

Embedded Systems

Coursework 1, Sensors and I²C

24 January 2017

Last time...

- Looked at the market for Internet of Things

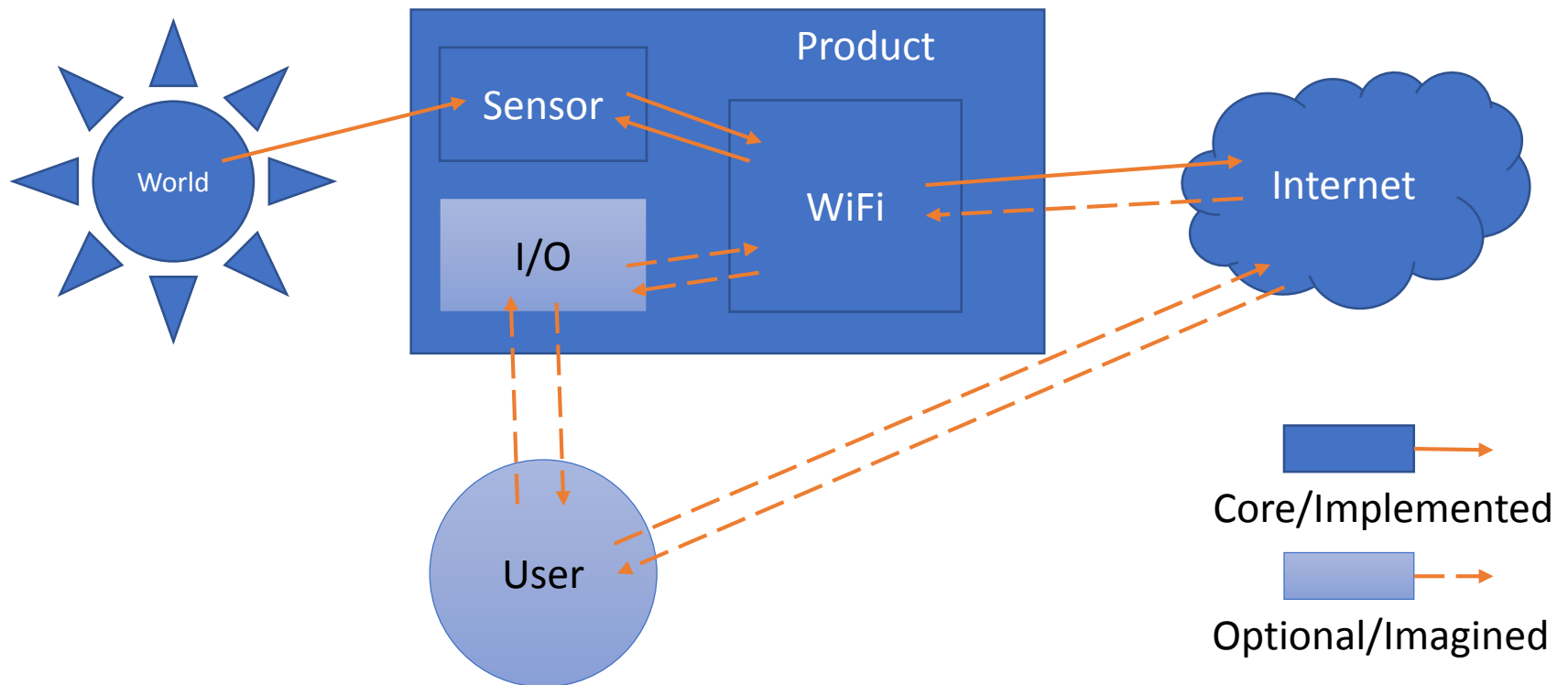
This time...

