BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

DEVICE DRIVER PROJECT

M.E EMBEDDED SYSTEMS

Electrical and Electronics Engineering

Instructor In-Charge:
Dr. Devesh Samaiya

Submitted on:
2nd May 2022

Submitted by:

- **Anurag Dobriyal (2021H1400346P)**
- **Vivek Kumar(2021H1400163P)**

Aim of the project:

Interfacing the BMP280 (Temperature and pressure) sensor with single board computer (RaspberryPi), writing driver for the sensor used using Industrial Input output (IIO) subsystem and exporting the functionality to the user space.

Introduction:

Through this project we aim to gain lucidity regarding how interfacing of I2C based sensors are done at kernel level. The project demonstrates the functionality of BMP280(I2C protocol-based sensor) that is interfaced via I2C to Raspberry Pi 4. BMP280 works using I2C protocol.

I2C is a serial communication protocol for a two-wire interface to connect low-speed devices like A/D and D/A converters, I/O interfaces, and other similar peripherals in embedded systems. Each I2C slave device needs an address.

In this code, we have used the standard kernel module bmp280.c to implement the functionality of client registration to adapter. Read/Write mutex is implemented while reading and writing on the Bus, device file is dynamically allocated and IIO subsystem is used to indicate the temperature and pressure level.

IIO is a Unified framework for writing drivers for many different types of embedded sensors. Standard interface to user space applications manipulating sensors. IIO is a core subsystem dedicated to ADC and DAC converter. To unify more and more sensor functionality to the kernel space.

An IIO device usually corresponds to a single hardware sensor and it provides all the information needed by a driver handling a device. Let's first have a look at the functionality embedded in an IIO device then we will show how a device driver makes use of an IIO device. IIO category would be connected via SPI or I2C.

Requirements:

Hardware:

- Raspberry Pi 4
  (Data sheet : **https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf)**
- Memory card & Card Reader
- BMP280 Temperature and pressure Sensor
- Ethernet cable
- USB cable (Type C) and Jumper wires

Software:

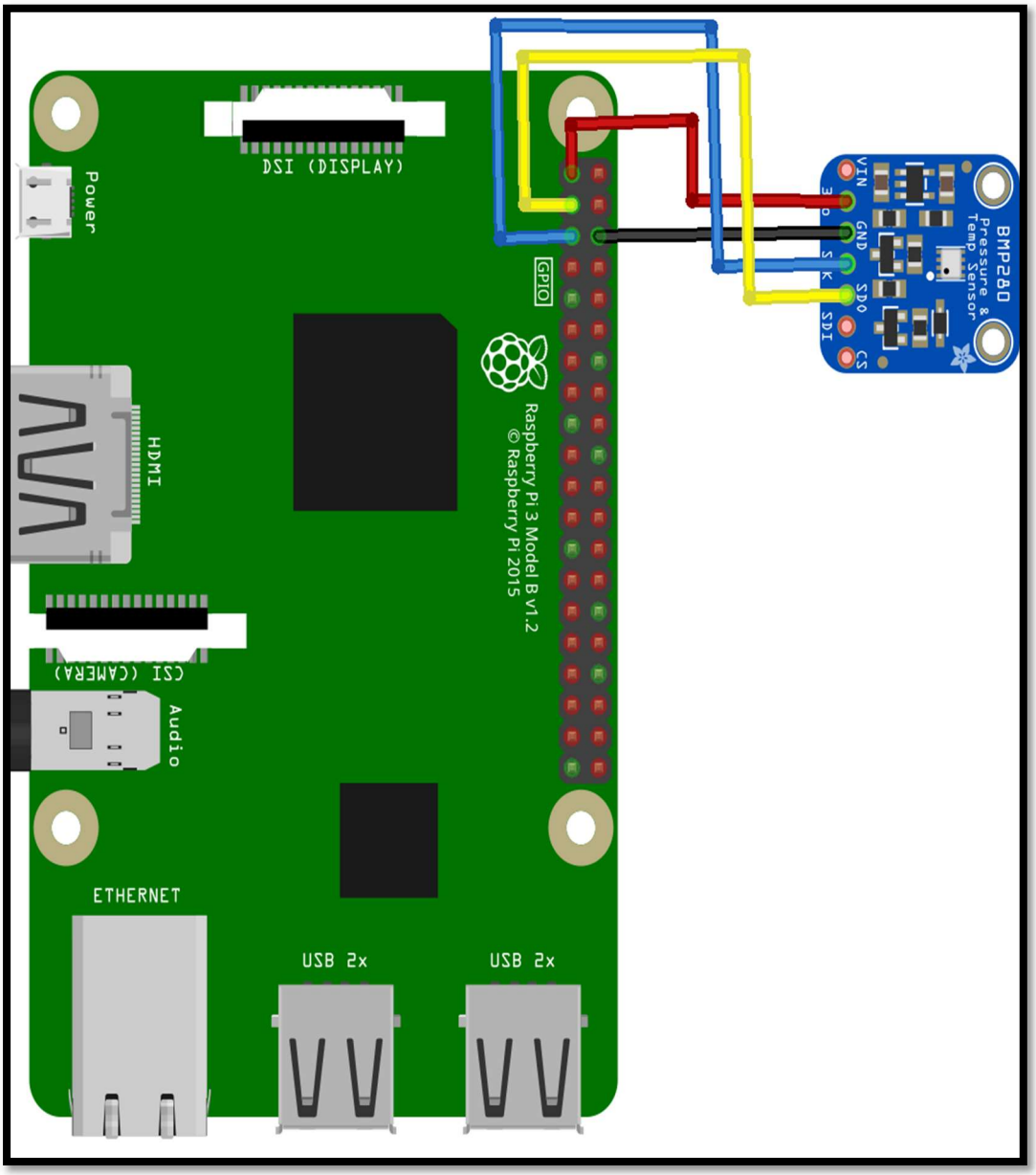
- Raspbian OS installed on Raspberry Pi 4 via Raspberry Pi imager.
- VNC viewer installed.
- Compatible Linux headers with your kernel versions.

