POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Dispositivi IoT basati su Tasmota nel contesto del mining di Bitcoin

Relatore Aziendale
Biomine S.r.l.
Fabrizio Amodio

Relatore Accademico
Prof. Antonio Jose' Di Scala

Candidato
Riccardo Bertelli

Anno Accademico 2023-2024

Sommario

Questa tesi si concentra sull'estensione e la modifica del firmware Tasmota per dispositivi IoT, trattando un caso d'uso pratico nel contesto del mining di Bitcoin.

Il firmware open-source Tasmota, progettato per dispositivi basati su ESP8266/ESP32, fornisce una piattaforma flessibile ed efficiente per la gestione dei dispositivi smart. Tuttavia, la sua configurazione standard potrebbe non soddisfare pienamente i requisiti specifici di tutte le applicazioni, in particolare quelle che richiedono alta affidabilità e funzionalità specializzate, come il mining di Bitcoin.

La ricerca descrive le modifiche necessarie al firmware Tasmota per adattarlo a casi d'uso specifici. Questo include impostazioni configurabili dall'utente e modifiche avanzate al codice, enfatizzando l'uso del protocollo MQTT per delegare l'elaborazione logica a componenti esterni. Tale approccio è essenziale poiché la potenza di elaborazione dell'ESP32 è insufficiente per le operazioni logiche complesse richieste in un'installazione di mining.

Viene mostrata un'implementazione pratica di Tasmota tramite la BioPDU, una PDU basata sul firmware Tasmota, che permette la gestione remota e fornisce notifiche automatiche di malfunzionamenti via Telegram.

La tesi si pone l'obbiettivo di evidenziare le possibilità offerte dalle soluzioni open-source come Tasmota, mostrando la loro flessibilità, convenienza e supporto della comunità. Bitcoin stesso serve come esempio di un progetto open-source di successo, rivoluzionando la valuta digitale e decentralizzando le transazioni finanziarie. Sia Tasmota che Bitcoin esemplificano il potere e il potenziale dell'innovazione open-source.

Ringraziamenti

Desidero in primo luogo ringraziare il Professore Antonio Jose' Di Scala per i preziosi insegnamenti ricevuti durante la laurea magistrale e per le ore che ha dedicato alla mia tesi.

Inoltre, vorrei esprimere la mia sincera gratitudine all'amministratore delegato di Biomine S.r.l., Fabrizio Amodio, per il suo supporto e la sua disponibilità durante il mio percorso di ricerca.

Un sentito ringraziamento va anche a Theo Arends, per aver realizzato Tasmota, e a Satoshi Nakamoto, per la realizzazione del protocollo Bitcoin, le cui innovazioni hanno reso possibile la scrittura della mia tesi.

Infine, desidero ringraziare la mia famiglia per il supporto fornito durante il mio percorso di studi.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Il settore del mining è caratterizzato da un'elevata competitività, una marcata volatilità e un rapido ritmo di cambiamento, richiedendo agli operatori una grande flessibilità e un costante aggiornamento delle competenze. Per mantenere la profittabilità in un ambiente così dinamico, è fondamentale che aziende e professionisti del settore adottino un approccio proattivo e lungimirante. Questo implica non solo l'adeguamento alle nuove tecnologie e metodologie, ma anche la capacità di anticipare le tendenze di mercato. Solo attraverso un impegno continuo nella formazione e nell'innovazione, i miner possono affrontare con successo le sfide di un mercato in continua evoluzione e garantire la sostenibilità delle loro attività.

Una delle aree di maggiore interesse è l'integrazione di dispositivi IoT, che offre potenzialità significative per migliorare l'efficienza e la gestione delle operazioni. Questa tesi si propone di esplorare l'utilizzo e l'estensione del firmware Tasmota per l'integrazione e l'utilizzo di dispositivi IoT nel contesto del mining di Bitcoin. Nel corso di questo lavoro, saranno presentati esempi di modifica e di estensione per consentire l'integrazione di Tasmota in casi d'uso specifici. In particolare, verrà illustrato un esempio di creazione di un driver per uno schermo e-paper non supportato da Tasmota.

Infine, sarà esaminato un caso studio per dimostrare l'applicazione pratica di Tasmota tramite la BioPDU, evidenziando i benefici permessi da questo firmware.

Il resto del documento è organizzato come segue. Nel Capitolo 2 viene fornito il background teorico, con una panoramica sull'IoT, il microcontrollore ESP32, il protocollo MQTT, il firmware Tasmota e la blockchain Bitcoin. Nel Capitolo 3 si analizzano le modalità di estensione del firmware Tasmota, comprese le configurazioni dei driver, le integrazioni esterne e le modifiche del codice sorgente. Nel Capitolo 4 viene presentato un caso studio sull'uso delle BioPDU nel mining di Bitcoin.

Questa struttura permette ai lettori di scegliere quali capitoli approfondire o eventualmente saltare, in base alle loro esigenze specifiche.

Capitolo 2

Background Teorico

2.1 Bitcoin

2.1.1 La blockchain

La blockchain è una tecnologia di registro distribuito che consente di mantenere un elenco crescente di record, chiamati blocchi, in modo sicuro, trasparente e immutabile. Ogni blocco contiene una serie di transazioni e un riferimento criptografico (hash) al blocco precedente, formando così una catena continua.

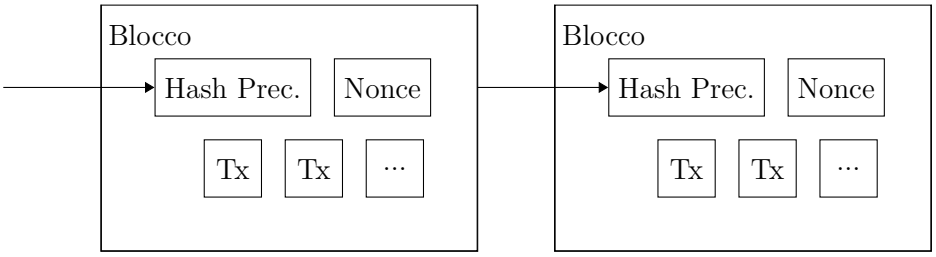

Figura 2.1. Schema di Blockchain basata sulla Proof Of Woork [21]

Questa struttura rende la blockchain resistente alle manomissioni, poiché modificare un singolo blocco richiederebbe il consenso della maggioranza della rete e la revisione di tutti i blocchi successivi.

La blockchain è il fondamento su cui si basano le criptovalute come Bitcoin, garantendo la sicurezza e l'integrità delle transazioni senza la necessità di intermediari centralizzati.

2.1.2 Introduzione a Bitcoin

Bitcoin è una criptovaluta e un sistema di pagamento elettronico peer-to-peer introdotto nel 2008 da un anonimo conosciuto con lo pseudonimo di Satoshi Nakamoto. Il Bitcoin è

stato concepito come una risposta alla crisi finanziaria globale, con l'obiettivo di creare un sistema di transazioni finanziarie decentralizzato, senza la necessità di intermediari come banche o governi.

Le transazioni in Bitcoin vengono verificate attraverso una rete di nodi distribuiti in tutto il mondo. Ogni nodo è un computer che partecipa alla rete Bitcoin e svolge diverse funzioni, tra cui la verifica delle transazioni. Quando una transazione viene inviata alla rete, essa viene trasmessa a tutti i nodi, che verificano la validità della transazione controllando la firma digitale del mittente e assicurandosi che il mittente possieda effettivamente i Bitcoin che sta cercando di inviare.

Il Bitcoin è diventato una delle criptovalute più conosciute e utilizzate al mondo, aprendo la strada a molte altre criptovalute e innovazioni nel campo delle tecnologie finanziarie.

2.1.3 Mining

Quando un utente effettua una transazione Bitcoin, questa viene trasmessa alla rete e raccolta dai nodi in attesa di essere verificata. Alcuni nodi della rete svolgono la funzione di minatori e raggruppano queste transazioni non confermate in un blocco candidato.

Prima di essere aggiunto alla blockchain, ogni transazione nel blocco candidato deve essere verificata dai minatori. Questo include la verifica della firma digitale del mittente e la verifica che il mittente possieda effettivamente i Bitcoin che sta cercando di inviare.

Il cuore del processo di mining è la risoluzione di un complesso problema matematico. Questo problema richiede di trovare un valore, chiamato "nonce", che, quando hashato insieme ai dati del blocco, produce un hash che soddisfa determinati requisiti di difficoltà (un numero con un certo numero di zeri iniziali).

Il processo di risolvere questo problema crittografico è noto come "proof of work" [21].

Il primo minatore che riesce a trovare il nonce corretto condivide la soluzione con il resto della rete.

Una volta verificato che il nonce risolve effettivamente il problema, il blocco candidato viene aggiunto alla blockchain. Il nuovo blocco viene propagato a tutti gli altri nodi della rete, che aggiornano le loro copie della blockchain. Il minatore che risolve il problema e aggiunge il blocco alla blockchain riceve una ricompensa. Questa ricompensa è composta da:

- Un certo numero di nuovi Bitcoin generati (attualmente 3.125 Bitcoin per blocco, ma questo valore si dimezza approssimativamente ogni quattro anni in un evento noto come "halving").
- Le commissioni di transazione pagate dagli utenti che hanno effettuato le transazioni incluse nel blocco.

L'halving

L'halving è un evento significativo nella rete Bitcoin che si verifica approssimativamente ogni quattro anni (o ogni 210.000 blocchi). Durante l'halving, la ricompensa che i minatori ricevono per aggiungere un nuovo blocco alla blockchain viene dimezzata. Questo

evento è codificato nel protocollo Bitcoin e ha implicazioni importanti per l'economia della criptovaluta.

Per capire meglio questo fenomeno, nella figura 2.2 è riportato un grafico che mostra la media su base giornaliera delle ricompense che sono state erogate in ogni blocco durante il tempo, realizzato partendo dai dati grezzi della blockchain [17].

Quando Bitcoin è stato lanciato nel 2009, la ricompensa per ogni blocco minato era di 50 Bitcoin.

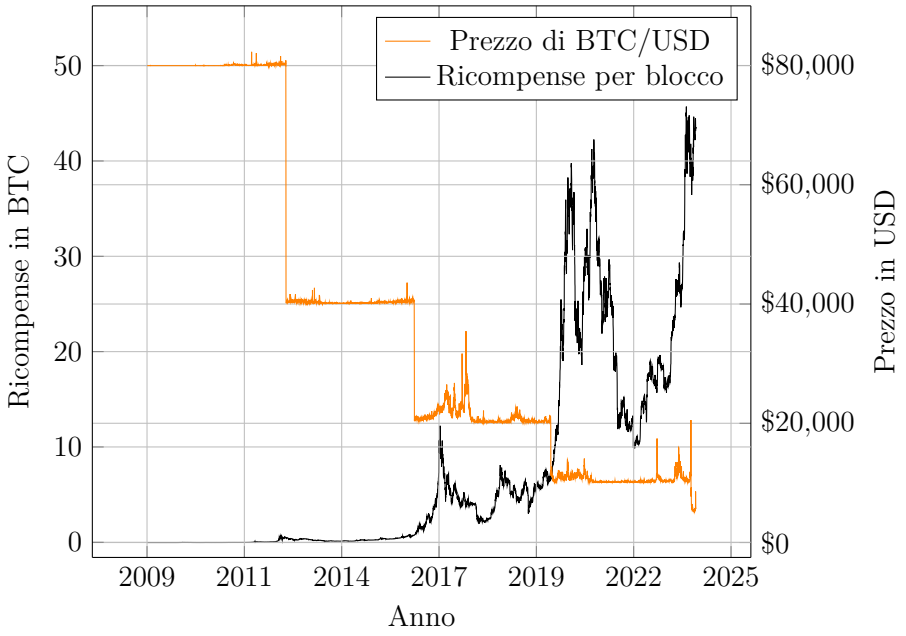

Figura 2.2. Ricompense per blocco nel tempo

Il protocollo Bitcoin è progettato per ridurre la ricompensa del blocco di metà ogni 210.000 blocchi. Dato che, in media, un blocco viene minato ogni 10 minuti, questo significa che l'halving avviene circa ogni quattro anni.

L'halving continuerà ad avvenire fino a quando la ricompensa del blocco non diventerà così piccola da essere trascurabile. Questo è previsto intorno all'anno 2140 [42]. A quel punto, i minatori saranno incentivati principalmente dalle commissioni di transazione piuttosto che dalla ricompensa del blocco.

Nella figura 2.3 è rappresenta l'andamento del numero totale di Bitcoin che sono stati emessi nel tempo. La curva mostra come la quantità di Bitcoin disponibili aumenti in modo prevedibile e limitato [17].

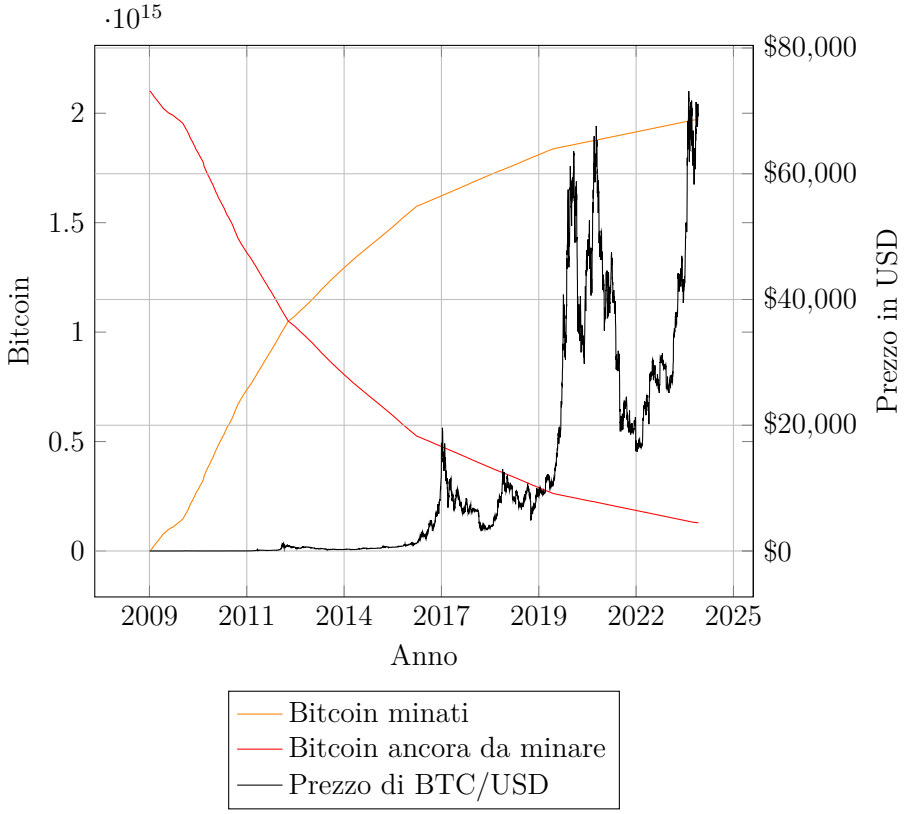

Figura 2.3. Bitcoin supply curve

L'halving è uno degli elementi chiave che distingue Bitcoin da altre forme di valuta. La riduzione programmata della nuova offerta è progettata per imitare l'estrazione di risorse finite, come l'oro, che diventa più difficile da ottenere col passare del tempo. Questo aspetto della scarsità programmata è spesso citato come uno dei motivi per cui Bitcoin è visto come una riserva di valore.

La difficoltà di Mining

La rete Bitcoin regola automaticamente la difficoltà del problema crittografico ogni 2016 blocchi (circa ogni due settimane) per garantire che i blocchi vengano aggiunti alla blockchain a una velocità costante di circa uno ogni dieci minuti. La velocità dei miner dipende dall'**hashrate**, una misura della potenza computazionale totale che i miner stanno utilizzando per estrarre e processare le transazioni Bitcoin. Esprime il numero di calcoli hash che possono essere eseguiti in un secondo. Più alto è l'hashrate, maggiore è la capacità della rete di risolvere il problema crittografico necessario per aggiungere nuovi blocchi alla blockchain. Se i blocchi vengono minati più rapidamente, la difficoltà aumenta; se vengono minati più lentamente, la difficoltà diminuisce.

