Internet of Things
class 11

Sensors, Multi-processing, Etc..

I2C Sensor Reliability

- BME280 I2C interface is unstable..
  - SW reset might not be enough for HW rest
  - Fixup code: by Using GPIO as Power-source
    - BME280 requires a little current to be supplied by GPIO

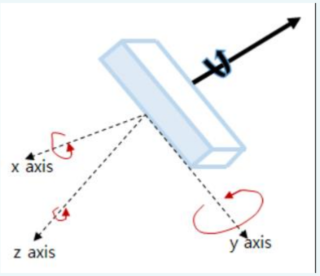
```
  . . .
void setup() {
  . . .
  pinMode(BME_Power, OUTPUT);     // use GPIO as Power source
  while (!bme.begin(0x76)) {
    Serial.println("BME280 will be reset by SW..");
    digitalWrite(BME_Power, LOW);
    delay(50);
    digitalWrite(BME_Power, HIGH);
    delay(100);
  }
  bmeID = bme.sensorID();
  . . .

void loop() {
  . . .
```

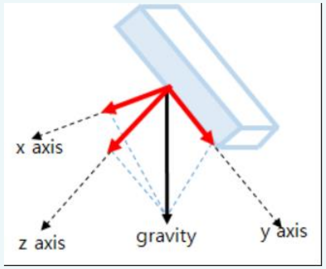
```
. . .
if (bme.sensorID() != bmeID) {
        while (!bme.begin(0x76)) {
          Serial.println("BME280 will be reset by SW..");
          digitalWrite(BME_Power, LOW);
          delay(50);
          digitalWrite(BME_Power, HIGH);
          delay(100);
        }
      }
. . .
```

Mpu9250- Motion Tracking Sensor

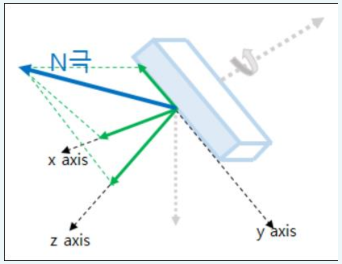
- Combines two chips
  - MPU-6500 : a 3-axis gyroscope, a 3-axis accelerometer, and an onboard Digital Motion Processor™ (DMP™) capable of processing complex MotionFusion algorithms
  - AK8963 : a 3-axis magnetometer (digital compass)



