Imperial College London

Topic 15

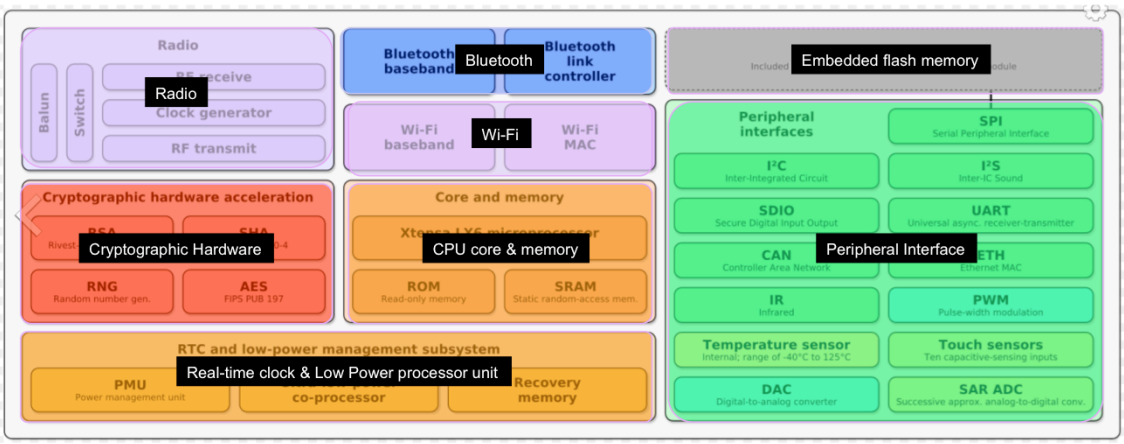
Using MicroPython on ESP32

Professor Peter YK Cheung
Dyson School of Design Engineering

URL: www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/
E-mail: p.cheung@imperial.ac.uk

In this lecture, you will learn about what is inside the ESP32 chip and how to set up your laptop and the ESP32 module so that you can program the ESP32 in MicroPython.

ESP32 IoT Microcontroller (1)



The microprocessor chip on the ESP32 module is an Espressif ESP32 microcontroller chip designed for Internet of Things (IoT) applications. This chip is called a microcontroller because it has the processor itself (CPU core), on-chip memory, and many other built-in components that allows user to almost build an entire computer system on one chip (called System-on-Chip or SoC).

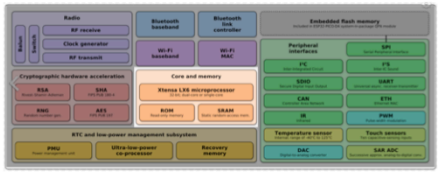
The entire chip can be divided into a number of sub-modules. They are:

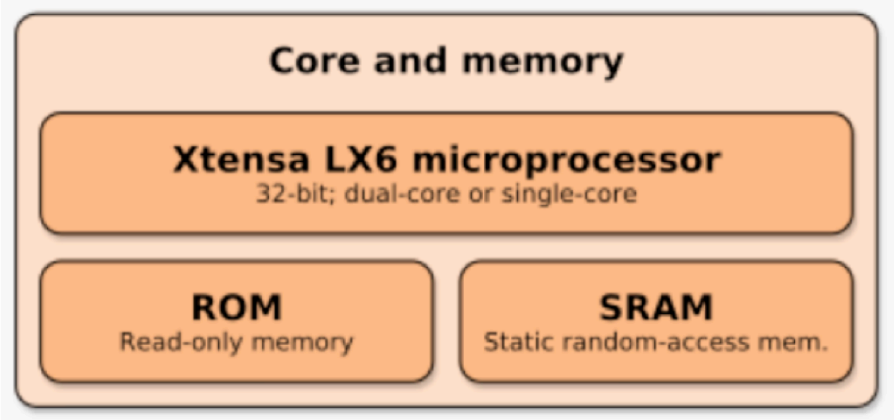
- Core and Memory (brain of the chip)
- Cryptographic Hardware Engines (for encryption & decryption)
- Wireless communications (wifi and bluetooth)
- Power management subsystem (for low power IoT applications)
- Peripheral interfaces (for connection with other devices)
- Extra flash memory (SPI RAM)

The datasheet of the ESP32 can be found here:

https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

ESP32 CPU Core & Memory





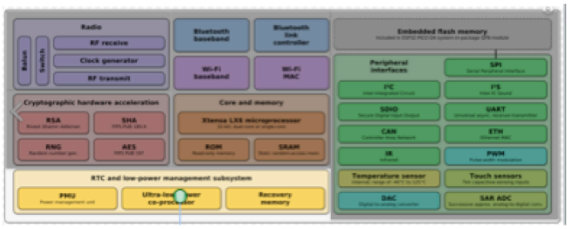
The CPU core and memory contains a 32-bit Xtensa LX6 microprocessor, which is a silicon IP (intellectual property) core designed and owned by Tensilica, a company in California.

