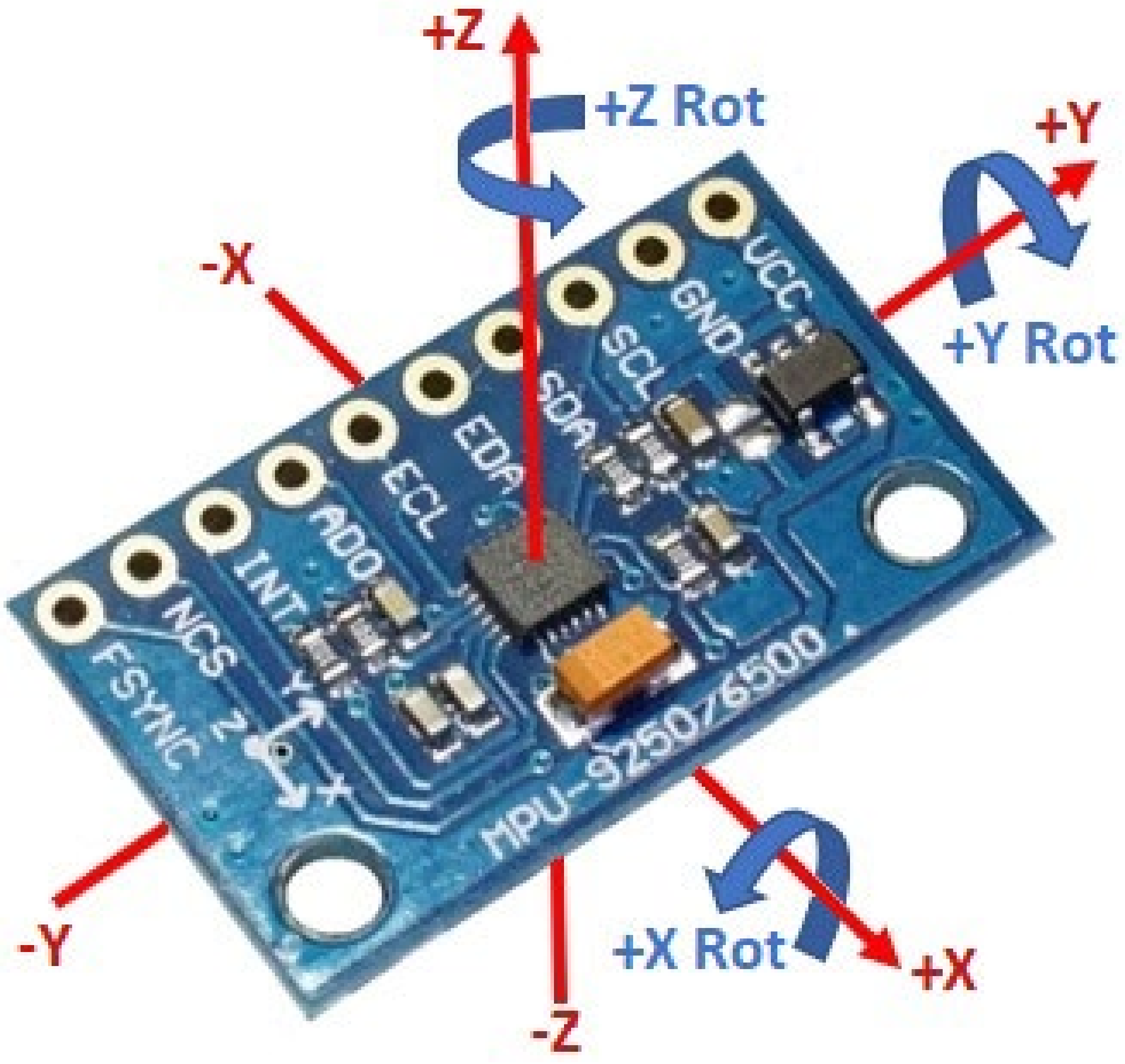
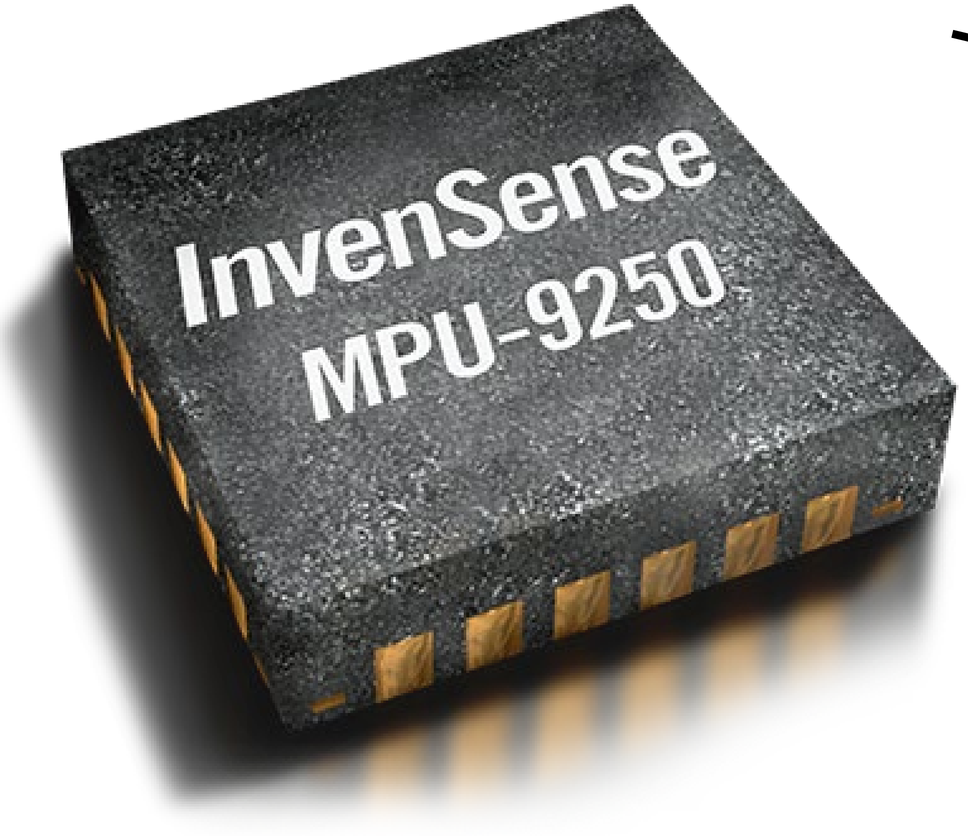
