

Az-Delivery

Wellcome!

Thank you for choosing our LCD screen from AZ-Delivery. These are available individually or bundled with I2C converter in green or blue, and in two sizes: 16x02 and 20x04. In the following pages we will explain how to set up and use the device.

Have fun!

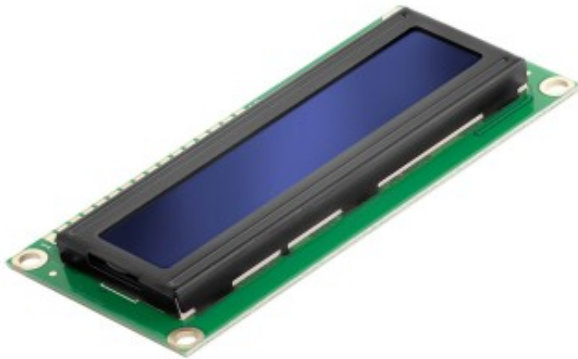


Table of contents

Introduction.....	3
Technical data.....	6
Pinout.....	8
How to set up the Arduino IDE.....	10
How to set up the Raspberry Pi and Python.....	15
Connection of the module with the microcontroller.....	16
Example sketch.....	18
Connection of the module with the microcontroller via I2C adapter.....	19
Arduino IDE library.....	20
Example sketch.....	21
Connecting the screen to the Raspberry Pi.....	24
Python-Skript.....	26
Connecting the screen to the Raspberry Pi via I2C adapter.....	34
Libraries and tools for Python.....	36
Python-Skript.....	38

Az-Delivery

Introduction

A liquid crystal display, or LCD, is a device that uses liquid crystals to display visual characters.

Liquid crystals do not emit light themselves. LCD screens use a backlight and liquid crystals to block light. When there is no current flowing through the liquid crystals, they are in a chaotic state. Light from the backlight passes through liquid crystals effortlessly. When current flows through liquid crystals, they arrange themselves in a uniform state. This creates a shadow on the screen that creates objects.

Simply put, LCD use liquid crystals to pass or block light coming from the backlight. Pixels are created in this way. They are combined to display any visible 2D objects on the screen. Each pixel area can be turned ON and OFF by supplying electricity through an on-board controller chip.

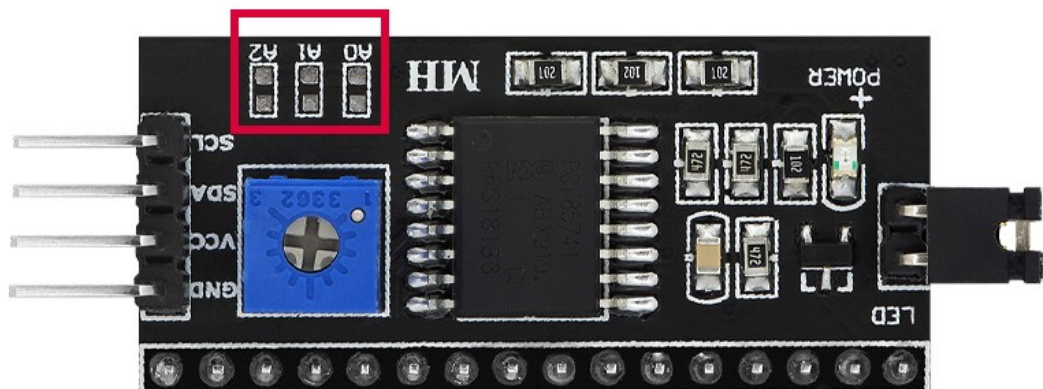
Each segment of the liquid crystal display represents a pixel, and must be controlled separately. For this reason, a special integrated circuit called a driver chip is required. The 16x02(20x04) LCD has a driver chip called "HD44780". To control the screen, the microcontroller must communicate with the driver chip. The driver chip uses a kind of SPI interface to communicate with a microcontroller.

The I2C adapter

The I2C adapter is a device that simplifies the connection between LCD screens and microcontrollers. It uses the I2C interface to communicate with the microcontroller. Multiple adapters can be connected to the same I2C interface. The adapter is compatible with LCD screens, with integrated HD44780 chips. The I2C adapter is compatible with both 16x02 LCD panels and 20x04 LCD panels offered by AZ-Delivery.

The I2C adapter comes with a predefined I2C address, which is 0x27. However, it can be changed by soldering the pads labeled A0, A1 and A2 on the adapter.

SOLDERING PADS (A0-A1)



Az-Delivery

How to set a specific I2C address of the adapter is shown in the following table:

PADS			I2C ADDRESS
A2	A1	A0	
C	C	C	0x20
C	C	O	0x21
O	C	O	0x22
C	O	O	0x23
O	C	C	0x24
O	C	O	0x25
O	O	C	0x26
O	O	O	0x27
O - Open			C - Closed

Technical data

» Operating voltage range:	3.3V to 5V
» Display area:	12 x 56mm
» LCD-type:	STN, positiv, transfective, green/blue
» Backlight:	ED, white
» Viewpoint:	180°
» Modes:	parallel (8-bit und 4-bit)
» Operating temperature:	-10 °C to 60 °C
» Dimensions:	36 x 80 x 12.5mm [1.4 x 3.1 x 0.5in]
» Interface:	I2C/parallel
» I2C-adress:	0x20 – 0x27
» Adjusting the contrast :	Potentiometer
» Setting the background.:	Jumper

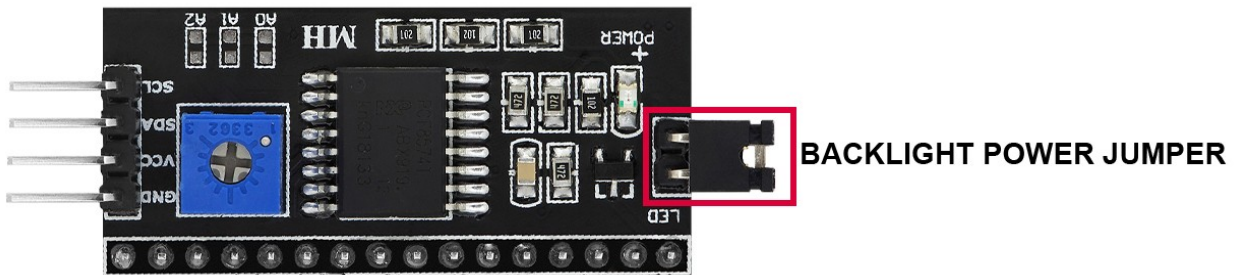
The voltage for the backlight is 5V DC. For logic functions the current consumption in operation is 1.5mA and for the backlight 30mA.

The resolution of the screen is 16x02, which means that characters can be displayed in 2 lines with 16 characters per line. Each character consists of 5x7 pixels.

To adjust the contrast of the display, the I2C adapter has a built-in potentiometer. Therefore a small screwdriver is needed for the adjustment.