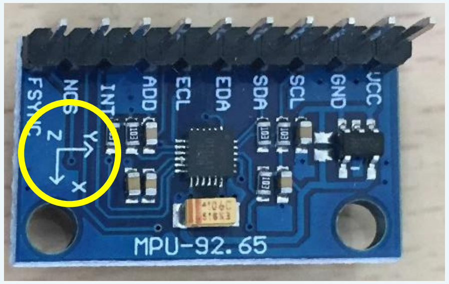
gyroscope



accelerometer

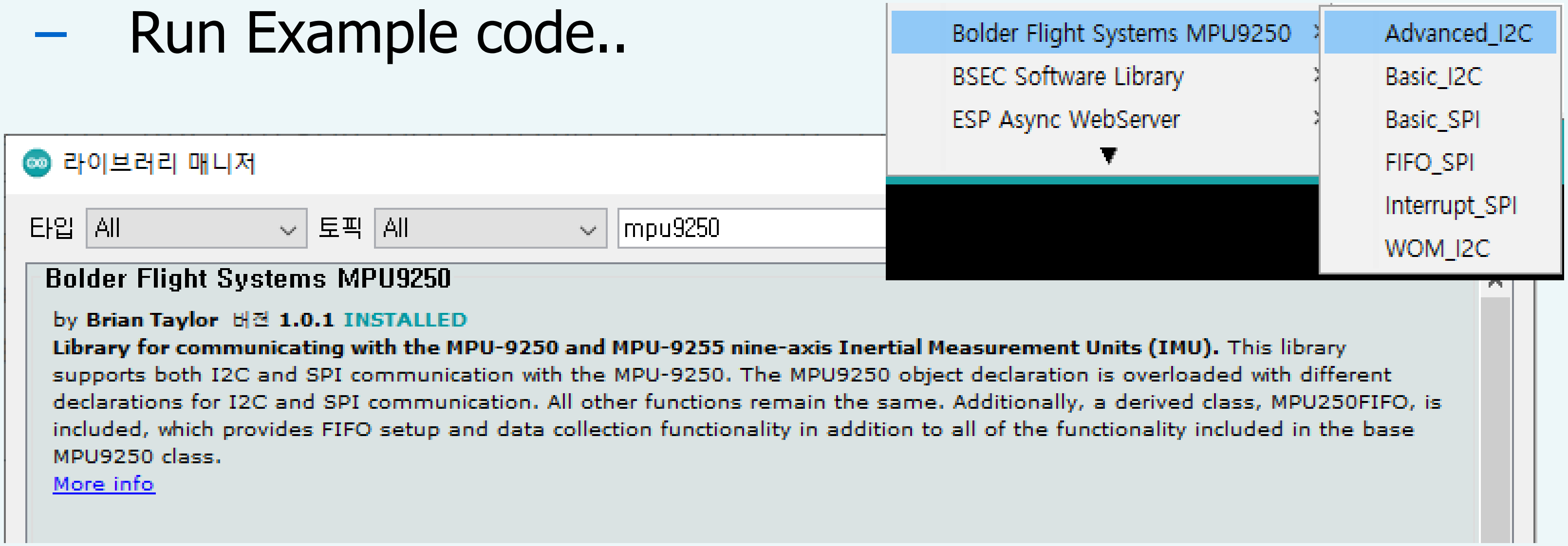


digital compass



Mpu9250- Motion Tracking Sensor

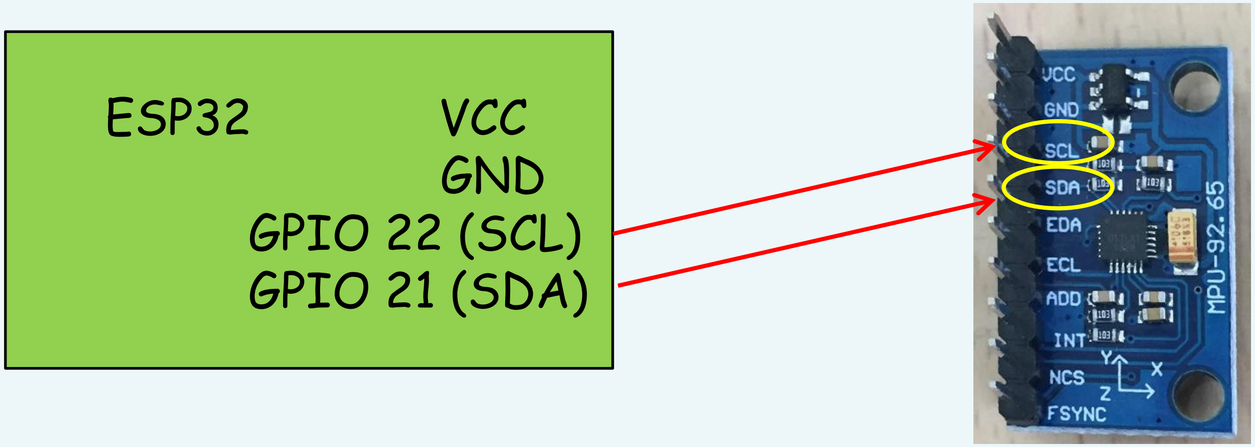
- Install Library: (anything you want)
  - Run Example code..



