AS7341-via µC

Arduino I2C interface to AS7341 Sensor

Content Guide

1 General Description 3

2 Prepare the Connection 3

3 Installation requirement 5

4 Creating an Arduino sketch with Wire Library 5

5 Structure of the Coding and functionalities 6

5.1 Steps to read out raw channel values 6

5.2 Steps in Flicker default detection in sensor 7

5.3 Measure raw channel values in SYNS MODE 8

5.4 Measure raw channel values in SYND MODE 9

5.5 Flicker Detection at 1000 and 1200 Hz 10

6 Code debugging and sensor reading using IDE 11

7 Contact Information 13

8 Copyrights & Disclaimer 14

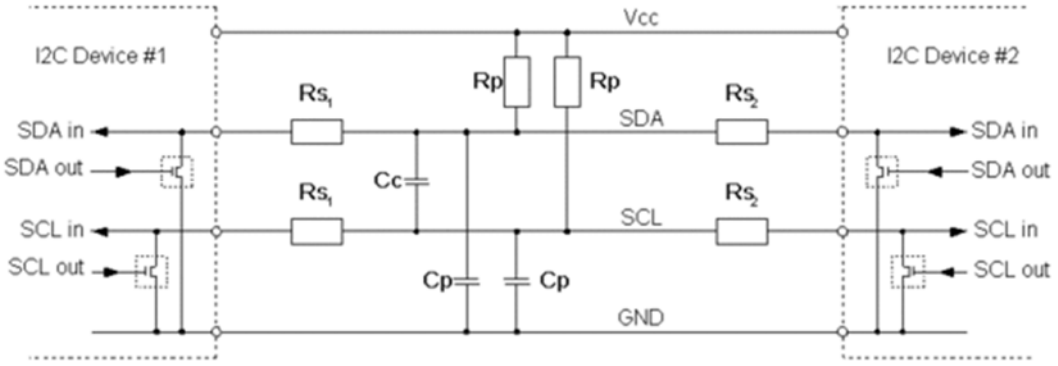
1 General Description

This “Hello World” - Application Note describes how to implement a connection between ams AS7341 sensor and a microcontroller via I2C Communication. This operates with an industrial-standard I2C connection. The I2C 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.

