Module 8 - Power Management

Tujuan Pembelajaran

Setelah menyelesaikan modul ini, praktikan diharapkan mampu:

- Memahami tingkat konsumsi daya pada ESP32 berdasarkan proses yang dijalankan di dalamnya.
- Mengetahui metode-metode yang ada untuk mengurangi jumlah konsumsi daya ESP32.
- Mengetahui berbagai konfigurasi ESP32 berdasarkan tingkat konsumsi dayanya serta menentukan mode penggunaan yang paling cocok dengan kebutuhan aplikasi.

Konsumsi Daya pada ESP-32

Sebagai mikrokontroler, ESP-32 memerlukan pasokan daya yang stabil agar dapat beroperasi dengan optimal. Daya tersebut digunakan oleh core prosesor untuk menjalankan berbagai, serta oleh komponen pendukung seperti I^2C, Wi-Fi, dan Bluetooth untuk melakukan komunikasi.

Apabila ESP32 terhubung secara langsung dan terus-menerus ke sumber daya seperti USB power, kebutuhan daya ini umumnya tidak menjadi masalah. Namun, ketika ESP32 digunakan dengan sumber daya terbatas seperti baterai ataupun power bank dan diharapkan dapat beroperasi secara remote dalam jangka waktu lama, maka efisiensi konsumsi daya menjadi faktor penting agar perangkat dapat berfungsi selama mungkin.

Besaran Konsumsi Daya ESP-32

Konsumsi daya ESP-32 berkisar dari rentang ratusan mA hingga satuan µA. Besaran ini bergantung pada jenis mode kerja yang digunakan oleh ESP-32.

Detail dari jenis mode kerja beserta besaran konsumsi daya ESP-32 dapat dilihat pada tabel berikut : image Pembahasan mendetail mengenai setiap jenis power mode akan dilakukan pada subbab selanjutnya.

Pengukuran Daya ESP-32

Pembacaan konsumsi daya pada ESP32 ditunjukkan melalui besaran arus yang diterima yang memiliki skala kecil, sehingga diperlukan ampermeter dengan tingkat akurasi tinggi. Hal ini dikarenakan ESP32 memiliki rentang arus yang sangat dinamis pada mode aktif, arus dapat mencapai ratusan miliampere, sedangkan pada mode deep sleep, arus dapat turun hingga beberapa mikroampere saja.

Ampermeter umum seperti yang digunakan pada multimeter tidak mampu merespons perubahan arus yang terjadi dengan sangat cepat ini. Alat tersebut hanya akan menampilkan nilai rata-rata, bukan fluktuasi aktual yang mencerminkan perbedaan konsumsi daya antar mode. Selain itu, multimeter konvensional memiliki resistansi internal yang relatif tinggi, yang dapat menyebabkan penurunan tegangan yang menyebabkan ketidakstabilan ESP-32.

Perlu diketahui bahwa jika pengukuran daya dilakukan pada development board seperti ESP32 Doit Devkit, pembacaan tidak akan merepresentasikan mikrokontroler dikarenakan development board sudah dilengkapi komponen-komponen lainnya seperti voltage regulator dan LED yang tetap akan mengonsumsi daya meskipun ESP-32 dikonfigurasikan dalam mode deep sleep. Informasi lebih lanjut mengenai pengukuran arus dapat ditemukan pada

dokumentasi berikut dan demonstrasi dalam video berikut.

Metode Mengurangi Konsumsi Daya ESP-32 : Mengurangi Clock Speed CPU

Terdapat beberapa faktor yang mempengaruhi tingkat konsumsi daya ESP-32, diantaranya sebagai berikut:

Mengurangi Clock Speed CPU

Menurunkan clockspeed (kecepatan clock) pada ESP32 dapat secara langsung menurunkan konsumsi daya karena frekuensi clock berpengaruh terhadap jumlah siklus kerja prosesor per detik. Semakin rendah clockspeed, semakin sedikit jumlah operasi yang dilakukan dalam satu waktu, sehingga arus yang digunakan juga berkurang.

Penggantian frequensi dari CPU hanya dapat dilakukan ke beberapa fixed value, diantaranya

```
240, 160, 80    <<< For all XTAL types
40, 20, 10      <<< For 40MHz XTAL
26, 13          <<< For 26MHz XTAL
24, 12          <<< For 24MHz XTAL
```

XTAL dalam sini merupakan crystal oscillator yang digunakan oleh mikrokontroler. Untuk ESP32, umumnya XTAL 40Mhz digunakan sehingga ESP32 juga mendukung frekuensi 40, 20, dan 10.

