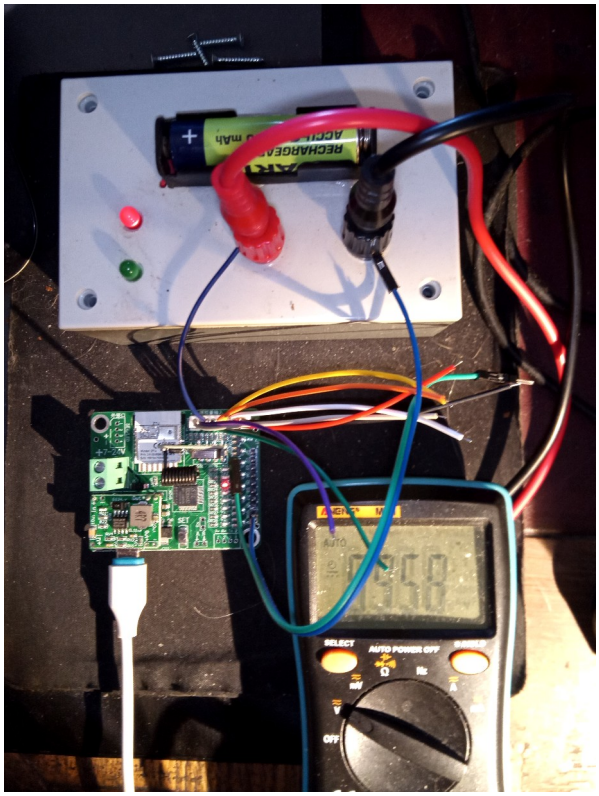
Tuya: NiMH Battery Tester 2024

Una potenzialità mi ha sempre attirato: la possibilità di creare strumentazione usando i device Tuya e il contributo di IoTwebUI. Ecco il primo tentativo in questa direzione: un NiMH BatteryTester, un semplice progetto che permette testare le possibilità e i limiti di questa strategia.



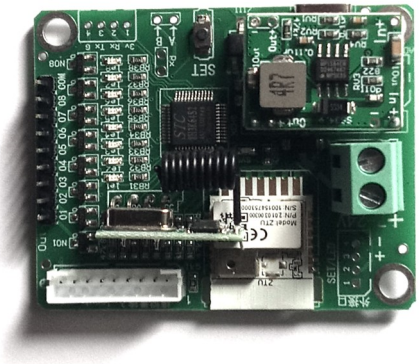
Il problema

Le batterie ricaricabili NiMH con l'uso degradano le loro caratteristiche, e muoino dopo 2 o 3 anni. Ne ho una discreta collezione, ma non è semplice caratterizzarle: individuare quali siano ancora valide, e quali invece siano distrutte dall'uso (e maluso).

Anni fa progettai e costruii uno 'scarica-batterie': una batteria NiMH è scaricata su un carico fisso a circa 1A, e un comparatore a µP interrompe la scarica raggiunto un valore minimo di sicurezza (0.9V sotto carico). Il progetto è molto 'ecologico' perchè non usa batterie proprie, ma l'energia residua della batteria da scaricare, fino a 0.6 V. Per il progetto completo, vedere https://github.com/msillano/e3DHW-PMS/blob/master/applications/Battery%20discharger_it.pdf.

Gli usi principali di uno 'scarica-batterie' sono la scarica completa prima della ricarica, per ridurre l'effetto 'memoria', e la creazione di battery pack, con batterie tutte allo stesso livello di carica (minimo).
Questo apparecchio, da solo, non permette di selezionare le batterie neanche con una logica 'passo / non passo': non effettua alcuna misura.

Progetto Tuya 2024



La scoperta di una scheda Tuya compatibile con ADC a bordo (TY-08Z) mi ha portato a riprendere quel progetto e a completarlo con tre obiettivi:

- Tracciamento del grafico di scarica della batteria.
- Calcolo della capacità (mAh) della batteria.
- Uso di Tuya in una applicazione strumentale.

