



## Application Note

AN000601

# Hello World for AS726xN Sensors

**How to Implement a Connection between Sensor  
and  $\mu$ Controller**

v1-00 • 2019-May-07

---

---

# Content Guide

---

<b>1</b>	<b>Introduction.....</b>	<b>3</b>	<b>6</b>	<b>Code Debugging and Sensor</b>	
<b>2</b>	<b>Prepare the Connection .....</b>	<b>4</b>		<b>Reading Using IDE .....</b>	<b>13</b>
<b>3</b>	<b>Installation Requirement .....</b>	<b>6</b>	<b>7</b>	<b>Revision Information.....</b>	<b>16</b>
<b>4</b>	<b>Creating an Arduino Sketch with</b>		<b>8</b>	<b>Legal Information .....</b>	<b>17</b>
	<b>Wire Library.....</b>	<b>7</b>			
<b>5</b>	<b>Programming Sequence .....</b>	<b>9</b>			

---

# 1 Introduction

---

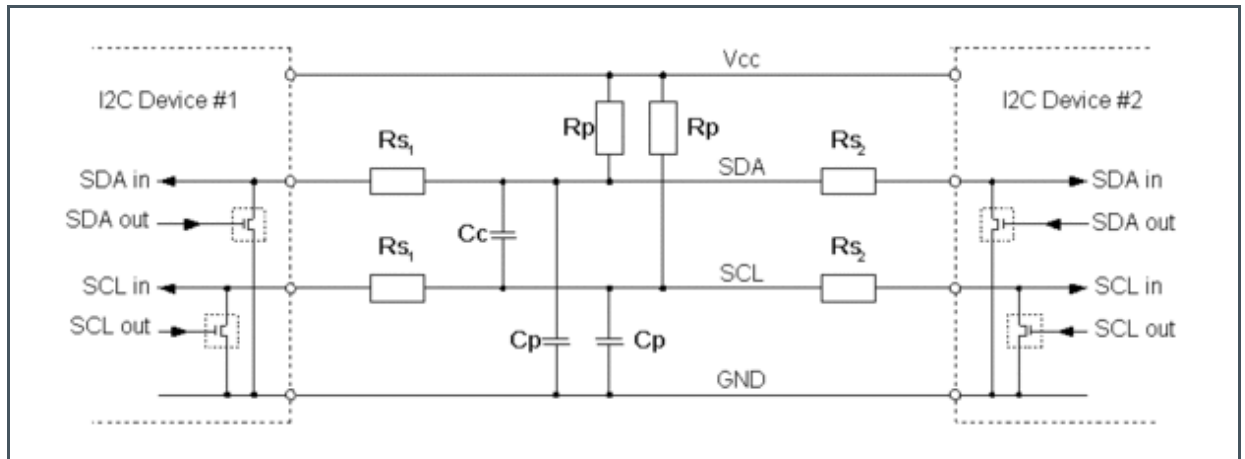
This “Hello World” – Application Note describes how to implement a connection between **ams** AS7264N sensor and a microcontroller via I<sup>2</sup>C Communication. This operates with an industrial-standard I<sup>2</sup>C connection. The I<sup>2</sup>C communication bus is very popular and broadly used by many electronic devices because it can be easily implemented in many electronic designs which require communication between a master and multiple slave devices or even multiple master devices. The easy implementations comes with the fact that only two wires are required for communication between up to almost 128 (112) devices when using 7 bits addressing and up to almost 1024 (1008) devices when using 10 bits addressing.

## 2 Prepare the Connection

The device has a preset ID or a unique device address so the master can choose with which devices will be communicated.