- Introduction to Coursework 1
- Interfacing sensors with I²C

Coursework 1

Coursework 1

- Use a WiFi microcontroller and a random sensor to make an IoT product concept



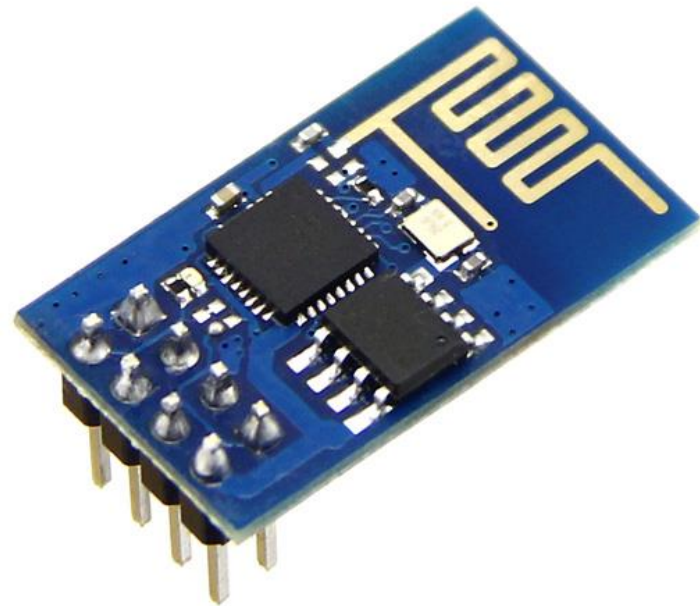
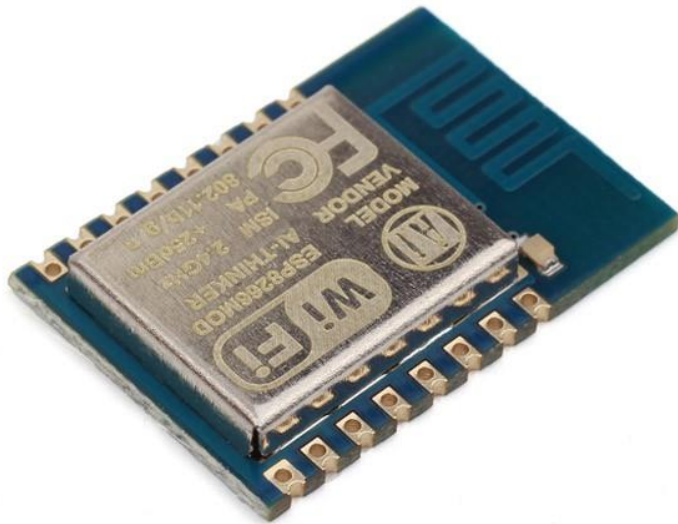
Coursework 1

- Deliverables
 - Demo showing sensing, processing and upload of data
 - Code submission via GitHub
 - Website* marketing concept for product
- Timescale
 - This week: lab on sensor communication
 - Next week: lab on MQTT
 - Week after that: demo
 - Week after that: code and website submission

