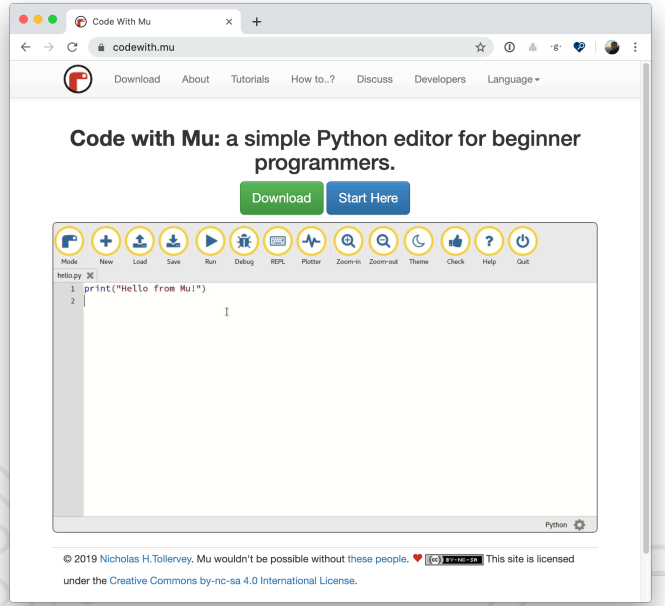
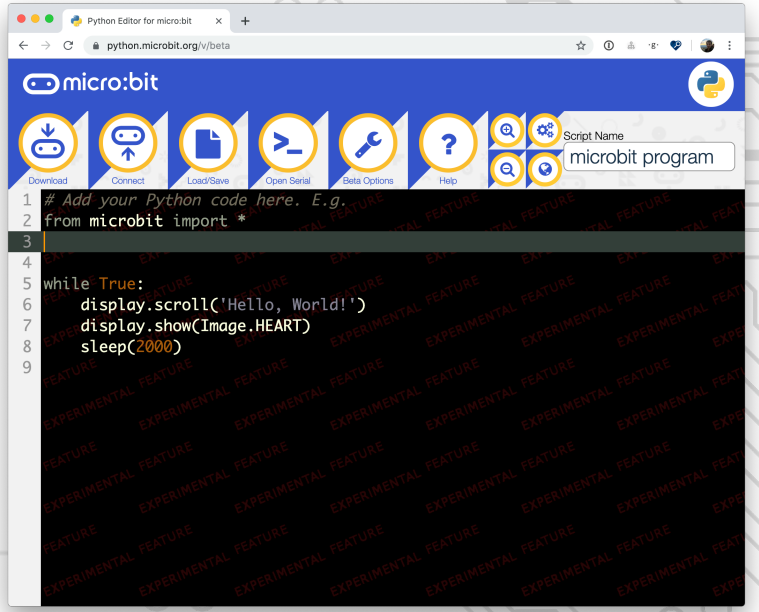
Software You'll Need

Install Mu:
https://codewith.mu/



OR

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



@carlosperate
@oxfordpython

Hardware Drivers in MicroPython

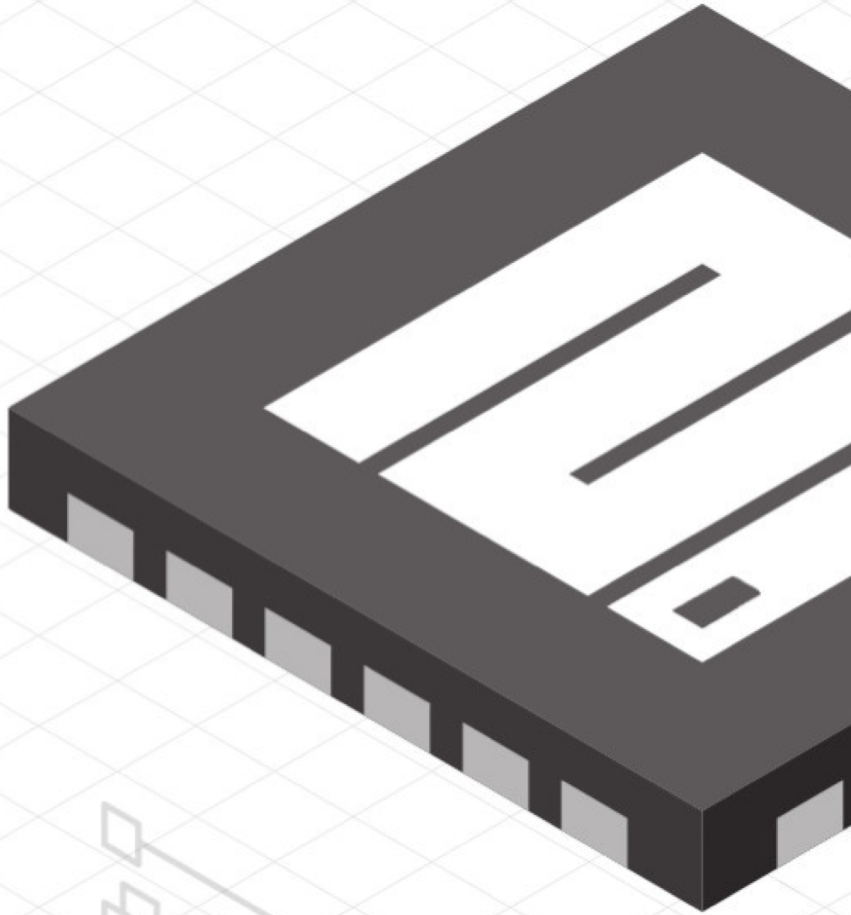
CARLOS PEREIRA ATENCIO
Software Engineer @ micro:bit

Outline

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

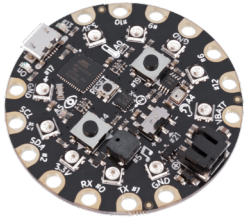
MicroPython

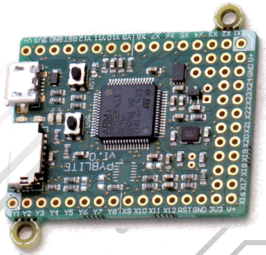
- Lean and efficient Python 3 implementation
- Can run in 16KBs RAM, 256KBs ROM
- Includes parser, compiler and virtual machine
- Small subset of the standard library
- Available in a wide range of platforms