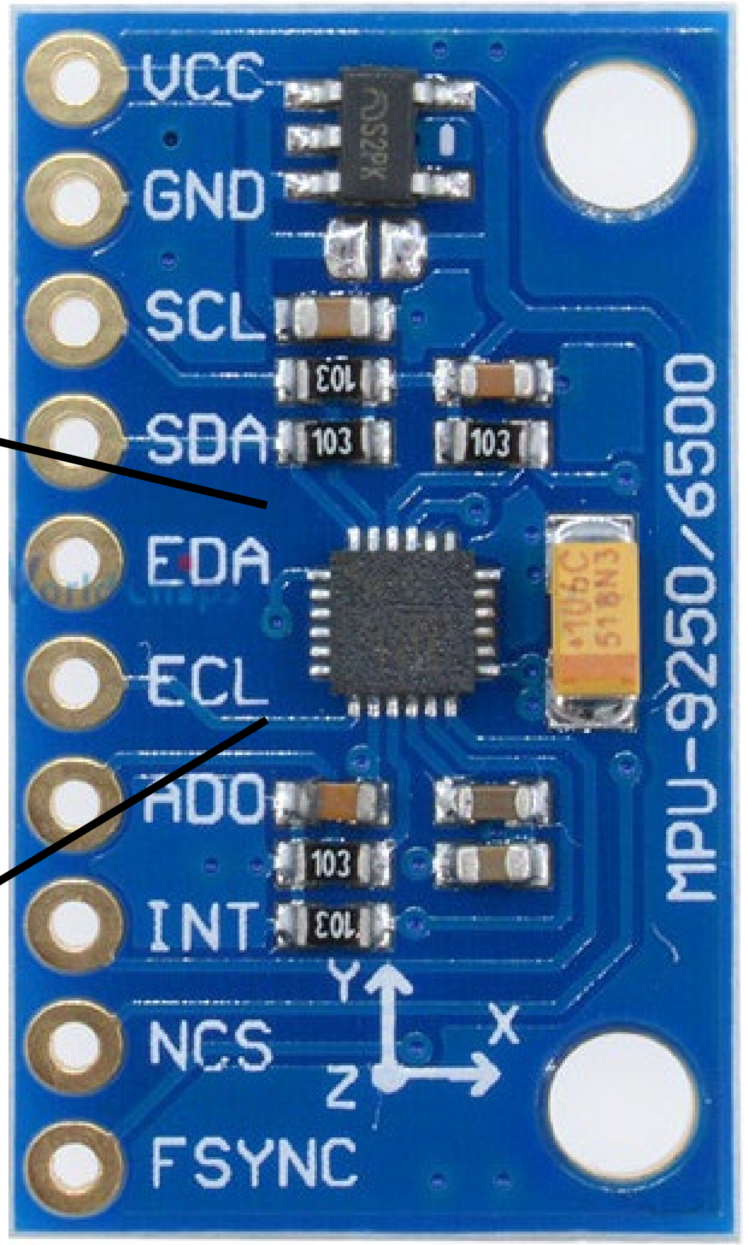
I^2C and IMU