mpu9250

Bolder Flight Systems MPU9250

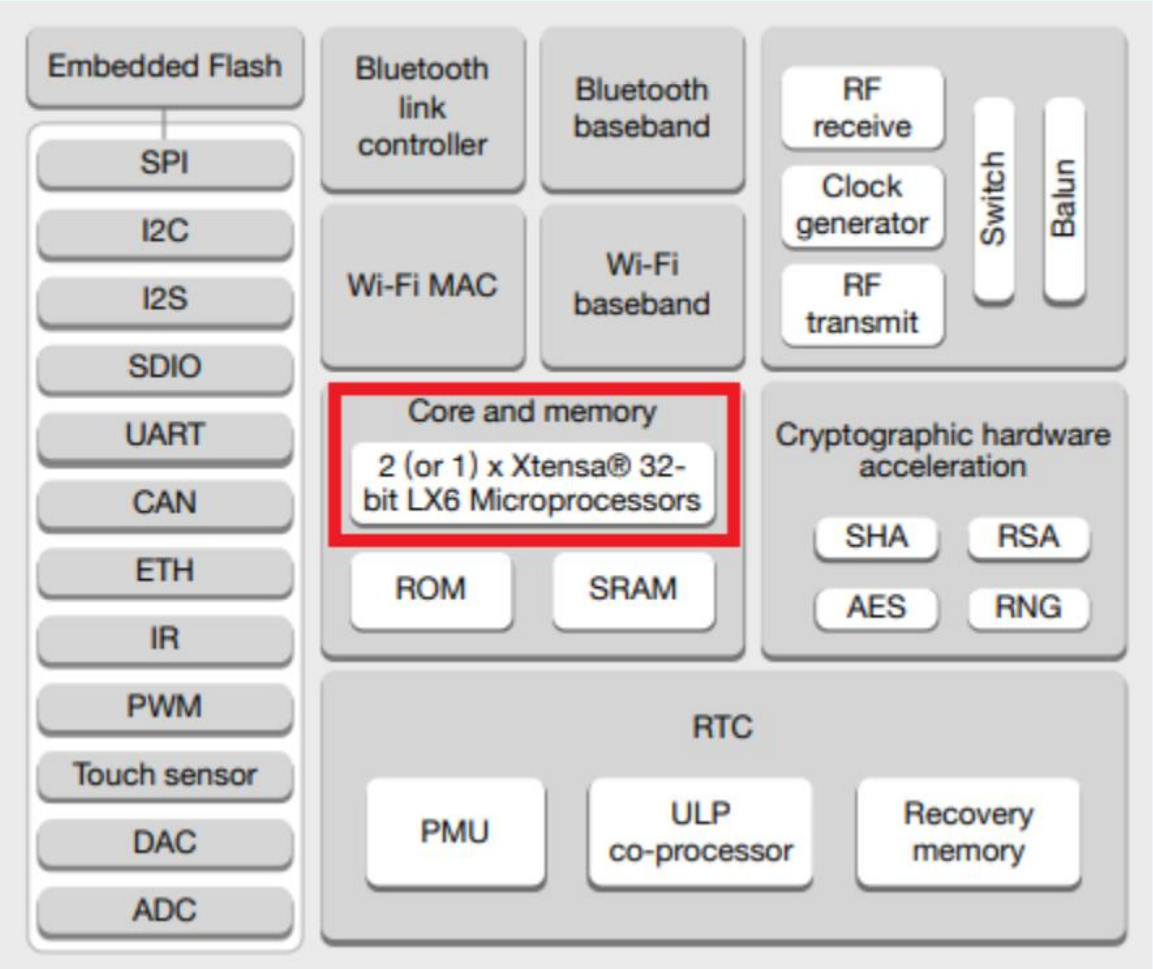
by **Brian Taylor** 버전 **1.0.1** **INSTALLED**
Library for communicating with the MPU-9250 and MPU-9255 nine-axis Inertial Measurement Units (IMU). This library supports both I2C and SPI communication with the MPU-9250. The MPU9250 object declaration is overloaded with different declarations for I2C and SPI communication. All other functions remain the same. Additionally, a derived class, MPU250FIFO, is included, which provides FIFO setup and data collection functionality in addition to all of the functionality included in the base MPU9250 class.
More info



ESP32 Dual Core

- Dual Core: 2 Xtensa 32-bit LX6 microprocessors
  - Core 0
  - Core 1

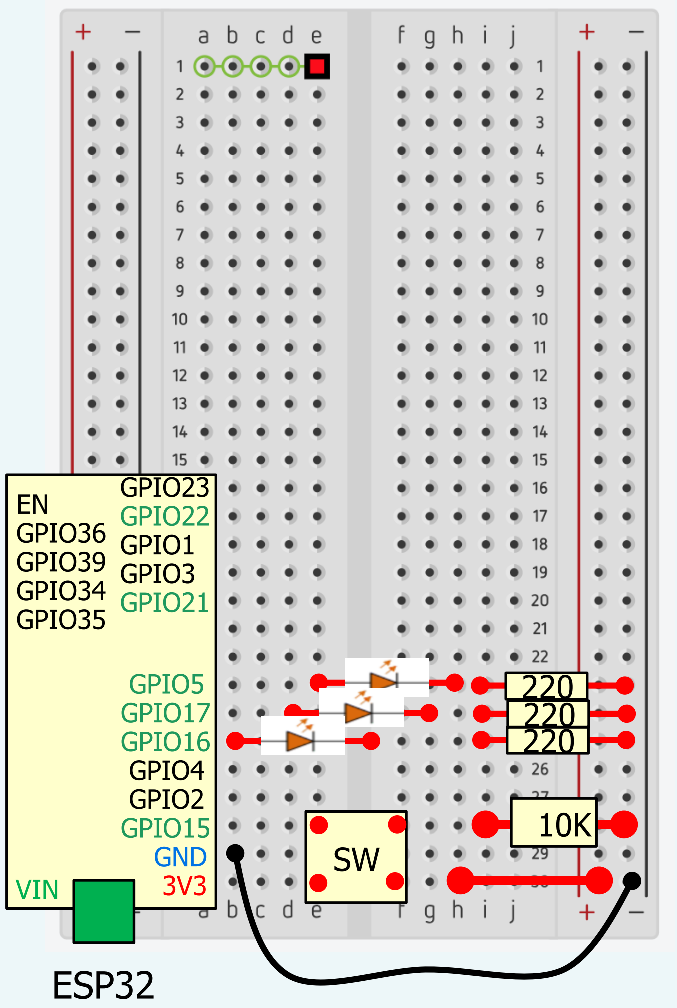
```
xTaskCreatePinnedToCore(
   Task1code, // Function task
   "Task1",     // Task name
   10000,  // Stack size in words
   NULL,        // Task input
   0,             // Priority
   &Task1,    // Task handle
   0             // Core number
);
```

