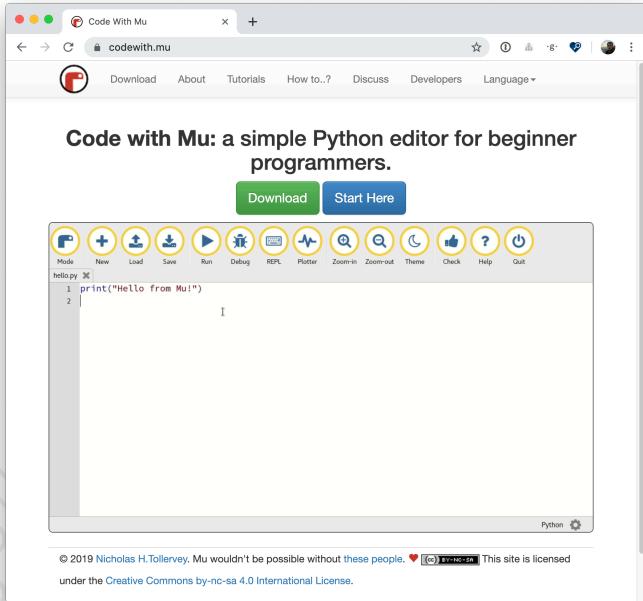


Software You'll Need

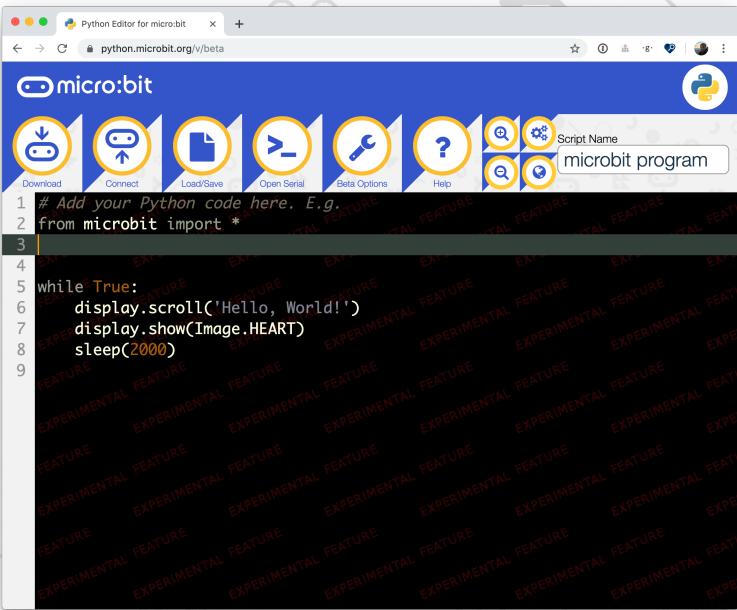
Install Mu:

<https://codewith.mu/>



OR

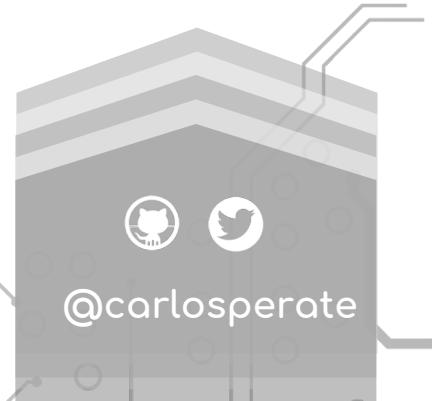
Online Editor (with Chrome):
<https://python.microbit.org/v/beta>



Hardware Drivers in MicroPython

CARLOS PEREIRA ATENCIO

Software Engineer @  micro:bit



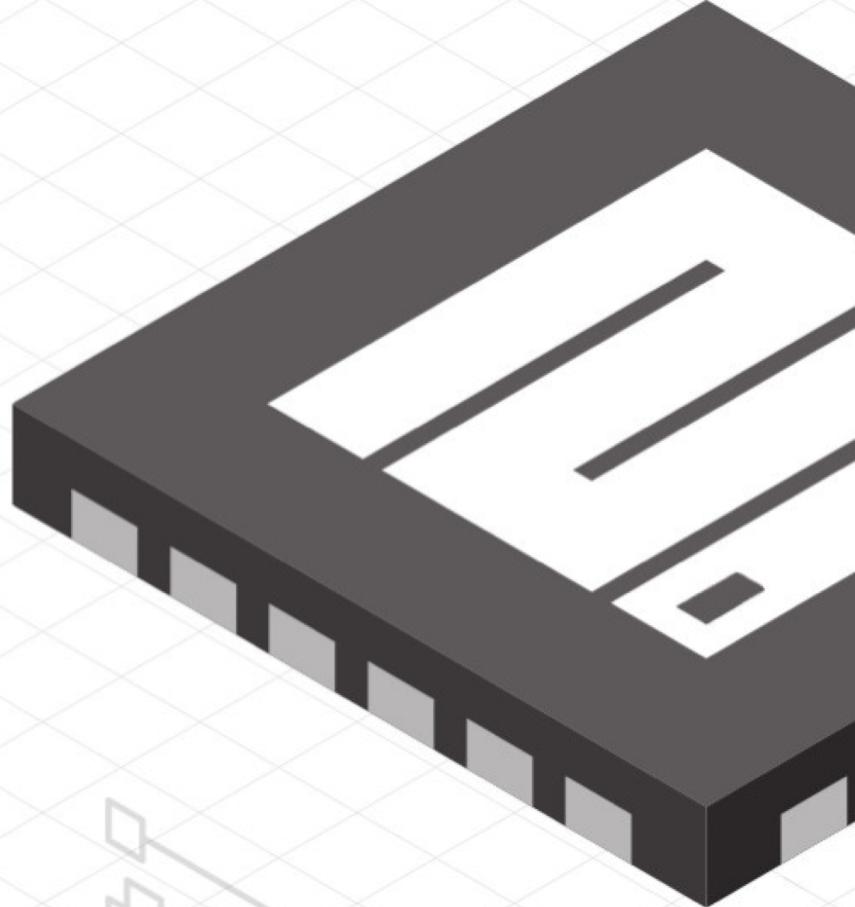
Outline

🐍 Presentation:

- 🐍 MicroPython
- 🐍 How to drive hardware
- 🐍 Communicating with other devices
- 🐍 Hardware documentation
- 🐍 Writing some code
- 🐍 Debugging
- 🐍 Workshop!