(Scheda Tuya di difficile reperibilità: vedi post https://www.facebook.com/groups/tuyaitalia/permalink/1499620787338936/)

Procedura:

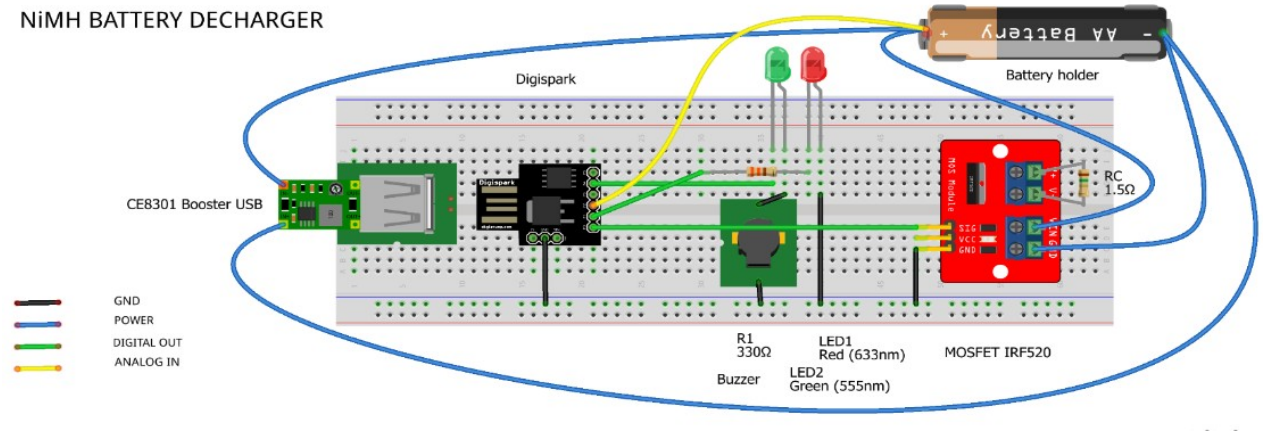
a) Carico al massimo le batterie, usando un carica batterie al NiMH marca Adeaska, e accettando il livello 'FULL' deciso dal caricabatterie.

b) Scarico la batteria con l'apparecchio scarica-batterie, con una unità ADC che misura la tensione della batteria ed invia i dati a TuyaCloud.
Durata: 1h..2h proporzionale alla capacità della batteria, corrente di scarica circa 1A.

c) Utilizzo di IoTwebUI per:

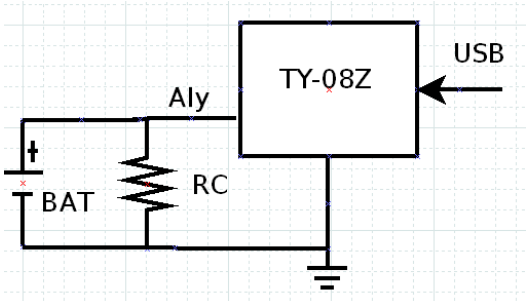
- Esportare le misure RT (V, T, mAh) su un file di log CSV, da cui ottenere un grafico su uno spreadsheet.
- Avere un'interfaccia utente grafica specializzata per una comoda gestione.

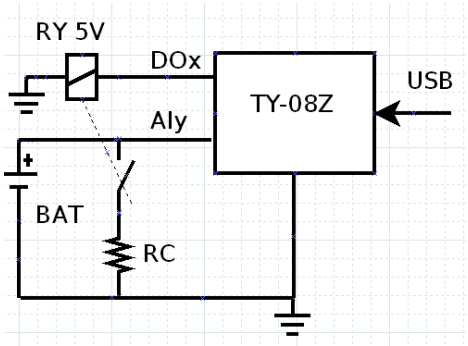
PRO
1. L'uso di moduli (standalone o per Tuya) semplifica molto l'HW: i componenti discreti sono in tutto 2 led, 2 resistori !
2. Anche il FW ed il SW applicativo beneficiano di un approccio modulare.
3. Ha una funzione di test e verifica delle effettive potenzialità in questo ambito di Tuya + IoTwebUI.



CONTRO
 Una soluzione monolitica (e.g. con Arduino) è forse concettualmente più semplice, ma non sarebbe IoT, quindi non riusabile, nè integrata con Tuya, nè modulare.