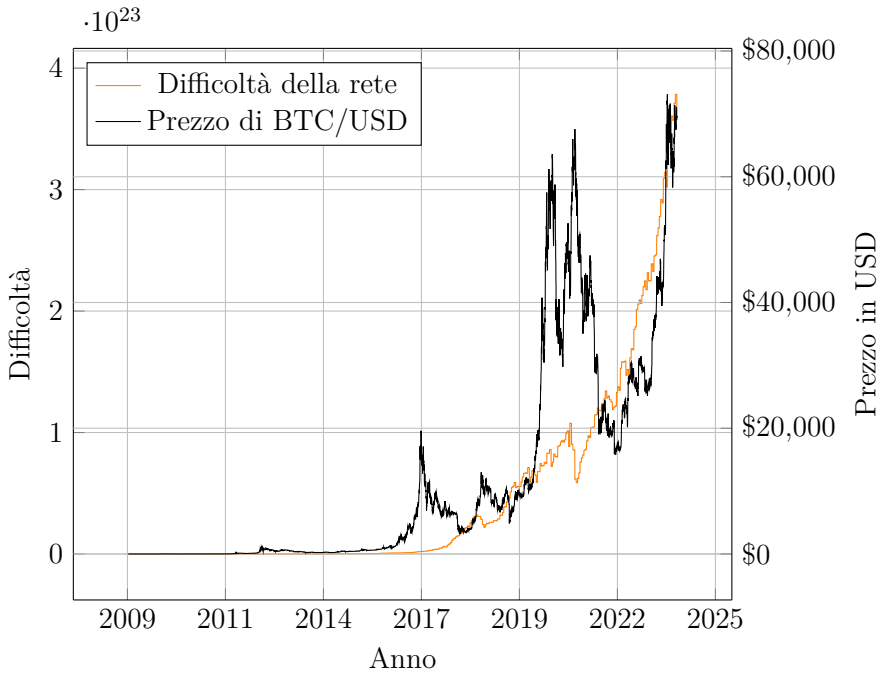

Figura 2.4. Difficoltà della rete nel tempo

La difficoltà di mining è strettamente legata all'hashrate. Non esiste una relazione diretta e immediata tra l'hashrate attuale e la difficoltà, ma si possono effettuare solamente stime, perché l'hashrate attuale non può essere misurato esattamente in tempo reale. Ogni 2016 blocchi, la rete Bitcoin calcola l'hashrate medio di quel periodo e regola la difficoltà di conseguenza. Se l'hashrate medio è aumentato, la difficoltà aumenta per mantenere costante il tempo di generazione dei blocchi. Viceversa, se l'hashrate medio è diminuito, la difficoltà diminuisce.

Sicurezza della rete

Il mining contribuisce alla sicurezza della rete Bitcoin. Poiché i blocchi sono collegati in una catena, modificare un blocco richiederebbe la modifica di tutti i blocchi successivi, cosa che richiederebbe un'enorme quantità di potenza computazionale. Questo rende la blockchain estremamente resistente a manomissioni e frodi. L'elevato hashrate della rete rende infatti praticamente impossibile per un singolo attore ottenere abbastanza potenza computazionale per alterare la blockchain, contribuendo così alla sicurezza e alla robustezza del sistema Bitcoin.

Strumentazione per il mining

Il mining di Bitcoin è iniziato nel 2009 con l'uso delle CPU dei normali computer. Inizialmente, la difficoltà del mining era molto bassa e chiunque con un computer poteva

partecipare e guadagnare Bitcoin.

Con l'aumento della popolarità di Bitcoin e il numero crescente di partecipanti alla rete, la difficoltà del mining è aumentata, rendendo le CPU meno efficienti. Di conseguenza, i miner hanno iniziato a utilizzare le GPU, che, vista la loro predisposizione per i calcoli paralleli, sono in grado di eseguire un numero maggiore di calcoli di hashing per secondo rispetto alle CPU. Le GPU hanno dominato la scena del mining per un certo periodo, offrendo un significativo miglioramento delle prestazioni.

Tuttavia, con il continuo aumento della difficoltà del mining, si è passati a una tecnologia ancora più avanzata: le FPGA, che hanno ulteriormente migliorato l'efficienza e la velocità del mining rispetto alle GPU, ma la vera rivoluzione è arrivata con l'introduzione degli ASIC.

Figura 2.5. Bitmain Bitcoin Miner S21 Pro [5]

Gli ASIC (Application-Specific Integrated Circuits) sono dispositivi hardware progettati specificamente per eseguire il mining di criptovalute in modo estremamente efficiente. A differenza delle CPU, GPU o FPGA, gli ASIC sono ottimizzati per eseguire le operazioni di hashing richieste dalla Proof of Work con una velocità e un'efficienza molto superiori. L'introduzione degli ASIC ha rivoluzionato il settore del mining di Bitcoin, rendendo obsoleti altri tipi di hardware e aumentando notevolmente la difficoltà e la competitività del mining.

2.1.4 Sfide e opportunità nel mining

Il mining di Bitcoin presenta diverse sfide e opportunità. Una delle principali sfide è il consumo energetico elevato associato alla Proof of Work, che ha sollevato preoccupazioni ambientali e ha spinto la ricerca di soluzioni più sostenibili. Inoltre, la centralizzazione del mining in grandi pool e la dipendenza dagli ASIC possono minacciare la decentralizzazione della rete. Tuttavia, il mining offre anche opportunità significative. Può essere una fonte di reddito per individui e aziende che investono in hardware efficiente e si trovano in

aree con costi energetici bassi. Inoltre, le innovazioni tecnologiche e le nuove fonti di energia rinnovabile potrebbero rendere il mining più sostenibile in futuro. Infine, il mining continua a svolgere un ruolo cruciale nella sicurezza e nella stabilità della rete Bitcoin, incentivando la partecipazione e l'innovazione nel settore delle criptovalute.

2.2 Internet of Things (IoT)

2.2.1 Definizione e caratteristiche

L'Internet of Things (IoT) si riferisce all'interconnessione di dispositivi fisici, veicoli, edifici e altri oggetti dotati di sensori, software e altre tecnologie con l'obiettivo di raccogliere e scambiare dati. Questa rete di dispositivi connessi permette loro di comunicare e interagire tra loro e con sistemi centrali, migliorando l'efficienza e l'automazione in vari settori. L'IoT rappresenta un'espansione significativa dell'Internet tradizionale, estendendo la connettività oltre i computer e gli smartphone per includere oggetti quotidiani. Questi dispositivi, noti come "smart objects" o "oggetti intelligenti", sono in grado di percepire l'ambiente circostante, elaborare informazioni e agire in base a determinati criteri predefiniti.

La definizione di IoT comprende anche la capacità di questi oggetti di operare in modo autonomo, riducendo la necessità di intervento umano e migliorando l'efficacia operativa. Grazie ai progressi nei campi della microelettronica, della comunicazione wireless e della tecnologia dei sensori, l'IoT è diventato una realtà concreta con un impatto crescente sulla nostra vita quotidiana e sull'industria.

2.2.2 Applicazioni comuni dell'IoT

Le applicazioni dell'IoT sono molteplici e coprono una vasta gamma di settori, migliorando l'efficienza, la produttività e la qualità della vita. Una delle applicazioni più comuni dell'IoT è nella domotica, dove i dispositivi intelligenti come termostati, luci, serrature e elettrodomestici connessi permettono di automatizzare e controllare gli ambienti domestici da remoto. Questo non solo migliora il comfort e la convenienza, ma può anche contribuire a risparmiare energia e aumentare la sicurezza.

2.3 Il protocolo MQTT

MQTT (Message Queuing Telemetry Transport) è un protocollo di messaggistica leggero e a basso consumo energetico, progettato per la comunicazione asincrona Machine-to-Machine (M2M) e Internet of Things (IoT). Originariamente sviluppato da IBM, MQTT è ora uno standard aperto gestito da OASIS. Il protocollo si distingue per la sua efficacia nelle reti con larghezza di banda limitata o alta latenza, grazie alla sua struttura semplice che riduce l'overhead di comunicazione.

MQTT utilizza un modello di comunicazione pubblicazione-sottoscrizione, in cui i dispositivi (client) si connettono ad un server centrale chiamato broker per inviare e ricevere messaggi. Questo approccio permette una comunicazione efficiente e scalabile tra molti dispositivi, senza richiedere una connessione diretta tra di essi.

La leggerezza di MQTT, unita alla sua efficienza, lo rende ideale per applicazioni IoT che necessitano di una comunicazione affidabile e in tempo reale tra dispositivi distribuiti.

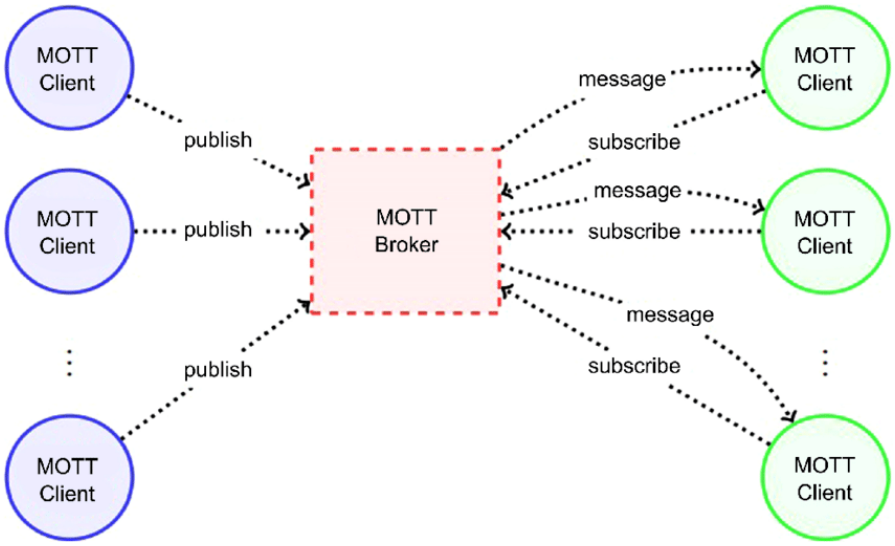

Figura 2.6. Modello di funzionammento di MQTT [1]

MQTT supporta anche la sicurezza tramite TLS, garantendo la protezione dei dati in transito mediante crittografia, autenticazione dei client e integrità dei messaggi. Questo è cruciale per assicurare che le informazioni scambiate tra i dispositivi siano protette da accessi non autorizzati.

2.3.1 Message Brokers

I message brokers sono elementi centrali nell'architettura MQTT. Un broker MQTT gestisce la comunicazione tra i dispositivi, agendo come intermediario tra i client che pubblicano messaggi e quelli che sottoscrivono per riceverli. Questo approccio consente di separare i produttori di dati dai consumatori, migliorando la flessibilità e la scalabilità del sistema.

I broker MQTT si occupano di vari aspetti critici della comunicazione, come la gestione delle connessioni dei client, la distribuzione dei messaggi ai sottoscrittori appropriati e il mantenimento della qualità del servizio (QoS). Utilizzare un broker permette una gestione centralizzata delle comunicazioni, semplificando l'implementazione e la manutenzione delle reti IoT.

Alcuni dei broker MQTT più popolari sono Mosquitto [8], HiveMQ [16] e RabbitMQ [7], ognuno con funzionalità avanzate come la persistenza dei messaggi, la sicurezza e l'autenticazione dei client. L'adozione di un broker è essenziale per garantire che i messaggi siano consegnati in modo affidabile e tempestivo, assicurando una comunicazione fluida e continua tra i dispositivi connessi.

In sintesi, MQTT rappresenta una soluzione potente e scalabile per la comunicazione M2M e IoT, grazie alla sua struttura leggera, efficace gestione della sicurezza e capacità di supportare una vasta gamma di applicazioni distribuite.

2.4 Il microcontrollore ESP32

2.4.1 Panoramica e caratteristiche principali

L'ESP32 è un microcontrollore sviluppato da Espressif Systems, noto per le sue elevate capacità di connettività wireless e le sue prestazioni avanzate. Questo SoC (System-on-Chip) è dotato di un processore dual-core e offre supporto integrato sia per Wi-Fi che per Bluetooth, rendendolo ideale per una vasta gamma di applicazioni IoT.

Oltre alla connettività, l'ESP32 include numerose periferiche integrate, come ADC (Analog-to-Digital Converter), DAC (Digital-to-Analog Converter), interfacce SPI, I2C, UART, PWM e molti GPIO (General Purpose Input/Output), che facilitano l'interazione con sensori, attuatori e altre periferiche.

Grazie alla sua versatilità e alle sue caratteristiche avanzate, l'ESP32 è diventato una scelta popolare tra maker e professionisti per la prototipazione e lo sviluppo di progetti IoT.

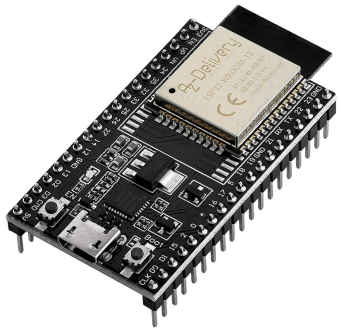

Figura 2.7. Scheda di sviluppo ESP32

2.4.2 Vantaggi dell'uso dell'ESP32

L'ESP32 offre numerosi vantaggi, tra cui un'elevata capacità di integrazione e una riduzione dei costi e della complessità del design hardware. La presenza di connettività Wi-Fi e Bluetooth a bordo elimina la necessità di moduli esterni, rendendo l'ESP32 una soluzione compatta e conveniente. La sua versatilità nelle interfacce di comunicazione consente di collegare facilmente una vasta gamma di sensori e dispositivi, mentre le sue elevate prestazioni e l'efficienza energetica lo rendono adatto per applicazioni critiche in termini di consumo energetico. Inoltre, la comunità di sviluppo attiva e le numerose risorse disponibili online facilitano l'adozione e l'implementazione dell'ESP32 in progetti sia hobbistici che professionali.

L'ESP32 rappresenta una soluzione potente e versatile per l'Internet of Things, offrendo una combinazione unica di connettività, prestazioni e flessibilità che lo rende ideale per una vasta gamma di applicazioni, dalle smart home all'industria 4.0, dalla sanità alle città intelligenti.

2.5 Il firmware Tasmota

2.5.1 Panoramica e storia di Tasmota

Tasmota è un firmware open-source sviluppato per dispositivi basati su ESP8266 e successivamente su ESP32, che permette di controllare e monitorare una vasta gamma di dispositivi IoT. Creato inizialmente da Theo Arends nel gennaio 2016, Tasmota è nato come soluzione per fornire un'alternativa open-source ai firmware proprietari utilizzati nei dispositivi smart home, offrendo maggiore controllo, flessibilità e sicurezza agli utenti.

Il progetto Tasmota è nato come iniziativa per migliorare il firmware originale fornito con le lampadine smart basate su ESP8266. Theo Arends, sviluppatore e fondatore del progetto, ha visto l'opportunità di creare un firmware che non solo sostituisse quello esistente, ma che fosse anche altamente configurabile e supportasse un'ampia varietà di sensori e attuatori. Questo ha portato alla creazione di Tasmota, un firmware che poteva essere facilmente flashato sui dispositivi ESP8266, permettendo agli utenti di controllarli tramite comandi MQTT, HTTP e WebSocket. Con il passare del tempo, Tasmota ha visto una rapida crescita in termini di funzionalità e popolarità. La comunità open-source ha giocato un ruolo cruciale in questo sviluppo, contribuendo con codice, segnalazioni di bug, e documentazione. Il firmware ha iniziato a supportare non solo luci smart, ma anche interruttori, prese di corrente, sensori di temperatura e umidità, e molti altri dispositivi. Grazie alla sua flessibilità e alla vasta gamma di dispositivi supportati, Tasmota è diventato uno dei firmware più popolari per i progetti DIY IoT.

2.5.2 Vantaggi dell'uso di Tasmota

L'uso di Tasmota offre numerosi vantaggi che lo rendono una scelta preferita per molti appassionati di IoT. La sua compatibilità hardware è uno dei principali punti di forza, supportando una vasta gamma di dispositivi basati su ESP8266 e ESP32. Questo permette agli utenti di utilizzare Tasmota con molti dei dispositivi disponibili sul mercato, ampliando così le possibilità di implementazione. I protocolli di comunicazione supportati, come MQTT, HTTP, WebSocket, e mDNS, facilitano l'integrazione con vari ecosistemi smart home, rendendo Tasmota una soluzione versatile e interoperabile. La configurabilità del firmware è un altro grande vantaggio, permettendo un'ampia configurazione tramite interfaccia web, comandi console e file di configurazione JSON. Questo consente agli utenti di adattare il firmware alle proprie esigenze specifiche senza necessità di complesse modifiche al codice sorgente.

