Indice

1. **INTRODUZIONE ALLE ESP32**

Le schede ESP32, sviluppate da Espressif Systems, costituiscono una famiglia di microcontrollori basati sull'architettura Xtensa LX6. Questi dispositivi sono ampiamente impiegati nei progetti IoT grazie alla loro flessibilità, potenza di calcolo e capacità di gestire diverse connessioni. L'architettura Xtensa LX6, multicore, offre prestazioni elevate e un set completo di funzionalità, rendendo le ESP32 adatte a una vasta gamma di applicazioni.

La versatilità delle ESP32 le rende ideali per progetti IoT, in cui la connettività e il risparmio energetico sono essenziali. La capacità di gestire in modo efficace la modalità a basso consumo di energia le rende adatte per dispositivi alimentati a batteria.

**1.1 Architettura Xtensa LX6**

L'architettura Xtensa LX6, progettata per offrire prestazioni elevate e flessibilità nelle istruzioni, include due core multicore dedicati all'esecuzione di task. Questa caratteristica consente una gestione efficiente delle attività, separando le funzioni utente da quelle di sistema.

**1.2 Flessibilità e Potenza nelle Applicazioni IoT**

Le ESP32 sono particolarmente adatte per le applicazioni IoT grazie alla loro flessibilità e alle capacità di connessione integrate, come Wi-Fi e Bluetooth. Possono essere impiegate in progetti che richiedono comunicazioni wireless, acquisizione di dati da sensori e gestione di dispositivi remoti. La presenza di funzionalità avanzate le rende una scelta popolare per sviluppatori e progettisti di sistemi embedded.

**1.3 Utilizzo di FreeRTOS**

FreeRTOS, un sistema operativo in tempo reale open-source, è spesso impiegato nelle ESP32 per gestire attività in modo efficiente e sincronizzato. Grazie alla natura multicore delle ESP32, FreeRTOS consente di distribuire attività su core specifici, ottimizzando l'utilizzo delle risorse.

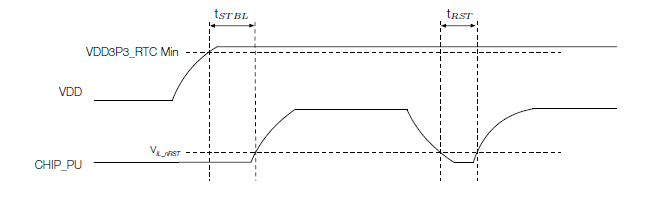
L'utilizzo di FreeRTOS consente una gestione avanzata delle attività, inclusa l'assegnazione di priorità, la sincronizzazione delle operazioni e la gestione delle risorse in modo flessibile. Questo è particolarmente utile in applicazioni complesse, come progetti IoT, in cui è necessario gestire diverse attività simultaneamente.

La combinazione di ESP32 e FreeRTOS offre agli sviluppatori la potenza di calcolo e la flessibilità necessarie per affrontare sfide complesse nelle applicazioni embedded, consentendo una gestione più efficiente delle risorse e una migliore ottimizzazione del consumo energetico. La loro versatilità le rende ideali per progetti IoT, dove la connettività e il risparmio energetico sono cruciali. Inoltre, le ESP32 sono conosciute per la loro capacità di gestire in modo efficace la modalità a basso consumo di energia, rendendole adatte per dispositivi alimentati a batteria.

1. **POWER MANAGEMENT**

La sezione power management è dedicata alla spiegazione di come il dispositivo gestisce l'alimentazione e il consumo energetico.

Una volta alimentato il chip, è necessario un breve lasso di tempo per stabilizzare il circuito interno. Dopo questo, CHIP\_PU (il pin usato per power-up e reset) viene alimentato per attivare tutto il dispositivo.



Quando viene utilizzata una batteria per alimentare la ESP32 viene raccomandato un modulo di gestione della tensione (supervisor) per evitare avvengano errori di avvio, dovuti a scarsa tensione. Viene raccomandato di impostare CHIP\_PU basso se la tensione di alimentazione è inferiore a 2.3 V.

**2.1 Modalita’ di alimentazione**

Vi sono cinque modalità di alimentazione progettate per scenari diversi: Modalità attiva, Modem-sleep mode, light-sleep mode, Deep-sleep mode, Hibernation.

