Recap: Installation on windows

- Problem on Windows: Scripts are not executable
  - This makes the activation of the virtual python environment impossible.
  - It depends how the powershell policies are set up.
  - To make scripts excecutable you can type the command

```
Set-ExecutionPolicy Unrestricted -Scope Process
```

  - Afterwards you can activate the virtual environment

- Mind file extensions on Windows
  - On windows computers the file extension might be different then on linux (.exe instead of .py : you can find this out using command completion in the power shell (i.e. pressing on Tab to complete the command)

- It is possible to use version 2 of WSL (Windows Subsystem for Linux)
  - Instructions can be found on the web

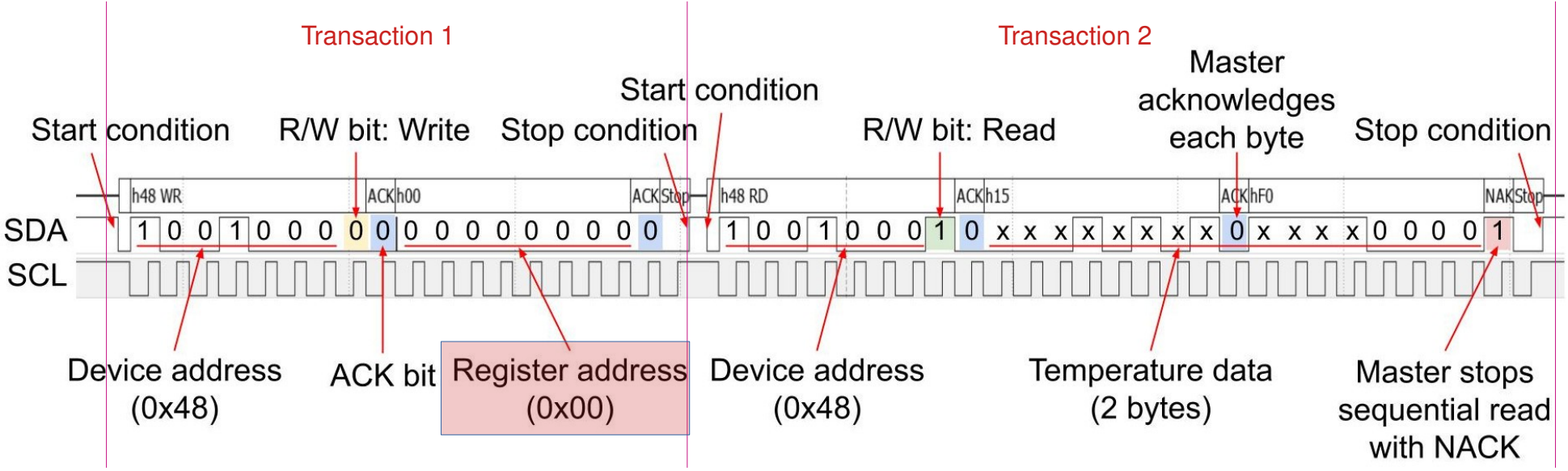
Recap : Serial Connections

- Typical serial connections used with Microcontrollers:
  - UART : robust, for long-range inter-device connections, slow typically 9600 – 115200 kBit/s
    - High voltage levels, re-synchronisation after each transferred word, full duplex
  - I2C : for on board connections, 2 wires only, middle speed range 400kBit/s
    - Pull-up resistors, many devices on one bus, addressing scheme, half duplex
    - Protocol has a concept of addressing devices
      - First frame of a transaction: Controller sends device address, and informs target if a read or a write will be performed
  - SPI : for on-board connections, 3 wires + 1 CS for each target device, high speed up to 100Mbit/s
    - No pull-ups, full duplex, Chip Select signals for each target

(More features are listed on the course web-site)

Typical usage of I2C for device with “registers”

- Many devices (like our Sensor) have internal registers which have to be read and written
  - These registers have also “addresses” to distinguish them
  - These register addresses have nothing to do with the device address
  - I2C protocol “knows” about device addresses but nothing about register addresses
- However, devices with registers fortunately behave all in the same way (by convention):



Exercise 1: read out the chip-ID of the BME280

- Write a small program to read out the register containing the chip-ID of the BME280
  - First scan the I2C bus to check that the sensor is correctly connected and responds.
  - In the manual you find the address of this register.

