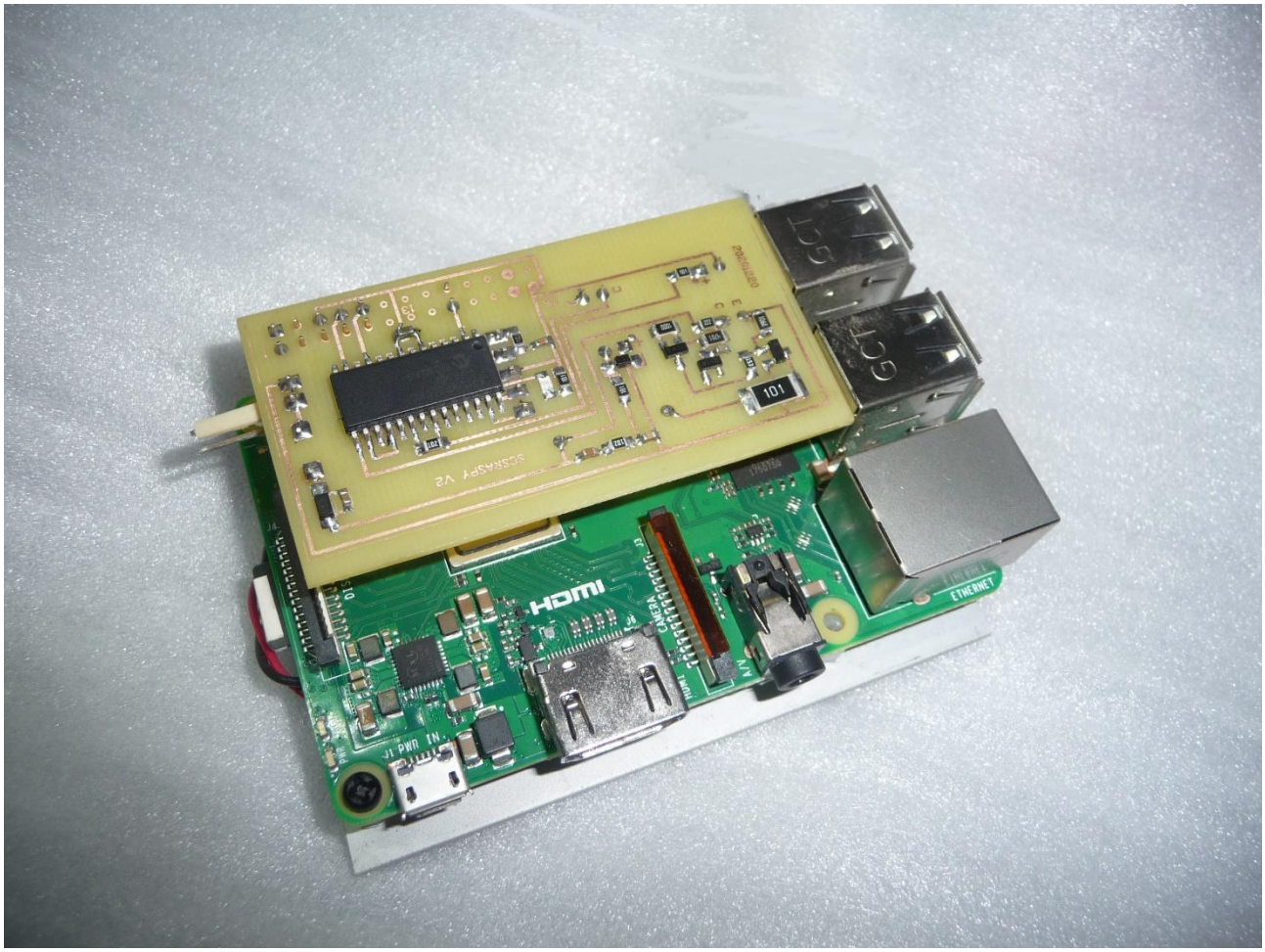
RASPY_SCSGATE V 80.8



SOMMARIO

INTRODUZIONE:

RASPY_SCSgate è un modulo di interfaccia tra il bus Konnex e RASPBERRY, disponendo di una serie di moduli software (c++) che ne consentono l'utilizzo; è un dispositivo amatoriale autocostruito e come tale privo di qualsiasi garanzia in merito al corretto funzionamento ed interfacciamento – SCS è un bus "proprietario" che non è lecito replicare con dispositivi commerciali senza le dovute autorizzazioni.

Esiste un isolamento galvanico tra la scheda ed il bus SCS (sono elettricamente isolati) garantito da due foto-accoppiatori con una tensione di isolamento superiore a 5KV.

Scopo del modulo è di consentire a dispositivi locali (raspberry) o esterni (computers, Android, alexa…) di ricevere e inviare messaggi sulla rete SCS (accendere e spegnere luci, tapparelle, ecc...) - l'interfaccia si riferisce SOLO ai moduli di automazione, non alla trasmissione dati (citofonia, video, ecc...).

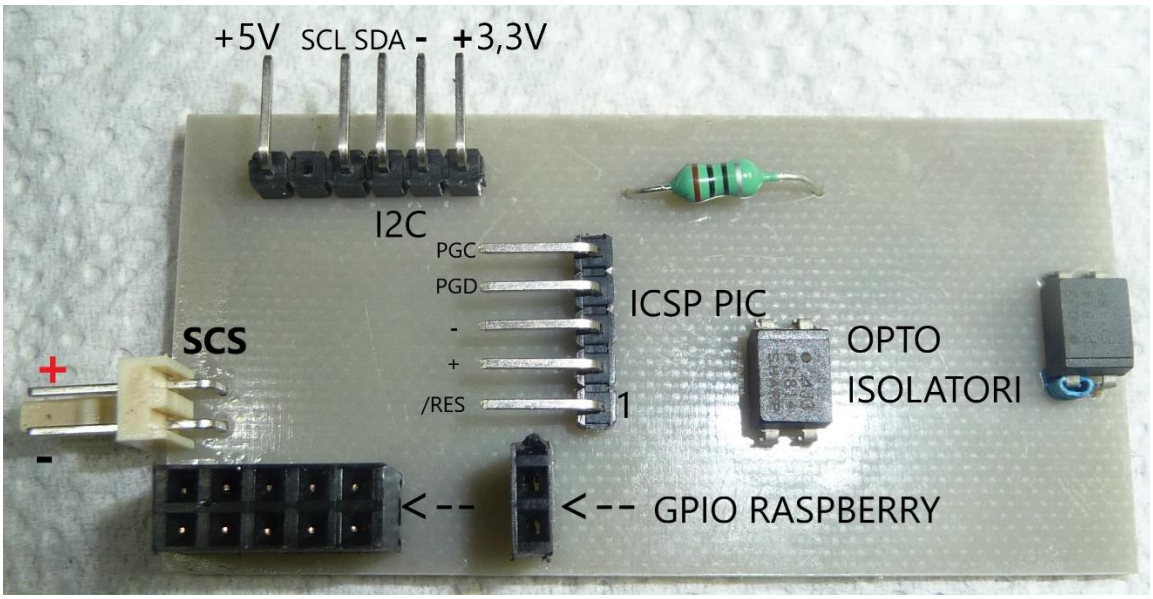
Il modulo è stato testato su di un bus scs bTicino. E' stato testato su moduli raspberry zero W, raspberry3B+ e raspberry 4.

Il firmware è aggiornabile sia tramite i programmatori standard "Microchip" con uscita ICSP (es. pickit3) sia attraverso un software apposito direttamente da raspberry.

Questo manuale deriva da un precedente documento relativo alla scheda ESP_SCSGATE. Se trovate in qualche immagine o frase il termine "esp_scsgate" sappiate che si riferisce comunque a RASPY_SCSGATE.

La differenza di questo documento rispetto alla versione 80.6 è unicamente dovuta all'aggiunta di alcuni paragrafi relativi all'utilizzo con arduino.

INTERFACCIA RASPY_SCSGATE: CONNESSIONI



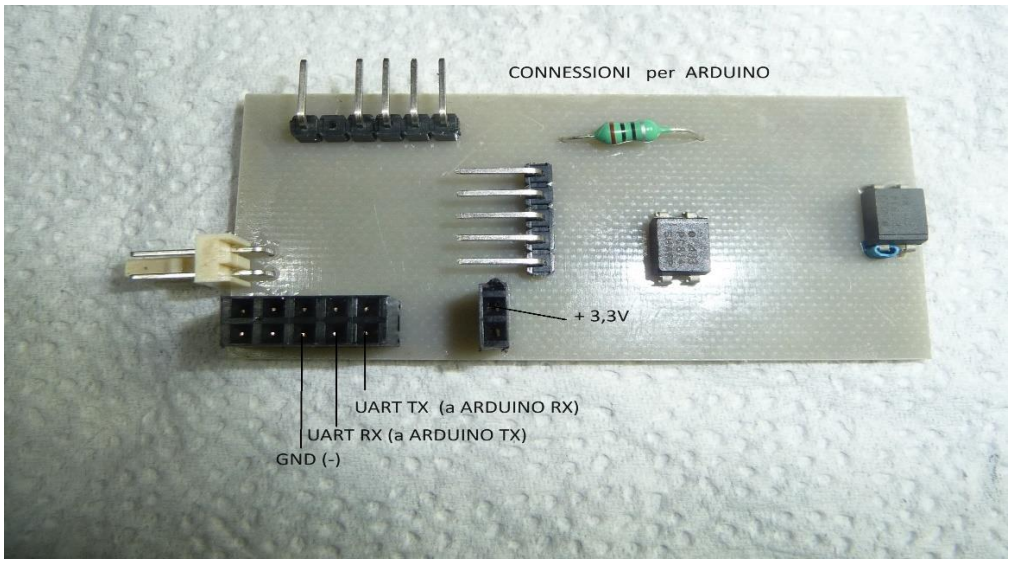
La scheda (RASPY_SCSGATE) ha 4 connettori:

- Il connettore di sinistra va collegato al bus SCS **rispettando le polarità indicate + e –** .

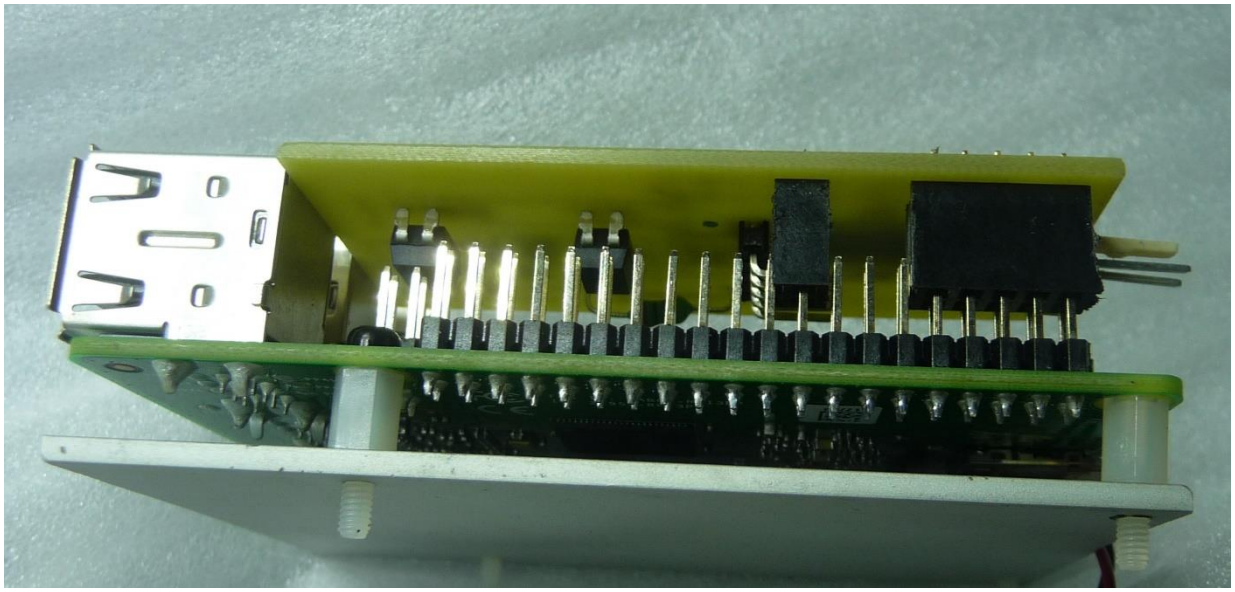
Prima di collegarlo al bus controllate con un voltmetro. Il collegamento serve per i segnali di ingresso e uscita.

