Sistemi Embedded e Internet-Of-Things

Relazione sul progetto SmartPlug

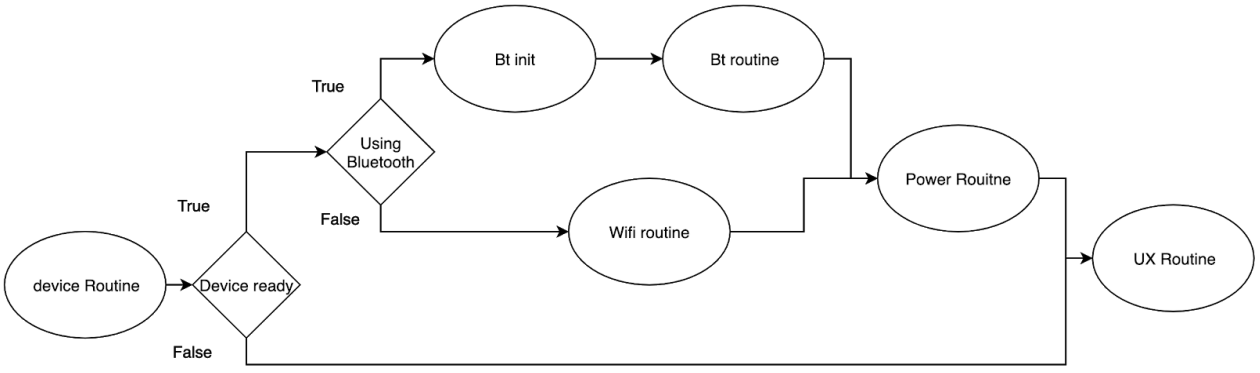
Igor Ershov
Matricola n. 0000789525
Corso in Scienze e Tecnologie Informatiche (8009)

Introduzione

Si è realizzato un sistema embedded che rappresenta una presa elettrica “intelligente” di un’abitazione.
Il compito del sistema è la lettura dei consumi di corrente nelle varie parti della casa per calcolare il consumo elettrico totale dell’abitazione, è inoltre possibile accendere e spegnere il dispositivo tramite applicazioni Android e Web. Con l’applicazione Android si avrà inoltre la possibilità di aggiungere/rimuovere prese alla rete domestica in modo dinamico. Infine il sistema fornirà la possibilità di agire come un semplice salvavita, spegnendo i dispositivi collegati all’elettricità se essi fanno superare complessivamente un valore impostato come sicuro dall’utente.
Insieme al dispositivo vengono forniti anche l'apk per Android che permette l'interazione con il dispositivo e il server Node.js che permette la connettività tramite WIFI.

Arduino

Il sistema è stato realizzato seguendo un modello Time-triggered FSM a Task.
Il main loop è composto da una sequenza di task con periodi diversi.
Il periodo di base del loop è di 50ms, questa quantità è stata scelta per garantire la responsivness nelle modalità di interazione con l’utente, che in questa versione sono limitate alla pressione di un bottone e all’accensione dei vari led.



Durante ogni loop viene aggiornato il valore del tempo trascorso nei vari task, e se coincide con il valore del periodo viene richiamato il tick del task.
L’ordine dei task, che verranno spiegati in più dettaglio in seguito, è il seguente:

- DeviceRoutine : Carica le impostazioni , pesa il sensore ed esegue le letture
- PowerRoutine: Controlla se fornire o meno elettricità alla presa di corrente.
- BTInit : Gestisce l’accensione e il setup del dispositivo Bluetooth
- BTRoutine: Gestisce la routine di pairing e di scambio di messaggio Bluetooth
- WifiRoutine: Gestisce la routine di invio/ricezione/conferma di messaggi Wifi
- UXRoutine: Gestisce lo stato dei led e del bottone

Classi generali

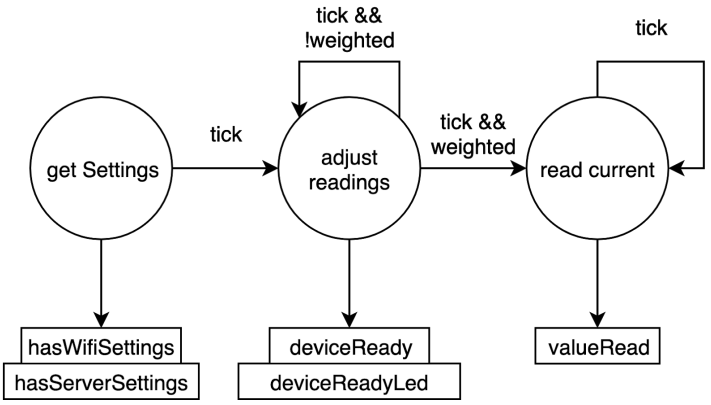
Durante lo sviluppo sono state create alcune classi che vengono richiamate ed usate nelle varie routine, e sono le seguenti:

- **Settings:** Questa classe è creata come un singleton, ed è composta da varie funzioni di set e get delle varie impostazioni del dispositivo, tali la rete wifi e la password, l'indirizzo del server, l'id e la chiave del dispositivo, e il wattaggio massimo di salvavita. Ogni routine può accedere in lettura a queste informazioni ma si è prestata attenzione ad accederein scrittura solo in momenti in cui si è certi che nessun'altra funzione può aver bisogno di scriverci. Inoltre queste impostazioni sono salvate nella EEPROM del dispostivo in modo da mantenerle tra diverse esecuzioni.
- **Flags:** Anche questa classe viene creata come singleton, è composta dalle variabili "globali" su cui le varie routine baseranno il loro funzionamento o per comunicare tra di loro. Anche questa classe è composta principalmente di setter e getter ma a differenza dei settings queste informazioni sono necessare solo all'esecuzione corrente del dispositivo , quindi non è necessario mantenerle tra esecuzioni. Alcuni esempi di variabili globali, sono gli stati dei vari led, lo stato di pressione del bottone ed il valore letto dal sensore
- **WordFinder:** Classe di supporto, permette di impostare una parola e la cerca di nella sequenza di caratteri fornita. Ogni carattere viene fornito tramite una chiamata alla sua funzione parse(char) diversa, questo permette di trovare parole quando il buffer Seriale viene riempito durante diversi tick. Quando la parola è stata trovata la chiamata a parse() ritornerà vero.
- **CommandParser:** Una versione avanzata di WordFinder usata nelle comunicazioni con il dispositivo wifi e il dispositivo bluetooth,permette di impostare i diversi comandi che il dispositivo può ricevere e supporta la lettura di eventuali parametri forniti assieme al comando. La chiamata parse(char) ritorna il numero del comando trovato, oppure -1 in caso non è ancora stato trovato nessun comando, cosi da poter gestire l'eventuale comando ricevuto dentro un semplice switch. Inoltre fornisce la funzione "getParam(int)" che restituirà il puntatore alla stringa contenente il parametro. Il programmatore sa durante la scrittura del codice l'ordine dei parametri e la loro funzionalità, quindi la chiamata al parametro giusto, e il suo corretto utilizzo, sono sua responsabilità.

Device Routine

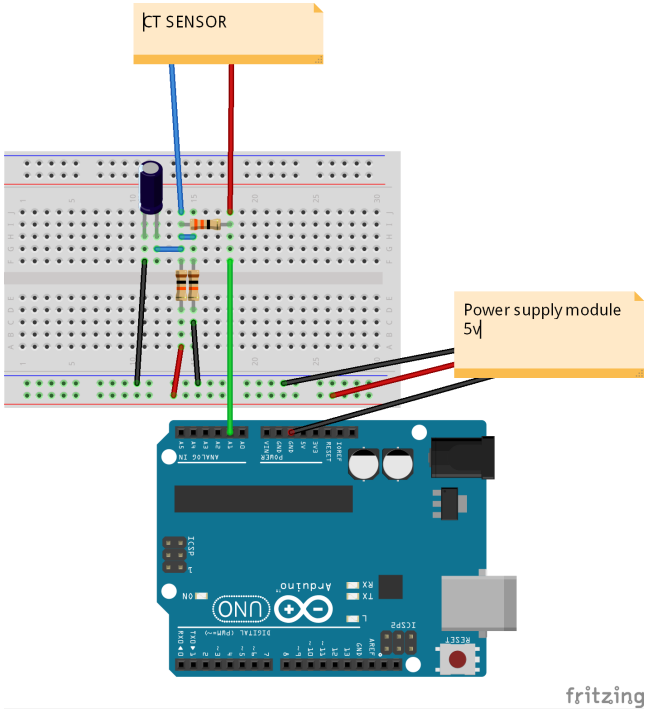
La device routine gestisce il caricamento di informazioni dalla EEPROM, il pesaggio del sensore e la seguente lettura. E' stato scelto un periodo di 200ms per fornire informazioni sempre aggiornate, non è stato scelto un periodo più breve perché l'operazione di lettura del sensore è bloccante e viene eseguita con tempistiche non banali.
Utilizza un sensore CT per rilevare la corrente.

Routine



Lo stato inziiale di questa routine è getSettings, dove il dispositivo raccoglie le informazioni dalla EEPROM, controlla inoltre le informazioni raccolte e vede se sono presenti le impostazioni del wifi e l'indirizzo del server, impostando di conseguenza i flag associati.
Passa al tick seguente allo stato adjustReadings, dove chiama l'operazione "weight" del sensore CT. Quando essa ritornerà vero il sensore può considerarsi pesato, quindi il dispositivo può iniziare il suo funzionamento.
Viene quindi settata la flag deviceReady e viene impostato il led del dispositvo come acceso.
La routine entrerà quindi nella modalità "readCurrent" dove leggerà la corrente attuale e aggiornerà il valore da lui letto di conseguenza,. La routine rimarrà bloccata in questo stato fino al riavvio del dispositivo.

Dispositivo utilizzato



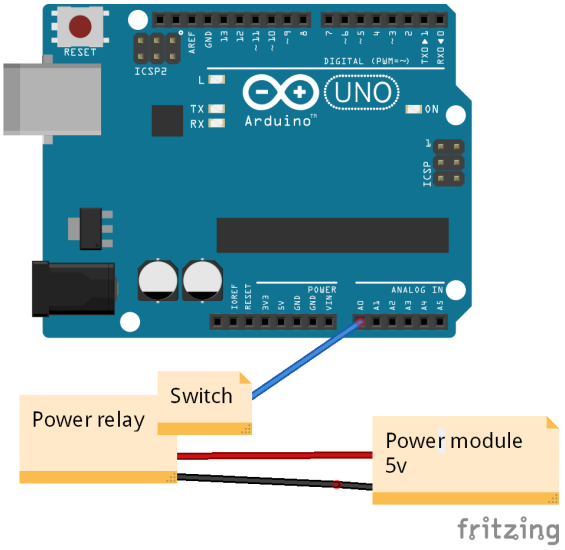
In questa routine viene utilizzato un sensore di corrente "SCT-013-000 AC". E' stata creata una classe virtuale per un rilevatore di corrente generico, con funzioni di lettura e pesatura. La classe CTSensor le implementa.
La pesatura avviene a corrente spenta, quindi sappiamo che il valore che dovremmo rilevare deve essere il più vicino possibile a zero. Viene letto il valore di corrente e se

il valore è superiore a zero viene incrementato un peso. Questo viene ripetuto finchè il valore letto non è vicino allo zero.
Nella funzione di lettura il dispositivo legge il valore e lo restituisce, se la presa elettrica è spenta usa l'opportunità per eseguire delle operazioni di pesatura aggiuntive in modo da migliorare l'accuratezza.
Per l'effettiva lettura della corrente vengono utilizzate le librerie di OpenEnergyMonitor.