Tasmota supporta anche l'implementazione di regole e automazioni locali senza la necessità di server esterni, offrendo una maggiore autonomia e sicurezza operativa. Gli aggiornamenti Over-The-Air (OTA) semplificano il processo di aggiornamento del firmware, riducendo i tempi di inattività e migliorando la gestione del dispositivo.

2.5.3 Caratteristiche tecniche

Le caratteristiche tecniche di Tasmota sono ampie e diversificate, riflettendo la sua natura flessibile e potente. Il firmware è compatibile con un'ampia gamma di dispositivi

basati su ESP8266 e ESP32, offrendo supporto per molteplici sensori e attuatori. Questa compatibilità permette di controllare luci smart, interruttori, prese di corrente, sensori di temperatura e umidità, tra molti altri dispositivi, tutti tramite una singola piattaforma firmware. Tasmota include supporto per diversi protocolli di comunicazione come MQTT, HTTP, WebSocket, e mDNS, che facilitano l'integrazione con altri sistemi e piattaforme IoT. L'interfaccia web di Tasmota consente una facile configurazione e gestione dei dispositivi, offrendo un accesso immediato a tutte le funzionalità principali senza la necessità di strumenti aggiuntivi. I comandi console e i file di configurazione JSON aggiungono un ulteriore livello di configurabilità, permettendo agli utenti più avanzati di personalizzare profondamente il comportamento del firmware. La possibilità di implementare automazioni e regole locali è una caratteristica tecnica importante, che consente di eseguire operazioni complesse e rispondere a eventi senza la necessità di un server esterno. Questo migliora l'affidabilità e la sicurezza dei sistemi IoT basati su Tasmota. Infine, il supporto per gli aggiornamenti Over-The-Air (OTA) rende la manutenzione del firmware semplice e diretta, permettendo agli utenti di aggiornare i propri dispositivi senza doverli rimuovere o ricollegare fisicamente, aumentando così la convenienza e l'efficienza operativa.

Il linguaggio Berry

Una delle caratteristiche più recenti e innovative introdotte in Tasmota è il supporto per il linguaggio di scripting Berry. Berry è un linguaggio di programmazione leggero, progettato specificamente per essere eseguito su dispositivi embedded. L'inclusione di Berry in Tasmota consente agli utenti di scrivere script personalizzati per automatizzare compiti, processare dati dei sensori e implementare logiche di controllo avanzate direttamente sul dispositivo. Berry è progettato per essere semplice da imparare e utilizzare, pur mantenendo una grande flessibilità e potenza espressiva. Questo permette di creare soluzioni IoT su misura senza la necessità di riscrivere o modificare il firmware principale, offrendo un ulteriore livello di personalizzazione e controllo per gli utenti avanzati.

Capitolo 3

Estendere Tasmota

La natura modulare del firmware Tasmota offre agli utenti la possibilità di personalizzarlo e di estenderlo, rendendolo un buon candidato per una vasta gamma di progetti, dalle semplici automazioni amatoriali alle integrazioni con ecosistemi più ampi, come data center o mining farm.

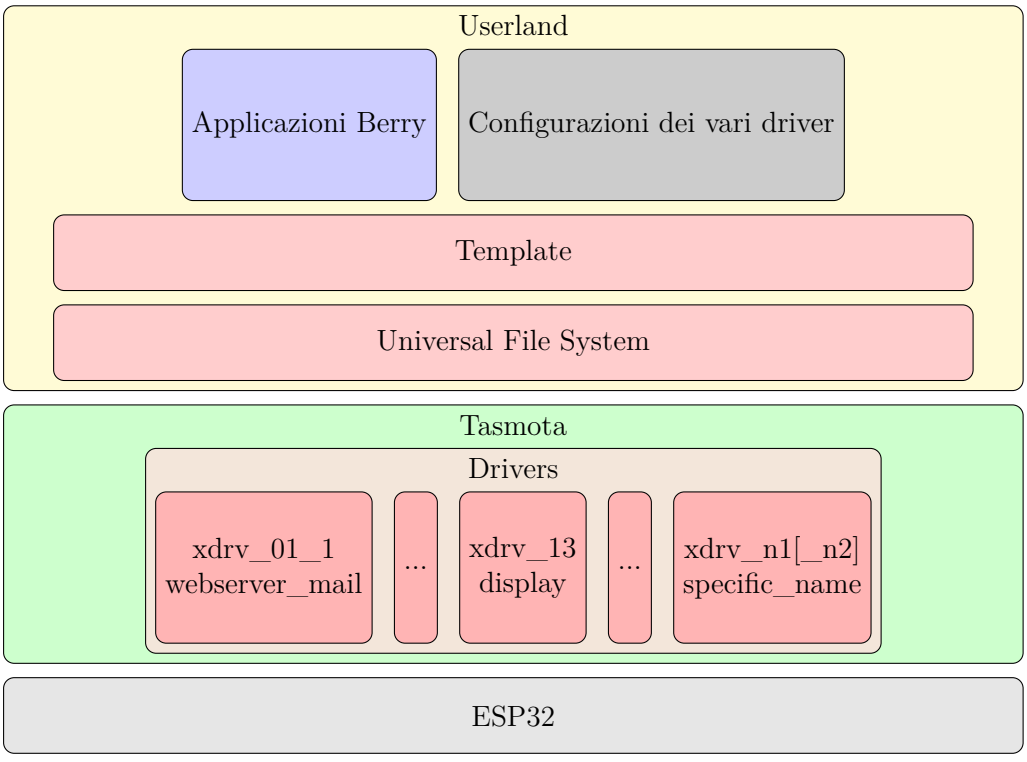

Figura 3.1. Architettura di Tasmota

Nonostante Tasmota, grazie al contributo molto attivo della comunità supporti molti

dispositivi [40], moduli [32] e periferiche [30], dove specifiche esigenze di progetto richiedono soluzioni su misura, si può presentare la necessità di utilizzare periferiche per le quali non sia incluso il supporto.

Nel corso di questo capitolo verrà mostrato come modificare Tasmota per adattarlo alle proprie esigenze, partendo dalle modifiche che l'utente può effettuare tramite le impostazioni disponibili, fino ad arrivare a soluzioni avanzate che richiedono la modifica del codice sorgente.

Inizialmente verrà illustrato come utilizzare i template, che permettono di definire facilmente configurazioni hardware specifiche senza dover modificare il codice.

Successivamente, verrà esplorato l'uso delle applicazioni Berry, un linguaggio di scripting integrato in Tasmota, per la creazione e il caricamento di applicazioni personalizzate.

Un'altra importante funzionalità, oggetto di trattazione, è rappresentata dai file di configurazione, come il file display.ini, che permette di configurare display e touchscreen senza dover ricompilare il firmware. Questo approccio semplifica notevolmente l'integrazione di nuovi hardware, riducendo il bisogno di interventi tecnici complessi.

Per gli utenti avanzati, discuteremo le modalità di modifica diretta del codice di Tasmota, fornendo le informazioni necessarie per identificare i componenti da modificare, gli strumenti richiesti e i passaggi per configurare il debug tramite JTAG.

Inoltre, sarà mostrato un esempio pratico di creazione di un driver personalizzato, illustrando come aggiungere supporto per nuovi dispositivi hardware direttamente nel firmware.

3.1 Lato utente

3.1.1 I template

I template sono oggetti, scritti con la codifica JSON, contenenti informazioni di base del dispositvo e dei pin GPIO di cui dispone, vengono usati per permettere a Tasmota di interfacciarsi correttamente con le periferiche ad esso collegabili.

I template per una vasta gamma di board possono essere scaricati direttamente da Internet, in alternativa, possono essere creati direttamente dall'interfaccia web di Tasmota [33], realizzandoli da zero, partendo dal modulo corrente, oppure da un modulo supportato.

Particolare attenzione va posta al significato dei pin, poiché alcuni valori specifici consentono a Tasmota di determinare se attivare o meno un driver.

Esempio di template

Nella figura 3.2 è mostrato un esempio di template utilizzato per configurare il dispositivo che sarà poi trattato nel capitolo 3.3.

```
{
    "NAME": "ESP32-EINK",
    "GPIO":[1,1,1,1,832,6496,1,1,1,1,1,1,1024,960,896,1,0,0,1,864,0,1,
    ↪  1,1,0,0,0,0,1,1,1,1,1,0,0,1],
    "FLAG": 0,
    "BASE": 1
}
```

Figura 3.2. Esempio di Template

Per caricare un template, è possibile avvalersi dell'interfaccia web, tramite l'apposito campo `Other Paramethers -> Template`, disponibile nel menu `Configuration` dell'interfaccia web.

In alternativa, è possibile effettuare il caricamento tramite console, utilizzando il comando `Template` seguito dalla configurazione del template in formato JSON. In seguito, per applicare il template e riavviare il dispositivo con le nuove impostazioni, è necessario invocare il comando `Module 0`.

3.1.2 Le applicazioni Berry

Berry è un linguaggio di scripting molto leggero, che supporta più paradigmi di programmazione, progettato specificamente per sistemi embedded. L'interprete di Berry è interamente scritto in C99, garantendo semplicità, velocità e un'ottimizzazione efficace dell'uso della memoria RAM [4].

Questo linguaggio è particolarmente utile per lo sviluppo rapido di applicazioni grazie alla sua natura di linguaggio di scripting. Tuttavia, come per qualsiasi linguaggio di alto livello, presenta un livello di astrazione che può introdurre complessità e potenziali problemi in ambienti che richiedono la massima affidabilità. In questi casi, lo sviluppo in C rimane preferibile, offrendo un controllo più diretto e dettagliato sull'hardware e la gestione delle risorse di sistema.

Dal 2021, Tasmota include un porting di Berry che comprende anche alcune estensioni per interfacciarsi con il firmware [4].

All'interno di Tasmota, Berry può essere utilizzato per vari scopi: dalla creazione di semplici script all'aggiunta di driver Tasmota completi.

Inoltre, è possibile utilizzare Berry per integrare librerie native, come LVGL [18], ampliando così le capacità e le funzionalità del firmware.

Caricare un'applicazione in formato Tapp

Le applicazioni scritte con Berry sono salvate in file con estensione .tapp. Questi file sono conservati come archivi zip tradizionali, ma senza compressione, un concetto molto simile agli archivi tar in Linux.

```
1 $ file Example.tapp
2     Example.tapp: Zip archive data, at least v1.0 to extract,
      ↪  compression method=store
```

Figura 3.3. Output del comando `file` eseguito su `Example.tapp`

Le applicazioni .tapp presenti nella scheda vengono caricate automaticamente da Tasmota all'avvio. Per aggiungerne di nuove, è necessario caricarle nella radice del filesystem. Esistono varie strategie per eseguire questa operazione, tra cui l'utilizzo dell'interfaccia web. Tramite questa modalità, è possibile caricare direttamente i file .tapp nella radice del filesystem in modo semplice e immediato.

Successivamente, dopo aver riavviato la scheda, sarà possibile invocare i comandi che le applicazioni mettono a disposizione dell'utente.

A titolo di esempio, si segnala che anche l'interfaccia web inclusa in Tasmota per il caricamento dei file all'interno dell'Universal File System è scritta utilizzando Berry [28]. Infatti, le versioni di Tasmota compilate per schede ESP82xx, per le quali non esiste il supporto a Berry, non includono tale applicazione.

HelloBerry

In questa sezione verrà illustrato un esempio di applicazione .tapp scritta in Berry, che aggiunge al firmware un comando denominato `'Example'` per stampare due messaggi, uno sulla console e uno nei log.

Ogni applicazione .tapp deve necessariamente includere un file denominato `autoexec.be`, che viene eseguito automaticamente al momento del caricamento come punto di partenza, da cui possono essere richiamati eventuali altri file e/o librerie.

```
HelloBerry.tapp
└─ autoexec.be
```

Figura 3.4. Contenuto dell'applicazione di esempio HelloBerry.tapp

La figura 3.4 mostra la struttura del contenuto dell'archivio `HelloBerry.tapp`. Il file `autoexec.be` contiene il codice principale dell'applicazione.

```
# Register the command 'Example'

tasmota.add_cmd("Example",
    def ()
        print('Hello from Berry Example App!')
        log('This is a log message!')
        tasmota.resp_cmnd_done()
    end
)
```

Figura 3.5. File autoexec.be dell'applicazione di esempio HelloBerry.tapp

Per creare l'archivio .tapp senza compressione, può essere utilizzato il comando zip, disponibile su molti sistemi operativi [15].

```
$ zip -0 -r HelloBerry.tapp autoexec.be
  adding: autoexec.be (stored 0%)
```

Figura 3.6. Comando zip per creare l'archivio .tapp

Una volta caricato l'archivio .tapp e riavviata la scheda, il log di Tasmota mostra che l'applicazione è stata caricata con successo.

Il log fornisce informazioni dettagliate sul processo di caricamento e sull'esecuzione del file `autoexec.be`, come si può vedere nella figura 3.7.

Questo è un passaggio importante per verificare che l'applicazione sia stata installata correttamente e che il nuovo comando `'Example'` sia operativo.

```
1  00:00:00.002 HDW: ESP32-D0WD-V3 v3.0
2  00:00:00.064 UFS: FlashFS mounted with 304 kB free
3  00:00:00.096 CFG: Loaded from File, Count 15
4  00:00:00.106 QPC: Count 1
5  00:00:00.208 BRY: Berry initialized, RAM used 4446 bytes
6  00:00:00.230 Project tasmota - Tasmota Version
   ↪ 13.4.0.1(tasmota32)-2_0_14(2024-05-29T18:11:32)
7  00:00:00.272 TAP: Loaded Tasmota App 'HelloBerry.tapp'
8  00:00:02.001 WIF: Connecting to AP1 dlink Channel 11 BSSId
   ↪ 6A:88:12:54:AA:03 in mode 11n as tasmota-CDA098-0152...
9  00:00:03.929 WIF: Connected
10 00:00:04.141 HTP: Web server active on tasmota-CDA098-0152 with IP
   ↪ address 192.168.178.193
11 17:38:19.132 RSL: INFO1 = {"Info1":{"Module":"ESP32-DevKit","Version":
   ↪ "13.4.0.1(tasmota32)","FallbackTopic":"cmnd/DVES_CDA098_fb/","Grou
   ↪ pTopic":"cmnd/tasmotas/"}}
12 17:38:19.147 RSL: INFO2 = {"Info2":{"WebServerMode":"Admin","Hostname"
   ↪ :"tasmota-CDA098-0152","IPAddress":"192.168.178.193","IP6Global":"
   ↪ ","IP6Local":"fe80::c249:efff:fecd:a098%st1"}}
13 17:38:19.160 RSL: INFO3 = {"Info3":{"RestartReason":"Vbat power on
   ↪ reset","BootCount":6}}
14 17:38:21.929 QPC: Reset
15 17:38:23.918 RSL: STATE =
   ↪ {"Time":"2024-05-29T17:38:23","Uptime":"0T00:00:09","UptimeSec":9,
   ↪ "Heap":152,"SleepMode":"Dynamic","Sleep":50,"LoadAvg":19,"MqttCoun
   ↪ t":0,"Berry":{"HeapUsed":4,"Objects":54},"Wifi":{"AP":1,"SSId":"dl
   ↪ ink","BSSId":"6A:88:12:54:AA:03","Channel":11,"Mode":"11n","RSSI":
   ↪ 52,"Signal":-74,"LinkCount":1,"Downtime":"0T00:00:04"}}
16 17:38:58.687 CMD: Example
17 17:38:58.693 Hello from Berry Example App!
18 17:38:58.694 This is a log message!
19 17:38:58.696 RSL: RESULT = {"Example":"Done"}
```

Figura 3.7. File autoexec.be dell'applicazione di esempio HelloBerry.tapp

Come evidenziato nella riga 7 e nelle righe da 16 a 19, il contenuto del file autoexec.be è stato eseguito e il nuovo comando `"Example"` è ora disponibile.

3.1.3 File di configurazione

Driver configurabili

Universal Display (uDisplay) driver e Universal Touch (uTouch) driver A partire dalla versione di Tasmota 13.4, il team di sviluppo ha scelto di abbandonare gradualmente l'utilizzo dei driver specifici per gli schermi touchscreen e per i display, in favore del nuovo Universal Display Driver [37]

L'uDisplay driver semplifica di molto la vita agli utenti, in quanto non devono manualmente ricompilare Tasmota per poter utilizzare lo schermo, ma gli basta solamente caricare nella radice dell'universal file system della propria board un file demoninato display.ini, contenente tutte le specifiche del loro schermo.