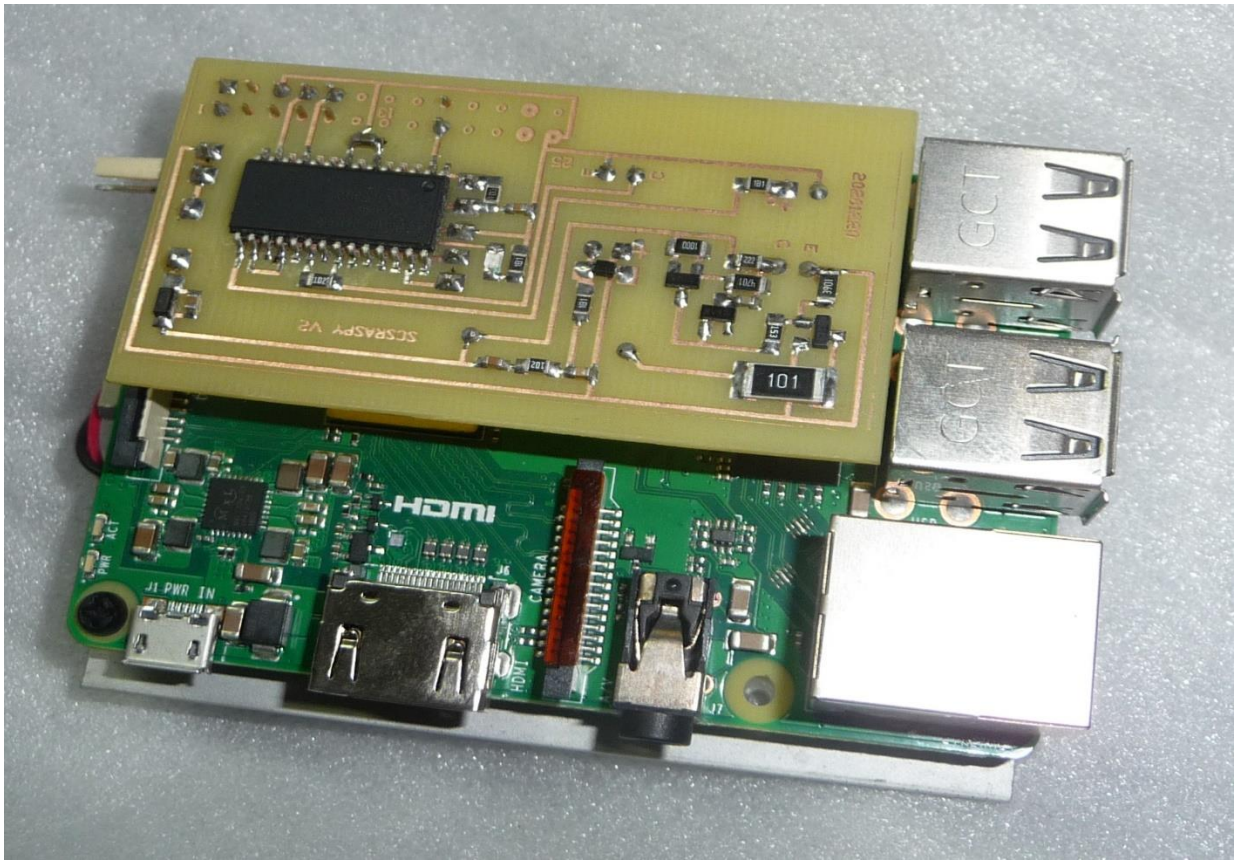
- Il connettore in basso a doppia fila è la connessione al bus gpio del raspberry da cui viene prelevata sia l'alimentazione che i segnali di comunicazione.
- Il connettore centrale a 5 pin (ICSP) serve a riprogrammare il PIC tramite un programmatore Microchip (es. pickit3): 1=/reset 2=positivo 3.3V 3=negativo 4=PGD 5=PGC
- Il connettore in alto è destinato (ad uso futuro) al collegamento con una o più schede relè e una o più schede pulsanti tramite apposite interfacce I2C.

Per connettere la scheda ad ARDUINO riferirsi al seguente schema:

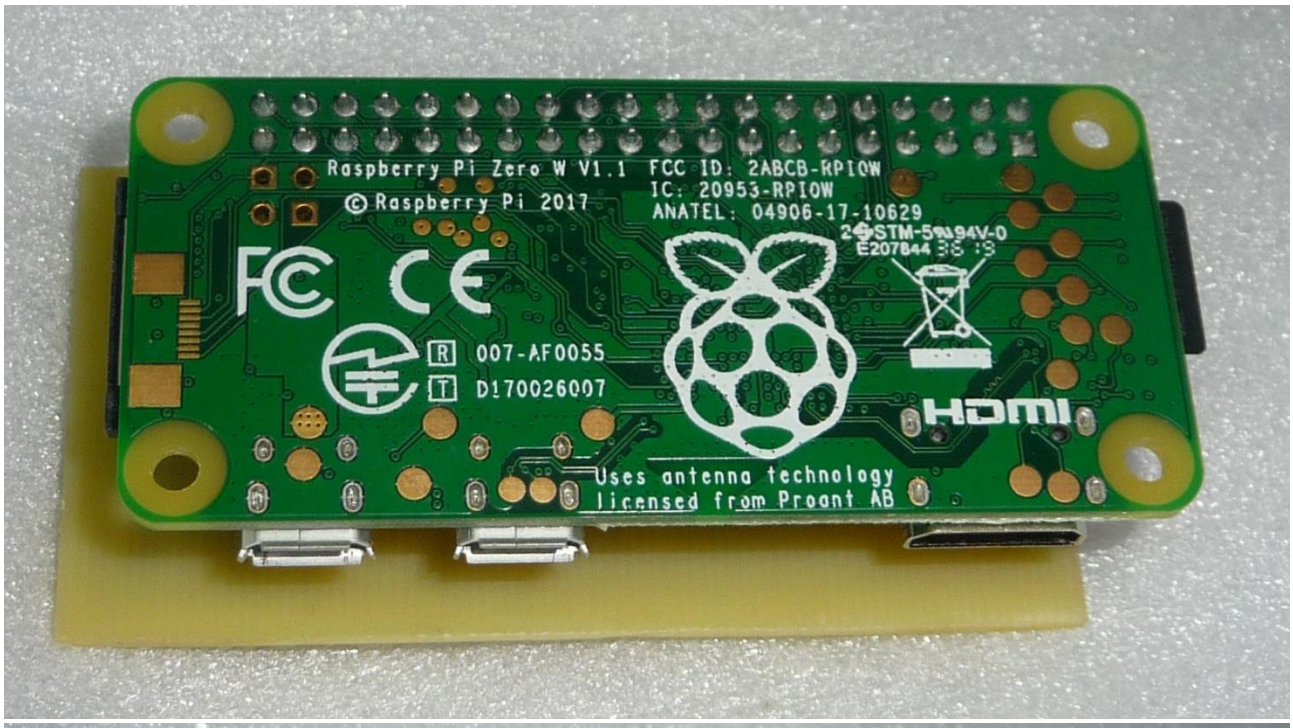


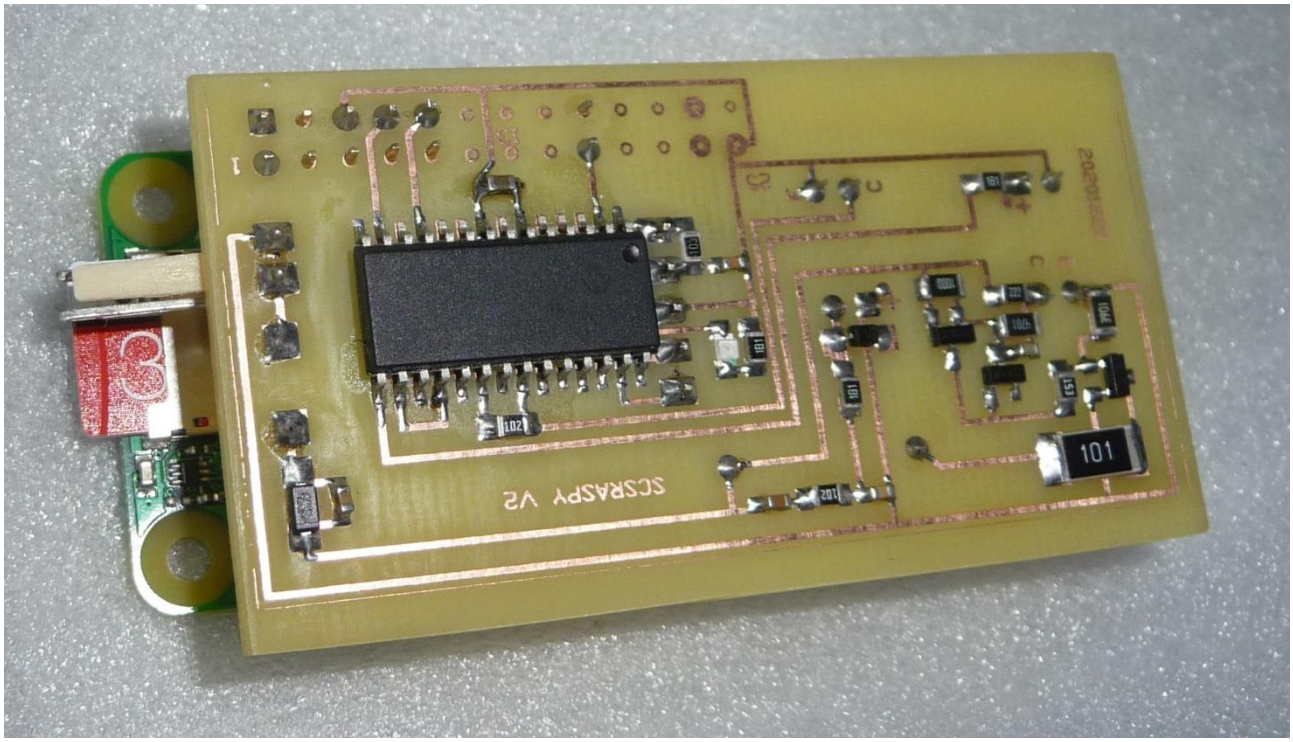
La scheda va innestata sul connettore gpio di raspberry:





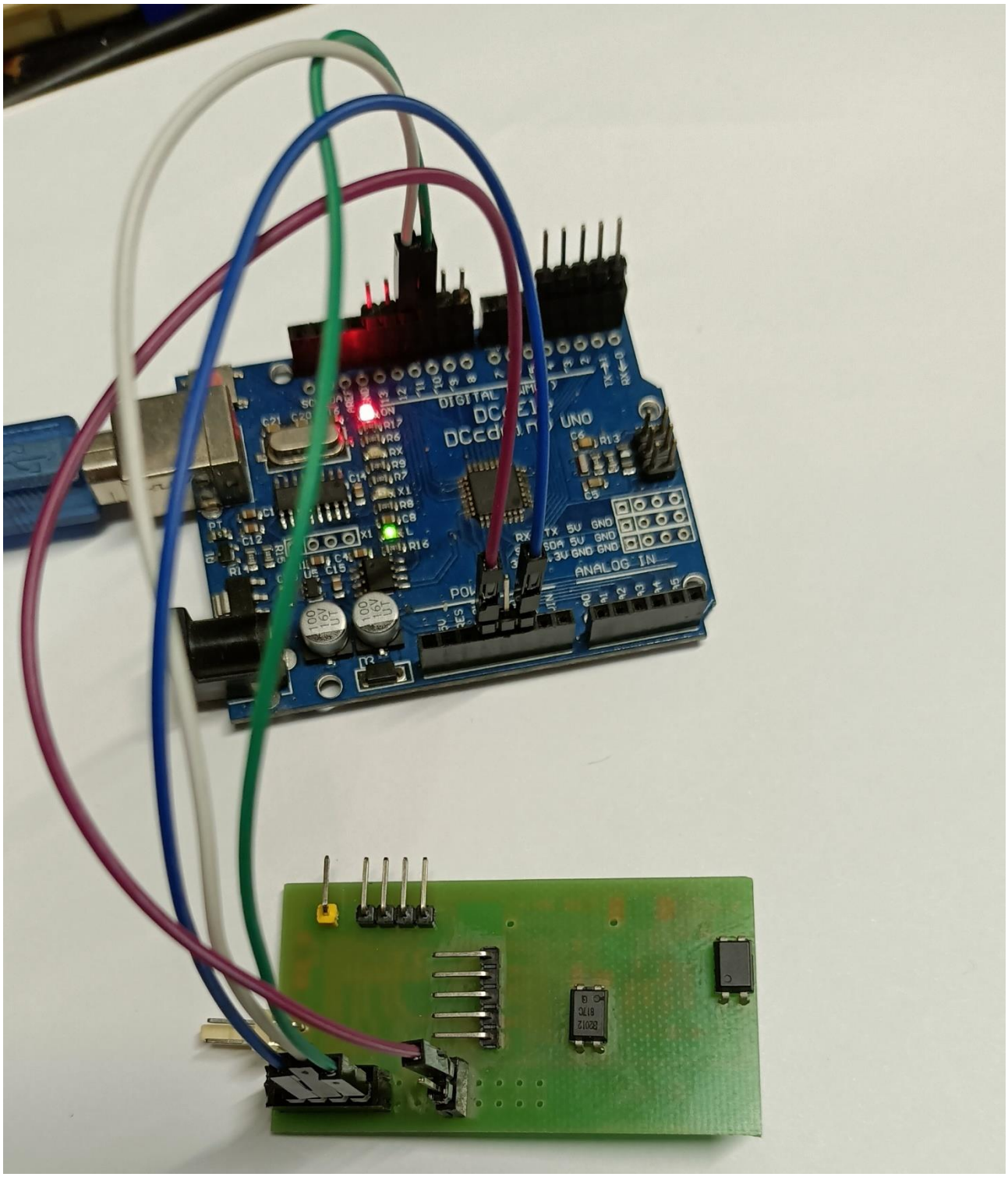
Su raspberry zero W:





Potete anche lasciare la scheda staccata dal raspberry e collegarla con cavetti maschio-femmina. I pin da collegare obbligatoriamente sono: 6-8-10-17 (+3,3 gnd uart rx uart tx).

Con Arduino UNO (sulla seriale software):



ACCENSIONE

Accendendo raspberry o Arduino il led lampeggerà con una frequenza molto bassa (1 lampeggio ogni 10 secondi o più); con il software raspberry attivato il lampeggio si stabilizzerà a circa 1 lampeggio al secondo.

Non è indispensabile collegare il bus scs, lo potrete fare anche dopo, al momento di testare la scheda.

SOFTWARE RASPBERRY

E' RICHIESTA un minimo di competenza sulla connessione SSH con raspberry, sui comandi raspberry linux (raspbian) e (per l'aggiornamento manuale) su un software di trasferimento files ftp (es. filezilla).

Il software rilasciato è compilato per un s.o. a 32 bit – se intendete usarlo su di un s.o. a 64 bit leggete in internet le configurazioni che dovete fare.

Si sconsiglia l'uso con il sistema operativo HA OS perché in tal caso home assistant pretende il pieno controllo dell'hardware comprese le periferiche seriali, che quindi non possono essere utilizzate da dispositivi personalizzati come questo.

Caricamento manuale:

Entrate in SSH e fate il login.

Create una directory, per esempio "raspy_scsgate_exe" e poi entrateci:

```
mkir raspy_scsgate_exe
```

```
cd raspy_scsgate_exe
```

Usando il software ftp trasferite in questa directory i files qui riportati, scaricandoli dapprima dal repository https://github.com/papergion/raspy_scsgate_exe :

- scsfirmware
- scsgate_x
- scsgate_y
- scslog
- scsmonitor
- scsdiscover
- scstcp
- libeasysocket.so
- libpaho-mqtt3c.so.1

tutti questi files dovranno essere dichiarati eseguibili, quindi date il comando

```
chmod +x *
```

Caricamento automatico:

Entrate in SSH e fate il login.