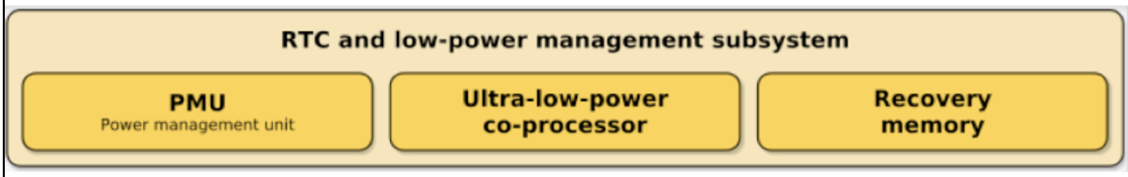
This block inside the ESP32 also contains:

- 448kB of lash memory (ROM) that is non-volatile (i.e. its content is stored even if power is removed)
- 520kB of SRAM (static RAM) which is used to store all variables, heap and stack

The microprocessor is quite power, capable of executing 240 million instructions (single-core) and running at 240MHz clock.

ESP32 RTC and Power Management Sub-system



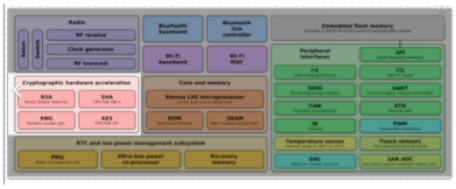


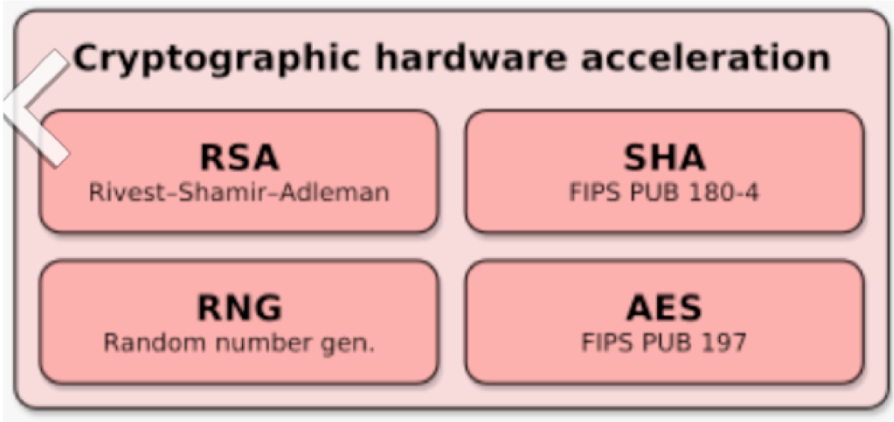
One requirement of an IoT device is ultra low-power. The ESP32 contains an entire low-power management subsystem that handles applications where speed is NOT the most important feature. Instead it allows the device to go to sleep and wakeup only when necessary to do some basic tasks such as take a reading or two, and send them wirelessly via WiFi or Bluetooth to a base-station. Furthermore, this module also contains a Real-time Clock (RTC) that allows scheduling of events, and wake up the Ultra Low-power co-processor (ULP) to do its thing!

Whenever the processor goes to sleep, it stores away what its doing (called its "state" or "context") in a block of recovery memory, so that when it wakes up, the ULP can be restored to its original state.

We will not be using this subsystem in Electronics 1.

ESP32 Crypto Hardware





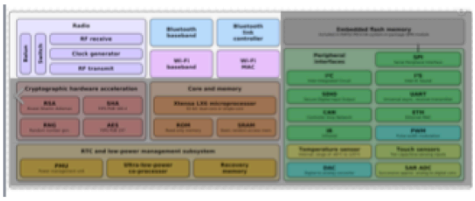
PYKC 9 June 2020 DE 1.3 - Electronics 1 Topic 15 Slide 5