Le immagini pre compilate di tasmota tasmota-display.bin.[1], tasmota32-display.bin e tasmota32-lvgl.bin contengono già al loro interno il driver universale [37]

```
:H,E-PAPER-42,400,300,1,SPI,4,*,*,*,*,*,*,*,10
:S,1,1,1,0,10,10
:B,60,8
:I
01,5,03,00,2b,2b,ff
06,3,17,17,17
04,80
00,1,3F
30,1,3C
61,4,01,90,01,2C
82,1,12
50,1,97
:a,10,13,12
:T,450,10,450
```

Figura 3.8. Esempio di intestazione del file display.ini "WS_epaper42_display.ini" [34]

Di seguito è riportata una spiegazione dettagliata delle parti più importanti del descriptor preso in esempio nelle figure 3.8 e 3.9.

L'header (sezione :H) deve contenere il nome del display, la risoluzione in pixel specificata come larghezza (x) e altezza (y), il numero di colori per pixel in bit, in questo caso 1, trattandosi di display in bianco e nero, e l'interfaccia utilizzata (I2C o SPI).

In base all'interfaccia selezionata, seguono varie specifiche.

[1] Utilizzando questa versione, realizzata per l'ESP82xx, il file display.ini deve essere caricato utilizzano metodi alternativi [38]. Tali schede non supportano l'upload tramite interfaccia web

La splash screen :S definisce la configurazione dello splash screen e i colori iniziali del display. Se omessa, lo schermo non viene pulito inizialmente. I parametri sono: numero del font, dimensione del font, colore del testo (indice), colore di sfondo (indice), posizione x del testo, e posizione y del testo.

L'opzione :B è utilizzata per configurare la modalità di aggiornamento dello schermo. Il primo parametro indica il numero di linee del display da aggiornare contemporaneamente, in questo caso 60. Il secondo parametro, svolge da maschera per abilitare varie funzioni diverse, il valore 8 indica di invertire il segnale di busy (molto comune per display epaper).

La configurazione iniziale (sezione :I) del registro del controller del display, tutti i valori sono in esadecimale.

In questo caso, utilizzando l'interfaccia SPI, il primo valore è il comando, seguito dal numero di argomenti e dagli argomenti stessi.

L'opzione :a è specifica per i display e-ink e non è documentata nella wiki. Questa opzione definisce i comandi SPI da inviare ai display e-ink per eseguire determinate operazioni, i parametri che accetta sono gli opcode per le seguenti operazioni i seguenti:

- `DATA_START_TRANSMISSION_1` (0x10 nell'esempio)
- `DATA_START_TRANSMISSION_2` (0x12 nell'esempio)
- `DISPLAY_REFRESH` (0x13 nell'esempio)

Tempi di attesa (sezione :T) utilizzati per i display e-ink, includendo l'attesa per l'aggiornamento completo in millisecondi, l'attesa per l'aggiornamento parziale e l'attesa dopo l'aggiornamento.

Tabelle di lookup (sezioni :Lx) utilizzate da alcuni display epaper, come quello di esempio, che richiede tabelle di lookup per specificare i parametri necessari per l'aggiornamento completo del display. La notazione da usare è `:Lx,size,OP`, seguita dai byte che compongono la tabella.

Dove `x` è un numero che identifica la tabella specifica, `size`: indica la dimensione in byte della tabella di lookup. `OP`: È l'opcode per inviare la tabella di aggiornamento al display.

L'opcode è un comando che indica al display come utilizzare i dati della tabella di lookup.

Approfondimenti Esistono numerose altre notazioni utilizzabili nel descriptor per configurare e personalizzare il comportamento dei display.

Per una descrizione completa e aggiornata di queste notazioni, si consiglia di consultare la wiki ufficiale del progetto [37], dove è possibile trovare una guida esaustiva e dettagliata sulle varie opzioni disponibili e sul loro utilizzo ottimale.

```
:L1,44,20
40,17,00,00,00,02
00,17,17,00,00,02
00,0A,01,00,00,01
00,0E,0E,00,00,02
00,00,00,00,00,00
00,00,00,00,00,00
00,00,00,00,00,00,00,00
:L2,42,21
40,17,00,00,00,02
90,17,17,00,00,02
40,0A,01,00,00,01
A0,0E,0E,00,00,02
00,00,00,00,00,00
00,00,00,00,00,00
00,00,00,00,00,00
:L3,42,22
40,17,00,00,00,02
90,17,17,00,00,02
40,0A,01,00,00,01
A0,0E,0E,00,00,02
00,00,00,00,00,00
00,00,00,00,00,00
00,00,00,00,00,00
:L4,42,23
80,17,00,00,00,02
90,17,17,00,00,02
80,0A,01,00,00,01
50,0E,0E,00,00,02
00,00,00,00,00,00
00,00,00,00,00,00
00,00,00,00,00,00
:L5,42,24
80,17,00,00,00,02
90,17,17,00,00,02
80,0A,01,00,00,01
50,0E,0E,00,00,02
00,00,00,00,00,00
00,00,00,00,00,00
00,00,00,00,00,00
#
```

Figura 3.9. LUT del file "WS_epaper42_display.ini" [34]

3.1.4 Integrazioni esterne

Tramite il protocollo MQTT, è possibile integrare Tasmota con componenti esterne, sia per la ricezione, che per l'invio di dati e comandi, tutto tramite messaggi MQTT.

Di default, Tasmota utilizza tre topic predefiniti [35] (con la dicitura `<DeviceTopic>` si intende il nome del dispositivo configurato nel firmware)

Topic di comando (`cmnd/<DeviceTopic>/<Command>`) Usato per inviare il comando `<Command>` con i parametri presenti nel body del messaggio. Ad esempio, il comando POWER può essere usato per accendere o spegnere il dispositivo. Un'altro caso d'uso, è la stampa su schermo, che può essere effettuata con il comando displaytext.

Inviando un messaggio MQTT al topic `cmnd/<DeviceTopic>/DISPLAYTEXT` sarà possibile stampare a schermo il contenuto del body del messaggio.

Topic di risposta (`stat/<DeviceTopic>/<Type>`) Usato per ottenere informazioni sullo stato del dispositivo, ad esempio il <Type> "RESULT" conterrà il risultato dell'ultimo comando inviato.

Un esempio di risultato per l'esecuzione del comando DISPLAYTEXT, precedentemente citato, è mostrato in figura 3.10.

```
1 {"DisplayText":"Prova stampa da MQTT"}
```

Figura 3.10. Output del comando displaytext inviato con MQTT, ottenuto dal topic di risposta

Topic di telemetria (`tele/<DeviceTopic>/<Type>`) I topic di telemetria, invece, sono utilizzati per inviare aggiornamenti periodici sullo stato del dispositivo e altri dati dei sensori. Ad esempio, il topic `tele/<DeviceTopic/INFO2` contiene alcune informazioni sulla rete 3.11.

```
1 {
2     "Info2": {
3         "WebServerMode": "Admin",
4         "Hostname": "tasmota-13F088-4232",
5         "IPAddress": "192.168.178.197",
6         "IP6Global": "",
7         "IP6Local": "fe80::96b5:55ff:fe13:f088%st1"
8     }
9 }
```

Figura 3.11. Messaggio di esempio per il topic `tele/<DeviceTopic>/INFO2`

Un caso concreto potrebbe essere l'utilizzo di Telegram per interagire con Tasmota. Esistono client MQTT in grado di interagire con Telegram e inviare messaggi agli utenti in caso di eventi pubblicati sul topic scelto, oppure di mandare comandi a Tasmota partendo da messaggi inviati dai client MQTT sul topic.

3.1.5 Rules

Tasmota mette a disposizione dell'utente una funzione chiamata "Rules" che permette di costruire automazioni senza la necessità di codice dedicato.

Queste regole permettono di rispondere a eventi specifici, modificare configurazioni e inviare comandi, tutto internamente al dispositivo senza la necessità di un server esterno.

Rappresentano quindi un valido strumento per operazioni che non necessitano grandi potenze di calcolo.

Una scenario di applicazione per le Rules, potrebbe essere la creazione di una regola che azioni un sistema di raffreddamento se la temperatura rilevata da uno specifico sensore sia superiore ad una determinata soglia.

Le regole di Tasmota sono costituite da tre parti principali: evento, condizione opzionale e azione.

L'evento è ciò che scatena la regola, come la pressione di un pulsante o l'attivazione di un interruttore. La condizione è un criterio che deve essere soddisfatto affinché l'azione venga eseguita. L'azione è il comando o la serie di comandi da eseguire in risposta all'evento.

Per creare una regola, si utilizza la sintassi 3.12

```
1 Rule<x> on <evento> do <azione> endon
```

Figura 3.12. Sintassi di esempio Tasmota rules

Dove `<x>` indica la ruleset, sono possibili valori tra 1 e 3 (ciascuna ruleset può contenere molte regole), `<evento>` è il trigger della regola e `<azione>` è ciò che verrà eseguito.

Ad esempio, per accendere il relè 1 quando il pulsante 1 viene premuto si userà la regola 3.13.

```
1 Rule1 on Button1#State=2 do Power1 ON endon
```

Figura 3.13. Esempio di regola

Gli eventi possono variare a seconda dei dispositivi e dei sensori collegati. Alcuni esempi comuni includono `"Button1#State=2"` per un pulsante premuto, `"Switch1#State=1"` per un interruttore attivato e `"Time#Minute=0"` per l'inizio di un'ora. Le azioni possono essere comandi Tasmota come accendere o spegnere un dispositivo, inviare messaggi MQTT o cambiare configurazioni.

Tasmota permette l'uso di variabili di memoria per memorizzare valori temporanei e utilizzarli nelle regole. Le variabili di memoria sono definite come `"%mem<x>%"` e possono essere usate per creare condizioni più complesse.

Ad esempio, una regola potrebbe verificare se una variabile di memoria ha un determinato valore prima di eseguire un'azione.

Le regole possono essere abilitate o disabilitate tramite comandi. Ad esempio, `"Rule1 1"` abilita la regola 1, mentre `"Rule1 0"` la disabilita. Questa flessibilità permette di attivare e disattivare le regole a seconda delle necessità.

Le regole di Tasmota offrono un modo flessibile e potente per automatizzare i dispositivi IoT senza bisogno di server esterni. Comprendere come creare eventi, azioni e condizioni consente di sfruttare al meglio le potenzialità di Tasmota per rispondere a vari scenari e requisiti di automazione.

3.2 Modificare il codice di Tasmota

La community di Tasmota è molto vasta, per questo motivo la maggior parte dei casi d'uso sono coperte da soluzioni già sviluppate da altri programmatori. Tuttavia, in alcune circostanze, può essere necessario modificare modificare Tasmota per adattarlo alle proprie esigenze, ad esempio in caso di sviluppo di schede su misura per progetti specifici.

3.2.1 Identificare il componente da modificare

Il punto di partenza per modificare il firmware è l'identificazione del modulo responsabile del comportamento che si intende modificare o estendere.

Per determinare la componente interessata esistono varie strategie: Una delle tecniche più semplici consiste nell'inserire nel codice dei messaggi di log, per monitorare il flusso di esecuzione e lo stato delle variabili, utilizzando funzioni come `Serial.print()`.

Tuttavia, questo approccio richiede una compilazione per ogni test che si desidera effettuare. Considerando che la dimensione della memoria flash dell'ESP32 è di 4MB [9] e che l'immagine di `tasmota32` occupa approssimativamente qualche megabyte, a seconda della versione in uso [31], il caricamento di Tasmota richiede un certo tempo, poiché la velocità della porta seriale, all'atto pratico, si aggira intorno a 500 kbit/s (0,06 MB/s) 3.14. Di conseguenza, tale metodo risulta poco pratico.

```
1 Wrote 1366096 bytes (942624 compressed) at 0x000e0000 in 21.6 seconds
  ↪  (effective 505.1 kbit/s)...
```

Figura 3.14. Porzione di log dell'invocazione di `esptool.py` per il caricamento di Tasmota tramite porta seriale

Un'alternativa consiste nell'utilizzare strumenti hardware appositi, come l'ESP-Prog 3.15, per interfacciarsi con l'interfaccia JTAG dell'ESP32, consentendo l'aggiunta di breakpoint e watchpoint per fermare l'esecuzione del programma e monitorare le variabili senza necessità di ricompilazione.

3.2.2 Strumenti necessari

Strumenti Hardware

È necessario disporre di un debugger hardware compatibile con JTAG, come l'ESP-Prog 3.15 o il Segger J-Link 3.16, insieme a cavi e connettori specifici, come fili Dupont, per connettere il debugger alla scheda ESP32. Inoltre, un adattatore da porta USB a seriale è indispensabile per monitorare l'output e inviare comandi all'ESP32.

Molte schede di sviluppo ESP32 includono già un adattatore integrato, ma è possibile utilizzare anche adattatori esterni. L'ESP-Prog 3.15, ad esempio, integra un adattatore

che fornisce sia l'interfaccia JTAG sia quella seriale, richiedendo il collegamento di un solo cavo.

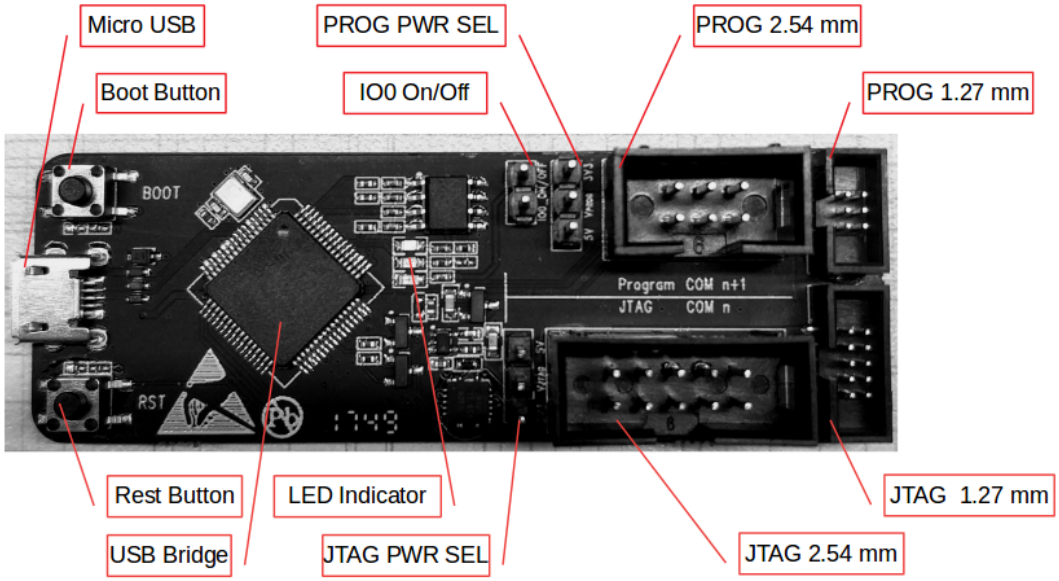

Figura 3.15. ESP-Prog [10]

Figura 3.16. Segger J-Link [24]

IDE di Sviluppo

Sono disponibili diversi ambienti di sviluppo per la programmazione dell'ESP32. L'Arduino IDE è spresso preferito per la sua semplicità e il supporto integrato per ESP32.

Eclipse, con l'add-on Espressif IDF plugin, offre un supporto completo di debug.

Un'altra alternativa popolare è Visual Studio Code (VS Code) [20], il quale, tramite l'estensione PlatformIO [23], fornisce un'eccellente esperienza di sviluppo e debugging. Tale configurazione include anche un monitor seriale integrato, il che permette di visualizzare direttamente l'output seriale nell'IDE, semplificando ulteriormente il processo di debug e l'interazione con il dispositivo.