MicroPython

- Lean and efficient Python 3 implementation
- Can run in 16KBs RAM, 256KBs ROM
- Includes parser, compiler and virtual machine
- Small subset of the standard library
- Available in a wide range of platforms



MicroPython



Finding A Driver

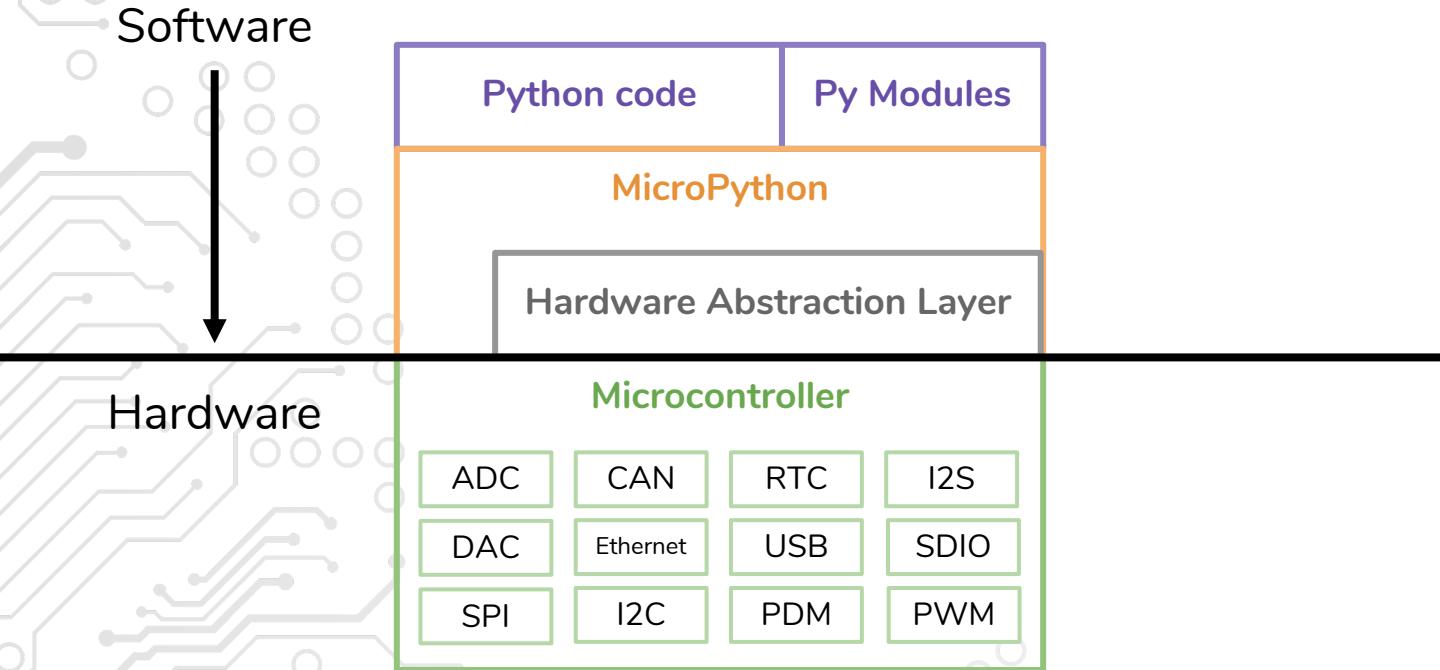
- **Google** is your friend
- GitHub is your **best** friend
- Use implementations in other languages as reference
- Look for the **official** documentation
 - Product page might contain additional documentation
 - Google search part number
 - Datasheets often found in random pages
 - Look for the newest version

```
def __init__(self, i2c, *, address=_MMA8451_DEFAULT_ADDRESS):
    self._device = i2c_device.I2CDevice(i2c, address)
    # Verify device ID.
    if self._read_u8(_MMA8451_REG_WHOAMI) != 0x1A:
        raise RuntimeError('Failed to find MMA8451, check wiring!')
    # Reset and wait for chip to be ready.
    self._write_u8(_MMA8451_REG_CTRL_REG2, 0x40)
    while self._read_u8(_MMA8451_REG_CTRL_REG2) & 0x40 > 0:
        pass
    # Enable 4G range.
    self._write_u8(_MMA8451_REG_XYZ_DATA_CFG, RANGE_4G)
    # High resolution mode.
    self._write_u8(_MMA8451_REG_CTRL_REG2, 0x02)
    # DRDY on INT1
    self._write_u8(_MMA8451_REG_CTRL_REG4, 0x01)
    self._write_u8(_MMA8451_REG_CTRL_REG5, 0x01)
```