2024: L'obiettivo del 'battery decharger' è quello di individuare con certezza il livello basso della batteria NiMH. Usando una scheda ADC TY-08Z, si può affidare questo compito al SW di gestione, e quindi il 'battery decharger' standalone diventa superfluo.
In una nuova realizzazione, si può usare uno di questi schemi, molto semplificati rispetto alla soluzione attuale:



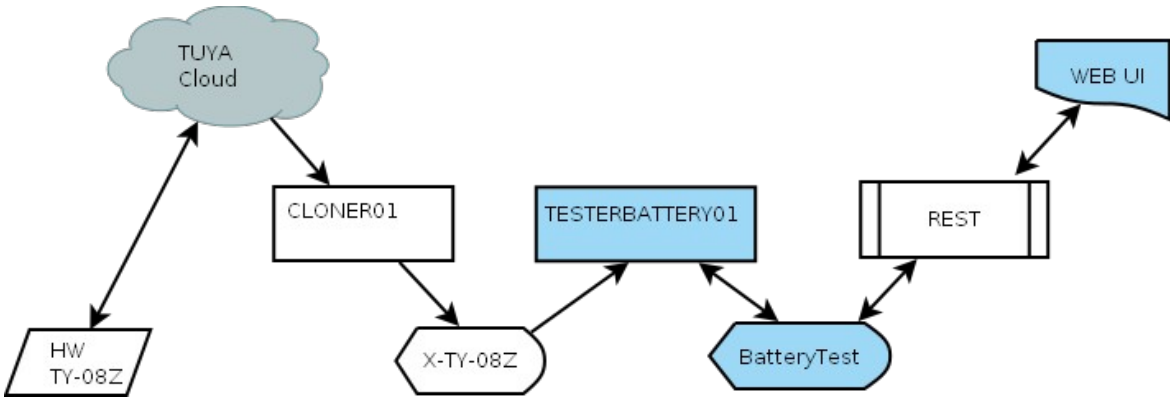


note: *Il secondo circuito prevede il distacco automatico della batteria a fine scarica.*

- *AIy rappresenta uno degli 8 ingressi analogici (io ho usato il 3)*
- *DOx è una delle 8 uscite digitali, che possono pilotare direttamente un relay a 5V*
- *BAT è la batteria Ni-MH sotto test*
- *RC è la resistenza di carico, ho usato 1 Ω 5W*
- *Il circuito può essere moltiplicato fino a gestire 8 batterie contemporaneamente!*

Per la misura di V si possono usare due strategie: interrompere la scarica per 0,2-0,4 s e in questo intervallo misurare la tensione. Oppure misurarla con il carico.
Considerando l'obiettivo di questo dispositivo, essenzialmente comparativo, si è scelta la seconda, più semplice, soluzione.

Architettura SW



Data processing (model):

HW TV-08Z: device hardware Tuya standard, salva i propri dati nel **Tuya Cloud**.
I dati che interessano (valore di Num3 in Smartlife) non sono disponibili tramite IoTwebUI, che legge Tuya Cloud usando l'API "Query Devices in Home"
Quindi si dve creare un x-device mirror *più ricco di informazioni rispetto TY-08Z*

CLONER01: funzione (MACRO) che crea una *x-device mirror* (X-TY-08Z), leggendo i dati da Tuya Cloud ed usando ora l'API "Query Properties" che fornisce ogni dettaglio delle proprietà di un singolo device *(modulo universale).*

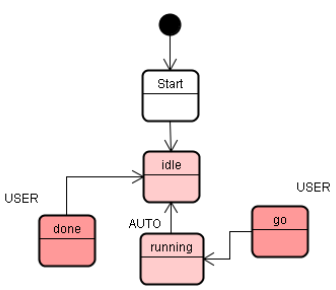
X-TY-08Z: x-device clone: contiene tutti i dati disponibili di TY-08Z, tra cui il valore del ADC, con 'code' *value_13*.

Backend processing (presenter):

TESTERBATTERY01: funzione (MACRO) che crea una *x-device* (BatteryTest), con tutti i dati necessari per l'applicazione, in particolare per WEB-UI. Questa funzione contiene tutta la logica necessaria (*modulo specializzato per questo progetto*)
Richiede una taratura una tantum, effettuata misurando 2 punti (4 valori) con X-TY-08Z.