*Just a static document

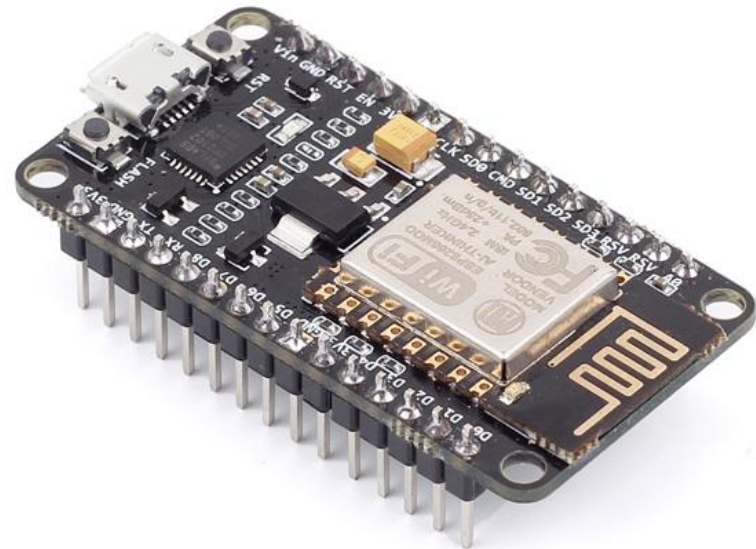
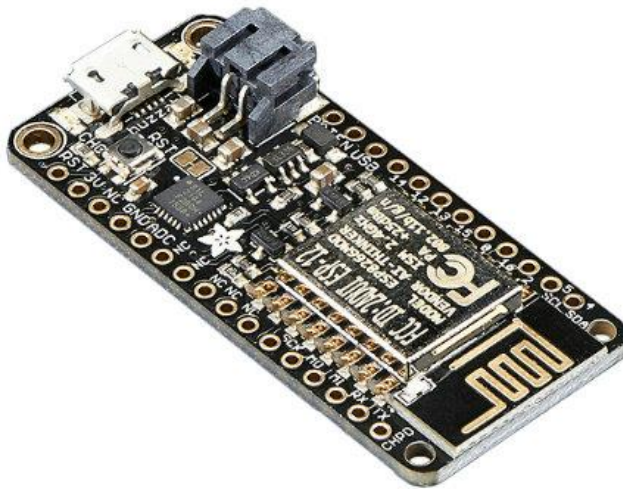
ESP8266

- WiFi System-on-chip
 - WiFi, processor, RAM
- Flash memory, antenna



ESP8266 hosts

- Hosted on a breakout board
 - Adafruit Huzzah Feather
 - Or NodeMCU
 - Pins, power supply, USB interface, reset circuitry



ESP8266 documentation

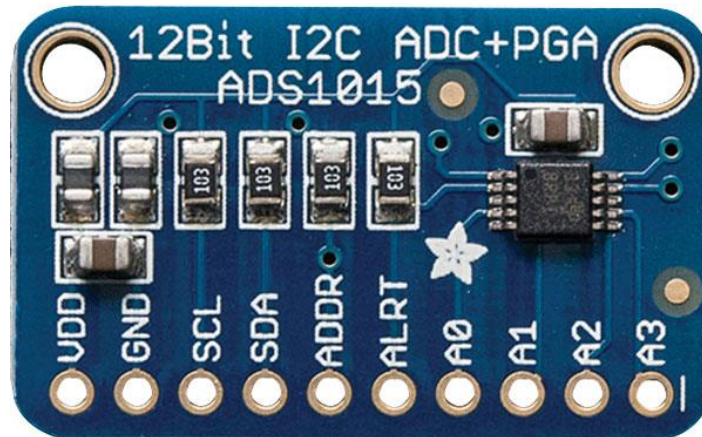
- Look for documentation for the ESP8266 and the host board
 - Except the ESP8266 datasheet – it's too low level
- Adafruit host board:
<https://learn.adafruit.com/adafruit-feather-huzzah-esp8266/overview>
- NodeMCU host board:
http://www.nodemcu.com/index_en.html
 - Just using the NodeMCU hardware, not the firmware

Sensors

- 13 different types. You get a random one.
- Three kinds of interface
 - I²C
 - Analogue (with I²C ADC)
 - One sensor has a digital, edge-triggered output