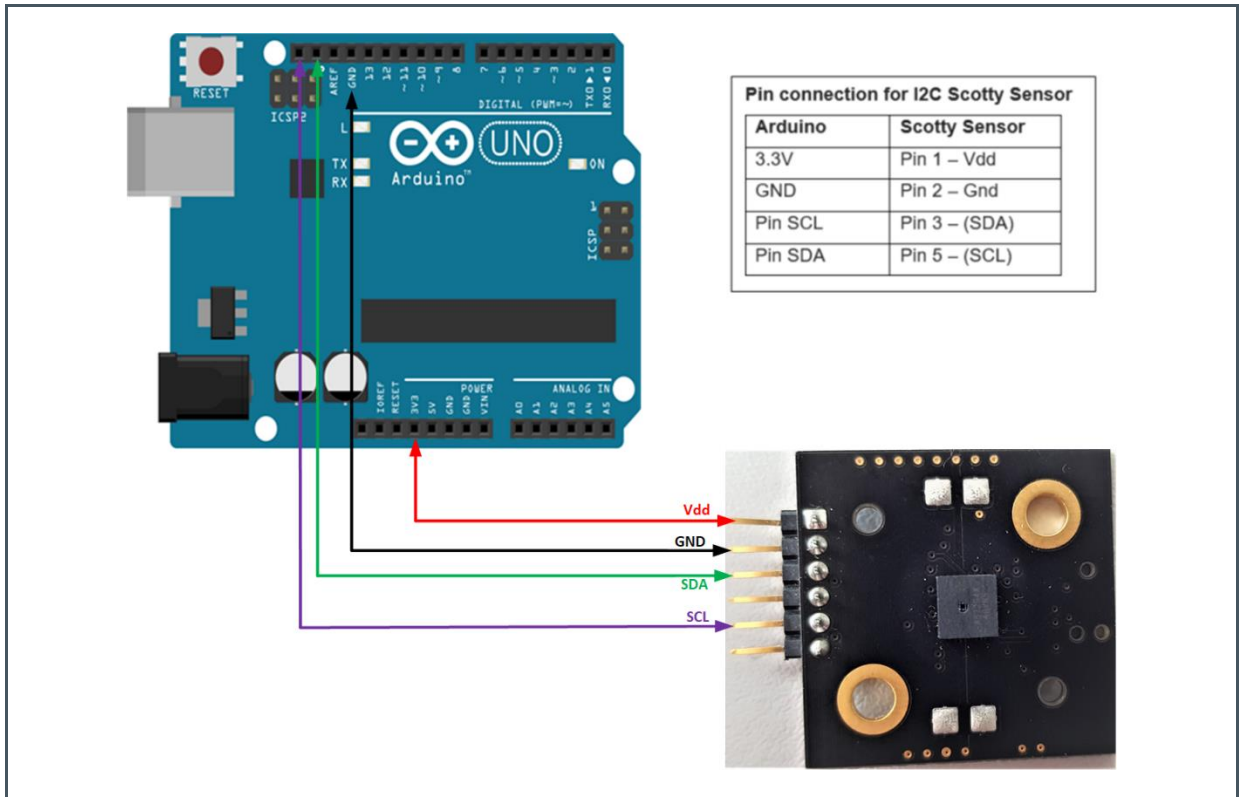
The two wires, or lines are called Serial Clock (or SCL) and Serial Data (or SDA). The SCL line is the clock signal that synchronize the data transfer between the devices on the I<sup>2</sup>C bus and the master device generates it. The other line is the SDA line, which carries the data. The two lines are “open-drain” which means that pull up resistors needs to be attached to them so that the lines are high because the devices on the I<sup>2</sup>C bus are active low. Commonly used values for the resistors are from 2K for higher speeds at about 400 kbit/s, to 10K for lower speed at about 100 kbit/s.