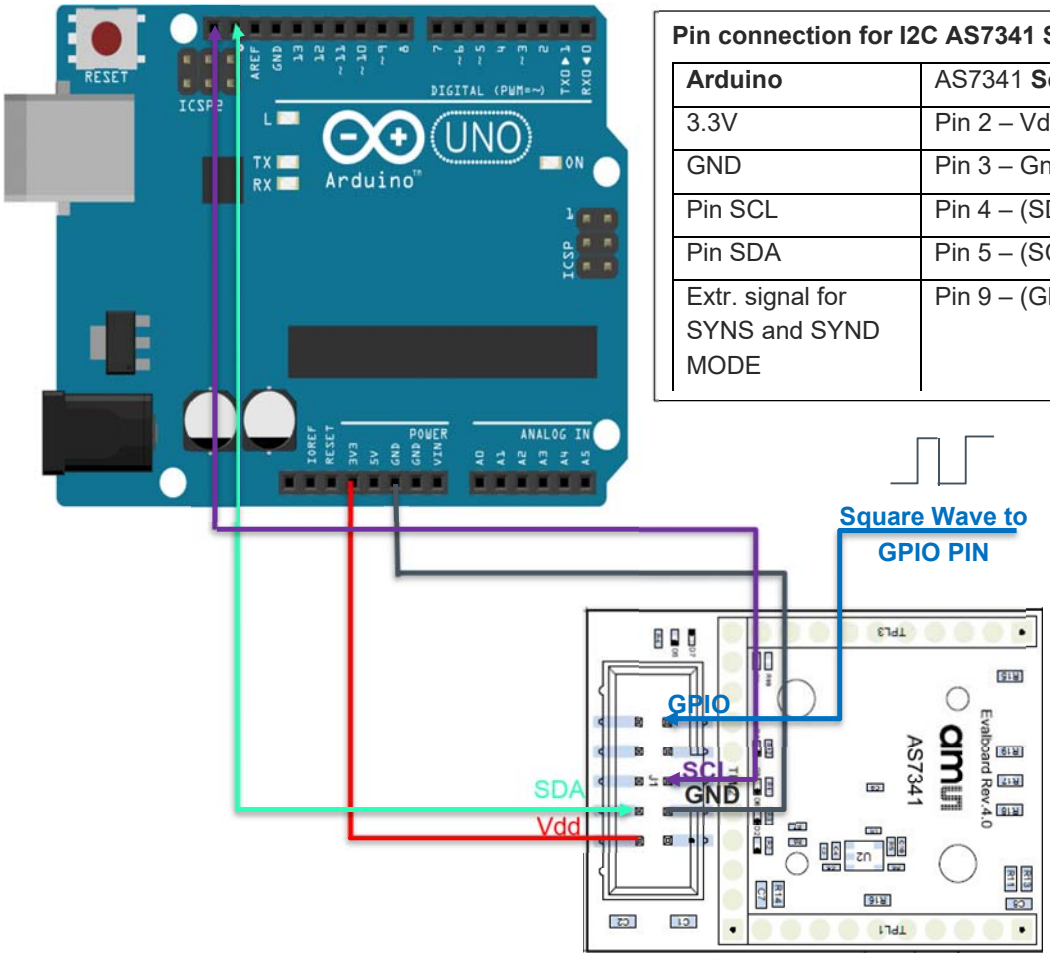
2 Prepare the Connection

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

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 I2C 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 I2C bus are active low. Commonly used values for the resistors are from 2K for higher speeds at about 400 kbps, to 10K for lower speed at about 100 kbps.



VCC	I2C supply voltage, typically ranging from 1.2 V to 3.6 V
GND	Common ground
SDA	Serial data (I2C data line)
SCL	Serial clock (I2C clock line)
Rp	Pull-up resistance (a.k.a. I2C termination)
Rs	Serial resistance
Cp	Wire capacitance
Cc	Cross channel capacitance



Pin connection for I2C AS7341 Sensor

Arduino	AS7341 **Sensor**
3.3V	Pin 2 – Vdd 3v3
GND	Pin 3 – Gnd
Pin SCL	Pin 4 – (SDA)
Pin SDA	Pin 5 – (SCL)
Extr. signal for SYNS and SYND MODE	Pin 9 – (GPIO)

Figure 1: Hardware connection of Arduino µC and AS7341 sensor

Insert the device you want to communicate with in the breadboard. Connect ground on the breadboard to ground from the microcontroller. The AS7341 sensor can be equipped with 3.3V power connection. Connect the Serial data pin of sensor (PIN 4) 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 1.

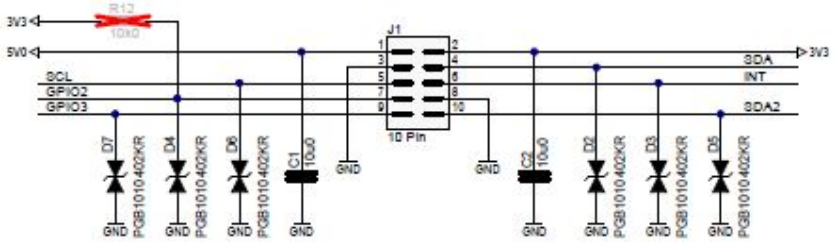


Figure 2: Jumper 1 pinning and its connections in AS7341

Note: Read the datasheet for the details regarding I2C communication

3 Installation requirement

Download the Arduino Software (IDE)

Get the latest version from the Arduino Software 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 finishes, 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.