Routine Power

La routine power ha un singolo stato ed esegue una singola operazione, ad ogni tick controlla attraverso il flag "MasterOnOff" se la presa elettrica deve ricevere corrente o meno, ed agisce di conseguenza sul suo dispositivo.

Dispositivo utilizzato

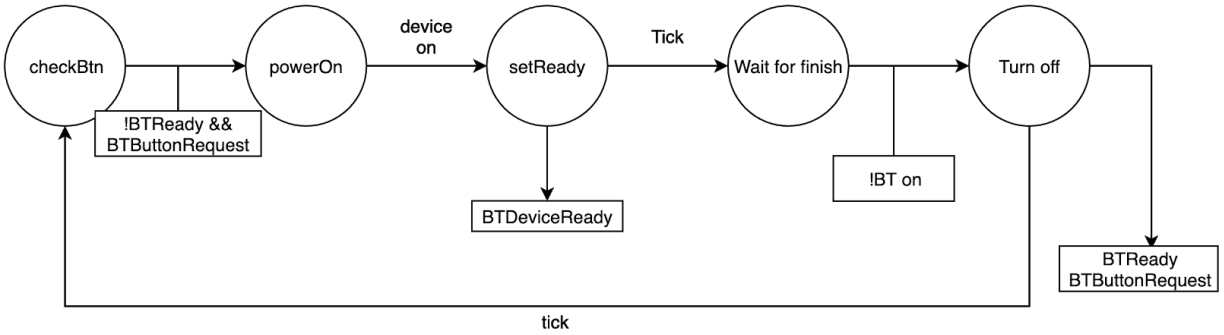


In questa routine viene utilizzato un modulo relè di corrente KY-019 da 5V.
E' stata creata una classe generica virtuale PowerRelay con le funzioni di turnOn e turnOff che vengono implementate nella classe KYPowRel passando valori HIGH o LOW a seconda al pin dello switch.

Routine Bluetooth Init

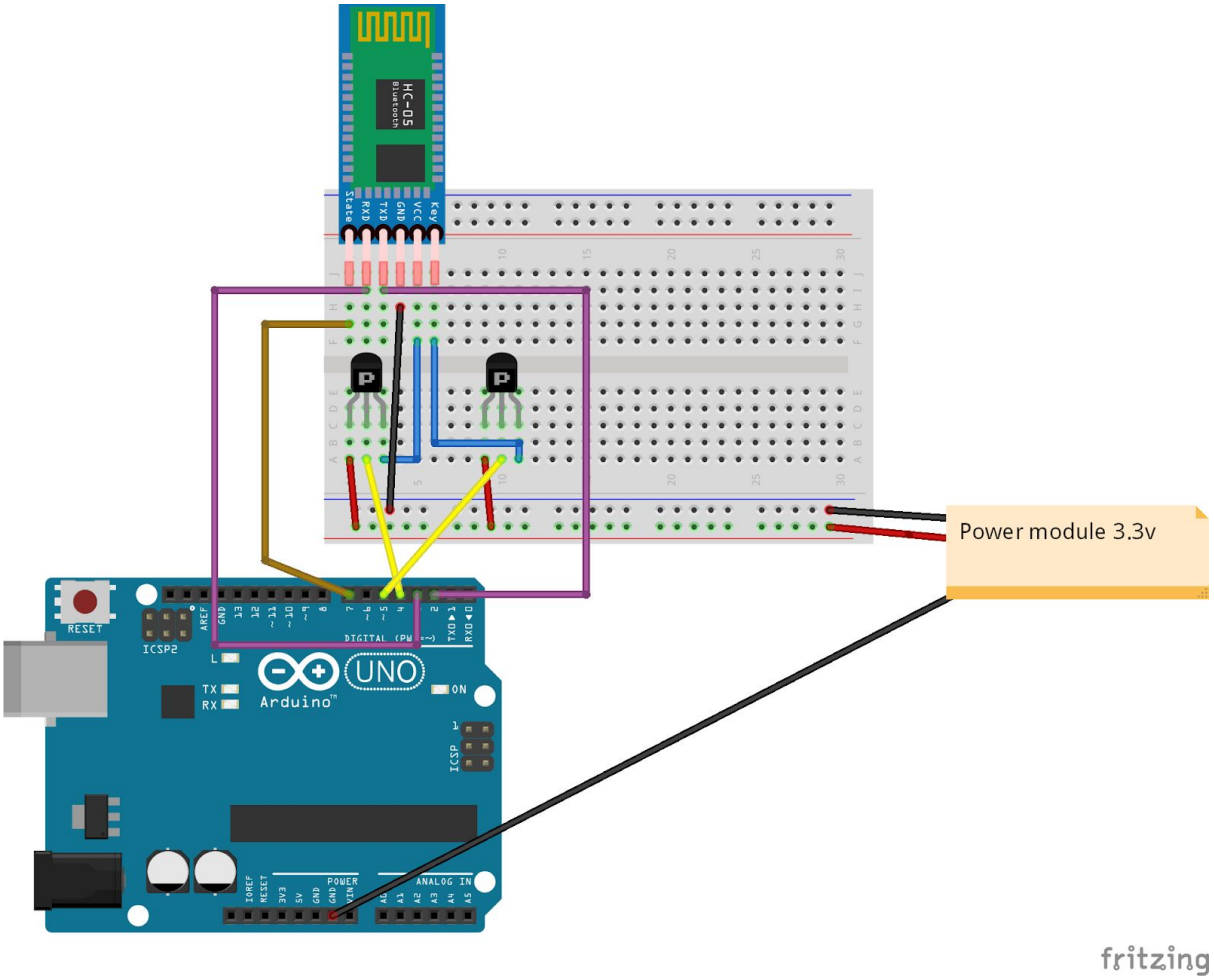
Questa routine si occupa di accendere e impostare correttamente il dispositivo Bluetooth al rilevamento della pressione del bottone di pairing. Una volta che il dispositivo è accesso aspetta fino al suo spegnimento. Allo spegnimento del dispostivo riporta i flag globali riferiti al Bluetooth allo stato originale e ritorna in ascolto