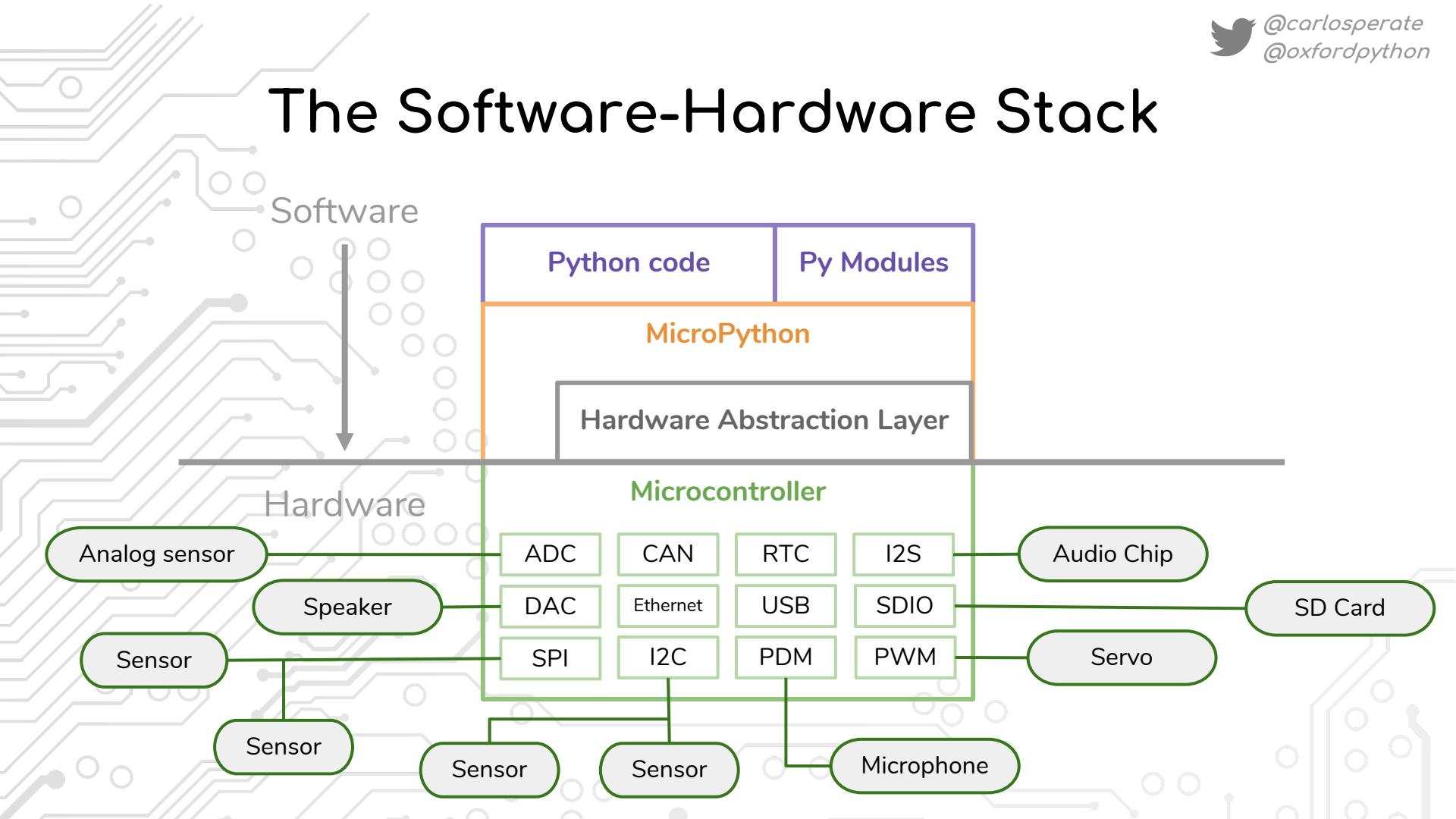
Adafruit CircuitPython mma8451 driver
https://github.com/adafruit/Adafruit_CircuitPython_MMA8451/

How Does Code Move Electrons

The Software Stack



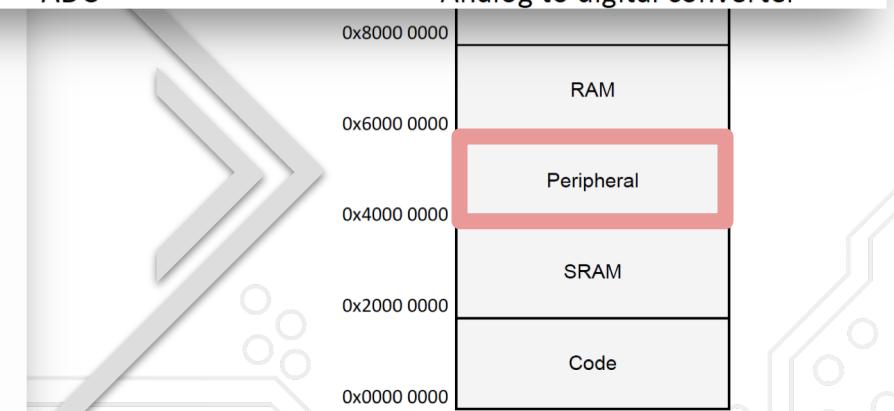
The Software-Hardware Stack



Memory Map And Peripherals

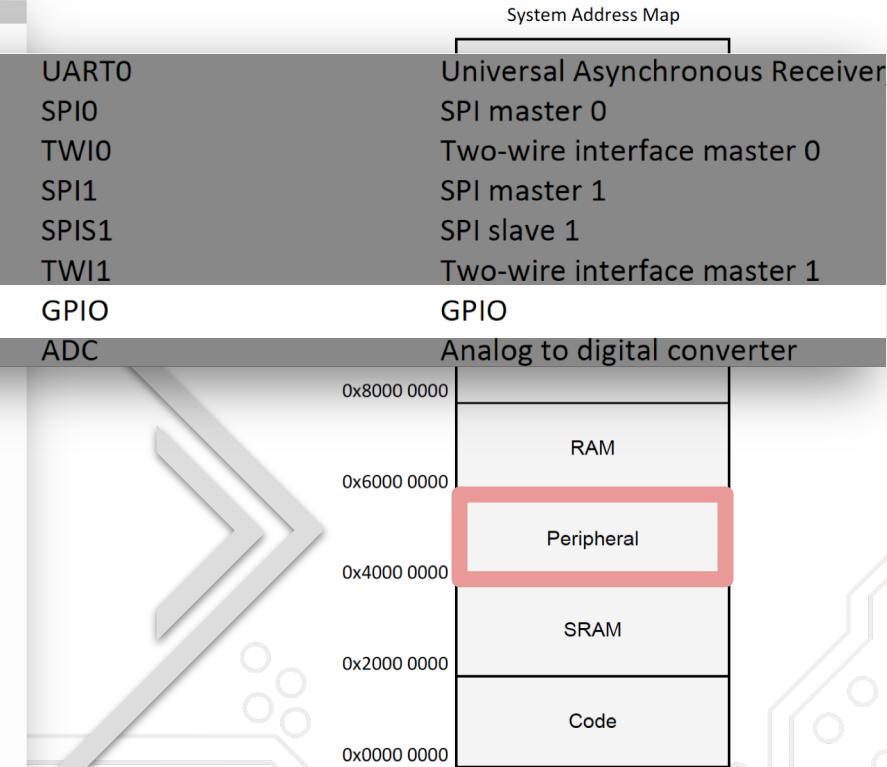
	Base address	Peripheral	Description
2	0x40002000	UART	UART0
3	0x40003000	SPI	SPI0
3	0x40003000	TWI	TWI0
4	0x40004000	SPI	SPI1
4	0x40004000	SPIS	SPIS1
4	0x40004000	TWI	TWI1
6	0x40006000	GPIO	GPIO
7	0x40007000	ADC	Analog to digital converter