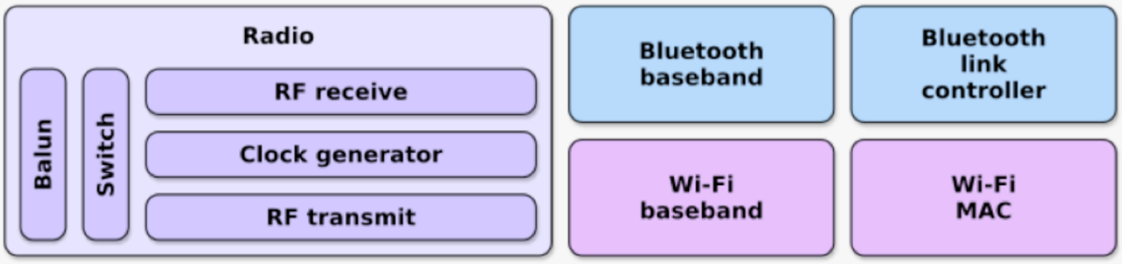
Since the ESP32 is designed for IoT applications, one requirement for such applications is privacy and security. Therefore information sent or received are often encrypted. Doing encryption and decryption using the microprocessor is not only wasteful (because it won't be able to other useful things while performing encryption/decryption), it is also slow when performing such tasks using software. Worse, it is power hungry.

Therefore the ESP32 integrates three different encryption/decryption engines on-chip, which implement the most common encryption/decryption algorithms. These are: Rivest-Shamir-Adleman (RSA), Secure Hash Algorithm (SHA) and Advanced Encryption Standard (AES). Don't worry exact what these are – you will however come across them in many applications later during your degree program.

Finally, this subsystem also contains a hardware random number generator engine for any applications that requires high quality random numbers.

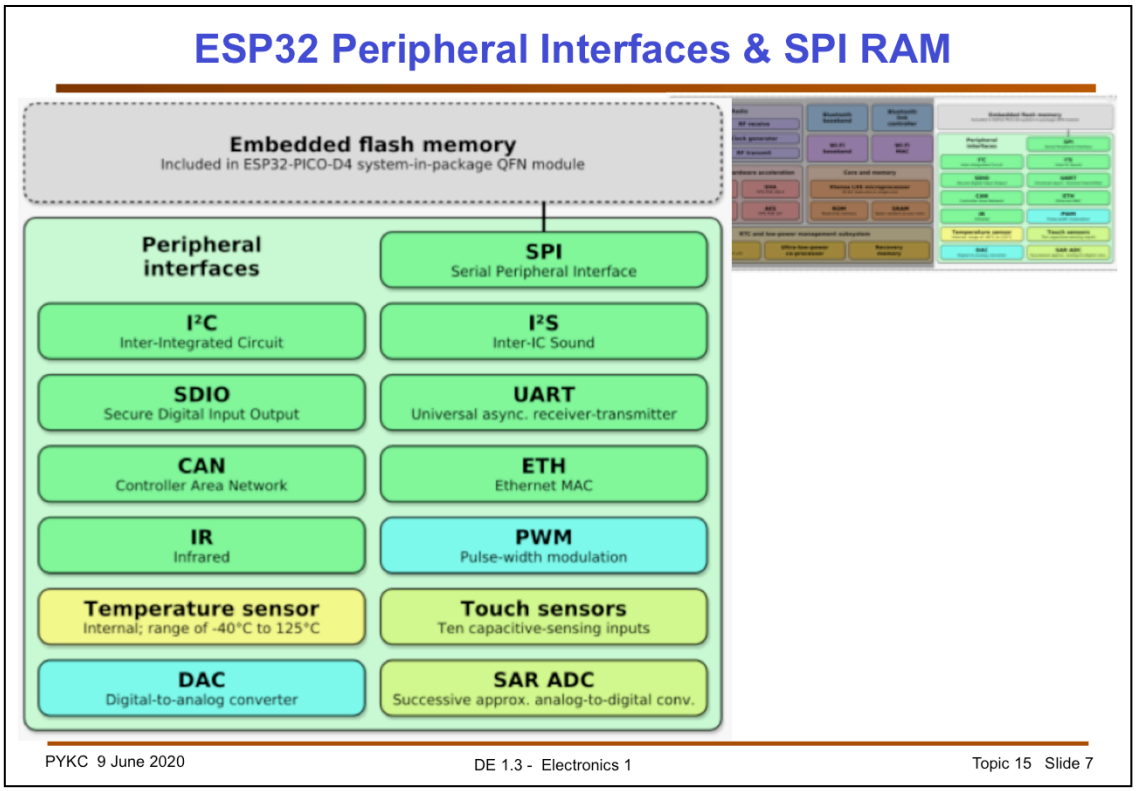
ESP32 wireless links





Being an IoT specific device, the ESP32 is one of the chip on the market to have integrated with the microprocessor both WiFi and Bluetooth communications. Both these communication standards require both digital and analogue hardware (the radio). Again, we do not want you to worry about these at this stage. Nevertheless you will find that this sub-system is most useful for many stand-alone products that you may design in the future.