**Figure 1:**  
I<sup>2</sup>C Connection



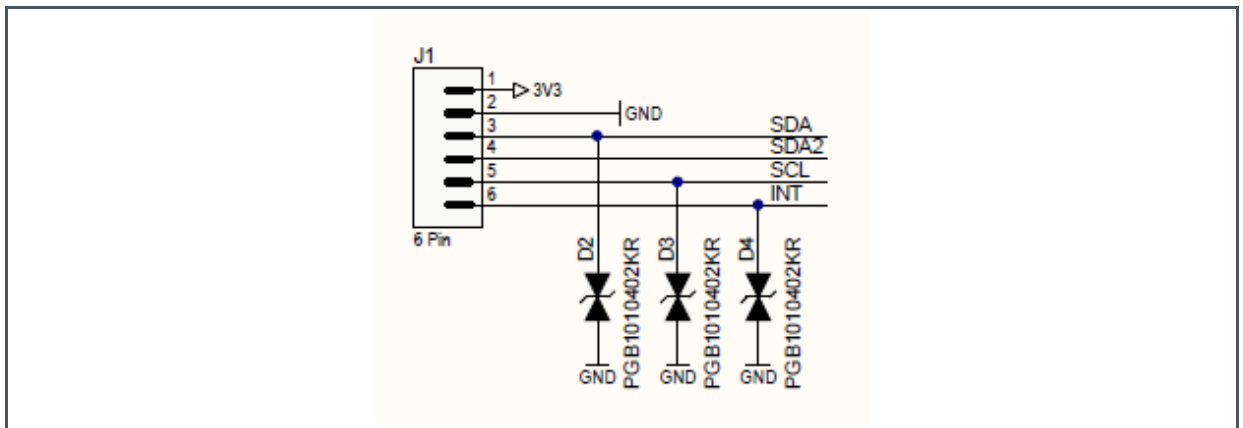
- VCC I<sup>2</sup>C supply voltage, typically ranging from 1.2 V to 3.6 V
- GND Common ground
- SDA Serial data (I<sup>2</sup>C data line)
- SCL Serial clock (I<sup>2</sup>C clock line)
- Rp Pull-up resistance (a.k.a. I<sup>2</sup>C termination)
- Rs Serial resistance
- Cp Wire capacitance
- Cc Cross channel capacitance

**Figure 2:**  
Hardware Connection of  $\mu$ C and AS7264N Sensor



Insert the device you want to communicate with in the sensor board. Connect ground on the breadboard to ground from the microcontroller. The AS6264N sensor can be equipped with 3.3V power connection. Connect the Serial data pin of sensor (PIN 3) to hardware Serial data pin of the microcontroller and Serial clock pin of sensor (PIN 5) to the hardware Serial clock pin of microcontroller as described in above Figure 2.

**Figure 3:**  
Connector J1 Pinning of AS7264N



---

## 3 Installation Requirement

---

Download the Arduino Software (IDE) from the website <https://www.arduino.cc/en/Main/Software>. Get the latest version from this download page. You can choose between the Installer (.exe) and the Zip packages. We suggest you use the first one that installs directly everything you need to use the Arduino Software (IDE), including the drivers. With the Zip package, you need to install the drivers manually.

When the download finished, proceed with the installation and please allow the driver installation process when you get a warning from the operating system.

The best way to become familiar with the Arduino Graphical User's Interface (GUI) is to verify your Arduino board is operating properly. Create an Arduino project and run the example Blink.

This simple test program confirms that a number of connection details and that the GUI are working properly.

---

## 4 Creating an Arduino Sketch with Wire Library

---

Once having Arduino development environment set up, you are ready to start working on projects. The section covers the basics that need to know to start writing your sketches and getting them to run on your Arduino. When using the Arduino IDE package, the sketches must follow a specific coding format.

This coding format differs a bit from what seen in a standard C language program. In a standard C language program, there is always a function named `main` that defines the code that starts the program. When the CPU starts to run the program, it begins with the code in the `main` function. On the other hand, Arduino sketches do not have a `main` function in the code. The Arduino bootloader program that is preloaded onto the Arduino functions as the sketch's `main` function. The Arduino starts the bootloader, and the bootloader program starts to run the code in your sketch. The bootloader program specifically looks for two separate functions in the sketch:

- `setup`
- `loop`

The Arduino bootloader calls the `setup` function as the first thing when the Arduino unit powers up. The code placed in the `setup` function in sketch only runs one time; then the boot-loader moves on to the `loop` function code.

The `setup` function definition uses the standard C language format for defining functions:

```
void setup () {  
  
  //code lines  
  
}
```

Just place the code to run at startup time inside the `setup` function code block. After the bootloader calls the `setup` function, it calls the `loop` function repeatedly, until power down the Arduino unit.