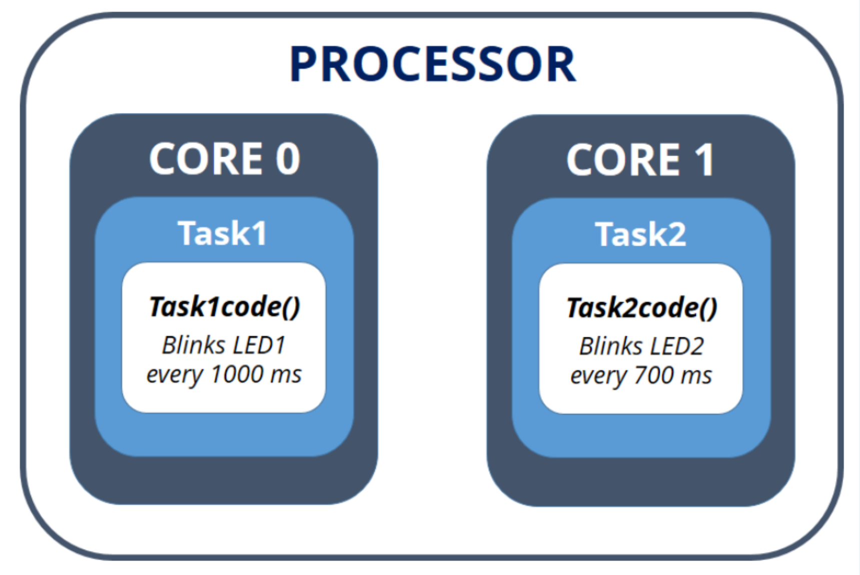


ESP32 Dual Core

- Dual Core Example

LED1: GPIO16,
LED2: GPIO17





ESP32 Dual Core

```
TaskHandle_t Task1;
TaskHandle_t Task2;
// LED pins
const int led1 = 16;
const int led2 = 17;

void setup() {
    Serial.begin(115200);
    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);

   //create a task that will be executed in the Task1code() function,
   //with priority 1 and executed on core 0
   xTaskCreatePinnedToCore(
                  Task1code, /* Task function. */
                  "Task1", /* name of task. */
                  10000, /* Stack size of task */
                  NULL, /* parameter of the task */
                  1, /* priority of the task */
                  &Task1, /* handle to keep track of task */
                  0); /* pin task to core0*/

    delay(500);
```

ESP32 Dual Core

```
    //create a task that will be executed in the Task2code() function,
    // with priority 1 and executed on core 1
    xTaskCreatePinnedToCore(
                    Task2code, /* Task function. */
                    "Task2", /* name of task. */
                    10000, /* Stack size of task */
                    NULL, /* parameter of the task */
                    1, /* priority of the task */
                    &Task2, /* handle to keep track of task */
                    1); /* pin task to core 1 */

    delay(500);
}

//Task1code: blinks an LED every 1000 ms
void Task1code(void * pvParameters) {
                    Serial.print("Task1 running on core ");
                    Serial.println(xPortGetCoreID());
                    for(;;) {
                                        digitalWrite(led1, HIGH);
                                        delay(1000);
                                        digitalWrite(led1, LOW);
                                        delay(1000);
                    }
}
```