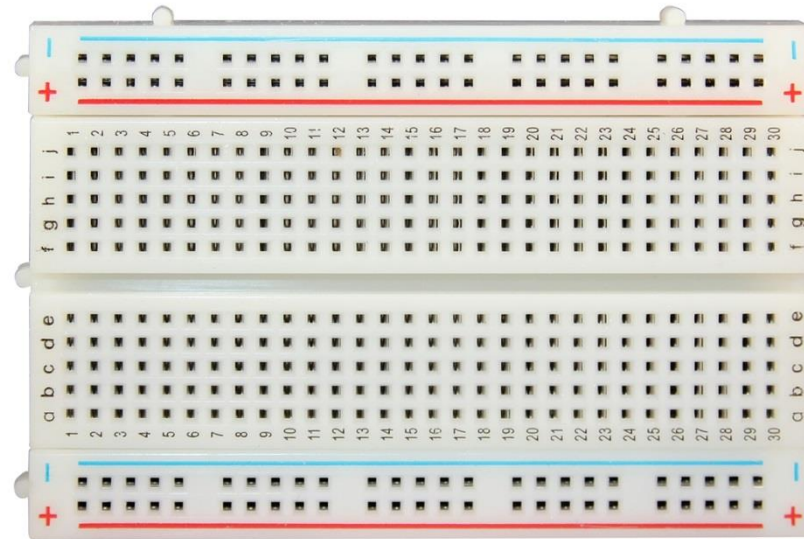
Sensor modules

- I²C sensors and ADCs come on breakout boards/modules
 - Pins, Power supply
- Two information reference points here too!
 1. The sensor or ADC chip datasheet
 2. The module support page and circuit diagram



Breadboard

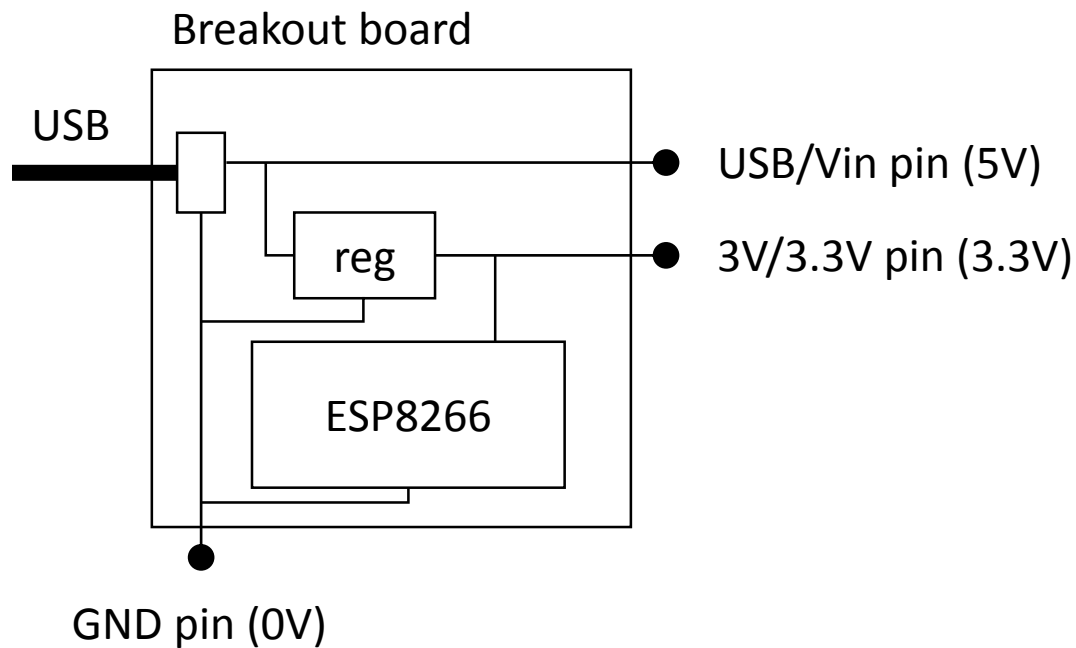
- For connecting up your modules
- Power is supplied by USB via the ESP8266 module
- Beware of voltage levels



Power supplies – caution needed!

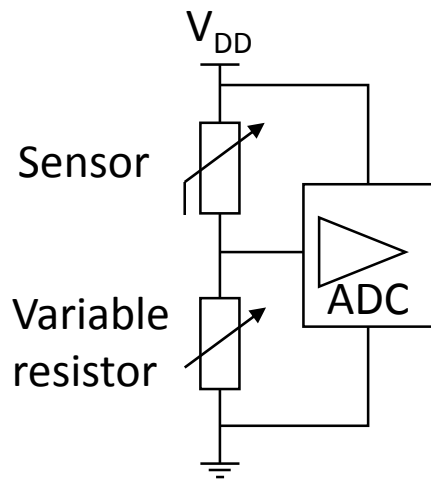
- ESP8266 modules supply USB voltage (5V) and digital logic voltage (3.3V)
- Inputs to ESP8266 must be no greater than 3.3V
- Some sensors are 3.3V, some are 5V and some don't mind either
- I²C uses open-drain signalling
 - Safe to connect 5V I²C slaves to 3.3V inputs
- Use resistors to divide down other inputs
 - Including analogue signals

Power supplies – caution needed!