* Active Mode: Il chip per la gestione delle comunicazioni radio è alimentato. Può ricevere, trasmettere o rimanere in ascolto.
* Modem-sleep mode: la CPU lavora correttamente e il clock è configurabile. Wi-fi, Bluetooth sono disabilitati
* Light-sleep mode: la CPU è in pausa. Sono in esecuzione la memoria RAM e le periferiche RTC (Real Time Clock - tutte quelle periferiche che tengono traccia della data e dell'ora). Queste periferiche appunto sono progettate per essere precise e mantenere l'ora corretta anche quando il dispositivo è spento o in modalità di risparmio energetico. Ogni sollecito (MAC, SDIO host, RTC timer o interruzioni esterne) sveglia il chip. All’uscita dalla light-sleep, le periferiche e le CPU riprendono a funzionare, ed il loro stato interno è preservato.
* Deep-sleep mode: modalità sonno profondo. Solamente la memoria e le periferiche RTC sono alimentate. Connessioni Bluetooth e wi-fi sono immagazzinate nella memoria RTC. Il coprocessore ULP è in funzione (ULP coprocessor è un processore progettato per eseguire solo operazioni specifiche in modalità a bassissima potenza, quindi alto risparmio energetico).
* Hibernation Mode: l'oscillatore interno a 8 MHz e il coprocessore ULP sono disabilitati. La recovery memory RTC non è alimentata. Solamente il timer RTC a basse frequenze e alcuni pin GPIO RTC è in grado di risvegliare il chip dalla modalità di ibernazione

**2.2 Tabella consumi**

Una volta analizzate le varie modalità possiamo consultare lo schema dei consumi presente nel datasheet per ogni modalità.

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

Quando il Wi-fi viene attivato il chip cambia automaticamente tra Modalità attiva e modem-sleep mode, in quest’ultima la frequenza della CPU varia automaticamente in base al carico su di essa e le periferiche usate.

Nella modalita’ attiva (RF working) la descrizione “Wi-fi/BT Tx Packet indica che il dispositivo è attivo e sta trasmettendo pacchetti dati attraverso le due connessioni Wi-fi o BT. In questo stato serve maggiore energia per supportare la trasmissione radio (RF – Radio Frequency) e il consumo dipende dalla durata della trasmissione.

Di seguito la tabella dei consumi della ESP nel suo stato attivo di trasmissione dati

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Nella stessa modalita’ il dispositivo è attivo per la ricezione dei dati attraverso le connessioni indicate.

Per ridurre al minimo il consumo di energia su una scheda ESP32 durante lo stato di ricezione dati mantenendo la capacità di attivarsi tramite segnale esterno, si possono considerare l’utilizzo di modalita’ a basso consumo come la Light Sleep e la Modem Sleep.

Modalità Light Sleep (Sleep Light): In questa modalità, la CPU è spenta, ma la periferica Wi-Fi, a differenza della modalità deep-sleep, può essere mantenuta attiva tramite comando per continuare a ricevere pacchetti. La modalità Light Sleep consente anche di programmare interrupt esterni che possono risvegliare il dispositivo. Questa modalità è utile quando è necessario mantenere la connettività Wi-Fi e la capacità di risvegliarsi rapidamente in risposta a segnali esterni.

Modalità Modem Sleep (Sleep Modem): In questa modalità, il modem Wi-Fi è spento quando non è coinvolto in attività di trasmissione o ricezione, ma il processore rimane attivo. Questa modalità è più efficace in termini di risparmio energetico rispetto alla modalità Light Sleep se la CPU deve rimanere attiva, ma si vuole disattivare il modem Wi-Fi quando non è necessario. Anche in questo caso, si possono programmare interrupt esterni per risvegliare il dispositivo.

Essendo le due modalita’ molto simili, la scelta tra Modem Sleep e Light Sleep dipenderà esclusivamente dall’equilibrio tra la necessità di risparmio energetico e la riduzione dei tempi di risposta del dispositivo. Il Modem Sleep è utile per applicazioni che necessitano di rimanere connesse alla rete con una pronta capacità di trasmissione dati, mentre il Light Sleep è preferibile quando è possibile tollerare tempi di risveglio leggermente più lunghi in cambio di un significativo risparmio energetico.