Routine



Questa routine richiama le funzioni del dispositivo Bluetooth associato.
E' stata creata una classe generica che supporta diverse funzioni, che sono poi state implementate nella classe BT05.
All'inizio della FSM la routine aspetta che che il bottone di pairing bluetooth sia premuto. Se rileva che il bottone è stato premuto e che il dispotivio Bt non è già pronto, per evitare i casi di errore, allora passa allo stato powerOn.
Durante questo stato la routine richiama le funzioni del dispositivo di powerOn e Setup.
La funzione di setup è implementata come un'ulteriore macchina a stato finito a due fasi, che verrà illustrata nella descrizione del dispositivo. Una volta che il dispositivo è pronto la routine imposterà le flag corrisponenti e inizierà lo stato di attesa. Resterà in questo stato finchè il dispositivo Bluetooth non avrà finito la sua funzione.
Una volta che viene rilevato che il dispositivo deve essere spento, la routine lo spegne e resetta le flag allo stato iniziale per permettere il prossimo ciclo, per poi ritornare allo stato iniziale di checkBtn.

Dispositivo utilizzato



Come anticipato precedentemente è stata creata una classe generica virtuale di un “Bluetooth Device” con diverse funzioni implementabili:

- bool powerOn(): la quale eseguirà tutte le operazioni necessarie per accendere il dispositivo prima di ritornare true
- bool setup(): la quale eseguirà tutte le operazioni necessare per rendere il dispositivo pairable prima di ritornare ture
- bool powerOff(): la quale eseguirà tutte le operazioni necessarie per spegnere il dispositivo prima di ritornare true
- void resetFSM(): la quale riporterà tutte le FSM del dispositivo al loro stato iniziale
- bool isPaired(): la quale resituità se il dispositivo attuale è connesso ad un altro dispositivo o no
- bool handleMessages(): la quale controllerà il buffer seriale del dispositivo e si comporterà di conseguenza ai messaggi ricevuti.

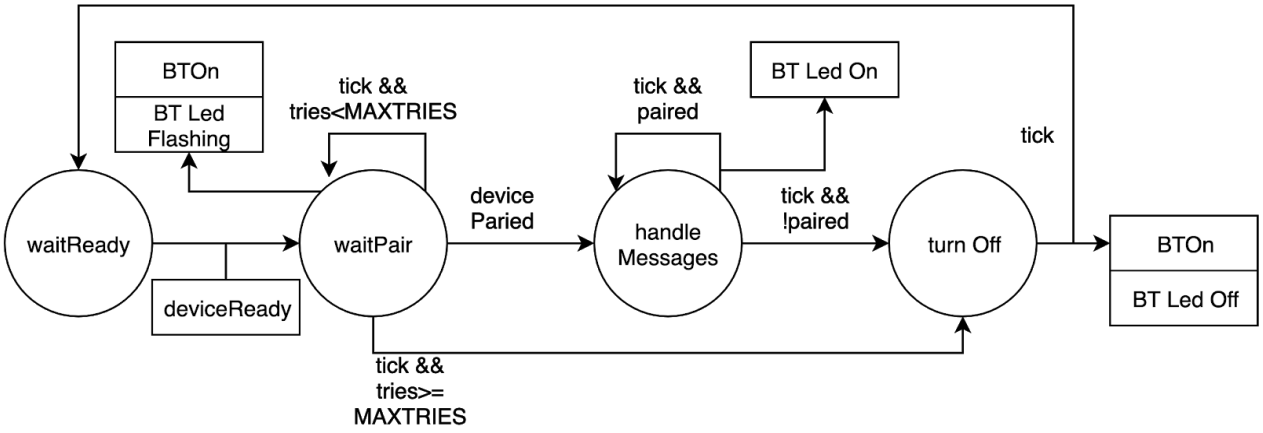
Per l’implementazione della routine di inizializzazione sono interessanti le implementazioni da parte del dispositivo BT05 delle funzioni powerOn, setup e powerOff.

Una decisione presa per risparmiare energia durante l'utilizzo del sistema è stata quella di spegnere completamente il dispositivo Bluetooth quando esso non serve, e di accenderlo solo su richiesta tramite la pressione di un pulsante. Per implementare questa decisione è stato necessario l'uso di un transistor che controlla il pin VCC del dispositivo Bluetooth.
E' per questo motivo che per il dispositivo Bluetooth sono implementate due routine che vanno in parallelo. Nella routine di Inizializzazione, a differenza della routine di funzionamento, vengono gestiti anche i transistor che forniscono energia al dispositivo.
Infatti l'unica azione che esegue l'implementazione della funzione di powerOn è quella di fornire un valore HIGH al transistor che controlla la VCC del dispostivo, e la funzione powerOff fa il contrario.
La funzione di setup invece implementa una piccola FSM interna. L'unico modo per controllare se il dispositivo BT05 è in stato pairable è fare una chiamata "AT+STATE?" verso il dispositivo, che però non ha attivata di base la modalità di AT. Quindi nel primo stato della funzione setup viene passato un valore HIGH ad un secondo transistor che controlla il pin di modalità del dispostivo, facendolo passare in modalità AT. Il secondo stato continuerà ad eseguire una richiesta "AT+STATE?" e a cercare tramite il WordFinder la parola "PAIRABLE". Una volta trovata uscità dalla modalità AT e informerà che il dispositivo è pronto ad essere usato.