Verificare di avere già installato su raspberry il software GIT:

```
git --version
```

Se il software GIT non è installato:

```
sudo apt update
sudo apt install git
git --version
```

Clonate su raspberry il repository:

```
git clone https://github.com/papergion/raspy_scsgate_exe.git
```

Entrate nel repository:

```
cd raspy_scsgate_exe
```

Rendete eseguibili i files con:

```
chmod +x *
```

Per aggiornare (in futuro) il repository, entrateci e date il comando:

```
git pull
```

ATTENZIONE:
Se state usando un sistema operativo a 64bit gli eseguibili stanno nella sotto-cartella bin64.
Quindi

```
cd bin64
chmod +x *
```

Dopo il caricamento:

Se doveste avere l'esigenza di cancellare il repository locale, uscite dalla directory del repository e poi date il comando:

```
rm -rf raspy_scsgate_exe
```

Ora bisogna attivare l'interfaccia seriale GPIO, usando il comando RASPI-CONFIG (le opzioni riportate possono essere differenti, in base alla versione di raspi-config che avete sul vostro raspberry).

```
sudo raspi-config
```

Interfacing options

Serial port

Would you like a login shell to be accessible over serial? **<NO>**

Would you like the serial port hardware to be enabled? **<YES>**

Uscendo da raspi-config vi verrà chiesto di fare il reboot di Raspberry – eseguite.

Ora potete rientrare con SSH, rifare il login e ritornare alla directory scsgate.

```
cd raspy_scsgate_exe
```

Questo l'elenco delle applicazioni ad ora disponibili ed una breve descrizione, il dettaglio a seguire.

- **scslog** sniffer dispositivi sul bus scs
- **scsmonitor** monitor per visualizzare e inviare manualmente messaggi
- **scsdiscover** pubblica in mqtt i messaggi di discover dei dispositivi
- **scsfirmware** consente di aggiornare il software PIC
- **scsgate_x** gateway tra SCS e MQTT e/o ALEXA
- **scsgate_y** gateway tra SCS e MQTT e/o ALEXA e/o schede rele locali
- **scstcp** gateway tra SCS e client TCP
- **gatexbatch** esempio di startup asincrono di scsgate_x
- **gatexbatchend** esempio di stop asincrono di scsgate_x

Ogni applicazione può essere lanciata singolarmente – l'esecuzione termina con <ctrl> C o con la caduta della sessione.

NON lanciate mai contemporaneamente due applicazioni da sessioni differenti – andrebbero in collisione tra loro.

Per rendere una applicazione permanente (per alcune ha senso) potete, per esempio, usare il comando linux "*nohup*" di cui potete trovare logica e sintassi su qualche manuale linux. L'applicazione partirà in maniera asincrona, lasciandovi libero il terminale SSH, e continuerà anche dopo il logout.

Un esempio lo trovate nei files "gatexbatch" (fa partire l'app scsgate_x) e "gatexbatchend" (ferma l'app scsgate_x).

Per rendere una applicazione permanente ed autostartante potete, per esempio, usare la tecnica "crontab" di cui potete prendere visione su internet.

ATTENZIONE: se lanciate una applicazione in modalità permanente RICORDATEVI di non lanciare mai contemporaneamente un'altra applicazione scs… perché avreste dei risultati fasulli.

Le applicazioni sono scritte in C++ - delle applicazioni "scslog" e "scsmonitor" sono disponibili anche i sorgenti, come esempio di collegamento seriale con scsgate per lo sviluppo eventuale di nuove applicazioni.

La sequenza corretta di messa in funzione di raspy_scsgate è la seguente (i dettagli nei capitoli seguenti) :

- loggare i messaggi e riconoscere i vari dispositivi (scslog / scsmonitor)
- in seguito al log generare il file "scsconfig"
- correggere manualmente il file scsconfig sistemando le descrizioni e, se necessario, aggiungendo o togliendo dispositivi
- avviare il server mqtt e l'applicativo domotico (home assistant, domoticz, …) ed eseguire "scsdiscover"
- avviare il programma/servizio scsgate_x – usate l'opzione -u
- se usate alexa fare (da alexa) il discover dei dispositivi
- se usate anche una scheda locale con rele e/o interruttori non usate scsgate_x ma scsgate_y

SCSLOG

E' la più semplice delle applicazioni: si connette a scsgate a 112500 baud – se specificato il parametro -v (verbose) scrive a video **tutti** i messaggi che passano sul bus.

Si lancia con il comando:

./scslog

Oppure

./scslog -v

Ogni volta che intercetta i messaggi da un nuovo dispositivo scrive una riga colorata con i dati del dispositivo riconosciuto:

verdi se non ci sono incoerenze

giallo nel caso in cui il dispositivo era precedentemente stato riconosciuto come switch (luce) ma ora in base ad un nuovo comando intercettato appare essere un dimmer.

Rosso nel caso in cui venga intercettato un dispositivo di tipo incoerente con precedenti intercettazioni.

Quindi l'uso più naturale di questo software è quello di fare un censimento dei dispositivi.

Alla chiusura (ctrl-C) l'applicazione scriverà nella directory corrente un file di nome "**discovered**" – tale file può essere rinominato in "**scsconfig**", eventualmente modificato ed utilizzato dalle altre applicazioni.

ATTENZIONE: se eseguite una seconda volta **scslog** perderete le eventuali modifiche fatte a mano sul file "discovered".

SCSMONITOR

L' applicazione si connette a scsgate a 112500 baud – e consente di interagire usando i comandi di base di lettura e scrittura elencati nel capitolo "il protocollo seriale di SCSGATE".

Si lancia con il comando:

./scsmonitor

L'applicazione mette la scheda scsgate in modalità ascii, attiva il log e richiama il messaggio scsgate di help @h.

Rimane poi in attesa di comandi da tastiera fino a quando riceve il comando <ctrl C>.

I comandi validi sono quelli riportati nel paragrafo "il protocollo seriale di scsgate" e consentono lettura e scrittura sul bus scs.

Attenzione, i comandi vengono letti da tastiera byte per byte e talquali rigirati verso scsgate, non sono quindi leciti comandi di movimento cursore, tab, cancellazione, ecc…

SCSFIRMWARE

L' applicazione si connette a scsgate a 112500 baud – e consente di aggiornare il firmware caricato nel PIC da un file locale (.bin) contenente la nuova versione.

Si lancia con il comando:

./scsfirmware -fnome del file bin test di aggiornamento (aggiornamento fittizio)

./scsfirmware -u -fnome del file bin aggiornamento reale firmware

L'applicazione visualizzerà la versione corrente del firmware caricato, poi chiederà conferma dell'aggiornamento (reale o fittizio che sia) e quindi procederà.

Per ogni blocco inviato/aggiornato visualizzerà i caratteri "." (blocco inviato) e poi "k" (blocco accettato/aggiornato).

Al termine attende la ripartenza del PIC e visualizza la versione corrente del firmware sul PIC.

SCSDISCOVER

Serve a pubblicare immediatamente in MQTT le informazioni sui dispositivi trovati e memorizzati nel file scsconfig.

E' necessario che il broker mqtt sia attivo e che sia attiva una applicazione in grado di recepire i messaggi di scoperta dei dispositivi (homeassistant, domoticz,…)

L'applicazione necessita di avere (nella directory stessa di lancio) un file denominato "**scsconfig**" contenente le definizioni di tutti i dispositivi scs riconoscibili e trattabili. Il formato di questo file è riportato nell'apposito capitolo.

Non interagisce con il bus scs.

Si lancia con il comando:

./scsdiscover ***opzioni***

Le opzioni valide sono le seguenti:

-v1/2/3 verbose a livello 1 oppure 2 oppure 3

Ogni livello rappresenta una modalità crescente di display a video, utili sostanzialmente a verificare e debuggare eventuali problemi

-Bbrokername:port

Attiva la connessione ad un broker MQTT – può essere digitato il nome o l'indirizzo ip del broker e la porta (ad esempio **-B192.168.1.150:1883**) – se il nome/indirizzo:porta non è digitato assume che sia **localhost:1883** (cioè un broker installato sullo stesso raspberry).

-Uusername

-Ppassword

Eventuali user e password per la connessione al broker MQTT

SCSGATE_X

E' una applicazione "finale" - si connette a scsgate a 112500 baud – ed ha la possibilità di connettersi ad un broker MQTT e ad un dispositivo ECHO-DOT (alexa).

L'applicazione necessita di avere (nella directory stessa di lancio) un file denominato "**scsconfig**" contenente le definizioni di tutti i dispositivi scs riconoscibili e trattabili. Il formato di questo file è riportato nell'apposito capitolo.

Si lancia con il comando:

./scsgate_x ***opzioni***

Le opzioni valide sono le seguenti:

-v1/2/3 verbose a livello 1 oppure 2 oppure 3

Ogni livello rappresenta una modalità crescente di display a video, utili sostanzialmente a verificare e debuggare eventuali problemi

-u

Aggiorna immediatamente nel PIC l'elenco dei dispositivi scs ed il relativo tipo. L'opzione va usata ogni volta che nel file scsconfig si cambia l'indirizzo o il tipo o i dati di un dispositivo, oppure se ne aggiunge uno nuovo o se ne elimina uno, o se si cambia l'ordine dei dispositivi. Non è necessario se si cambiano solo le descrizioni. Non è opportuno usarlo se il file scsconfig non è cambiato.

-H

Attiva l'emulazione di un dispositivo philips HUE, indispensabile alla connessione con echo-dot ALEXA.

-Bbrokername:port

Attiva la connessione ad un broker MQTT – può essere digitato il nome o l'indirizzo ip del broker e la porta (ad esempio **-B192.168.1.150:1883**) – se il nome/indirizzo:porta non è digitato assume che sia **localhost:1883** (cioè un broker installato sullo stesso raspberry).

-Uusername

-Ppassword

Eventuali user e password per la connessione al broker MQTT

-D1/2

Connessione diretta HUE-MQTT – se specificata questa opzione i comandi hue (alexa) transitano direttamente nel server MQTT come stati (opzione -D1) o come comandi (opzione -D2) – anche in assenza della scheda SCSGATE.

SCSGATE_Y

E' una applicazione "finale" - si connette a scsgate a 112500 baud – ed ha la possibilità di connettersi ad un broker MQTT e ad un dispositivo ECHO-DOT (alexa) e a schede locali rele / interruttori.

L'applicazione necessita di avere (nella directory stessa di lancio) un file denominato "**scsconfig**" contenente le definizioni di tutti i dispositivi scs riconoscibili e trattabili. Il formato di questo file è riportato nell'apposito capitolo.

Si lancia con il comando:

./scsgate_y ***opzioni***

Le opzioni valide sono le seguenti:

-v1/2/3 verbose a livello 1 oppure 2 oppure 3

Ogni livello rappresenta una modalità crescente di display a video, utili sostanzialmente a verificare e debuggare eventuali problemi

-u

Aggiorna immediatamente nel PIC l'elenco dei dispositivi scs ed il relativo tipo. L'opzione va usata ogni volta che nel file scsconfig si cambia l'indirizzo o il tipo o i dati di un dispositivo, oppure se ne aggiunge uno nuovo o se ne elimina uno, o se si cambia l'ordine dei dispositivi. Non è necessario se si cambiano solo le descrizioni. Non è opportuno usarlo se il file scsconfig non è cambiato.

-H

Attiva l'emulazione di un dispositivo philips HUE, indispensabile alla connessione con echo-dot ALEXA.

-Bbrokername:port

Attiva la connessione ad un broker MQTT – può essere digitato il nome o l'indirizzo ip del broker e la porta (ad esempio **-B192.168.1.150:1883**) – se il nome/indirizzo:porta non è digitato assume che sia **localhost:1883** (cioè un broker installato sullo stesso raspberry).

-Uusername

-Ppassword

Eventuali user e password per la connessione al broker MQTT

-D1/2

Connessione diretta HUE-MQTT – se specificata questa opzione i comandi hue (alexa) transitano direttamente nel server MQTT come stati (opzione -D1) o come comandi (opzione -D2) – anche in assenza della scheda SCSGATE.

-Ii2c address (high byte) (i maiuscolo)

Attiva la connessione alle schede i2c locali (massimo 8) – può essere digitato il primo carattere dell'indirizzo i2c della scheda (o delle schede) collegata – se non digitato viene assunto l'indirizzo 2 (tipico delle schede basate su PCF8574). Il secondo carattere dell'indirizzo (quello che dipende dai jumper della schedina) viene invece specificato per ogni device (relè o interruttore) insieme al numero di porta. Vedi paragrafo "dispositivi locali – relè e interruttori.

-s0/1/2/3/4

Indica la modalità di funzionamento degli switches locali nel caso si usi l'apposita espansione (paragrafo "dispositivi locali), distinguendo gli switches usati per comandare dispositivi scs da quelli usati per comandare relè locali.

Valore di 's'	**Comando per SCS**	**Comando per relè locale**
0 (default)	Pulsante on/off	Fisso 0=accende 1=spegne
1	Pulsante on/off	Deviatore accende/spegne
2	Pulsante on/off	Pulsante on/off
3	Deviatore accende/spegne	Pulsante on/off
4	Deviatore accende/spegne	Deviatore accende/spegne
5	Deviatore accende/spegne	Fisso 0=accende 1=spegne

SCSTCP

E' un ponte TCP che si connette a scsgate a 112500 baud – ed ha la possibilità sia di ricevere chiamate TCP dirottandone i dati verso scsgate che viceversa. Risponde anche a comandi http verso scsgate e viceversa.

L'applicazione necessita di avere (nella directory stessa di lancio) un file denominato "**scsconfig**" contenente le definizioni di tutti i dispositivi scs riconoscibili e trattabili. Il formato di questo file è riportato nell'apposito capitolo.

Si lancia con il comando:

./scstcp ***opzioni***

Le opzioni valide sono le seguenti:

-v (verbose)

Emette una mole di display a video, utili sostanzialmente a verificare e debuggare eventuali problemi

-u

Aggiorna immediatamente nel PIC l'elenco dei dispositivi scs ed il relativo tipo. L'opzione va usata ogni volta che nel file scsconfig si cambia l'indirizzo o il tipo o i dati di un dispositivo, oppure se ne aggiunge uno nuovo o se ne elimina uno, o se si cambia l'ordine dei dispositivi. Non è necessario se si cambiano solo le descrizioni. Non è opportuno usarlo se il file scsconfig non è cambiato.

-pnnnnn

Indica di utilizzare per la comunicazione tcp e http la porta "nnnnn", se non indicata assume per default la porta 5045.