La repository di Tasmota è predisposta per l'utilizzo di PlatformIO [39], come si evince anche dalle figure 3.18 e 3.17, con numerosi ambienti già configurati per le varie immagini disponibili.

```
[env:tasmota32_base]
framework               = ${common.framework}
platform                = ${core32.platform}
platform_packages       = ${core32.platform_packages}
board_build.filesystem  = ${common.board_build.filesystem}
custom_unpack_dir       = ${common.custom_unpack_dir}
board_build.variants_dir = ${common.board_build.variants_dir}
board                   = esp32
monitor_speed           = ${common.monitor_speed}
monitor_echo            = ${common.monitor_echo}
upload_resetmethod      = ${common.upload_resetmethod}
extra_scripts           = ${esp32_defaults.extra_scripts}
build_unflags           = ${core32.build_unflags}
build_flags             = ${core32.build_flags}
lib_ldf_mode            = ${common.lib_ldf_mode}
lib_compat_mode         = ${common.lib_compat_mode}
lib_extra_dirs          = ${common.lib_extra_dirs}
                          lib/libesp32
                          lib/libesp32_lvgl
                          lib/libesp32_audio
lib_ignore              = ${esp32_defaults.lib_ignore}
; Add files to Filesystem for all env (global). Remove no files entry
↪  and add add a line with the file to include
; Example for adding the Partition Manager
; custom_files_upload =
; tasmota/berry/modules/Partition_Manager.tapp
custom_files_upload     = no_files
```

Figura 3.17. Esempio di environment dal file `platformio_tasmota_env32.ini`

```
<Codebase Tasmota>
├── API.md
├── CODE_OF_CONDUCT.md
├── CODE_OWNERS.md
├── CONTRIBUTING.md
├── Doxyfile
├── FIRMWARE.md
├── I2CDEVICES.md
├── LICENSE.txt
├── MODULES.md
├── README.md
├── RELEASENOTES.md
├── SECURITY.md
├── TEMPLATES-PRE9.md
├── TEMPLATES.md
├── platformio.ini
├── platformio_override_sample.ini
├── platformio_tasmota32.ini
├── platformio_tasmota_cenv_sample.ini
├── platformio_tasmota_core3_env_sample.ini
├── platformio_tasmota_env.ini
├── platformio_tasmota_env32.ini
├── BUILDS.md
├── CHANGELOG.md
├── platformio_tasmota_cenv.ini
├── platformio_tasmota_core3_env.ini
├── api
├── boards
├── include
├── info
├── lib
├── partitions
├── pio-tools
├── tasmota
└── ...
```

Figura 3.18. Codebase di Tasmota

3.2.3 Passaggi per Configurare il Debug JTAG

La connessione hardware per il debugging dell'ESP32 tramite JTAG richiede il collegamento del debugger JTAG ai relativi pin dell'ESP32. In particolare, è necessario collegare i pin `TDI, TDO, TCK, TMS, GND e VCC`.

Per utilizzare anche il monitor seriale integrato nell'ESP-Prog è necessario collegare i pin `TX, RX, IO0, EN e GND`.

È fondamentale verificare che tutti i cavi siano collegati correttamente, rispettando il pinout sia dell'ESP32 sia del debugger, per garantire una connessione stabile e funzionante. Nella figura 3.19, realizzata con il software open source fritzing [14], è mostrato il cablaggio necessario per poter connettere alla scheda di sviluppo sia l'interfaccia USB-UART sia l'interfaccia JTAG di ESP-Prog.

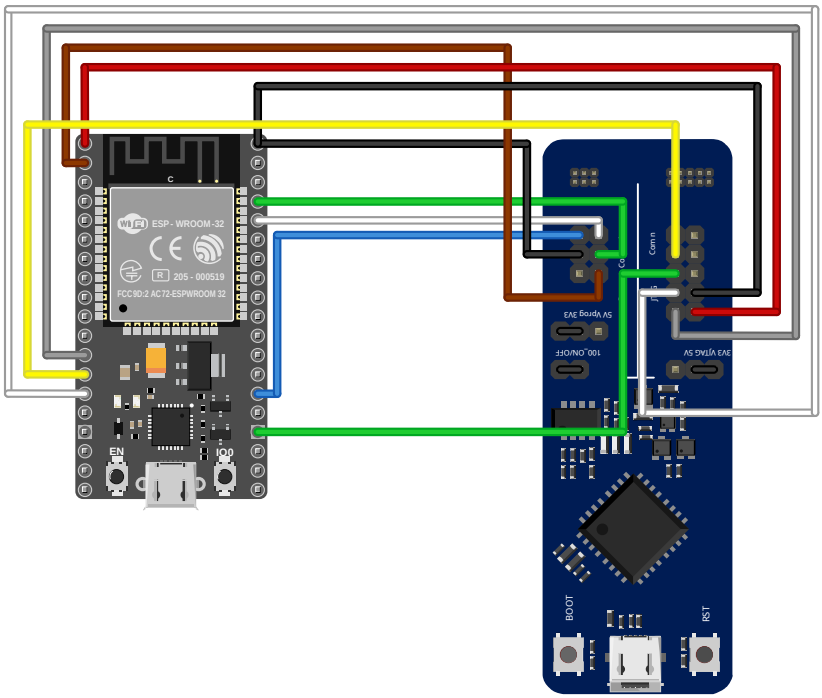

Figura 3.19. Cablaggio necessario per usare interfaccia seriale e JTAG dell'ESP-Prog

Configurazione Software

Per effettuare il debugging dell'ESP32 tramite interfaccia JTAG, con lo scopo di analizzare il programma durante l'esecuzione, sono coinvolti più componenti software, vedi figura 3.20.

In primo luogo, per interfacciarsi con l'adattatore, è richiesta l'installazione di OpenOCD [22], uno strumento open-source che facilita la comunicazione tra il debugger e il microcontrollore.

Infine, è necessario un debugger come ESP GDB per eseguire il debug effettivo del codice.

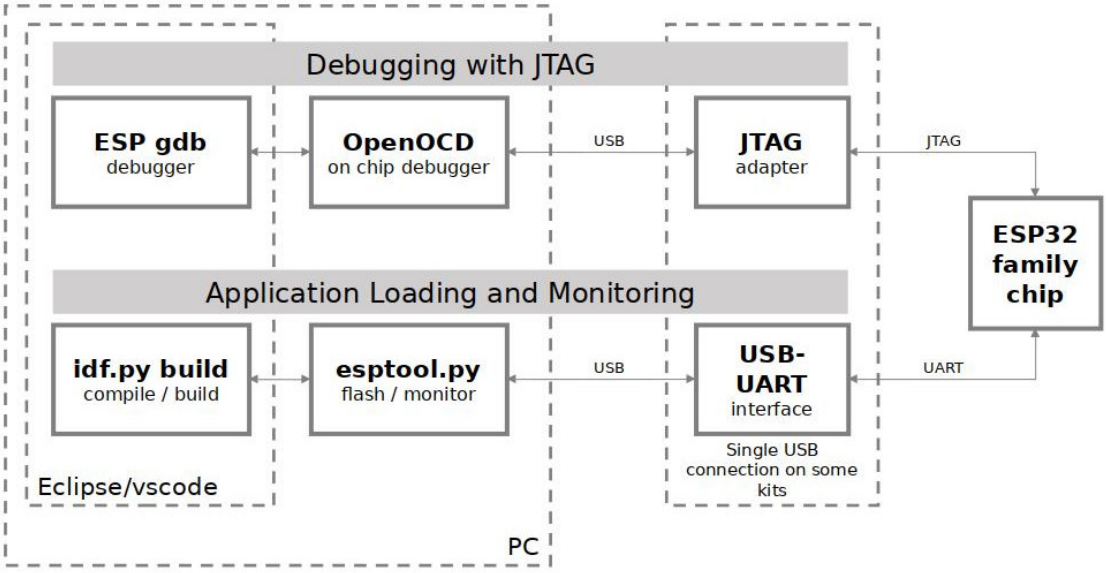

Figura 3.20. Schema generale interfacciamento all'ESP32 [11]

PlatformIO si occupa di configurare tutte queste componenti, semplificando il processo di setup e integrazione.

Inoltre, VSCode ha un supporto integrato per il debugger, rendendo l'esperienza di debugging più fluida e intuitiva per gli sviluppatori.

L'unico passaggio necessario è creare un file chiamato `platformio_override.ini` nella radice della repository di Tasmota.

Questo file permette di sovrascrivere la configurazione predefinita di PlatformIO con parametri specifici per il nostro sistema.

Nel caso trattato, il file dovrebbe contenere un environment contenente le configurazioni mostrate nella figura 3.21.

```
1 [env:tasmota32-display]
2 upload_protocol = esp-prog
3 debug_tool = esp-prog
4 debug_init_break = tbreak setup
```

Figura 3.21. Esempio di platformio_override.ini

Il significato delle varie direttive è il seguente:

- `upload_protocol = esp-prog`: specifica che il protocollo di caricamento utilizzato è `esp-prog`, l'adattatore hardware per JTAG.

- `debug_tool = esp-prog`: indica che l'adattatore `esp-prog` sarà utilizzato come strumento di debug.
- `debug_init_break = tbreak setup`: imposta un punto di interruzione temporaneo all'inizio della funzione `setup` (punto di partenza per tutti i progetti arduino), permettendo al debugger di interrompere l'esecuzione del programma in quel punto iniziale.

Questi parametri garantiscono che PlatformIO configuri correttamente il processo di caricamento e debug per l'ESP32.

La sessione di debug, in VS Code, si potrà invocare premendo F5 o selezionando `Run -> Start Debugging`.

3.3 Driver display e-ink

Attualmente, soltanto pochi display e-ink del marchio Waveshare sono supportati nativamente da Tasmota. Ad esempio, lo schermo Waveshare da 7.5" illustrato nella figura 3.22 non è supportato.

Pertanto, è necessario effettuare un porting completo di una delle librerie esistenti [2] per ESP-32 al fine di rendere questo schermo compatibile con il firmware Tasmota.

Figura 3.22. Waveshare 7.5 display [44]

In questa sezione verranno illustrati i concetti necessari per la creazione di un driver, includendo un esempio di porting della libreria GxEPD2 [46], selezionata per il suo ampio supporto di schermi e per essere la più aggiornata.

[2] Alcune delle librerie più popolari sono GxEPD [45], GxEPD2 [46], EPD_Libraries [6] e epd-library-arduino [27]

3.3.1 La struttura della codebase

Nella repository (figura 3.18), all'interno della cartella `tasmota` (figura 3.23) è presente il nucleo del codice di Tasmota.

Il file del progetto principale è `tasmota.ino`, il quale si occupa di caricare i vari driver, situati nella cartella `tasmota_xdrv_driver`. Ogni driver include le sue dipendenze necessarie, in base a delle direttive al preprocessore. Le direttive predefinite sono specificate nel file `my_user_config.h` e possono essere sovrascritte creando un file denominato `user_config_override.h`.

```
<Codebase Tasmota>
└── tasmota
    ├── my_user_config.h
    ├── tasmota.ino
    ├── user_config_override_sample.h
    ├── berry
    ├── displaydesc
    ├── energy_modbus_configs
    ├── html_compressed
    ├── html_uncompressed
    ├── include
    ├── language
    ├── lvgl_berry
    ├── tasmota_support
    ├── tasmota_xdrv_driver
    │   ├── xdrv_01_1_webserver_mail.ino
    │   ├── ...
    │   ├── xdrv_13_display.ino
    │   ├── ...
    │   └── xdrv_54_lvgl.ino
    ├── tasmota_xdsp_display
    │   ├── xdsp_01_lcd.ino
    │   ├── ...
    │   ├── xdsp_17_universal.ino
    │   ├── ...
    │   └── xdsp_21_waveshare.ino
    ├── tasmota_xlgt_light
    ├── tasmota_xnrg_energy
    ├── tasmota_xsns_sensor
    └── tasmota_xx2c_global
        ├── xdrv_interface.ino
        └── xdsp_interface.ino
```

Figura 3.23. Codebase di Tasmota con i punti salienti per modificare driver

3.3.2 File soggetti a modifiche per aggiungere un driver grafico

Il punto di partenza per creare un driver grafico a Tasmota, è la modifica del file sorgente `tasmota/tasmota_xx2c_global/xdsp_interface.ino`. Tale file contiene i puntatori alle funzioni dei vari driver grafici che è possibile richiamare.

Ogni driver possiede una funzione centrale che gestisce tutte le operazioni specifiche del driver stesso. Questa funzione centrale, chiamata XdspN (dove N è un numero identificativo del driver), riceve un parametro che indica quale operazione eseguire (vedi 3.27). Nella figura 3.24 è mostrata la funzione che si occupa di effettuare le chiamate.

Il driver trattato in questa sezione utilizzerà il numero 21.

```
1  #ifdef USE_DISPLAY
2
3  #ifdef XFUNC_PTR_IN_ROM
4  bool (* const xdsp_func_ptr[])(uint32_t) PROGMEM = {   // Display
   ↪  Function Pointers
5  #else
6  bool (* const xdsp_func_ptr[])(uint32_t) = {   // Display Function
   ↪  Pointers
7  #endif
8
9  #ifdef XDSP_01
10   &Xdsp01,
11 #endif
12
13 ...
14
15 #ifdef XDSP_21
16   &Xdsp21,
17 #endif
18
19 ...
20
21 };
```

Figura 3.24. Prima parte del file `tasmota_xx2c_global/xdsp_interface.ino`

Come mostrato nelle righe 15, 16 e 17 della figura 3.24, è necessario aggiungere una direttiva per il preprocessore, che andrà ad aggiungere la display function `&Xdsp21`, che sarà poi implementata nel nuovo file `tasmota/tasmota_xdsp_display/xdsp_21_waveshare.ino`

```
22  const uint8_t xdsp_present = sizeof(xdsp_func_ptr) /
    ↪  sizeof(xdsp_func_ptr[0]);  // Number of drivers found
23  uint8_t XdspPresent(void) {
24    return xdsp_present;
25  }
26  bool XdspCall(uint32_t function) {
27    bool result = false;
28    DEBUG_TRACE_LOG(PSTR("DSP: %d"), function);
29    for (uint32_t x = 0; x < xdsp_present; x++) {
30      result = xdsp_func_ptr[x](function);
31      if (result && (FUNC_DISPLAY_MODEL == function)) {
32        break;
33      }
34    }
35    return result;
36  }
37  #endif  // USE_DISPLAY
```

Figura 3.25. Seconda parte del file `tasmota_xx2c_global/xdsp_interface.ino`

Nella figura 3.26 è mostrato uno scheletro di driver grafico, il cui scopo è stampare nella console un testo di esempio, per testare il corretto caricamento del driver.

```
1   #ifdef USE_SPI
2   #ifdef USE_DISPLAY
3   #ifdef USE_DISPLAY_WAVESHARE
4
5   #define XDSP_21     21
6
7   bool Xdsp21(uint32_t function)
8   {
9       Serial.println("Custom Waveshare is loaded");
10      bool result = false;
11      return result;
12  }
13
14  #endif  // USE_DISPLAY_WAVESHARE
15  #endif  // USE_DISPLAY
16  #endif  // USE_SPI
```

Figura 3.26. Scheletro di display driver (`xdsp_21_waveshare.ino`)

Per garantire una gestione appropriata del driver, è consigliabile creare un nuovo tipo di pin CS specifico per esso. Questa pratica consente di aggiungere una condizione all'interno della funzione del driver, per permettere all'utente di attivare o disattivare il driver stesso. Nel caso specifico di questo driver è stato aggiunto il pin `Waveshare CS`.

I file coinvolti da questa modifica sono:

- `tasmota/berry/include/be_gpio_defines.h`
- `tasmota/include/tasmota_template.h`
- `tasmota/tasmota_support/support_tasmota.ino`
- `tasmota/tools/lv_gpio/lv_gpio_enum.h`
- I vari file di traduzione per le varie lingue, all'interno della cartella `tasmota/language`

Le possibli chiamate al driver appena creato, che verranno effettuate dalla funzione XdspCall (figura 3.25), sono definite dall'enum `XdspFunctions`, dichiarato all'interno del file `tasmota/tasmota_xdrv_driver/xdrv_13_display.ino`

```
1  enum XdspFunctions { FUNC_DISPLAY_INIT_DRIVER, FUNC_DISPLAY_INIT,
   ↪  FUNC_DISPLAY_EVERY_50_MSECOND, FUNC_DISPLAY_EVERY_SECOND,
2                       FUNC_DISPLAY_MODEL, FUNC_DISPLAY_MODE,
                        ↪  FUNC_DISPLAY_POWER,
3                       FUNC_DISPLAY_CLEAR, FUNC_DISPLAY_DRAW_FRAME,
4                       FUNC_DISPLAY_DRAW_HLINE, FUNC_DISPLAY_DRAW_VLINE,
                        ↪  FUNC_DISPLAY_DRAW_LINE,
5                       FUNC_DISPLAY_DRAW_CIRCLE,
                        ↪  FUNC_DISPLAY_FILL_CIRCLE,
6                       FUNC_DISPLAY_DRAW_RECTANGLE,
                        ↪  FUNC_DISPLAY_FILL_RECTANGLE,
7                       FUNC_DISPLAY_TEXT_SIZE, FUNC_DISPLAY_FONT_SIZE,
                        ↪  FUNC_DISPLAY_ROTATION,
                        ↪  FUNC_DISPLAY_DRAW_STRING,
8                       FUNC_DISPLAY_DIM, FUNC_DISPLAY_BLINKRATE,
9  #ifdef USE_UFILESYS
10                      FUNC_DISPLAY_BATCH,
11 #endif
12                      FUNC_DISPLAY_NUMBER, FUNC_DISPLAY_FLOAT,
                        ↪  FUNC_DISPLAY_NUMBERNC, FUNC_DISPLAY_FLOATNC,
13                      FUNC_DISPLAY_RAW, FUNC_DISPLAY_LEVEL,
                        ↪  FUNC_DISPLAY_SEVENSEG_TEXT,
                        ↪  FUNC_DISPLAY_SEVENSEG_TEXTNC,
14                      FUNC_DISPLAY_SCROLLDELAY, FUNC_DISPLAY_CLOCK,
                        ↪  FUNC_DISPLAY_SCROLLTEXT
15                    };
```

Figura 3.27. Elenco delle possibili chiamate al display driver (`xdrv_13_display.ino`)

Per sviluppare un driver pienamente integrato con Tasmota, è necessario implementare all'interno della funzione principale del nostro driver, le varie chiamate mostrate nella figura 3.27.

