* ***App\_main***- is the first function called by FreeRTOS application startup flow. we have to add our code in it. Allows the main function to return at any given point.
* ***freertos/FreeRTOS.h*** : Inclusion of this sets configuration required to run freeRTOS on ESP32.
* ***freertos/task.h***: The tasks as you can guess provide the multitasking functionality, which we will explore in the blinky with hello world example in some time.

**## *what is RTOS***= IT is a type pf OS that allows user to perform multiple tasks at the same time.

it has a schedular which divides the processing time for different tasks based on priority, FreeRTOS is a just a type of RTOS.

* ***sdkconfig.h:*** files get saved in sdkconfig root directory after updating using menuconfig.
* ***-***
* ***struct esp\_chip\_info\_t*** = The structure represents information about the chip. esp\_chip\_info() function fills esp\_chip\_info\_t structure with information about the chip. This includes the chip revision, number of CPU cores, and a bit mask of features enabled in the chip.

Parameters

out\_info – [out] structure to be filled

* Public Members

[esp\_chip\_model\_t](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/misc_system_api.html" \l "_CPPv416esp_chip_model_t) model[](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/misc_system_api.html" \l "_CPPv4N15esp_chip_info_t5modelE)

chip model, one of esp\_chip\_model\_t

uint32\_t features[](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/misc_system_api.html" \l "_CPPv4N15esp_chip_info_t8featuresE)

bit mask of CHIP\_FEATURE\_x feature flags

uint16\_t revision[](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/misc_system_api.html" \l "_CPPv4N15esp_chip_info_t8revisionE)

chip revision number (in format MXX; where M - wafer major version, XX - wafer minor version)

uint8\_t cores[](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/misc_system_api.html" \l "_CPPv4N15esp_chip_info_t5coresE)

number of CPU cores

* ***esp\_get\_minimum\_free\_heap\_size()*** returns the minimum size of free heap memory that has ever been available (i.e., the smallest size of free heap memory in the applications lifetime).
* **esp\_restart() function** :To perform software reset of the chip, the esp\_restart() function is provided. When the function is called, execution of the program stops, both CPUs are reset, and the application is loaded by the bootloader and starts execution again.
* ***esp\_err\_t esp\_flash\_get\_size***: Detect flash size based on flash ID. Returns ESP\_OK on success or a flash error code if operation is failed.

***##Flash memory***: also known as flash storage, is a type of nonvolatile memory that erases data in units called blocks and rewrites data at the byte level. The data stored in flash memory remains there even when ESP32 resets or power is removed.

* ***Uint32\_t*** : unsigned integer
* ***VtaskDelay***: Delay a task for a given number of ticks. The actual time that the task remains blocked depends on the tick rate. The constant portTICK\_PERIOD\_MS can be used to calculate real time from the tick rate - with the resolution of one tick period.

portTICK\_PERIOD\_MS = 10

1000 / portTICK\_PERIOD\_MS = 100 ticks

100 \* 10ms = 1 sec

* ***Fflush(stdout):*** The C library function int fflush(FILE \*stream) flushes the output buffer of a stream .You would use fflush(stdout) to ensure that whatever you just wrote to a file/the console is indeed written out on disk/the console. The reason is that actually writing, whether to disk, to the terminal, or pretty much anywhere else, is pretty slow. Further, writing 1 byte takes roughly the same time as writing, say, a few hundred bytes[1]. Because of this, data you write to a stream is actually stored in a buffer which is flushed when it is full or when you call fflush. Calling fflush means you are accepting that your function call will take a bit of time but that you are 100% sure that you want this out right away.

SPI stands for [Serial Peripheral Interface](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus)

he **Serial Peripheral Interface** (**SPI**) is a [synchronous](https://en.wikipedia.org/wiki/Synchronous_circuit) [serial communication](https://en.wikipedia.org/wiki/Serial_communication) interface specification used for short-distance communication, primarily in [embedded systems](https://en.wikipedia.org/wiki/Embedded_systems).