	Base address	Peripheral	Description
	0x4000A000	TIMER	Timer 2
	0x4000B000	RTC	Real time counter 0
	0x4000C000	TEMP	Temperature Sensor
	0x4000D000	RNG	Random Number Generator
	0x4000E000	ECB	AES ECB Mode Encryption
	0x4000F000	AAR	Accelerated Address Resolver
	0x4000F000	CCM	AES CCM Mode Encryption
	0x40010000	WDT	Watchdog Timer
	0x40011000	RTC	Real time counter 1
	0x40012000	QDEC	Quadrature decoder
	0x40013000	LPCOMP	Low power comparator
	0x40014000	SWI	Software interrupt 0
	0x40015000	SWI	Software interrupt 1
	0x40016000	SWI	Software interrupt 2
	0x40017000	SWI	Software interrupt 3
	0x40018000	SWI	Software interrupt 4
	0x40019000	SWI	Software interrupt 5
	0x4001E000	NVMC	Non Volatile Memory Controller
	0x4001F000	PPI	PPI controller



Memory Map And Peripherals

	Base address	Peripheral	Description
0x40000000	CLOCK	Clock control	
2	0x40002000	UART	UART0
3	0x40003000	SPI	SPI0
3	0x40003000	TWI	TWI0
4	0x40004000	SPI	SPI1
4	0x40004000	SPIS	SPIS1
4	0x40004000	TWI	TWI1
6	0x40006000	GPIO	GPIO
7	0x40007000	ADC	Analog to digital converter
	0x4000A000	TIMER	
	0x4000B000	RTC	Real time counter 0
	0x4000C000	TEMP	Temperature Sensor
	0x4000D000	RNG	Random Number Generator
	0x4000E000	ECB	AES ECB Mode Encryption
	0x4000F000	AAR	Accelerated Address Resolver
	0x4000F000	CCM	AES CCM Mode Encryption
	0x40010000	WDT	Watchdog Timer
	0x40011000	RTC	Real time counter 1
	0x40012000	QDEC	Quadrature decoder
	0x40013000	LPCOMP	Low power comparator
	0x40014000	SWI	Software interrupt 0
	0x40015000	SWI	Software interrupt 1
	0x40016000	SWI	Software interrupt 2
	0x40017000	SWI	Software interrupt 3
	0x40018000	SWI	Software interrupt 4
	0x40019000	SWI	Software interrupt 5
	0x4001E000	NVMC	Non Volatile Memory Controller
	0x4001F000	PPI	PPI controller



Registers

Table 75: Register Overview

Register	Offset	Description
Registers		
<i>OUT</i>	0x504	Write GPIO port
<i>OUTSET</i>	0x508	Set individual bits in GPIO port
<i>OUTCLR</i>	0x50C	Clear individual bits in GPIO port
<i>IN</i>	0x510	Read GPIO port
<i>DIR</i>	0x514	Direction of GPIO pins
<i>DIRSET</i>	0x518	DIR set register
<i>DIRCLR</i>	0x51C	DIR clear register
<i>PIN_CNF[0]</i>	0x700	Configuration of GPIO pins
<i>PIN_CNF[1]</i>	0x704	Configuration of GPIO pins
<i>PIN_CNF[2]</i>	0x708	Configuration of GPIO pins
<i>PIN_CNF[3]</i>	0x70C	Configuration of GPIO pins
<i>PIN_CNF[4]</i>	0x710	Configuration of GPIO pins
<i>PIN_CNF[5]</i>	0x714	Configuration of GPIO pins

Registers

14.3 Register Details

Table 76: OUT

Bit number			31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Id			AF	AE	AC	AC	AB	AA	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
Reset			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Id	RW	Field	Value																								Description							
A			RW PIN0																								Pin 0							
B	RW	PIN1	Low																								Pin driver is low							
			High																								Pin driver is high							
B			Low																								Pin 1							
B	RW	PIN1	High																								Pin driver is low							
			High																								Pin driver is high							

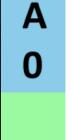
Registers

14.3 Register Details

Table 76: OUT

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	
Id	AF	AE	AC	AC	AB	AA	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Id	RW	Field	Value																											
A	RW	PINO	Value																											
High																														
1																														
Pin driver is high																														
B																														
Low																														
High																														
0																														
1																														
Pin driver is low																														
Pin driver is high																														

0
A
0



Registers

14.3 Register Details

Table 76: OUT

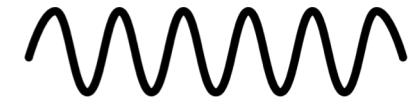
Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	
Id	AF	AE	AC	AC	AB	AA	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Id	RW	Field	Value Id																											
A	Low		0																										Pin driver is low	
B	RW	PIN1	Low																										Pin 1	
			High																										Pin driver is high	

0
A
0

Talking With Other Electronic Devices

Talking With Electronic Devices

- Some devices use analog signals as inputs or outputs
 - Analog sensors encode simple data in the voltage magnitude
 - Optic fibre modulates data using multiple techniques
- Most electronic components in a PCB use digital signals
- Data is sent in serial/parallel communication protocols
- We'll cover the most common digital protocols