Bluetooth Routine

Questa routine si occupa di scambiare messaggi tra il dispositivo Bluetooth e il dispositivo mobile associato.

Routine



La routine inziierà nello stato waitReady dove aspetterà che il flag "Device Ready" diventi vero, questo succederà quando la routine BtInit avrà finito di eseguire le sue operazioni. Una volta che il dispositivo è pronto si imposterà il led del Bluetooth in modo che sia in modalità flash e impostertà il flag BTOn a true per segnalare che il dispositivo è accesso ed operativo, per poi iniziare ad aspettare che il dispositivo sia in modalità Pair. Questo si controlla utilizzato l'implementazione di isPaired() che nel caso di BT05 consiste nel controllare il pin di stato; se esso restituisce un valore alto allora siamo collegati.

Se questo stato viene eseguito MAXTRIES volte senza mai rilevare un collegamento verrà eseguito lo stato di spegnimento del dispositivo.
Se invece viene rilevato che il dispitivo è connesso si entrerà nello stato handleMessages. All'entrata di questo stato il Led Bluetooth verrà impostato per essere accesso in modo fisso e si inizierà ad attivamente ascoltare.
Il CommandParser nell'implementazione del BT05 ha i seguenti comandi:

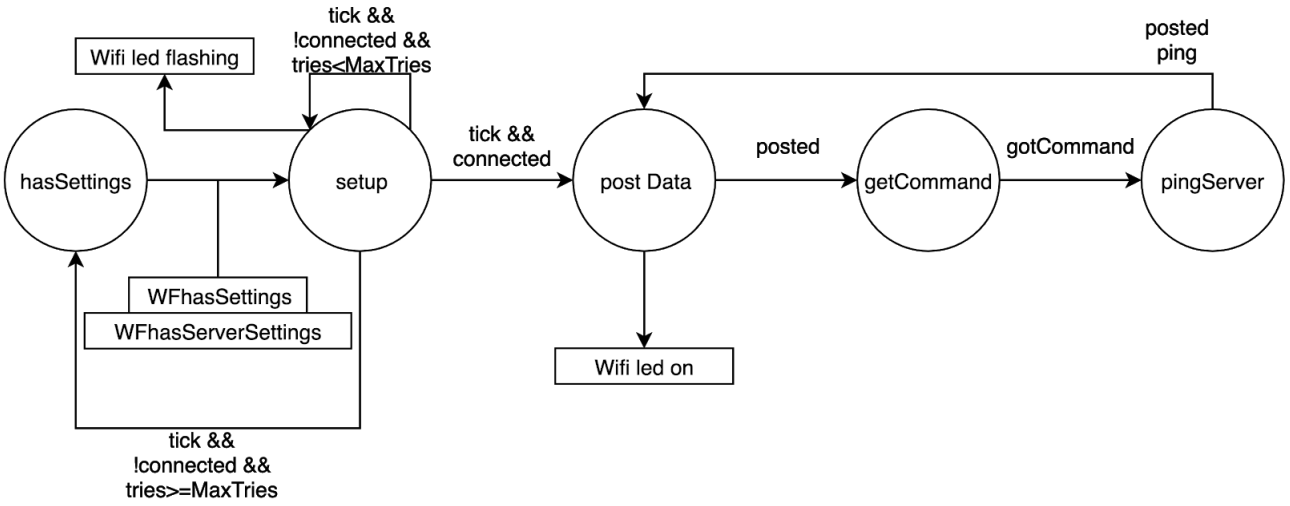
- WS=<ssid>;<pswd>;- : Usato per le imposazioni del wifi
- DSW=<wattage> ;- : Usato per le impostazioni di salvavita
- SS=<serverip>;<serverport>;<DevID>;<DevKey>;- : Questo e i precendenti comandi prendono i parametri e li inseriscono nei corrispettivi setting
- MM=<onoff>;- : Imposta il flag che farà passare o bloccherà la corente alla presa elettrica
- MF=<fastmode>;- : Avvia la "FastMode", cioè una modalità di lettura esclusiva al dispositivo Bluetooth in cui si riceverà la lettura del sensore ogni secondo.

Una volta scollegato dal collegamento si passerà allo stato turnOff che imposterà il flag BTOn a falso e spegnerà il Led Bluetooth.

Routine Wifi

Questa routine gestisce le comunicazioni con il dispositivo wifi, inizia con la connessione al wifi impostato, per poi comunicare con il server impostato.