Huaishu Peng | UMD CS | Fall 2023



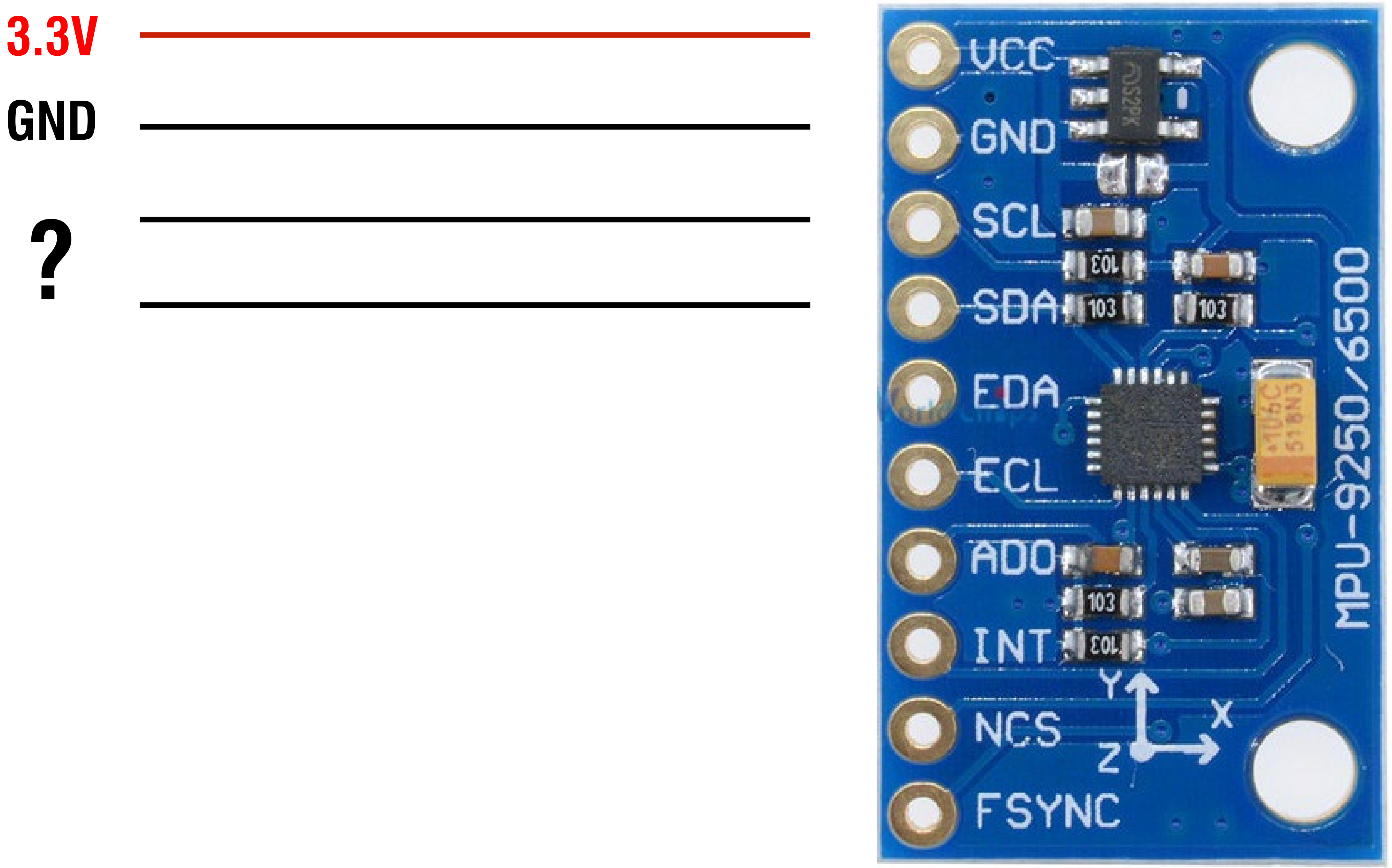
Accelerometer | Gyro | Magnetometer





Invensense MPU-9250

https://invensense.tdk.com/products/motion-tracking/9-axis/mpu-9250/


3.3V
GND
?
VCC
GND
SCL
SDA
EDA
ECL
ADO
INT
NCS
FSYNC
MPU-9250/6500
Y
X
Z

I^2C

I2C stands for Inter-Integrated Circuit

is a serial protocol for two-wire interface to connect devices like microcontrollers and other similar peripherals in embedded systems

It is used by almost all major IC manufacturers

I2C bus is popular because it is simple to use (only need **2** signal lines)

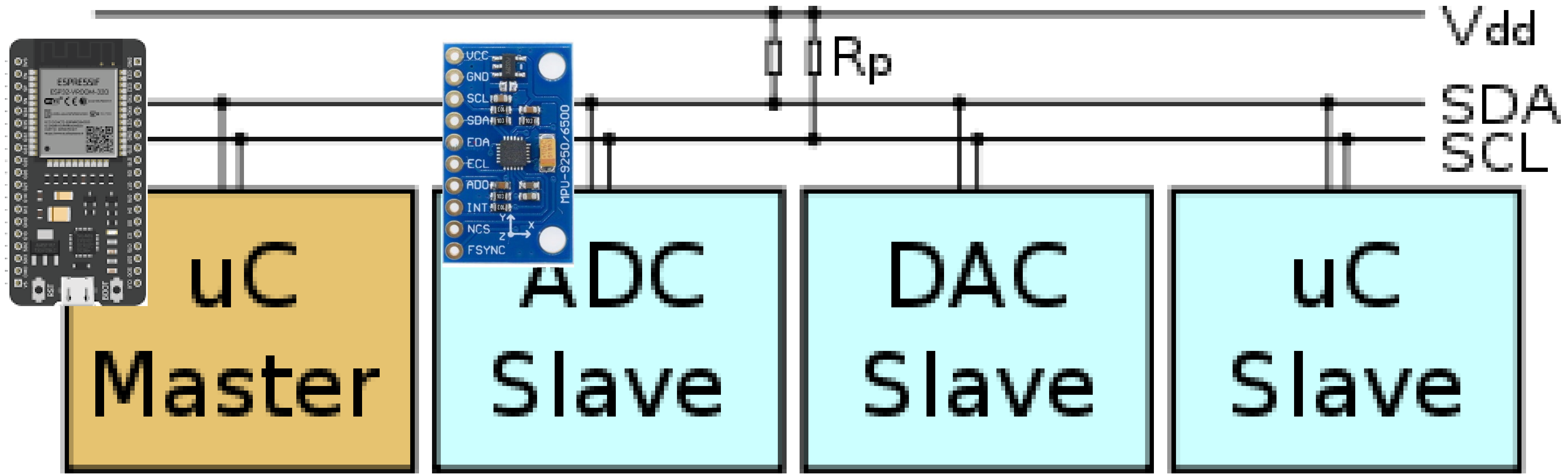
I^2C

4 wires in total

VCC and GND

SDA (serial data) and SCL (serial clock)

Primary-standby (Master-Slave) architecture (bi-directional)