The `loop` function uses the same format as the `setup` function:

```
void loop () {  
  
  //code lines  
  
}
```

The main logic of the application code will be in the `loop` function section. This is where you place code to read sensors and send output signals to the outputs based on events detected by the sensors. The `setup` function is a great place to initialize input and output pins so that they are ready when the `loop` runs, then the `loop` function is where you use them.

## Including Libraries

Depending on how advanced your Arduino program is, you may or may not need to use other functions found in external library files. If you do need to use external libraries, you first need to define them at the start of your Arduino program, using the `#include` directive:

```
#include <library>
```

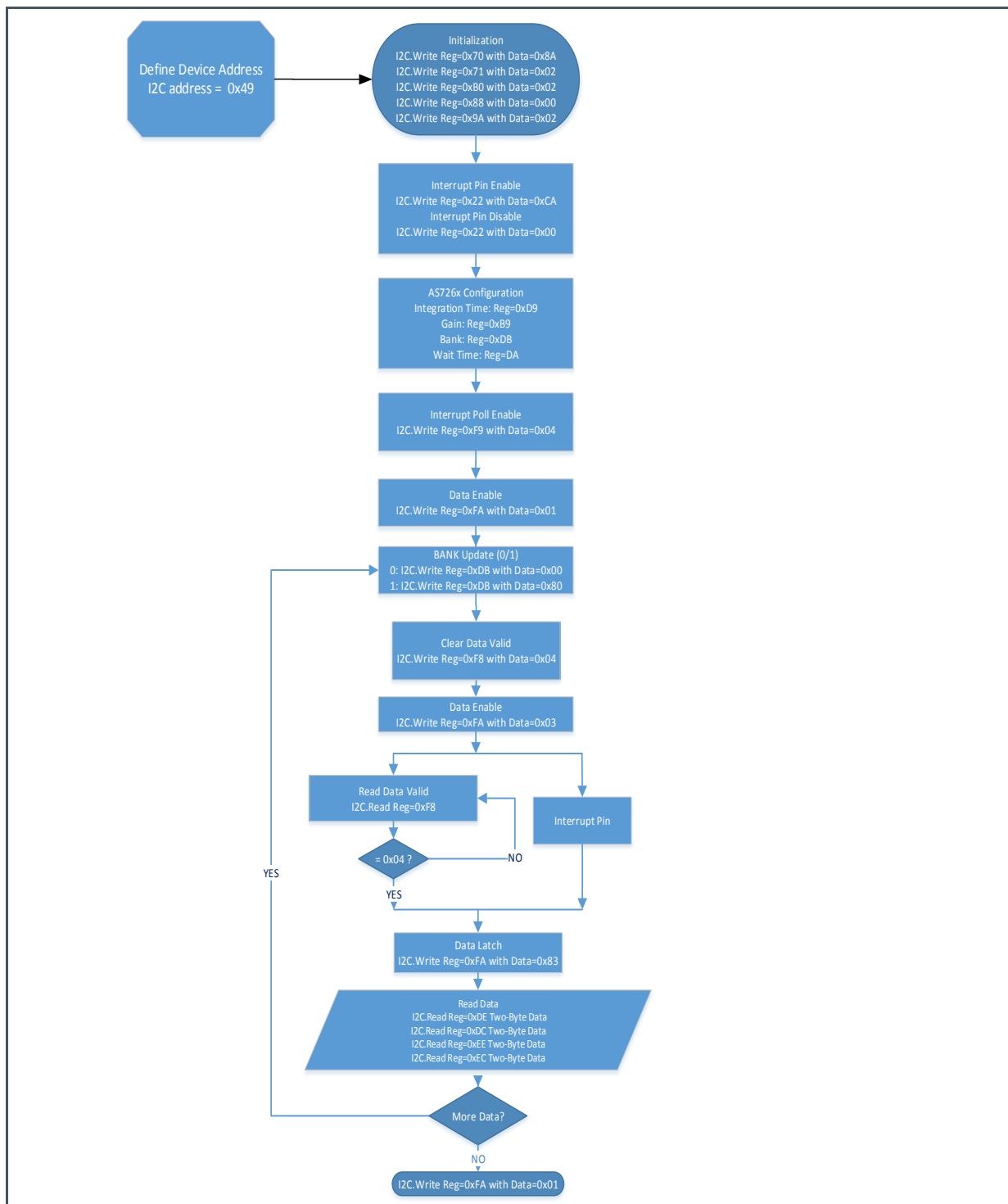
The `#include` directives will be the first lines in your sketch, before any other code. If you are using a standard Arduino shield, most likely the shield library code is already included in the Arduino IDE package. Just choose Sketch > Import Library from the menu bar, and then select the shield that you are using. The Arduino IDE automatically adds the `#include` directives required to write code for the requested shield.