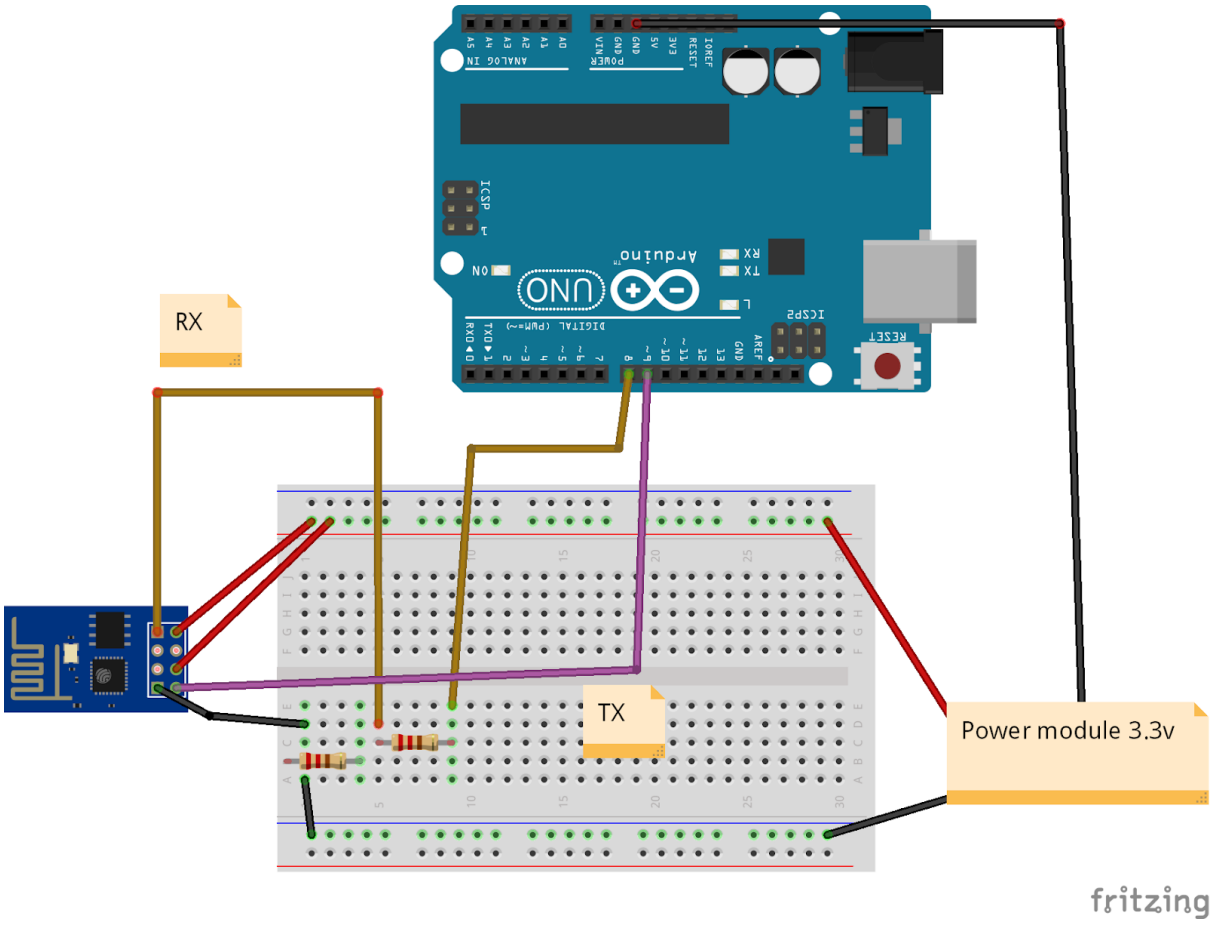
Routine



La routine inizia nello stato hasSettings, dove controlla se il dispositivo ha sia i settaggi della rete wifi che quelli del server. Se li ha entrambi fa partire la vera routine.
Nello stato di setup verrà chiamato il metodo setup() del dispositivo wifi associato, il cui obiettivo sarà quello di collegarsi alla rete wifi specificata. La fuzione può anche generare uno stato di errore in caso la connessione non va a buon fine, verrà tentata la connessione MaxTries volte prima di interrompere i tentativi e segnalare uno stato d'errore. Durante l'esecuzione di questo stato inoltre si imposterà lo stato del led wifi in modalità flashing in modo da indicare che sta avvenendo la procedura di setup.
Se la connessione ha successo la routine entrerà nel suo loop principale, impostando anche il Led Wifi come acceso.
Nel primo passo del loop principale si chiamarà il metodo postData() che invierà le letture del dispositivo al server

Il secondo passo chiama il metodo getCommand che interrogerà il server per i comandi da eseguire, e se li riceve li eseguirà
Il terzo passo è quello di chiamare il metodo pingBack() che farà sapere al server che il comando è stato ricevuto.

Dispositivo utilizzato

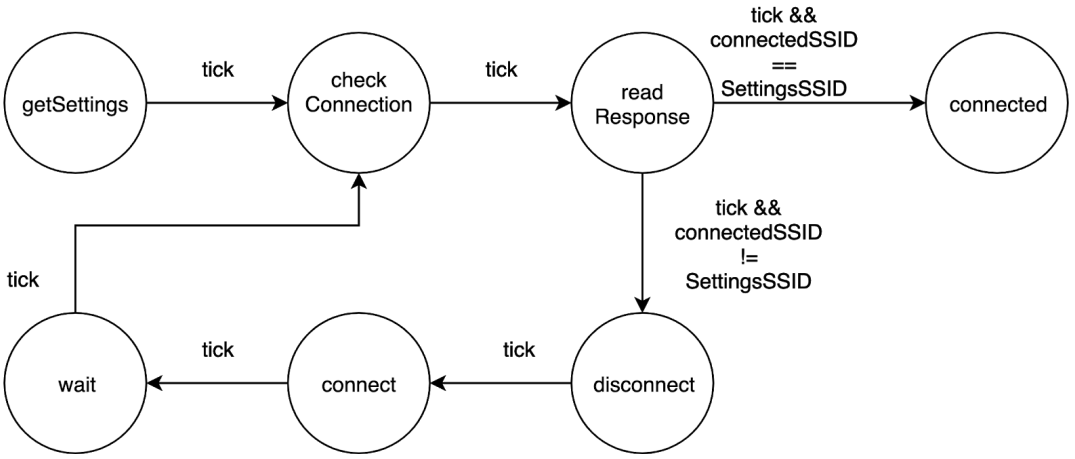


In questa implementazione viene utilizzata una scheda ESP8266-01.
E' stata creata una classe virtuale generica WifiDevice con i seguenti metodi:

- int setup() : utilizzata per connettersi alla wifi
- int postData() : utilizzata per mandare valori al server
- bool getCommands() : utilizzata per ricevere valori dal server
- bool pingBack() : utilizzata per spedire conferme al server
- void resetFSM(): utilizzata per resettare le eventuali FSM interne delle implementazioni.