Connections:

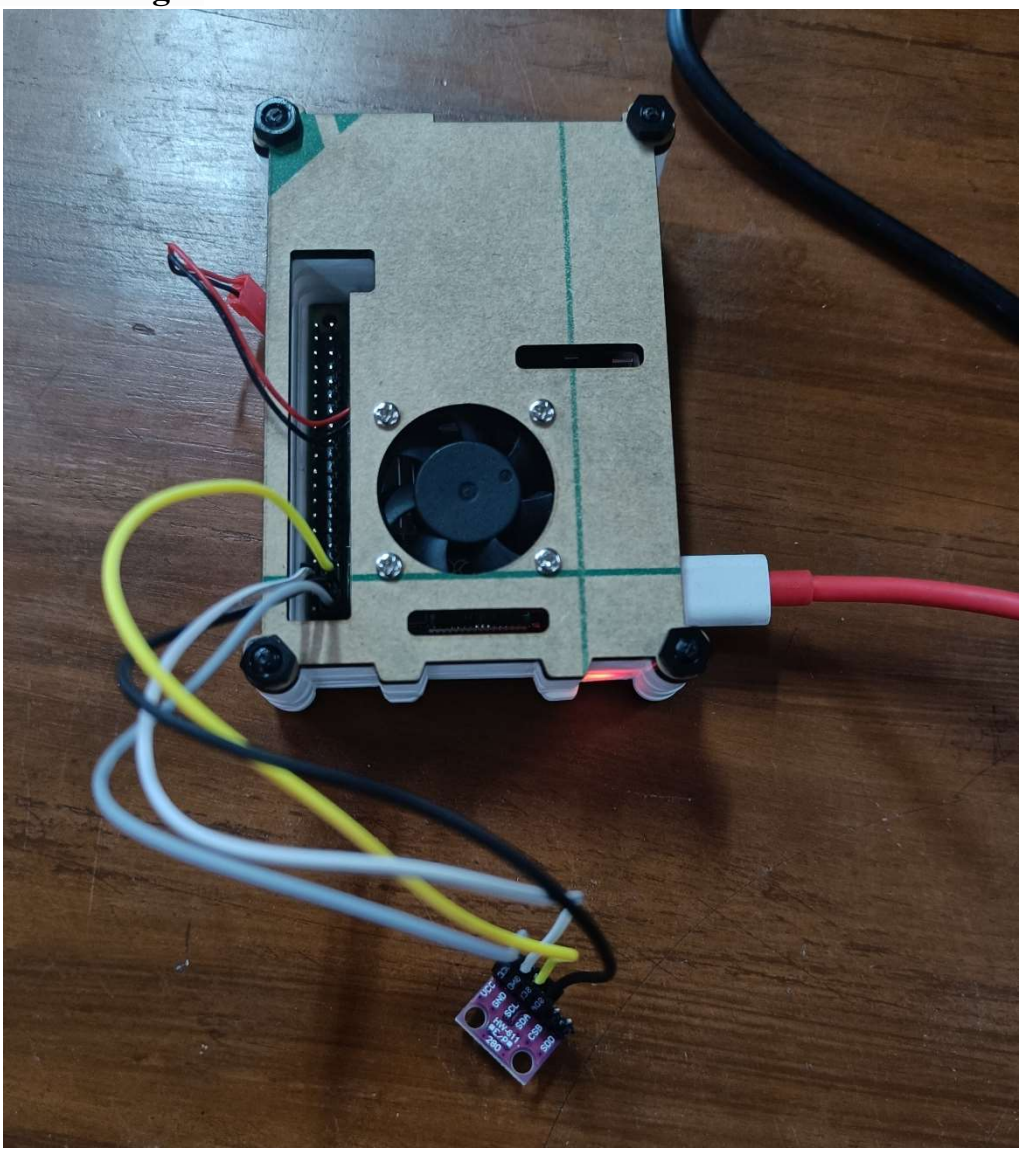
- Connect ground of BMP280 to the ground of the Raspberry pi.
- Connect SDA pin of BMP280 to the SDA pin (pin no 2) of the Raspberry pi.
- Connect SCL pin of BMP280 to the SCL pin (pin no 3) of the Raspberry pi.

