

BMP180

DIGITAL PRESSURE SENSOR

**Introduction**

The BMP180 chip is a pressure and temperature sensor that can be used to measure air pressure and environmental temperature. It is manufactured by Bosch Sensortec and is commonly used in pressure measurement applications, such as altitude measurement, weather monitoring, environmental control, and air conditioning.

The BMP180 uses piezo-resistive pressure sensor technology, in which pressure is converted into a resistance value. This chip offers high accuracy and low power consumption.

**Table of contents**

**1. BMP180 general description 1**

1.1. Overview of the BMP180 sensor **1**

1.2. Introduction to Drivers and Their Purpose **1**

1.3. Supported platform 1

## 2. Hardware and software requirements 2

## 2.1. Hardware 2

## 2.2. Software 2

**3. Connect hardware 3**

3.1. Connection pin diagram of BMP180 **3**

3.2. Describe the functions of the pins **3**

3.3. Guide to connect with Raspberry Pi **4**

**4. Basic usage guide 4**

4.1. Initialize the sensor using the Driver **5**

4.2. Advance functons and features **6**

4.3. Code explanation **6**

**5. EXAMPLE 7**

**1. BMP180 general description**

**1.1. Overview of the BMP180 sensor**

The BMP180 chip is a pressure and temperature sensor that can be used to measure air pressure and environmental temperature. It is manufactured by Bosch Sensortec and is commonly used in pressure measurement applications, such as altitude measurement, weather monitoring, environmental control, and air conditioning.

The BMP180 uses piezo-resistive pressure sensor technology, in which pressure is converted into a resistance value. This chip offers high accuracy and low power consumption.

The BMP180 can measure air pressure from 300 to 1100 hPa (hectopascals). This corresponds to approximately 9000 meters below sea level to 500 meters above sea level. However, to achieve the best accuracy, the recommended parameters for measuring air pressure are in the range of 700 to 900 hPa.

**1.2. Introduction to Drivers and Their Purpose**

This driver is developed to simplify the process of communication and data collection from the BMP180 pressure and temperature sensor. Instead of having to directly handle complex communication protocols like I2C and perform raw data calibration calculations, this driver provides a convenient abstraction layer. As a result, users can easily read processed and calibrated pressure and temperature values quickly with just a few simple lines of code. The main goal of the driver is to create a user-friendly and intuitive interface, allowing developers and users, including beginners, to effectively integrate the BMP180 sensor into their projects in a time-efficient manner.

**1.3.** **Supported platform**

Warning - I2C lines are not protected by any converter, so You will need some bidirectional logic level converter (like [this](https://nettigo.eu/products/636) or [this](https://nettigo.eu/products/1016)) to get the board working with Arduino. You can connect module directly to microcontrollers running on 3.3V like Raspberry Pi or [Teensy 3.2](https://nettigo.eu/products/919).

You can power module with 5V, there is 3.3V regulator on board. In this design Vdd = Vddio, and according to BMP180 datasheet maximum voltage on SDA/SCL pins is equal Vddio. Since it is only 3.3V connecting module to Arduino running on 5V TTL means You are working outside voltage specified in BMP180 datasheet. It may work or it may get damaged, YMMV. We recommend using logic level converter with an Arduino.

## 2. Hardware and software requirements

## 2.1. Hardware

This driver is designed to operate on a variety of popular hardware and software platforms, making it easy for you to integrate the BMP180 sensor into your diverse projects. The platforms that have been tested and are supported include:

*a.Arduino Microcontrollers*

Compatible with popular Arduino board families such as Uno, Nano, Mega, ESP32, ESP8266, and other variants.

Provides a convenient library that is easily integrated into the Arduino IDE development environment.

*b. Raspberry Pi Single-Board Computers*

Supports various Raspberry Pi versions (e.g., Raspberry Pi 4, 3, Zero) through standard I2C communication.

Offers a powerful Python library, allowing you to develop complex applications on the Linux platform.

*c. MicroPython Platforms*

Supports boards and microcontrollers running MicroPython firmware, expanding usability across other embedded devices.