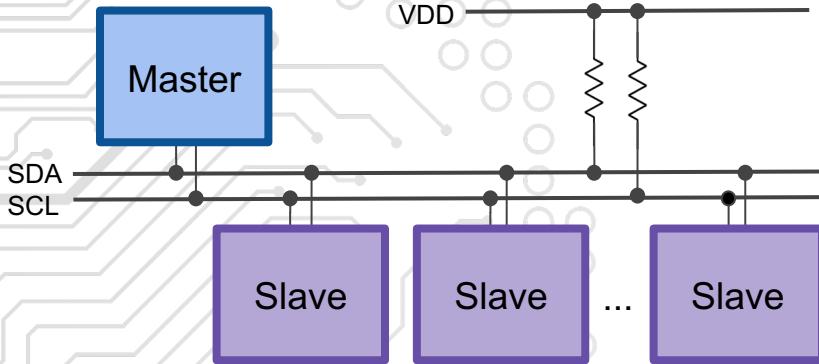


Analog Signal



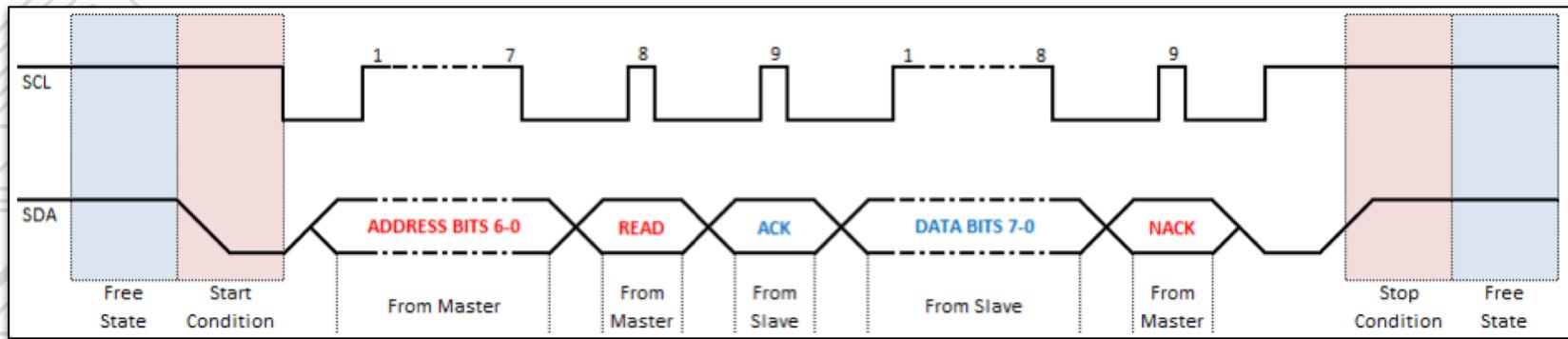
Digital Signal

I²C Protocol



- Inter-Integrated Circuit
- Synchronous serial protocol
- 1 clock signal, 1 data signal
- Single master, multiple slaves
 - Multi-master possible
- All sharing the same bus
- Each slave has individual address
- Commonly at 100 KHz or 400 KHz
 - sometimes higher speeds available
- SMB based on I²C

I²C Protocol



```
from microbit import i2c

i2c.write(SLAVE_ADDRESS, bytes([DATA...]), repeat=True)

data = i2c.read(SLAVE_ADDRESS, NUMBER_OF_BYTES)
```

How do we know what to send down the wire?

Datasheets!

Putting Things Together

Example Driver Initialisation

```
def __init__(self, i2c, *, address=_MMA8451_DEFAULT_ADDRESS):
    self._device = i2c_device.I2CDevice(i2c, address)
    # Verify device ID.
    if self._read_u8(_MMA8451_REG_WHOAMI) != 0x1A:
        raise RuntimeError('Failed to find MMA8451, check wiring!')
    # Reset and wait for chip to be ready.
    self._write_u8(_MMA8451_REG_CTRL_REG2, 0x40)
    while self._read_u8(_MMA8451_REG_CTRL_REG2) & 0x40 > 0:
        pass
    # Enable 4G range.
    self._write_u8(_MMA8451_REG_XYZ_DATA_CFG, RANGE_4G)
    # High resolution mode.
    self._write_u8(_MMA8451_REG_CTRL_REG2, 0x02)
    # DRDY on INT1
    self._write_u8(_MMA8451_REG_CTRL_REG4, 0x01)
    self._write_u8(_MMA8451_REG_CTRL_REG5, 0x01)
```

Example Driver Initialisation

```
def __init__(self, i2c, *, address=MMA8451_DEFAULT_ADDRESS):
    self._device = i2c_device.I2CDevice(i2c, address)
    # Verify device ID.
    if self._read_u8(_MMA8451_REG_WHOAMI) != 0x1A:
        raise RuntimeError('Failed to find MMA8451, check wiring!')
    # Reset and wait for chip to be ready.
    self._write_u8(_MMA8451_REG_CTRL_REG2, 0x40)
    while self._read_u8(_MMA8451_REG_CTRL_REG2) & 0x40 > 0:
        pass
    # Enable 4G range.
    self._write_u8(_MMA8451_REG_XYZ_DATA_CFG, RANGE_4G)
    # High resolution mode.
    self._write_u8(_MMA8451_REG_CTRL_REG2, 0x02)
    # DRDY on INT1
    self._write_u8(_MMA8451_REG_CTRL_REG4, 0x01)
    self._write_u8(_MMA8451_REG_CTRL_REG5, 0x01)
```

Example Driver Initialisation

6 Register Descriptions						
Table 12. Register address map						
Name	Type	Register address	Address increment	Auto-increment address	Default value	Hex value
		0x00000000		0x00000000	0x00000000	0x00000000
STATUS_REG[16]	A	0x001	0x1	0x00000000	0x00000000	0x00000000
OUT_X_AMG[8]	R	0x002	0x002	0x00000000	0x00000000	0x00000000
OUT_Y_AMG[8]	R	0x003	0x002	0x00000000	0x00000000	0x00000000
OUT_Z_AMG[8]	R	0x004	0x002	0x00000000	0x00000000	0x00000000
OUT_X_GYRO[8]	R	0x005	0x002	0x00000000	0x00000000	0x00000000
OUT_Y_GYRO[8]	R	0x006	0x002	0x00000000	0x00000000	0x00000000
OUT_Z_GYRO[8]	R	0x007	0x002	0x00000000	0x00000000	0x00000000
Reserved	R	0x008	—	—	—	—
Reserved	R	0x009	—	—	—	—
FIFO_CTRL[8]	R/W	0x00A	0x001	0x00000000	0x00000000	0x00000000