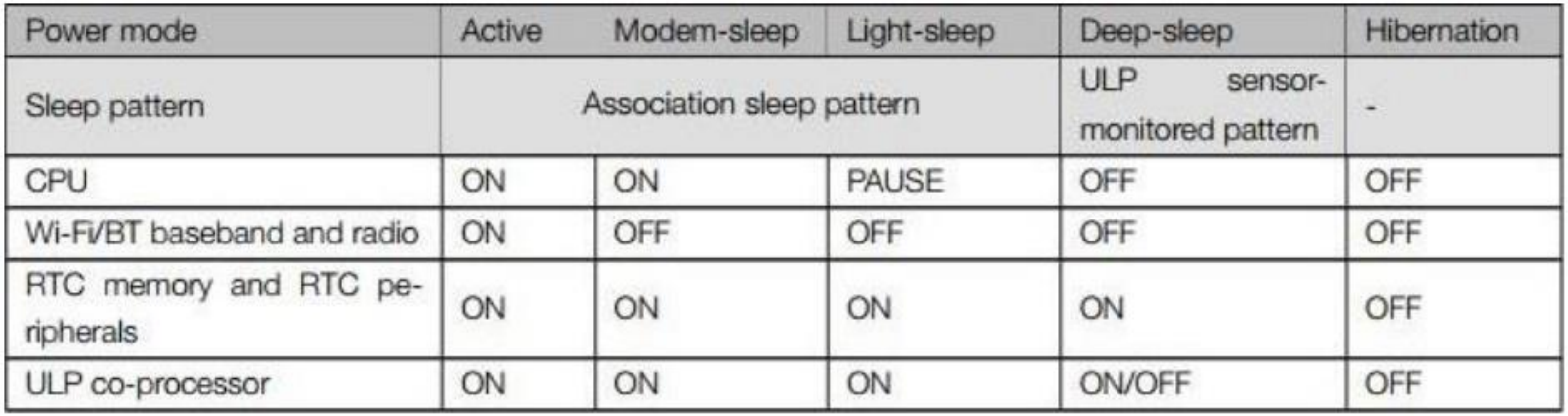
ESP32 Dual Core

```
//Task2code: blinks an LED every 700 ms
void Task2code( void * pvParameters ) {
        Serial.print("Task2 running on core ");
        Serial.println(xPortGetCoreID());
        for(;;) {
                digitalWrite(led2, HIGH);
                delay(700);
                digitalWrite(led2, LOW);
                delay(700);
        }
}


void loop() {
}
```

ESP32 Deep Sleep

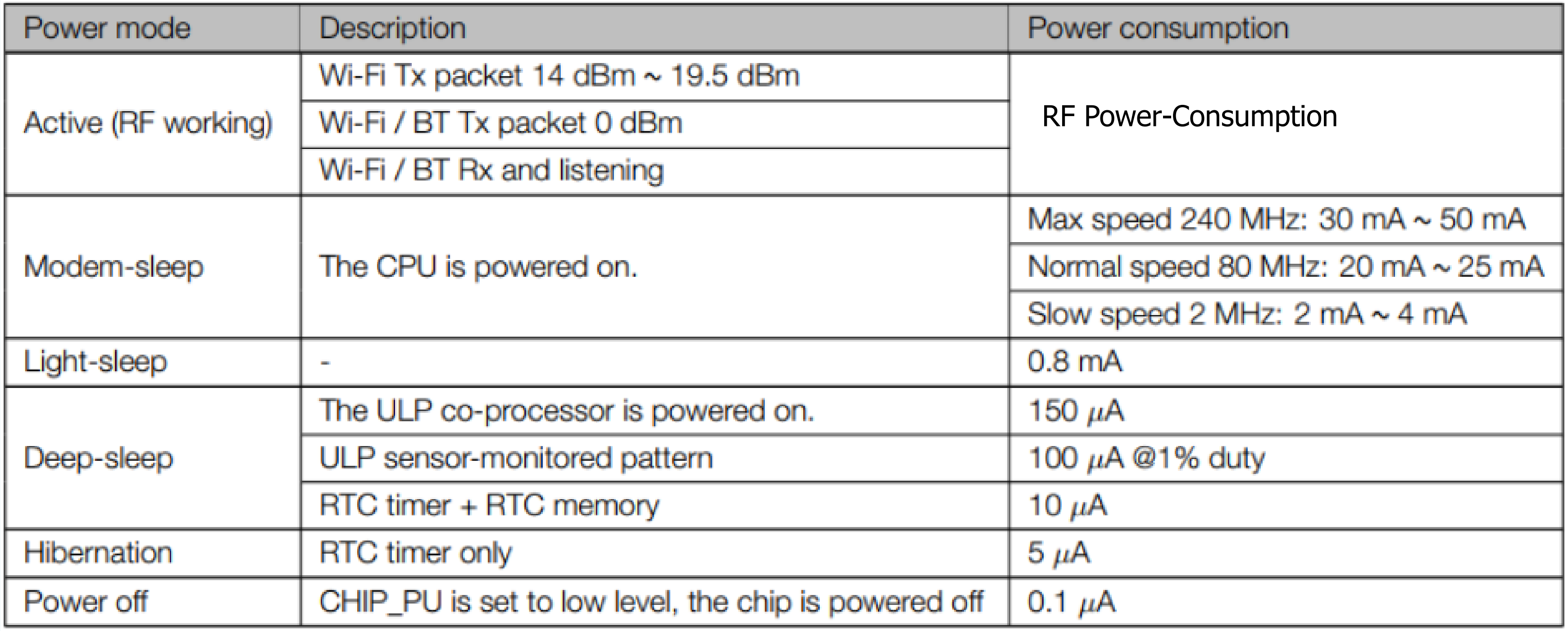
- ESP32 has 5 different power modes:
  - Active mode
  - Modem Sleep mode
  - Light Sleep mode
  - Deep Sleep mode
  - Hibernation mode