Interface:

BatteryTest: *x-device di interfaccia* verso utenti e verso i frontend custom. Presenta i dati ed accetta comandi usando l'attributo *step*, che rapresenta gli stati dell'automa gestionale (*start - idle - **go** - running – **done**): go* e *done* sono aggiornati dall'esterno come START – STOP

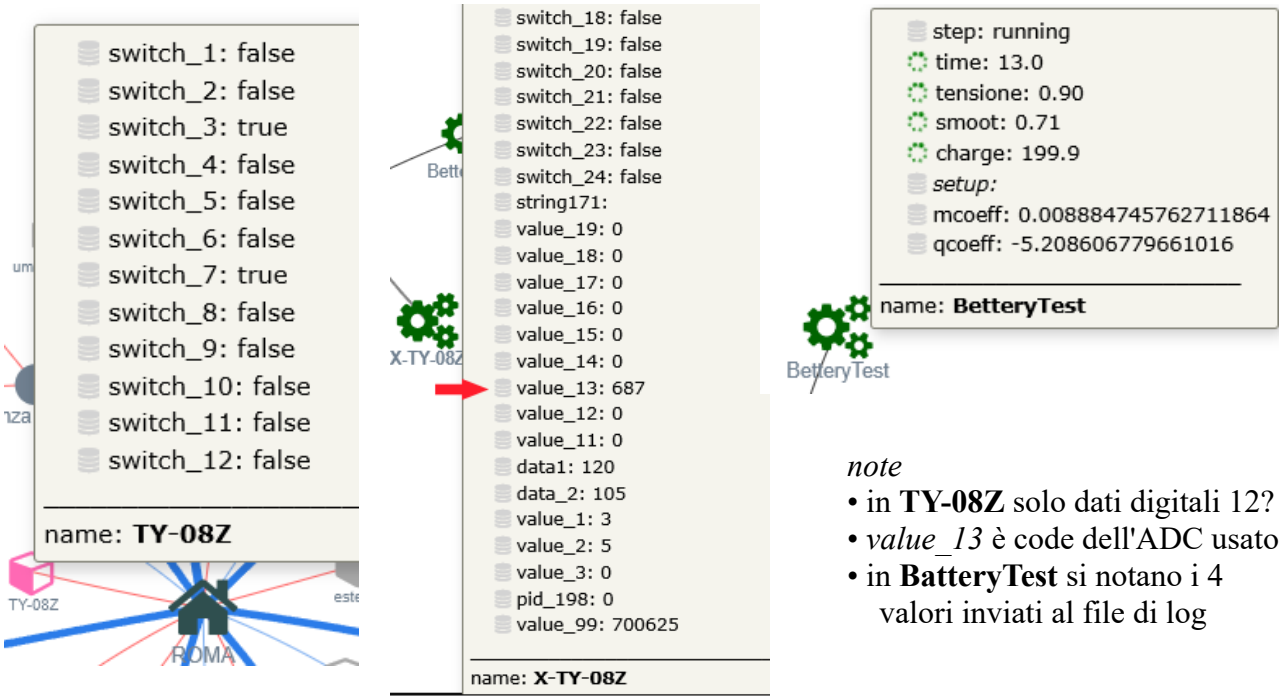


Front End (view):

REST; estensione di IoTwebUI che offre accesso a device e comandi via REST

WEB-UI: Interfaccia grafica WEB, presenta i dati ed invia i comadi utente a *BatteryTest,*

Ecco come si presentano i tre device interessati in IoTwebUI, con i pop-up che mostrano tutte le proprietà



note

- in **TY-08Z** solo dati digitali 12?
- *value_13* è code dell'ADC usato
- in **BatteryTest** si notano i 4 valori inviati al file di log

Vantaggi di questa architettura

- Il principale vantaggio è la modularità: la scomposizione in x-device e web interface è molto naturale e funzionalmente corretta. Le x-device sono ottime interfacce tra i vari processi. Tutto ciò semplifica e velocizza lo sviluppo.
- Il secondo vantaggio è la semplicità della composizione (o derivazione in senso OOP) di device: ogni device Tuya ed ogni x-device può essere riutilizzata da un'altra x-device!
- Terzo vantaggio è la possibilità di ri-uso anche multiplo: in questo esempio si possono avere più interfacce utente, anche contemporanemente, uguali o diverse, oppure un'altra applicazione che sfrutti i dati dello stesso circuito TY-08Z, etc.