5 Structure of the Coding and functionalities

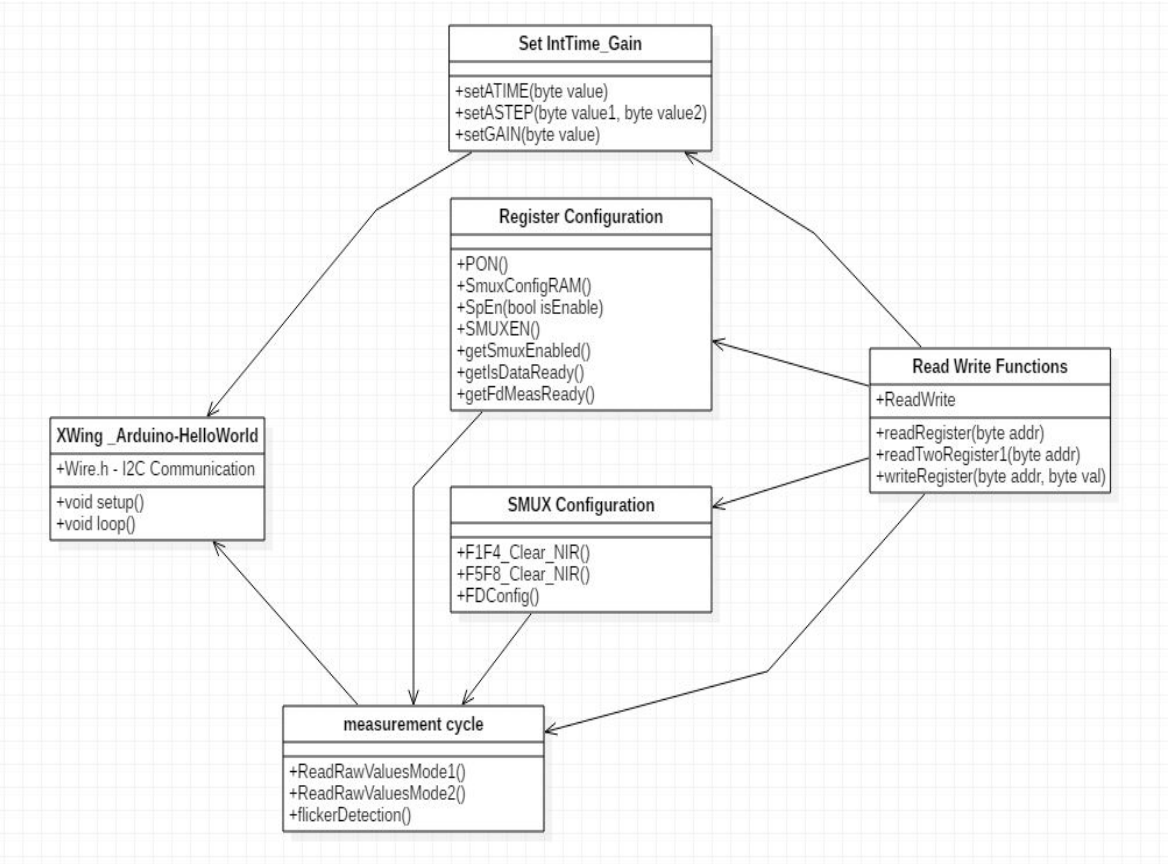


Figure 3: Structure of the code

Note: Please refer the code for detail description of each functionalities.

5.1 Steps to read out raw channel values

Considering a case of executing raw data measurement for channels F1, F2, F3, F4, NIR and Clear. In the code, ReadRawValuesMode1 () function does this job for details please check through the code script.

For reading out raw values, the following steps could be used:

- Setting the PON bit in Enable register 0x80 by the function PON()
- Write SMUX configuration from RAM to set SMUX chain registers (Write 0x10 to CFG6) by the function SmuxConfigRAM()
- Define an SMUX configuration to all the 20 registers for reading channels from F1-F4, Clear and NIR
- Start SMUX command: Enable the SMUXEN bit (bit 4) in register ENABLE with the function SMUXEN();
- Poll on the SMUXEN bit enabled in the previous step whether back to zero, if it is 0 SMUX command is started
- Now enable the SP_EN (spectral measurement enabled) bit on the chip (bit 1 in register ENABLE)
- Now when the AVALID bit in Status 2 Register 0xA3 is ready (high), raw data measurement cycle for 6 channels F1,F2,F3,F4,NIR,and Clear can be read by readTwoRegister1()
- These steps repeats for the measurement of another SMUX configuration for measure the raw data for channels F5, F6, F7, F8, NIR, and Clear. ReadRawValuesMode2() function defined in the code is defined to read out these channels

5.2 Steps in Flicker default detection in sensor

Similar to reading out the raw values, flicker detection also requires certain procedure and steps to follow. There are several methods to do. Here in this example, explaining the default flicker detection for 100 and 120 Hz in the AS7341 sensors.