*d. General Linux Systems*

Can be utilized on general Linux systems through programming language libraries such as Python with I2C support modules.

**2.2. Software**

**a. Required Software/Libraries (C Language in Visual Studio Code):**

To work with this BMP180 driver using the C language within the Visual Studio Code environment, you will need the following software and libraries installed:

**Visual Studio Code (VS Code):** A lightweight but powerful source code editor. You can download it for free from the official Visual Studio Code website.

**C/C++ Extension for VS Code:** This extension provides language support for C/C++ within VS Code.

**b. BMP180 C Driver/Library:** You will need the specific C driver or library for the BMP180 sensor that you intend to use.

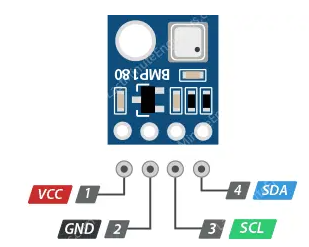
**c. Platform-Specific Libraries:**

Depending on the target hardware you are using the BMP180 with (e.g., Arduino, Raspberry Pi), you might need to include additional libraries for tasks like I2C communication.

**For Raspberry Pi:** You might use libraries like wiringPi or bcm2835 for I2C access from C.

**3. Connect hardware**

**3.1. Connection pin diagram of BMP180**

****

**3.2. Describe the functions of the pins**

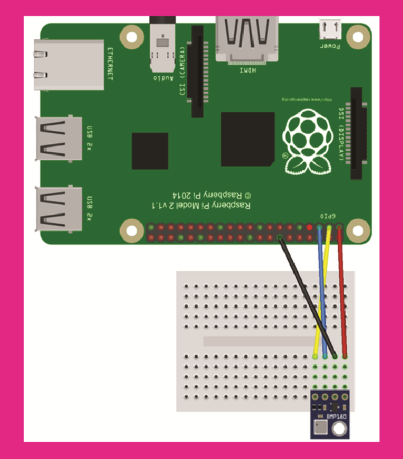
VCC: Power supply pin for the module (3.3V or 5V).

GND: Ground pin.

SDA: Data pin (Serial Data) in I2C communication, connected to the SDA pin of the Arduino.

SCL: Clock pulse pin (Serial Clock) in I2C communication, connected to the SCL pin of the Arduino.

**3.3. Guide to connect with Raspberry Pi**

****

Connect the VCC pin of the BMP180 to the 3.3V pin on the Raspberry Pi (Pin 1 or Pin 17 on most models). It's generally recommended to use 3.3V for I2C peripherals with the Raspberry Pi.

Connect the GND pin of the BMP180 to a GND pin on the Raspberry Pi (e.g., Pin 6, 9, 14, 20, 25, 30, 34, or 39).

Connect the SCL pin of the BMP180 to the SCL pin on the Raspberry Pi (Pin 5 - GPIO3).

Connect the SDA pin of the BMP180 to the SDA pin on the Raspberry Pi (Pin 3 - GPIO2).

**4. Basic usage guide**

**4.1. Initialize the sensor using the Driver**

**4.1.1. Prerequisites**

**Raspberry Pi (or Linux system):** You need a system with I2C enabled and the BMP180 sensor connected to the I2C bus. Ensure the physical connections are correct (refer to the hardware connection guide).

**Kernel Development Environment:** You need the necessary tools to compile kernel modules on your system. This usually includes the kernel headers corresponding to your running kernel.

**4.1.2. Building the Driver**

Save the code: Save the provided C code as a .c file “bmp180\_driver.c”.

Create a Makefile: In the same directory as your .c file.

obj-m += bmp180\_driver.o

all:

    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:

    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

**Compile the driver:** Open a terminal in the directory where you saved the files and run with syntax: “make”.

This command will compile the “bmp180\_driver.c” file and create the kernel module “bmp180\_driver.ko”.