WHO_AM_I⁽¹⁾ R 0x0D 0x0E 00011010 0x1A Device ID (0x1A) pass

PL_COF1[16]	R/W	0x011	0x1	0x00000000	0x00000000	Landspace/microstrain detection.
PL_COEF2[16]	R/W	0x012	0x1	0x00000000	0x00000000	Landspace/microstrain detection.
PL_M_2000HZ[8]	R/W	0x013	0x1	0x00000000	0x00000000	Backplane 2.0 Hz low pass filter.
P_XL_16BIT[8]	R/W	0x014	0x1	0x00000000	0x00000000	16 bit resolution.
P_YL_16BIT[8]	R/W	0x015	0x1	0x00000000	0x00000000	16 bit resolution.
P_ZL_16BIT[8]	R/W	0x016	0x1	0x00000000	0x00000000	16 bit resolution.
PT_M_16BIT[8]	R/W	0x017	0x1	0x00000000	0x00000000	16 bit resolution.
PT_XL_16BIT[8]	R/W	0x018	0x1	0x00000000	0x00000000	16 bit resolution.
PT_YL_16BIT[8]	R/W	0x019	0x1	0x00000000	0x00000000	16 bit resolution.
PT_ZL_16BIT[8]	R/W	0x01A	0x1	0x00000000	0x00000000	16 bit resolution.
PT_M_COEF[8]	R/W	0x01B	0x1	0x00000000	0x00000000	Frontend detection coefficient.
PT_XL_COEF[8]	R/W	0x01C	0x1	0x00000000	0x00000000	Frontend detection coefficient.
PT_YL_COEF[8]	R/W	0x01D	0x1	0x00000000	0x00000000	Frontend detection coefficient.
PT_ZL_COEF[8]	R/W	0x01E	0x1	0x00000000	0x00000000	Frontend detection coefficient.
TRANSIENT[8]	R/W	0x01F	0x1	0x00000000	0x00000000	Transient event register.
TRANSIENT_THRESH[8]	R/W	0x020	0x1	0x00000000	0x00000000	Transient event threshold.
TRANSIENT_COUNT[8]	R/W	0x021	0x1	0x00000000	0x00000000	Transient detection counter.

20 Sensors
NXP Semiconductors N.V.

```
def __init__(self, i2c, *, address=_MMA8451_DEFAULT_ADDRESS):
    self.device = i2c.device.I2CDevice(i2c, address)

    # Verify device ID.
    if self._read_u8(_MMA8451_REG_WHOAMI) != 0x1A:
        raise RuntimeError('Failed to find MMA8451, check wiring!')

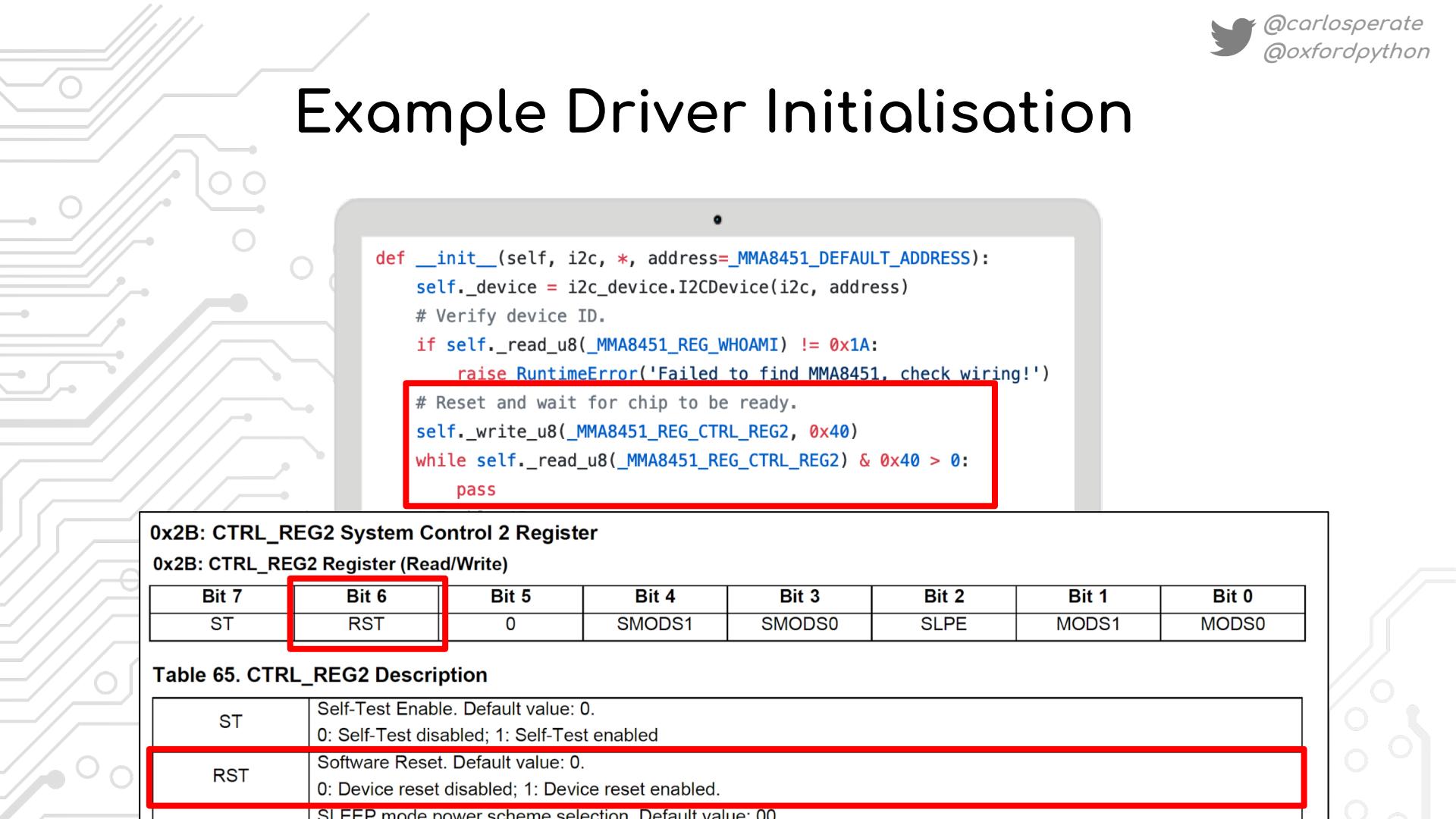
    # Reset and wait for chip to be ready.
    self.write_u8(_MMA8451_REG_CTRL_REG2, 0x40)

    # Enable 4G range.
    self._write_u8(_MMA8451_REG_XYZ_DATA_CFG, RANGE_4G)

    # High resolution mode.
    self._write_u8(_MMA8451_REG_CTRL_REG2, 0x02)

    # DRDY on INT1
    self._write_u8(_MMA8451_REG_CTRL_REG4, 0x01)
    self._write_u8(_MMA8451_REG_CTRL_REG5, 0x01)
```

Example Driver Initialisation



```
def __init__(self, i2c, *, address=_MMA8451_DEFAULT_ADDRESS):
    self._device = i2c_device.I2CDevice(i2c, address)
    # Verify device ID.
    if self._read_u8(_MMA8451_REG_WHOAMI) != 0x1A:
        raise RuntimeError('Failed to find MMA8451, check wiring!')
    # Reset and wait for chip to be ready.
    self._write_u8(_MMA8451_REG_CTRL_REG2, 0x40)
    while self._read_u8(_MMA8451_REG_CTRL_REG2) & 0x40 > 0:
        pass
```

0x2B: CTRL_REG2 System Control 2 Register

0x2B: CTRL_REG2 Register (Read/Write)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ST	RST	0	SMODS1	SMODS0	SLPE	MODS1	MODS0

Table 65. CTRL_REG2 Description

ST	Self-Test Enable. Default value: 0. 0: Self-Test disabled; 1: Self-Test enabled
RST	Software Reset. Default value: 0. 0: Device reset disabled; 1: Device reset enabled.