• Connect Vin pin of BMP280 to the 5v pin (pin no 2) of the Raspberry pi.

Schematic:



Hardware Design:



About Sensor:

The BMP280 is an absolute barometric pressure sensor, feasible for mobile applications. Its small dimensions and its low power consumption allow for the implementation in battery-powered devices such as mobile phones, GPS modules, or watches. The BMP280 is based on Bosch's piezo-resistive pressure sensor technology featuring high accuracy and linearity. The device is optimized in terms of power consumption, resolution, and filter performance.BMP280 is equipped with a IIR filter in order to minimize disturbances. The filter coefficient ranges from 0 to 16. The BMP280 supports the I²C and SPI digital interfaces. It acts as a slave for both protocols. The I²C interface supports the Standard, Fast and High-Speed modes.

https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf

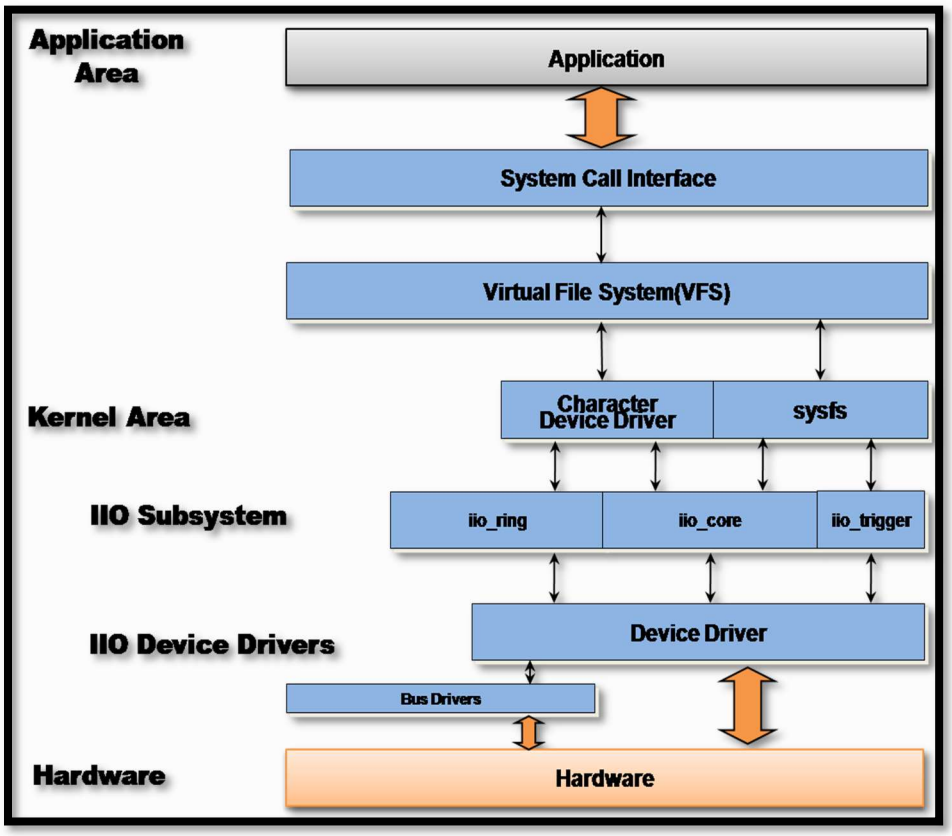
Sensor Test:

Once the connections are complete sensor is being checked whether is getting detected in Raspberry pi.