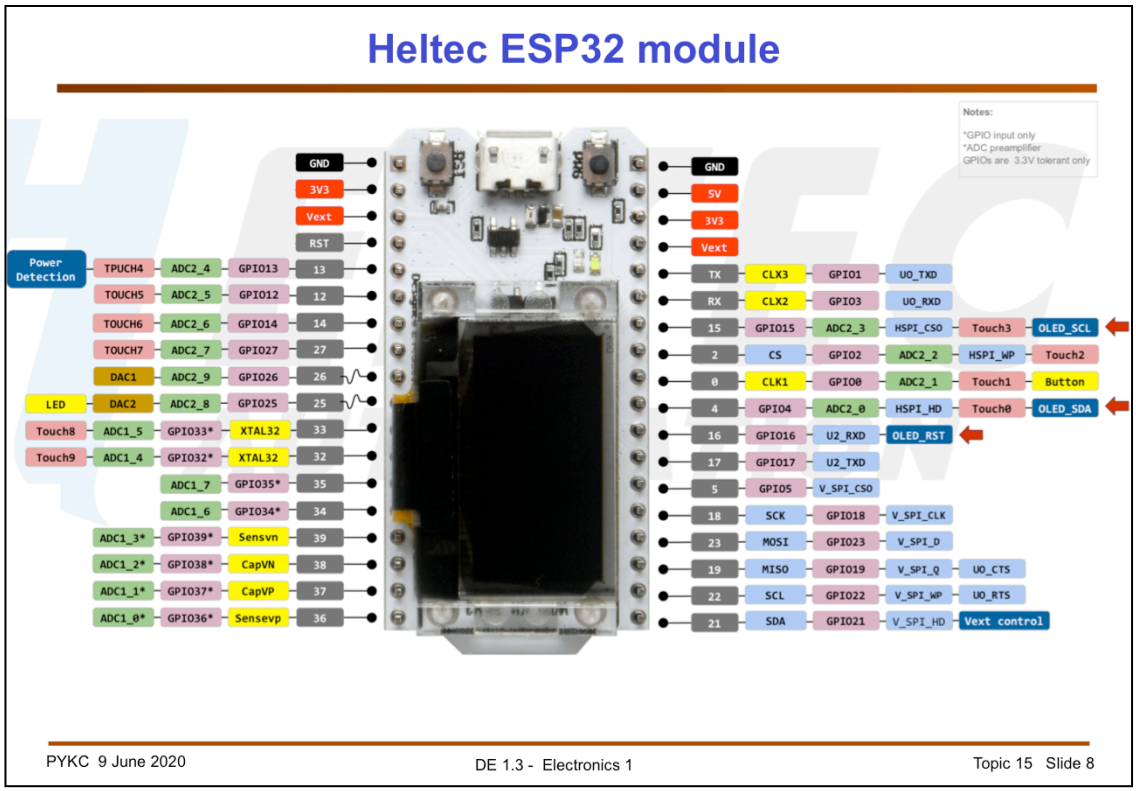
We will not be using this subsystem in Electronics 1.



The final subsystem to consider is large. It contains many separate IP blocks that allows the ESP32 to interface to the outside world directly. Included here are:

- **I2C interface** – this is what we use to drive the OLED display on the Heltec module
- **SPI interface** – this is a fast interface and it is used to link with the 4MB SPI RAM (extract storage for our programs)
- I2S interface – This is a dedicated interface for audio signals
- **UART** – You learned about this back in Lab 1 and in Digital Basics lecture
- SDIO, CAN – both are other digital interfaces for various applications
- IR – infra-red interface as used by remote control (e.g. on your TV)
- **PWM** – this generates PWM signals for driving motors and LED lights
- Sensor interfaces – Temperature and touch sensor interfaces
- EH – Ethernet interface
- **DAC** – 8-bit digital converter (used for SIG_GEN for analogue outputs)
- **ADC** – Analogue to Digital converter

Those shown in BLUE above are features that you have use or will be using in Electronics 1.