Power mode	Active	Modem-sleep	Light-sleep	Deep-sleep	Hibernation
Sleep pattern	Association sleep pattern			ULP sensor-monitored pattern	-
CPU	ON	ON	PAUSE	OFF	OFF
Wi-Fi/BT baseband and radio	ON	OFF	OFF	OFF	OFF
RTC memory and RTC peripherals	ON	ON	ON	ON	OFF
ULP co-processor	ON	ON	ON	ON/OFF	OFF

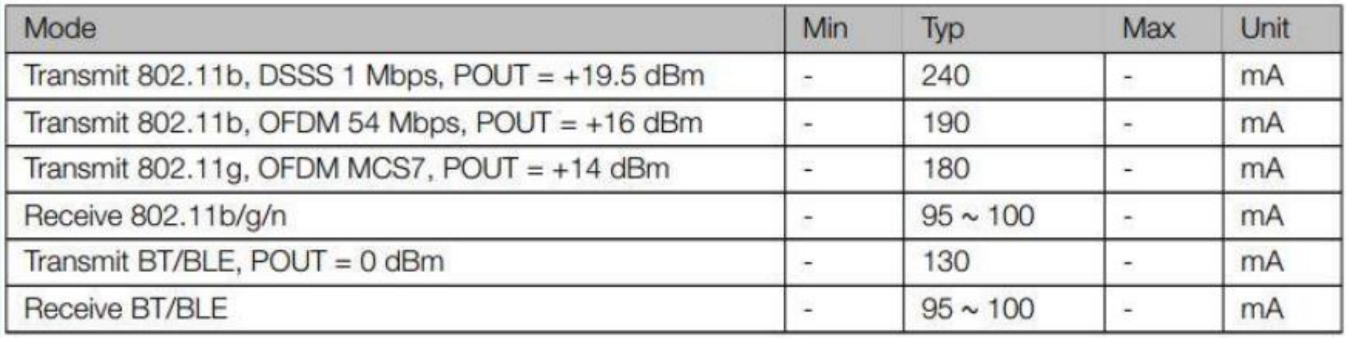
ESP32 Deep Sleep

- Power Consumption



Power mode	Description	Power consumption
Active (RF working)	Wi-Fi Tx packet 14 dBm ~ 19.5 dBm	RF Power-Consumption
	Wi-Fi / BT Tx packet 0 dBm	
	Wi-Fi / BT Rx and listening	
Modem-sleep	The CPU is powered on.	Max speed 240 MHz: 30 mA ~ 50 mA
		Normal speed 80 MHz: 20 mA ~ 25 mA
		Slow speed 2 MHz: 2 mA ~ 4 mA
Light-sleep	-	0.8 mA
Deep-sleep	The ULP co-processor is powered on.	150 μA
	ULP sensor-monitored pattern	100 μA @1% duty
	RTC timer + RTC memory	10 μA
Hibernation	RTC timer only	5 μA
Power off	CHIP_PU is set to low level, the chip is powered off	0.1 μA

RF Power-Consumption



Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	-	240	-	mA
Transmit 802.11b, OFDM 54 Mbps, POUT = +16 dBm	-	190	-	mA
Transmit 802.11g, OFDM MCS7, POUT = +14 dBm	-	180	-	mA
Receive 802.11b/g/n	-	95 ~ 100	-	mA
Transmit BT/BLE, POUT = 0 dBm	-	130	-	mA
Receive BT/BLE	-	95 ~ 100	-	mA