**4.1.3. Loading the Driver**

**Load the module:** Use the “insmod” the compiled kernel module with syntax:

sudo insmod bmp180\_driver.ko

**4.1.4. Interacting with the Driver**

**Find the device node:** The driver creates a device node in the /dev directory. You can usually find it with the name bmp180, with syntax:

ls /dev/bmp180

**Compile the user-space application:** Save the C application code “bmp180\_driver.c” and compile it using GCC:

gcc bmp180\_driver.c -o bmp180\_driver

**Run the user-space application:** Execute the compiled application with root privileges (as it interacts with a device file):

sudo ./bmp180\_driver

**4.1.5. Unloading the Driver**

When you are finished using the driver, you can unload it using the “rmmod” command:

sudo rmmod bmp180\_driver

**4.2. Advance functons and features**

Opening the Device File: It opens the character device file “/dev/bmp180" for reading and writing. This establishes a connection to the kernel driver.

Reading Temperature: It uses the “ioctl()” system call with the “BMP180\_IOCTL\_READ\_TEMP” command to request the raw temperature reading from the kernel driver. The result is stored in the temp variable.

Reading Pressure: It uses the “ioctl()” system call with the “BMP180\_IOCTL\_READ\_PRESSURE” command to request the raw pressure reading from the kernel driver. The result is stored in the pressure variable.

Printing Data: It prints the retrieved raw temperature and pressure values to the standard output.

Closing the Device File: It closes the connection to the kernel driver by closing the file descriptor “fd”.

Error Handling: It includes basic error checking after the “open()” and “ioctl()” calls using “perror()” to print error messages if these operations fail.

**4.3. Code explanation**

Include the library

#include <stdio.h>

#include <fcntl.h>

#include <sys/ioctl.h>

#include <unistd.h>

## Define IOCTL to read data

#define BMP180\_IOCTL\_READ\_TEMP \_IOR('b', 1, int)

#define BMP180\_IOCTL\_READ\_PRESSURE \_IOR('b', 2, int)

## The main() function begins

int main() {

    int fd;

    int temp, pressure;

## Turn on the device

fd = open("/dev/bmp180", O\_RDWR);

    if (fd < 0) {

        perror("Failed to open BMP180 device");

        return -1;

    }

## Send IOCTL to read the temperature

// Đọc nhiệt độ

    if (ioctl(fd, BMP180\_IOCTL\_READ\_TEMP, &temp) < 0) {

        perror("Failed to read temperature");

        close(fd);

        return -1;

    }

    printf("Temperature: %d\n", temp);

Send IOCTL to read pressure

// Đọc áp suất

    if (ioctl(fd, BMP180\_IOCTL\_READ\_PRESSURE, &pressure) < 0) {

        perror("Failed to read pressure");

        close(fd);

        return -1;

    }

    printf("Pressure: %d\n", pressure);

Shut down the device and finish

close(fd);

    return 0;

}

**5. Example**

#### example basic

#include <stdio.h>

#include <fcntl.h>

#include <sys/ioctl.h>

#include <unistd.h>

#define BMP180\_IOCTL\_READ\_TEMP \_IOR('b', 1, int)

#define BMP180\_IOCTL\_READ\_PRESSURE \_IOR('b', 2, int)

int main() {

    int fd;

    int temp, pressure;

    fd = open("/dev/bmp180", O\_RDWR);

    if (fd < 0) {

        perror("Failed to open BMP180 device");

        return -1;

    }

    // Đọc nhiệt độ

    if (ioctl(fd, BMP180\_IOCTL\_READ\_TEMP, &temp) < 0) {

        perror("Failed to read temperature");

        close(fd);

        return -1;

    }

    printf("Temperature: %d\n", temp);

    // Đọc áp suất

    if (ioctl(fd, BMP180\_IOCTL\_READ\_PRESSURE, &pressure) < 0) {

        perror("Failed to read pressure");

        close(fd);

        return -1;

    }

    printf("Pressure: %d\n", pressure);

    close(fd);

    return 0;

}

Kết quả mẫu sau khi chạy thành công

Nhiệt độ: 25.3°C → thường trả về 253 nếu chưa chia.

Áp suất: 101325 Pa → driver có thể trả về 101325.

Temperature: 253

Pressure: 101325