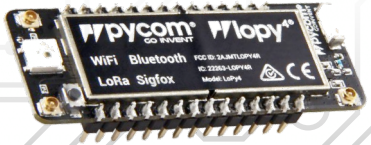


@carlosperate
@oxfordpython

MicroPython

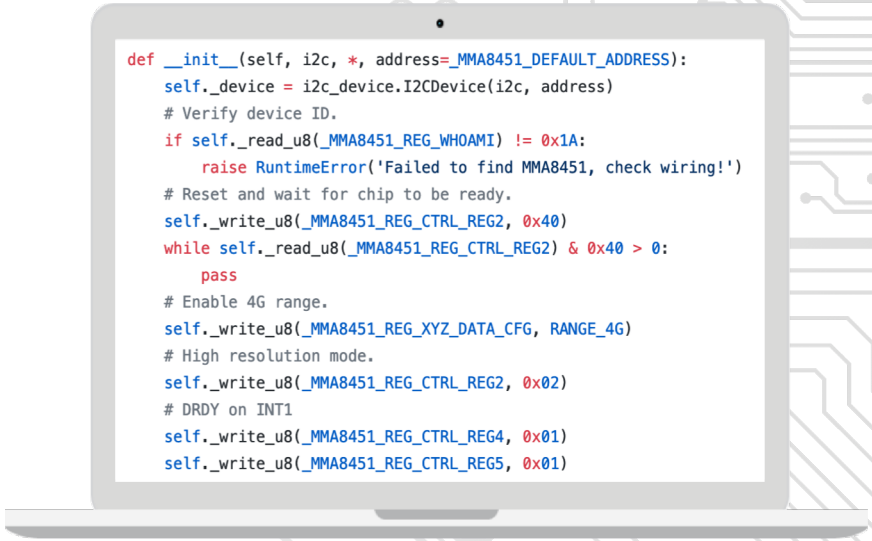






Finding A Driver

- **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



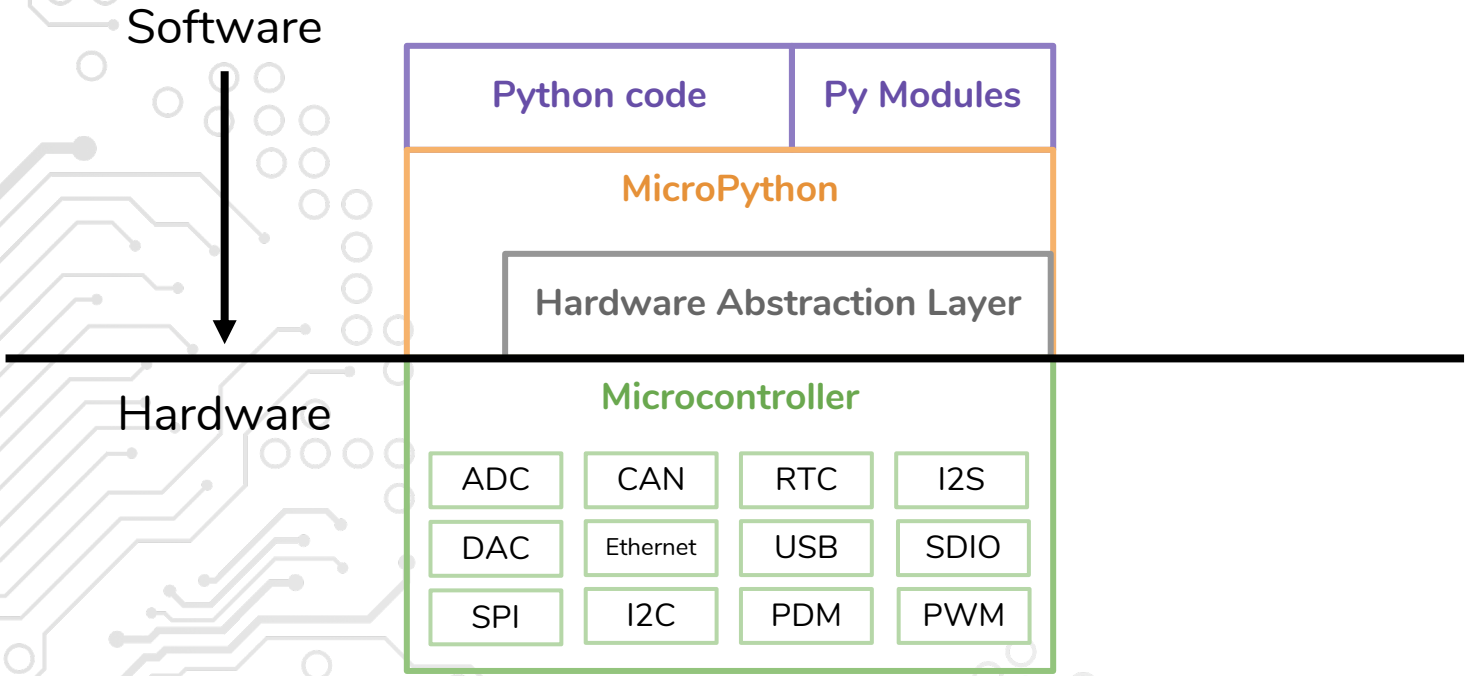
```
def __init__(self, i2c, *, address=_MMA8451_DEFAULT_ADDRESS):
    self._device = i2c_device.I2CDevice(i2c, address)
    # Verify device ID.
    if self._read_u8(_MMA8451_REG_WHOAMI) != 0x1A:
        raise RuntimeError('Failed to find MMA8451, check wiring!')
    # Reset and wait for chip to be ready.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

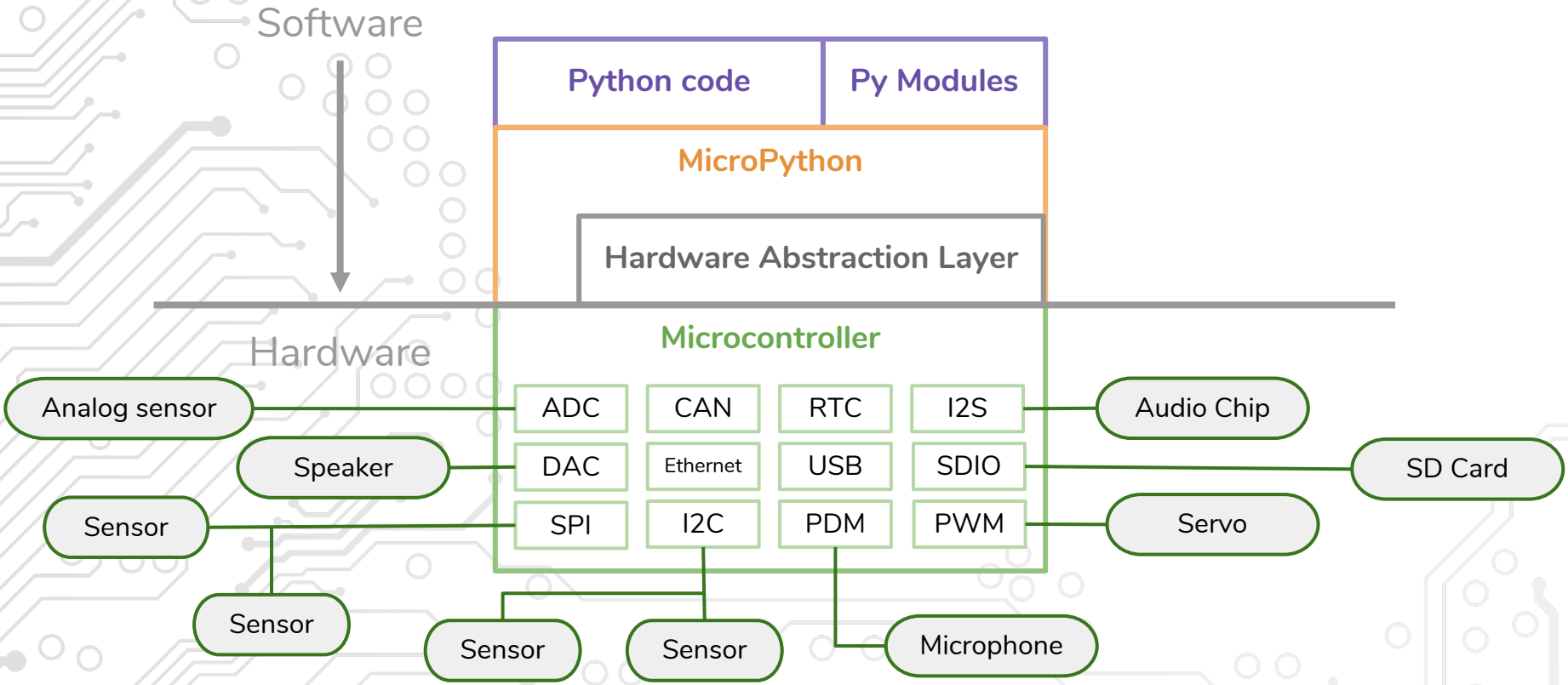
@carlosperate
@oxfordpython

How Does Code Move Electrons

The Software Stack



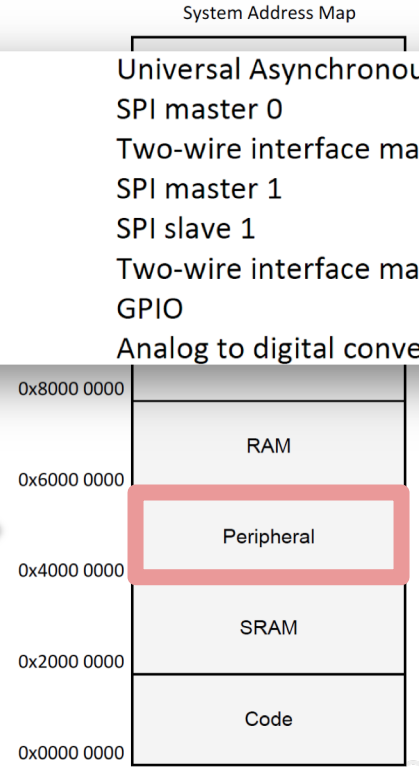
The Software-Hardware Stack



Memory Map And Peripherals

Base address	Peripheral	Description
0x40000000	CLOCK	Clock control
0x40000000	POWER	Power control
0x40001000	RADIO	2.4 GHz Radio
0x40002000	UART	Universal Asynchronous Receiver/Transmitter
0x40003000	SPI	SPI master
0x40003000	TWI	Two-wire interface master
0x40004000	SPI	SPI master
0x40004000	SPIS	SPI slave
0x40004000	TWI	Two-wire interface master
0x40006000	GPIO	GPIO
0x40007000	ADC	Analog to digital converter
0x40008000	TIMER	Timer 0
0x40009000	TIMER	Timer 1
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

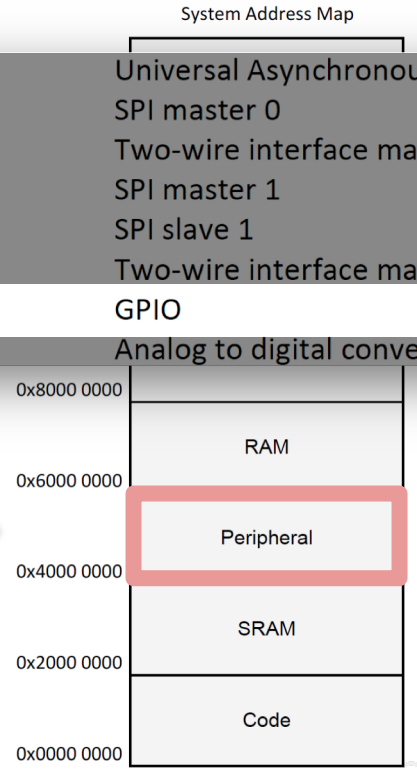
2	0x40002000	UART	UART0	Universal Asynchronous Receiver
3	0x40003000	SPI	SPI0	SPI master 0
3	0x40003000	TWI	TWI0	Two-wire interface master 0
4	0x40004000	SPI	SPI1	SPI master 1
4	0x40004000	SPIS	SPIS1	SPI slave 1
4	0x40004000	TWI	TWI1	Two-wire interface master 1
6	0x40006000	GPIO	GPIO	GPIO
7	0x40007000	ADC	ADC	Analog to digital converter



Memory Map And Peripherals

Base address	Peripheral	Description
0x40000000	CLOCK	Clock control
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

2	0x40002000	UART	UART0	Universal Asynchronous Receiver
3	0x40003000	SPI	SPI0	SPI master 0
3	0x40003000	TWI	TWI0	Two-wire interface master 0
4	0x40004000	SPI	SPI1	SPI master 1
4	0x40004000	SPIS	SPIS1	SPI slave 1
4	0x40004000	TWI	TWI1	Two-wire interface master 1
6	0x40006000	GPIO	GPIO	GPIO
7	0x40007000	ADC	ADC	Analog to digital converter



Registers

Table 75: Register Overview

Register	Offset	Description
Registers		