- Use the official micropython documentation to find and understand the micropython function calls you use:
  - You will need
    - The Pin class of the machine package
      - Defines input and output pins
    - The I2C class of the machine package
      - Look at the documentation of the constructor

Depending on your experience you can try this on your own or use the code template provided on the webpage of the course

Exercise 2: Write a Class for the Sensor

- The Class should have the following methods:
  - A constructor which initialises the sensor
  - A method which performs a single measurement of Temperature, Humidity and Pressure
    - First you need to trigger the measurement
    - You need to wait until the chip has completed the measurement
    - You need to read out the raw values of the measurements
    - You need to do the necessary calculations (indicated in the data sheet) to turn the raw values into T in Celsius, Humidity in %, Pressure in mb.
  - A method to calculate and estimated value of the Altitude (from the pressure)
  - May be a method to convert the absolute pressure to pressure at 0 Altitude (this is what is given in weather forecasts to compare air pressure at different locations)
  - A debugging method which dumps the measured values to the console (print statement)

Exercise 2: Write a Class for the Sensor

- Methods to use from the micropython machine.I2C Class: Look up the documentation!
  - This Class has the functions writeto(...) and writeto_mem(…) and readfrom(…) and readfrom_mem(): Look at the micropython documentation and understand the difference.
- The Constructor should do the following
  - Read out the calibration constants from the internal ROM memory of the sensor and store them in the class object for the calculations.
    - This involves to read out bytes and bitfields from the ROM memory and put them together and covert them to different python types like (un)signed Short and (un)signedChar.
      To do this you will have to use the “unpack” function from the “struct” module in python. Study it in the python documentation
  - Configure the measurements in a way that the highest possible precision is achieved (lowest measurement speed, highest possible “oversampling”)
- Once you have written your class write a small routine to test the class:
  - The program should do a measurement every second and dump the result to the screen.

There is a code template available in the exercise section of the web-site

Tools

- Once you have written your Class you need to transfer the file to the microcontroller
  - You can do this by using the mpremote command:

    mpremote cp {myfile} :.

    Copies the file “myfile” to the mini filesystem of micropython. If a program imports the Class from this file, it will be found by micropython. (You can also create a directory called “lib” and move packages and modules there. It should be also found by micropython. More info in the micropython doc)

- To run a program which is still on the development computer do:

  mpremote run {filename.py}

- To transfer a program permanently to the microcontroller and run it at each power up:
  - Give the python file the name “main.py” and copy it to the micropython filesystem
    (Be aware that you need to attach to the controller with a terminal emulator in case you want to see the output on the console (i.e. minicom on linux/mac or putty on Windows).

Python: Handling of bits and bytes

- Handling of bits and bytes in python is very painfull
- The micropython routines to read and write via I2C deal with **`bytes`** objects which are "immutable sequences of bytes"
  - They cannot be changed
  - Single bytes can be accessed with an index : [n] (like arrays)
- To convert series of bytes to "normal" python numbers we can use struct.unpack()

```
import struct
…
num_arr=struct.unpack( format, buf )
```

  - The format specifier may contain identifiers for multiple values
  - num_arr is the return value : and tuple of "normal" python numbers
    format defines what kind of number will be extracted from buf
    buf is a sequence of bytes read out from the hardware
  - Depending on the length of buf, 1 or many numbers are extracted from the buf
    - This is why the return value is an array

Recap: Wednesday

- Examples of format specifiers for the unpack routine (only those we need in our exercise; there are many more in the python documentation):

The first character of the format specifier specifies the endianess of the buffer.

'<': little endian
'>' : bif endian

ESP32 is little endian.

Format	Type	No of bytes
'<h'	signed short	2
'<H'	unsigned short	2
'<b'	signed char	1
'<B'	unsigned char	1

Little endian means: in memory the least significant bytes of a multi-byte number are stored at the lower addresses.

Recap : Wednesday

- I2C Access in micropython:
  - `i2c.readfrom(DevAddr, nbytes, stop=True)`
    - reads n bytes from target Device with address DevAddr; if stop=True (default) generate Stop condition afterwards. (If stop=False more bytes can be read after this function call until the Stop condition is created)
  - `i2c.readfrom_mem( DevAdr, memaddr, nbytes, addrsize=8)`
    - Assumes the targetDevice needs to be given an internal address from which to read (e.g. to access a specific register or a specific memory location)
    - This generates the following transactions
      1) Writes the memaddr to the device (addrsize specifies the number of bits for memadr. The number of bytes to be transferred in this first transaction is ceil(addrsize/8)
      2) Read n bytes from the device
- Equivalent functions exist for writing