ESP32 Deep Sleep

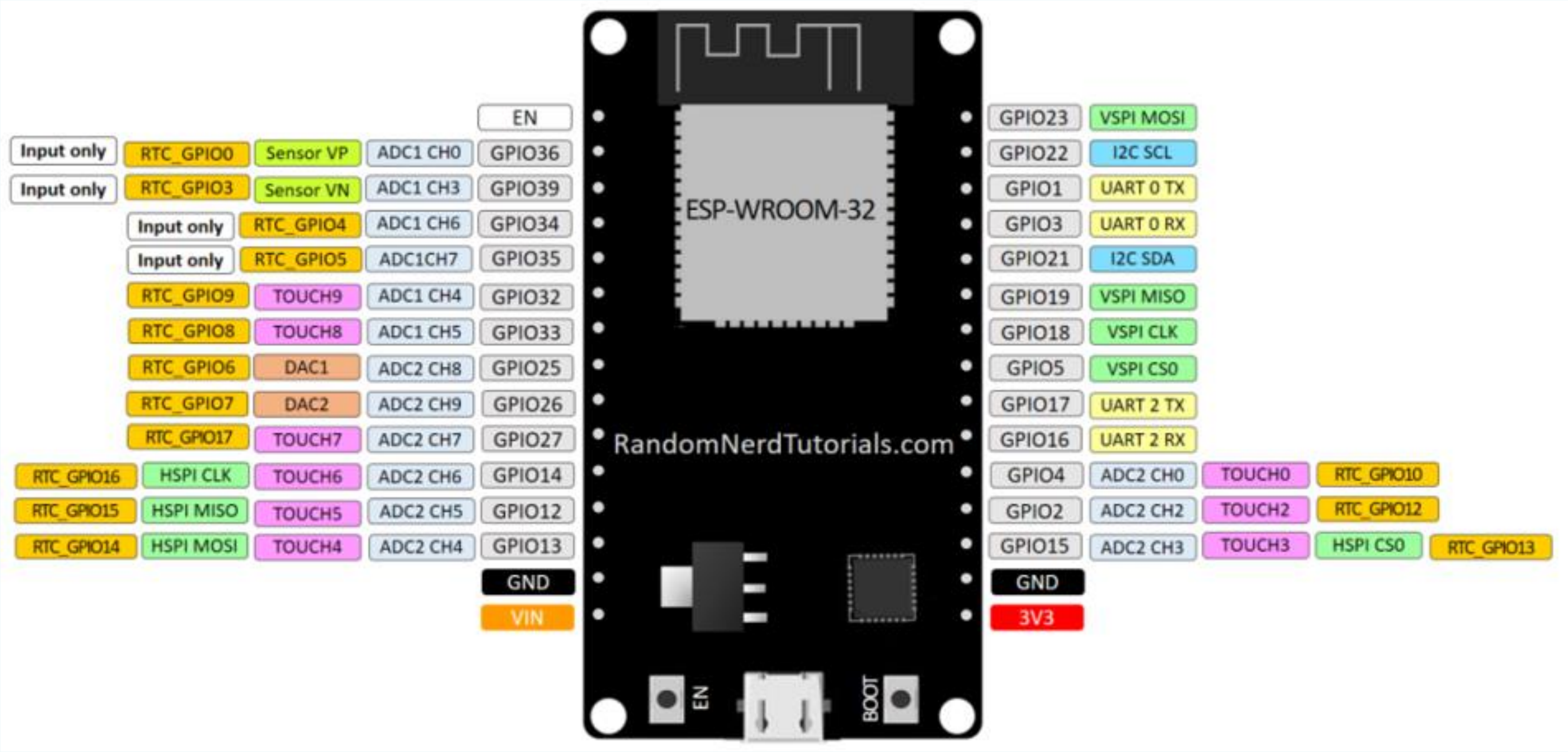
- Why deep sleep?

- ✓ Batteries drain very quickly in active mode
- ✓ Deep sleep mode makes batteries last longer
  - ✓ Cutting activities with more power, but leave just enough activity to wake up the processor when something interesting happens
- ✓ In deep sleep mode neither CPU nor Wi-Fi activities take place
  - ✓ but the Ultra Low Power (ULP) co-processor can still be powered on
  - ✓ RTC memory also remains powered on
- ✓ Useful to wake up CPU by an external event or timer, or both
  - ✓ Maintaining minimal power consumption.peripheral devices

ESP32 Deep Sleep

- RTC_GPIO Pins

✓ During deep sleep, some ESP32 pins can be used by ULP co-processor
: RTC_GPIO pins (0,2,4,12-15,25-27,32-39), Touch Pins