Terdapat dua cara penggantian frekuensi CPU, yaitu sebagai berikut:

1.Mengganti langsung melalui Arduino IDE

Pada bagian tools > CPU Frequency > dengan tampilan berikut

image

Dari menu ini, dapat dilakukan pemilihan nilai frekuensi yang diinginkan.

2. Menggunakan Function setCpuFrequencyMhz()

Nilai-nilai yang serupa dapat dipilih dan diganti menggunakan function berikut

```
bool setCpuFrequencyMhz(uint32_t cpu_freq_mhz);

//Contoh Penggunaan:

setCpuFrequencyMhz(80);
//Kode diatas akan mengubah clockspeed menjadi 80MHz
```

Berdasarkan gambar setting pada Arduino IDE, dapat dilihat bahwa (WiFi/BT) hanya terdapat pada rentang frekuensi yang lebih besar atau sama dengan 80MHz, ini artinya modul WiFi atau bluetooth dari ESP32 tidak dapat digunakan pada rentang frekuensi lebih kecil dari 80MHz

Cara ESP32 membentuk frekuensi 80, 160, dan 240 MHz adalah dengan memanfaatkan PLL (Phased Locked Loop) yang cara kerja nya dapat dipelajari melalui video ini.

Metode Mengurangi Konsumsi Daya ESP-32 : Mengganti Operating Mode

Seperti pada tabel sebelumnya, terdapat bebeerapa operating mode yang didukung oleh ESP-32, diantaranya sebagai berikut:

Deep Sleep

Mode Deep Sleep merupakan mode daya sangat rendah di mana hampir seluruh sistem pada ESP32 dimatikan, sehingga menghasilkan konsumsi energi yang sangat kecil.

Dalam mode ini, ESP32 hanya mengonsumsi arus sebesar beberapa mikroampere, menjadikannya sangat ideal untuk aplikasi yang menggunakan daya baterai atau sistem dengan pasokan energi terbatas.

Saat berada dalam Deep Sleep, ESP32 berhenti mengeksekusi program dan masuk ke kondisi suspended. Sebagian besar sirkuit internal mikrokontroler akan dimatikan, kecuali beberapa komponen penting seperti Real-Time Clock (RTC) yang tetap aktif untuk menjaga waktu dan mendeteksi *wakeup source*.

Bila wakeup source juga dimatikan sehingga ESP hanya bekerja berdasarkan RTC, maka ESP32 memasuki mode Hibernation yang lebih hemat daya namun terbatas, silahkan pelajari perbedaannya pada link berikut

Ketika ESP32 keluar dari mode Deep Sleep, program akan dijalankan kembali dari awal seperti saat perangkat baru dinyalakan. Semua variabel yang disimpan di memori biasa akan hilang, karena daya pada bagian tersebut telah dimatikan. Dengan kata lain, proses ini mirip dengan melakukan reboot pada ESP32.

Penggunaan deep sleep hanya dapat dilakukan bila dilakukan import header berikut `#include <esp_sleep.h>` Sedangkan syntax penggunaan deep sleep adalah sebagai berikut

```
esp_sleep_enable_timer_wakeup(5 * 1000000);
esp_deep_sleep_start();
```

Menggunakan RTC Memory untuk Menyimpan State antar Sleep

Karena RTC Memory tetap dijalankan dalam kondisi sleep, ia dapat digunakan untuk menyimpan state state sederhana yang tetap bertahan antar state boot, seperti penggunaan counter untuk menghitung berapa kali kita memasuki kondisi deep sleep dibawah ini.

```
RTC_DATA_ATTR int bootCount = 0;

void setup()
{
  Serial.begin(115200);
  Serial.println("Starting...");

  bootCount++;
  Serial.println(bootCount);
  esp_deep_sleep(2 * 1000000);  // enter deep sleep for 10 seconds

  // This function will never execute due to Deep Sleep mode
}

void loop()
{
  // This function will never execute due to Deep Sleep mode
}
```

Light Sleep

Mode Light Sleep merupakan salah satu mode konsumsi daya rendah yang tersedia pada ESP32, dan bekerja seperti mode “suspend” pada komputer.Dalam mode ini, ESP32 mengonsumsi daya kurang dari 1 mA yaitu sekitar 800 µA pada ESP32.

Pada kondisi ini, CPU, RAM, dan periferal digital akan diputus dari sumber clock dan tegangan kerjanya diturunkan. Ketika terputus dari clock, komponen-komponen tersebut berhenti berfungsi sementara, namun tetap mendapatkan daya dan mempertahankan statusnya sehingga dapat kembali aktif dengan cepat saat dibangunkan.