When you open the Arduino IDE, the editor window starts a new sketch. The name of the new sketch appears in the tab at the top of the editor window area, in the following format:



# 5 Programming Sequence

Figure 4:  
Flow Chart



Define the hardware by I<sup>2</sup>C address and include the “Wire.h” file

```
#include <Wire.h>
#define _i2cAddr (0x49)
```

Function to read the values from the hardware registers

```
byte readRegister(byte addr)
{
    Wire.beginTransmission(_i2cAddr);
    Wire.write(addr);
    Wire.endTransmission();

    Wire.requestFrom(_i2cAddr, 1);

    if (Wire.available())
    {
        //Serial.println(Wire.read());
        return (Wire.read());
    }

    else
    {
        Serial.println("I2C Error");
        return (0xFF); //Error
    }
}
```

Function to write values into the hardware registers

```
void writeRegister(byte addr, byte val)
{
    Wire.beginTransmission(_i2cAddr);
    Wire.write(addr);
    Wire.write(val);
    Wire.endTransmission();
}
```

The `readTwoRegister1()` function reads the values from the corresponding data register. A 16-bit value is returned (high byte and lower byte combine) by the `readingL` and `readingH` variable

```
uint16_t readTwoRegister1(byte addr)
{
  uint8_t readingL; uint16_t readingH; uint16_t reading = 0;
  Wire.beginTransmission(_i2cAddr);
  Wire.write(addr);
  Wire.endTransmission();

  Wire.requestFrom(_i2cAddr, 2);

  if (2<=Wire.available())
  {
    readingL = Wire.read();
    readingH = Wire.read();
    readingH = readingH << 8;
    reading = (readingH | readingL);
    return(reading);
  }
  else
  {
    Serial.println("I2C Error");
    return (0xFFFF); //Error
  }
}
```

Like the `setup` line before it, `void loop ()` is another required Arduino-sketch function. While the `setup ()` function sets your Arduino up, the `loop ()` function executes the main logic in a loop.

This is where the bulk of your Arduino sketch is executed. The program starts directly after the opening curly bracket (`{`), runs until it sees the closing curly bracket (`}`), and jumps back up to the first line in `loop ()` and starts all over. The below sketch will get you up and running taking spectral readings. Once this sketch is uploaded, open the serial monitor with a baud rate of 115200 to display the spectral data from the sensor.

```

#include <Wire.h>
#define _i2cAddr (0x49)
void setup()
{
  // Initiate the Wire library and join the I2C bus as a master or slave
  Wire.begin();
  // communication with the host computer serial monitor
  Serial.begin(115200);
}

