I2C with PyCom modules

1 I2C

The I2C is a serial bus designed by Philips in the early ’80s, to allow easy communication between components which reside on the same circuit board.

Philips Semiconductors migrated to NXP in 2006.

1.1 What is a bus ?

from: https://en.wikipedia.org/wiki/Bus_(computing)

In computer architecture, a bus is a communication system that transfers data between components inside a computer, or between computers.

1.2 Serial buses

from: https://en.wikipedia.org/wiki/Serial_communication

A shared channel that transmits data one bit after the other over a single wire or fiber

1. Integrated circuits are more expensive when they have more pins.
2. when speed is not important, to reduce the number of pins in a package, many ICs use a serial bus to transfer data.
3. Some examples of such low-cost serial buses include: SPI, I²C, DC-BUS, UNI/O, and 1-Wire.

1.3 I2C communication speed

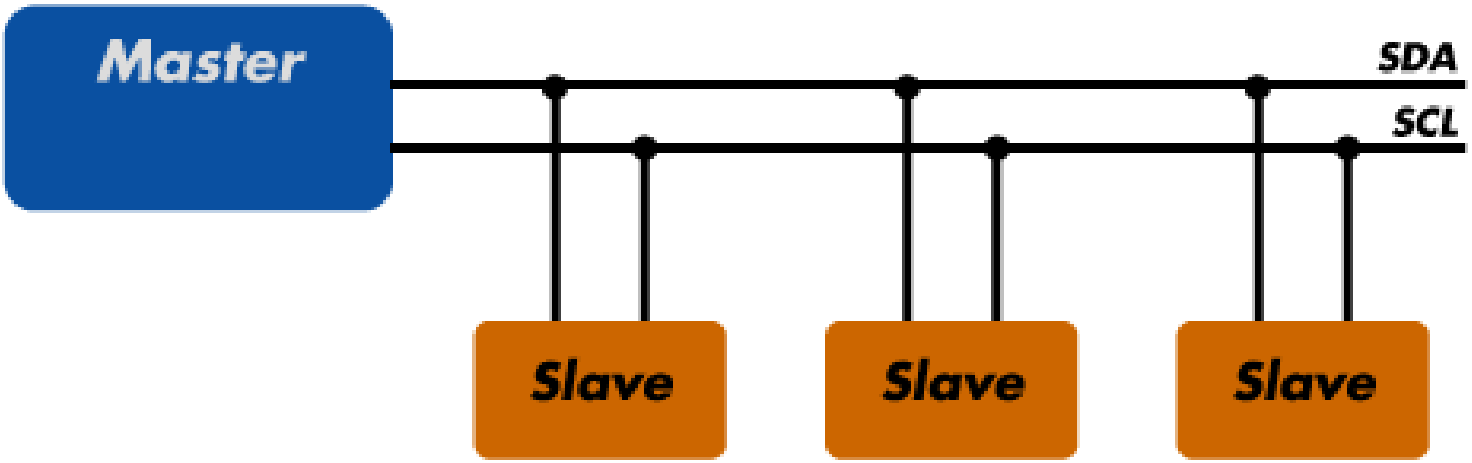
1. was defined with a maximum of 100 kbit per second
2. for some devices, there is a 400 kbit fastmode and – since 1998 – a high speed 3.4 Mbit option available.

1.4 Reference design

I²C define a bus composed of two bidirectional serial signals:

1. Serial Data Line (SDA)
2. Serial Clock Line (SCL)

Typical voltages used are +5 V or +3.3 V.



from: https://www.totalphase.com/support/articles/200349156-I2C-Background

Regardless of how many slave units are attached to the I2C bus, there are only two signals connected to all of them.

1.5 Bus characteristics

The bus has two roles for nodes:

1. Master node – node that generates the clock and initiates communication with slaves.
2. Slave node – node that receives the clock and responds when addressed by the master.

1.6 I2C protocol states

- master transmit: master node is sending data to a slave,
- master receive: master node is receiving data from a slave,
- slave transmit: slave node is sending data to the master,
- slave receive: slave node is receiving data from the master.