Microcontrollers

Part III

An OLED display

(Christoph.Schwick@cern.ch)

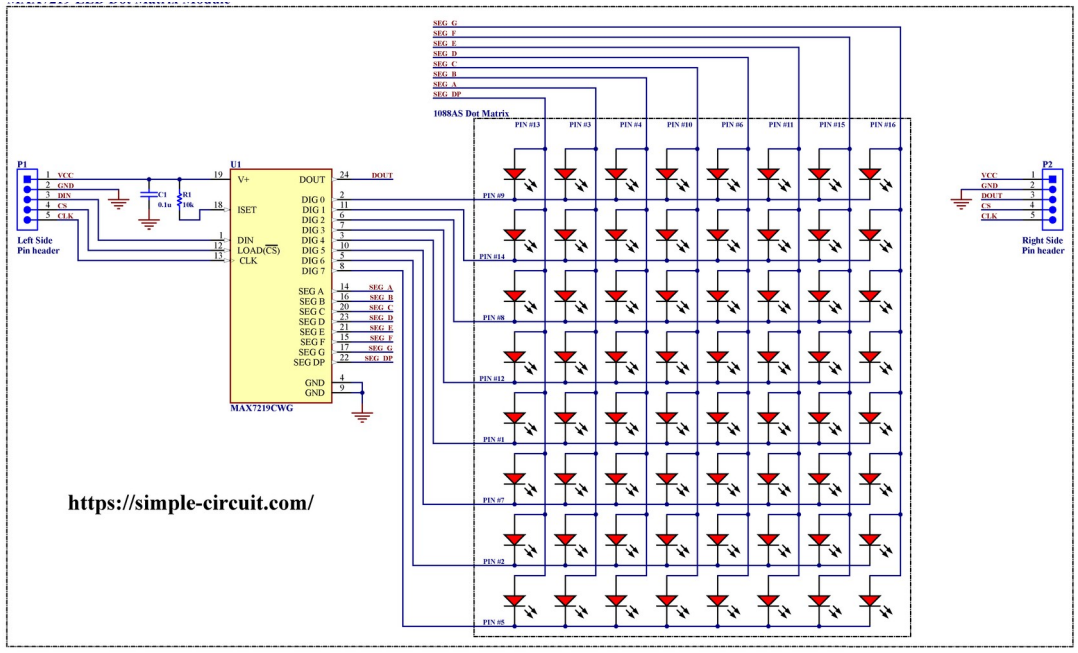
OLED Displays

- OLED displays are similar to TFT displays
  - But they do not need background light: the Pixels are creating light themselves
  - The black is really black : very high contrast
  - They have a shorter lifetime than TFT displays and an image shown for long periods might “burn in”
  - They come in “modules” containing:
    - The OLED pixel matrix matrix itself (128 x 64 in our case)
    - A controller chip
    - Some peripheral electronics to connect to the outside world
- Common interfaces to small OLED displays
  - I2C, SPI, parallel interfaces
  - Operating voltage 5V (or 3V3 depending on the product)
    - Internally higher voltages are required to operated the OLED pixels. These voltages are generated from the 5V or 3V3 with a DC/DC converter in the module or in the controller chip.

OLED display: how it displays the Pixels

- OLED displays work similar to old fashioned Cathod Ray Tubes (CRT of old televisions):
  - It is unpractical to switch on and off 128x64=8192 individual LEDs (a lot of control lines would be needed.
  - Therefore the Pixel are arranged in a matrix with rows and columns and only the rows and the columns are connected to the controller chip. Only one Pixel can be switched on at one moment. But all Pixels of the display are scanned through many times per second so the eyes does not realize this (like the in the old CRT)

Principle functioning of a LED matrix

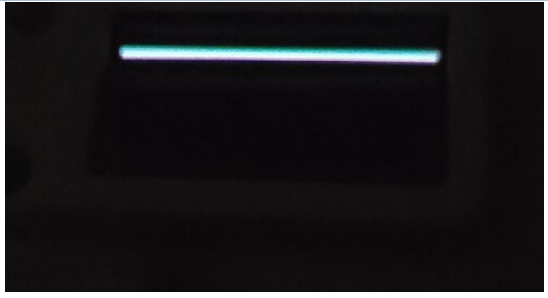


OLED Display: making the pixel scanning visible

- Photos of a completely white OLED display (all pixels on) with different exposure times