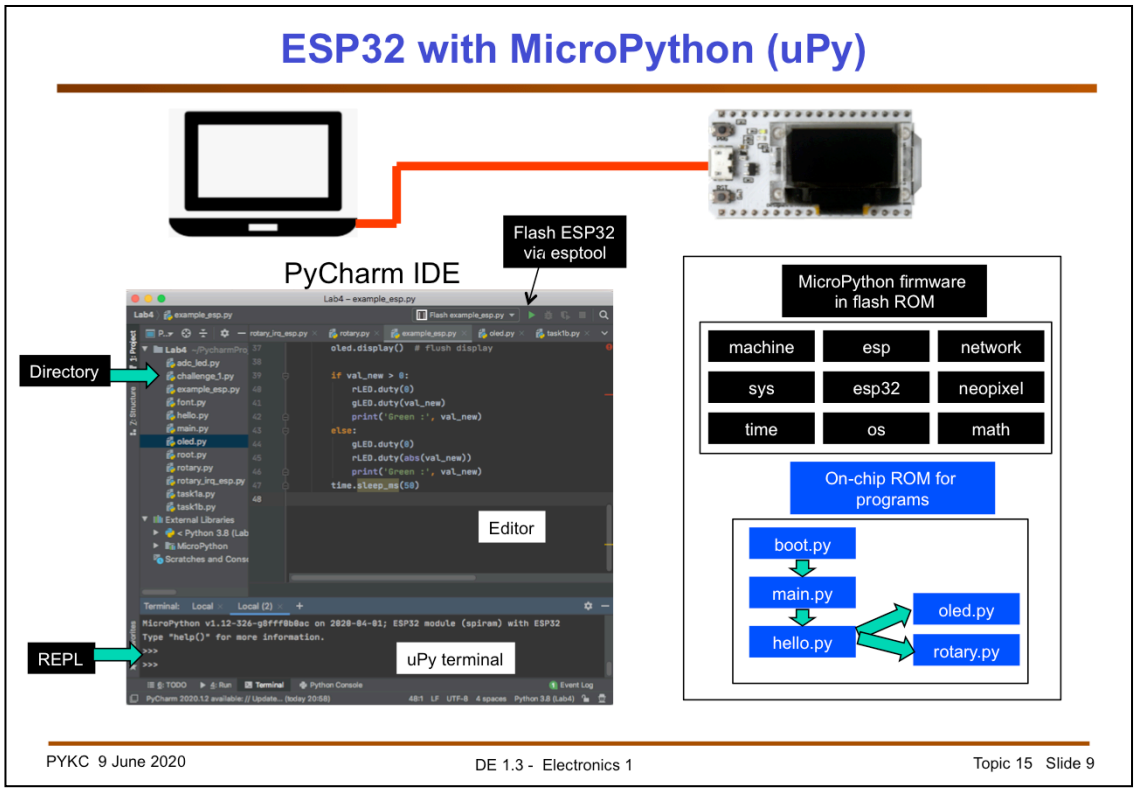
The ESP32 module you use is designed and manufactured by Heltec. In addition to the ESP32 chip, the module also include a 128 x 64 0.96” OLED. It is connected to the ESP32 using the I2C interface. (We will discuss this interface during the lecture on “Link” later.)

Here is a pinout diagram for the Heltec module. You can find a high quality PDF file of this pinout diagram on:

https://resource.heltec.cn/download/WiFi_Kit_32/WIFI_Kit_32_pinoutDiagram_V2.pdf

Note that:

- Almost all pins are multipurpose. User can program the pins for different use. For example pin 26 can be used for: digital input, digital output, analogue input and analogue output.
- The colour code shows a type of usage for a pin. For example, GREEN is for digital input (i.e. to A-D converter), PINK is for General Purpose IO (GPIO), which is digital input or output, BROWN is for analogue output (i.e. via a D-A converter).
- Three of the pins are already connected to OLED display on the module using the I2C interface.
- There are also dedicated power pins.



The ESP32 will be programmed using MicroPython. The slide provides an overview of the programming environment that you will be using.