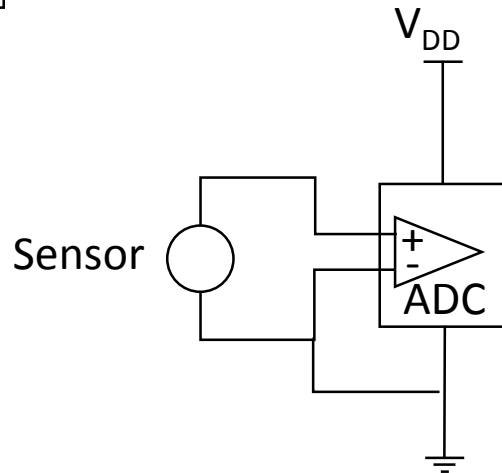
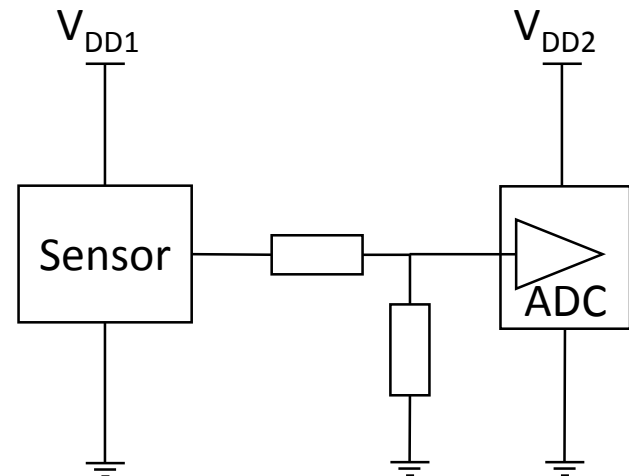