Il ponte TCP può essere usato per diversi scopi, i principali sono i seguenti:

- consentire una connessione tramite il (precedente) modulo vb6 scsknxgate v5.5
- consentire una connessione non-mqtt con domoticz o simili
- consentire una connessione con propri applicativi tramite chiamate tcp e/o http

i comandi accettati tramite TCP sono al paragrafo "comunicazione tcp", informazioni ed esempi di chiamate e risposte http nel paragrafo "domoticz - http"

SCSCONFIG – IL FILE DI CONFIGURAZIONE

Come già detto il file può essere generato automaticamente con “scslog” e poi sistemato manualmente. Esempio di file:

{"coverpct":"false","devclear":"true"}
{"device":"21","type":"8","maxp":"","descr":"tapparella sala"}
{"device":"22","type":"9","maxp":"200","descr":" tapparella cucina"}
{"device":"26","type":"1","descr":"salotto"}
{"device":"27","type":"3","descr":"scala"}
{"device":"29","type":"1","descr":"bagnetto"}
{"device":"01","type":"15","descr":"temp. zona notte”}
{"device":"85","type":"x30","descr":"uscita 5 interfaccia i2c: 20 ”}
{"device":"86","type":"x41","cmd":"21","descr":"ingresso 6 interfaccia i2c: 21”}

Non modificate la prima riga, è necessaria quando il file viene caricato nel PIC per cancellare la situazione precedente.

Il parametro “device” deve indicare esattamente l’indirizzo SCS. I dispositivi locali (relè ed interruttori) devono avere un indirizzo univoco, non già esistente nell’impianto scs, dove il secondo carattere indica il numero di ingresso o uscita della scheda di espansione i2c. Ad esempio il dispositivo “85” si riferisce ad una espansione che fa riferimento all’ingresso 5.

Ogni riga successiva rappresenta un dispositivo scs presente sul bus: il valore successivo a **“device”** indica l’indirizzo scs assegnato a quel dispositivo da chi ha configurato l’impianto. Il valore indicato da **“type”** invece indica la tipologia a cui appartiene – i tipi trattati da scsgate sono i seguenti:

“1” – switch o luce (attuatore on/off)
“3” – dimmer (luce regolabile) che può essere spenta/accesa/alzata/abbassata
“8” – tapparella, può essere alzata/abbassata/fermata
“9” – tapparella a percentuale, può essere alzata/abbassata/fermata/impostata ad un certo valore
“11” – dispositivo generico da cui possono arrivare telegrammi
“12” – dispositivo generico a cui possono essere inviati telegrammi
“14” – centralina di allarme
“15” – termostato
“x3N” – (da x30 a x37) - relè su scheda espansione i2c con indirizzo “iN” (“i” è specificato dal parametro di lancio I) – per esempio se lancio scsgate_y con parametro -I2 tutti i dispositivi con tipo “x37” useranno l’indirizzo i2c “27”, tutti i dispositivi con tipo “x34” useranno l’indirizzo i2c “24” (Vedi paragrafo “dispositivi locali – relè e interruttori.)
“x4N” – (da x40 a x47) - input da scheda espansione i2c con indirizzo “iN” (“i” è specificato dal parametro di lancio I) – per esempio se lancio scsgate_y con parametro -I2 tutti i dispositivi con tipo “x47” useranno l’indirizzo i2c “27”, tutti i dispositivi con tipo “x45” useranno l’indirizzo i2c “25” (Vedi paragrafo “dispositivi locali – relè e interruttori.)

I dispositivi di tipo 1 / 3 / 8 / 9 possono interagire tramite mqtt e tramite alexa. I dispositivi di tipo 11 / 12 / 14 / 15 / x3N / x4N solo con mqtt e solo in lettura.

Il parametro indicato da **“cmd”** è applicabile solo ai dispositivi di tipo x4N (interruttori locali) ed indica il device che deve essere comandato dall’interruttore locale. Il dispositivo comandato deve essere presente anch’esso nel file scsconfig, può essere un dispositivo scs od un relè locale.

Il parametro indicato da **“maxp”** è applicabile solo ai dispositivi di tipo 9 (tapparelle a percentuale) ed indica il tempo di salita o discesa in decimi di secondo.

Il parametro indicato da **"descr"** indica la descrizione mnemonica del dispositivo. Per i tipi 1-3-8-9 è il nome che viene poi censito da Alexa nella fase di "scoperta" e che verrà poi riconosciuto nei comandi vocali.

CONNESSIONE MQTT

NON inserite l'IP di un broker se non lo userete, appesantireste inutilmente il lavoro dell'applicazione in operazioni inutili.

La connessione con il broker avviene all'inizio e se andata a buon fine viene mantenuta – se poi la connessione cade l'applicazione ritenta la connessione ogni minuto.

La connessione MQTT è relativa a luci, dimmer, tapparelle, termostati. e sottoscrive i seguenti topics:

scs/+/set/+ per i comandi on/off/stop dei dispositivi

luci o dimmer scs/switch/set/<*scs address*> payload ON o OFF

cover (tapparelle) scs/cover/set<*scs address*> payload ON o OFF o STOP

scs/+/setlevel/+

luci o dimmer scs/switch/setlevel/<scs address> payload da 10 a 90

scs/+/setposition/+

cover (tapparelle a %) scs/cover/setposition<*scs address*> payload da 1 a 100 (%)

PUBBLICAZIONE STATO DISPOSITIVI:

La connessione MQTT pubblica, (ad ogni notifica di stato che transita sul bus scs), i seguenti topics di aggiornamento stato dispositivi:

scs/switch/state/<*scs address*> payload "ON" o "OFF" (per luci/attuatori/dimmer)

scs/switch/value/<*scs address*> payload 00-01 oppure 10-255 (per dimmer)

scs/cover/state/<*scs address*> payload "open" o "closed" o "STOP" (per tapparelle)

scs/cover/value/<*scs address*> payload 0-100 (per tapparelle gestite a percentuale)

scs/sensor/temp/state/<*scs address*> payload temperature con 1 decimale (per termostati)

Se è stato settato l'uso per HOME-ASSISTANT gli stati pubblicati per le tapparelle invece saranno "open" e "closed".

CAMBIO DI STATO DISPOSITIVI:

La connessione MQTT accetta i seguenti topics di comando sui dispositivi, li trasforma in telegrammi SCS e li invia sul bus tramite scsgate:

scs/switch/set/<*scs address*> payload “ON” o “OFF” (per luci/attuatori/dimmer)

scs/switch/setlevel/<*scs address*> payload 10-90 oppure 10-255 (per dimmer)

scs/cover/set/<*scs address*> payload “OPEN” o “CLOSE” o “STOP” (per tapparelle)

scs/cover/setposition/<*scs address*> payload 0-100 (per tapparelle gestite a percentuale)

DISPOSITIVI "GENERICI":

Alcuni telegrammi che transitano sul bus SCS possono non riguardare i classici dispositivi elencati (switch, dimmer, tapparelle) ma dispositivi differenti di cui non mi è noto il significato del messaggio; per esempio videocitofoni, impianti di allarme, ecc… Questi dispositivi possono essere censiti come "GENERICI" (tipo 11 e 12).

Ricordo che la struttura dei messaggi SCS è:

A8 <destinazione> <provenienza> <tipo comando> <valore> <checkbyte> A3

Alla ricezione dal bus di un messaggio viene analizzata la <destinazione> e se l'indirizzo risulta censito come dispositivo ti tipo 11 viene pubblicato un topic:

scs/generic/to/<*destinazione*> payload: <provenienza> <tipo comando> <valore>

Viene analizzata la <provenienza> e se l'indirizzo risulta censito come dispositivo di tipo 12 viene pubblicato un topic:

scs/generic/from/<provenienza> payload: <destinazione> <tipo comando> <valore>

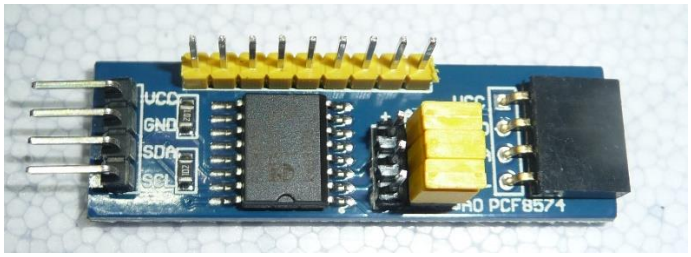
Viene invece generato un messaggio sul bus scs se in MQTT viene ricevuto un topic:

scs/generic/set/<*destinazione*> payload: <provenienza> <tipo comando> <valore>

INTERESSANTE:
La tecnica dei dispositivi "generici" è utile per generare qualunque comando di lunghezza standard. Per esempio, anche definendo un interruttore "luce" come normale switch, se scrivi in mqtt il comando con topic **scs/generic/set/62** con payload **601266** ed otterrai in scs il comando A8 62 60 12 66 76 A3 (che genera una accensione temporizzata della luce 62 - vedi appunti.doc). Se scrivi in mqtt il comando con topic **scs/generic/set/B3** con payload **021201** ed otterrai in scs il comando A8 B3 02 12 01 A2 A3 (comando di ambiente).

DISPOSITIVI LOCALI (RELÈ E SWITCHES):

Possono essere installate localmente schede pilotate da interruttori o pulsanti e schede che pilotano dei relè. Sia gli ingressi (interruttori) che le uscite vanno pilotati con schede di espansione i2c tipo questa:



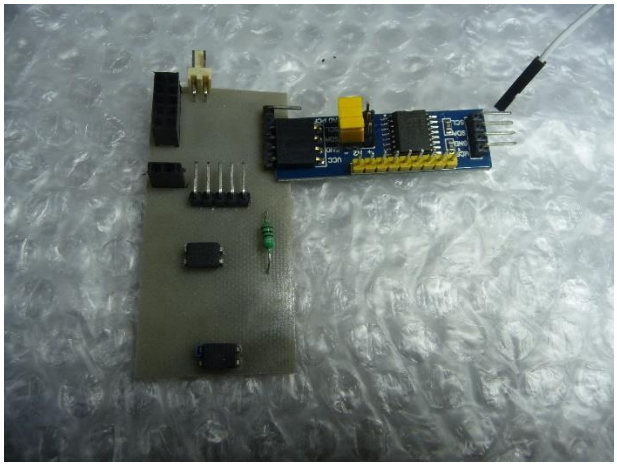
Potete acquistarle su ebay o su Aliexpress – ogni scheda pilota in totale 8 tra ingressi e uscite, costa circa 50 centesimi più spedizione. Es: Modulo PCF8574T modulo PCF8574 modulo di espansione I/O modulo di espansione I/O I2C|Circuiti integrati| - AliExpress

Le schede basate su PCF8574 hanno un indirizzo che va da “20” a “27” – il “2” iniziale è fisso, tipico di pcf8574, il secondo carattere va settato con i jumper (o i microswitch) presenti sulla scheda:

J3	J2	J1	Indirizzo
Off	Off	Off	20
Off	Off	On	21
Off	On	Off	22
0ff	On	On	23
On	Off	Off	24
On	Off	On	25
On	On	Off	26
On	On	On	27

Ogni scheda ha due connettori (equivalenti, uno maschio e l’altro femmina) da collegare a raspy_scsgate con i segnali gnd, +3,3, sda, scl – tale per cui più schede possono essere connesse sequenzialmente – ogni scheda deve avere un indirizzo differente. La prima è connessa a raspy_scsgate, la seconda alla prima, eccetera.

La scheda può essere innestata direttamente a raspy_scsgate solo nel modello W0. Su raspberry 3 e 4 vanno usati dei cavetti di prolunga, perché non c’è sufficiente spazio verso il raspberry.



Ogni scheda ha 8 pin di ingresso/uscita. Ogni pin può essere connesso ad un ingresso di scheda relè, oppure ad un interruttore o pulsante che chiude verso gnd. I pin sono numerati **da 1 a 8** e corrispondono in configurazione (scsconfig) agli indirizzi che terminano con numeri da **0 a 7**.

Questo un esempio di scheda a 8 relè acquistabile su ebay o Aliexpress. Costo indicativo 2 euro più spedizione:

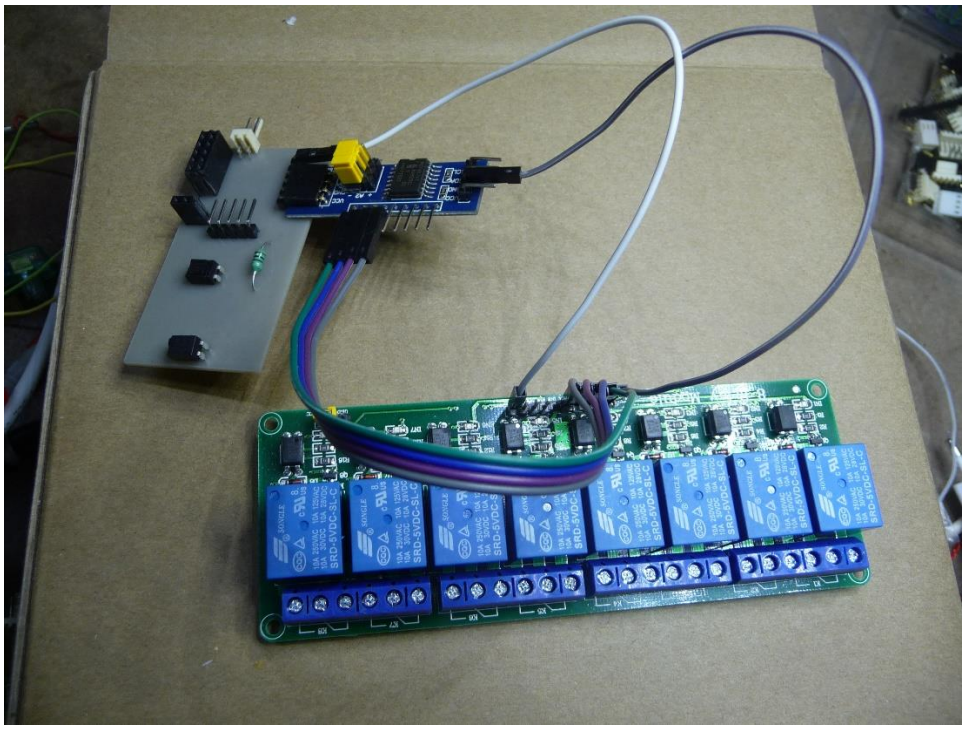
[Canale DC 5V Modulo di Relè della manica con Accoppiatore Ottico A Basso Livello di Trigger Scheda di Espansione per arduino Raspberry Pi|Relè| - AliExpress]