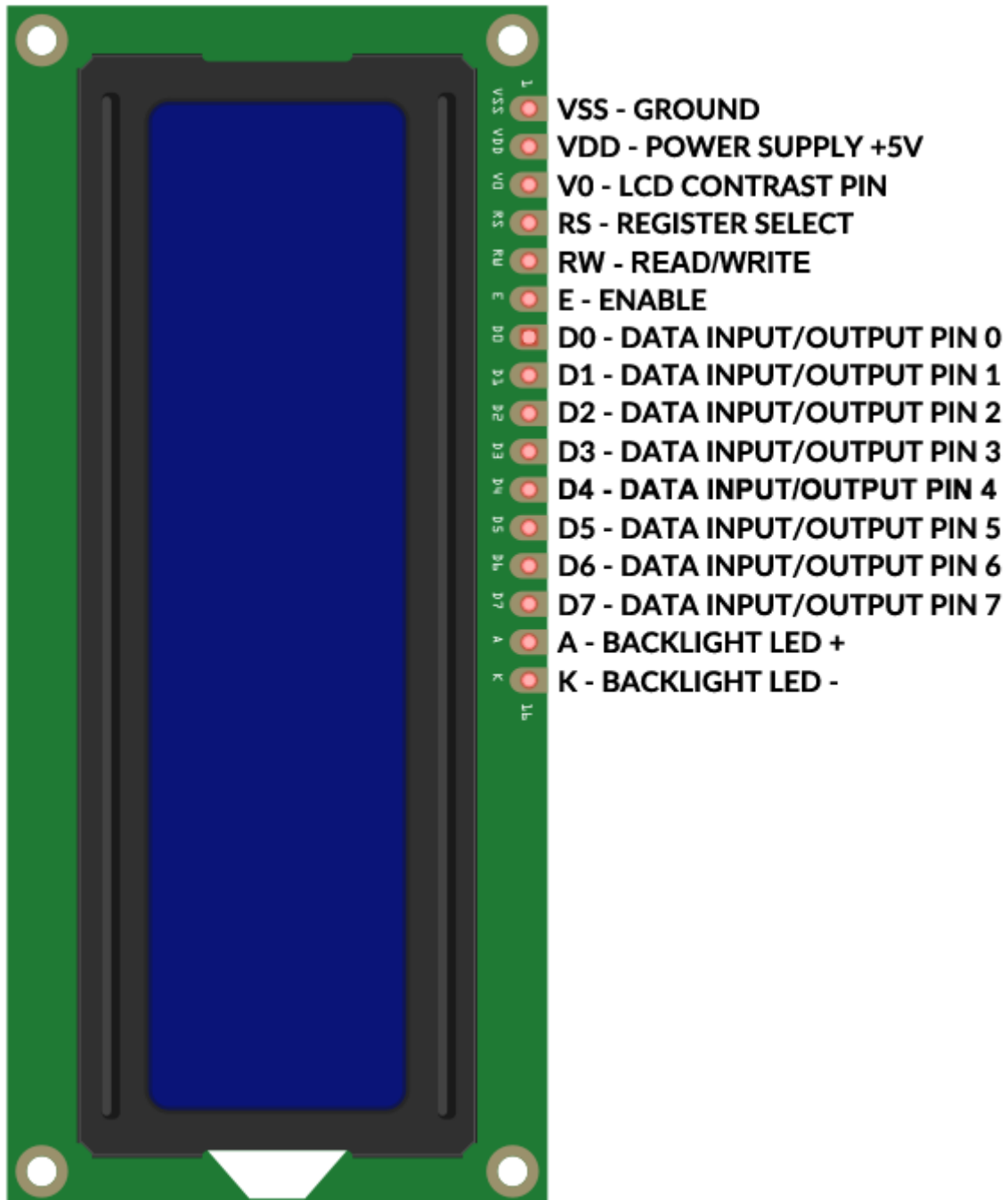
Az-Delivery

To control the backlight of the LCD screen, the I2C adapter has a power jumper for the backlight. The jumper is used to turn the backlight on/off. When the jumper is connected, it is used to connect the power supply for the backlight. When the jumper is unplugged, the backlight power supply is disconnected.



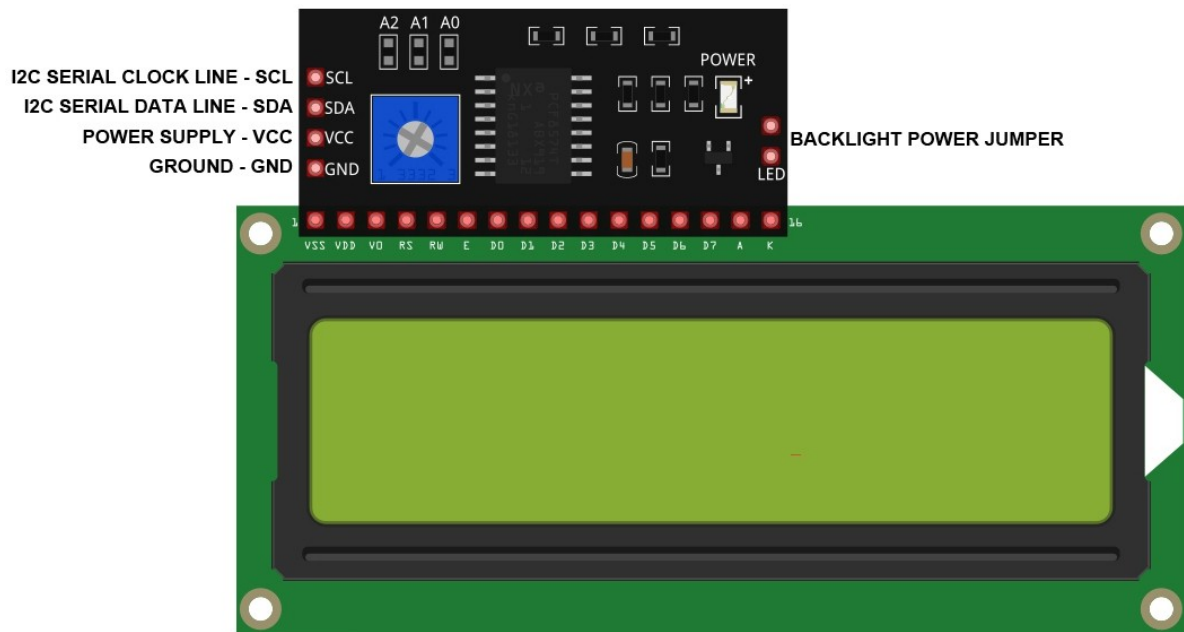
Pinout without I2C adapter

The 16x02 LCD has 16 pins. The pin assignment is as follows (also applies to 20x04):



Pinout with I2C adapter

The 16x02 LCD has 16 pins and the I2C adapter has 20. The adapter is connected to the LCD as follows (this also applies to the 20x04):



Note: It is necessary to connect the I2C adapter and LCD display exactly as shown above. If connected differently, the devices may be damaged.

AZ-Delivery

Note for the Raspberry Pi: The voltage of the TTL logic level of the I/O pins is 5V. To use the LCD panel and the I2C adapter with the Raspberry Pi, a logic level converter must be used. Otherwise, feeding the signal through the I/O pins of the module to the GPIO pins of the Raspberry Pi may cause damage. Therefore, use the [TXS0108E 8ch Logic Level Converter](#). offered by AZ-Delivery.

Az-Delivery

How to set up the Arduino IDE

If the Arduino IDE is not installed, follow the [link](#) and download the installation file for the operating system of your choice.

Download the Arduino IDE



The screenshot shows the Arduino IDE download page. On the left, there is a teal circular logo with a white infinity symbol containing a minus and a plus sign. To the right of the logo, the text reads: **ARDUINO 1.8.9**. Below this, it states: "The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software. This software can be used with any Arduino board. Refer to the [Getting Started](#) page for Installation instructions." On the right side of the page, there are several download options: "Windows Installer, for Windows XP and up" with a note "Windows ZIP file for non admin install"; "Windows app" with a "Get" button and the note "Requires Win 8.1 or 10"; "Mac OS X 10.8 Mountain Lion or newer"; "Linux 32 bits"; "Linux 64 bits"; "Linux ARM 32 bits"; and "Linux ARM 64 bits". At the bottom right, there are links for "Release Notes", "Source Code", and "Checksums (sha512)".