Da qui possiamo capire come le due modalita’ ci possono tornare utili per l’acquisizione di dati tramite sensori esterni collegati direttamente. In seguito, tramite l’RTC timer sarà possibile effettuare un cambio di modalita’ e attivare le connessioni per l’invio di dati in maniera schedulata nel tempo.

1. **SISTEMA E FRAMEWORKS**

**3.1 Sistemi Operativi: FreeRTOS**

Una volta capiti e studiati gli stati presenti nel microcontrollore dobbiamo capire come questi possano interagire correttamente con i sistemi che sceglieremo. In particolar modo di concentreremo su sistemi con kernel di sistema real-time freeRTOS e ambienti di sviluppo ideali per dispositivi embedded. L’intenzione è quella di far lavorare i microcontrollori in stati di risparmio energetico per poterli utilizzare alimentati da batteria esterna e non direttamente da cavo, permettendo una maggiore elasticità di utilizzo del dispositivo.

Come abbiamo visto con dai grafici lo stato ideale per permettere al dispositivo di elaborare informazioni rimanendo comunque in risparmio energetico è la Deep-Sleep, in questa modalita’ infatti sia la CPU che il Wi-fi non sono attivi, ma può’ rimanerlo ULP Co-processor.

In questo stato ci sono 3 modi per consentire il risveglio del sistema:

* External Wake Up
* Timer Wake Up
* Touch Wake Up
  1. **Framework di sviluppo: Arduino-IDE - Espressif-IDE**

Vi sono diversi ambienti di sviluppo (IDE) che possono essere utilizzati per programmare i microcontrollori ESP32, ciascuno con le proprie caratteristiche e vantaggi. Su tutti i più utilizzati sono 2: Arduino-IDE e Espressi-IDE. E’ importante notare che la scelta tra i due dipende dalle esigenze specifiche del progetto e dall’esperienza di sviluppo. Arduino infatti offre un approccio più intuitivo e familiare, rendendo più semplice, diversamente da ESP-IDF più completo e potente.

L’installazione della libreria ESP32 all’interno dell’Arduino IDE ci consente di avere a disposizione tutte le funzionalità per poter gestire al meglio tutti gli stati delle nostre schede. Per un corretto funzionamento è ideale realizzare e codificare una strategia di wake-up dalla modalita Deep-Sleep, in questo modo saremo in grado di tenere sotto controllo l’esecuzione del dispositivo riducendo al minimo i consumi energetici incrementando le prestazioni e la durata.

Nel pacchetto ESP32 la libreria per Arduino-IDE, in grado di gestire le modalita’ di sleep. E’ la **esp\_sleep.h**. All’interno di questa vi sono diverse funzioni ideali ai nostri interessi, in particolare:

* **void esp\_deep\_sleep\_start(void)**

Questa ci permette di mettere nello stato di Deep-Sleep la nostra scheda. Attenzione perché’ questa funzione senza una corretta configurazione di risveglio prima della sia chiamata porta la scheda in uno stato di inattività il cui risveglio è possibile solo attraverso un reset dell’alimentazione. Per evitare questo è opportuno configurare correttamente le impostazioni di “Wake-Up”.

* **esp\_err\_t esp\_sleep\_enable\_timer\_wakeup(uint64\_t time\_in\_us);**

Va invocate prima di **esp\_deep\_sleep\_start()** e si usa per configurare un risveglio da un timer, in microsecondi, l’unico parametro da passare alla funzione durante l’invocazione. Prevede anche un errore di ritorno che potrebbe essere gestito per incrementare l’affidabilità del proprio codice.

* **void esp\_deep\_sleep(uint64\_t time\_in\_us)**

Per comodità di può utilizzare questa funzione in grado di settare il timer e avviare automaticamente la modalita’ Deep-Sleep

Come possiamo immaginare “Timer Wake-Up” diventa fondamentale per quelle applicazioni che richiedono dei task temporali definiti. La funzione sfrutta l’ RTC timer della recovery memory, questa include al suo interno una memoria SRAM da 8kB utilizzabile per salvare dati da non perdere durante la fase di Deep-Sleep. Questi dati ovviamente verranno persi se il dispositivo verrà spento o riavviato.