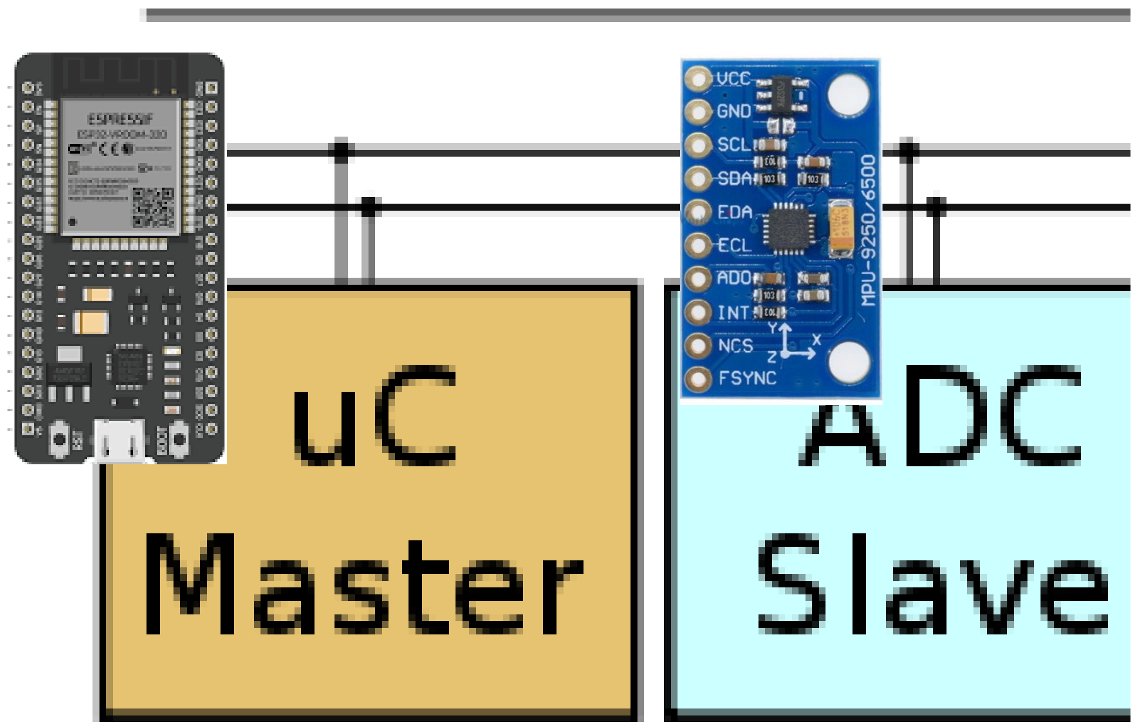
I^2C

How does the ESP32 know which peri devices to talk/listen to?

Each of the peripheral device has a 7-bit **address** in the datasheet – it's the **"name"** of the device

The address is written in the datasheet

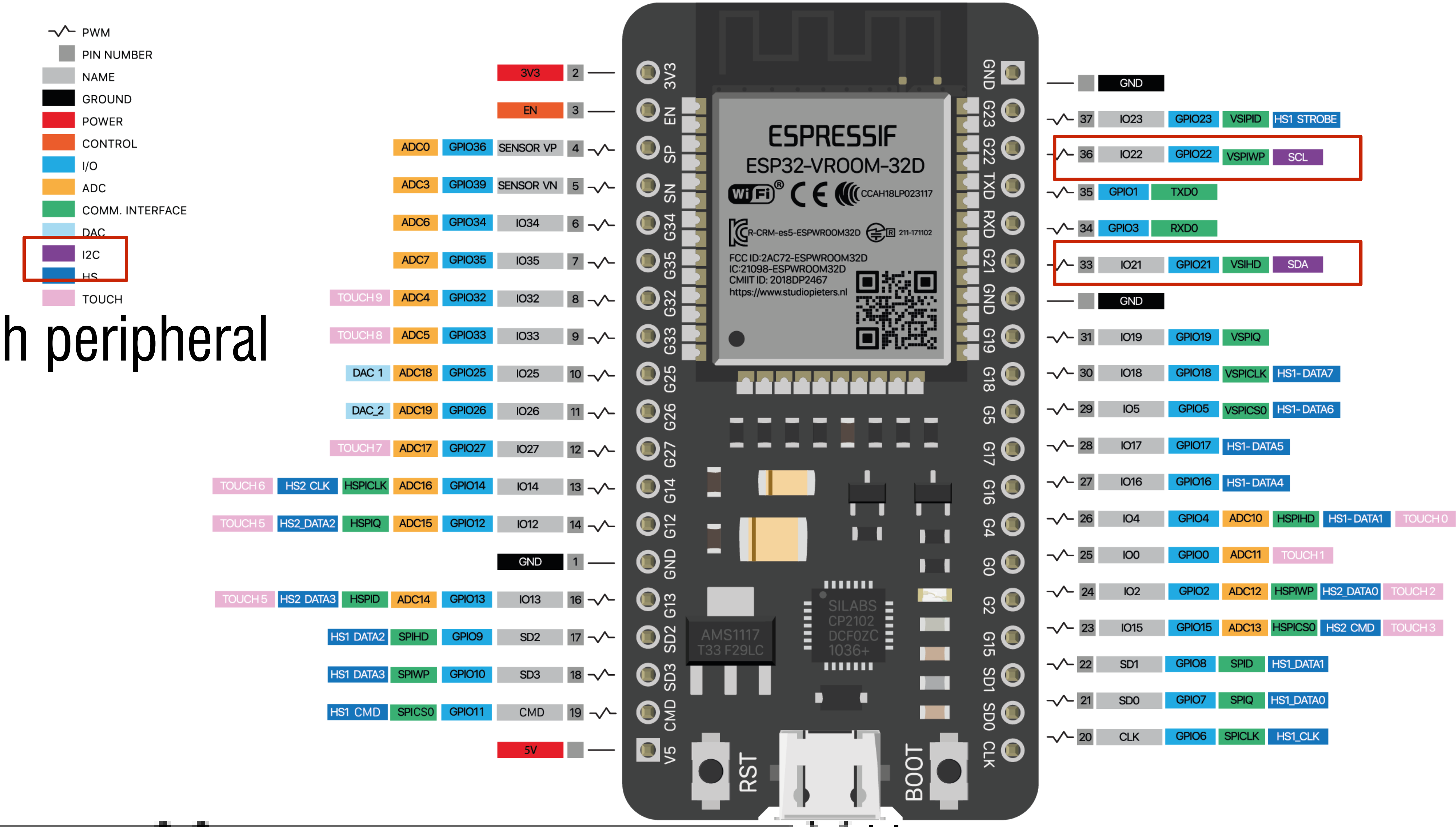
https://invensense.tdk.com/wp-content/uploads/2