1.7 Message protocols

I²C defines basic types of messages, each of which begins with a START and ends with a STOP:

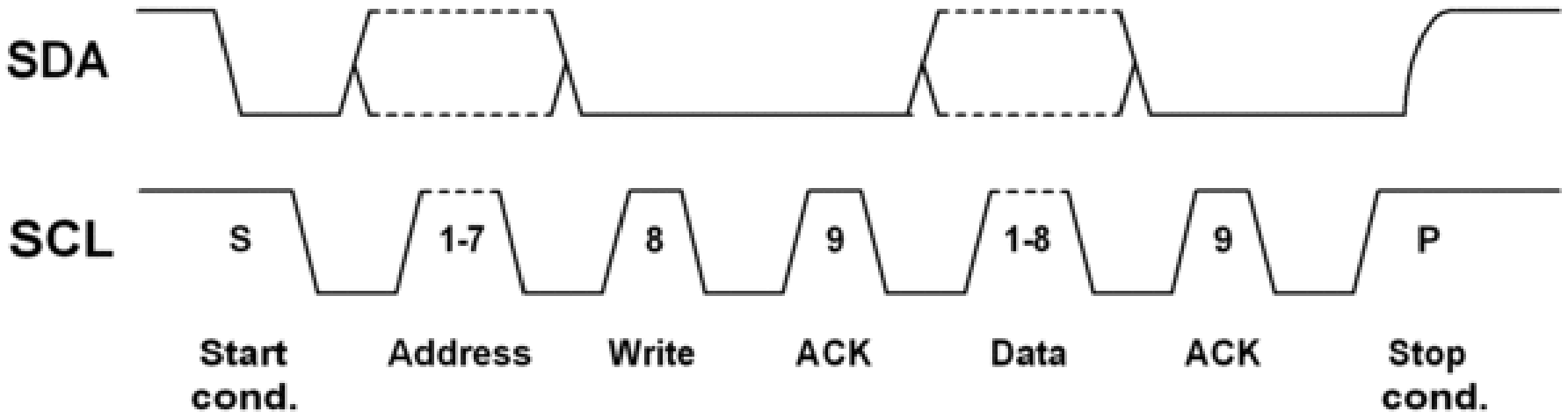
1. Single message where a master writes data to a slave.
2. Single message where a master reads data from a slave.
3. Combined messages, where a master issues at least two reads or writes to one or more slaves.

1.8 I2C addressing

https://www.i2c-bus.org/addressing/

The first byte of an I2C transfer contains:

1. the slave address
2. the data direction.



1.8.1 Address characteristics

- **The address is 7 bits long, followed by the direction bit.**
- **A seven bit wide address space theoretically allows 128 I2C addresses**
- Like all data bytes, the address is transferred with the most significant bit first.
- Some addresses are reserved for special purposes. so only 112 addresses are available with the 7 bit address scheme.
- To get rid of this a special method for using 10 bit addresses is defined.

1.8.2 Reserved Addresses

from: https://www.totalphase.com/support/articles/200349176-7-bit-8-bit-and-10-bit-I2C-Slave-Addressing

The I2C specification has reserved two sets of eight addresses, 1111XXX and 0000XXX.

These addresses are used for special purposes.

Slave Address	R/W Bit	Description
000 0000	0	General call address
000 0000	1	START byte(1)
000 0001	X	CBUS address(2)
000 0010	X	Reserved for different bus format (3)
000 0011	X	Reserved for future purposes
000 01XX	X	Hs-mode master code
111 10XX	X	10-bit slave addressing
111 11XX	X	Reserved for future purposes

(1) No device is allowed to acknowledge at the reception of the START byte.

1.8.3 Example of master transmit mode

The master is initially in master transmit mode:

1. send a start bit
2. follow the 7-bit address of the slave it wishes to communicate with (in total the address space is between 0x00 and 0x7F)
3. finally follow a single bit representing whether it wishes to write (0) to or read (1) from the slave.

1.9 How a programmer see an I2C peripheral

A programmer see an I2C peripheral as a set of registers.

For each programming languages and microcontrollers there are specific software libraries that help the programmer to communicate with devices connected via I2C.

1.10 What are registers?

In general:

- Registers consist of a small amount of fast memory storage
- some registers have specific hardware functions, and may be read-only or write-only.

- each Register has an unique address

1.11 I2C register types

Inside an I2C peripheral, normally each register occupy a multiple of bytes (1 byte = 8 bits)