Il file di documentazione `tasmota/API.md`,fornisce l'elenco delle funzioni richieste per ottenere un servizio di display funzionante.

3.3.3 Implementazione funzionante

In questa sezione è presentato il codice di un driver minimale, per abilitare l'uso della funzione displaytext tramite l'interfaccia HardwareSPI.

Nella figura 3.28 è riportato un esempio del driver in funzione.

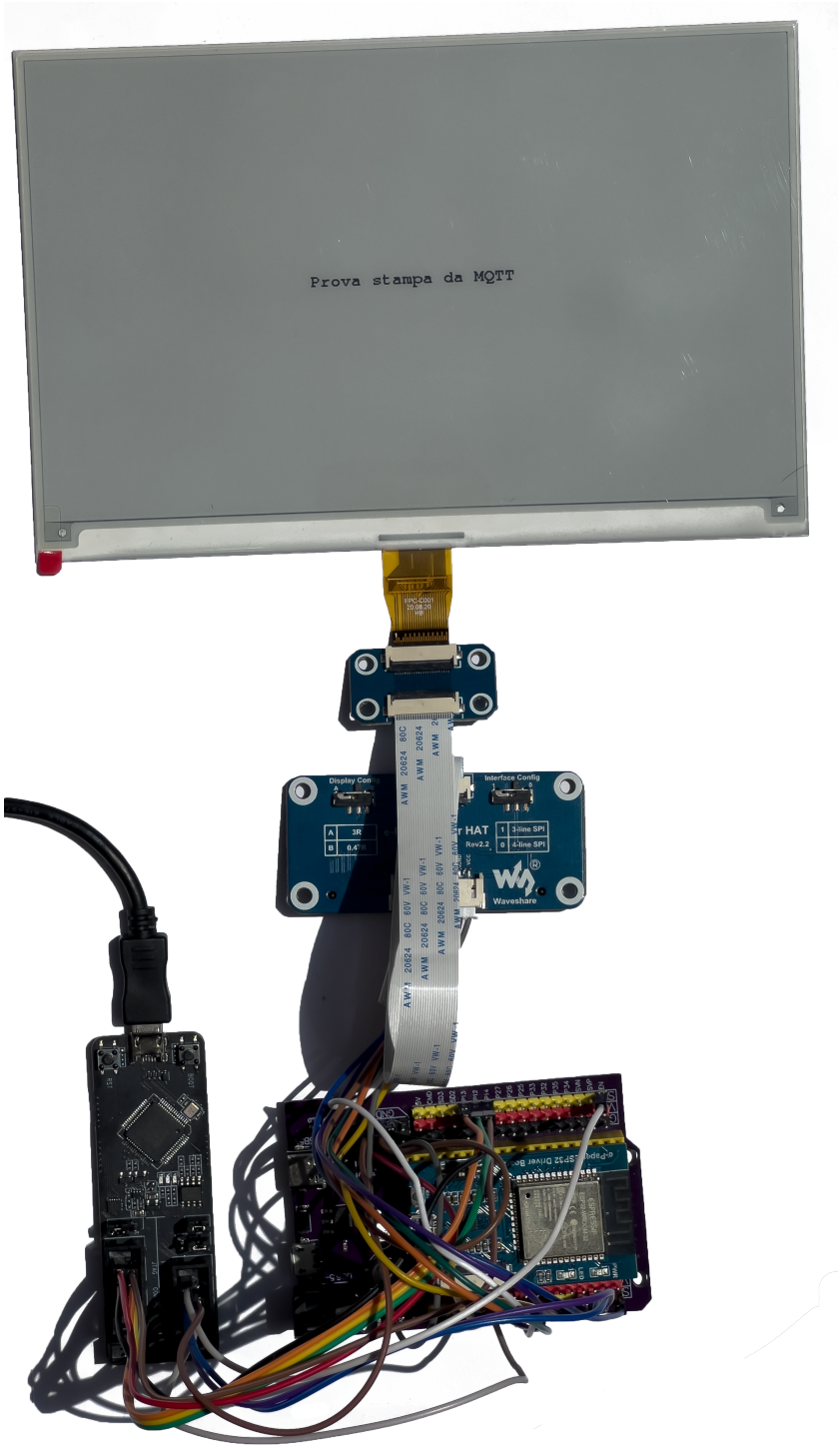

Figura 3.28. La soluzione di esempio in funzione

Codice

Segue l'implementazione del codice, del driver, contenuta all'interno del file sorgente `tasmota/tasmota_xdsp_display/xdrv_13_display_drawtext.ino`

```
#ifdef USE_SPI
#ifdef USE_DISPLAY
#ifdef USE_DISPLAY_WAVESHARE

#define XDSP_21 21

#define GxEPD2_DISPLAY_CLASS GxEPD2_BW
#define GxEPD2_DRIVER_CLASS GxEPD2_750_T7
#include <GxEPD2_BW.h>
#include <Fonts/FreeMonoBold9pt7b.h>
#include <GxEPD2_display_selection_new_style.h>

bool Xdsp21(uint32_t function)
{

    bool result = false;
    if (PinUsed(GPIO_WAVESHARE_CS) && ((TasmotaGlobal.soft_spi_enabled
    ↪  & SPI_MOSI) || (TasmotaGlobal.spi_enabled & SPI_MOSI)))
    {
        switch (function)
        {
        case FUNC_DISPLAY_INIT_DRIVER:

            AddLog(LOG_LEVEL_INFO, PSTR("DSP: Waveshare Driver"));

            Settings->display_model = XDSP_21;

            if (TasmotaGlobal.soft_spi_enabled)
            {
                Serial.println("SoftSPI");
                Serial.print("CS: ");
                Serial.println(Pin(GPIO_WAVESHARE_CS));
                Serial.print("MOSI: ");
                Serial.println(Pin(GPIO_SSPI_MOSI));
                Serial.print("SCLK: ");
                Serial.println(Pin(GPIO_SSPI_SCLK));
                Serial.print("DC: ");
                Serial.println(Pin(GPIO_SSPI_DC));
                Serial.print("BUSY/MISO: ");
                Serial.println(Pin(GPIO_SSPI_MISO));
                Serial.print("Rst: ");
                Serial.println(Pin(GPIO_OLED_RESET));
            }
            else if (TasmotaGlobal.spi_enabled)
```

```
44              {
45                  Serial.println("HardSPI");
46                  Serial.print("CS: ");
47                  Serial.println(Pin(GPIO_WAVESHARE_CS));
48                  Serial.print("MOSI: ");
49                  Serial.println(Pin(GPIO_SPI_MOSI));
50                  Serial.print("CLK: ");
51                  Serial.println(Pin(GPIO_SPI_CLK));
52                  Serial.print("DC: ");
53                  Serial.println(Pin(GPIO_SPI_DC));
54                  Serial.print("BUSY/MISO: ");
55                  Serial.println(Pin(GPIO_SPI_MISO));
56                  Serial.print("Rst: ");
57                  Serial.println(Pin(GPIO_OLED_RESET));
58                  const char splash_text[] = "Waveshare Universal
                    ↪  Driver";
59
60                  display.init(115200, true, 2, false); // USE THIS for
                    ↪  Waveshare boards with "clever" reset circuit, 2ms
                    ↪  reset pulse
61                  display.setRotation(0);
62                  display.setFont(&FreeMonoBold9pt7b);
63
64                  display.setTextColor(GxEPD_BLACK);
65                  int16_t tbx, tby;
66                  uint16_t tbw, tbh;
67                  display.getTextBounds(splash_text, 0, 0, &tbx, &tby,
                    ↪  &tbw, &tbh);
68                  uint16_t x = ((display.width() - tbw) / 2) - tbx;
69                  uint16_t y = ((display.height() - tbh) / 2) - tby;
70                  display.setFullWindow();
71
72                  display.firstPage();
73                  do
74                  {
75                      display.setCursor(x, y);
76                      display.print(splash_text);
77                  } while (display.nextPage());
78
79                  delay(1000);
80                  display.firstPage();
81                  display.nextPage();
82                  display.hibernate();
83              }
84
85              break;
86          case FUNC_DISPLAY_DRAW_STRING:
87              int16_t tbx, tby;
88              uint16_t tbw, tbh;
```

```
89              display.getTextBounds(dsp_str, 0, 0, &tbx, &tby, &tbw,
                ↪  &tbh);
90              // center the bounding box by transposition of the origin:
91              uint16_t x = ((display.width() - tbw) / 2) - tbx;
92              uint16_t y = ((display.height() - tbh) / 2) - tby;
93              display.setFullWindow();
94
95              display.firstPage();
96              do
97              {
98                  display.setCursor(x, y);
99                  display.print(dsp_str);
100             } while (display.nextPage());
101             display.hibernate();
102
103             break;
104         }
105     }
106     return result;
107 }
108
109 #endif // USE_DISPLAY_EPAPER42
110 #endif // USE_DISPLAY
111 #endif // USE_SPI
```

3.3.4 Vantaggi e Svantaggi della soluzione realizzata

Effettuare un porting delle librerie preesistenti per ESP32 offre innegabili vantaggi in termini di flessibilità. Questa approccio consente a Tasmota di adottare una vasta gamma di dispositivi e-ink con una semplice modifica della direttiva prima della compilazione. Tuttavia, questa soluzione comporta la necessità di ricompilare l'intero firmware di Tasmota, il che può essere un processo laborioso e richiedere tempo.

Per evitare la ricompilazione completa, un'alternativa sarebbe riscrivere il driver utilizzando il linguaggio Berry e creare un'applicazione .tapp dedicata. In alternativa, si potrebbero sviluppare diverse applicazioni Berry, ognuna adattata a uno specifico schermo, al fine di ottenere un utilizzo delle risorse simile a quello dell'Universal Display Driver. Tuttavia, caricare l'intero driver in memoria occupa spazio aggiuntivo rispetto alle soluzioni specifiche, aumentando il consumo di risorse.

D'altro canto, l'utilizzo del driver universale, mediante l'implementazione di file di configurazione `display.ini` per ciascun schermo della libreria, offre un approccio più efficiente nell'utilizzo delle risorse disponibili. Tuttavia, la creazione e la manutenzione di questi file di configurazione possono richiedere competenze tecniche avanzate e un investimento di tempo iniziale maggiore. Nonostante ciò, i benefici a lungo termine in termini di gestione delle risorse e facilità d'uso per l'utente possono ampiamente giustificare questo sforzo iniziale.

Capitolo 4

Caso studio: BioPDU nel Mining di Bitcoin

4.1 Introduzione

Nel settore del mining, è fondamentale controllare l'uso dell'elettricità per garantire la redditività dell'operazione. Le PDU (dall'inglese Power Distribution Units) sono diventate dispositivi essenziali per migliorare l'efficienza delle attività di mining.

Una PDU è un dispositivo dalla costruzione robusta, progettato per la massima efficienza e affidabilità nel distribuire l'alimentazione elettrica a varie apparecchiature all'interno di un data center o di una struttura di mining.

Nel contesto del mining di criptovalute, è necessaria una PDU specializzata per gestire i requisiti di alimentazione e le configurazioni uniche degli impianti di mining.

Le PDU commercializzate per il mining di criptovalute sono progettate con capacità di alimentazione elevata, che vanno da 20A a 60A o anche superiori, consentendo loro di supportare più impianti di mining contemporaneamente. Alcuni dispositivi offrono inoltre monitoraggio e gestione avanzati, tra cui monitoraggio del consumo energetico in tempo reale e rilevamento della temperatura.

4.2 Le PDU tradizionali

Una PDU è un dispositivo utilizzato per distribuire energia elettrica a vari dispositivi all'interno di un data center, di una struttura di mining o di altre installazioni che richiedono un alto livello di gestione dell'energia.

Le PDU possono essere considerate come delle prese multiple avanzate, progettate per ambienti professionali, in cui è necessario garantire un'alimentazione affidabile e gestibile per numerosi dispositivi elettronici.

Figura 4.1. Esempio di Basic Rack PDU [43]

4.2.1 Caratteristiche principali delle PDU

La funzione principale di una PDU è distribuire l'energia elettrica proveniente da una singola fonte a molteplici dispositivi collegati.

Per motivi di praticità, oltre alla distribuzione dell'energia, molte PDU sono dotate di capacità di monitoraggio in tempo reale del consumo energetico. Questo monitoraggio fornisce dati preziosi che aiutano a gestire l'efficienza energetica e prevenire sovraccarichi e sovratensioni, garantendo che le apparecchiature collegate siano protette da eventuali danni causati da fluttuazioni nell'alimentazione elettrica.

Inoltre, alcune PDU avanzate offrono funzionalità di controllo remoto, permettendo agli amministratori di accendere o spegnere dispositivi, riavviare apparecchiature e configurare impostazioni specifiche senza dover essere fisicamente presenti.

4.2.2 Tipologie di PDU

Le PDU possono essere classificate in base, metered, monitored oppure switched.

Le **PDU base** sono una soluzione entry-level a basso costo che offre una semplice distribuzione dell'energia senza funzionalità avanzate di monitoraggio o controllo. Invece, le **PDU metered**, forniscono informazioni sul consumo energetico in tempo reale tramite un display locale, ma senza capacità di controllo remoto.

Monitoraggio a livello di rack e accesso remoto sono offerti dalle PDU monitored. Mentre le **PDU switched** offrono la massima flessibilità e gestione, combinando monitoraggio e controllo remoto.

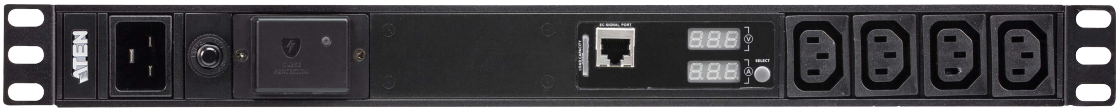

Figura 4.2. Esempio di PDU Metered [3]

4.3 La necessità delle PDU intelligenti

Le PDU intelligenti svolgono un ruolo essenziale nel contesto del mining per diverse ragioni. Per esempio, senza di esse, le mining farm non sarebbero in grado di offrire servizi di housing in modo efficiente. È fondamentale monitorare con precisione il consumo energetico di ciascun cliente per poter offrire tali servizi. Senza queste informazioni dettagliate, sarebbe praticamente impossibile determinare con esattezza i costi da addebitare ai singoli clienti per l'utilizzo dell'energia.

Inoltre, queste PDU avanzate supportano un monitoraggio proattivo delle apparecchiature, permettendo di individuare tempestivamente eventuali guasti e di intervenire rapidamente per risolverli.

Un altro vantaggio significativo delle PDU smart è la loro capacità di implementare logiche specifiche per la gestione dell'accensione e dello spegnimento degli ASIC.

Ad esempio, chi utilizza energia fotovoltaica per il mining può configurare le PDU affinché alimentino i dispositivi solo quando è disponibile energia solare, ottimizzando così l'uso delle risorse energetiche rinnovabili.

In aggiunta, il trasporto dell'energia è un processo costoso e può comportare sprechi di corrente. In alcune circostanze, è più conveniente non produrre energia piuttosto che generare un surplus. Le PDU intelligenti possono essere utili in questo contesto, poiché permettono di accendere o spegnere le apparecchiature per il mining in caso di surplus energetico. In questo scenario, i dispositivi di mining agirebbero come carico noto, assorbendo l'energia elettrica che altrimenti andrebbe sprecata a causa della mancanza di domanda.