I^2C ADDRESS	AD0 = 0 AD0 = 1		1101000 1101001		
V_{IH}, High Level Input Voltage		0.7*VDDIO			V
V_{IL}, Low Level Input Voltage				0.3*VDDIO	V
C_I, Input Capacitance			< 10		pF
V_{OH}, High Level Output Voltage	R_{LOAD}=1MΩ;	0.9*VDDIO			V
V_{OL1}, LOW-Level Output Voltage	R_{LOAD}=1MΩ;			0.1*VDDIO	V
$V_{OL.INT1}$, INT Low-Level Output Voltage	OPEN=1, 0.3mA sink Current			0.1	V
Output Leakage Current	OPEN=1		100		nA
t_{INT}, INT Pulse Width	LATCH_INT_EN=0		50		μs
V_{IL}, LOW Level Input Voltage		-0.5V		0.3*VDDIO	V
V_{IH}, HIGH-Level Input Voltage		0.7*VDDIO		VDDIO + 0.5V	V
V_{hys}, Hysteresis			0.1*VDDIO		V
V_{OL}, LOW-Level Output Voltage	3mA sink current	0		0.4	V
I_{OL}, LOW-Level Output Current	V_{OL}=0.4V V_{OL}=0.6V		3 6		mA mA
Output Leakage Current			100		nA
t_{of}, Output Fall Time from V_{IHmax} to V_{ILmax}	C_b bus capacitance in pf	20+0.1C_b		250	ns
V_{IL}, LOW-Level Input Voltage		-0.5V		0.3*VDDIO	V
V_{IH}, HIGH-Level Input Voltage		0.7* VDDIO		VDDIO + 0.5V	V
V_{hys}, Hysteresis			0.1* VDDIO		V
V_{OL1}, LOW-Level Output Voltage	VDDIO > 2V; 1mA sink current	0		0.4	V
V_{OL3}, LOW-Level Output Voltage	VDDIO < 2V; 1mA sink current	0		0.2* VDDIO	V
I_{OL}, LOW-Level Output Current	V_{OL} = 0.4V V_{OL} = 0.6V		3 6		mA mA
Output Leakage Current			100		nA
t_{of}, Output Fall Time from V_{IHmax} to V_{ILmax}	C_b bus capacitance in pF	20+0.1C_b		250	ns
Sample Rate	Fchoice=0,1,2 SMPLRT_DIV=0		32		kHz
	Fchoice=3; DLPFCFG=0 or 7 SMPLRT_DIV=0		8		kHz
	Fchoice=3; DLPFCFG=1,2,3,4,5,6; SMPLRT_DIV=0		1		kHz
Clock Frequency Initial Tolerance	CLK_SEL=0, 6; 25°C	-2		+2	%

Page **12** of **42**

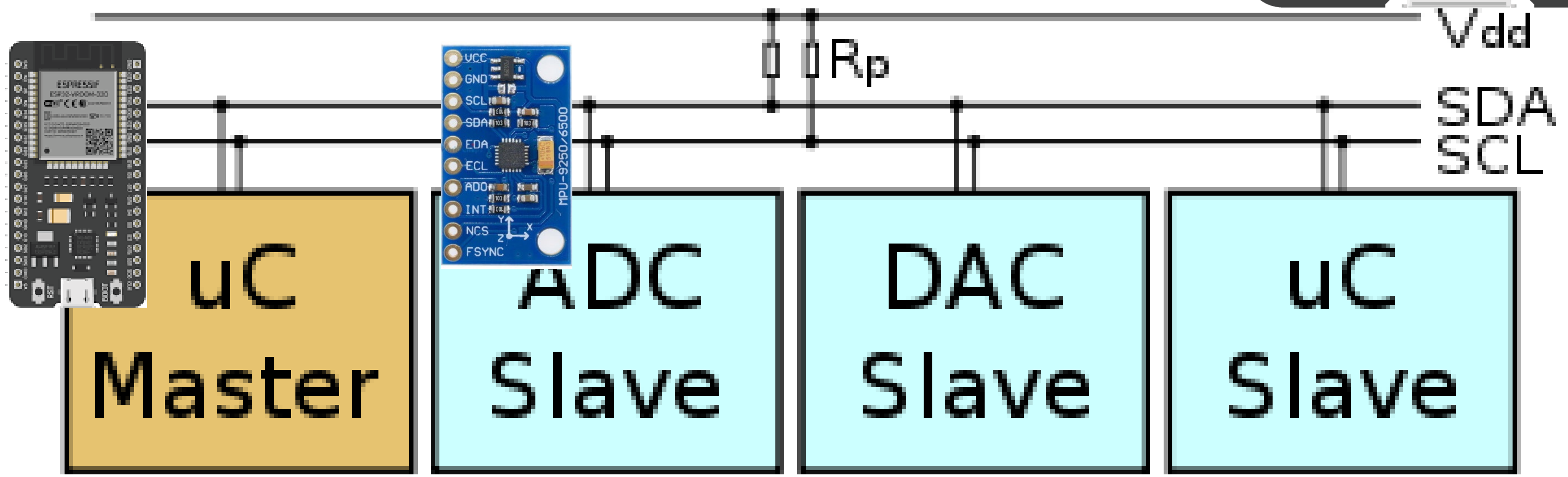
I^{2}C



How does the ESP32 know which peripheral devices to talk/listen to?