```

```

void loop()
{
writeRegister(byte (0x70), (0x8A)); //Device Config 1
writeRegister(byte (0x71), (0x02)); //Device Config 2
writeRegister(byte (0xB0), (0x02)); //Device Config 3
writeRegister(byte (0x88), (0x00)); //Device Config 4
writeRegister(byte (0x9A), (0x02)); //Device Config 5

writeRegister(byte (0x22), (0xCA)); //Int pin Enable

writeRegister(byte (0xD9), (0x00)); //Intergration TIME
writeRegister(byte (0xB9), (0x00)); //Gain IDRV
writeRegister(byte (0xDB), (0x80)); //Bank Mode
writeRegister(byte (0xDA), (0x80)); //WTIME

writeRegister(byte (0xF9), (0x04)); //Intterrupt and Polling enable

writeRegister(byte (0xFA), (0x83)); //Data Enable

writeRegister(byte (0xBA), (0x80)); //Auto Zero
writeRegister(byte (0xD3), (0x80)); //Temp Config
writeRegister(byte (0xEA), (0x03)); //LEN ON
writeRegister(byte (0x84), (0x01)); //LEN IND

writeRegister(byte (0xF8), (0x04)); //Intterrupt and Polling clear

  Serial.println(readRegister(byte(0x10))); //Device Identification
  Serial.println(readRegister(byte(0x11))); //Device Version
  Serial.println("Channels");
  Serial.println(readTwoRegister1(byte(0xDC))); //N LOW/HIGH
  Serial.println(readTwoRegister1(byte(0xDE))); //Y LOW/HIGH
  Serial.println(readTwoRegister1(byte(0xEC))); //Z LOW/HIGH
  Serial.println(readTwoRegister1(byte(0xEE))); //X LOW/HIGH
  Serial.println("Next");
  delay(2000);
}

```

---

## 6 Code Debugging and Sensor Reading Using IDE

---

One should start with the hardware connections and begin hardware debugging. Check wiring – one of the first things to do when wiring the circuit is to check that connected everything correctly.

If test code works, it is time to open the example code and compile. If gets a compilation error when trying to compile or upload code to the board check for errors in syntax, typos and more. Using the correct syntax is vital for making sure code compiles. When compilation fails, the IDE will present with the errors on its bottom part. However, the error messages generated by the Arduino IDE are limited in their description and therefore not always very helpful.