Power supplies – analogue sensors

Resistive sensor



Active sensor with output divider



Voltage source sensor with differential ADC

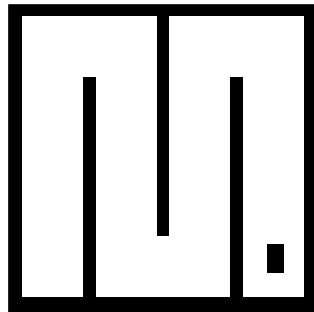
Additional I/O

- Selection of other components to add to your product



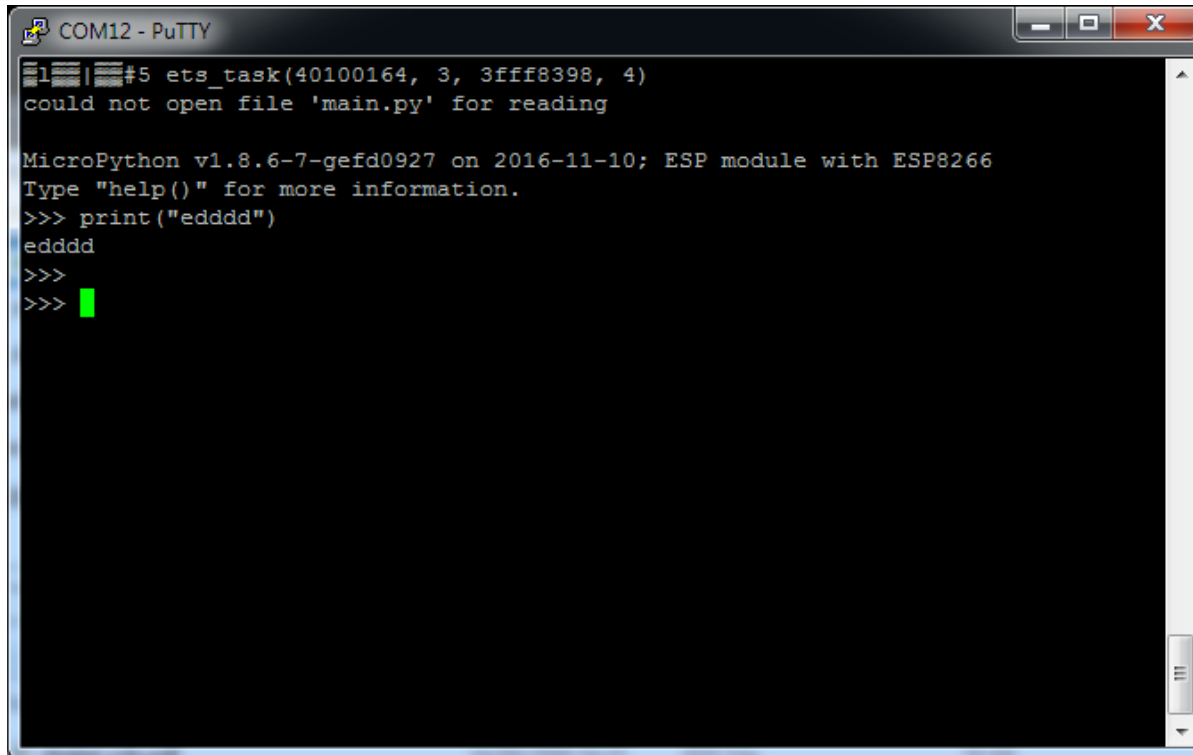
Micropython

- ESP8266 has a micropython interpreter loaded on it
- Micropython is an interpreted language
 - No compiler
 - A program (interpreter) decodes the source file (script) on-the-fly
 - Easy to develop but not efficient



Micropython

- USB interface provides a serial terminal
 - Just connect and type commands!



```
COM12 - PuTTY
#5 ets_task(40100164, 3, 3fff8398, 4)
could not open file 'main.py' for reading

MicroPython v1.8.6-7-gefd0927 on 2016-11-10; ESP module with ESP8266
Type "help()" for more information.
>>> print("edddd")
edddd
>>>
>>> █
```

Micropython

- Micropython is a fork of desktop python
 - You cannot install extra modules
 - Hardware interface modules are built in
- See the reference guide (ESP8266 specific):
<https://docs.micropython.org/en/latest/esp8266/index.html>

Downloading code

- Typing in code is tiresome
- But there's no drag-and-drop file interface to copy script files
- Use ampy to copy scripts
 - Ampy is a command-line tool that uses desktop Python
 - Install: `pip install adafruit-ampy`
 - Doc: <https://learn.adafruit.com/micropython-basics-load-files-and-run-code/overview>

Downloading code

- You can save any .py (or other) file you like
 - `ampy --port COM12 put script.py`
- If `main.py` exists in the root folder, it runs when ESP8266 starts up

Product concept

- Find out what your sensor can do
- Think of a novel IoT product that could use the sensor
 - Your concept can include other sensors or interfaces
 - Any feasible output is fine too
 - Bonus for additional I/O
 - Implemented if you can, otherwise just add it to your documentation
 - Does it need to be useful to be a successful product?