Executing a flicker measurement cycle:-

- Firstly, all the bits of the ENABLE register (0x80) cleared by writing 0x00 to it.
- Setting the PON bit in Enable register 0x80 to ‘1’.
- Write SMUX configuration from RAM to set SMUX chain registers (Write 0x10 to CFG6) by the function SmuxConfigRAM ().
- Write new configuration to all the 20 registers for detecting Flicker by the function FDConfig (). Flicker detection is set to the ADC5 i.e. to the left of 0x13 register.
- Start SMUX command: Enable the SMUXEN bit (bit 4) in register ENABLE with the function SMUXEN()
- Poll on the SMUXEN bit enabled in the previous step whether back to zero, if it is 0 SMUX command is started
- Now enable the SP_EN (spectral measurement enabled) bit on the chip (bit 1 in register ENABLE)
- Functions for setting Flicker Sample, Flicker time, Flicker Gain (not implemented for default flicker detection in code) are optional. When not defined, these takes default values.
- Enable the Flicker detection via setting the fden bit in 0x80 register by writing value 0x41
- Reading the flicker status in FD_STATUS register in 0xDB register
- If the Flicker value in FD_STATUS register(0xDB) is 0x2C or 44 in decimal Flicker Detection measurement was finished and is valid but the frequency is not in range.

- If flicker value is 0x2D or 45, a flicker of 100 Hz is detected
- Whereas the flicker value is 0x2E or 46, a flicker of 120 Hz is detected

5.3 Measure raw channel values in SYNS MODE

The AS7341 device features three modes to perform a spectral measurement. The integration mode (INT_MODE) can be configured in register 0x70 (CONFIG). The SYNS Mode is configured by setting INT_MODE = 0x01 in Register CONFIG (0x70).

The GPIO synchronizes input to start/stop the spectral measurement in SYNS/SYND mode. The interrupt output pin INT can also be used to indicate the status (READY/BUSY) of the spectral measurement in mode SYNS and SYND.

In SYNS, Integration start with external Sync signal. Integration starts with rising/falling edge on pin GPIO. Integration Time set by register ATIME and ASTEP,

Where integration time = (ATIME + 1) * (ASTEP + 1) * 2.78µS

For reading out raw values using SYNS MODE, the following steps used:

- Setting the PON bit in Enable register 0x80 by the function PON()
- Disable SP_EN bit in Enable register 0x80 by SpEn() function
- Write SMUX configuration from RAM to set SMUX chain registers (Write 0x10 to CFG6) by the function SmuxConfigRAM()
- Define an SMUX configuration to all the 20 registers for reading channels, in the example code F5-F8, Clear and NIR channels are configured
- Start SMUX command: Enable the SMUXEN bit (bit 4) in register ENABLE with the function SMUXEN();
- Poll on the SMUXEN bit enabled in the previous step whether back to zero, if it is 0 SMUX command is started
- Function GPIO_MODE Enables the gpio_in_en (Bit 2) and gpio_out (Bit 1) in GPIO register 0xBE. The gpio_in_en indicates the GPIO pin accepts a non-floating input and gpio_out indicates the output state of the GPIO is directly active
- The RegBankConfig function sets reg_bank bit (4) to '1' for setting the 0x00-0x7f register to reg_bank register in CFG0 register (0xA9).

  REG_BANK Bit - 0: Register access to register 0x80 and above

  1: Register access to register 0x60 to 0x74

- INT_MODE() configures INT_MODE (Bit 1:0) bit in the CONFIG (0x70) register to 0x01

  0x00: SPM mode (spectral measurement, normal mode)

  0x01: SYNS mode

  0x02: reserved

  0x03: SYND mode
- In order to use the RAM Bank writing back the Reg_bank priority to RAM bank in CFG0 register (0xA9).