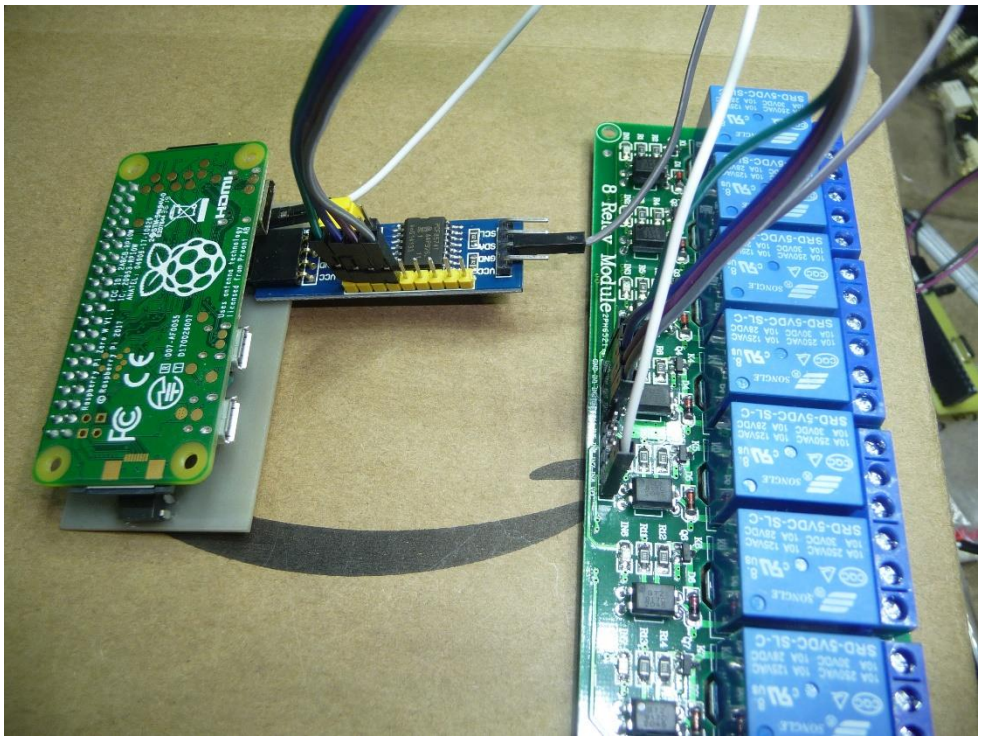
Sempre su ebay o Aliexpress trovate i cavetti femmina-femmina di varie lunghezze necessari a connettere il modulo relè – le connessioni necessarie sono:

gnd (-) va connesso a gnd di raspberry o dell'ultima scheda di espansione i2c.

Vcc (+5V) va connesso con il pin +5V sulla scheda raspy_scsgate (o su raspberry o su qualunque altro alimentatore a 5V).

Gli ingressi 1-8 vanno connessi (per i relè che si intendono usare) alle uscite 1-8 del modulo di espansione.





Le connessioni di input (switches) si fanno semplicemente connettendo ad un interruttore o ad un pulsante un ingresso del modulo di espansione i2c ed una connessione gnd, in modo tale che azionando l'interruttore od il pulsante l'ingresso venga portato a gnd. Se si usano più ingressi è opportuno usare o tutti interruttori o tutti pulsanti.

SOFTWARE ARDUINO

I file dimostrativi vanno caricati in Arduino nella maniera standard solita.

Questo l'elenco delle applicazioni demo ad ora disponibili ed una breve descrizione, il dettaglio a seguire.

- **scsmonitor** monitor per visualizzare e inviare manualmente messaggi
- **scsmulticommand1**
- **scsmulticommand2**
- **scsmulticommand3**
- **scsmulticommand4**
  consentono con diverse modalità di usare arduino come pulsante di comando o attuatore SCS

SCSMONITOR

L' applicazione si connette a scsgate a 19200 baud usando "SoftwareSerial" sui pin 10 e 11. Quindi permette di usare il "serial monitor" (USB) a 115200 baud per visualizzare il traffico sul bus SCS permettendo anche di scrivere messaggi sul bus – consente di interagire usando i comandi di base di lettura e scrittura elencati nel capitolo "il protocollo seriale di SCSGATE".

Con questo software potete scoprire gli indirizzi dei vostri dispositivi.

```
/*
  SCS monitor

 Use pin 10 and 11 for software UART
       connect pin 10 (rx) with scsgate uart tx
       connect pin 11 (tx) with scsgate uart rx
       connect pin GND     with scsgate GND
       connect pin +3.3V   with scsgate +3.3V

 Use USB as serial monitor

 This demo initialize scsgate , then wait for SCS messages and display it over USB
 created 1 Sep 2023 by Guido Pagani as example of SCSgate utilization
 */
#define VERSION "SCSmonitor V1.3"

#include <SoftwareSerial.h>

byte s, m;
byte strMySerial[64];

SoftwareSerial mySerial(10, 11); // RX, TX

void setup()
{

  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while (!Serial) { ; }
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on

  Serial.println(VERSION " started!");

  // set the data rate for the SoftwareSerial port
  mySerial.begin(115200);
  mySerial.write("@");  // set SCSgate/KNXgate slow speed
  mySerial.write(0xEE);  // set SCSgate/KNXgate very slow speed (19200 baud)
  delay(100);            // wait 100ms
  mySerial.flush();
```

```
  mySerial.end();
  Serial.flush();

  // reopen serial port at slow speed
  mySerial.begin(19200);

  mySerial.write("@MA");  // set SCSgate/KNXgate ascii mode
  delay(50);             // wait 50ms
  mySerial.write("@l");  // serial log ON
  delay(50);             // wait 50ms
  mySerial.write("@q");  // query firmware version
  delay(50);             // wait 50ms
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off

  Serial.println("Initialized!");
}

void loop() // run over and over
{
// ================================================================================================
  m = 0;
  while (mySerial.available())
// read from scsgate output
  {
    strMySerial[m++] = mySerial.read();       // receive from KNXgate/SCSgate
  }
  s = 0;
  while (m)
  // write to USB
  {
    digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on
    Serial.write(strMySerial[s++]);          // write on serial USB
    m--;
  }
// ================================================================================================


// ================================================================================================
  if (Serial.available())
  {
  // read from USB
    s = Serial.read();        // receive from serial USB
    Serial.write(s);          // echo to serial USB
  // write to scsgate
    delayMicroseconds(100);
    mySerial.write(s);        // write to scsgate
  }
// ================================================================================================
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED oFF
}
```

SCSMULTICOMMANDX (1-2-3-4)

Software dimostrativo per interagire con il bus SCS.

Nel programma si dichiarano (massimo 6) i pin di arduino da usare come switch di input, i pin da usare come output (led o relè) e gli indirizzi SCS da simulare o copiare.

SCSMULTICOMMAND2 e 4 utilizzano come rx/tx i pin 10 e 11 e consentono quindi l'utilizzo contemporaneo della USB per verifica o debug. Invece SCSMULTICOMMAND1 e 3 lavorano direttamente con i pin rx/tx nativi (0 e 1) direttamente a 115200 baud.

SCSMULTICOMMAND1 e 2 utilizzano i pin di input come "toggle": ad ogni impulso basso (connessione verso GND) la corrispondente luce SCS viene alternativamente accesa e spenta.

SCSMULTICOMMAND3 e 4 utilizzano i pin di input come "on/off": la corrispondente luce SCS viene accesa collegando il pin di input al +, viene spenta collegandolo a GND, non cambia stato lasciandolo scollegato.

SCSMULTICOMMAND1

```
/*
  SCS multi - command - 6  SWITCHES   +   6 ACTUATORS/LED

  ------------------------------------------------------------------------------------
       connect pin 0 (rx)  with scsgate tx
       connect pin 1 (tx)  with scsgate rx
       connect pin GND     with scsgate GND
       connect pin +3.3V   with scsgate +3.3V
 ------------------------------------------------------------------------------------
  by Guido Pagani as example of SCSgate utilization
 */

#define VERSION "SCScommand V1.3"

//
//=============================================================================================
//                      DEFINIZIONI  DA  PERSONALIZZARE
//
//=============================================================================================

//  SWITCH ARDUINO:   ETICHETTA SCS DISPOSITIVI DA ACCENDERE / SPEGNERE

#define SW_1_DEVICE   0x31
#define SW_2_DEVICE   0x32
#define SW_3_DEVICE   0x33
#define SW_4_DEVICE   0x34
#define SW_5_DEVICE   0x35
#define SW_6_DEVICE   0x36

//  SWITCH ARDUINO:   PIN ARDUINO A CUI SONO COLLEGATI gli SWITCH
//                    ogni SW accende/spegne alterntivamente la luce scs corrispondente

#define SW_1_PIN      8
#define SW_2_PIN      9
#define SW_3_PIN      12
#define SW_4_PIN      13
#define SW_5_PIN      14
#define SW_6_PIN      15
#define SW_A_PIN      16  // e' una sorta di reset interno degli stati degli sw e led


//  LED O ATTUATORI ARDUINO: ETICHETTA SCS A CUI CORRISPONDONO

#define LED_1_DEVICE  0x31
#define LED_2_DEVICE  0x32
#define LED_3_DEVICE  0x33
#define LED_4_DEVICE  0x34
#define LED_5_DEVICE  0x35
#define LED_6_DEVICE  0x36

//  LED O ATTUATORI ARDUINO:   PIN ARDUINO A CUI SONO COLLEGATI

#define LED_1_PIN    2
#define LED_2_PIN    3
#define LED_3_PIN    4
#define LED_4_PIN    5
#define LED_5_PIN    6
#define LED_6_PIN    7

//
//=============================================================================================
//                      FINE  DEFINIZIONI  DA  PERSONALIZZARE
//
//=============================================================================================



// ============================================================================
unsigned char mylen, myindex, s, m, a, b, t;
unsigned char SerialStatus = 0;  // 1=reading  2=readed
int  mytime;
unsigned char dDevice = 0;
unsigned char dStato = 0xFF;
```

```
unsigned char sBufferIn[16];
unsigned char sBufferOu[16];
unsigned char scsStato[32];
unsigned char swStato[32];
int  paused;
int  timeled = 0;
// ================================================================================
void setup()
{
  pinMode(SW_1_PIN, INPUT_PULLUP);
  pinMode(SW_2_PIN, INPUT_PULLUP);
  pinMode(SW_3_PIN, INPUT_PULLUP);
  pinMode(SW_4_PIN, INPUT_PULLUP);
  pinMode(SW_5_PIN, INPUT_PULLUP);
  pinMode(SW_6_PIN, INPUT_PULLUP);
  pinMode(SW_A_PIN, INPUT_PULLUP);

  pinMode(LED_1_PIN, OUTPUT);
  pinMode(LED_2_PIN, OUTPUT);
  pinMode(LED_3_PIN, OUTPUT);
  pinMode(LED_4_PIN, OUTPUT);
  pinMode(LED_5_PIN, OUTPUT);
  pinMode(LED_6_PIN, OUTPUT);

  for (s=0; s<sizeof(scsStato); s++)
  {
    scsStato[s] = 0xFF;
    swStato[s] = 0xFF;
  }

  allLed(0);

  // Open serial communications and wait for port to open:
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on
  // set the data rate for the SoftwareSerial port
  Serial.begin(115200);

//  Serial.write("@");  // set SCSgate/KNXgate slow speed
//  Serial.write(0xEE);  // set SCSgate/KNXgate very slow speed (19200 baud)
//  delay(100);            // wait 100ms
//  Serial.end();
// reopen serial port at slow speed
//  Serial.begin(19200);

  Serial.flush();

  Serial.write("@");
  Serial.write(0x15);    // avoid to memorize next setup in eeprom
  delay(10);            // wait 10ms
  Serial.write("@MX");  // set SCSgate hex mode
  delay(10);            // wait 10ms
  Serial.write("@Y1");  // set read short
  delay(10);            // wait 10ms
  Serial.write("@F3");  // filter set for ignore ack and double msg
  delay(10);            // wait 10ms
  Serial.write("@b");   // clear  buffer
  Serial.setTimeout(2); // timeout is 2mS
  delay(10);            // wait 10ms
  Serial.write("@l");   // log mode
  delay(10);            // wait 10ms
  Serial.flush();
  paused = 0;
  allLed(1);
}

void allLed(unsigned char stato)
{
  digitalWrite(LED_1_PIN, stato);
  digitalWrite(LED_2_PIN, stato);
  digitalWrite(LED_3_PIN, stato);
  digitalWrite(LED_4_PIN, stato);
  digitalWrite(LED_5_PIN, stato);
  digitalWrite(LED_6_PIN, stato);
}
// ================================================================================
void setupStato(void)
```

```
{
  a = dDevice>>3;
  b = dDevice & 0x07;
  dStato &= 0x01;
  bitWrite(scsStato[a], b, dStato);

  switch (dDevice)
  {
#ifdef LED_1_PIN
    case LED_1_DEVICE:
          digitalWrite(LED_1_PIN, dStato);
          break;
#endif

#ifdef LED_2_PIN
    case LED_2_DEVICE:
          digitalWrite(LED_2_PIN, dStato);
          break;
#endif

#ifdef LED_3_PIN
    case LED_3_DEVICE:
          digitalWrite(LED_3_PIN, dStato);
          break;
#endif

#ifdef LED_4_PIN
    case LED_4_DEVICE:
          digitalWrite(LED_4_PIN, dStato);
          break;
#endif

#ifdef LED_5_PIN
    case LED_5_DEVICE:
          digitalWrite(LED_5_PIN, dStato);
          break;
#endif

#ifdef LED_6_PIN
    case LED_6_DEVICE:
          digitalWrite(LED_6_PIN, dStato);
          break;
#endif

  }
}
// ================================================================================
unsigned char readStato(unsigned char address)
{
  a = address>>3;
  b = address & 0x07;
  return bitRead(scsStato[a], b);
}
  // ================================================================================
char receiveCommand(void) // try to receive telegram and setup state
{
// intero:    07 A8 16 00 12 01 05 A3
//            07 A8 B8 16 12 01 BD A3
// compresso: 04 16 00 12 01 (ignoro)
//            04 B8 16 12 01
  if (SerialStatus == 0)
  {
       mylen = Serial.read();
       if ((mylen > 0) && (mylen < 5) && (mylen != 0xFF))
       {
      SerialStatus = 1;
      myindex = 0;
      mytime = 0;
       }
    return 0;
  }
  if (SerialStatus == 1)
  {
    if (mytime++ > 30000)
    {
      SerialStatus = 0;
```

```
    }
    sBufferIn[myindex] = Serial.read();
    if (sBufferIn[myindex] == 0xFF)
      return 0;
    myindex++;
    if (myindex >= mylen)
    {
      SerialStatus = 2;
    }
  }

  if (SerialStatus == 2)
  {
    SerialStatus = 0;
    if (mylen == 4)
    {
      if (sBufferIn[0] != 0xB8)
      {
        return 0;
      }
      else
      {
        if ((dDevice != sBufferIn[1]) || (dStato != sBufferIn[3]))
        {
                        dDevice = sBufferIn[1];
                        dStato = sBufferIn[3];
                        setupStato();
        }
               return 1;
      }
    }
  }
  return 0;
}
// ================================================================================================
void sendCommand(unsigned char index, unsigned char address) // send command if digital input high
{
  unsigned char rds, tasto;
  tasto = digitalRead((int)index);

  if (tasto != swStato[index])
  {
    swStato[index] = tasto;
    if (tasto == 0)
    {
      if (!paused)
      {
        sBufferOu[0] = '@';
        sBufferOu[1] = 'w';

        sBufferOu[2] = (readStato(address)) ^ 0x01;
        sBufferOu[3] = address;

        s = 4;
        m = 0;

        while (s)
        {
          Serial.write(sBufferOu[m++]);   // write on serial SCSgate
          s--;
          delayMicroseconds(50);
        }
        paused = 2000; // x 0,1mS = 200mS
        Serial.flush();
      }
    }
    else
        paused = 2000; // x 0,1mS = 200mS

  }
}
// ================================================================================================


// ================================================================================================
```

```
void loop() // run over and over
{

// ================================================================================================
  receiveCommand();

#ifdef SW_1_PIN
  sendCommand(SW_1_PIN, SW_1_DEVICE);   // ARDUINO PHISICAL PORT ADDRESS,  SWITCH NUMBER (is device
address)
#endif

#ifdef SW_2_PIN
  sendCommand(SW_2_PIN, SW_2_DEVICE);
#endif

#ifdef SW_3_PIN
  sendCommand(SW_3_PIN, SW_3_DEVICE);
#endif

#ifdef SW_4_PIN
  sendCommand(SW_4_PIN, SW_4_DEVICE);
#endif

#ifdef SW_5_PIN
  sendCommand(SW_5_PIN, SW_5_DEVICE);
#endif

#ifdef SW_6_PIN
  sendCommand(SW_6_PIN, SW_6_DEVICE);
#endif

#ifdef SW_A_PIN
if ((digitalRead(SW_A_PIN) == 0) && (!paused))
{

  allLed(0);
  Serial.write("@p");   // write on serial KNXgate/SCSgate
  delay(100);
  paused = 1000;
  allLed(1);
}
#endif

  if (paused)
  {
    paused--;
    delayMicroseconds(100);
  }
// ================================================================================================
  timeled++;
  if (timeled > 32700)
  {
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
    timeled = 0;
  }
  else
  if (timeled == 32000)
    digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on

}
```