Data processing

- Micropython can do moderate maths
- Use it to reduce the data upload overhead
 - Event detection
 - Averaging or other statistics
- Your concept can include further server-side processing
 - You don't need to implement this

Internet communication

- Details will be covered next week
- Send JSON formatted messages to MQTT broker
- Can also subscribe to receive messages from the broker

Inter-integrated circuit
(I²C) bus

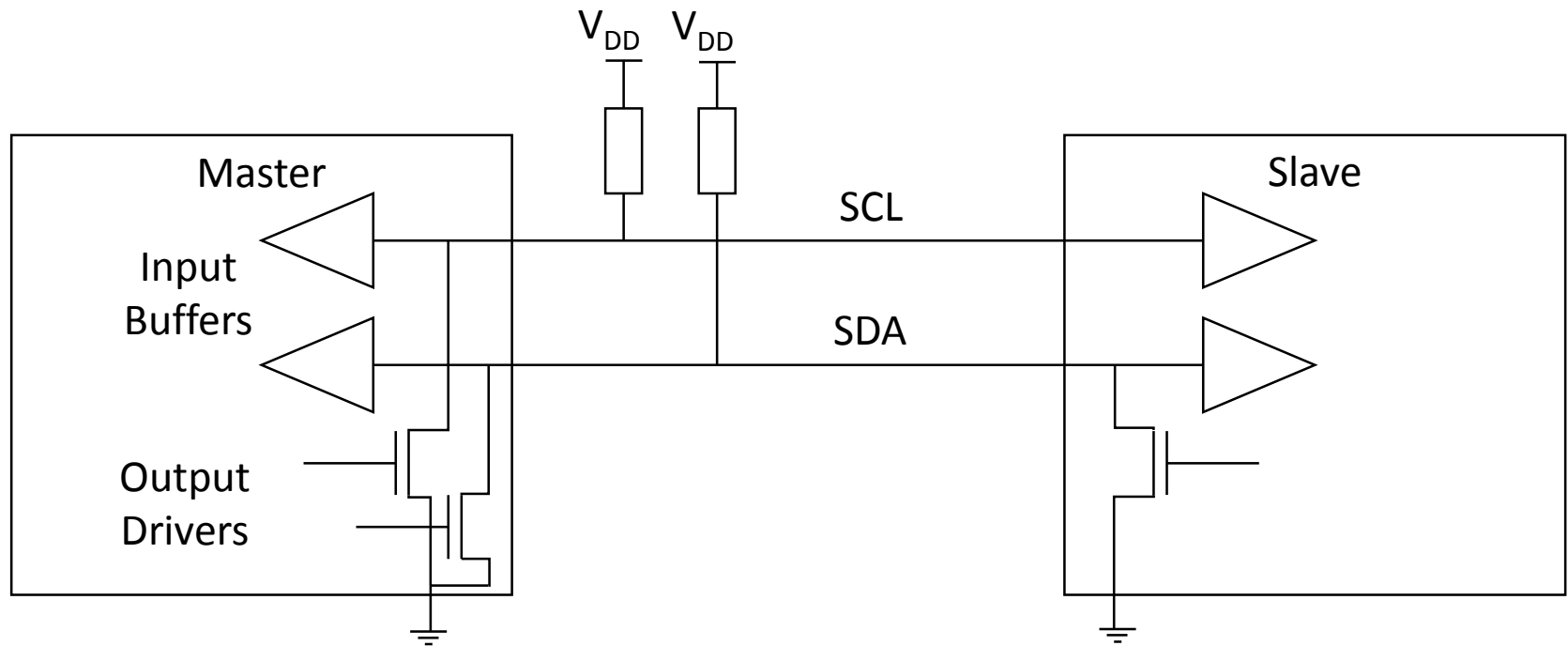
What is I²C for?

- Communication within a system
 - Short distance
- Low space requirements
 - 2 wires, 2 pins per device
 - Addressable with multiple masters and slaves
- Low data rates
 - Usually 100kHz or 400kHz
- Well-defined standard
 - Unlike SPI and UART
 - Can use it without worrying about timing diagrams