For Windows users: Double-click the downloaded .exe file and follow the instructions in the installation window.

Az-Delivery

For Linux users, download a file with the extension `.tar.xz`, which must be extracted. Once it is extracted, go to the extracted directory and open the terminal in that directory. Two `.sh` scripts need to be run, the first called **arduino-linux-setup.sh** and the second called `install.sh`.

To run the first script in the terminal, open the terminal in the extracted folder and run the following command:

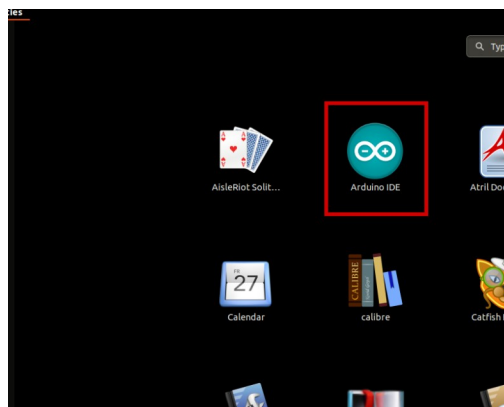
```
sh arduino-linux-setup.sh user_name
```

`user_name` - is the name of a superuser in the Linux operating system. A password for the superuser must be entered when starting the command. Wait a few minutes for the script to complete.

The second script, named `install.sh` script, must be used after the first script is installed. Run the following command in the terminal (extracted directory):

```
sh install.sh
```

After installing these scripts, go to All Apps where the Arduino IDE is installed.



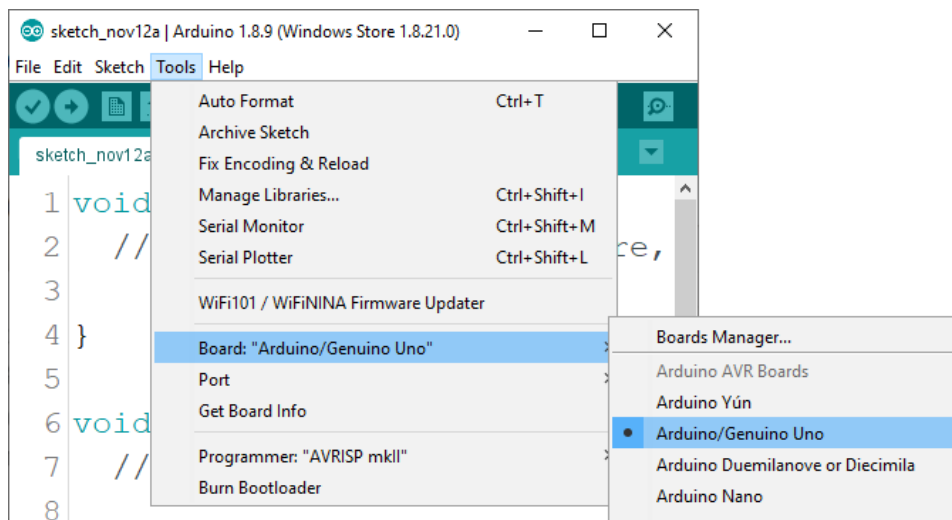
Az-Delivery

Almost all operating systems come with a pre-installed text editor (e.g. Windows with Notepad, Linux Ubuntu with Gedit, Linux Raspbian with Leafpad, etc.). All of these text editors are perfectly fine for the purpose of the eBook.

The first thing to check is whether your PC can recognize an Arduino board. Open the freshly installed Arduino IDE, and go to:

Tools > Board > {your board name here}

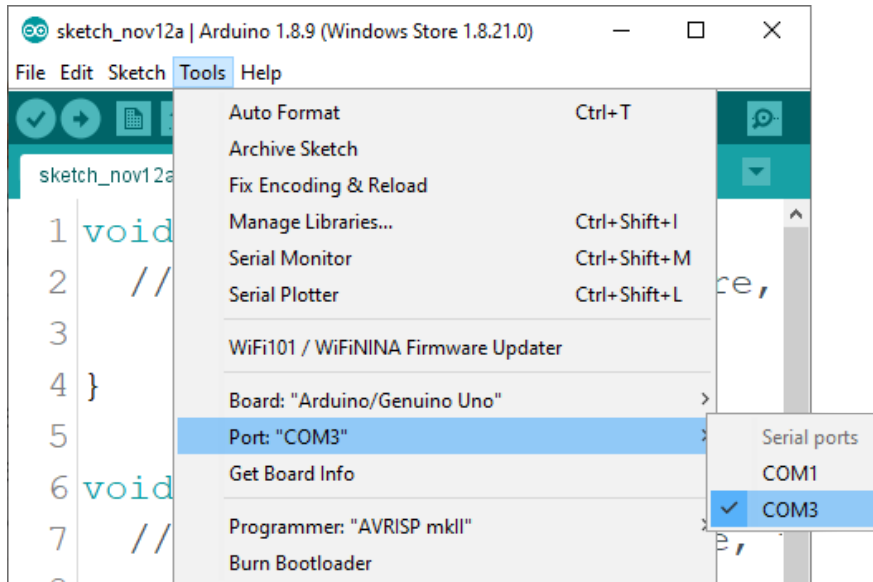
{your board name here} should be the Arduino/Genuino Uno, as it can be seen on the following picture:



The port where the microcontroller board is connected must be selected. Go to: Tools > Port > {port name goes here} and if the microcontroller board is connected to the USB port, the port name can be seen in the drop-down menu on the previous image.

Az-Delivery

If the Arduino IDE is used under Windows, the port names are as follows:



For Linux users, for example, the port name is `/dev/ttyUSBx`, where `x` is an integer between 0 and 9.



How to set up the Raspberry Pi and Python

For the Raspberry Pi, the operating system must first be installed, then everything must be set up so that it can be used in headless mode. Headless mode allows you to connect to the Raspberry Pi remotely without the need for a PC screen, mouse or keyboard. The only things used in this mode are the Raspberry Pi itself, the power supply and the internet connection. All of this is explained in detail in the free eBook:

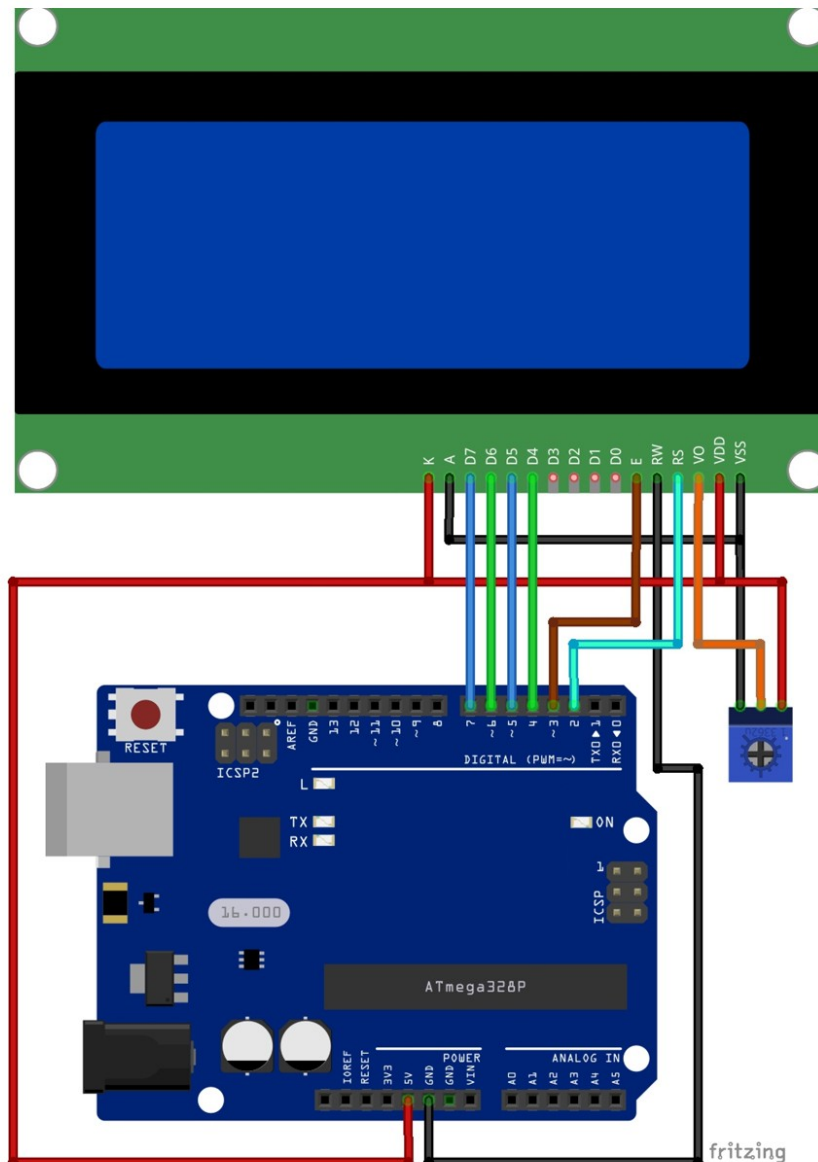
[*Raspberry Pi Quick Startup Guide*](#)

The Raspbian operating system comes with Python pre-installed.

Connection of the module with the microcontroller

Connect the screen to the microcontroller as shown below

(applies to 16x02 as well):



Az-Delivery

LCD Pin	MC Pin	Draht Farbe
VSS	GND	Schwarzer Draht
VDD	5V	Roter Draht
V0	Poti	Oranger Draht
RS	D2	Ochre Draht
RW	GND	Schwarzer Draht
E	D3	Ochre Draht
D4	D4	Grüner Draht
D5	D5	Blauer Draht
D6	D6	Grüner Draht
D7	D7	Blauer Draht
K	5V	Roter Draht
A	GND	Schwarzer Draht

AZ-Delivery

Example sketch

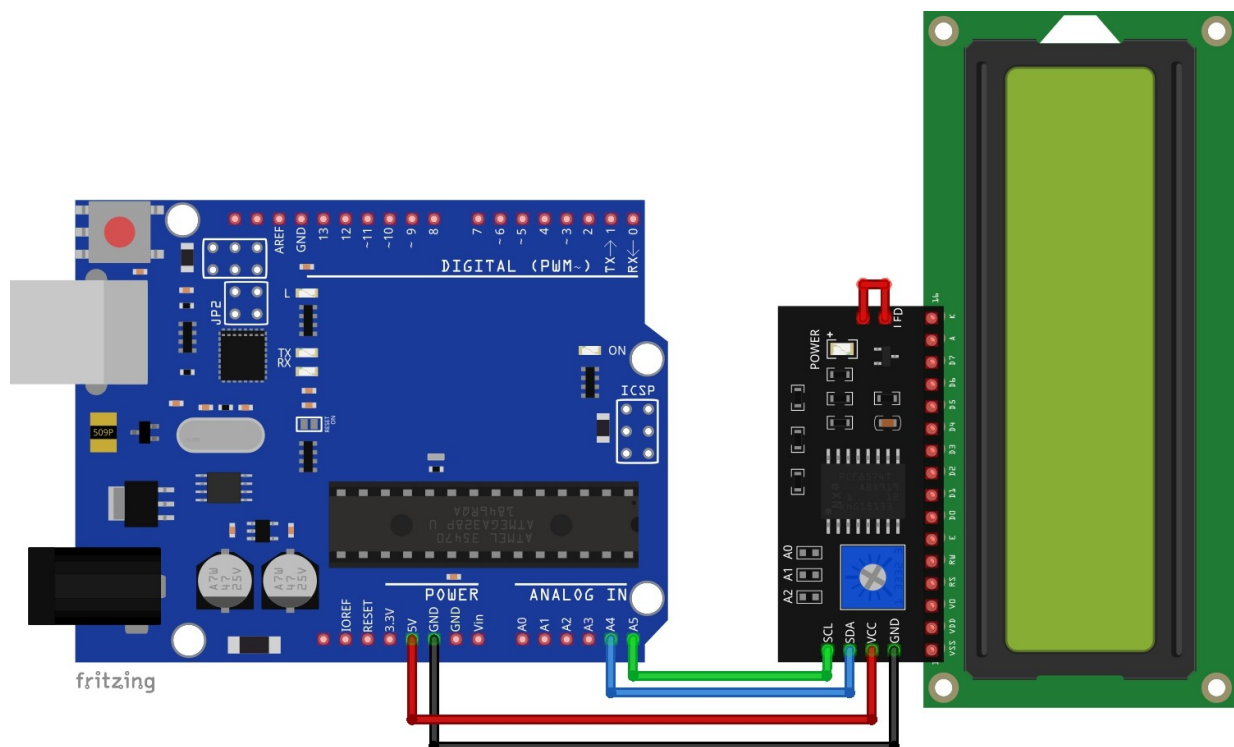
The following sketch example is a modified sketch from the Arduino IDE:

File > Examples > LiquidCrystal > HelloWorld

```
#include <LiquidCrystal.h>
const uint8_t rs = 2, en = 3, d4 = 4, d5 = 5, d6 = 6, d7 = 7;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
void setup() {
  lcd.begin(16, 2); // change to "20, 4" if 20x04 is used
  lcd.clear();
}
void loop() {
  lcd.setCursor(0, 0);
  lcd.print("AZ-Delivery");
  lcd.setCursor(0, 1);
  lcd.print(millis() / 1000);
}
```

Connection of the module to the microcontroller with I2C adapter

Connect the 16x02 screen and the I2C adapter to the microcontroller as shown below (applies to 20x04 as well):



I2C-Adapter Pin	MC Pin	Color
SCL	A5	green wire
SDA	A4	blue wire
VCC	5V	red wire
GND	GND	black wire

Az-Delivery

Library for Arduino IDE

To use the module with the Arduino IDE, it is recommended to download an external library for it. The library used in this eBook is called LiquidCrystal_I2C. To download it, click on this [link](#) and download the .zip file.

To include the library, go to the Arduino IDE and go to:

Sketch > Include Library > Add .ZIP Library

and when a new window opens, locate and select the downloaded .zip file.