Figure 5:  
Arduino IDE with I<sup>2</sup>C Interface Code for AS7341 Sensors

```

I2C_Connection_Test_01 $

#include <Wire.h>
#define _i2cAddr (0x49)
void setup()
{
  // Initiate the Wire library and join the I2C bus as a master or slave
  Wire.begin();
  // communication with the host computer serial monitor
  Serial.begin(115200);
}
void loop()
{
writeRegister(byte (0x70), (0x8A)); //Device Config 1
writeRegister(byte (0x71), (0x02)); //Device Config 2
writeRegister(byte (0xB0), (0x02)); //Device Config 3
writeRegister(byte (0x88), (0x00)); //Device Config 4
writeRegister(byte (0x9A), (0x02)); //Device Config 5
writeRegister(byte (0x22), (0xCA)); //Int pin Enable
writeRegister(byte (0xD9), (0x00)); //Intergration TIME
writeRegister(byte (0xB9), (0x00)); //Gain IDRv
writeRegister(byte (0xDB), (0x80)); //Bank Mode
writeRegister(byte (0xDA), (0x80)); //WTIME
writeRegister(byte (0xF9), (0x04)); //Intterrupt and Polling enble
writeRegister(byte (0xFA), (0x83)); //Data Enable
writeRegister(byte (0xBA), (0x80)); //Auto Zero
writeRegister(byte (0xD3), (0x80)); //Temp Config
writeRegister(byte (0xEA), (0x03)); //LEN ON
writeRegister(byte (0x84), (0x01)); //LEN IND
writeRegister(byte (0xF8), (0x04)); //Intterrupt and Polling clear

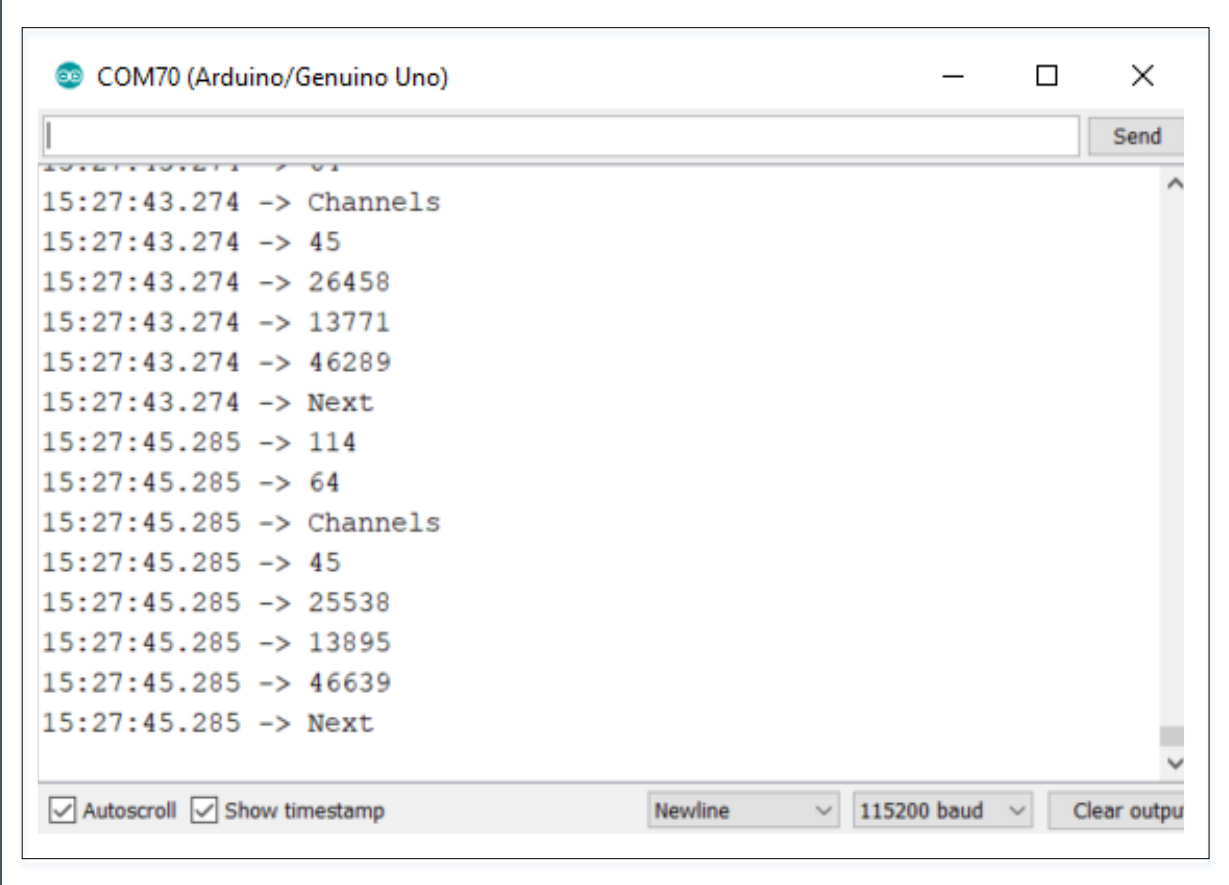
Serial.println(readRegister(byte(0x10))); //Device Identification
Serial.println(readRegister(byte(0x11))); //Device Version
Serial.println("Channels");
Serial.println(readTwoRegister1(byte(0xDC))); //N LOW/HIGH
Serial.println(readTwoRegister1(byte(0xDE))); //Y LOW/HIGH
Serial.println(readTwoRegister1(byte(0xEC))); //Z LOW/HIGH
Serial.println(readTwoRegister1(byte(0xEE))); //X LOW/HIGH
Serial.println("Next");
delay(2000);
}

```

Done uploading.

Sketch uses 4068 bytes (12%) of program storage space. Maximum is 32256 bytes.  
Global variables use 422 bytes (20%) of dynamic memory, leaving 1626 bytes for local variables. Maximum is 2048 bytes.