- Now enable the SP_EN (spectral measurement enabled) bit on the chip (bit 1 in register ENABLE)
- Now when the AVALID bit in Status 2 Register 0xA3 is ready (high), raw data measurement cycle for 6 channels F1,F2,F3,F4,NIR,and Clear can be read by readTwoRegister1()
- Sets the Atime and Astep for the integration time = (ATIME + 1) * (ASTEP + 1) * 2.78µS
- It is also recommended to set the Spectral Gain in CFG1 Register (0xAA) in [4:0] bit

5.4 Measure raw channel values in SYND MODE

The AS7341 device features three modes to perform a spectral measurement. The integration mode (INT_MODE) can be configured in register 0x70 (CONFIG). The SYND Mode is configured by setting INT_MODE = 0x03 in Register CONFIG (0x70).

The GPIO synchronizes input to start/stop the spectral measurement in SYNS/SYND mode. The interrupt output pin INT can also be used to indicate the status (READY/BUSY) of the spectral measurement in mode SYNS and SYND.

In SYND, Integration controlled by external Start and Stop. Integration controlled with rising/falling edge on pin GPIO and the register EDGE. If the number of edges defined in register EDGE is reached on pin GPIO, integration time is stopped. Actual integration time can be read out in ITIME Register (Address 0x63, 0x64, 0x65) and integration time is $\mathrm{Tint} = \mathrm{ITIME} \times 2{,}78\mu s$

For reading out raw values using SYND MODE, the following steps used:

- Setting the PON bit in Enable register 0x80 by the function PON()
- Disable SP_EN bit in Enable register 0x80 by SpEn() function
- Write SMUX configuration from RAM to set SMUX chain registers (Write 0x10 to CFG6) by the function SmuxConfigRAM()
- Define an SMUX configuration to all the 20 registers for reading channels, in the example code F1-F4, Clear and NIR channels are configured
- Start SMUX command: Enable the SMUXEN bit (bit 4) in register ENABLE with the function SMUXEN();
- Poll on the SMUXEN bit enabled in the previous step whether back to zero, if it is 0 SMUX command is started
- Function GPIO_MODE Enables the gpio_in_en (Bit 2) and gpio_out (Bit 1) in GPIO register 0xBE. The gpio_in_en indicates the GPIO pin accepts a non-floating input and gpio_out indicates the output state of the GPIO is directly active
- The RegBankConfig function sets reg_bank bit (4) to '1' for setting the 0x00-0x7f register to reg_bank register in CFG0 register (0xA9).

  REG_BANK Bit - 0: Register access to register 0x80 and above

  1: Register access to register 0x60 to 0x74
- INT_MODE() configures INT_MODE (Bit 1:0) bit in the CONFIG (0x70) register to 0x03

  0x00: SPM mode (spectral measurement, normal mode)

  0x01: SYNS mode

  0x02: reserved

0x03: SYND mode

- Number of falling/raising SYNC-edges between start and stop integration in SynD mode is set to register EDGE (0x72) set via setSynEdge (). Integration time is determined by the integration_time = (syn_edge+1) * sync period
- In order to use the RAM Bank writing back the Reg_bank priority to RAM bank in CFG0 register (0xA9).
- Now enable the SP_EN (spectral measurement enabled) bit on the chip (bit 1 in register ENABLE)
- Now when the AVALID bit in Status 2 Register 0xA3 is ready (high), raw data measurement cycle for 6 channels F1,F2,F3,F4,NIR,and Clear can be read by readTwoRegister1()
- Analog and digital saturation read in Status2 register (0xA3). ASAT_DIGITAL (4th bit) high indicates the Digital saturation and ASAT_ANALOG (3rd bit) high indicates the Analog saturation.
- Again, the RegBankConfig function sets reg_bank bit (4) to '1' for setting the 0x00-0x7f register to reg_bank register in CFG0 register (0xA9).
- Reading the ASTATUS, register (0x60 or 0x94) latches all 12 spectral data bytes to that status read. Then reading the bytes from register 0x60 to 0x6F consecutively for all channel measurement and iTIME.