SCSMULTICOMMAND2

```
/*
  SCS multi - command - 6  SWITCHES   +   6 ACTUATORS/LED

  ------------------------------------------------------------------------------
  SW basso -> TOGGLE
  COME SCSMULTICOMMAND1 MA con seriale su pin 10-11 e monitor su 0/1 (usb)
  ------------------------------------------------------------------------------
        connect pin 10 (rx) with scsgate uart tx
        connect pin 11 (tx) with scsgate uart rx
        connect pin GND     with scsgate GND
        connect pin +3.3V   with scsgate +3.3V
```

```
 ------------------------------------------------------------------------------
  by Guido Pagani as example of SCSgate utilization
 */

#define VERSION "SCScommand V2.3"
//#define DEBUG
#define TRACE

#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX

// ================================================================================================================
//                      DEFINIZIONI  DA  PERSONALIZZARE
// ================================================================================================================

//  SWITCH ARDUINO:   ETICHETTA SCS DISPOSITIVI DA ACCENDERE / SPEGNERE

#define SW_1_DEVICE   0x31
#define SW_2_DEVICE   0x32
#define SW_3_DEVICE   0x33
#define SW_4_DEVICE   0x34
#define SW_5_DEVICE   0x35
#define SW_6_DEVICE   0x36

//  SWITCH ARDUINO:   PIN ARDUINO A CUI SONO COLLEGATI gli SWITCH
//                    ogni SW accende/spegne alterntivamente la luce scs corrispondente

#define SW_1_PIN      8
#define SW_2_PIN      9
#define SW_3_PIN      12
#define SW_4_PIN      13
#define SW_5_PIN      14
#define SW_6_PIN      15
#define SW_A_PIN      16  // e' una sorta di reset interno degli stati degli sw e led


//  LED O ATTUATORI ARDUINO: ETICHETTA SCS A CUI CORRISPONDONO

#define LED_1_DEVICE  0x31
#define LED_2_DEVICE  0x32
#define LED_3_DEVICE  0x33
#define LED_4_DEVICE  0x34
#define LED_5_DEVICE  0x35
#define LED_6_DEVICE  0x36

//  LED O ATTUATORI ARDUINO:   PIN ARDUINO A CUI SONO COLLEGATI

#define LED_1_PIN    2
#define LED_2_PIN    3
#define LED_3_PIN    4
#define LED_4_PIN    5
#define LED_5_PIN    6
#define LED_6_PIN    7

// ================================================================================================================
//                      FINE  DEFINIZIONI  DA  PERSONALIZZARE
// ================================================================================================================



// ================================================================================
unsigned char mylen, myindex, s, m, a, b, t;
unsigned char mySerialStatus = 0;  // 1=reading  2=readed
int  mytime;
unsigned char dDevice = 0;
unsigned char dStato = 0xFF;
unsigned char sBufferIn[16];
unsigned char sBufferOu[16];
unsigned char scsStato[32];
unsigned char swStato[32];
int  paused;
int  timeled = 0;
// ================================================================================
void setup()
{
  pinMode(SW_1_PIN, INPUT_PULLUP);
  pinMode(SW_2_PIN, INPUT_PULLUP);
  pinMode(SW_3_PIN, INPUT_PULLUP);
  pinMode(SW_4_PIN, INPUT_PULLUP);
  pinMode(SW_5_PIN, INPUT_PULLUP);
  pinMode(SW_6_PIN, INPUT_PULLUP);
  pinMode(SW_A_PIN, INPUT_PULLUP);

  pinMode(LED_1_PIN, OUTPUT);
  pinMode(LED_2_PIN, OUTPUT);
```

```
  pinMode(LED_3_PIN, OUTPUT);
  pinMode(LED_4_PIN, OUTPUT);
  pinMode(LED_5_PIN, OUTPUT);
  pinMode(LED_6_PIN, OUTPUT);

  for (s=0; s<sizeof(scsStato); s++)
  {
    scsStato[s] = 0xFF;
    swStato[s] = 0xFF;
  }

  allLed(0);

  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while (!Serial) { ; }
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on
#ifdef DEBUG
  Serial.println(VERSION " started!");
#endif
#ifdef TRACE
  Serial.println(VERSION " started!");
#endif
  // set the data rate for the SoftwareSerial port
  mySerial.begin(115200);
  mySerial.write("@");  // set SCSgate/KNXgate slow speed
  mySerial.write(0xEE);  // set SCSgate/KNXgate very slow speed (19200 baud)
  delay(100);            // wait 100ms
  mySerial.end();

  // reopen serial port at slow speed
  mySerial.begin(19200);

  mySerial.flush();

  mySerial.write("@");
  mySerial.write(0x15);    // avoid to memorize next setup in eeprom
  delay(10);            // wait 10ms
  mySerial.write("@MX");  // set SCSgate hex mode
  delay(10);            // wait 10ms
  mySerial.write("@Y1");  // set read short
  delay(10);            // wait 10ms
  mySerial.write("@F3");  // filter set for ignore ack and double msg
  delay(10);            // wait 10ms
  mySerial.write("@b");   // clear  buffer
  mySerial.setTimeout(2); // timeout is 2mS
  delay(10);            // wait 10ms
  mySerial.write("@l");   // log mode
  delay(10);            // wait 10ms
  mySerial.flush();
  paused = 0;
  allLed(1);
}

void allLed(unsigned char stato)
{
  digitalWrite(LED_1_PIN, stato);
  digitalWrite(LED_2_PIN, stato);
  digitalWrite(LED_3_PIN, stato);
  digitalWrite(LED_4_PIN, stato);
  digitalWrite(LED_5_PIN, stato);
  digitalWrite(LED_6_PIN, stato);
}
// ================================================================================================
void setupStato(void)
{
  a = dDevice>>3;
  b = dDevice & 0x07;
  dStato &= 0x01;
#ifdef DEBUG
  Serial.print("bw: ");
  Serial.print(a);
  Serial.print(" bit:");
  Serial.println(b);
#endif
#ifdef TRACE
  if (dStato == 0)
  {
        Serial.print("acceso: ");
    Serial.println(dDevice,HEX);
  }
  else
  {
        Serial.print("spento: ");
    Serial.println(dDevice,HEX);
  }
#endif
```

```
  bitWrite(scsStato[a], b, dStato);

  switch (dDevice)
  {
#ifdef LED_1_PIN
    case LED_1_DEVICE:
          digitalWrite(LED_1_PIN, dStato);
          break;
#endif

#ifdef LED_2_PIN
    case LED_2_DEVICE:
          digitalWrite(LED_2_PIN, dStato);
          break;
#endif

#ifdef LED_3_PIN
    case LED_3_DEVICE:
          digitalWrite(LED_3_PIN, dStato);
          break;
#endif

#ifdef LED_4_PIN
    case LED_4_DEVICE:
          digitalWrite(LED_4_PIN, dStato);
          break;
#endif

#ifdef LED_5_PIN
    case LED_5_DEVICE:
          digitalWrite(LED_5_PIN, dStato);
          break;
#endif

#ifdef LED_6_PIN
    case LED_6_DEVICE:
          digitalWrite(LED_6_PIN, dStato);
          break;
#endif

  }
}
// ================================================================================================
unsigned char readStato(unsigned char address)
{
  a = address>>3;
  b = address & 0x07;
  return bitRead(scsStato[a], b);
}
  // ================================================================================================
char receiveCommand(void) // try to receive telegram and setup state
{
// intero:    07 A8 16 00 12 01 05 A3
//            07 A8 B8 16 12 01 BD A3
// compresso: 04 16 00 12 01 (ignoro)
//            04 B8 16 12 01
  if (mySerialStatus == 0)
  {
        mylen = mySerial.read();
        if ((mylen > 0) && (mylen < 5) && (mylen != 0xFF))
        {
#ifdef DEBUG
      Serial.print("L: ");
      Serial.print(mylen, HEX);
      Serial.print(" -> X: [");
#endif
      mySerialStatus = 1;
      myindex = 0;
      mytime = 0;
        }
    return 0;
  }
  if (mySerialStatus == 1)
  {
    if (mytime++ > 30000)
    {
#ifdef DEBUG
      Serial.println("<T>");
#endif
      mySerialStatus = 0;
    }
    sBufferIn[myindex] = mySerial.read();
    if (sBufferIn[myindex] == 0xFF)
      return 0;
#ifdef DEBUG
    Serial.print(sBufferIn[myindex], HEX);
#endif
```

```
    myindex++;
    if (myindex >= mylen)
    {
      mySerialStatus = 2;
#ifdef DEBUG
      Serial.println("]");
#endif
    }
#ifdef DEBUG
    else
      Serial.print(",");
#endif
  }

  if (mySerialStatus == 2)
  {
    mySerialStatus = 0;
    if (mylen == 4)
    {
      if (sBufferIn[0] != 0xB8)
      {
#ifdef DEBUG
        Serial.print("r: ");
        Serial.println(sBufferIn[0], HEX);
#endif
        return 0;
      }
      else
      {
#ifdef DEBUG
        Serial.print("R: ");
        Serial.print(sBufferIn[1], HEX);
#endif
        if ((dDevice != sBufferIn[1]) || (dStato != sBufferIn[3]))
        {
#ifdef DEBUG
                Serial.print(" -S: ");
                     Serial.print(sBufferIn[3],HEX);
#endif
                             dDevice = sBufferIn[1];
                             dStato = sBufferIn[3];
                             setupStato();
        }
#ifdef DEBUG
        Serial.println(".");
#endif
                return 1;
      }
    }
#ifdef DEBUG
    else
    {
      Serial.println("?");
//    mySerial.flush();
    }
#endif
  }
  return 0;
}
// ================================================================================================
void sendCommand(unsigned char index, unsigned char address) // send command if digital input high
{
  unsigned char tasto;
  tasto = digitalRead((int)index);

  if (tasto != swStato[index])
  {
#ifdef DEBUG
    Serial.print("CHG ");
    Serial.println(tasto,HEX);
#endif
    swStato[index] = tasto;
    if (tasto == 0)
    {
      if (!paused)
      {
        sBufferOu[0] = '@';
        sBufferOu[1] = 'w';

        sBufferOu[2] = (readStato(address)) ^ 0x01;
        sBufferOu[3] = address;

        s = 4;
        m = 0;
#ifdef DEBUG
        Serial.print("W: ");
        Serial.print(sBufferOu[3], HEX);
```

```
        Serial.print("S: ");
        Serial.println(sBufferOu[2]);
#endif
#ifdef TRACE
               Serial.print("switch: ");
        Serial.println(index);
#endif

        while (s)
        {
          mySerial.write(sBufferOu[m++]);   // write on serial SCSgate
          s--;
          delayMicroseconds(50);
        }
        paused = 2000; // x 0,1mS = 200mS
        mySerial.flush();
      }
    }
    else
        paused = 2000; // x 0,1mS = 200mS

  }
}
// ================================================================================================



// ================================================================================================
void loop() // run over and over
{

// ================================================================================================
  receiveCommand();

#ifdef SW_1_PIN
  sendCommand(SW_1_PIN, SW_1_DEVICE);   // ARDUINO PHISICAL PORT ADDRESS,  SWITCH NUMBER (is device address)
#endif

#ifdef SW_2_PIN
  sendCommand(SW_2_PIN, SW_2_DEVICE);
#endif

#ifdef SW_3_PIN
  sendCommand(SW_3_PIN, SW_3_DEVICE);
#endif

#ifdef SW_4_PIN
  sendCommand(SW_4_PIN, SW_4_DEVICE);
#endif

#ifdef SW_5_PIN
  sendCommand(SW_5_PIN, SW_5_DEVICE);
#endif

#ifdef SW_6_PIN
  sendCommand(SW_6_PIN, SW_6_DEVICE);
#endif

#ifdef SW_A_PIN
if ((digitalRead(SW_A_PIN) == 0) && (!paused))
{
#ifdef DEBUG
  Serial.println("@p");
#endif

  allLed(0);
  mySerial.write("@p");   // write on serial KNXgate/SCSgate
  delay(100);
  paused = 1000;
  allLed(1);
}
#endif

  if (paused)
  {
    paused--;
    delayMicroseconds(100);
  }
// ================================================================================================
  timeled++;
  if (timeled > 32700)
  {
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
    timeled = 0;
  }
  else
  if (timeled == 32000)
```

```
    digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on

}
```