Alcune PDU specifiche per il mining, includono switch di rete integrati. Questa integrazione semplifica l'installazione e la gestione complessiva delle apparecchiature di mining, eliminando la necessità di dispositivi di rete aggiuntivi e consentendo una configurazione più snella e integrata dell'intera infrastruttura di mining.

4.4 Il ruolo di un firmware open source come Tasmota

Un firmware open source come Tasmota svolge un ruolo fondamentale nel panorama delle soluzioni per il mining. Mentre molte soluzioni attualmente disponibili sul mercato utilizzano firmware proprietario, che limita la flessibilità e l'adattabilità dei dispositivi, Tasmota rappresenta un'alternativa più versatile. Questo perché, essendo software libero e sviluppato dalla comunità open source, può essere facilmente modificato e adattato alle esigenze specifiche degli utenti. Di conseguenza, le soluzioni open source sono spesso preferite da coloro che necessitano di un elevato livello di personalizzazione per le proprie apparecchiature di mining.

Tuttavia, è importante notare che installare Tasmota su PDU preconfezionate risulta spesso difficile, se non impossibile. Questo è dovuto al fatto che tali dispositivi generalmente vengono progettati per non permette modifiche o sostituzioni del software. Pertanto, per utilizzare Tasmota, gli utenti devono ricorrere ad altri dispositivi compatibili e unire varie componenti hardware per creare una soluzione personalizzata.

Questo processo richiede una buona conoscenza tecnica e la capacità di integrare diverse tecnologie, ma offre in cambio un livello di controllo e personalizzazione che le soluzioni preconfezionate non possono eguagliare. Per esempio il firmware Tasmota, di default include il supporto al monitoraggio di corrente 4.3.

Figura 4.3. Monitoraggio di corrente tramite l'interfaccia web di Tasmota

Inoltre, l'uso di Tasmota consente agli utenti di beneficiare di aggiornamenti regolari e di nuove funzionalità sviluppate dalla comunità, migliorando continuamente l'efficienza e la sicurezza delle proprie apparecchiature di mining.

4.5 Installazione di Tasmota su dispositivi commerciali

Sul mercato sono disponibili vari dispositivi che consentono di erogare corrente e monitorare i consumi, alcuni dei quali dotati del SoC ESP32. Un esempio è il SONOFF POW Origin (Figura 4.4).

Figura 4.4. Sonoff POW Origin [26]

Tuttavia, questi dispositivi, incluso il Sonoff, sono forniti con un firmware proprietario che presenta una rigidità troppo elevata per poter essere utilizzati direttamente nel contesto del mining. Essi risultano più adatti a un utilizzo plug-and-play per applicazioni di domotica domestica.

Fortunatamente, tutte le schede del marchio Sonoff, sotto le varie coperture plastiche, forniscono l'accesso ai pin di programmazione (GPIO0, RX, TX e GND), i quali possono essere utilizzati per riprogrammare completamente il dispositivo. Questa caratteristica consente ai Sonoff di essere riflashati con Tasmota, utilizzando un adattatore FTDI.

L'adattatore FTDI è un dispositivo che permette la comunicazione tra il computer e il microcontrollore tramite l'interfaccia seriale USB, facilitando così il processo di riprogrammazione. Il programmatore seriale più affidabile è CH340G [29]. In alternativa, anche una board Arduino UNO può essere utilizzata come adattatore seriale [25].

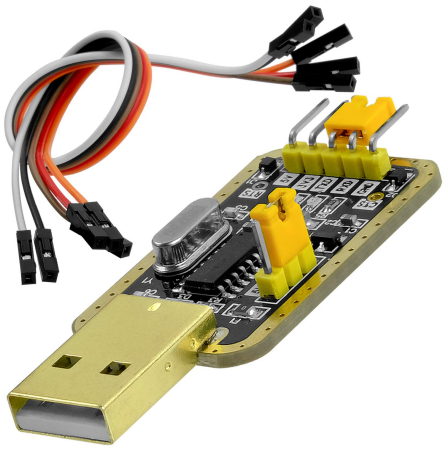

Figura 4.5. Adattatore USB da UART a TTL basato su chip CH340G [2]

4.5.1 Procedura di flashing

Prima di iniziare la procedura di flashing, è necessario disconnettere il dispositivo Sonoff dalla corrente elettrica[1] per garantire la sicurezza. In secondo luogo, è necessario smontare il dispositivo per accedere ai pin del microcontrollore presenti all'interno.

Successivamente, bisogna collegare l'adattatore USB a TTL al Sonoff utilizzando i cavi jumper seguendo queste istruzioni: il pin `TX` dell'adattatore dev'essere collegato al pin `RX` del Sonoff, il pin `RX` dell'adattatore al pin `TX` del Sonoff, il `GND` dell'adattatore al `GND` del Sonoff e il `VCC` (3.3V) dell'adattatore al `VCC` del Sonoff.

Per mettere il Sonoff in modalità di flash, è necessario premere e tenere premuto il pulsante presente sul dispositivo. Bisogna quindi collegare l'adattatore USB al computer mentre si continua a tenere premuto il pulsante. Questo passaggio consente di mettere il Sonoff in modalità di flash.

Nella figura 4.6 è mostrato un dispositivo Sonoff POW Origin connesso al PC tramite adattatore FTDI.

[1] È importante operare in sicurezza, specialmente quando si manipolano dispositivi elettrici. Se possibile, è consigliabile eseguire un backup del firmware originale del Sonoff prima di procedere con il flashing di Tasmota. All'interno della documentazione ufficiale di Tasmota sono forniti ulteriori dettagli e soluzioni per risolvere eventuali problemi che potrebbero sorgere durante il processo.

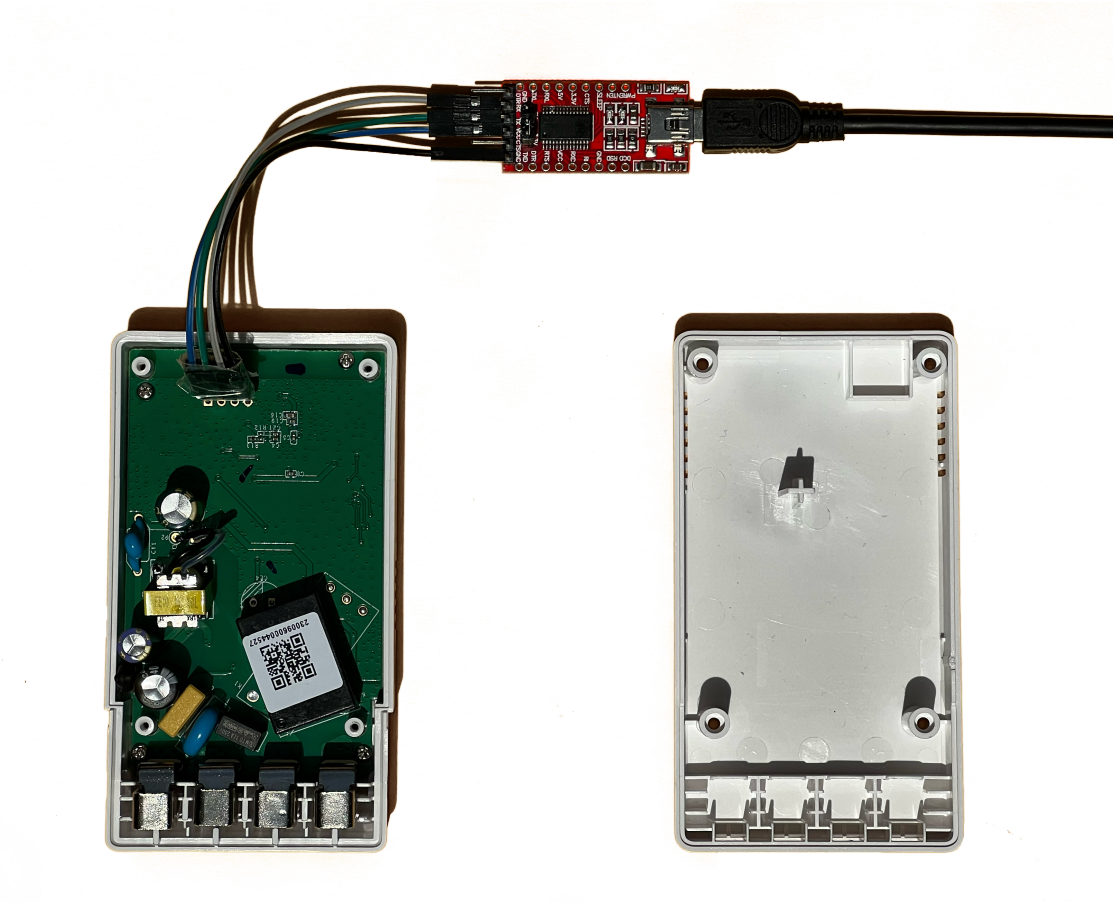

Figura 4.6. Cablaggio per il flashing di Sonoff POW Origin

Dal computer bisogna scaricare l'ultima versione del firmware Tasmota dal sito ufficiale e installare un software di flashing come esptool.py [12] (che permette anche di effettuare il backup del firmware originale del dispositivo) oppure Tasmotizer, [36], consigliato per la sua semplicità d'uso.

Utilizzando Tasmotizer, basterà solamente scaricare e installare il programma dal sito ufficiale, aprire Tasmotizer e selezionare la porta seriale a cui è collegato l'adattatore USB a TTL. Infine, bisogna caricare il firmware Tasmota (.bin) precedentemente scaricato e cliccare su "Tasmotize!" per avviare il processo di flashing.

Al termine del processo, Tasmotizer potrebbe richiedere di configurare il Wi-Fi; è possibile eseguire questa configurazione immediatamente oppure successivamente tramite l'interfaccia web di Tasmota.

4.6 La BioPDU

Il SONOFF POW Origin, risulta una soluzione ottima per alimentare degli ASIC, tuttavia, su larga scala, dato che si basa sull'utilizzo di connettività Wi-Fi, può generare una situazione chiamata boot storm.

Un bootstorm si verifica quando più dispositivi tentano di connettersi contemporaneamente alla rete Wi-Fi. Questa azione può comportare un'elevata richiesta sull'intera rete, causando ritardi indesiderati. I boot storm influiscono inevitabilmente sulle prestazioni della rete a causa dell'elevato consumo energetico, che può anche causare il crash della rete, impedendo agli utenti di accedervi completamente [19]. [4pt] Questo fenomeno è inaccettabile nel contesto del mining, dove ogni ottimizzazione e il tempo di attività sono essenziali per garantire la redditività. Per questi motivi, è nata la BioPDU, una PDU realizzata con una scheda custom, che utilizza il SoC ESP32 e il firmware Tasmota, e che include anche al suo interno uno switch ethernet.

La BioPDU è stata realizzata per essere modulare, infatti dispone di tre porte I2C, che possono essere utilizzate per aggiungere display e sonde di temperatura e umidità. Tasmota include dalla versione 12.4.0 il supporto alla BioPDU [13].

Nella figura 4.7 si può vedere il primo prototipo di BioPDU in funzione.

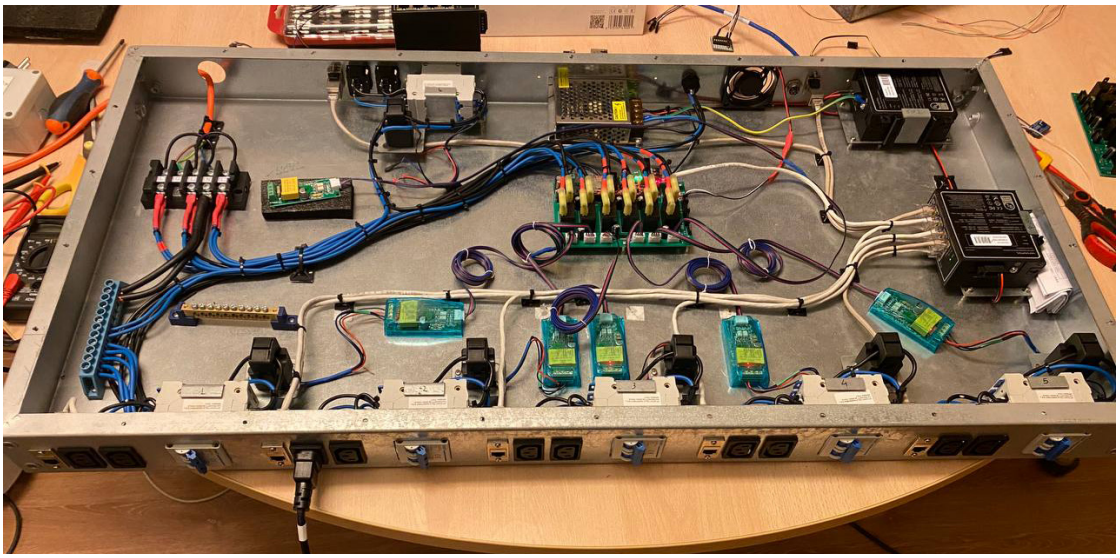

Figura 4.7. Primo prototipo di BioPDU [13]

Inoltre, questa PDU, è stata pensata per essere usata come ripiano di uno scaffale, in modo da ottimizzare gli spazi e i cablaggi, garantendo un setup più rapido e pulito.

Velocizzare il provisioning, fa la differenza nel mining, pochi minuti di inattività possono essere decisivi, anche un breve periodo di inattività di mining può comportare la perdita di opportunità preziose.

Risparmiare spazio consente anche di risparmiare sul costo del raffreddamento, in quanto più apparati possono essere posizionati nello stesso volume raffreddato.

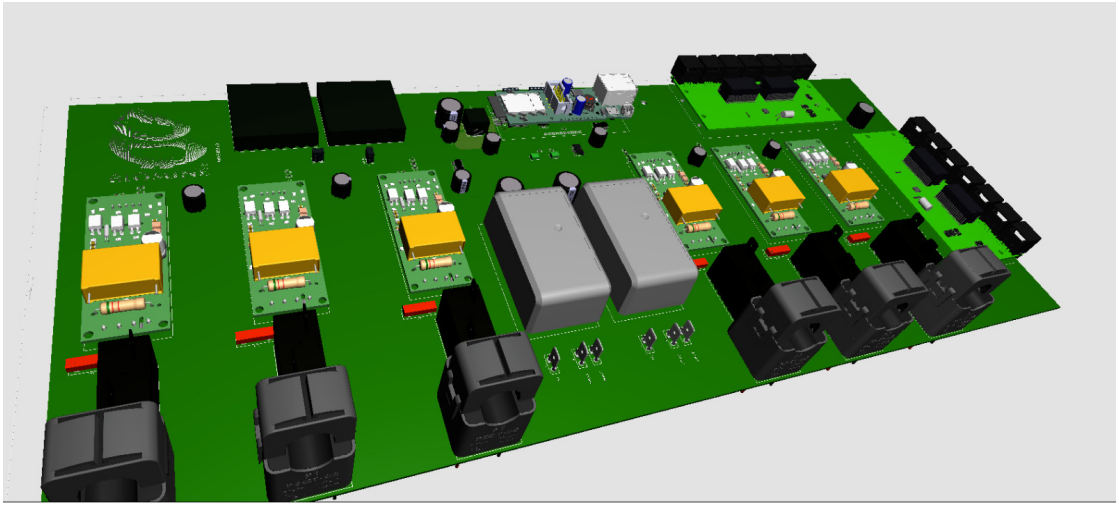

Figura 4.8. Design della board definitiva di BioPDU [13]

4.6.1 Architettura

Le possibilità offerte da MQTT

L'uso di MQTT permette la gestione della logica tramite componenti esterni, che possono essere implementati anche da individui non direttamente coinvolti nel progetto. Il microcontrollore ESP32, infatti, non dispone della potenza sufficiente per ospitare logiche complesse, è pertanto possibile delegare tutte le funzioni necessarie a strutture esterne. La BioPDU è progettata per essere integrata in un'infrastruttura più ampia, dotata di un broker MQTT e di datastore per archiviare i dati di telemetria nel tempo.

Utilizzando il firmware Tasmota e il protocollo MQTT, la BioPDU offre numerose possibilità: ad esempio, è in grado di segnalare guasti tramite Telegram. Un client esterno monitora il Topic della telemetria e, rilevando anomalie come interruzioni anomale nel consumo di corrente, invia notifiche all'utente tramite messaggi Telegram.

Raccolta e storicizzazione dei dati

Il sistema della BioPDU è stato progettato specificamente per funzionare in un contesto in cui è necessario monitorare e gestire dispositivi in tempo reale. Il processo inizia con un client MQTT che riceve telemetrie dai dispositivi ogni 30 secondi. Questo client ha il compito di memorizzare i dati ricevuti in un database, garantendo che tutte le informazioni siano storicizzate correttamente per analisi future.