OUT	0x504	Write GPIO port
OUTSET	0x508	Set individual bits in GPIO port
OUTCLR	0x50C	Clear individual bits in GPIO port
IN	0x510	Read GPIO port
DIR	0x514	Direction of GPIO pins
DIRSET	0x518	DIR set register
DIRCLR	0x51C	DIR clear register
PIN_CNF[0]	0x700	Configuration of GPIO pins
PIN_CNF[1]	0x704	Configuration of GPIO pins
PIN_CNF[2]	0x708	Configuration of GPIO pins
PIN_CNF[3]	0x70C	Configuration of GPIO pins
PIN_CNF[4]	0x710	Configuration of GPIO pins
PIN_CNF[5]	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	AD	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 Id**	**Value**								**Description**																							
A	RW	PIN0										Pin 0																							
			Low	0								Pin driver is low																							
			High	1								Pin driver is high																							
B	RW	PIN1										Pin 1																							
			Low	0								Pin driver is low																							
			High	1								Pin driver is high																							

Registers

14.3 Register Details

Table 76: OUT

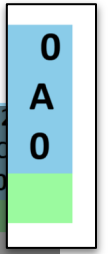
Bit number				31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
Id				AF	AE	AD	AC	AB	AA	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C
Reset				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Id	RW	Field	Value Id	Value																													
A	RW	PIN0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

Pin 0

Pin driver is high

Pin driver is low

Pin driver is high

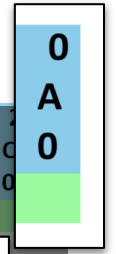


Registers

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	2
Id				AF	AE	AD	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	Value	Description																												
A																																	
			Low	0	Pin driver is low																												
			High	1	Pin driver is high																												
B	RW	PIN1			Pin 1																												
			Low	0	Pin driver is low																												
			High	1	Pin driver is high																												



@carlosperate
@oxfordpython

Talking With Other Electronic Devices

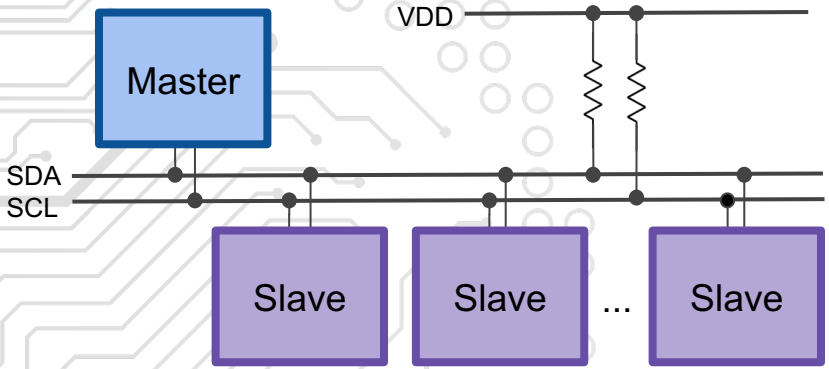
Talking With Electronic Devices

- 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

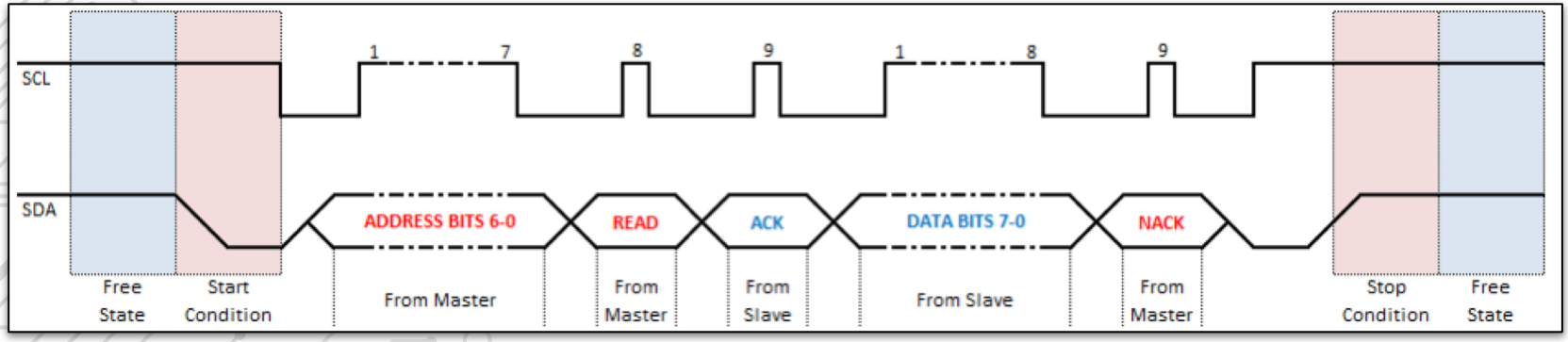
Digital Signal

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)
```