5.5 Flicker Detection at 1000 and 1200 Hz

This is an example code shows the AS7241 Spectral Sensor I2C interface with Arduino µC for Flicker Detection. This section explains the configurations to detect 1.0kHz and 1.2kHz flicker - sample frequency fs = 5538Hz

For reading out Flickering for 1.0 kHz and 1.2 kHz, the following steps used:

- select RAM_BANK 0 which RAM bank to access in register addresses 0x00-0x7f
- The coefficient calculated are stored into the RAM bank 0 and RAM bank 1, they are used instead of 100Hz and 120Hz coefficients, 100Hz and 120Hz coefficients are the default flicker detection coefficients. Write new coefficients to detect the 1000Hz and 1200Hz
- Select RAM coefficients for flicker detection by setting fd_disable_constant_init to „1“ in FD_CFG0 register - 0xd7, select fd_disable_constant_init = 1 and fd_samples = 4
- In FD_CFG1 register - 0xd8, set the fd_time(7:0) = 0x40 for Integration time for flicker detection
- In FD_CFG2 register - 0xd9, set fd_dcr_filter_size=1 for Number of DC averaging filter size to use and fd_nr_data_sets(2:0)=5 for Number of coefficients to use
- In FD_CFG3 register - 0xda, set fd_gain=9 defines the gain during flicker detection
- In CFG9 register - 0xb2, set sien_fd=1 enables generation of sint interrupt as soon as flicker detection update happened
- In ENABLE register - 0x80 set fden=1 to enable Flicker Detect and pon=1 to enable Power on bit. This field activates the internal oscillator to permit functionality
- Finally, reading the flicker status in FD_STATUS register 0xDB
  - The value 44d= 0x2c=0b00101100 FD_STATUS (fd_measurement_valid=1 fd_120Hz_flicker_valid=1 fd_100Hz_flicker_valid=1)
  - The value 45d= 0x2d=0b00101101 FD_STATUS(fd_measurement_valid=1 fd_1200Hz_flicker_valid=1 fd_1000Hz_flicker_valid=1 fd_1000Hz_flicker)

- The value 46d =0x2e=0b00101110 FD_STATUS(fd_measurement_valid=1 fd_1200Hz_flicker_valid=1 fd_1000Hz_flicker_valid=1 fd_1200Hz_flicker)

6 Code debugging and sensor reading using IDE

One should start with the hard ware 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 4: Arduino IDE with I2C interface code for AS7341 Sensors