Figure 6:  
Arduino IDE with Serial Monitor Response after Executing the One of the Script



```
COM70 (Arduino/Genuino Uno)
15:27:43.274 -> Channels
15:27:43.274 -> 45
15:27:43.274 -> 26458
15:27:43.274 -> 13771
15:27:43.274 -> 46289
15:27:43.274 -> Next
15:27:45.285 -> 114
15:27:45.285 -> 64
15:27:45.285 -> Channels
15:27:45.285 -> 45
15:27:45.285 -> 25538
15:27:45.285 -> 13895
15:27:45.285 -> 46639
15:27:45.285 -> Next
```

The screenshot shows the Serial Monitor window for COM70 (Arduino/Genuino Uno). The window title bar includes standard window controls (minimize, maximize, close) and a 'Send' button. The main area displays two sets of sensor data output, each starting with a timestamp and the word 'Channels'. The first set of data is from 15:27:43.274 and includes values 45, 26458, 13771, and 46289, followed by 'Next'. The second set is from 15:27:45.285 and includes values 114, 64, 45, 25538, 13895, and 46639, followed by 'Next'. At the bottom, there are checkboxes for 'Autoscroll' and 'Show timestamp', a 'Newline' dropdown menu, a '115200 baud' dropdown menu, and a 'Clear output' button.

As outlined above, if the code fails when trying to run the sketch, need to start debugging the code. First, think and define which parameters to print and use the serial monitor to monitor them on screen.

Once uploaded to your Arduino, open up the serial console at baud speed of serial monitor defined in the code to begin the tester sketch and its results. When executing the sketch, user should see the following output on the serial monitor as shown Figure 6.

---

## 7 Revision Information

---

Changes from previous version to current revision v1-00	Page
Initial version	

- Page and figure numbers for the previous version may differ from page and figure numbers in the current revision.
- Correction of typographical errors is not explicitly mentioned.



## 8 Legal Information

### Copyrights & Disclaimer

Copyright ams AG, Tobelbader Strasse 30, 8141 Premstaetten, Austria-Europe. Trademarks Registered. All rights reserved. The material herein may not be reproduced, adapted, merged, translated, stored, or used without the prior written consent of the copyright owner.

Information in this document is believed to be accurate and reliable. However, ams AG does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Applications that are described herein are for illustrative purposes only. ams AG makes no representation or warranty that such applications will be appropriate for the specified use without further testing or modification. ams AG takes no responsibility for the design, operation and testing of the applications and end-products as well as assistance with the applications or end-product designs when using ams AG products. ams AG is not liable for the suitability and fit of ams AG products in applications and end-products planned.

ams AG shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interruption of business or indirect, special, incidental or consequential damages, of any kind, in connection with or arising out of the furnishing, performance or use of the technical data or applications described herein. No obligation or liability to recipient or any third party shall arise or flow out of ams AG rendering of technical or other services.

ams AG reserves the right to change information in this document at any time and without notice.

### RoHS Compliant & ams Green Statement

**RoHS Compliant:** The term RoHS compliant means that ams AG products fully comply with current RoHS directives. Our semiconductor products do not contain any chemicals for all 6 substance categories, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, RoHS compliant products are suitable for use in specified lead-free processes.

**ams Green (RoHS compliant and no Sb/Br):** ams Green defines that in addition to RoHS compliance, our products are free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material).

**Important Information:** The information provided in this statement represents ams AG knowledge and belief as of the date that it is provided. ams AG bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. ams AG has taken and continues to take reasonable steps to provide representative and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. ams AG and ams AG suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

### Headquarters

ams AG  
Tobelbader Strasse 30  
8141 Premstaetten  
Austria, Europe  
Tel: +43 (0) 3136 500 0

Please visit our website at [www.ams.com](http://www.ams.com)

Buy our products or get free samples online at [www.ams.com/Products](http://www.ams.com/Products)

Technical Support is available at [www.ams.com/Technical-Support](http://www.ams.com/Technical-Support)

Provide feedback about this document at [www.ams.com/Document-Feedback](http://www.ams.com/Document-Feedback)

For sales offices, distributors and representatives go to [www.ams.com/Contact](http://www.ams.com/Contact)

For further information and requests, e-mail us at [ams\\_sales@ams.com](mailto:ams_sales@ams.com)