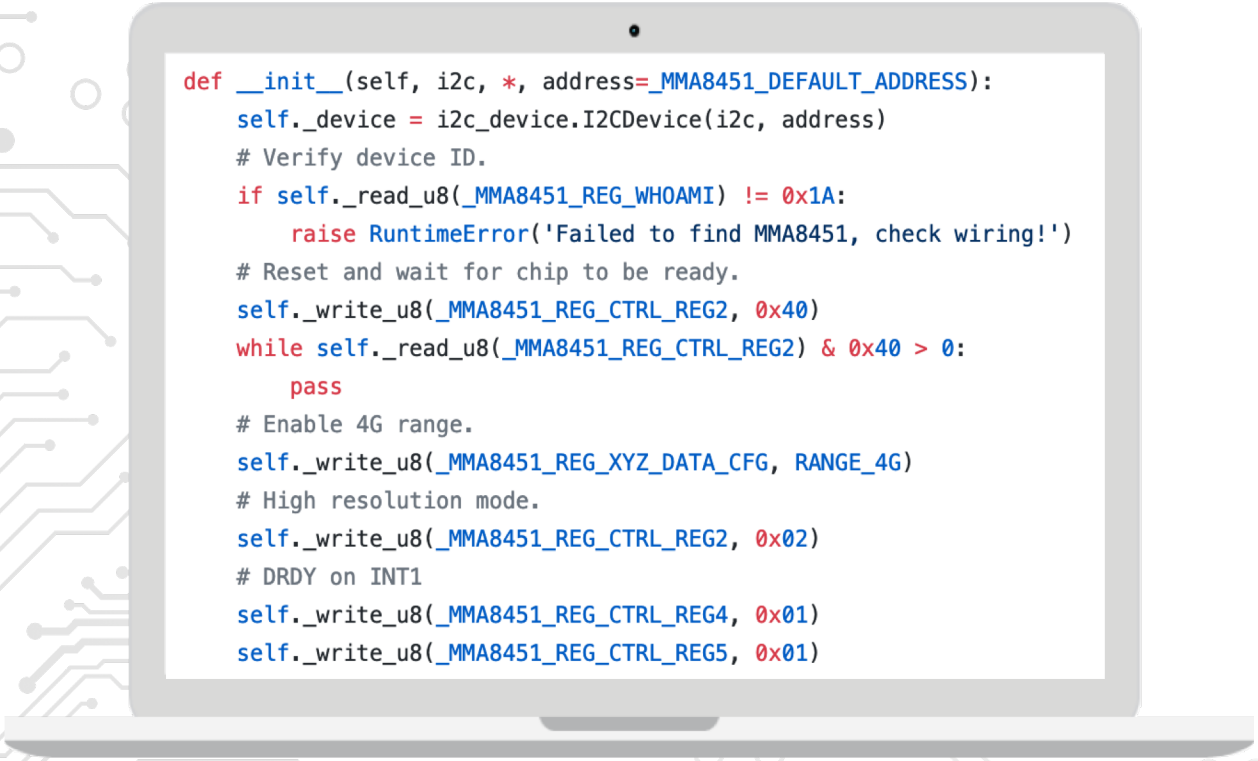
Protocol diagram from: https://rheingoldheavy.com/i2c-signals/

How do we know what to send down the wire?

Datasheets!