Can we use any pins on Arduino to connect to SDA and SCL?

GPIO 21 -> SDA; GPIO 22->SCL



I²C

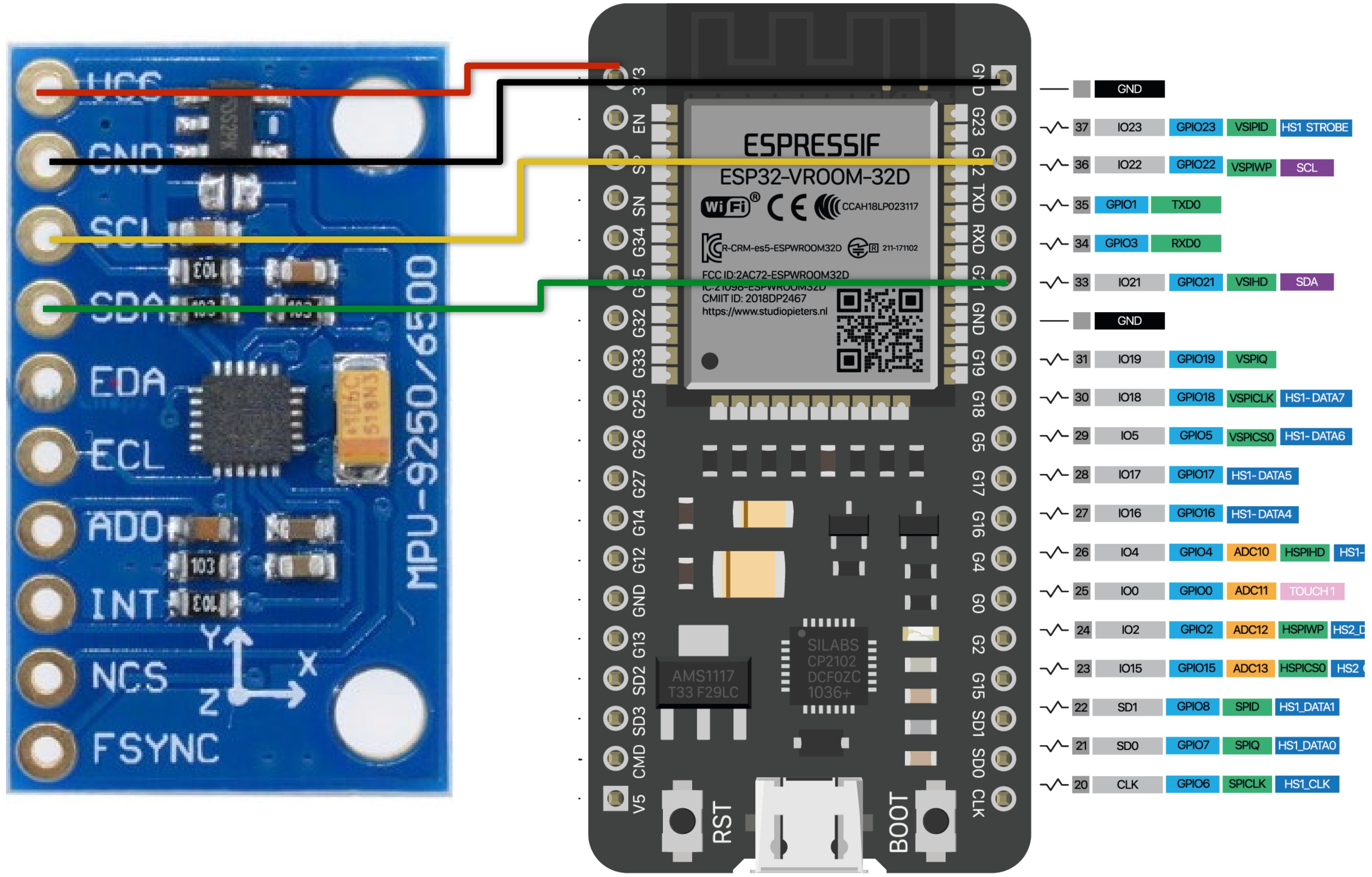
4 wires in total signal lines

VCC and GND

SDA (serial data) and SCL (serial clock)

Address: 0b1101000 (0x68)

SDA
SCL



How to read data from the IC?

I^2C library in Arduino **- Wire library**

Setup

Reading a register

Updating a register

I^2C library in Arduino **- Wire library**

#include <Wire.h>

Setup

Reading a register

Updating a register

What are register(s)?