Exposure: 125ms

Exposure: 330µs



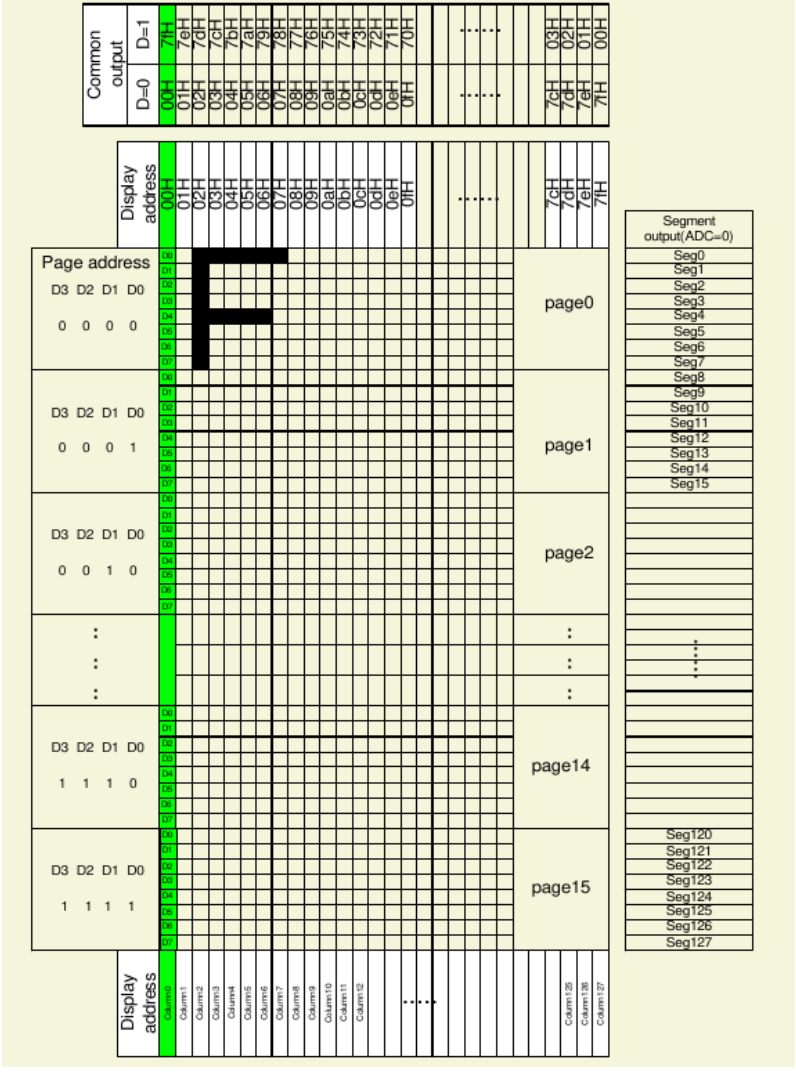
Exposure 1ms

Operating the OLED display

- Configuration of the module:
  - The controller chip is a dedicated chip designed to control OLED displays. However, the chip is not designed for a specific OLED matrix: different matrices can be operated with the controller.
  - Hence it is necessary to configure the controller to adapt it to the OLED matrix connected to the Controller.
  - The following configuration items need to be set:
    - Clock related constants have to be set as a function of the external clock provided to the chip and the desired “pixel clock”
    - Some offsets have to be set: they describe how the rows and the columns of the Pixel matrix are connected to the controller chip outputs and how many pixel rows and columns the display really has.
    - A pre-charge time needs to be set: It has to do with the fact that it takes time to ramp up the voltage for a powered pixel to the necessary voltage to emit light. To shorten this period (and to get more contrast and luminosity) certain lines of the pixel module are “pre-charged” to a voltage which is between 0 and the desired final voltage. The voltage step to switch on the pixel is then smaller and the switching on of the pixel needs less time.
    - A reference voltage VCOM used internally in the display has to be adjusted
    - How the Pixels should be addressed when transferring data (see next slides)

