

|  |  |  |
| --- | --- | --- |
| **Laboratory for Instrumentation and Measurement**  Department of Electrical Engineering  University of Applied Sciences Hamburg | | |
| Group No: 02 | **LAB 02** | **Celestine Machuca**  2570138 |
| Date: 26.04.2023 | **Soodeh Mousavi asl**  2571713 |
| Professor:  Prof.Dr. Rasmus Retting |
| Function and Characterization of  an Inertial Sensor Cluster / I2C bus | | |

**Contents**

[Objectives 3](#_Toc133540851)

[Materials used 3](#_Toc133540852)

[1 Setup 3](#_Toc133540853)

[2 The Interface 6](#_Toc133540854)

[3 Sensor characteristics (Accuracy/Resolution/Quantization) 7](#_Toc133540855)

[4 Modifying the program 10](#_Toc133540856)

[5 Ultrasound sensing of different objects. 10](#_Toc133540857)

[6 Programming a threshold 13](#_Toc133540858)

# Objectives

# Materials used

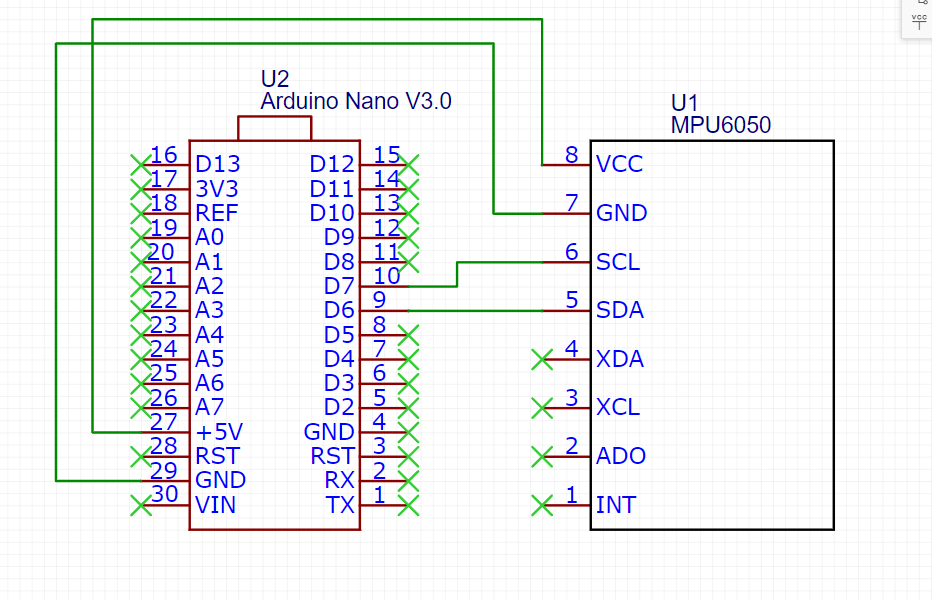
* Arduino nano microcontroller
* Breadboard for prototyping
* MPU\_6050 Inertial sensor
* USB-Mini Kabel
* 4 Male-Male Connectors
* Terminal-Software (HTerm), Processing (https://processing.org)
* PlatformIO for code compilation and uploading
* Jupiter Notebook running Python For histograms
* Oscilloscope
* Personal PC

# Setup

The MPU6050 inertial sensor cluster has been connected to Arduino using the I2C protocol for communication. Only two wires were needed for connection, and two wires for the power. See tables below:

|  |  |
| --- | --- |
| MPU-6050 | Arduino |
| VCC | +5V |
| GND | GND |
| SCL | D7 |
| SDA | D6 |

The diagram in figure 1 shows the hardware setup and pin connections:



Next, the given program (BasisMPU6050) was compiled and transferred to the Arduino. After that the "Basisprogramm\_ReadUSB\_MPU" program was loaded into the editor (Spyder?Jupiter?).

To investigate the data flow, both programs were analyzed.

**BasisMPU6050**: this program is reading and recording data from the sensor. It uses the I2C protocol to communicate with sensor using the Wire library and obtain the measurement data including acceleration and gyroscope values along x, y, and z axis as well as the temperature value. At the end the program outputs these values to the serial monitor for display using Serial.print() function. The overall data flow of this program can be described as follows: it initializes communication with the sensor, then it requests the data, after it extracts and stores the data and outputs them for display.

**Basisprogramm\_ReadUSB\_MPU**: this one is to record and plot measurement data from the module connected to the computer through a serial port. It communicates with the sensor via necessary libraries (serial, csv, numpy, matplotlib) and stores the data then plot them and runs statistical analysis on the data.