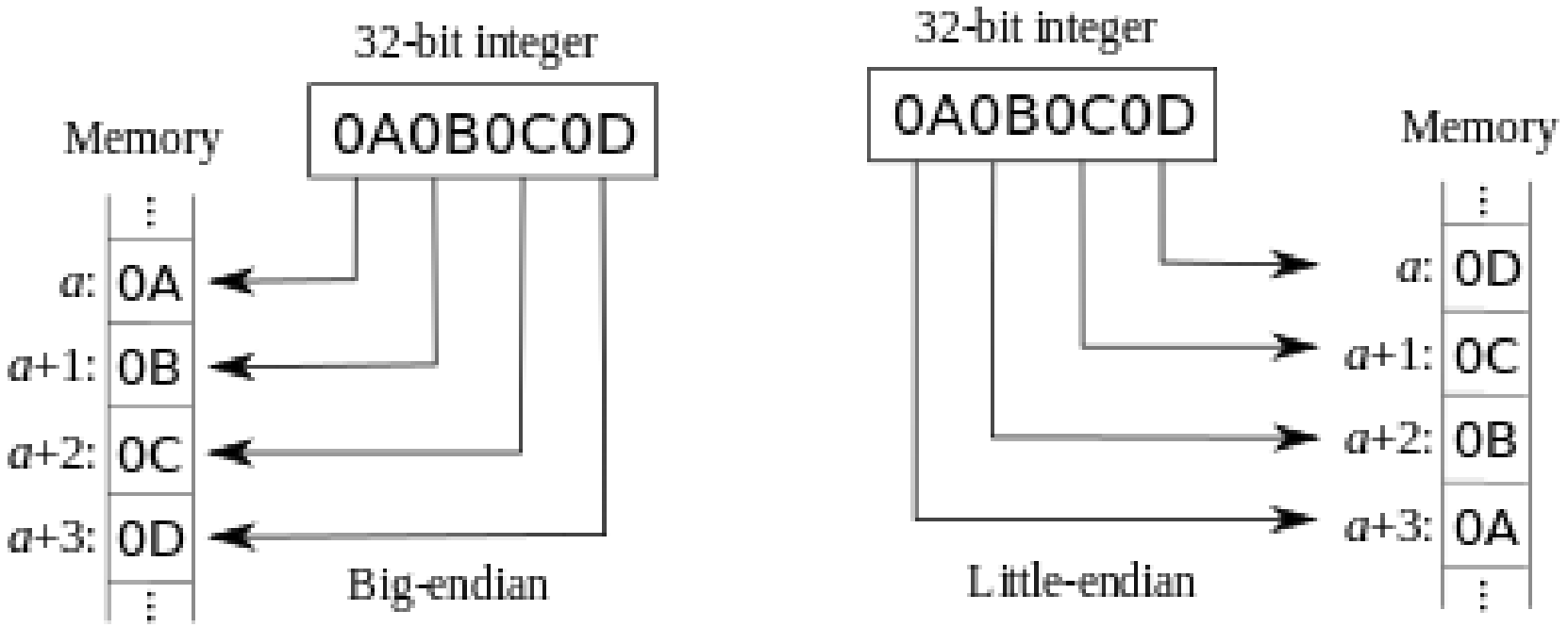
1. register of 1 byte (8 bits)
2. register of 2 bytes (16 bits)
3. register of 3 bytes (24 bits)
4. register of 4 bytes (32 bits)

And so on …

1.12 Endianness

In order to properly manage the data in the registers, is important to know Endianness:

It refers to the sequential order used to numerically interpret a range of bytes



from Wikipedia

The endianness is specific of the component.

1.13 Steps to manage an I2C peripheral

If you must connect a new I2C peripheral to a microcontroller you must:

1. download and read the datasheet of the component from the electronic producer
2. identify the 7bit I2C address of the component
3. See the maximum size of internal registers, how they are addressed and the endianness
4. Shortcut: check the internet examples of device programming and compare how they manage the peripheral with the datasheet information

1.14 PyCom I2C class

https://docs.pycom.io/pycom_esp32/library/machine.I2C.html

To connect an I2C device to a PyCom unit, we must:

- Set the unit as master
- specify the I2C peripheral address
- specify the baudrate

Example:

```
from machine import I2C
i2c = I2C(0)                                  # create on bus 0
i2c = I2C(0, I2C.MASTER)                      # create and init as a master
i2c.init(I2C.MASTER, baudrate=100000)         # init as a master
```

1.14.1 Write to register functions

i2c.writeto(addr, buf)
Write the bytes from buf to the slave specified by addr.

Return value is the number of bytes written.

Examples:

```
i2c.writeto(SHT25Address, b'\xFE')
….
elements = [0, 200, 50]
# Create bytearray from list of integers.
data = bytearray(elements)
# write 3 bytes and store it in data.
# The variable nb has the number of bytes written
nb = i2c.writeto(address, data)
```

Write to Memory registers:

Some I2C devices act as a memory device that can be written to.

i2c.writeto_mem(*addr, memaddr, buf*)

Write *buf* to the slave specified by *addr* starting from the memory address specified by *memaddr*.