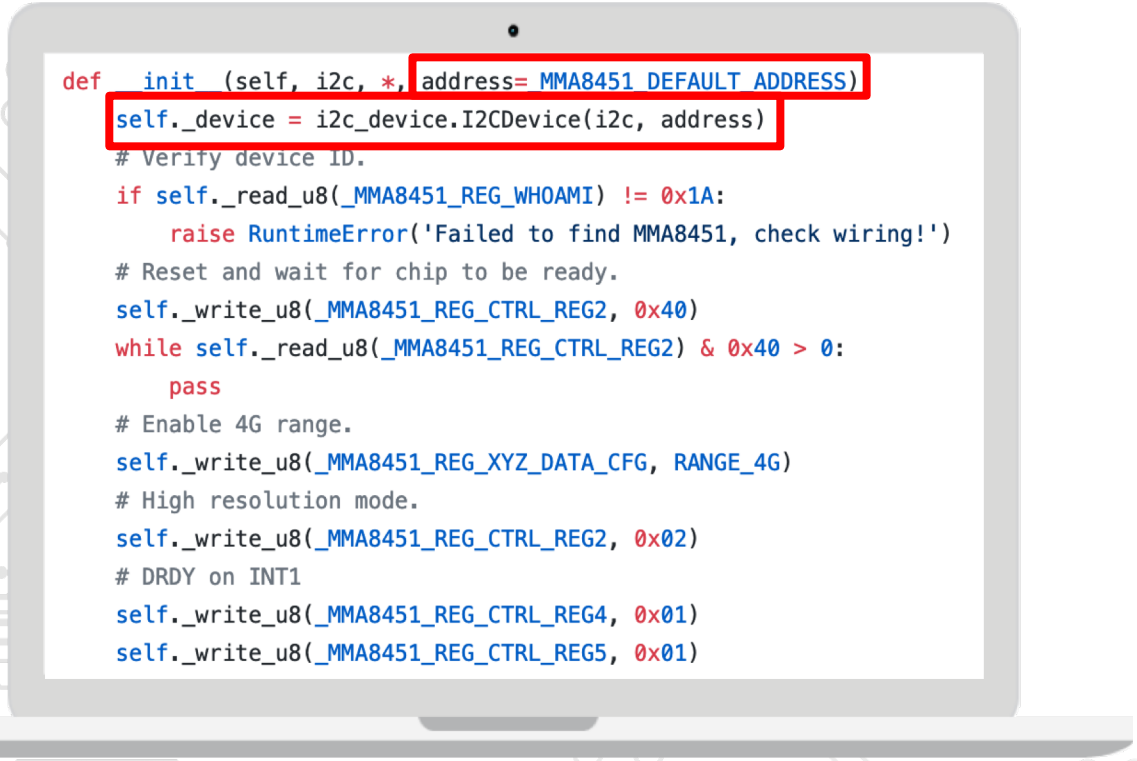
Putting Things Together

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

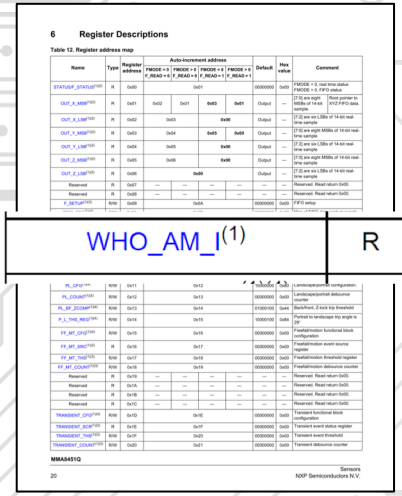

6 Register Descriptions

Table 12. Register address map

Name	Type	Register address	Auto-increment address: FMODE = 0 F_READ = 0	FMODE = 0 F_READ = 1	FMODE = 1 F_READ = 0	FMODE = 1 F_READ = 1	Default	Hex value	Comment
STATUS/F_STATUS[1][2]	R	0x00	0x01				00000000	0x00	FMODE = 0, real time status FMODE = 0, FIFO status
OUT_X_MSB[1][2]	R	0x01	0x02	0x01	0x03	0x01	Output	—	[7:0] are eight MSBits of 14-bit sample.
OUT_X_LSB[1][2]	R	0x02	0x03		0x00		Output	—	[7:2] are six LSBs of 14-bit real-time sample
OUT_Y_MSB[1][2]	R	0x03	0x04		0x05	0x03	Output	—	[7:0] are eight MSBits of 14-bit real-time sample
OUT_Y_LSB[1][2]	R	0x04	0x05		0x00		Output	—	[7:2] are six LSBs of 14-bit real-time sample
OUT_Z_MSB[1][2]	R	0x05	0x06		0x00		Output	—	[7:0] are eight MSBits of 14-bit real-time sample
OUT_Z_LSB[1][2]	R	0x06	0x00				Output	—	[7:2] are six LSBs of 14-bit real-time sample
Reserved	R	0x07	—	—	—	—	—	—	Reserved. Read return 0x00.
Reserved	R	0x08	—	—	—	—	—	—	Reserved. Read return 0x00.
F_SETUP[1][2]	R/W	0x09	0x0A				00000000	0x00	FIFO setup
TRIG_CFG[1][2]	R/W	0x0A	0x0B				00000000	0x00	Map of FIFO data capture events
SYSMOD[1]	R	0x0B	0x0C				00000000	0x00	System mode register
INT_SOURCE[1][2]	R	0x0C	0x0D				00000000	0x00	System interrupt status register
WHO_AM_I[1][2]	R	0x0D	0x0E				00011010	0x1A	Device ID (0x1A)
XYZ_DATA_CFG[1][2]	R/W	0x0E	0x0F				00000000	0x00	Dynamic range settings
HP_FILTER_CUTOFF[1][2]	R/W	0x0F	0x10				00000000	0x00	Cut-off frequency is set to 16 Hz
PL_STATUS[1][2]	R	0x10	0x11				00000000	0x00	Landscape/portrait orientation status
PL_CFG[1][2][3]	R/W	0x11	0x12				00000000	0x00	Landscape/portrait configuration.
PL_COUNT[1][2][3]	R/W	0x12	0x13				00000000	0x00	Landscape/portrait debounce counter
PL_BF_ZCOMP[1][2][3]	R/W	0x13	0x14				01000100	0x44	Back/Front_Z-lock trip threshold
P_L_THS_REG[1][2][3]	R/W	0x14	0x15				10000100	0x84	Portrait to landscape trip angle is 29°
FF_MT_CFG[1][2][3]	R/W	0x15	0x16				00000000	0x00	Freefall/motion functional block configuration
FF_MT_SRC[1][2][3]	R	0x16	0x17				00000000	0x00	Freefall/motion event source register
FF_MT_THS[1][2][3]	R/W	0x17	0x18				00000000	0x00	Freefall/motion threshold register
FF_MT_COUNT[1][2][3]	R/W	0x18	0x19				00000000	0x00	Freefall/motion debounce counter
Reserved	R	0x19	—	—	—	—	—	—	Reserved. Read return 0x00.
Reserved	R	0x1A	—	—	—	—	—	—	Reserved. Read return 0x00.
Reserved	R	0x1B	—	—	—	—	—	—	Reserved. Read return 0x00.
Reserved	R	0x1C	—	—	—	—	—	—	Reserved. Read return 0x00.
TRANSIENT_CFG[1][2][3]	R/W	0x1D	0x1E				00000000	0x00	Transient functional block configuration
TRANSIENT_SCR[1][2][3]	R	0x1E	0x1F				00000000	0x00	Transient event status register
TRANSIENT_THS[1][2][3]	R/W	0x1F	0x20				00000000	0x00	Transient event threshold
TRANSIENT_COUNT[1][2][3]	R/W	0x20	0x21				00000000	0x00	Transient debounce counter