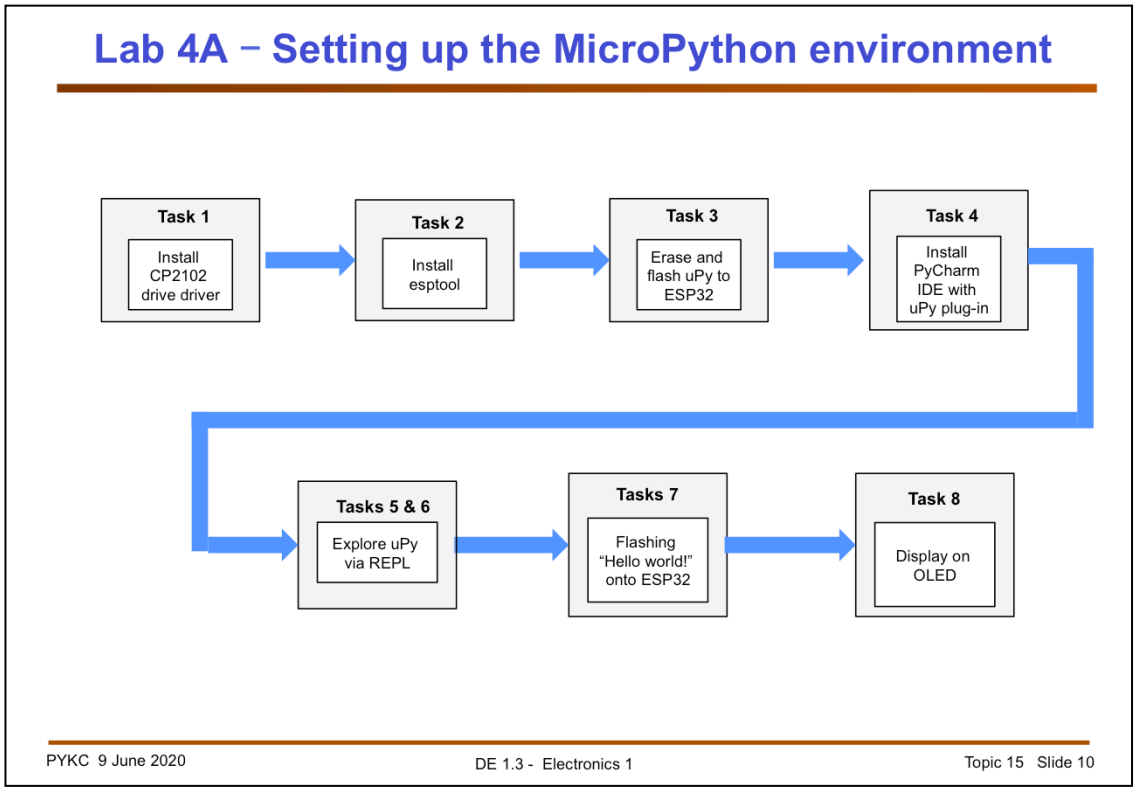
1. You must first load the MicroPython code onto the ESP internal flash ROM (shown in black on the right).
2. Later you will load into the internal flash memory your own program code (shown in blue).
3. To control the ESP32 with uPy, you can type directly into the REPL >>> a python code (such as: println("Hello world!"), one line at a time.
4. Alternatively you can create a uPy script and load this into ESP32 flash RAM.
5. To create the uPy script, you will use the PyCharm IDE on the laptop. This environment also provide an editor, ability navigate the project directory, communicate with uPy REPL directly via a terminal window, and even flash your program code onto the ESP32. All these can be done within the PyCharm IDE.

When you first power up the ESP32, uPy will execute the boot.py file. The boot.py file will run the main.py file. The main.py will have one single uPy line:

execfile('user_program.py')

Your program will be stored in the user program file: 'user_program.py'). (Change the name of the file to suit.)

Also stored in the on-chip flash RAM (ROM) are other modules that your program may use. For example, you will be using oled.py, which is the driver for the OLED display.