Example Driver Initialisation

```
def __init__(self, i2c, *, address=_MMA8451_DEFAULT_ADDRESS):
    self._device = i2c_device.I2CDevice(i2c, address)
    # Verify device ID.
    if self._read_u8(_MMA8451_REG_WHOAMI) != 0x1A:
        raise RuntimeError('Failed to find MMA8451, check wiring!')
    # Reset and wait for chip to be ready.
    self._write_u8(_MMA8451_REG_CTRL_REG2, 0x40)
    while self._read_u8(_MMA8451_REG_CTRL_REG2) & 0x40 > 0:
        pass
    # Enable 4G range.
    self._write_u8(_MMA8451_REG_XYZ_DATA_CFG, RANGE_4G)
    # High resolution mode.
    self._write_u8(_MMA8451_REG_CTRL_REG2, 0x02)
    # DRDY on INT1
    self._write_u8(_MMA8451_REG_CTRL_REG4, 0x01)
    self._write_u8(_MMA8451_REG_CTRL_REG5, 0x01)
```

Code Tips

- For constant values use `const()` for compiler optimisation
- `ustruct` very useful to pack/unpack blocks of data to/from variables
- Bitwise operations (`>>` `<<` `&` `|` `^` `~`) to target bits in registers
- Need to be careful about memory allocation (including temp ones!)
- Buffer preallocation for driver read data is more efficient
- The `gc` and `micropython` modules can be used to control collection and profile memory
- Documentation
 - http://docs.micropython.org/en/latest/reference/speed_python.html
 - <http://docs.micropython.org/en/latest/reference/constrained.html>

What if Something Goes Wrong?

Debugging



- Check all connections and power supply
- **Double** check all connections and power supply
- Make sure all devices **share ground**
- Make sure all devices have **same logic level** (5V vs 3.3V)
- **Reduce** the data transfer **speed**
- Find the **simplest** possible code **example** to replicate your issue
- Search for application notes and other implementations
- Search for **erratas**

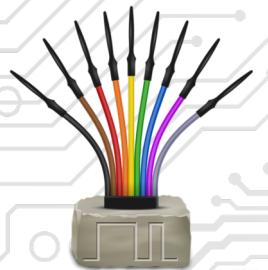
Debugging



Capture and retrieve the data sent and received

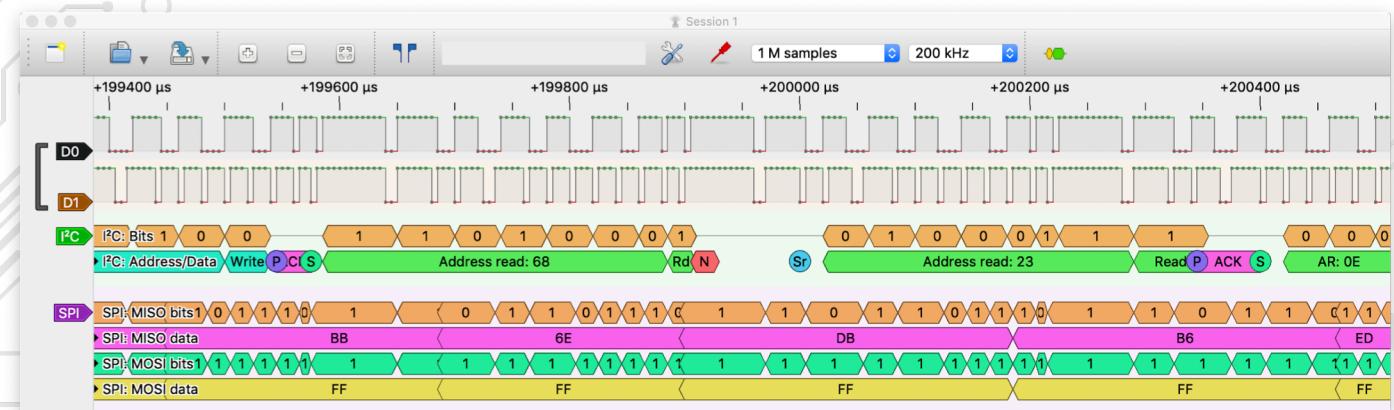
- Print debug
- Hardware debuggers
 - SEGGER J-Link EDU Mini
 - Black Magic Probe
 - Some development boards have integrated debuggers
 - DAPLink, ST-Link, J-Link
- Logic analyser
- LEDs

Logic Analysers



sigrok

- **Saleae** is a nice entry level professional tool
- **SigRok** (with PulseView) is an open source signal analysis software
 - Low cost SigRok-compatible devices start at £10