Limiti dell'architettura

Latenza: Siamo in presenza di due cicli principali

- Tuya polling: min 20s per breve tempo. Per uso continuo > 80s (limiti di Tuya free). L'elaborazione locale (regole) avviene subito dopo un Tuya polling.
- Rest polling: 30s. Le singole chiamate sono distanziate 1s per distribuire il carico.

Inoltre:

- L'evoluzione degli stati può richiedere dei cicli per avanzare.
- In alcuni casi (carico, priorità, sospensioni...) il browser può saltare un ciclo.

Note: Ho inserito dei messaggi vocali per avere un feedbak più rapido dell'aggiornamento dell'UI.

E quindi un ambiente soprattutto adatto a processi lenti, tipo questo, con durate dell'ordine di ore.

REGOLE

Una volta installati gli addon, bastano queste 2 righe per creare e aggiornare gli x-device ogni loop:

```
CLONER01("12345abcdefg", "X-TY-08Z");
TESTERBATTERY01("BetteryTest");
```

e queste due regole sono usate come comandi

```
if(TRIGBYNAME("Start BattTEST")) SETXDEVICESTATUS("BatteryTest",'step','go');
if(TRIGBYNAME("Stop BattTEST")) SETXDEVICESTATUS("BatteryTest",'step','done');
```

Processo di sviluppo

Il flusso di sviluppo che ho seguito e che consiglio per analoghi progetti, si ispira alle tecniche XP (eXtreme Programming) ed alle 'best practices' OOP.

Step 1:

- *Si identificano il/i device che forniscono i dati iniziali (e.g. X-TY-08Z).*
- *Si installano le MACRO necessarie (e.g. CLONER01) in IoTwebUI (e.g. modificando IoTwebUI.html).*
- *Si aggiungono a IoTwebUI le* regole/rule *necessarie*

Step 2:

- *Si definiscono le caratteristiche dell'interfaccia verso le user interface, cioè le proprietà di un nuovo* x-device *(e.g. BatteryTest), inizialmente minimali.*

Step-3: Si implementa una funzione js, che sarà usata come MACRO *(e.g TESTERBATTERYY01):*

- *Come input ha i dati identificati in Step 1*
- *Come output crea e aggiorna l' x-device definito in Step 2.*
- *Viene eseguita dopo ogni loop che legge i dati da Tuya Cloud (analogo alla struttura del SW Arduino).*
- *Si presta bene un'architettura di automa a stati finiti.*

Step 4:

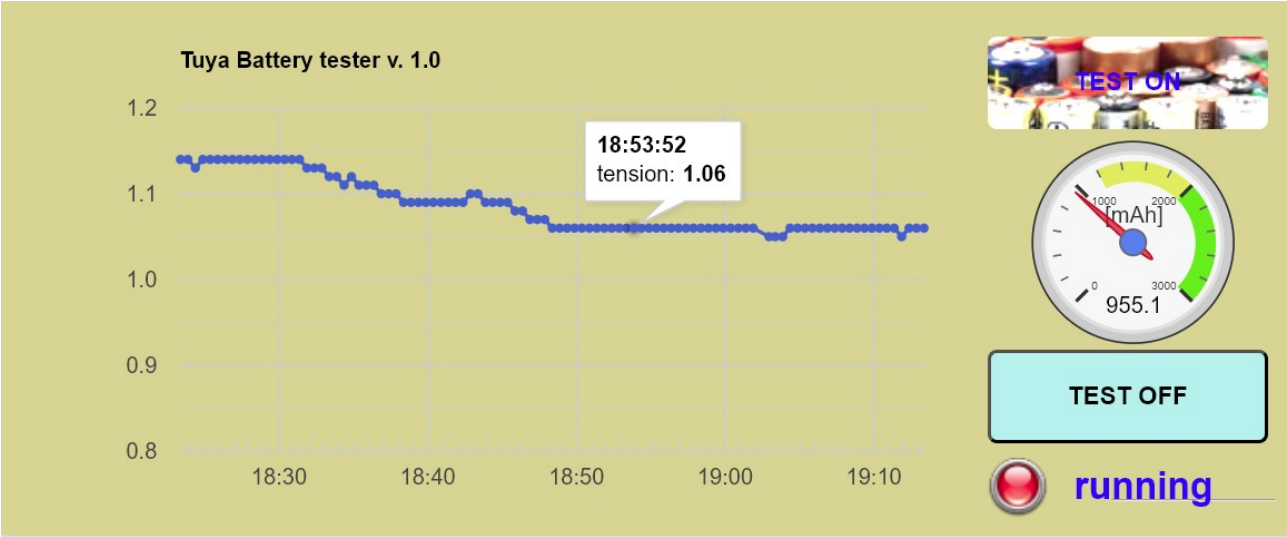
- *Si installano le MACRO necessarie (TESTERBATTERY01) in IoTwebUI (e.g. modificando IoTwebUI.html).*
- *Si aggiungono a IoTwebUI le* regole/rule *per creare e per intragire con il nuovo x-device: sono i comandi utente che saranno usati anche dalle UI.*
- *A questo punto il progetto è funzionalmente completo. Si può testare il progetto. I risultati si vedono con i pop-up di IoTwebUI.*
- *Si cicla da Step 2 a Step 4 con successivi affinamenti incrementali.*