MMA8451Q

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)
    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
```

0x2B: CTRL_REG2 System Control 2 Register

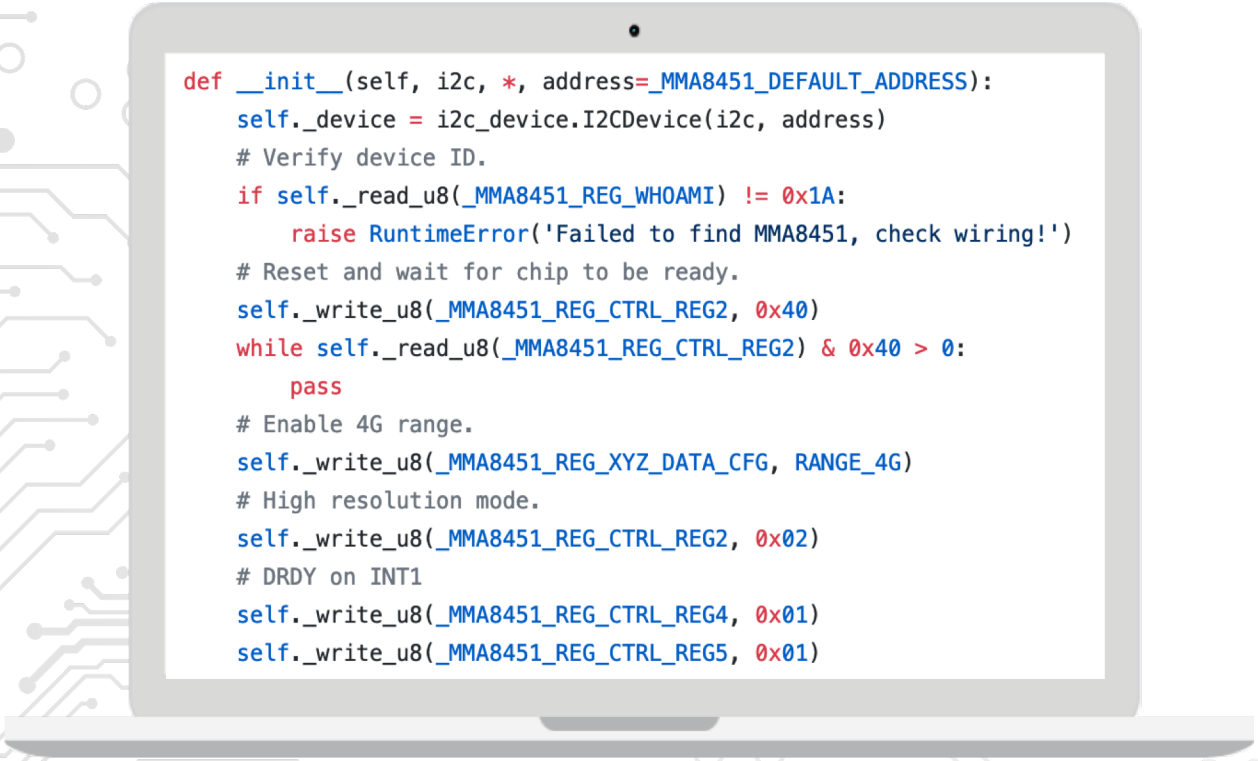
0x2B: CTRL_REG2 Register (Read/Write)

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.
	SLEEP mode power scheme selection. Default value: 00

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**

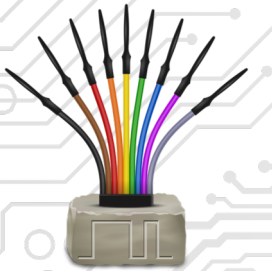
Debugging 

Capture and **retrieve** the data sent and received

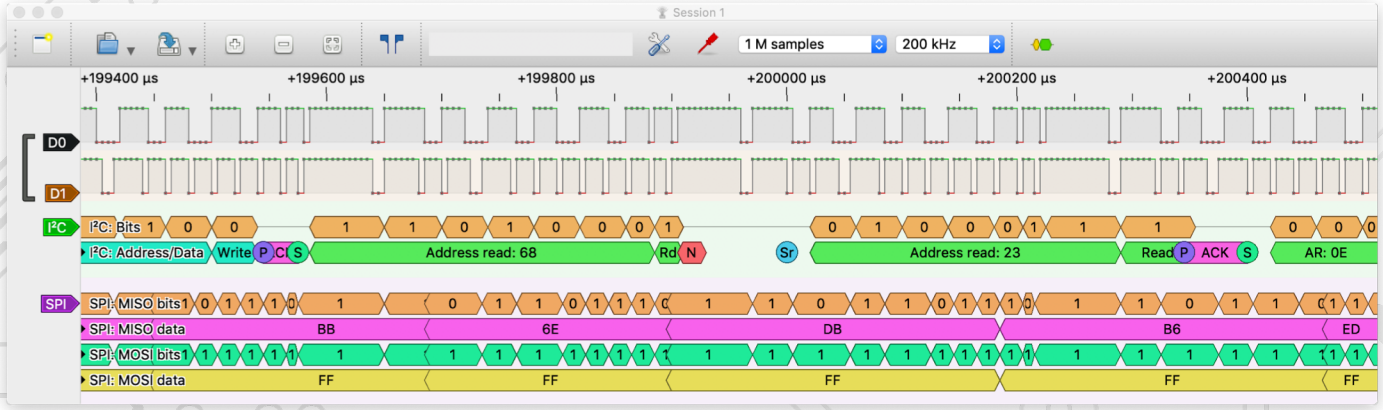
- Print debug
- Hardware debuggers
  - SEGGER J-Link EDU Mini
  - Black Magic Probe
  - Some development boards have integrated debuggers
    - ➢ DAPLink, ST-Link, J-Link
- Logic analyser
- LEDs

Logic Analysers

- **Saleale** 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**



sigrok



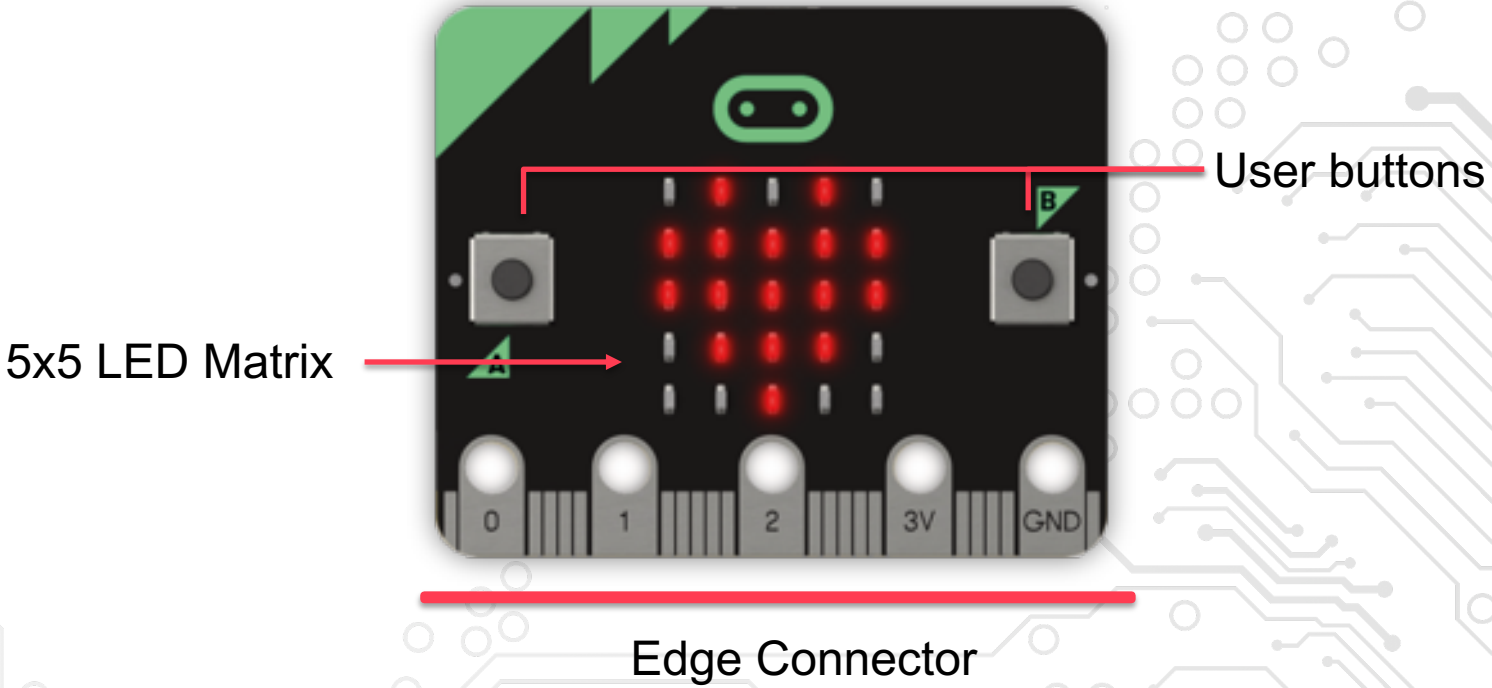
Let's get coding!

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

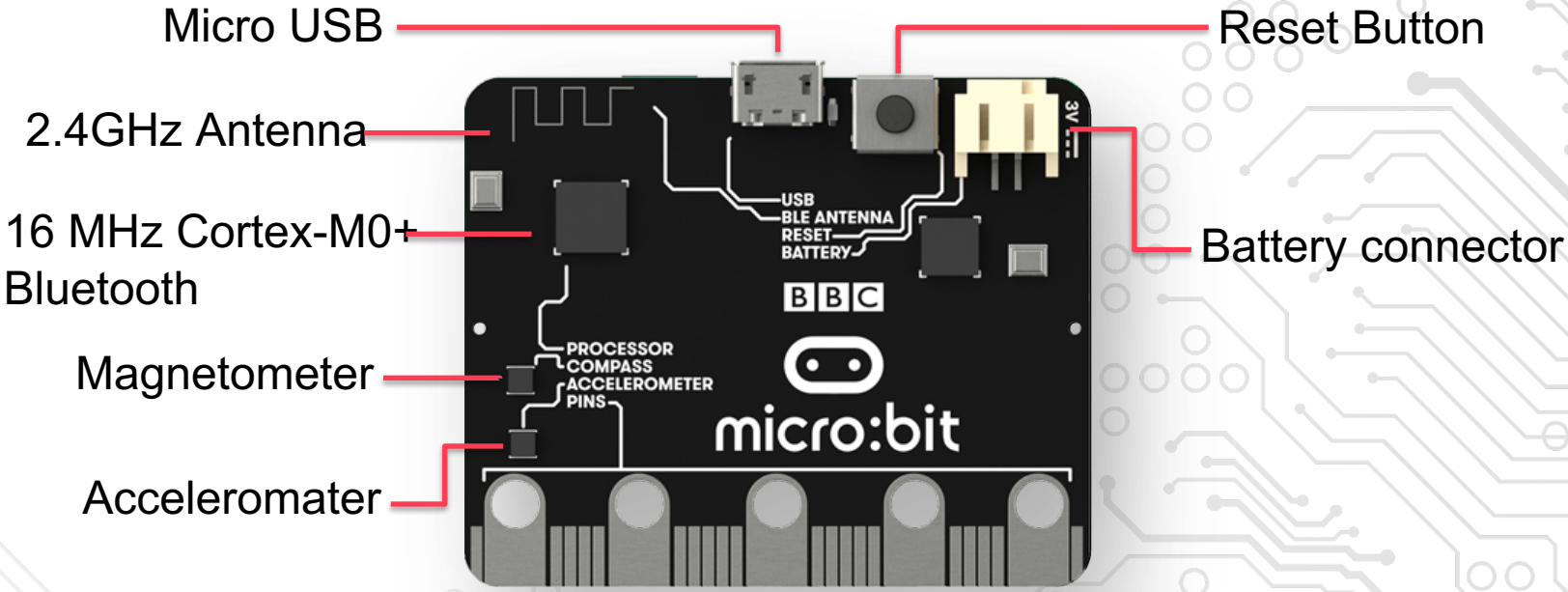
OR

https://git.io/Je3tM

BBC micro:bit



BBC micro:bit



Code Tips

- For constant values use `const()` for compiler optimisation
  - `VARIABLE = const(0x55)`
- `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
  - Shit 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

Summary

Summary

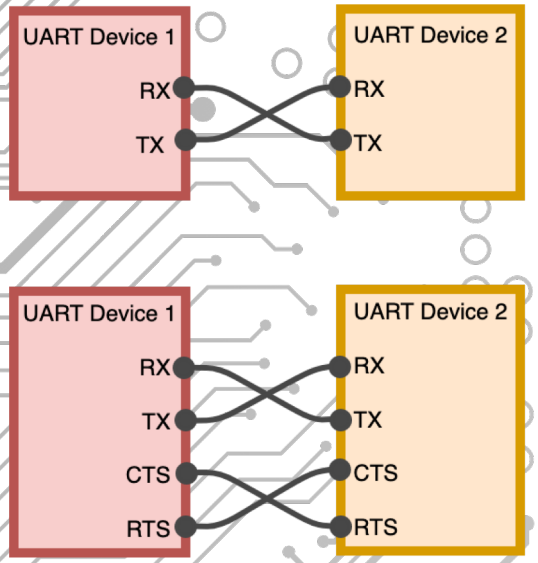
- Peripherals inside the microcontroller are responsible for controlling specific hardware features