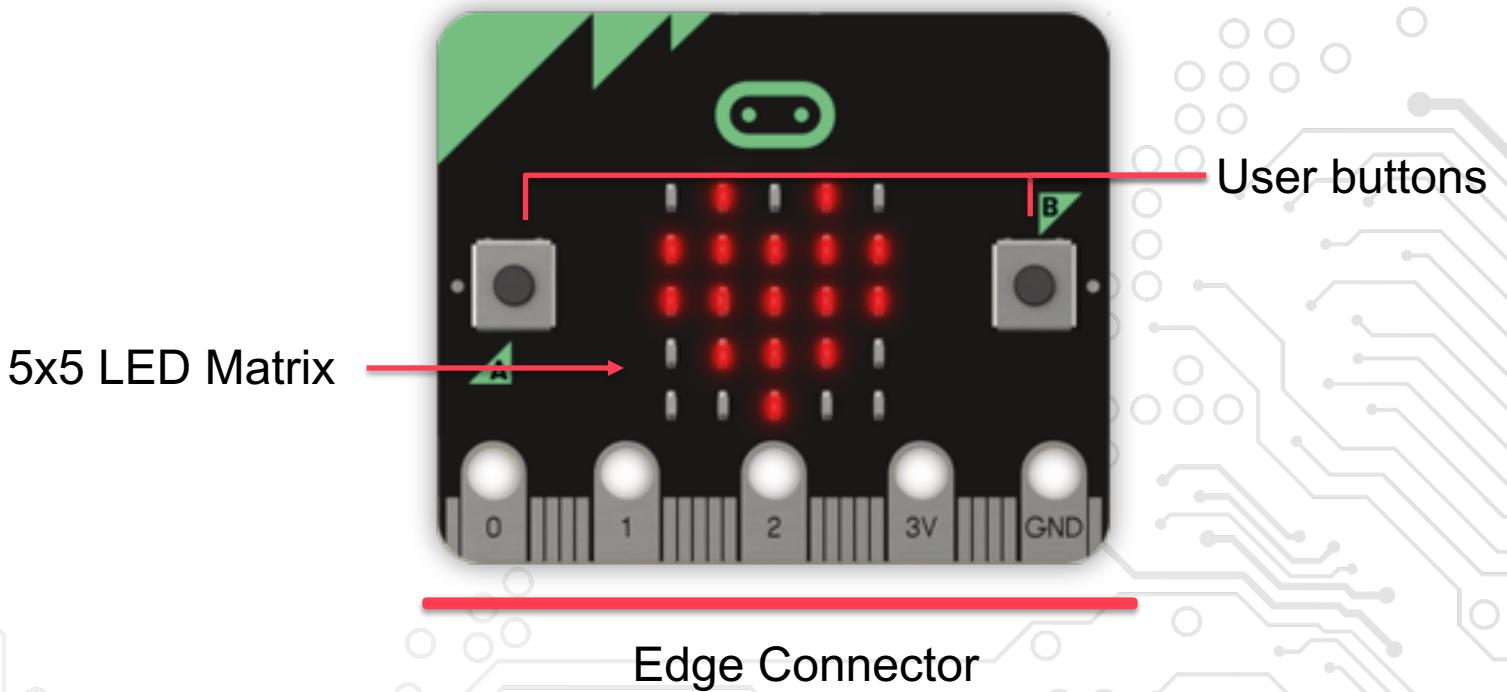
Let's get coding!

<https://github.com/carlosperate/micropython-driver-workshop>

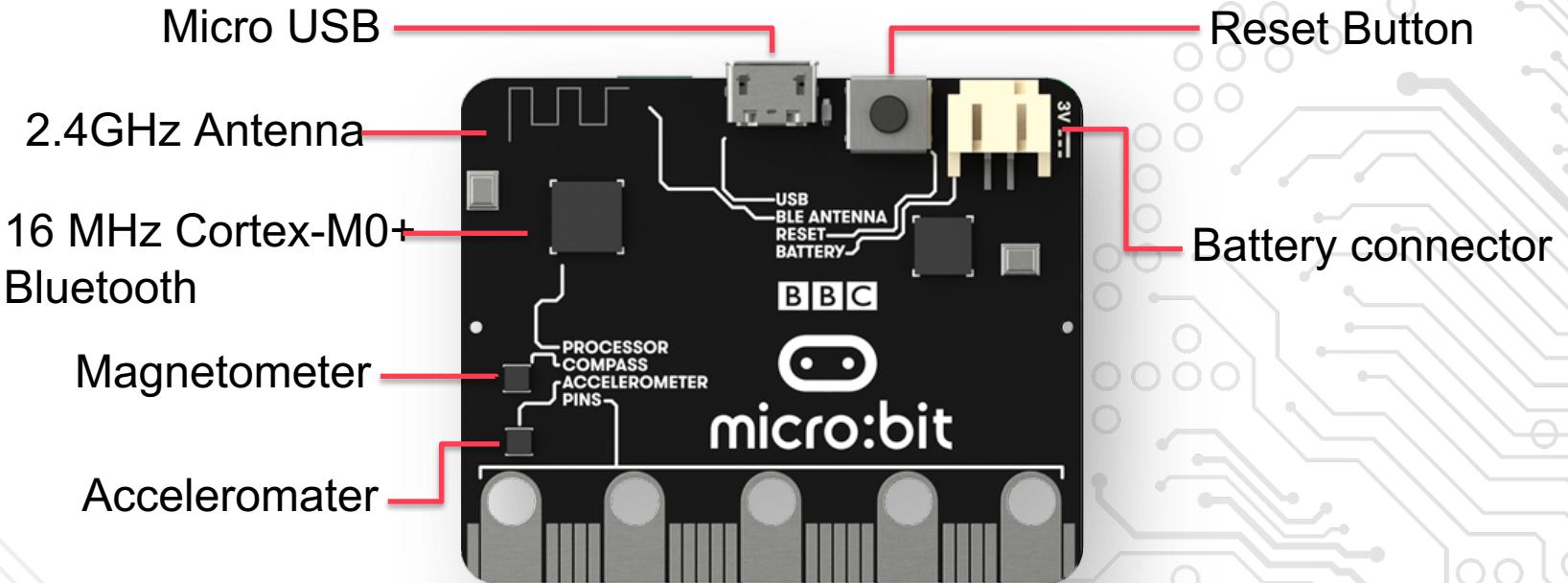
OR

<https://git.io/Je3tM>

BBC micro:bit



BBC micro:bit



Code Tips

29 == 0x1D

- For constant values use `micropython.const()` for compiler optimisation
 - `VARIABLE = const(0x55)`
- Creating a “buffer” (to send via I2C):
 - `bytes([DATA_LIST])`
- `ustruct` very useful to pack/unpack blocks of data to/from variables
 - `output_tuple = ustruct.unpack('b', b'\x55')`
 - <https://docs.micropython.org/en/latest/library/ustruct.html>
- Bitwise operations (`>>` `<<` `&` `|` `^` `~`) to target bits in registers
 - Shift a bit 3 places to the left: `result = 1 << 3`
 - Set bit 3: `result = original | (1 << 3)`
 - Flip all bits: `0b11110111 = ~0b00001000`
 - Clear bit 3: `result = original & ~(1 << 3)`
 - <https://wiki.python.org/moin/BitwiseOperators>