Nel nostro progetto sfrutteremo l’ambiente di sviluppo di ESPRESSIF IoT Development Framework (ESP-IDF) che offre funzionalità più avanzate rispetto le piattaforme di terze parti. Ecco alcune differenze principali:

* **Controllo completo e flessibile del sistema operativo:** ESP-IDF utilizza FreeRTOS come sistema operativo in tempo reale sottostante, offrendo un controllo più flessibile sulle attività e sulle risorse del sistema rispetto all'ambiente di sviluppo di Arduino. Questo è particolarmente vantaggioso per progetti che richiedono gestione avanzata delle attività in tempo reale e bassa latenza.
* **Gestione Avanzata delle Modalità di Consumo Energetico:** ESP-IDF fornisce una gestione avanzata delle modalità di consumo energetico, consentendo agli sviluppatori di sfruttare funzionalità come Deep Sleep, Light Sleep e altre per ridurre il consumo di energia in applicazioni a bassa potenza.
* **Accesso Diretto alle Funzionalità Hardware:** ESP-IDF offre un accesso più diretto e flessibile alle funzionalità hardware della ESP32 attraverso una vasta gamma di librerie e driver. Questo è particolarmente utile quando è necessario controllare dettagli specifici della periferica o implementare funzionalità personalizzate.
* **Strumenti di Sviluppo Avanzati:** ESP-IDF utilizza il sistema di build basato su CMake e fornisce strumenti avanzati di sviluppo, tra cui debugger e strumenti di analisi delle prestazioni. Questo offre maggiore controllo e facilità di sviluppo in scenari complessi.
* **OTA Updates (Over-The-Air):** ESP-IDF semplifica gli aggiornamenti del firmware in modalità OTA, consentendo di aggiornare il firmware dei dispositivi remotamente. Questa funzionalità può essere utile in scenari in cui è difficile o impossibile accedere fisicamente ai dispositivi per l'aggiornamento del firmware.
* **Flessibilità e Controllo Maggiore:** ESP-IDF offre una maggiore flessibilità e controllo per gli sviluppatori che desiderano ottimizzare le prestazioni e la gestione delle risorse per le loro applicazioni specifiche.

1. **COMPONENTI E DISPOSITIVI**

Una delle parti più interessanti nello sviluppo del funzionamento del microcontrollore a livello embedded è la possibilità di avere il massimo controllo possibile sulle singole componenti. In questo modo siamo in grado di attivare/disattivare tutti i dispositivi integrati nella scheda modificandone il comportamento e andando a influenzarne i consumi.

**4.1 WiFi e Bluetooth**

Come si può notare dallo schema, ci sono molte unità funzionali non necessarie durante i vari stati che possiamo decidere di disattivare manualmente. L’esempio più significativo lo possiamo trovare con i servizi wireless come WiFi e Bluetooth. Questi, infatti, consumano corrente durante la loro attivazione e disattivazione, ma è durante la trasmissione dei dati che richiedono maggiori risorse e quindi maggiori consumi.

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

Nel nostro caso, la fase di comunicazione avviene tramite porta seriale. Possiamo quindi procedere con la disattivazione dei servizi Bluetooth e WiFi nella fase di modem sleep, per fare ciò utilizziamo le funzioni:

* **esp\_err\_t esp\_bluedroid\_disable(void)** – Questa funzione disabilita il servizio BlueDroid, che è il framework di Bluetooth per ESP32. Chiamare questa funzione prima di entrare in modalità di risparmio energetico può aiutare a ridurre il consumo di energia associato al Bluetooth.
* **esp\_err\_t esp\_bt\_controller\_disable(void)** - Questa funzione disabilita il controller Bluetooth. Anche questa operazione contribuisce a ridurre il consumo di energia associato al Bluetooth.
* **esp\_err\_t esp\_wifi\_stop(void)** -Questa funzione interrompe tutti i servizi WiFi e disabilita il modulo WiFi. Disattivare il WiFi durante la fase di modem sleep aiuta a risparmiare energia.

In alcune applicazioni può essere utile riattivare questi servizi una volta usciti dalla modalità Sleep, lo facciamo con:

* **esp\_err\_t esp\_bluedroid\_enable(void);**
* **esp\_err\_t esp\_bt\_controller\_enable(esp\_bt\_mode\_t mode);**
* **esp\_err\_t esp\_wifi\_start(void);**

**4.2 Gestione Core**

Come noto le ESP32 sono dotate di architettura multicore, fornendo due core dedicati all'esecuzione di task. Solitamente, le funzioni invocate dall'utente lavorano sul core 1, mentre le funzioni di sistema sono gestite dal core 0.