- Hardware can be driven by setting or clearing bits in specific memory locations (registers) in your microcontroller or device
- There are multiple protocols to talk with other devices
- Datasheets documents what to send and what to expect
- MicroPython driver code is similar, but different, than other Python code
- Hardware is awesome, even more so with Python!

Summary

- Peripherals inside the microcontroller are responsible for controlling specific hardware features
- Hardware can be driven by setting or clearing bits in specific memory locations (registers) in your microcontroller or device
- There are multiple protocols to talk with other devices
- Datasheets documents what to send and what to expect
- MicroPython driver code is similar, but different, than other Python code
- **Hardware is awesome, even more so with Python!**

Other Serial Protocols (Bonus Slides)

UART Protocol



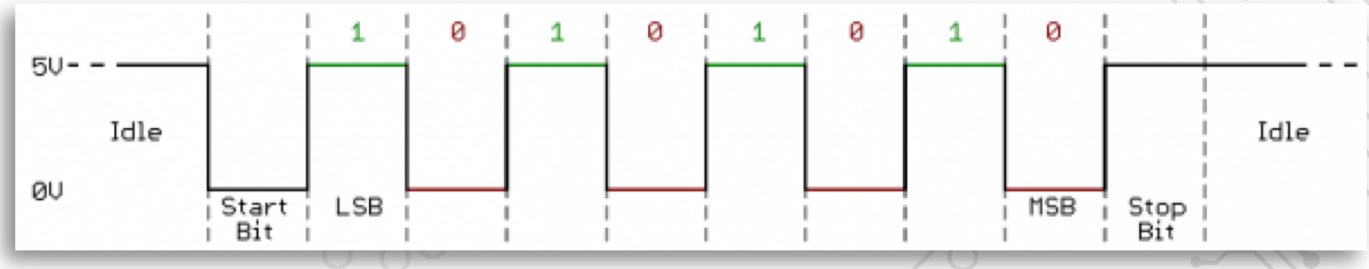
- Universal Asynchronous Receiver/Transmitter
- 2 unidirectional digital serial data lines
  - Optional 2 control signals
- Baud-rate predefined for both devices
- For new boards, good 1st driver to get working
- UART-to-USB bridges/cables are very common
- Similar standards:
  - RS232 / RS485
  - ISO7816 (Smart cards)

UART Protocol

Simple format:

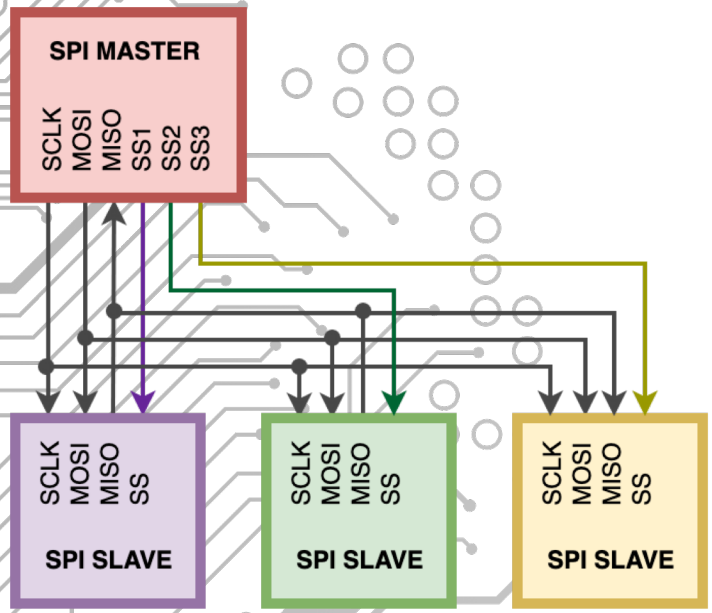
- 1 start bit
- 5-9 data (usually 8)
- 1 (or 0) parity bit
- 1 (or 2) stop bits