- consider it as a **specific multi-functional storage space** of an IC
- We can **read** sensor data from certain register(s)
- We can also **write** a **specific data to one register** to change the sensor's behavior

I^2C library in Arduino - Wire library

Setup

Updating a register

Reading a register

```
#include <Wire.h>
const int sda = 21;
const int scl = 22;


void setup() {
  Wire.begin(sda, scl); //SDA, SCL
 }
```

I^2C library in Arduino **- Wire library**

Setup

Reading a register

Updating a register

I^2C library in Arduino **- Wire library**

Setup

Reading a register

Updating a register

Let’s try to read the accelerometer data along X axis from our sensor

First, send a read request
– **Wire.beginTransmission(addr)** opens communication with addr -> **0x68**
– **Wire.write(register_name)** register that you are looking for
– **Wire.endTransmission()** sends the request and returns

Then, read the answer to the request
– **Wire.requestFrom(addr, length)** prepares to read *length* bytes from addr
– **Wire.read()** reads the next available byte

endTransmission()

Description

This function ends a transmission to a peripheral device that was begun by `beginTransmission()` and transmits the bytes that were queued by `write()`. As of Arduino 1.0.1, `endTransmission()` accepts a

https://www.arduino.cc/reference/en/language/functions/communication/wire/endtransmission/

I^{2}C library in Arduino - **Wire library**

Setup

Reading a register

Updating a register

https://cdn.sparkfun.com/assets/learn_tutorials/5/5/0/MPU-9250-Register-Map.pdf

First, send a read request
– **Wire.beginTransmission(addr)** opens communication with addr -> **0x68**
– **Wire.write(register_name)** register that you are looking for
– **Wire.endTransmission()** sends the request and returns

Then, read the answer to the request
– **Wire.requestFrom(addr, length)** prepares to read *length* bytes from addr
– **Wire.read()** reads the next available byte

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]							

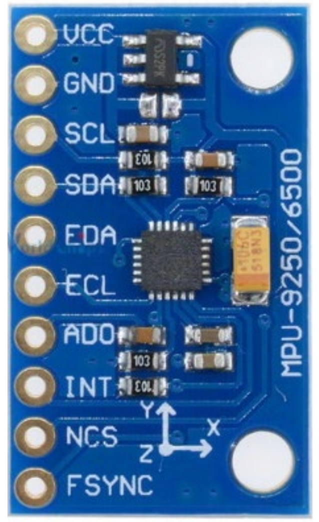
I²C library in Arduino - **Wire library**

Setup

Reading a register

Updating a register

Address: 0b1101000 (0x68)



We first need to read the left 8 bit of accel_x from register 3B

First, **send a read request**

– Wire.beginTransmission(**0x68**); opens communication with the sensor using its address
– Wire.write(**0x3B**); tell the sensor which register we are requesting
– Wire.endTransmission(); sends the request and returns

Then, **read the answer to the request**

– Wire.requestFrom(**0x68**, **1**); prepares to read *length* byte from the sensor address
– acc_x_h = Wire.read() reads the available byte

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]							

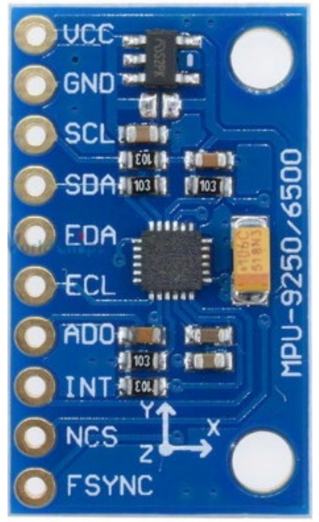
I²C library in Arduino - **Wire library**

Setup

Reading a register

Updating a register

Address: 0b1101000 (0x68)



To get the entire 16bit of the ACC_X data, we need the ACC_L as well

First, **send a read request**

– Wire.beginTransmission(**0x68**); opens communication with the sensor using its address
– Wire.write(**0x3C**); tell the sensor which register we are requesting
– Wire.endTransmission(); sends the request and returns

Then, **read the answer to the request**

– Wire.requestFrom(**0x68**, **1**); prepares to read *length* byte from the sensor address
– acc_x_l = Wire.read() reads the available byte

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]							

I²C library in Arduino **- Wire library**

Setup

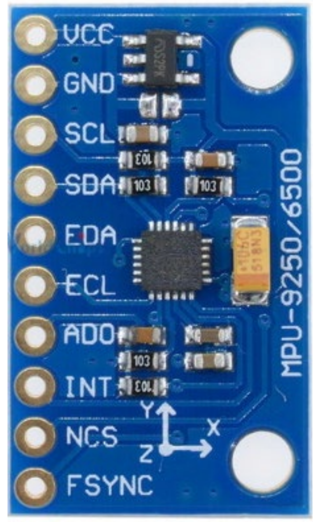
Reading a register

Updating a register

Next, we need to combine the data from 3B and 3C

```
acc_x_combined = acc_x_h << 8 | acc_x_l;
```

Address: 0b1101000 (0x68)



bitwise SHIFT and OR operation

A = 0b11110101
B = 0b01010101

A <<8	0b11110101 00000000
A<<8 | B	0b11110101 01010101

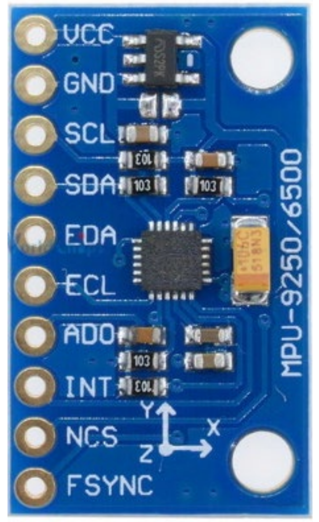
I^2C library in Arduino - **Wire library**

Setup

Reading a register

Updating a register

Address: 0b1101000 (0x68)



Last step, make the data mean something to us

3.2 Accelerometer Specifications

Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Full-Scale Range	AFS_SEL=0		±2		*g*
	AFS_SEL=1		±4		*g*
	AFS_SEL=2		±8		*g*
	AFS_SEL=3		±16		*g*
ADC Word Length	Output in two's complement format		16		bits
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*
	AFS_SEL=1		8,192		LSB/*g*
	AFS_SEL=2		4,096		LSB/*g*
	AFS_SEL=3		2,048		LSB/*g*