AZ-Delivery

Example sketch

The following sketch example is a modified sketch from the Arduino IDE:
File > Examples > LiquidCrystal > HelloWorld

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd = LiquidCrystal_I2C(0x27, 16, 2);
    //change to "20, 4" if 20x04 is used
void setup() {
  lcd.init();
  lcd.backlight();
  delay(250);
  lcd.noBacklight();
  delay(1000);
  lcd.backlight();
  delay(1000);
}

void loop() {
  lcd.setCursor(0, 0);
  lcd.print("AZ-Delivery");
  lcd.setCursor(0, 1);
  lcd.print(millis() / 1000);
  delay(100);
}
```

Az-Delivery

The sketch starts with the inclusion of libraries named Wire and LiquidCrystal_I2C.

Then an object named lcd is created. The object represents the display itself, and to create this object, the following line of code is used:

```
LiquidCrystal_I2C lcd = LiquidCrystal_I2C(0x27, 16, 2);
```

Where 0x27 is the I2C address of the I2C adapter. 16 is the number of characters per line and 2 is the number of lines.

In the setup() function, the lcd object is initialized with the following line of code: lcd.init();

At the end of the setup() function, the backlight is tested by turning it OFF and ON with a delay of 1000ms (1s).

In the loop() function, two predefined functions from the LiquidCrystal_I2C library are used.

AZ-Delivery

The first function is called `setCursor()`. The function has two arguments and returns no value. The values of the arguments are integers. The first number represents the Y position of the cursor, with values in the range 0 to 1, where 0 represents the first line and 1 represents the second line of the screen. The second argument represents the X position of the cursor, with values ranging from 0 to 15, where 0 reflects the first column and 15 the last column of the screen.

The function must be used before the `print()` function. To show the `print()` function where to display the text. If you do not use the `setCursor()` function, the `print()` function will display the text at position (0, 0).

The `print()` function has one argument and does not return a value. The argument represents the text, a string value, that will be displayed on the screen.

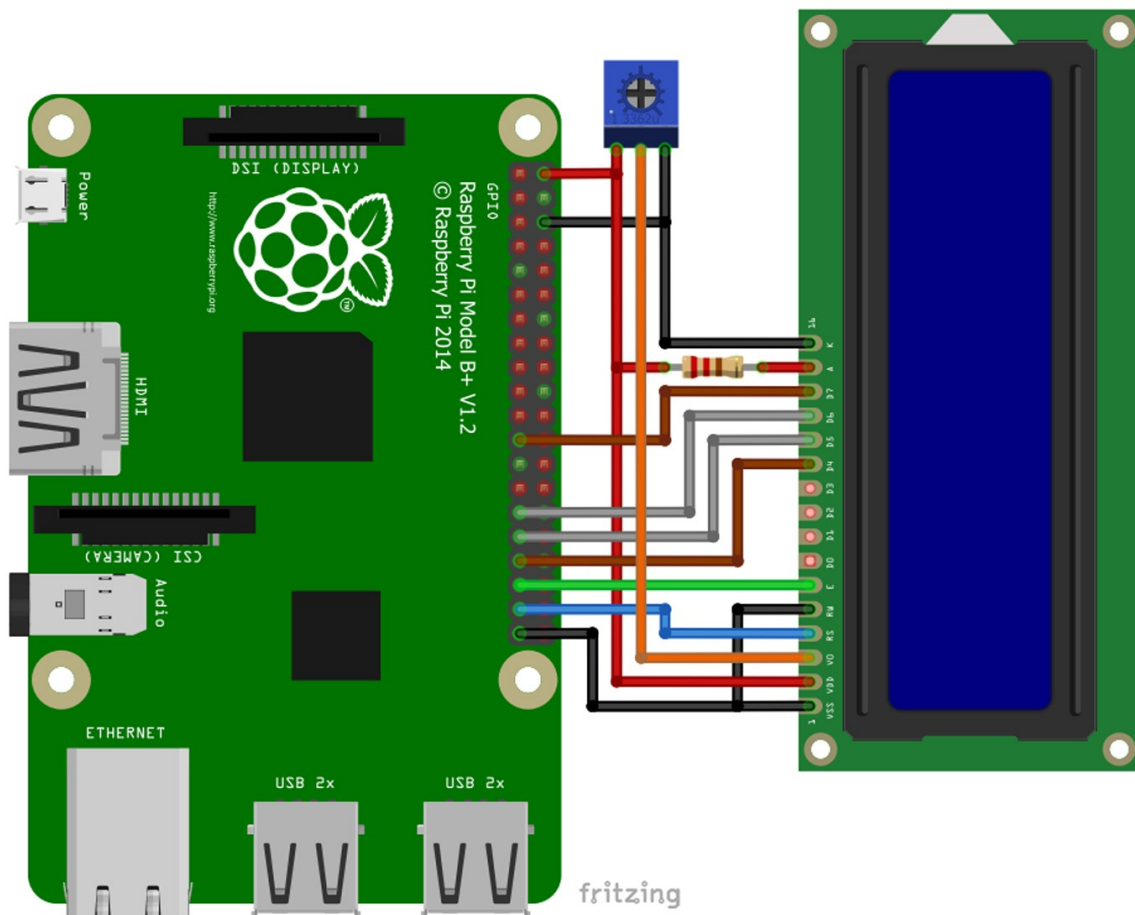
In the `loop()` function, first the cursor is placed on the first line and then the `print()` function displays the message AZ-Delivery. Then the cursor is placed on the second line and the number of seconds since the microcontroller was last powered up or reset is displayed.

NOTE: If your display does not show anything, you have to turn the potentiometer on the I2C adapter.

Connecting the screen to the Raspberry Pi

Connect the module to the Raspberry Pi as shown below (applies to 20x04 as well):

**16X02 Blue LCD Screen
Connection diagram with Raspberry Pi**



Az-Delivery

Screen Pin	Raspberry Pi Pin	Physical Pin	wire color
VSS	GND	6	black wire
VDD	5V	2	red wire
RS	GPIO26	37	blue wire
RW	GND	39	black wire
E	GPIO19	35	green wire
D4	GPIO13	33	brown wire
D5	GPIO6	31	gray wire
D6	GPIO5	29	gray wire
D7	GPIO11	23	brown wire
K	GND	6	red wire
A	5V, via 220Ω resistor	2	orange wire
V0	potentiometer center pin		

Potentiometer			
GND	right Pin	6	black wire
5V	left Pin	2	red wire

Az-Delivery

Python-Skript

Two scripts are created, one for all functions and the other to use these functions. Below you will find the code for the first script:

```
import RPi.GPIO as GPIO
import time

LCD_RS = 26
LCD_E  = 19
LCD_D4 = 13
LCD_D5 = 6
LCD_D6 = 5
LCD_D7 = 11

# Define some device constants
LCD_WIDTH = 16      # Maximum characters per line, change to "20"if
20x04
LCD_CHR = True
LCD_CMD = False

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005
```

Az-Delivery

```
# one tab
def lcd_init(RS, E, D4, D5, D6, D7):
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    global LCD_RS, LCD_E, LCD_D4, LCD_D5, LCD_D6, LCD_D7
    LCD_RS = RS
    LCD_E = E
    LCD_D4 = D4
    LCD_D5 = D5
    LCD_D6 = D6
    LCD_D7 = D7
    GPIO.setup(LCD_E, GPIO.OUT) # E
    GPIO.setup(LCD_RS, GPIO.OUT) # RS
    GPIO.setup(LCD_D4, GPIO.OUT) # DB4
    GPIO.setup(LCD_D5, GPIO.OUT) # DB5
    GPIO.setup(LCD_D6, GPIO.OUT) # DB6
    GPIO.setup(LCD_D7, GPIO.OUT) # DB7
    # Initialise display
    lcd_byte(0x33, LCD_CMD) # 110011 Initialise
    # 110010 Initialise
    lcd_byte(0x32, LCD_CMD)
    # 000110 Cursor move direction
    lcd_byte(0x06, LCD_CMD)
    # 001100 Display On,Cursor Off, Blink Off
    lcd_byte(0x0C, LCD_CMD)
    # 101000 Data length, number of lines, font size
    lcd_byte(0x28, LCD_CMD)
    # 000001 Clear display
    lcd_byte(0x01, LCD_CMD)
    time.sleep(E_DELAY)
```

Az-Delivery