Durante le chiamate delle funzioni, è possibile selezionare su quale core la funzione chiamata lavora. Per fare ciò, utilizziamo la funzione xTaskCreatePinnedToCore, inclusa nella libreria freertos/idf\_additions.h. Questa funzione appunto ha il compito di creare una nuova attività (task) ed assegnarla ad uno specifico core del processore.

**xTaskCreatePinnedToCore(**

**coreTask1, /\* Function to implement the task \*/**

**"coreTask1", /\* Name of the task \*/**

**10000, /\* Stack size in words \*/**

**NULL, /\* Task input parameter \*/**

**0, /\* Priority of the task \*/**

**NULL, /\* Task handle. \*/**

**1); /\* Core where the task should run \*/**

**pxTaskCode**: Il nome della funzione che verrà eseguita come task.

**pcName**: Una stringa descrittiva che rappresenta il nome del task.

**usStackDepth**: La dimensione dello stack del task, misurata in parole.

**pvParameters**: Parametri del task, passati alla funzione del task.

**uxPriority**: La priorità del task. Valori più alti indicano una priorità più elevata.

**pxCreatedTask**: Un puntatore a un TaskHandle\_t per ricevere il task handle del task appena creato.

**xCoreID**: Il core a cui assegnare il task (0 o 1).

Se proviamo ad impostare un valore del core diverso da 0 o 1 il task non va in esecuzione.