Hardware

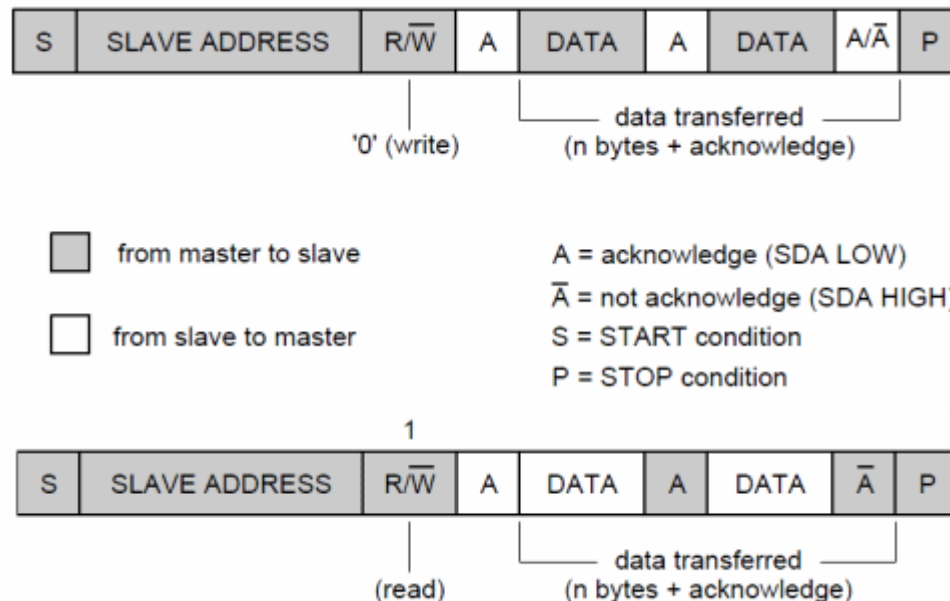
- Two wires: SCL (clock) and SDA (data)
 - Single-ended (not differential like USB and others)
- Devices use open-drain outputs
 - Can pull the wire low
 - Resistor pulls it high again
 - Slaves cannot control SCL
- Allows multiple masters and bi-directional communication with no risk of short circuit

I²C Physical layer



Message format

- All communications begin with master sending an address
- Last (LSB) bit of the address specifies read or write



Using an I²C peripheral

- Typical sensors have a register file (including coursework sensors)
 - Slave device has a number of registers
 - Also a register pointer
 - Often the first byte of a write message

Example register map

Reg. address	Contents
0x00	Status (R)
0x01	Mode (R/W)
0x02	DataH (R)
0x03	DataL (R)

Using an I²C peripheral

- Read the device datasheet!
 - Some devices auto-increment the pointer after each read/write
 - Some devices require you to read all data before it takes a new measurement
 - Some sensors are disabled by default, some are in read-on-demand mode, some read automatically
 - Some sensors have sensitivity or other parameters that must be set before use

Micropython implementation

- Software implementation
 - Hardware exists on ESP8266 but driver not written yet
- Create an instance of i2c class
 - Specify pins in initialiser arguments

```
from machine import Pin, I2C
i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000)
```
 - <https://docs.micropython.org/en/latest/esp8266/library/machine.I2C.html>

i2c instance methods

- `writeto(bus_addr, data)`
 - Opens write comm with slave at `bus_addr`, send each byte in `data`
- `readfrom(bus_addr, n)`
 - Opens read comm with slave at `bus_addr`, read `n` bytes
 - Returns `byte_array` of `n` bytes
- `writeto_mem(bus_addr, reg_addr, data)`
 - Opens write comm with slave at `bus_addr`, send `reg_addr`, send each byte in `data`
- `readfrom_mem(bus_addr, reg_addr, n)`
 - Opens write comm with slave at `bus_addr`, send `reg_addr`, switch to read mode, read `n` bytes
 - Returns `byte_array` of `n` bytes
- Other i2c methods are mostly lower-level, i.e. perform events that make up a message

Next time...

- Lab on Thursday
 - Collect a coursework kit
- Preparation
 - Install python and ampy on someone's computer
 - Investigate PuTTY (windows) or screen (Mac and Linux) for serial communication
 - Set up a GitHub account if you don't have one and find out how to use it
- Today: last chance to send me team preferences

Next week...

- Connecting your product to the internet
- More information about deliverables