```
from machine import UART

uart = UART(1, baudrate=9600,
            bits=8, parity=None,
            stop=1)

uart.write(b'\xAA')
```



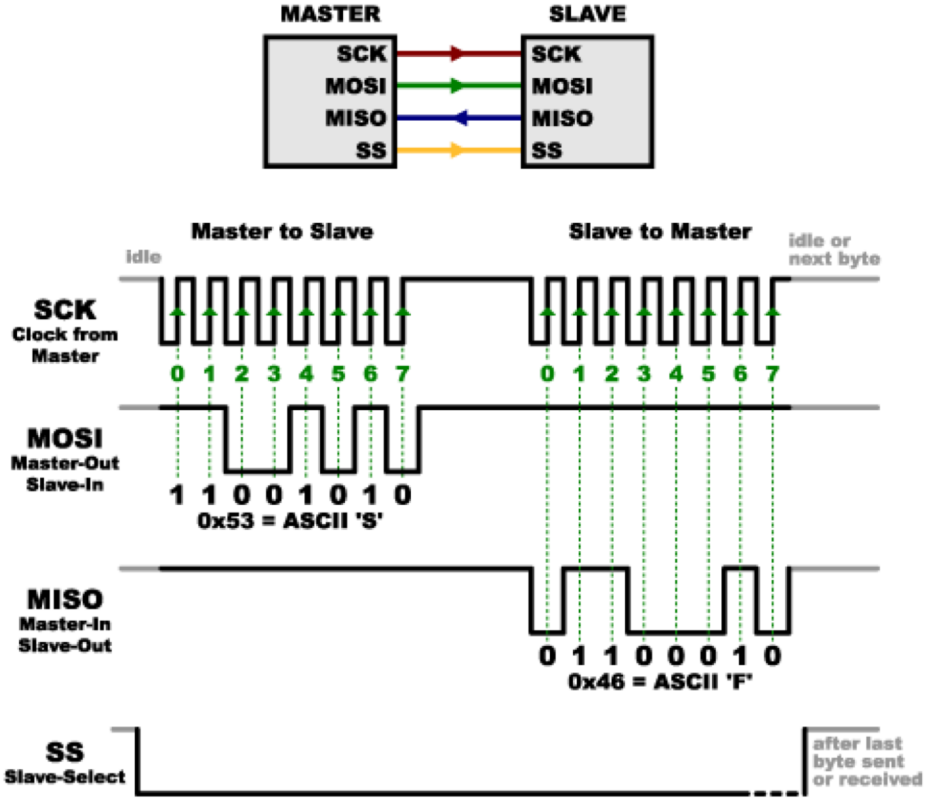
From SparkFun Serial Communication tutorial
https://learn.sparkfun.com/tutorials/serial-communication/all

SPI Protocol



- Serial Peripheral Interface
- Synchronous serial protocol
- 4 wires (+1 for each extra slave):
  - Master-Out-Slave-In shared with all
  - Master-In-Slave-Out shared with all
  - Clock shared with all devices
  - Slave Select individual signal per slave
- Clock frequencies can be in the MHz range

SPI Protocol



From SparkFun SPI tutorial:
https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

```
from machine import SPI

spi = SPI(1, SPI.MASTER,
          baudrate=600000)

data = spi.send_recv(b'S')
```