The result is raw data

Divided raw data by 16384.0 to get meaningful data

```
gX = acc_x_combined / 16384.0;
```

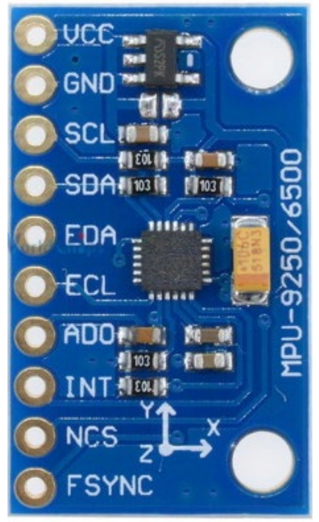
I^2C library in Arduino - **Wire library**

Setup

Reading a register

Updating a register

Address: 0b1101000 (0x68)



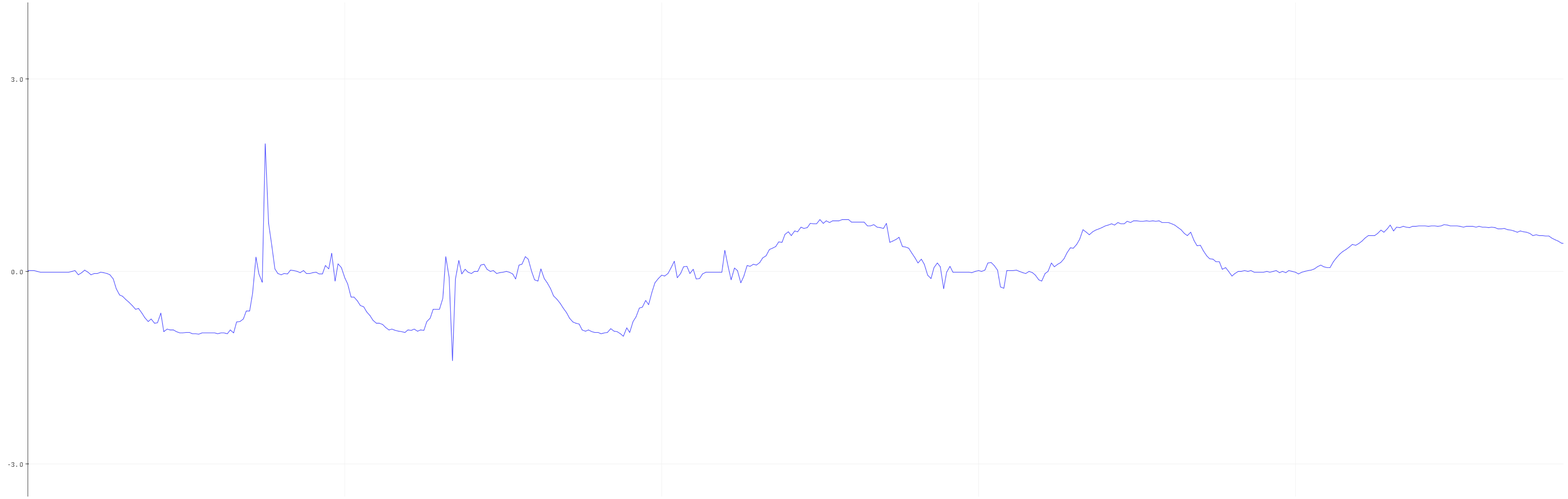

```
byte ACCEL_XOUT_H  = 0;
byte ACCEL_XOUT_L = 0;
int16_t ACCEL_X_RAW = 0;
float gX;
void loop() {
  // put your main code here, to run repeatedly:
  Wire.beginTransmission(address);
  Wire.write(0x3B);
  Wire.endTransmission();

  Wire.requestFrom(address, 1);
  ACCEL_XOUT_H = Wire.read();

  Wire.beginTransmission(address);
  Wire.write(0x3C);
  Wire.endTransmission();

  Wire.requestFrom(address, 1);
  ACCEL_XOUT_L = Wire.read();

  ACCEL_X_RAW = ACCEL_XOUT_H << 8 | ACCEL_XOUT_L;
  gX = ACCEL_X_RAW / 16384.0;
  Serial.println(gX);
  delay(10);
}
```

I^2C library in Arduino **- Wire library**

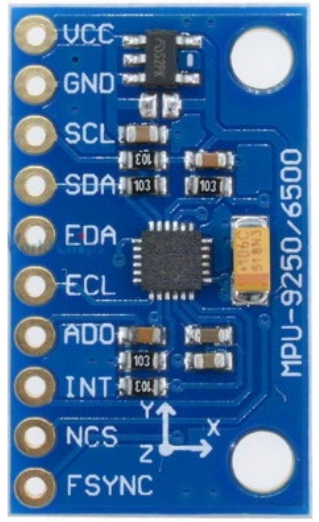
Setup

Reading a register

Updating a register



Address: 0b1101000
(0x68)



Assignment:

1. Read all XYZ Accelerometer Data and print them out in a meaningful way
2. Read all XYZ Gyro Data, print the raw value.
3. Move the IMU along each axis with acceleration, observe how the data looks like. You can use Serial plotter for observation.
4. Send us a video link of the experiment.

Register Map: https://smartlab.cs.umd.edu/CMSC730/assets/file/PS-MPU-9250A-01-v1.1.pdf
Datasheet: https://smartlab.cs.umd.edu/CMSC730/assets/file/MPU-9250-Register-Map.pdf