Karena mempertahankan seluruh nilai variable dan posisi program counter, light sleep dapat berperan sebagai delay, dengan syarat penggunaan ESP32 tidak memanfaatkan WiFi atau Bluetooth.

Penggunaan light sleep dapat dilakukan dengan syntax berikut.

```
esp_sleep_enable_timer_wakeup(2 * 1000000); //light sleep for 2 seconds
esp_light_sleep_start();
```

Wakeup Source

Parameter yang dipassing kedalam function esp_sleep_enable_timer_wakeup() adalah *Wake Up Source* berupa waktu dalam hal ini 2 × 10^6 µs mikroseconds atau 2 second.

Terdapat beberapa jenis wake up source yang dapat digunakan pada kondisi sleep di ESP32, diantaranya:

Yang didukung di Deep Sleep dan Light Sleep

- Timer
- Touchpad (TouchPin)
- ULP Coprocessor Wakeup
- External Wakeup (Ext0 and Ext1)

Yang hanya didukung oleh Light Sleep

- GPIO Wakeup
- UART Wakeup
- WIFI Wakeup

Syntax penggunaan tiap mode wakeup adalah sebagai berikut:

```
// Light Sleep and Deep Sleep
esp_err_t esp_sleep_enable_timer_wakeup(uint64_t time_in_us);
esp_err_t esp_sleep_enable_touchpad_wakeup(void);
esp_err_t esp_sleep_enable_ulp_wakeup(void);
esp_err_t esp_sleep_enable_ext0_wakeup(gpio_num_t gpio_num, int level);
esp_err_t esp_sleep_enable_ext1_wakeup(uint64_t mask, esp_sleep_ext1_wakeup_mode_t mode);

// Only in Light Sleep
esp_err_t esp_sleep_enable_gpio_wakeup(void);
esp_err_t esp_sleep_enable_uart_wakeup(int uart_num);
esp_err_t esp_sleep_enable_wifi_wakeup(void);
void esp_sleep_enable_gpio_switch(bool enable);
```

Informasi lebih lanjut mengenai wakeup modes dapat dibaca melalui link berikut.

Modem Sleep

Mode Modem Sleep pada ESP32 sering kali menimbulkan kebingungan karena penjelasannya berbeda-beda di berbagai sumber.

Beberapa blog dan pengguna komunitas menyebutkan bahwa Modem Sleep adalah mode di mana WiFi berada dalam keadaan tidur (sleep mode), sementara Bluetooth tetap aktif atau tidak digunakan sama sekali. Namun, menurut dokumentasi resmi dari Espressif, Modem Sleep memang secara khusus mengacu pada WiFi Sleep Mode, yaitu kondisi ketika radio WiFi dimatikan sementara CPU masih tetap berjalan untuk menjalankan tugas lain.

Sedangkan terdapat dua definisi untuk modem sleep. Kita akan membahas implementasi kedua definisi ini.

Definisi 1

```
#include <WiFi.h>
#include <BluetoothSerial.h>
#include "driver/adc.h"
#include <esp_bt.h>

#define STA_SSID "<YOUR-SSID>"
#define STA_PASS "<YOUR-PASSWD>"

BluetoothSerial SerialBT;

void setModemSleep();
void wakeModemSleep();

void setup() {
    Serial2.begin(115200);

    while(!Serial2){delay(500);}

    SerialBT.begin("ESP32test"); //Bluetooth device name
    SerialBT.println("START BT");

    Serial2.println("START WIFI");
```

```
    WiFi.begin(STA_SSID, STA_PASS);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial2.print(".");
    }

    Serial2.println("");
    Serial2.println("WiFi connected");
    Serial2.println("IP address: ");
    Serial2.println(WiFi.localIP());

    setModemSleep();
    Serial2.println("MODEM SLEEP ENABLED FOR 5secs");
}

//void loop() {}
unsigned long startLoop = millis();
bool started = false;
void loop() {
    if (!started && startLoop+5000<millis()){
        // Not use delay It has the own policy
        wakeModemSleep();
        Serial2.println("MODEM SLEEP DISABLED");
        started = true;
    }
}


void setModemSleep() {
    WiFi.setSleep(true);
    setCpuFrequencyMhz(80);
}

void wakeModemSleep() {
    setCpuFrequencyMhz(240);
}
```

Kode ini memperlihatkan bagaimana ESP32 masuk ke mode Modem Sleep setelah berhasil terkoneksi ke jaringan WiFi, dengan cara mengurangi frekuensi clock CPU dan memasukkan modul

WiFi ke mode tidur (sleep) selama 5 detik.