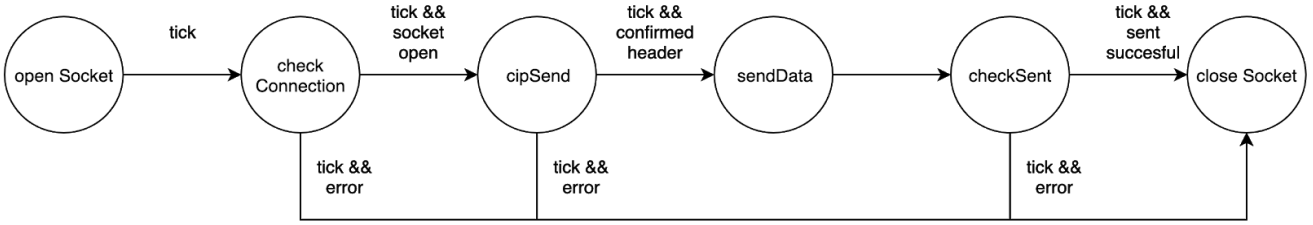
I vari metodi sono stati implementati utilizzando FSM interne. Ogni metodo continua a restituire un falore "false" o "-1" durante gli stati in cui sta eseguento operazioni, e restituisce un valore "true" o un codice di stato negli stati in cui la sua operazione è terminata. Si è scelto questo approccio perché visto che molte operazioni sono asincrone si può utilizzare il tick della routine come modo per aspettare un'eventuale risposta senza bloccare tutto il sistema.

Implementazione del metodo Setup



Il metodo setup inzia con lo stato GetSettings, nel quale imposta le varie variabili di stato correttamente.
Passa poi allo stato “check Connection” in cui richiede al dispositivo wifi a quale è connesso, con il comando “AT+CWJAP?”, e imposta come parola da trovare il SSID che ha il dispositivo nei suoi Setting
Nello stato successivo controlla la risposta del dispositivo, se è presente la wifi desiderata allora il dispositivo è connesso e il metodo restituisce “1”, altrimenti il dispositivo non è connesso a nessuna wifi, oppure è connesso ad una wifi che non è quella desiderata, quindi il metodo ritornerà “-1” per segnalare un errore.
Quindi si manda il comando di disconnessione , “AT+CWQAP”, per poi mandare il messaggio di connessione “AT+CWJAP=<ssid>,<psw>”. Si passa quindi nello stato wait dove resterà bloccato per un certo numero di tick impostato, per poi tornare allo stato checkConnection.

Implementazione del metodo PostData



Il metodo postData incomincia con l’apertura del socket con il server specificato nei settings, nello stato successivo controlla che il socket sia stato correttamente aperto, se non lo è passa direttamente all’ultimo stato.
Se invece il socket è aperto il metodo dice al dispositivo la quantità di dati da inviare utilizzando il comando “CIPSEND” , e controlla che il dispositivo ritorni un messaggio di conferma invece che di errore.
Una volta ricevuto il messaggio di conferma il metodo crea una richiesta POST al server e la spedisce.
Resta poi in ascolto fino alla ricezione della risposta di conferma da parte del server, per poi chiudere la connessione

Implementazione del metodo getCommand

La FSM del metodo getCommand è identica a quella del PostData, differisce solamente nel fatto che la richiesta è di tipo GET, e invece di solamente confermare la risposta prende i valori ricevuti ed esegue operazioni di conseguenza.

Il commandParser dell'implementazione del dispositivo wifi supporta i seguenti comandi:

- DSW=<wattage>;- : Usato per impostare il wattaggio massimo del salvavita
- MM=<onoff>;- : Usato per accendere o spegnere la presa di corrente
- SUN=;- : Rimuove tutti i settaggi della connessione dal dispositivo, disassociandolo dal server

Come primo parametro in questi comandi c'è sempre l'ID del comando, questo verrà usato per non eseguire lo stesso comando più volte.

Implementazione del metodo pingBack

La FSM del metodo getCommand è identica a quella del PostData, differisce nel messaggio spedito. Mentre PostData invia il valore letto, pingBack invia al server l'ID dell'ultimo comando ricevuto, in modo da confermare al server che il comando è stato eseguito.

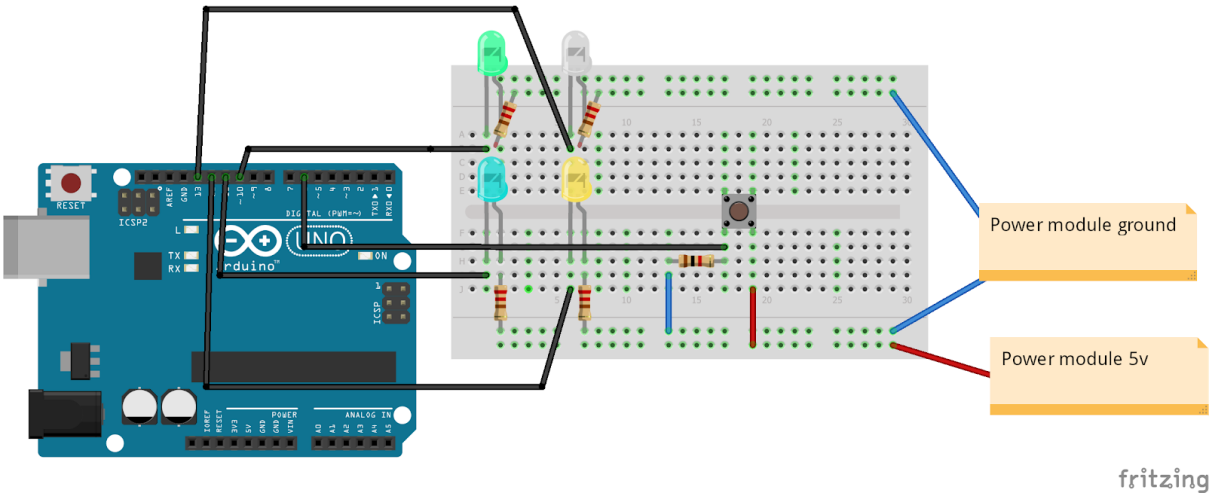
Alcune precisazioni sul funzionamento del Bluetooth e del Wifi