SCSMULTICOMMAND3

```
/*
  SCS multi - command - 6  SWITCHES   +   6 ACTUATORS/LED

  ------------------------------------------------------------------------------------
  SW alto -> ON    SW basso -> OFF
  ------------------------------------------------------------------------------------
        connect pin 0 (rx)  with scsgate tx
        connect pin 1 (tx)  with scsgate rx
        connect pin GND     with scsgate GND
        connect pin +3.3V   with scsgate +3.3V
 ------------------------------------------------------------------------------------
  by Guido Pagani as example of SCSgate utilization
 */

#define VERSION "SCScommand V4.3"

// ================================================================================================================
//                      DEFINIZIONI  DA  PERSONALIZZARE
// ================================================================================================================

//  SWITCH ARDUINO:   ETICHETTA SCS DISPOSITIVI DA ACCENDERE / SPEGNERE

#define SW_1_DEVICE   0x31
#define SW_2_DEVICE   0x32
#define SW_3_DEVICE   0x33
#define SW_4_DEVICE   0x34
#define SW_5_DEVICE   0x35
#define SW_6_DEVICE   0x36

//  SWITCH ARDUINO:   PIN ARDUINO A CUI SONO COLLEGATI gli SWITCH
//                    ogni SW accende/spegne alterntivamente la luce scs corrispondente

#define SW_1_PIN      8
#define SW_2_PIN      9
#define SW_3_PIN      12
#define SW_4_PIN      13
#define SW_5_PIN      14
#define SW_6_PIN      15
#define SW_A_PIN      16  // e' una sorta di reset interno degli stati degli sw e led


//  LED O ATTUATORI ARDUINO: ETICHETTA SCS A CUI CORRISPONDONO

#define LED_1_DEVICE  0x31
#define LED_2_DEVICE  0x32
#define LED_3_DEVICE  0x33
#define LED_4_DEVICE  0x34
#define LED_5_DEVICE  0x35
#define LED_6_DEVICE  0x36

//  LED O ATTUATORI ARDUINO:   PIN ARDUINO A CUI SONO COLLEGATI

#define LED_1_PIN    2
#define LED_2_PIN    3
#define LED_3_PIN    4
#define LED_4_PIN    5
#define LED_5_PIN    6
#define LED_6_PIN    7

// ================================================================================================================
//                      FINE  DEFINIZIONI  DA  PERSONALIZZARE
// ================================================================================================================



// ================================================================================
unsigned char mylen, myindex, s, m, a, b, t;
unsigned char SerialStatus = 0;  // 1=reading  2=readed
int  mytime;
unsigned char dDevice = 0;
unsigned char dStato = 0xFF;
unsigned char sBufferIn[16];
unsigned char sBufferOu[16];
unsigned char scsStato[32];
unsigned char swStato[32];
```

```
int  paused;
int  timeled = 0;
// ================================================================================================
void setup()
{
  pinMode(SW_1_PIN, INPUT_PULLUP);
  pinMode(SW_2_PIN, INPUT_PULLUP);
  pinMode(SW_3_PIN, INPUT_PULLUP);
  pinMode(SW_4_PIN, INPUT_PULLUP);
  pinMode(SW_5_PIN, INPUT_PULLUP);
  pinMode(SW_6_PIN, INPUT_PULLUP);
  pinMode(SW_A_PIN, INPUT_PULLUP);

  pinMode(LED_1_PIN, OUTPUT);
  pinMode(LED_2_PIN, OUTPUT);
  pinMode(LED_3_PIN, OUTPUT);
  pinMode(LED_4_PIN, OUTPUT);
  pinMode(LED_5_PIN, OUTPUT);
  pinMode(LED_6_PIN, OUTPUT);

  for (s=0; s<sizeof(scsStato); s++)
  {
    scsStato[s] = 0xFF;
    swStato[s] = 0xFF;
  }

  allLed(0);

  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while (!Serial) { ; }
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on
  // set the data rate for the SoftwareSerial port
  Serial.begin(115200);
  Serial.flush();

  Serial.write("@");
  Serial.write(0x15);    // avoid to memorize next setup in eeprom
  delay(10);             // wait 10ms
  Serial.write("@MX");  // set SCSgate hex mode
  delay(10);             // wait 10ms
  Serial.write("@Y1");  // set read short
  delay(10);             // wait 10ms
  Serial.write("@F3");  // filter set for ignore ack and double msg
  delay(10);             // wait 10ms
  Serial.write("@b");   // clear  buffer
  Serial.setTimeout(2); // timeout is 2mS
  delay(10);             // wait 10ms
  Serial.write("@l");   // log mode
  delay(10);             // wait 10ms
  Serial.flush();
  paused = 0;
  allLed(1);
}

void allLed(unsigned char stato)
{
  digitalWrite(LED_1_PIN, stato);
  digitalWrite(LED_2_PIN, stato);
  digitalWrite(LED_3_PIN, stato);
  digitalWrite(LED_4_PIN, stato);
  digitalWrite(LED_5_PIN, stato);
  digitalWrite(LED_6_PIN, stato);
}
// ================================================================================================
void setupStato(void)
{
  a = dDevice>>3;
  b = dDevice & 0x07;
  dStato &= 0x01;
  bitWrite(scsStato[a], b, dStato);

  switch (dDevice)
  {
#ifdef LED_1_PIN
    case LED_1_DEVICE:
          digitalWrite(LED_1_PIN, dStato);
          break;
#endif

#ifdef LED_2_PIN
    case LED_2_DEVICE:
          digitalWrite(LED_2_PIN, dStato);
          break;
#endif

#ifdef LED_3_PIN
```

```
    case LED_3_DEVICE:
          digitalWrite(LED_3_PIN, dStato);
          break;
#endif

#ifdef LED_4_PIN
    case LED_4_DEVICE:
          digitalWrite(LED_4_PIN, dStato);
          break;
#endif

#ifdef LED_5_PIN
    case LED_5_DEVICE:
          digitalWrite(LED_5_PIN, dStato);
          break;
#endif

#ifdef LED_6_PIN
    case LED_6_DEVICE:
          digitalWrite(LED_6_PIN, dStato);
          break;
#endif

  }
}
// ================================================================================
unsigned char readStato(unsigned char address)
{
  a = address>>3;
  b = address & 0x07;
  return bitRead(scsStato[a], b);
}
  // ================================================================================
char receiveCommand(void) // try to receive telegram and setup state
{
// intero:    07 A8 16 00 12 01 05 A3
//            07 A8 B8 16 12 01 BD A3
// compresso: 04 16 00 12 01 (ignoro)
//            04 B8 16 12 01
  if (SerialStatus == 0)
  {
        mylen = Serial.read();
        if ((mylen > 0) && (mylen < 5) && (mylen != 0xFF))
        {
      SerialStatus = 1;
      myindex = 0;
      mytime = 0;
        }
    return 0;
  }
  if (SerialStatus == 1)
  {
    if (mytime++ > 30000)
    {
      SerialStatus = 0;
    }
    sBufferIn[myindex] = Serial.read();
    if (sBufferIn[myindex] == 0xFF)
      return 0;
    myindex++;
    if (myindex >= mylen)
    {
      SerialStatus = 2;
    }
  }

  if (SerialStatus == 2)
  {
    SerialStatus = 0;
    if (mylen == 4)
    {
      if (sBufferIn[0] != 0xB8)
      {
        return 0;
      }
      else
      {
        if ((dDevice != sBufferIn[1]) || (dStato != sBufferIn[3]))
        {
                            dDevice = sBufferIn[1];
                            dStato = sBufferIn[3];
                            setupStato();
        }
                return 1;
      }
    }
  }
```

```
  return 0;
}
// ================================================================================================
void sendCommand(unsigned char index, unsigned char address) // send command if digital input high
{
  unsigned char tasto;

  tasto = 0xFF;

  pinMode((int)index, INPUT_PULLUP);
//  digitalWrite((int)index, HIGH);
  if (digitalRead((int)index) == 0) tasto = 0;  // LOW = spegni

  pinMode((int)index, OUTPUT);
  digitalWrite((int)index, LOW);
  pinMode((int)index, INPUT);
  if (digitalRead((int)index) == 1) tasto = 1;  // HIGH = accendi

  if ((tasto != 0xFF) && (tasto != swStato[index]) && (!paused))
  {
    swStato[index] = tasto;
    sBufferOu[0] = '@';
    sBufferOu[1] = 'w';

    sBufferOu[2] = tasto ^ 0x01;
    sBufferOu[3] = address;

    s = 4;
    m = 0;

    while (s)
    {
      Serial.write(sBufferOu[m++]);   // write on serial SCSgate
      s--;
      delayMicroseconds(50);
    }
    paused = 2000; // x 0,1mS = 200mS
    Serial.flush();
  }
}
// ================================================================================================



// ================================================================================================
void loop() // run over and over
{

// ================================================================================================
  receiveCommand();

#ifdef SW_1_PIN
  sendCommand(SW_1_PIN, SW_1_DEVICE);   // ARDUINO PHISICAL PORT ADDRESS,  SWITCH NUMBER (is device address)
#endif

#ifdef SW_2_PIN
  sendCommand(SW_2_PIN, SW_2_DEVICE);
#endif

#ifdef SW_3_PIN
  sendCommand(SW_3_PIN, SW_3_DEVICE);
#endif

#ifdef SW_4_PIN
  sendCommand(SW_4_PIN, SW_4_DEVICE);
#endif

#ifdef SW_5_PIN
  sendCommand(SW_5_PIN, SW_5_DEVICE);
#endif

#ifdef SW_6_PIN
  sendCommand(SW_6_PIN, SW_6_DEVICE);
#endif

#ifdef SW_A_PIN
if ((digitalRead(SW_A_PIN) == 0) && (!paused))
{

  allLed(0);
  Serial.write("@p");   // write on serial KNXgate/SCSgate
  delay(100);
  paused = 1000;
  allLed(1);
}
#endif
```

```
  if (paused)
  {
    paused--;
    delayMicroseconds(100);
  }
// ================================================================================
  timeled++;
  if (timeled > 32700)
  {
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
    timeled = 0;
  }
  else
  if (timeled == 32000)
    digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on

}
```