Steps:

- Run $ lsmod on the Terminal.
- If i2c_dev appears in the list go to step 4th.
- If it does not appear run $sudo modprobe device name (for our case it is sudomod probe industrialio) and check again using lsmod command.
- Assuming that i2c_dev driver is present run $ i2cdetect –y 1. This command should display the slave address of this sensor which is 0x76.

IIO Frame work Architecture and Layout:



Kernel Space driver and its building process:

Given below are the steps to compile and build the driver and user space programs.

1.The folder contains Kernel Level code “bmp280.c”, user space code “bmp280.c” and Makefile and save them on Desktop or any directory you want to save it in.

2. Go to the Directory, Let say, If you have saved in Desktop then use **cd /Desktop/bmp280** interface to reach the directory.

3. Run **$ make all** on terminal. This should generate bmp280.ko file successfully along with a few other files.

4. Compile the user code using command $ gcc bmp280. c.

5. This should generate an executable file.

6. Insert the module using $ sudo insmod bmp280.ko.

7. Give permission to the device file using $ sudo chmod 777 /dev/bmp280_sensor

8. If everything goes right you should see the proximity value on the Terminal.

9. To read the Status of Interrupt and other information run $dmesg.

10.To Remove the module run $sudo rmmod i2c.ko.

Brief Algorithms:

- Declare BMP280 specific registers.
- Device registration, at the virtual file system.
- Declare channel and read the channel.
- Get I2C device, BMP280 I2C address is 0x76(108)
- Read 24 bytes of data from address(0x88)
- Convert the data and calibrate it.
- Export to the use space.

On our understanding while going through the project

The kernel divided the I2C subsystem by **Buses** and **Devices**. The **Buses** are again divided into **Algorithms** and **Adapters**. The **devices** are again divided into **Drivers** and **Clients**. The below image will give you some understandings. Driver register the client in the sysf directory.

Library for Interfacing with Linux IIO Devices is a type of system call, Identify the IIO device that can be used and channels for the device. Attributes specific to channel and device. Creates the context where device will be placed. Interfaces through sysfs virtual file system.

Each and every IIO device, typically a hardware chip, has a device folder under **/sys/bus/iio/devices/iio:deviceX.** Where X is the IIO index of the device. This directory has set of files, depending on the characteristics and features of the hardware device. In order to determine which IIO deviceX corresponds to which hardware device, the user can read the name file **/sys/bus/iio/devices/iio:deviceX/name**.

Other way that the user space application to interact with the IIO driver **/dev/iio:deviceX,** character device node interface used for buffered data transfer and for events information retrieval.

/sys/bus/iio/iio: deviceX/: Represents sensors and their channels. **/dev/iio:deviceX:** Character device that represents the export of device events and data buffers.

Drivers use a tools and API provided by the IIO core to manage hardware and report the IIO core for Processing. Whatever the driver we are writing that has been exposed to the IIO core of the IIO subsystem where the hardware is manged.

IIO subsystem extracts the whole underlying mechanism into user space through sysfs (files system) interface and character device, on which users can execute system calls. All-important IIO structures are defined in **include/linux/iio/iio.h.**

Both pressure and temperature values are expected to be received in 20-bit format, positive, stored in a 32-bit signed integer [*according to Data sheet*].

The Industrial I/O core offers both a unified framework for writing drivers for many different types of embedded sensors and a standard interface to user space applications manipulating sensors. The implementation can be found under drivers/iio/industrialio.

IIO provides Generalised device registration and handling which is top of the character device. IIO will register like character device. IIO has Data trigger support to provide the user so that user can access the data.

IIO core provides functionality like device registration and handling structure. And IIO trigger provides handling of GPIO. Handling data to the user space.

Multiple use major no. and minor no. for every channel can be avoided using IIO frame work.

***

Authors:

- Anurag Dobriyal (2021H1400346P), BITS Pilani, Pilani campus.
- Vivek Kumar (2021H1400163P), BITS Pilani, Pilani campus.

***