- Le comunicazioni con entrambi i dispositivi sono gestite tramite le librerie Software Serial, che però supportano una singola comunicazione seriale alla volta. La decisione di interrompere le routine Wifi durante l'esecuzione di quelle Bluetooth è stata presa per questo motivo
- Il metodo getCommand è l'unica operazione per cui la mancata esecuzione entro il suo deadline porta al malfunzionamento del sistema. Mancare il deadline porta infatti al riempimento del buffer del dispositivo ESP, quindi una perdita di dati, tra cui i comandi da eseguire. Per questo durante l'esecuzione del metodo getCommand viene impostata una flag che avvisa il sistema che non deve eseguire operazioni che potrebbero impiegare una quanità significativa di tempo. In questa implementazione significa interompere le letture da sensore finchè il messaggio non è stato ricevuto.

Routine UX

La routine UX gestisce l'interfacciamento con l'utente. E' quella che controlla i led e avvisa quando il bottone è stato premuto. Ha un singolo stato in cui controlla la flag dei rispettivi led e agisce di conseguenza, per poi controllare lo stato del bottone.

Dispositivi utilizzati



E' stata creata una classe virtuale per un led generico che fornisce le funzioni di accensione, spegnimento o scambio di stato. E' stata creata anche un'implementazione che agisce di conseguenza impostando i valori dei pin a HIGH o LOW.
E' stata creata una classe virtuale anche per un bottone generico che fornisce la funzione "isPressed()". E' stata creata e usata l'implementazione di questa classe.
I led del dispostivo avranno le seguenti funzioni:

- Led Verde = Device Led , segnala che il dispostivo è stato avviato
- Led Bianco = Power Led, segnala che la presa elettrica sta fornendo elettricità
- Led Blu = Bluetooth led, segnala che il dispositivo sta eseguendo comunicazioni Bluetooth
- Led Giallo = Wifi Led, segnala che il dispositivo sta eseguendo comunicazioni wifi

Difficoltà incontrate

La principale difficoltà incontrata durante lo sviluppo è stata la gestione della memoria dinamica visto che si è fatto un uso significativo di stringhe data la necessità di creazione di richieste POST e GET.
Questo problema è stato risolto mantendo queste stringhe statiche in memoria Flash e copiandole nella memoria dinamica quando richieste, tramite le funzioni della libreria "<avr/pgmspace.h>"
Un'altra difficoltà incontrata,e non risolta, è l'inaccuratezza del sensore usato per misurare la corrente. Nonostante aver implementato tutto seguendo i manuali forniti, i valori del dispositivo sono spesso poco precisi e non sempre accurati.

App android

E' stata realizzata una semplice applicazione Android che permette il pairing con un dispositivo e l'invio di comandi.
E' strutturata in tre activity principali:

- Settings Activity: Usata per inserire l'indirizzo del server e il wifi che si vuole impostare alla presa. Prima di salvare le impostazioni viene mandato un ping all'indirizzo specificato per verificarne la correttezza
- Main Activity: L'attività che viene avviata insieme all'applicazione, permette la ricerca dei dispositivi bluetooth nella zona. Una volta conclusa la ricerca verrà mostrata una

lista di dispositivi trovati. Se l'utente premerà sul dispositivo desiderato inizierà la procedura di pairing e connessione, per poi passare all'activity dei comandi

- Command Activity: L'attività che viene visualizzata alla connessione con il dispositivo, permette all'utente di mandare i diversi comandi specificati al sistema. Una parte interessante di questa attività è la creazione dei settaggi da mandare al sistema. L'applicazione richiederà al server la creazione di un nuovo dispositivo, e come risposta riceverà l'id e la chiave appena generati, che a sua volta manderà al sistema.

Server NodeJS

L'architettura del server è stata realizzata utilizzando la tencologia Node JS.
Avviando il server e collegandosi all'indirizzo IP della macchina si potrà accedere ad un semplice sito internet in cui si potranno visionare tutti i dispositivi collegati , vedere le loro letture e mandargli dei semplici comandi.
Si potrà inoltre visionare la media delle letture di tutta l'abitazione e impostare il salvavita generale.
Tutto il sistema di gestione di dispositivi è completamente dinamico e quindi scalabile.
Il server fornisce diversi API Endpoint a cui i vari dispositivi faranno riferimento.
Il server inizia con 0 dispositivi associati, alla creazione di un dispositivo attraverso l'apposito API endpoint verrà creato un file che conterrà gli storici delle letture per quel dispositivo e verrà aggiunto alla lista di dispositivi riconosciuti.
Ogni dispositivo riconosciuto ha una coppia di valori "ID-Chiave". L'id è un identificativo numerico per semplificare la creazione della parte front end, la chiave invece è un valore di 4 cifre randomicamente generato per avere un'ulteriore conferma sull'autenticità della chiamata.
Se viene effettuata una chiamata al server senza senza una coppia ID-Chiave tra quelle riconosciute essa verrà ignorata e verrà mandato in risposta un comando di Unpair, in caso la chiamata da ignorare è generata da un dispositivo con delle impostazioni non più valide.

Conclusioni

Questo progetto è stato una fantastica opportunità per interfacciami con il mondo dei Sistemi Embedded e Internet of Things, ed è stato essenziale per capire molti concetti illustrati nella teoria del corso

Possibilità di espansione

Alcune idee per sviluppare ulteriormente il progetto possono essere

- L'utilizzo di una singola presa di corrente per fornire elettricità e per alimentare Arduino
- L'utilizzo di un sensore di voltaggio insieme a quello di corrente per avere delle letture più precise
- La creazione di un dispositivo stand-alone che faccia da server plug-and-play
- La possibilità di vedere quali impostazioni sono presenti sul dispositivo