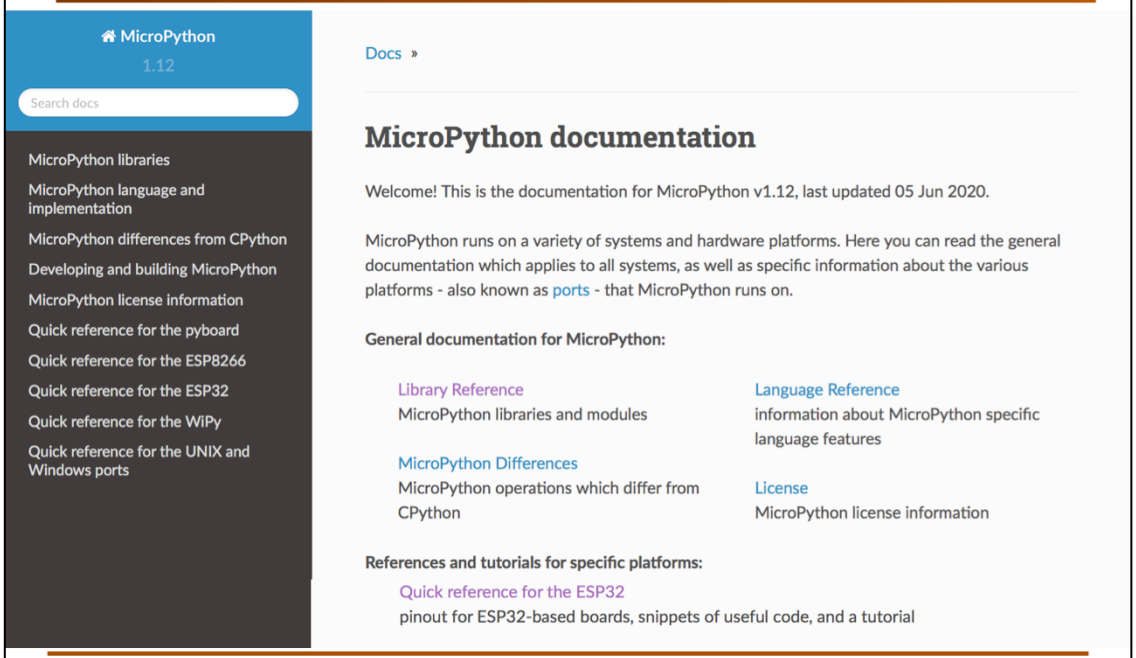


Before you can use the ESP32 for Lab 4, you need to first set up your environment for MicroPython. This involves quite a number of steps.

As a result, I have divided Lab 4 into Part A and Part B. Part A does not teaching you much except that you will gain some experience in setting up a software environment that is fairly complex, and learn about how to flash programs onto the ESP32.

The IDE you will be using is called PyCharm. This is similar to VSC, but it is designed purposefully for Python programming. The reason I chose PyCharm is that it has integrated into this through a MicroPython plug-in the necessary tools to run programs on the ESP32 and to flash new program scripts to its memory from within the IDE.

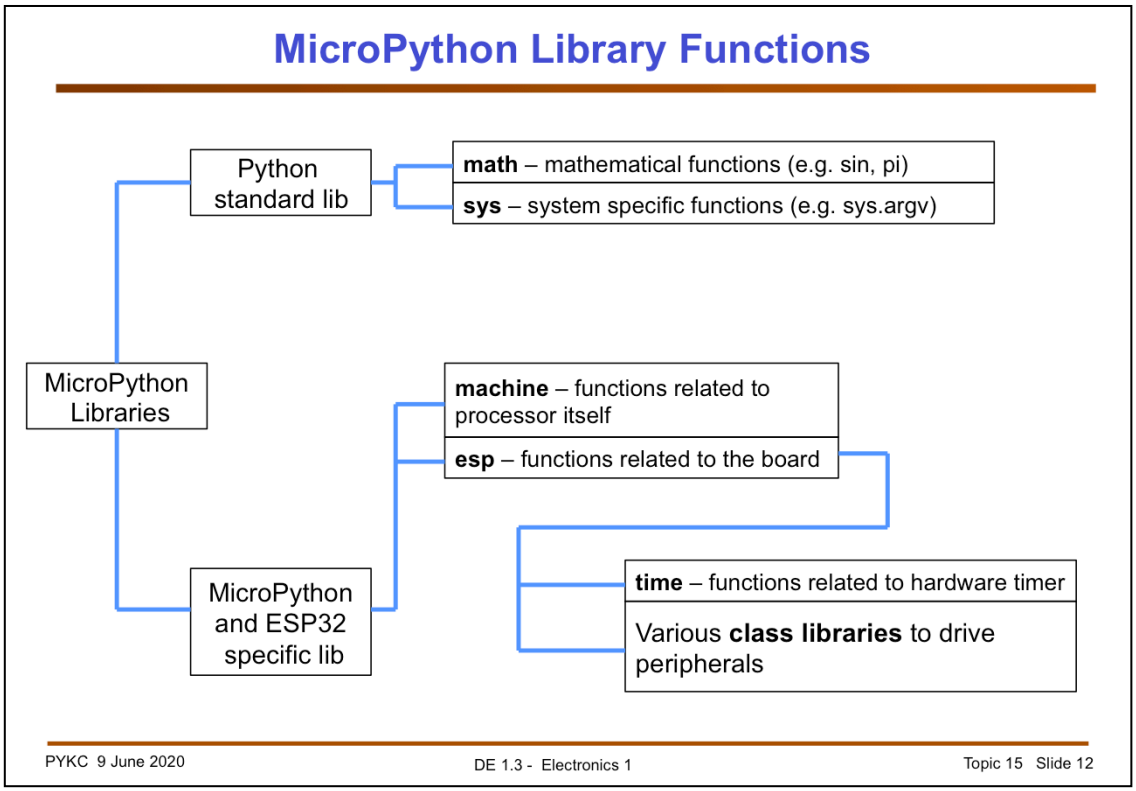
MicroPython Documentation



MicroPython is quite a large system and there are too much to learn. It is also good if you learn to read instructions from websites, rather than just been spoon fed by me. So, go to:

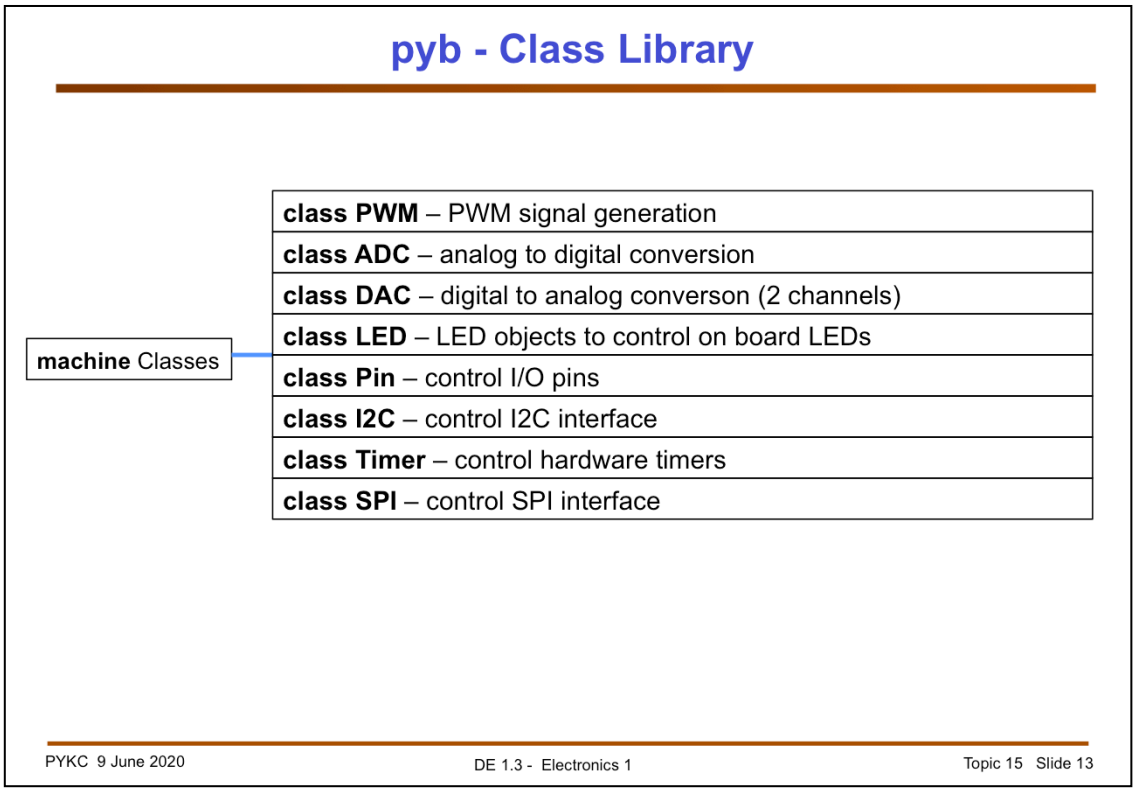
https://docs.micropython.org/en/latest/

You will see this page. There are lots of useful documents here. Explore!



MicroPython provides many library functions in order to make programming this board much easier. There are basically two main categories of library functions:

1. A subset of the standard Python library you can find everywhere (but implemented for MicroPython). Of these, the two that you would need are: math and sys.
2. There is a bunch of library functions that are written for the ESP32 specifically. The three that you need are: machine, esp and esp32. The machine library provides top level control of the board, and esp & esp32 provide detail control. *machine* is the most important library for you to know. Almost all functions we use so far are imported from the machine library.
3. There is another important module: time. This provides delay in millisecond or microseconds (e.g. time.sleep_ms()).
4. Then there is a large collection of peripheral specific libraries (they are written as object-oriented class libraries). You will be using many functions from these.



The machine class libraries includes those shown here. In Lab 4, you used the following library classes: PWM, ADC, Pin (everything), Timer (these are functions to control internal timer circuits), I2C and UART. I have provided examples in Lab 4B for you.

You don't need to know all these libraries. You should learn using this approach:

1. Learn from the example code provided.
2. If these do not do what you need, be clear what function you need to perform.
3. Look for a function in the library that is likely to give you what you needed.
4. Failing that, ask me or your friends and classmates, or your team project supervisor.