```
# one tab
def lcd_byte(bits, mode):
    GPIO.output(LCD_RS, mode) # RS
    GPIO.output(LCD_D4, False)
    GPIO.output(LCD_D5, False)
    GPIO.output(LCD_D6, False)
    GPIO.output(LCD_D7, False)
    if bits & 0x10 == 0x10:
        GPIO.output(LCD_D4, True)
    if bits & 0x20 == 0x20:
        GPIO.output(LCD_D5, True)
    if bits & 0x40 == 0x40:
        GPIO.output(LCD_D6, True)
    if bits & 0x80 == 0x80:
        GPIO.output(LCD_D7, True)
    lcd_toggle_enable() # Toggle 'Enable' pin
    GPIO.output(LCD_D4, False)
    GPIO.output(LCD_D5, False)
    GPIO.output(LCD_D6, False)
    GPIO.output(LCD_D7, False)
    if bits & 0x01 == 0x01:
        GPIO.output(LCD_D4, True)
    if bits & 0x02 == 0x02:
        GPIO.output(LCD_D5, True)
    if bits & 0x04 == 0x04:
        GPIO.output(LCD_D6, True)
    if bits & 0x08 == 0x08:
        GPIO.output(LCD_D7, True)
    # Toggle 'Enable' pin
    lcd_toggle_enable()
```

Az-Delivery

```
# one tab
def lcd_toggle_enable():
    # Toggle enable
    time.sleep(E_DELAY)
    GPIO.output(LCD_E, True)
    time.sleep(E_PULSE)
    GPIO.output(LCD_E, False)
    time.sleep(E_DELAY)

def lcd_string(message, line):
    # Send string to display
    LCD_LINE_1 = 0x80
    LCD_LINE_2 = 0xC0
    message = message.ljust(LCD_WIDTH, " ")
    if line == 0:
        lcd_byte(LCD_LINE_1, LCD_CMD)
    elif line == 1:
        lcd_byte(LCD_LINE_2, LCD_CMD)
    else:
        print('This lcd has two lines, line 0 and line 1!')
    for i in range(LCD_WIDTH):
        lcd_byte(ord(message[i]), LCD_CHR)

def lcd_clear():
    lcd_byte(0x01, LCD_CMD)
```

Save the script under the name *lcd16x02.py*.

The code in the script is a modified code from the script at [link](#):

AZ-Delivery

Below you will find the code for the main script:

```
import lcd16x02
from time import sleep
LCD_RS = 26
LCD_E  = 19
LCD_D4 = 13
LCD_D5 = 6
LCD_D6 = 5
LCD_D7 = 11
# Initialise display
lcd16x02.lcd_init(LCD_RS, LCD_E, LCD_D4, LCD_D5, LCD_D6, LCD_D7)
i = 0
print('[Press CTRL + C to end the script!]' )
try:
    lcd16x02.lcd_string('AZ-Delivery', 0)
    print('AZ-Delivery')
    print('Printing variable on the LCD...')
    while True:
        lcd16x02.lcd_string('{}'.format(i), 1)
        i+=1
        sleep(0.001) # 1 millisecond delay

except KeyboardInterrupt:
    print('Script end!')

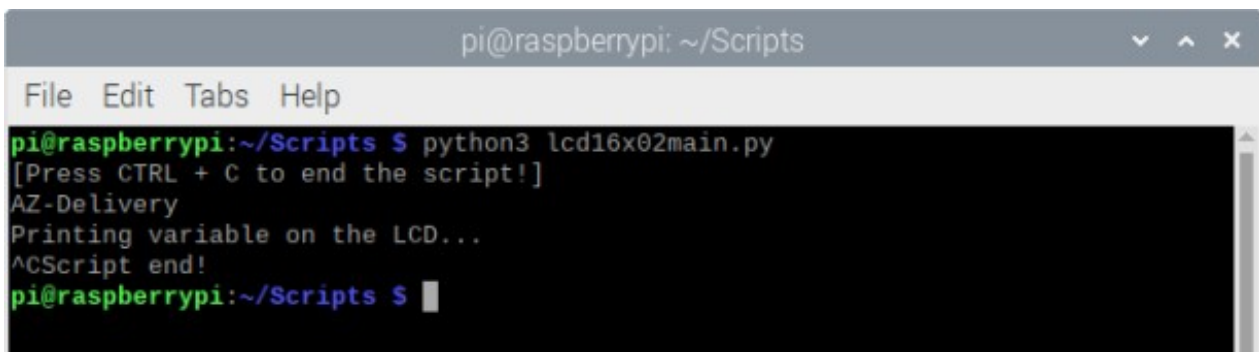
finally:
    lcd16x02.lcd_clear()
```

AZ-Delivery

Save the script under the name `lcd16x02main.py` in the same directory as the previous script. To run the script, open the terminal in the directory where the script is stored and run the following command:

```
python3 lcd16x02main.py
```

The output should look like this:

A terminal window titled 'pi@raspberrypi: ~/Scripts' with a menu bar containing 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'python3 lcd16x02main.py' being executed. The output is: '[Press CTRL + C to end the script!]', 'AZ-Delivery', 'Printing variable on the LCD...', and '^CScript end!'. The prompt returns to 'pi@raspberrypi:~/Scripts \$' with a cursor.

```
pi@raspberrypi:~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 lcd16x02main.py
[Press CTRL + C to end the script!]
AZ-Delivery
Printing variable on the LCD...
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script, press 'Ctrl + C' on the keyboard.

AZ-Delivery

The first script is used to create all the functions to control the LCD screen, which is not covered in this eBook. Only the main function of the script is explained.

The main script starts with importing the first script and importing the sleep function from the time library.

Then, six variables are defined to represent pins of the screen that are connected to the pins of the Raspberry Pi.

Next, the screen is initialized with the following line of code:

```
lcd16x02.lcd_init(LCD_RS, LCD_E, LCD_D4, LCD_D5, LCD_D6, LCD_D7).
```

Then the variable "i" is created and initialized with the value zero. This is used to indicate changing data on the display.

Then a try-except-finally code block is created. In the try block, the message AZ-Delivery is first displayed on the first line of the screen, and then an indefinite loop block (while True:) is created. In it, the variable i is timed on the second line of the display and the value of the variable i is incremented by 1. Between each repetition of the indefinite loop block there is a pause of one millisecond (sleep(0.001)).