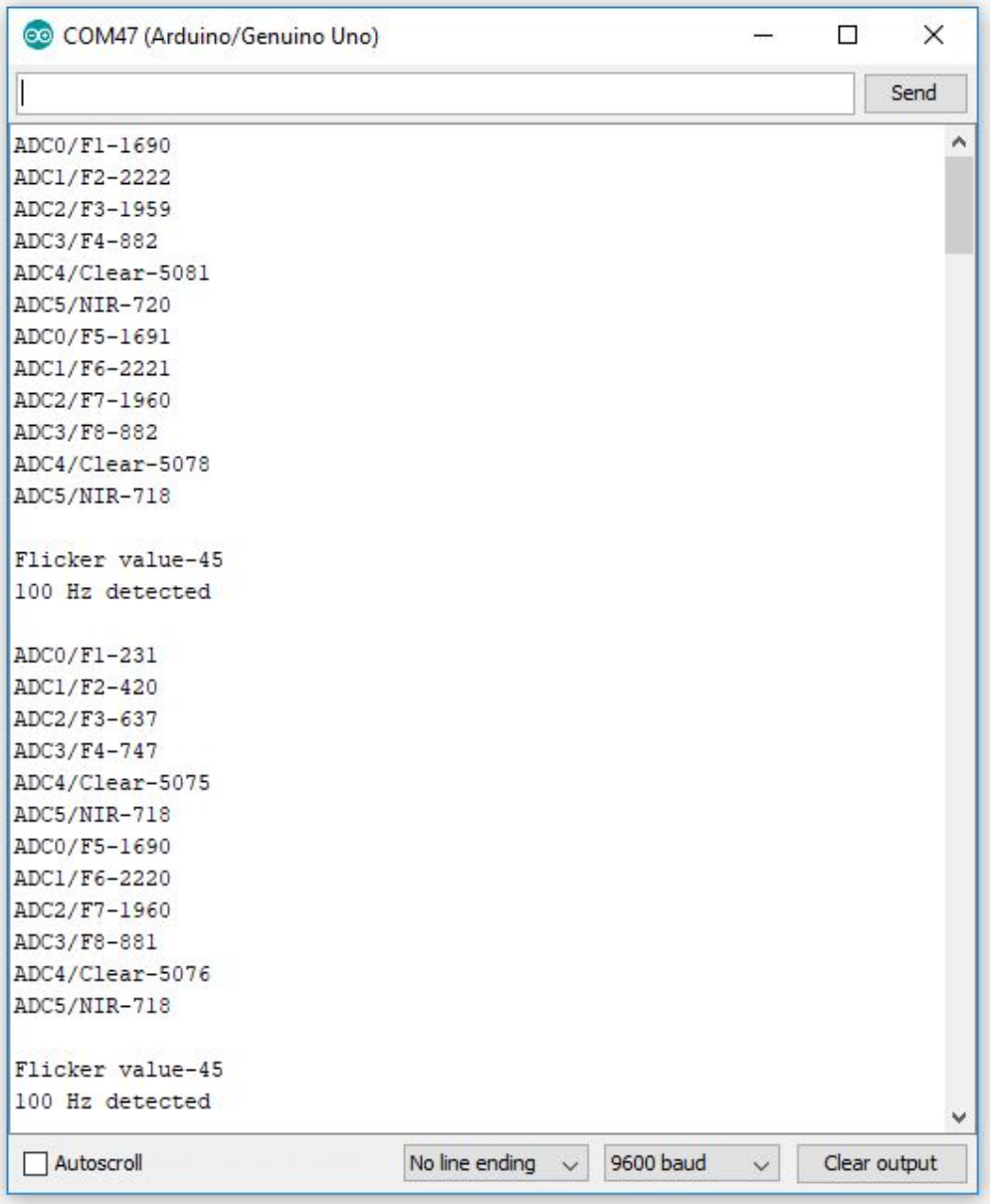


Figure 5: Arduino IDE with Serial Monitor Response after executing the one of the script

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 5.

7 Contact Information

Buy our products or get free samples online at:
www.ams.com/ICdirect

Technical Support is available at:
www.ams.com/Technical-Support

Provide feedback about this document at:
www.ams.com/Document-Feedback

For further information and requests, e-mail us at:
ams_sales@ams.com

For sales offices, distributors and representatives, please visit:
www.ams.com/contact

Headquarters
ams AG
Tobelbader Strasse 30
8141 Premstaetten
Austria, Europe

Tel: +43 (0) 3136 500 0
Website: www.ams.com

8 Copyrights & Disclaimer

Demo Kits, Evaluation Kits and Reference Designs are provided to recipient on an “as is” basis for demonstration and evaluation purposes only and are not considered to be finished end-products intended and fit for general consumer use, commercial applications and applications with special requirements such as but not limited to medical equipment or automotive applications. Demo Kits, Evaluation Kits and Reference Designs have not been tested for compliance with electromagnetic compatibility (EMC) standards and directives, unless otherwise specified. Demo Kits, Evaluation Kits and Reference Designs shall be used by qualified personnel only.

Ams AG reserves the right to change functionality and price of Demo Kits, Evaluation Kits and Reference Designs at any time and without notice.

Any express or implied warranties, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose are disclaimed. Any claims and demands and any direct, indirect, incidental, special, exemplary or consequential damages arising from the inadequacy of the provided Demo Kits, Evaluation Kits and Reference Designs or incurred losses of any kind (e.g. loss of use, data or profits or business interruption however caused) as a consequence of their use are excluded.

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 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.