Step 5:

- *Si possono realizzare una o più interfacce utente, che, a prescindere dal linguaggio, utilizzano il REST in polling per leggere i dati aggiornati dal x-device di interfaccia (e.g. BatteryTest) ed per eseguire i comandi, attivando le* regole/rule *di Step 4.*

Il pattern utilizzato è definito MPV: Model – Presenter – View. Presenta il vantaggio di poter sviluppare separatamente i due artefatti: Presenter e View.
Vedi: *https://github.com/msillano/IoTwebUI/blob/main/html/clima01-leggimi.md#pattern-mvp*

Risultati ver 1

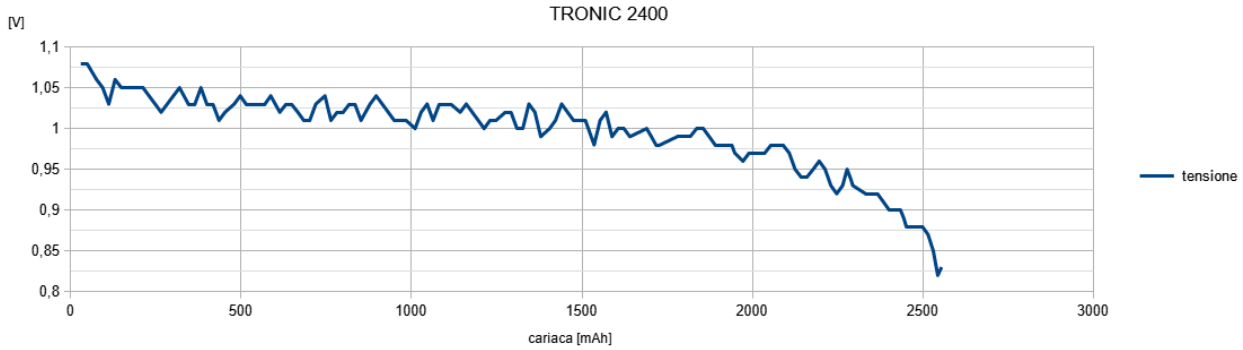


Questa è la prima interfaccia realizzata (terster01.html), catturata durante la scarica di una batteria Ni-MH da 2400 mAh.

I criteri per le batterie a Ni-MH da 2400 mAh sono (vedi meter):

0 – 1200 mAh	=> BAD
1200 mAh – 2000 mAh	=> USABILE (con precauzioni)
2000 mAh up	=> OK

Inoltre il log si presta a ulteriori elaborazioni offline con uno spredsheet:



Riferimenti:

IoTwebUI v. 2,2: https://github.com/msillano/IoTwebUI
REST v. 2.2: https://github.com/msillano/IoTwebUI/blob/main/RESTserver/LEGGIMI-REST22.md
TY-08Z: https://github.com/msillano/tuyaDAEMON/blob/main/devices/TY-08Z/TY_08Z_01.pdf
CLONER01: https://github.com/msillano/IoTwebUI/blob/main/addon/cloner01.js
TESTERBATTERY01:https://github.com/msillano/IoTwebUI/blob/main/addon/TestBattery01_leggimi.pdf
WEB-UI: https://github.com/msillano/IoTwebUI/blob/main/html/tester01.html