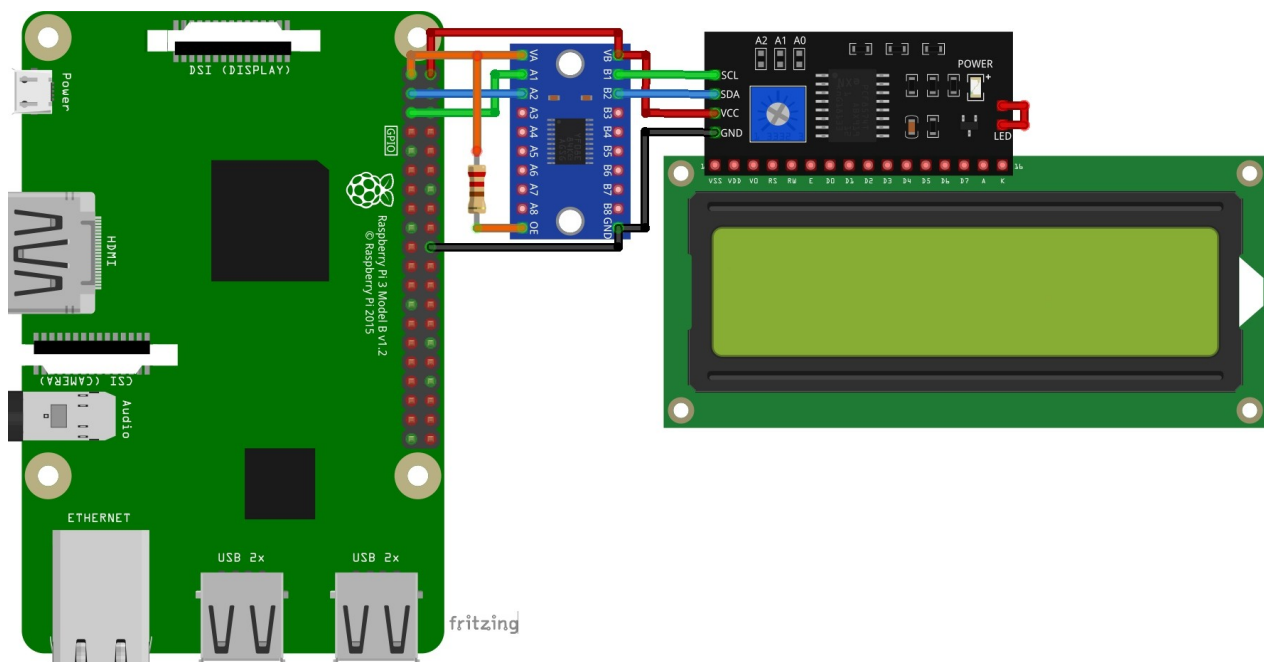
Az-Delivery

The except code block is executed with Ctrl+C. This is called KeyboardInterrupt. When this code block has been executed, the message Script end! is displayed in the terminal.

The finally code block is executed after the script. When the finally code block has been executed, the `lcd_clear()` function is called, which clears the screen's data buffer.

Connecting the screen to the Raspberry Pi with I2C adapter

Connect the 16x02 screen and the I2C adapter to the Raspberry Pi as shown below(applies the same to 20x04):



here the logic level converter must be used, because the I2C adapter works only in the 5V range. The logic level converter used in this eBook is called [TXS0108E Logic Level Converter](#).

Connect the I2C adapter to the LCD panel as shown in the connection diagram. Make sure that the integrated backlight jumper is connected (**Red wire**, the right side of the I2C adapter on the connection diagram).

Az-Delivery

I2C-Adapter Pin	LLC Pin	Color
SCL	B1	green wire
SDA	B2	blue wire
VCC	VB	red wire
GND	GND	black wire

LLC Pin	Raspberry Pi Pin	Physischer Pin	color
VA	3.3V	1	orange wire
A1	GPIO3	5	green wire
A2	GPIO2	3	blue wire
OE	3.3V (via resistor)	1	orange wire
GND	GND	20	black wire
VB	5V	2	red wire

Az-Delivery

Libraries and Tools for Python

To use the screen with the Raspberry Pi, it is recommended to download an external library. To download it, go to the following [link](#) and download the script `lcd_class_i2c.py`. Save the script in the same directory where the script of the next chapter is saved.

The `lcd_class_i2c.py` script uses the `smbus` library for Python. If it is not installed yet, open the terminal and execute the following commands one after the other:

```
sudo apt-get update
```

```
sudo apt-get install python3-smbus python3-dev
```

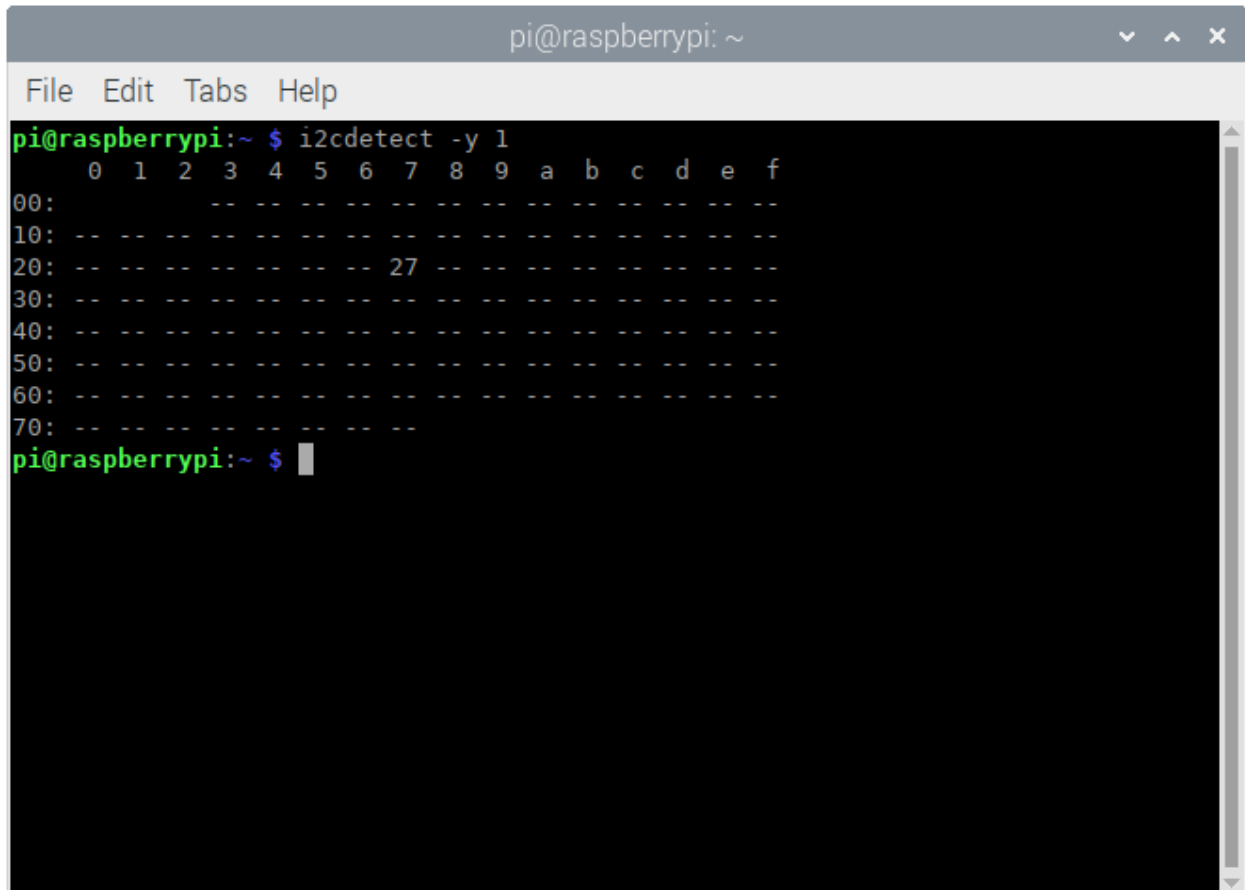
To determine the I2C address of the I2C adapter, the `i2c-tools` must be installed. If it is not already installed, run the following command in the terminal:

```
sudo apt-get install i2c-tools
```

Az-Delivery

To determine the I2C address of the I2C adapter, execute the following command:

```
i2cdetect -y 1
```