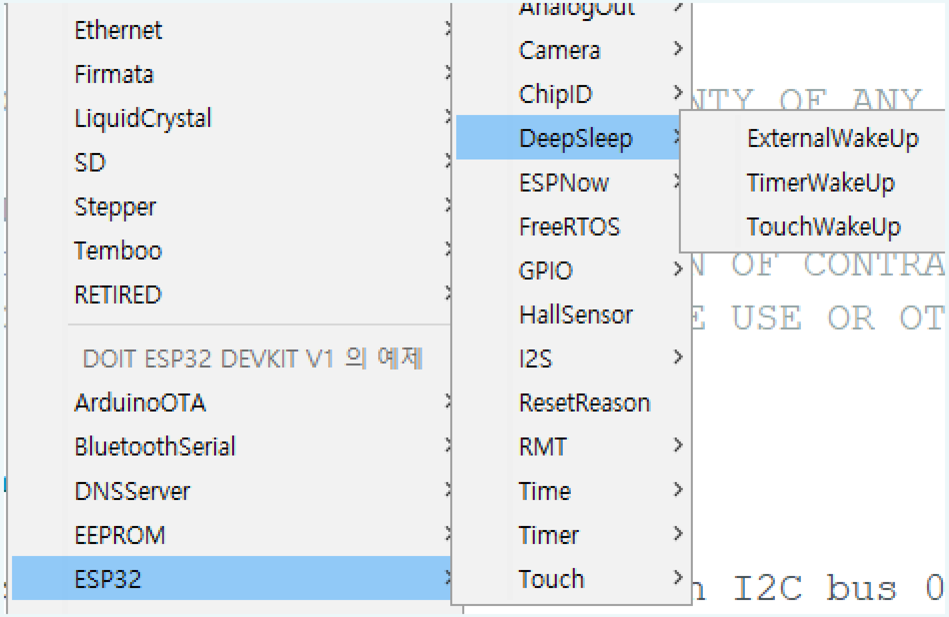


ESP32 Deep Sleep

- Wake-Up Sources
  - Timer : predefined periods of time
  - Two possibilities of external wake-up:
    - One external wake-up or
    - Several different external wake-ups
  - Touch pins
  - ULP co-processor
- Writing Deep Sleep codes
  - Configure the wake-up sources
  - Decide peripherals to shut down or to keep on
    - By default, ESP32 automatically powers down the peripherals that are not needed with the wake-up source you define
  - Use esp_deep_sleep_start() function to put ESP32 into deep sleep mode

ESP32 Deep Sleep

- Example code
  - Push Button to GPIO 33 pulled down with a 10K Ohm resistor



ESP32 Deep Sleep – Wakeup Reason

```
. . .
RTC_DATA_ATTR int bootCount = 0;

void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;
  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch(wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP program"); break;
    default : Serial.printf("Wakeup was not caused by deep sleep: %d\n",wakeup_reason); break;
  }
}

. . .
```

ESP32 Deep Sleep – Touch Reason

```
void print_wakeup_touchpad(){
  touchPin = esp_sleep_get_touchpad_wakeup_status();

  switch(touchPin)
  {
    case 0  : Serial.println("Touch detected on GPIO 4"); break;
    case 1  : Serial.println("Touch detected on GPIO 0"); break;
    case 2  : Serial.println("Touch detected on GPIO 2"); break;
    case 3  : Serial.println("Touch detected on GPIO 15"); break;
    case 4  : Serial.println("Touch detected on GPIO 13"); break;
    case 5  : Serial.println("Touch detected on GPIO 12"); break;
    case 6  : Serial.println("Touch detected on GPIO 14"); break;
    case 7  : Serial.println("Touch detected on GPIO 27"); break;
    case 8  : Serial.println("Touch detected on GPIO 33"); break;
    case 9  : Serial.println("Touch detected on GPIO 32"); break;
    default : Serial.println("Wakeup not by touchpad"); break;
  }
}

void setup(){
  . . .
  ++bootCount;
  . . .
  print_wakeup_reason();
  print_wakeup_touchpad();
  touchAttachInterrupt(T3, callback, Threshold); //Setup interrupt on Touch Pad 3 (GPIO15)
  esp_sleep_enable_touchpad_wakeup();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}
```

ESP32 Deep Sleep – External Trigger

```
void setup(){
  Serial.begin(115200);
  delay(1000);

  //Increment boot number and print it every reboot
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));

  //Print the wakeup reason for ESP32
  print_wakeup_reason();

  //Configure GPIO33 as ext0 wake up source for HIGH logic level
  esp_sleep_enable_ext0_wakeup(GPIO_NUM_33,1);

  //Go to sleep now
  esp_deep_sleep_start();
}
```

ESP32 Deep Sleep – External Trigger

```
//Pushbuttons connected to GPIO32 & GPIO33
#define BUTTON_PIN_BITMASK 0x300000000

RTC_DATA_ATTR int bootCount = 0;

void setup(){
  Serial.begin(115200);
  delay(1000);

  //Increment boot number and print it every reboot
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));

  //Print the wakeup reason for ESP32
  print_wakeup_reason();

  //Configure GPIO32 & GPIO33 as ext1 wake up source for HIGH logic level
  esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK,ESP_EXT1_WAKEUP_ANY_HIGH);

  //Go to sleep now
  esp_deep_sleep_start();
}
```