Immagine che contiene testo, software, Software multimediale, schermata

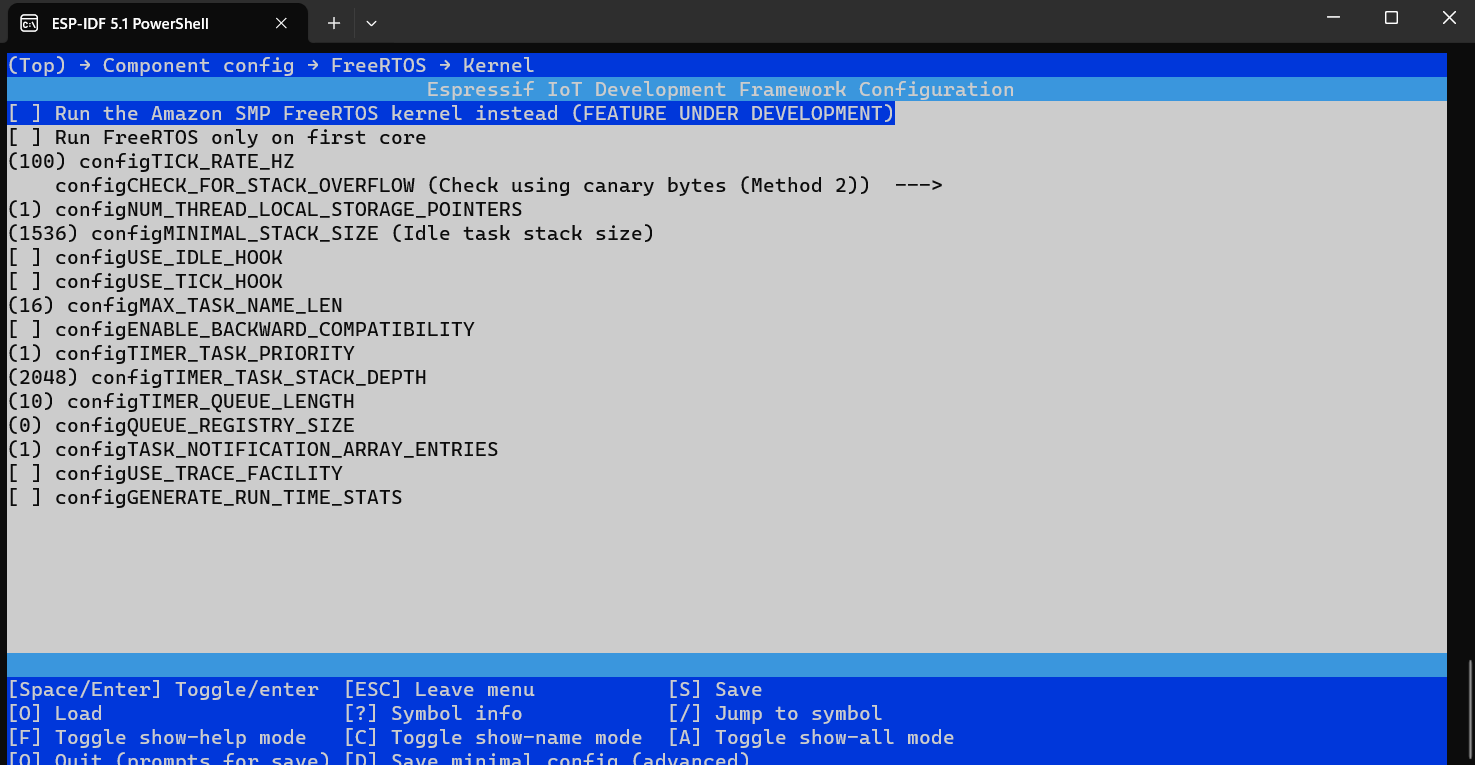
Descrizione generata automaticamente

L'utilizzo di xTaskCreatePinnedToCore è particolarmente utile su dispositivi dual-core come l'ESP32, in modo di poter assegnare diversi task a core diversi per sfruttare al massimo le risorse del processore.

Lavorare in Single Core

Potrebbe essere utile al fine di ridurre ulteriormente i consumi, lavorare su un singolo core. Durante le nostre prove simuleremo l’esecuzione di uno o più task sia nella modalita’ classica che con un core disattivato, per cercare di capire quanto questo incida nei consumi generali e nelle performance.

La disattivazione di uno dei due core però non avviene tramite linea di comando (anche se questo non è del tutto corretto in quanto è possibile inserire il comando all’interno di configurazione). Per impostare la configurare è necessario accedere al menu configurazione del progetto. Ogniuno di questi infatti ha una directory propria, entrando su questa da linea di comando ed eseguendo il comando ‘idf.py menuconfig’ si accede al menu. Qui e’ possibile scegliere l’opzione ‘**Run freeRTOS only on first core‘**



NOTE

## Modalità di sonno[ℑ](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html#id1)

Nella modalità Light-sleep, le periferiche digitali, la maggior parte della RAM e le CPU sono sincronizzate e la loro tensione di alimentazione è ridotta. All'uscita dal sonno leggero, le periferiche digitali, la RAM e le CPU riprendono a funzionare e i loro stati interni vengono preservati.

Nella modalità di sospensione profonda, le CPU, la maggior parte della RAM e tutte le periferiche digitali con clock da APB\_CLK sono spente. Le uniche parti del chip che rimangono accese sono:

* Controllore RTC
* Coprocessore ULP
* Memoria RTC VELOCE
* Memoria RTC LENTA

### Modalità Wi-Fi/Bluetooth e sospensione[ℑ](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html#wi-fi-bluetooth-and-sleep-modes)

Nelle modalità Sonno profondo e Sonno leggero, le periferiche wireless sono spente. Prima di accedere alle modalità di sonno profondo o di sonno leggero, l'applicazione deve disattivare Wi-Fi e Bluetooth utilizzando le chiamate appropriate (ad esempio, [**esp\_bluedroid\_disable()**](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/esp_bt_main.html#_CPPv421esp_bluedroid_disablev), [**esp\_bt\_controller\_disable()**](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/controller_vhci.html#_CPPv425esp_bt_controller_disablev), [**esp\_wifi\_stop()**](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html#_CPPv413esp_wifi_stopv)). Le connessioni Wi-Fi e Bluetooth non vengono mantenute in modalità Sonno profondo o Sonno leggero, anche se queste funzioni non vengono richiamate.

Se è necessario mantenere le connessioni Wi-Fi/Bluetooth, abilitare la modalità di sospensione del modem Wi-Fi/Bluetooth e la funzione di sospensione leggera automatica (vedere [API di risparmio energetico](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/power_management.html) ). Ciò consente al sistema di riattivarsi automaticamente dalla modalità di sospensione quando richiesto dal driver Wi-Fi/Bluetooth, mantenendo così la connessione.