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ i2cdetect -y 1  
  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
20:  --  --  --  --  --  --  27  --  --  --  --  --  --  --  --  
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
pi@raspberrypi:~ $
```

where 0x27 is the I2C address of the I2C adapter.

Az-Delivery

Python-Skript

Below you will find the code for the main script:

```
import lcd_class_i2c as LCD
import time

I2C_ADDR = 0x27
LINE_WIDTH = 16

screen = LCD.lcd(line_width=LINE_WIDTH, i2c_address=I2C_ADDR)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        print('Printing messages on the screen')
        screen.lcd_print('AZ-DELIVERY', 'LINE_1', 'CENTER')
        time.sleep(1) # 3 second delay

        print('Printing variable on the screen')
        for i in range(100):
            if i < 10:
                screen.lcd_print('0{}'.format(i), 'LINE_2', 'CENTER')
            else:
                screen.lcd_print('{}'.format(i), 'LINE_2', 'CENTER')
            time.sleep(0.00001)
```

Az-Delivery

```
# one tab
    print('Testing backlight')
    time.sleep(1)
    screen.backlight('OFF')
    time.sleep(1)
    screen.backlight('ON')
    time.sleep(1)
    print('Clearing the screen\n')
    # Blank display
    screen.clear_screen()
    time.sleep(1)

except KeyboardInterrupt:
    print('\nScript end!')

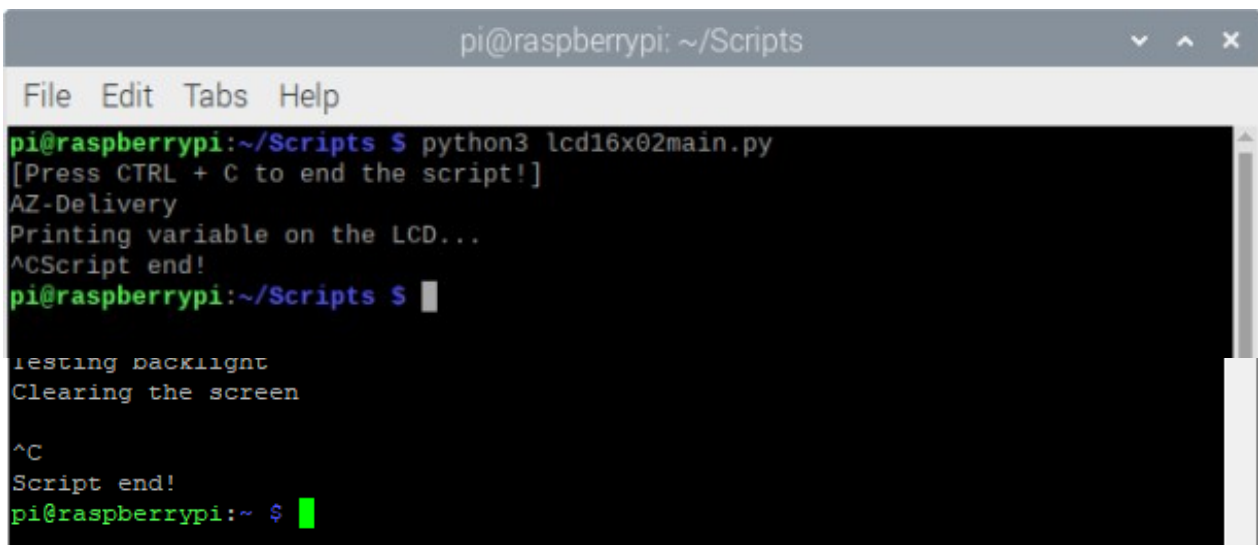
finally:
    screen.clear_screen()
```

AZ-Delivery

Save the script under the name `lcd16x02i2c.py` in the same directory as the `lcd_class_i2c.py` script. To run the script, open the terminal in the directory where the scripts are saved and run the following command:

```
python3 lcd16x02main.py
```

The output should look like this:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 lcd16x02main.py
[Press CTRL + C to end the script!]
AZ-Delivery
Printing variable on the LCD...
^CScript end!
pi@raspberrypi:~/Scripts $

Testing backlight
Clearing the screen

^C
Script end!
pi@raspberrypi:~ $
```

To stop the script, press 'Ctrl + C' on the keyboard.

Az-Delivery

The first script is used to create all the functions to control the LCD screen, which is not covered in this eBook.

The main script starts by importing the first script `lcd_class_i2c.py` and the `time` library.

Next, the screen object is created. This is used to control the screen. We create the object with the following line of code:

```
screen=LCD.lcd(line_width=LINE_WIDTH, i2c_address=I2C_ADDR)
```

Where the `lcd()` constructor has two arguments. The first argument is called `line_width` and represents the number of characters per line of the screen. The `lcd_class_i2c.py` script can be used for 16x02 LCD screens as well as for 20x04 screens. So the values passed to the `line_width` argument are 16 or 20. Any other value will result in an error, and the script sets the value of this argument to 16. The second argument, called `i2c_address` represents the I2C address of the I2C adapter, which in this case is 0x27.

AZ-Delivery

Then a try-except-finally code block is created. In the try block, an Indefinite loop block (while True:) is created. In the Indefinite loop block, first the message AZ-Delivery is displayed in the first line of the screen. The message is positioned in the middle of the line. This is done with the following line of code:

```
screen lcd_print('AZ-DELIVERY', 'LINE_1', 'CENTER')
```

Where the lcd_print() function is used. This function displays a message on the screen. It has three arguments and returns no value. The second and third arguments are optional. The first argument is a string representing the message displayed on the screen. The second argument, also a string, represents the line on which the message is displayed. The values for this argument are: LINE_1 or LINE_2. The default value is LINE_1, which is selected if the argument is not used. Any other value will result in an error and the value will be set to the default LINE_1. The third argument, also a string, represents the alignment of the text in the line. The values of this argument are: LEFT, CENTER or RIGHT. The default value is LEFT, which is selected if the argument is not used. Any other value will result in an error and the value will also be set to the default value LEFT.

To display a variable on the screen, use the following lines of code:

```
my_var = 10  
screen lcd_print('{}' .format(my_var))
```

Az-Delivery

To control the backlight of the screen, the `backlight()` function is used. This function has one argument and does not return a value. The value of the argument is a string that can have only two values: ON or OFF. To turn the screen ON, the following line of code is used:

```
screen.backlight('ON')
```

To turn it off, use the following:

```
screen.backlight('OFF')
```

To clear the screen (data buffer of the screen), use the `clear_screen()` function. This function has no arguments and returns no value.

The except code block is executed when CTRL + C is pressed on the keyboard. This is called KeyboardInterrupt. When this block has been executed, the message Script end! is displayed in the terminal.

The finally code block is executed after the script. When the finally code block has been executed, the `clear_screen()` function is executed. This function clears the data buffer of the screen.

You have done it. You can now use our module for your projects.

AZ-Delivery

Now it's your turn! Develop your own projects and smart home installations. We will show you how to do this in an uncomplicated and understandable way on our blog. There we offer you sample scripts and tutorials with interesting small projects to quickly get started in the world of microelectronics. In addition, the Internet also offers you countless opportunities to learn more about microelectronics.

If you are looking for more high-quality products for Arduino and Raspberry Pi, AZ-Delivery Vertriebs GmbH is the right place for you. We offer you numerous application examples, detailed installation instructions, e-books, libraries and of course the support of our technical experts.

<https://az-delivery.de>

Have Fun!

Impressum

<https://az-delivery.de/pages/about-us>