.05
Game Rotation Vector - r: 0.83 i: 0.37 j: -0.41 k: 0.05
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.42 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.42 k: 0.06
Game Rotation Vector - r: 0.83 i: 0.36 j: -0.42 k: 0.06
```

6.Mechanical Specification