The return value is the number of bytes written.

```
Examples:
I2CADDR = const(30)         # address of I2C defice
CTRL1_REG = const(0x20)     # address of control register 1

# write 2 bytes (16 bits) in register whose address
# is specified in  CTRL1_REG variable
#
reg = bytearray(2)
reg[0] = 0x55
reg[1] = 0xAA
# write in I2C device with address I2CADDR,
# in register address CTRL1_REG the values stored in reg
writeto_mem(I2CADDR, CTRL1_REG, reg)
```

1.14.2 Functions used to read register

i2c.readfrom(addr, nbytes)
Read nbytes from the slave specified by addr.
Returns a bytes object with the data read.

```
Example:
SHT25Address = const(0x40)            # address of SHT25 sensor
….
dt = i2c.readfrom(SHT25Address, 3)   # read 3 bytes
```

i2c.readfrom_into(*addr, buf*)
Read into *buf* from the slave specified by *addr*.

The number of bytes read will be the length of *buf*.

Return value is the number of bytes read.

```
Example:
reg2 = bytearray(2)
….
i2c.readfrom_into(SHT25Address, reg)
```

Function to read Memory registers

Some I2C devices act as a memory device that can be read from.

i2c.readfrom_mem(*addr, memaddr, nbytes*)
Read *nbytes* from the slave specified by *addr* starting from the memory address specified by *memaddr*.

Example:

```
I2CADDR = const(30)                 # device address
PRODUCTID_REG = const(0x0F)   # register address with product ID code
….
whoami = self.i2c.readfrom_mem(I2CADDR, PRODUCTID_REG, 1)
```

`i2c.readfrom_mem_into(`*addr*, *memaddr*, *buf*`)`
Read into *buf* from the slave specified by *addr* starting from the memory address specified by *memaddr*.
The number of bytes read is the length of *buf*.
The return value is the number of bytes read.

```
Example:

reg = bytearray(1)
…
# read CTRL1_REG and enable acceleration readings
i2c.readfrom_mem_into(I2CADDR, CTRL1_REG, reg)
# set to 1 the 3 least significant bits
reg[0] = reg[0] & 0xF8                # clear the 3 least significant bits
reg[0] = reg[0]  | 0x03               # set the 3 bits to 1
# write the new value in  CTRL1_REG
i2c.writeto_mem(I2CADDR, CTRL1_REG, self.reg)
```

1.15 Sensors on PySense

https://www.pycom.io/product/pysense/

https://docs.pycom.io/pycom_esp32/pycom_esp32/tutorial/includes/pysense-start.html

https://www.pycom.io/wp-content/uploads/2017/04/pysensePinoutCompN3.pdf

1. Accelerometer (LIS2HH12)
2. Ambient light sensor (LTR-329ALS-01)
3. Barometric pressure sensor and altimeter (MPL3115A2)
4. Temperature and humidity sensor (Si7006-A20)
5. Battery Management 800mA, Li-Ion Battery Charger (BQ24040)

https://docs.pycom.io/pycom_esp32/pycom_esp32/tutorial/includes/pysense-examples.html#pysense-examples

… Actually there is only an example for 3-Axis Accelerometer.

Other example today are still ‘Coming soon’

Component	Use	Manufacturer (*)	Technical info
LIS2HH12	Accelerometer	ST	http://www.st.com/en/mems-and-sensors/lis2hh12.html
LTR-329ALS-01	Ambient light sensor	Liteon	http://optoelectronics.liteon.com/%230%23/Led/LED-Component/Detail/518 http://www.mouser.com/ds/2/239/Lite-On_LTR-329ALS-01%20DS_ver1.1-348647.pdf
MPL3115A2	Barometric pressure sensor and altimeter	NXP Semiconductors	http://www.nxp.com/docs/en/data-sheet/MPL3115A2.pdf
Si7006-A20	Temperature and humidity sensor	Silicon Labs	https://www.silabs.com/documents/public/data-sheets/Si7006-A20.pdf
BQ24040	Battery Charger	Texas Instruments	http://www.ti.com/product/BQ24040

(*) the component can also be produced by other manufactured

1.15.1 I2C component addresses

Sensor	Use	Device Address
LIS2HH12	Accelerometer	0x1E
LTR-329ALS-01	Ambient light sensor	0x29
MPL3115A2	Barometric pressure sensor and altimeter	0x60
Si7006-A20	Temperature and humidity sensor	0x40
BQ24040	Battery Charger	

2 References

https://en.wikipedia.org/wiki/I²C

https://www.i2c-bus.org/i2c-primer/

https://www.totalphase.com/support/articles/200349156-I2C-Background

http://www.robot-electronics.co.uk/i2c-tutorial

http://www.nxp.com/docs/en/user-guide/UM10204.pdf