Penurunan frekuensi clock dilakukan menggunakan fungsi setCpuFrequencyMhz(80), yang mengubah kecepatan kerja prosesor dari frekuensi normalnya (misalnya 240 MHz) menjadi 80 MHz. Dengan frekuensi yang lebih rendah, kecepatan pemrosesan memang berkurang, tetapi konsumsi daya juga menurun karena arus dan tegangan yang digunakan CPU menjadi lebih kecil.

Selama periode ini, modul WiFi berada dalam keadaan tidur, sehingga tidak memancarkan atau menerima sinyal. Setelah 5 detik berlalu, sistem “dibangunkan” kembali melalui fungsi setCpuFrequencyMhz(240), yang mengembalikan clock CPU ke frekuensi normal (240 MHz) agar performa penuh dapat digunakan kembali untuk menjalankan tugas-tugas utama ESP32.

Definisi 2

Berdasarkan definisi ke 2 dari modem sleep, dapat dijabarkan sebagai proses untuk mengnonaktifkan Wi-Fi, Bluetooth, dan mengurangi frekuensi CPU seperti pada cuplikan kode berikut.

```
#include <WiFi.h>
#include <BluetoothSerial.h>
#include "driver/adc.h"
#include <esp_bt.h>

#define STA_SSID "<YOUR-SSID>"
#define STA_PASS "<YOUR-PASSWD>"

BluetoothSerial SerialBT;

void setModemSleep();
void wakeModemSleep();

void setup() {
    Serial2.begin(115200);

    while(!Serial2){delay(500);}

    SerialBT.begin("ESP32test"); //Bluetooth device name
    SerialBT.println("START BT");

    Serial2.println("START WIFI");
    WiFi.begin(STA_SSID, STA_PASS);
```

```
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial2.print(".");
    }

    Serial2.println("");
    Serial2.println("WiFi connected");
    Serial2.println("IP address: ");
    Serial2.println(WiFi.localIP());

    setModemSleep();
    Serial2.println("MODEM SLEEP ENABLED FOR 5secs");
}

//void loop() {}
unsigned long startLoop = millis();
bool started = false;
void loop() {
    if (!started && startLoop+5000<millis()){
        // Not use delay It has the own policy
        wakeModemSleep();
        Serial2.println("MODEM SLEEP DISABLED");
        started = true;
    }
}

void disableWiFi(){
    adc_power_off();
    WiFi.disconnect(true);  // Disconnect from the network
    WiFi.mode(WIFI_OFF);    // Switch WiFi off
    Serial2.println("");
    Serial2.println("WiFi disconnected!");
}
void disableBluetooth(){
    // Quite unusefully, no relevable power consumption
    btStop();
    Serial2.println("");
    Serial2.println("Bluetooth stop!");
}
```

```
void setModemSleep() {
    disableWiFi();
    disableBluetooth();
    setCpuFrequencyMhz(80);
}

void enableWiFi(){
    adc_power_on();
    delay(200);

    WiFi.disconnect(false);  // Reconnect the network
    WiFi.mode(WIFI_STA);    // Switch WiFi off

    delay(200);

    Serial2.println("START WIFI");
    WiFi.begin(STA_SSID, STA_PASS);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial2.print(".");
    }

    Serial2.println("");
    Serial2.println("WiFi connected");
    Serial2.println("IP address: ");
    Serial2.println(WiFi.localIP());
}

void wakeModemSleep() {
    setCpuFrequencyMhz(240);
    enableWiFi();
}
```

Referensi Lebih Lanjut

Penggunaan mode-mode sleep dalam kode belum dibahas secara terlalu detail dalam modul ini, silahkan refer ke Light Sleep untuk mempelajari penggunaan light sleep dan Deep Sleep untuk mempelajari penggunaan Deep Sleep pada kode.

Referensi Lainnya:

- “Power Management - ESP32 - — ESP-IDF Programming Guide v5.5.1 documentation,” Espressif.com, 2016. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/power_management
- L. Llamas, “Energy consumption in ESP32,” Luis Llamas, Sep. 22, 2023. https://www.luisllamas.es/en/esp32-power-consumption/ (accessed Nov. 02, 2025).
- Renzo Mischianti, “ESP32 practical power saving: manage WiFi and CPU - 1,” Renzo Mischianti, Mar. 06, 2021. https://mischianti.org/esp32-practical-power-saving-manage-wifi-and-cpu-1/ (accessed Nov. 02, 2025).