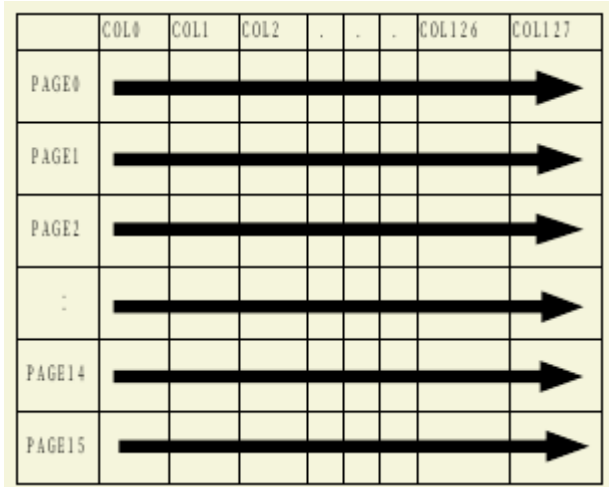
Controller Chip: pixel addressing

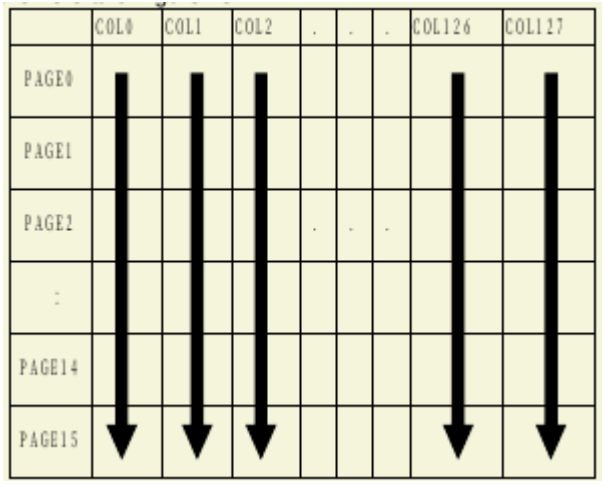
- Diagram from the data sheet:
  - Pixels are arranged in columns and rows
  - The rows are divided into “pages”
    - A page contains 8 rows for every column
    - Since always entire bytes are transferred to the controller, always 8 pixels belonging to the same page and to one specific column are set at the same time.



Controller Chip: pixel addressing

- Two possible addressing modes for writing data into the controller memory:
  - Subsequent bytes describe pixels in he same page (column number is increased)
  - Subsequent bytes describe pixels in the same column (page number is increased)





Operating the OLED display

- Operation of the Display
  - The Controller chip contains memory where every pixel is represented by a single bit (more bits in case of color or gray scale displays)
  - In our OLED display data is transferred in units of bytes. One byte has 8 bits and corresponds to 8 pixels (vertically arranged) in a page
  - For faster operation the Controller chip contains internal logic with a counter and you can transfer blocks of bytes in one go to the memory. They will be stored either (depending on configuration parameters which you have to set beforehand)
    - In all pages of the same column OR
    - In all columns of the same page

    This means, with one i2c transaction you can either
    - Set all pixels in one column to the desired values OR
    - Set a “band” of 8 pixels in all columns

How many transactions are needed to write the entire screen (one number for each method)?

Operating the OLED display

- Operation of the Display
  - The Controller chip contains memory where every pixel is represented by a single bit (more bits in case of color or gray scale displays)
  - In our OLED display data is transferred in units of bytes. One byte has 8 bits and corresponds to 8 pixels (vertically arranged) in a page
  - For faster operation the Controller chip contains internal logic with a counter and you can transfer blocks of bytes in one go to the memory. They will be stored either (depending on configuration parameters which you have to set beforehand)
    - In all pages of the same column OR
    - In all columns of the same page

    This means, with one i2c transaction you can either
    - Set all pixels in one column to the desired values OR
    - Set a “band” of 8 pixels in all columns
  - This means the entire memory of the controller (128 columns 16 pages (with 8 pixels each)) can be written with 16 transactions (each containing 128 data bytes), when using the second method above