A titolo di esempio, nella figura 4.9 è mostrato un messaggio MQTT grezzo contenente i dati energetici di telemetria.

```
{
    "Time": "2021-04-25T00:37:02",
    "ENERGY": {
        "TotalStartTime": "2020-10-02T00:00:00",
        "Total": 7477.6,
        "Yesterday": 42.495,
        "Today": 1.076,
        "Period": 14,
        "Power": 1263,
        "ApparentPower": 1263,
        "ReactivePower": 0,
        "Factor": 1,
        "Voltage": 226,
        "Current": 5.599
    }
}
```


Figura 4.9. Messaggio MQTT di telemetria di BioPDU

Mentre nella figura 4.10 sono mostrati i dati relativi all'umidità e alla temperatura del sensore installato all'interno di una BioPDU che dispone del sensore necessario.


```
{
    "Time": "2021-04-25T00:41:11",
    "AM2301": {
        "Temperature": 16.3,
        "Humidity": 67.1,
        "DewPoint": 10.2
    },
    "TempUnit": "C"
}
```


Figura 4.10. Messaggio MQTT di telemetria per i sensori di BioPDU

Monitorare la temperatura degli ASIC è fondamentale per garantire l'efficienza operativa. Questi dispositivi funzionano in modo ottimale entro un intervallo di temperature specifico. Quando si surriscaldano, la loro efficienza di calcolo diminuisce, portando a una riduzione delle prestazioni e, di conseguenza, a un calo dei profitti. Mantenere una temperatura adeguata assicura che gli ASIC operino al massimo delle loro capacità.

Un altro motivo cruciale per il monitoraggio della temperatura è la longevità dell'hardware. Temperature elevate accelerano il degrado dei componenti elettronici, riducendo la

vita utile degli ASIC. Un monitoraggio costante delle temperature permette di mantenerle entro livelli sicuri, prevenendo danni prematuri e aumentando la durata dell'hardware. Questo non solo riduce i costi di sostituzione, ma assicura anche un funzionamento continuo e affidabile.

Controllo dei dispositivi

Periodicamente, un server dedicato esegue delle operazioni programmate, partendo dai dati raccolti dalle varie PDU, contenuto all'interno del database citato in precedenza.

Una di queste operazioni di routine consiste nell'estrazione dal database dei dati di consumo attuale per ciascun dispositivo, in modo da poter intervenire se i consumi rilevati sono fuori dai parametri predefiniti, sia superiori che inferiori ai limiti impostati. In caso i consumi non rientrino nell'intervallo stabilito, il sistema esegue automaticamente le seguenti azioni: invia un comando tramite MQTT per spegnere il dispositivo e, dopo 30 secondi, invia un altro comando per riaccendere il dispositivo. Questo meccanismo è pensato per garantire che i dispositivi funzionino sempre entro i parametri ottimali, riducendo il rischio di malfunzionamenti o danni.

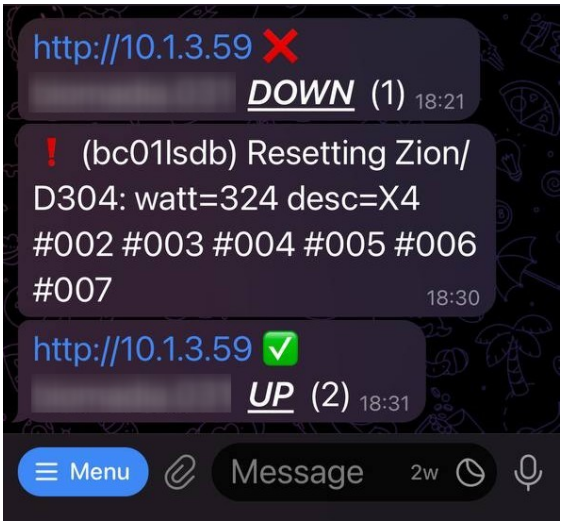

Figura 4.11. Notifiche ricevute durante una procedura di ripristino automatizzata

Notifiche e controllo via Telegram

La BioPDU è stata pensata per essere abbinata ad un bot di Telegram, che si occupa della gestione delle notifiche e del controllo manuale.

Nella figura 4.12 sono mostrati i comandi accettati dal bot.

Figura 4.12. Comandi disponibili dal bot Telegram

È richiesta un'istanza locale del Telegram Bot API server [41] per permettere l'interfacciamento di Telegram con la rete interna della mining farm, in modo che il bot possa accedere a risorse non pubbliche.

Il bot di Telegram può essere configurato per inviare notifiche all'utente su eventi e anomalie rilevate.

Inoltre, permette il controllo remoto dei dispositivi, come l'accensione e lo spegnimento di singoli dispositivi o dell'intera farm, e interrogando il database, può fornire informazioni in tempo reale sui consumi e altri dati rilevanti.

Nella figura 4.13 è mostrata la procedura di accensione di tutta la farm direttamente dal bot di Telegram.

Figura 4.13. Avvio della mining farm direttamente dal bot Telegram

Comunicazione e automazione

Il sistema della BioPDU è altamente automatizzato, ma offre anche strumenti per il controllo manuale e la ricezione di notifiche in tempo reale. Questo doppio approccio, che combina automazione con la possibilità di intervento manuale, è stato pensato per garantire che l'utente possa intervenire prontamente in caso di problemi o anomalie. In questo modo, si migliora l'efficienza e la gestione dei dispositivi, assicurando un funzionamento continuo e sicuro all'interno della rete locale della farm.

Capitolo 5

Conclusioni

In conclusione, il progetto presentato ha mostrato l'efficacia delle soluzioni open-source nel campo dell'Internet of Things (IoT) applicato al mining di Bitcoin. Attraverso l'implementazione di Tasmota su dispositivi di uso comune, come le PDU, è possibile ottenere un controllo avanzato e una gestione ottimizzata dell'energia, che sono cruciali per le operazioni delle mining farm.

La BioPDU rappresenta un significativo passo avanti in termini di efficienza e ottimizzazione. Grazie al firmware Tasmota ed all'uso del protocollo MQTT, offre vantaggi non solo in termini di monitoraggio e gestione dell'energia, ma anche di notifica e intervento tempestivo in caso di guasti, riducendo al minimo i tempi di inattività.

L'adozione di queste tecnologie consente di superare le limitazioni delle PDU tradizionali, fornendo soluzioni smart che migliorano la flessibilità e l'efficienza operativa. Questo progetto evidenzia come l'innovazione nel campo del firmware ed hardware possa contribuire a migliorare le performance delle mining farm, rendendole più sostenibili ed efficienti.

In definitiva, la combinazione di tecnologie open-source, come Tasmota, con soluzioni hardware personalizzate, rappresenta una strada promettente per l'evoluzione futura delle infrastrutture nel mining di Bitcoin e in altre applicazioni IoT.

Bibliografia

[1] Khalid Aloufi and Omar Alhazmi. A hybrid iot security model of mqtt and uma. *Communications and Network*, 12:155–173, 11 2020.

[2] Amazon.com, Inc. Azdelivery uart-ttl usb adattatore con convertitore ch340g 3.3v e 5v con cavo jumper compatibile con arduino incluso un e-book! : Amazon.it: Informatica. https://www.amazon.it/AZDelivery-Adattatore-CH340G-jumper-Parent/dp/B08T24NML9?th=1. [Online; Consultato il 10 Giugno 2024].

[3] ATEN INTERNATIONAL Co., Ltd. Basic metered 1u pdu with surge protection - pe1218s, aten rack pdu aten corporate headquarters. https://www.aten.com/global/en/products/power-distribution-&-racks/rack-pdu/pe1218s/. [Online; Consultato il 12 Giugno 2024].

[4] Berry Community. Github - berry-lang/berry: A ultra-lightweight embedded scripting language optimized for microcontrollers. https://github.com/berry-lang/berry/. [Online; Consultato il 29 Maggio 2024].

[5] Bitmain. Bitcoin miner s21 pro. https://m.bitmain.com/product/detail?pid=00020240402210032893O1Et7keX0682. [Online; Consultato il 11 Giugno 2024].

[6] Bodmer. Bodmer/epd_libraries: A set of adapted arduino libraries for waveshare epaper displays for esp8266 and esp32. https://github.com/Bodmer/EPD_Libraries. [Online; Consultato il 06 Giugno 2024].

[7] Broadcom. Rabbitmq: One broker to queue them all | rabbitmq. https://www.rabbitmq.com/. [Online; Consultato il 16 Giugno 2024].

[8] Eclipse Foundation AISBL. Eclipse mosquitto. https://mosquitto.org/. [Online; Consultato il 16 Giugno 2024].

[9] Espressif Systems (Shanghai) CO., LTD. Esp32 modules and boards — esp-idf programming guide v3.3 documentation. https://github.com/espressif/esp-idf/blob/v3.3/docs/en/hw-reference/modules-and-boards.rst. [Online; Consultato il 05 Giugno 2024].

[10] Espressif Systems (Shanghai) CO., LTD. Introduction to the esp-prog board - - — esp-iot-solution latest documentation. https://github.com/espressif/

esp-iot-solution/blob/c0550db6/docs/en/hw-reference/ESP-Prog_guide.rst, 2023. [Online; Consultato il 02 Giugno 2024].

[11] Espressif Systems (Shanghai) CO., LTD. Jtag debugging - esp32 - — esp-idf programming guide v5.2.1 documentation. https://github.com/espressif/esp-idf/blob/v5.2.1/docs/en/api-guides/jtag-debugging/index.rst, 2023. [Online; Consultato il 03 Giugno 2024].

[12] Esptool Community. espressif/esptool: Espressif soc serial bootloader utility. https://github.com/espressif/esptool. [Online; Consultato il 10 Giugno 2024].

[13] Fabrizio Amodio. Biopdu-v1.1.0 by ziofabry · pull request #17857 · arendst/tasmota. https://github.com/arendst/Tasmota/pull/17857, 2023. [Online; Consultato il 09 Giugno 2024].

[14] Fritzing GmbH. Welcome to fritzing. https://fritzing.org/. [Online; Consultato il 14 Giugno 2024].

[15] Greg Roelofs. Info-zip's zip. https://infozip.sourceforge.net/Zip.html. [Online; Consultato il 30 Maggio 2024].

[16] HiveMQ, Inc. Hivemq - the most trusted mqtt platform to transform your business. https://www.hivemq.com/. [Online; Consultato il 16 Giugno 2024].

[17] LoyceV. Bitcoin block data available in csv format. https://bitcointalk.org/index.php?topic=5246271.0, 05 2020. [Online; Consultato il 12 Giugno 2024].

[18] LVGL Community. Tasmota and berry — lvgl documentation. https://github.com/lvgl/lvgl/blob/d851fe0528fcb920fee86c944fe9dbbaf6fbb0c9/docs/get-started/tasmota-berry.md. [Online; Consultato il 05 Giugno 2024].

[19] Margaret Rouse. What is a boot storm? - definition from techopedia. https://www.techopedia.com/definition/31686/boot-storm. [Online; Consultato il 09 Giugno 2024].

[20] Microsoft. Visual studio code - code editing. redefined. https://code.visualstudio.com/. [Online; Consultato il 06 Giugno 2024].

[21] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008. [Online; Consultato il 14 Giugno 2024].

[22] Open On-Chip Debugger. Your gateway to embedded software development excellence - platformio. https://openocd.org/. [Online; Consultato il 06 Giugno 2024].

[23] PlatformIO Labs. Your gateway to embedded software development excellence - platformio. https://platformio.org/. [Online; Consultato il 06 Giugno 2024].

[24] SEGGER Microcontroller GmbH. Segger j-link debug probes. https://www.segger.com/products/debug-probes/j-link/. [Online; Consultato il 05 Giugno 2024].

[25] Simon Ogden. How to flash sonoff tasmota with an arduino uno and esptool - siytek. https://siytek.com/how-to-flash-sonoff-tasmota-with-an-arduino-uno-and-esptool/#Connect-to-the-Arduino, 2020. [Online; Consultato il 09 Giugno 2024].

[26] SONOFF. Pow origin - sonoff official. https://sonoff.tech/product/diy-smart-switches/pow-origin/. [Online; Consultato il 09 Giugno 2024].

[27] soonuse. soonuse/epd-library-arduino: Arduino libraries for waveshare e-paper series. https://github.com/soonuse/epd-library-arduino. [Online; Consultato il 06 Giugno 2024].

[28] Tasmota Community. arendst/tasmota: Alternative firmware for esp8266 and esp32 based devices with easy configuration using webui, ota updates, automation using timers or rules, expandability and entirely local control over mqtt, http, serial or knx. full documentation at https://tasmota.github.io/docs. https://github.com/arendst/Tasmota/tree/development/tasmota/berry/modules. [Online; Consultato il 30 Maggio 2024].

[29] Tasmota Community. Getting started - tasmota. https://github.com/tasmota/docs/blob/master/docs/Getting-Started.md. [Online; Consultato il 09 Giugno 2024].

[30] Tasmota Community. Peripherals - tasmota. https://github.com/tasmota/docs/blob/master/docs/Supported-Peripherals.md. [Online; Consultato il 27 Maggio 2024].

[31] Tasmota Community. Releases · arendst/tasmota. https://github.com/arendst/Tasmota/releases. [Online; Consultato il 02 Giugno 2024].

[32] Tasmota Community. Supported modules - tasmota. https://github.com/tasmota/docs/blob/master/docs/Supported-Modules.md. [Online; Consultato il 27 Maggio 2024].

[33] Tasmota Community. Tasmota supported devices repository. https://github.com/tasmota/docs/blob/master/docs/Templates.md. [Online; Consultato il 05 Giugno 2024].

[34] Tasmota Community. Tasmota/tasmota/displaydesc/ws_epaper42_display.ini at development · arendst/tasmota. https://github.com/arendst/Tasmota/blob/development/tasmota/displaydesc/WS_epaper42_display.ini. [Online; Consultato il 29 Maggio 2024].

[35] Tasmota Community. Tasmota/tasmota/displaydesc/ws_epaper42_display.ini at development · arendst/tasmota. https://github.com/tasmota/docs/blob/master/docs/MQTT.md. [Online; Consultato il 05 Giugno 2024].

[36] Tasmota Community. tasmota/tasmotizer: Esp... the time has come to... tasmotize! hhttps://github.com/tasmota/tasmotizer. [Online; Consultato il 10 Giugno 2024].

[37] Tasmota Community. Universal display and universal touch drivers (udisplay/utouch) - tasmota. https://github.com/tasmota/docs/blob/master/docs/Universal-Display-Driver.md. [Online; Consultato il 01 Giugno 2024].

[38] Tasmota Community. Universal display and universal touch drivers (udisplay/utouch) - tasmota. https://tasmota.github.io/docs/Universal-Display-Driver/#step-5-configure-the-displayini-descriptor-file. [Online; Consultato il 10 Giugno 2024].

[39] Tasmota Community. Visual studio code - tasmota. https://github.com/tasmota/docs/blob/master/docs/Visual-Studio-Code.md. [Online; Consultato il 06 Giugno 2024].

[40] Tasmota Community abd blakadder. Tasmota supported devices repository. https://templates.blakadder.com/. [Online; Consultato il 27 Maggio 2024].

[41] Telegram. tdlib/telegram-bot-api: Telegram bot api server. https://github.com/tdlib/telegram-bot-api. [Online; Consultato il 10 Giugno 2024].

[42] ThomasV. Controlled supply - bitcoin wiki, 2011. [Online; Consultato il 15 Giugno 2024].

[43] Vertiv Group Corp. Che cos'è una rack mount pdu? https://www.vertiv.com/it-emea/about/news-and-insights/articles/educational-articles/what-is-a-rack-pdu/. [Online; Consultato il 12 Giugno 2024].

[44] Waveshare Electronics. 7.5inch e-paper hat manual - waveshare wiki. https://www.waveshare.com/wiki/7.5inch_e-Paper_HAT_Manual. [Online; Consultato il 02 Giugno 2024].

[45] ZinggJM. Zinggjm/gxepd: A simple e-paper display library with common base class and separate io class for arduino. https://github.com/ZinggJM/GxEPD. [Online; Consultato il 06 Giugno 2024].

[46] ZinggJM. Zinggjm/gxepd2: Arduino display library for spi e-paper displays. https://github.com/ZinggJM/GxEPD2. [Online; Consultato il 06 Giugno 2024].