This part of the **BasisMPU6050** program shows the code for recording the data:

// Accelerometer sensor

int acc\_X = (((int)result[0]) << 8) | result[1];

int acc\_Y = (((int)result[2]) << 8) | result[3];

int acc\_Z = (((int)result[4]) << 8) | result[5];

// Temperature sensor

int temp = (((int)result[6]) << 8) | result[7];

// Gyroscope sensor

int gyr\_X = (((int)result[8]) << 8) | result[9];

int gyr\_Y = (((int)result[10]) << 8) | result[11];

int gyr\_Z = (((int)result[12]) << 8) | result[13];

Example output:

1482,3320,47558,64891,65331,63,62103

1478,3320,47552,64889,65331,62,62103

1478,3324,47550,64890,65331,63,62102

1482,3326,47550,64889,65333,63,62101

1484,3326,47546,64889,65332,62,62100

1480,3326,47550,64890,65332,63,62101

1474,3326,47540,64890,65331,63,62100

1468,3328,47548,64890,65331,62,62099

1468,3320,47556,64889,65332,62,62098

1468,3318,47562,64888,65331,63,62098

1474,3322,47562,64888,65331,62,62096

1470,3324,47562,64888,65331,62,62096

1468,3328,47558,64887,65332,62,62096

1468,3326,47556,64887,65332,61,62096

1464,3324,47552,64887,65332,62,62097

The complete flow-chart visualizing how data flows through the system (MPU6050 -> Arduino -> computer) is represented in the diagram on figure 2:

Diagram

Description automatically generated

# Configuration

Using the datasheet, the sensor cluster can be set to different bandwidth and measurements through modifying the values of specific registers in the MPU-6050’s register map.

The 0x1A register – CONFIG – is used to configure the DLPF (Digital Low-Pass Filter) and the sampling rate divider (SMPLRT\_DIV). The 0x1B register - GYRO\_CONFIG – is for configuring the Gyroscope’s full-scale range. And the 0x1C register - ACCEL\_CONFIG – is for configuring the full-scale range of the accelerometer.

Hence, to set different bandwidth the DLPF\_CFG bits of 0x1A register were modified. The following table from the datasheet (https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf) represents the available options:

Graphical user interface, application, Word

Description automatically generated

Modifying the FS\_SEL bits of the 0x1B register was used to set the full-scale range of the gyroscope according to the table below from the same source:

|  |  |
| --- | --- |
| **FS\_SEL** | **Full-Scale Range** |
| 0 | ±250 °/s |
| 1 | ±500 °/s |
| 2 | ±1000 °/s |
| 3 | ±2000 °/s |

And by modifying the AFS\_SEL bits of the 0x1C register the full-scale range of the accelerometer was set according to the table below from the same source:

|  |  |
| --- | --- |
| **AFS\_SEL** | **Full-Scale Range** |
| 0 | ±2g |
| 1 | ±4g |
| 2 | ±8g |
| 3 | ±16g |

Next, channel X of accelerometer was selected to be set on different configurations for the measuring ranges. The experiment was performed to measure the digital output values for accelerometer at -1g, 0g, and +1g positions.

The results were analyzed to get the resolution of each of these measurements. Documents your tests and results

# Oscilloscope measurements on the I2C Bus

In this part of the experiment the setup was connected to the oscilloscope generating square wave. One probe was connected to the SCL pin to measure the voltage between SCL and GND and the second probe was connected to SDA to investigate the data line.

A screenshot of a computer

Description automatically generated with medium confidence

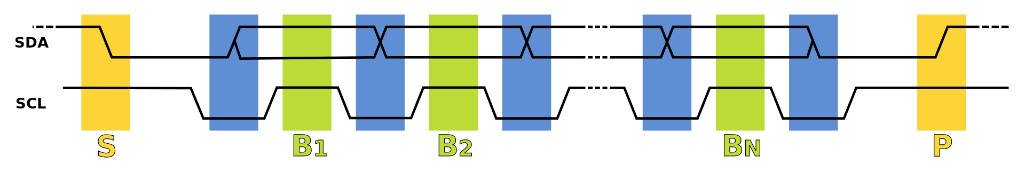
In order to investigate the behavior of the I2C protocol the results from this experiment were compared with the I2C protocol from https://de.wikipedia.org/wiki/I²C).

First the transmitted data between the start signal and the stop signal was matched to the one from the wiki page. The results of this comparisons can be seen in figures …, …, …. .

A screenshot of a computer

Description automatically generated with medium confidenceGraphical user interface

Description automatically generated



First observation was that the clock signal was present only during the data transmission. See figure… .

The start signal(S) which is a falling edge of data line while clock is high can be seen in figure… .

And the stop signal (P) that is rising edge of the SDA while SCL is high was also captured in figure… .

# Modifying the program

# Ultrasound sensing of different objects.

# Programming a threshold