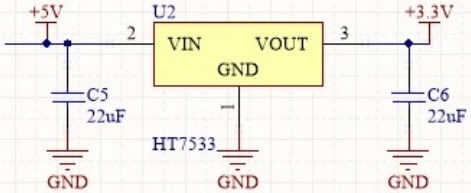
Operating the OLED display

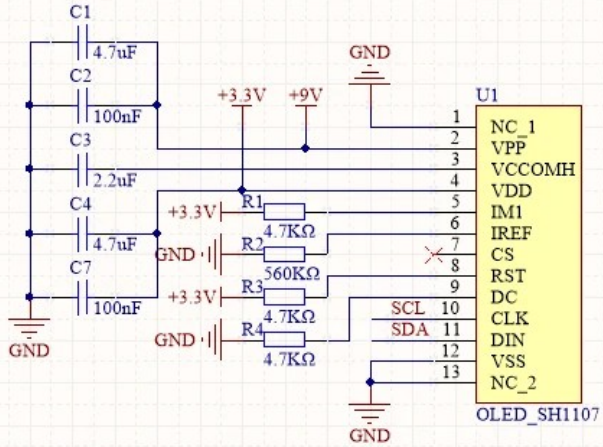
- Common practice: use a frame buffer
  - Fiddling around with the pages and columns can become quite a headache when addressing pixels.
  - For small displays the technique of a “framebuffer” is usually applied since it allows for simpler programming:
    - A framebuffer is a copy of the pixel buffer in the memory of the microcontroller.
    - The program can access this memory like accessing an 2 dimensional array
    - The application sets the the necessary pixels (=bits) in the framebuffer.
    - At end the entire buffer is written to the display in one go
  - **Pros :**
    - Easier programming
    - Framebuffer classes exist in libraries and do not have to be programmed by the user
      - These classes contain methods for usual operations: rectangle, lines, ellipses, text …
    - The display image is not “disturbed” during the possibly lengthy procedure to create the image.
  - **Cons :**
    - Requires memory in the microcontroller
    - In some situation might be slightly slower than using optimised code to draw directly in the display

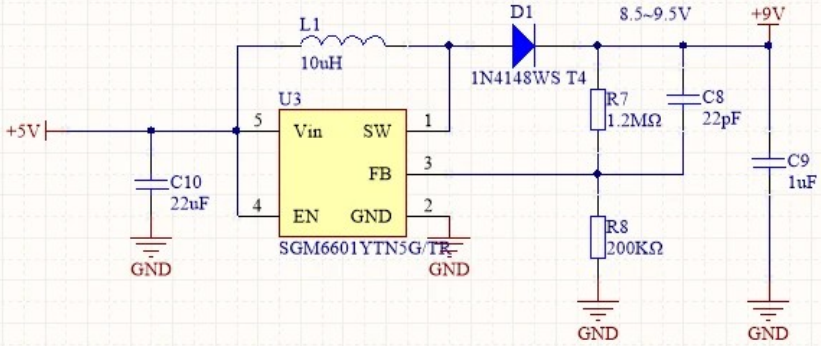
Access to the controller chip

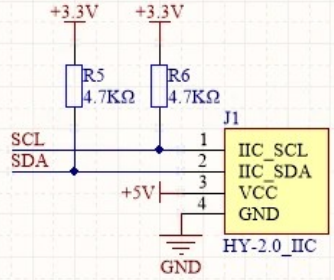
- The controller chip can be accessed with different interfaces:
  - I2C, SPI or a parallel interface.
  - Which interface is used is determined by the electronics around the controller chip (see the circuit diagram: The IM1 pin has to be pulled to 3V3 and the IM0 and IM2 pins have to be connected to to ground (default); see page 4 of data sheet)
  - We use I2C in our exercises.

- Supply voltage
  - The M5Stack module we are using in the course has to be supplied with 5V. Internally there is a Voltage regulator which generates 3V3 out of this voltage which is provided to the controller chip.
  - To operate the pixel matrix a voltage of 9V is required. It is generated in an external DC/DC converter (external means : not in the controller chip but an extra chip on the display module (see circuit diagram)


+5V
U2
VIN
VOUT
GND
HT7533
+3.3V
C5
22uF
C6
22uF
GND
GND
GND


C1
4.7uF
C2
100nF
C3
2.2uF
C4
4.7uF
C7
100nF
+3.3V
+9V
GND
R1
4.7KΩ
R2
560KΩ
R3
4.7KΩ
R4
4.7KΩ
SCL
SDA
U1
NC_1
VPP
VCCOMH
VDD
IM1
IREF
CS
RST
DC
CLK
DIN
VSS
NC_2
OLED_SH1107


L1
10uH
D1
1N4148WS T4
8.5~9.5V
+9V
+5V
U3
Vin
SW
FB
EN
GND
SGM6601YTN5G/TR
R7
1.2MΩ
C8
22pF
C9
1uF
R8
200KΩ
C10
22uF
GND
GND
GND
GND


+3.3V
+3.3V
R5
4.7KΩ
R6
4.7KΩ
SCL
SDA
+5V
J1
IIC_SCL
IIC_SDA
VCC
GND
HY-2.0_IIC
GND