SCSMULTICOMMAND4

```
/*
  SCS multi - command - 6  SWITCHES   +   6 ACTUATORS/LED

  ---------------------------------------------------------------------------------
  SW alto -> ON    SW basso -> OFF
  COME SCSMULTICOMMAND3 MA con seriale su pin 10-11 e monitor su 0/1 (usb)
  ---------------------------------------------------------------------------------
        connect pin 10 (rx) with scsgate uart tx
        connect pin 11 (tx) with scsgate uart rx
        connect pin GND     with scsgate GND
        connect pin +3.3V   with scsgate +3.3V
 ---------------------------------------------------------------------------------
  by Guido Pagani as example of SCSgate utilization
 */

#define VERSION "SCScommand V4.3"
#define DEBUG
//#define TRACE

#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX

// ================================================================================================================
//                       DEFINIZIONI  DA  PERSONALIZZARE
// ================================================================================================================

//  SWITCH ARDUINO:   ETICHETTA SCS DISPOSITIVI DA ACCENDERE / SPEGNERE

#define SW_1_DEVICE   0x31
#define SW_2_DEVICE   0x32
#define SW_3_DEVICE   0x33
#define SW_4_DEVICE   0x34
#define SW_5_DEVICE   0x35
#define SW_6_DEVICE   0x36

//  SWITCH ARDUINO:   PIN ARDUINO A CUI SONO COLLEGATI gli SWITCH
//                    ogni SW accende/spegne alterntivamente la luce scs corrispondente

#define SW_1_PIN      8
#define SW_2_PIN      9
#define SW_3_PIN      12
#define SW_4_PIN      13
#define SW_5_PIN      14
#define SW_6_PIN      15
#define SW_A_PIN      16  // e' una sorta di reset interno degli stati degli sw e led


//  LED O ATTUATORI ARDUINO: ETICHETTA SCS A CUI CORRISPONDONO

#define LED_1_DEVICE  0x31
#define LED_2_DEVICE  0x32
#define LED_3_DEVICE  0x33
#define LED_4_DEVICE  0x34
#define LED_5_DEVICE  0x35
#define LED_6_DEVICE  0x36

//  LED O ATTUATORI ARDUINO:   PIN ARDUINO A CUI SONO COLLEGATI

#define LED_1_PIN    2
```

```
#define LED_2_PIN    3
#define LED_3_PIN    4
#define LED_4_PIN    5
#define LED_5_PIN    6
#define LED_6_PIN    7

// ================================================================================================================
//                      FINE  DEFINIZIONI  DA  PERSONALIZZARE
// ================================================================================================================



// ================================================================================================
unsigned char mylen, myindex, s, m, a, b, t;
unsigned char mySerialStatus = 0;  // 1=reading  2=readed
int  mytime;
unsigned char dDevice = 0;
unsigned char dStato = 0xFF;
unsigned char sBufferIn[16];
unsigned char sBufferOu[16];
unsigned char scsStato[32];
unsigned char swStato[32];
int  paused;
int  timeled = 0;
// ================================================================================================
void setup()
{
  pinMode(SW_1_PIN, INPUT_PULLUP);
  pinMode(SW_2_PIN, INPUT_PULLUP);
  pinMode(SW_3_PIN, INPUT_PULLUP);
  pinMode(SW_4_PIN, INPUT_PULLUP);
  pinMode(SW_5_PIN, INPUT_PULLUP);
  pinMode(SW_6_PIN, INPUT_PULLUP);
  pinMode(SW_A_PIN, INPUT_PULLUP);

  pinMode(LED_1_PIN, OUTPUT);
  pinMode(LED_2_PIN, OUTPUT);
  pinMode(LED_3_PIN, OUTPUT);
  pinMode(LED_4_PIN, OUTPUT);
  pinMode(LED_5_PIN, OUTPUT);
  pinMode(LED_6_PIN, OUTPUT);

  for (s=0; s<sizeof(scsStato); s++)
  {
    scsStato[s] = 0xFF;
    swStato[s] = 0xFF;
  }

  allLed(0);

  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while (!Serial) { ; }
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on
#ifdef DEBUG
  Serial.println(VERSION " started!");
#endif
#ifdef TRACE
  Serial.println(VERSION " started!");
#endif
  // set the data rate for the SoftwareSerial port
  mySerial.begin(115200);
  mySerial.write("@");  // set SCSgate/KNXgate slow speed
  mySerial.write(0xEE);  // set SCSgate/KNXgate very slow speed (19200 baud)
  delay(100);            // wait 100ms
  mySerial.end();

  // reopen serial port at slow speed
  mySerial.begin(19200);

  mySerial.flush();

  mySerial.write("@");
  mySerial.write(0x15);    // avoid to memorize next setup in eeprom
  delay(10);             // wait 10ms
  mySerial.write("@MX");  // set SCSgate hex mode
  delay(10);             // wait 10ms
  mySerial.write("@Y1");  // set read short
  delay(10);             // wait 10ms
  mySerial.write("@F3");  // filter set for ignore ack and double msg
  delay(10);             // wait 10ms
  mySerial.write("@b");   // clear  buffer
  mySerial.setTimeout(2); // timeout is 2mS
  delay(10);             // wait 10ms
  mySerial.write("@l");   // log mode
  delay(10);             // wait 10ms
```

```
  mySerial.flush();
  paused = 0;
  allLed(1);
}

void allLed(unsigned char stato)
{
  digitalWrite(LED_1_PIN, stato);
  digitalWrite(LED_2_PIN, stato);
  digitalWrite(LED_3_PIN, stato);
  digitalWrite(LED_4_PIN, stato);
  digitalWrite(LED_5_PIN, stato);
  digitalWrite(LED_6_PIN, stato);
}
// ================================================================================
void setupStato(void)
{
  a = dDevice>>3;
  b = dDevice & 0x07;
  dStato &= 0x01;
#ifdef DEBUG
  Serial.print("bw: ");
  Serial.print(a);
  Serial.print(" bit:");
  Serial.println(b);
#endif
#ifdef TRACE
  if (dStato == 0)
  {
        Serial.print("acceso: ");
    Serial.println(dDevice,HEX);
  }
  else
  {
        Serial.print("spento: ");
    Serial.println(dDevice,HEX);
  }
#endif
  bitWrite(scsStato[a], b, dStato);

  switch (dDevice)
  {
#ifdef LED_1_PIN
    case LED_1_DEVICE:
          digitalWrite(LED_1_PIN, dStato);
          break;
#endif

#ifdef LED_2_PIN
    case LED_2_DEVICE:
          digitalWrite(LED_2_PIN, dStato);
          break;
#endif

#ifdef LED_3_PIN
    case LED_3_DEVICE:
          digitalWrite(LED_3_PIN, dStato);
          break;
#endif

#ifdef LED_4_PIN
    case LED_4_DEVICE:
          digitalWrite(LED_4_PIN, dStato);
          break;
#endif

#ifdef LED_5_PIN
    case LED_5_DEVICE:
          digitalWrite(LED_5_PIN, dStato);
          break;
#endif

#ifdef LED_6_PIN
    case LED_6_DEVICE:
          digitalWrite(LED_6_PIN, dStato);
          break;
#endif

  }
}
// ================================================================================
unsigned char readStato(unsigned char address)
{
  a = address>>3;
  b = address & 0x07;
  return bitRead(scsStato[a], b);
}
```

```
  // ================================================================================
char receiveCommand(void) // try to receive telegram and setup state
{
// intero:    07 A8 16 00 12 01 05 A3
//            07 A8 B8 16 12 01 BD A3
// compresso: 04 16 00 12 01 (ignoro)
//            04 B8 16 12 01
  if (mySerialStatus == 0)
  {
        mylen = mySerial.read();
        if ((mylen > 0) && (mylen < 5) && (mylen != 0xFF))
        {
#ifdef DEBUG
      Serial.print("L: ");
      Serial.print(mylen, HEX);
      Serial.print(" -> X: [");
#endif
      mySerialStatus = 1;
      myindex = 0;
      mytime = 0;
         }
    return 0;
  }
  if (mySerialStatus == 1)
  {
    if (mytime++ > 30000)
    {
#ifdef DEBUG
      Serial.println("<T>");
#endif
      mySerialStatus = 0;
    }
    sBufferIn[myindex] = mySerial.read();
    if (sBufferIn[myindex] == 0xFF)
      return 0;
#ifdef DEBUG
    Serial.print(sBufferIn[myindex], HEX);
#endif
    myindex++;
    if (myindex >= mylen)
    {
      mySerialStatus = 2;
#ifdef DEBUG
      Serial.println("]");
#endif
    }
#ifdef DEBUG
    else
      Serial.print(",");
#endif
  }

  if (mySerialStatus == 2)
  {
    mySerialStatus = 0;
    if (mylen == 4)
    {
      if (sBufferIn[0] != 0xB8)
      {
#ifdef DEBUG
        Serial.print("r: ");
        Serial.println(sBufferIn[0], HEX);
#endif
        return 0;
      }
      else
      {
#ifdef DEBUG
        Serial.print("R: ");
        Serial.print(sBufferIn[1], HEX);
#endif
        if ((dDevice != sBufferIn[1]) || (dStato != sBufferIn[3]))
        {
#ifdef DEBUG
                Serial.print(" -S: ");
                    Serial.print(sBufferIn[3],HEX);
#endif
                            dDevice = sBufferIn[1];
                            dStato = sBufferIn[3];
                            setupStato();
        }
#ifdef DEBUG
        Serial.println(".");
#endif
                return 1;
      }
    }
```

```
#ifdef DEBUG
    else
    {
      Serial.println("?");
//    mySerial.flush();
    }
#endif
  }
  return 0;
}
// ================================================================================
void sendCommand(unsigned char index, unsigned char address) // send command if digital input high
{
  unsigned char tasto;

  tasto = 0xFF;

  pinMode((int)index, INPUT_PULLUP);
//  digitalWrite((int)index, HIGH);
  if (digitalRead((int)index) == 0) tasto = 0;  // LOW = spegni

  pinMode((int)index, OUTPUT);
  digitalWrite((int)index, LOW);
  pinMode((int)index, INPUT);
  if (digitalRead((int)index) == 1) tasto = 1;  // HIGH = accendi

  if ((tasto != 0xFF) && (tasto != swStato[index]) && (!paused))
  {
#ifdef DEBUG
    Serial.print("CHG ");
    Serial.println(tasto,HEX);
#endif
    swStato[index] = tasto;
    sBufferOu[0] = '@';
    sBufferOu[1] = 'w';

    sBufferOu[2] = tasto ^ 0x01;
    sBufferOu[3] = address;

    s = 4;
    m = 0;
#ifdef DEBUG
        Serial.print("W: ");
        Serial.print(sBufferOu[3], HEX);
        Serial.print("S: ");
        Serial.println(sBufferOu[2]);
#endif
#ifdef TRACE
              Serial.print("switch: ");
        Serial.println(index);
#endif

    while (s)
    {
      mySerial.write(sBufferOu[m++]);   // write on serial SCSgate
      s--;
      delayMicroseconds(50);
    }
    paused = 2000; // x 0,1mS = 200mS
    mySerial.flush();
  }
}
// ================================================================================



// ================================================================================
void loop() // run over and over
{

// ================================================================================
  receiveCommand();

#ifdef SW_1_PIN
  sendCommand(SW_1_PIN, SW_1_DEVICE);   // ARDUINO PHISICAL PORT ADDRESS,  SWITCH NUMBER (is device address)
#endif

#ifdef SW_2_PIN
  sendCommand(SW_2_PIN, SW_2_DEVICE);
#endif

#ifdef SW_3_PIN
  sendCommand(SW_3_PIN, SW_3_DEVICE);
#endif

#ifdef SW_4_PIN
  sendCommand(SW_4_PIN, SW_4_DEVICE);
```

```
#endif

#ifdef SW_5_PIN
  sendCommand(SW_5_PIN, SW_5_DEVICE);
#endif

#ifdef SW_6_PIN
  sendCommand(SW_6_PIN, SW_6_DEVICE);
#endif

#ifdef SW_A_PIN
if ((digitalRead(SW_A_PIN) == 0) && (!paused))
{
#ifdef DEBUG
  Serial.println("@p");
#endif

  allLed(0);
  mySerial.write("@p");   // write on serial KNXgate/SCSgate
  delay(100);
  paused = 1000;
  allLed(1);
}
#endif

  if (paused)
  {
    paused--;
    delayMicroseconds(100);
  }
// ================================================================================
  timeled++;
  if (timeled > 32700)
  {
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
    timeled = 0;
  }
  else
  if (timeled == 32000)
    digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on

}
```

ALEXA

Sono del parere che questo tipo di interfaccia sia il futuro della domotica. Anche se ancora limitata e pomposamente chiamata “intelligenza artificiale” è comunque un ottimo primo passo (insieme ad altre realtà) nel mondo dell’interazione vocale.

Con il termine “Alexa” si definisce un processo complesso, che “gira” su dei server remoti e che interagisce con dei terminali locali “echo dot” o altro – ovviamente se non siete connessi ad internet nulla funziona.

La prima possibilità di integrazione tra raspy_scsgate ed Alexa passa attraverso HOME-ASSISTANT. Su Amazon-Alexa potete installare gratuitamente la skill di integrazione. Purtroppo questa skill utilizza il “cloud” come passaggio di integrazione, e purtroppo Amazon richiede un pedaggio mensile di qualche euro.

La seconda possibilità di integrazione tra raspy_scsgate ed Alexa passa ancora attraverso HOME-ASSISTANT. Se avete un po’ di dimestichezza con i sistemi di sviluppo software, seguendo questa guida (ce ne sono diverse) potrete scrivere (clonare) la vostra skill ed ottenere l’integrazione senza pagare canoni a nessuno: https://indomus.it/guide/integrare-gratuitamente-amazon-echo-alexa-con-home-assistant-via-haaska-e-aws/

La terza possibilità è la più semplice (e limitata) e consiste nel connettere direttamente raspy_scsgate con Alexa – il firmware di raspy_scsgate (scsgate_x) lo consente, utilizzando un software che emula un dispositivo "philips hue" (lampada intelligente), da cui derivano queste possibilità e limitazioni:

- Le luci SCS potranno essere comandate direttamente da Alexa con i comandi vocali "*accendi*" e "*spegni*"
- Le luci dimmerabili SCS potranno essere comandate direttamente da Alexa con i comandi vocali "*accendi*", "*spegni*", "*alza*", "*abbassa*", "*imposta al xx%*"
- Le tapparelle SCS, se gestite senza %, potranno essere comandate direttamente da Alexa con i comandi vocali "*alza*" , "*abbassa*" , "*ferma*" – il comando "ferma" arresterà il movimento della tapparella ma l'interfaccia vocale risponderà che qualcosa non ha funzionato
- Le tapparelle SCS, se gestite con %, potranno essere comandate direttamente da Alexa con i comandi vocali "*alza*", "*abbassa*", "*ferma*", "*imposta al xx%*" – i comandi "alza" e "abbassa" muoveranno la tapparella in su o in giù circa del 25% a botta (per alzare o abbassare totalmente una tapparella occorrerà usare i comandi "imposta al 100%" e "imposta al 0%" – inoltre il comando "ferma" arresterà il movimento della tapparella ma l'interfaccia vocale risponderà che qualcosa non ha funzionato

I passi necessari ad utilizzare l'interfaccia diretta Alexa-esp_scsgate sono i seguenti:

- Preliminarmente dovete aver individuato i dispositivi scs e aver compilato il file scsconfig.
- Il processo di Alexa lavora sulla porta 80 – se tale porta fosse già in uso da parte di altri processi nel raspberry riceverete un errore. Dovete liberarla.
- Attivate l'applicazione scsgate_x con il parametro -H (***sudo ./scsgate_x -H***)
- Se avete più di un echo-dot lasciate spenti tutti gli echo-dot secondari, va lasciato acceso solo quello principale (solitamente il primo che si è installato)
- A questo punto andate sull'APP (o sulla pagina https://alexa.amazon.it) di ALEXA – cliccate sull'opzione "Casa intelligente" e poi su "Dispositivi". In fondo alla pagina cliccate su "Trova" e aspettate che Alexa termini il lavoro di scoperta di nuovi dispositivi
- Se nessun dispositivo viene scoperto, spegnete e riaccendete echo-dot, aspettate circa 30 secondi e riprovate
- A volte è poi necessario refreshare la pagina. A volte capita che uno dei dispositivi appaia censito due volte – non importa.

Se avete un numero cospicuo di dispositivi (più di 30) alcuni potrebbero mancare – in tal caso ripetete l'operazione di "Trova" più volte, ogni volta se ne aggiungeranno circa 30.

Avete fatto tutto – provate a dire "Alexa, accendi <nome dispositivo>

Attenzione: se correggete il nome di un dispositivo, cancellatelo dai dispositivi scoperti su Alexa e poi rifate il "Trova".

Attenzione: se aggiungete o togliete dei dispositivi dalla lista "scsconfig" dovete farli "dimenticare" TUTTI ad alexa e quindi rifare il "Trova" perché il codice interno di alexa cambia

Come ho precisato, questa è una BETA-VERSION per quanto riguarda l'accesso diretto ad Alexa, può essere che qualcosa ancora non funzioni per il verso giusto… comunicatemi il vostro problema e cercherò di risolverlo

Attenzione ai nomi che date ai dispositivi, alexa può avere difficoltà a riconoscere certi nomi o fare confusione quando più dispositivi hanno nomi simili. Usando le "routine" potreste superare il problema.

In una prossima versione il software consentirà anche di utilizzare come dipositivi i pin gpio di raspberry, in ON/OFF.

COMUNICAZIONE TCP

La connessione TCP può essere aperta solo con esp_scsgate connesso come client di un router, sulla porta 5045.

I pacchetti che arrivano sulla porta TCP vengono dapprima analizzati nel contenuto: se contengono un comando valido secondo la lista che segue esso viene eseguito. Diversamente, se è stato "aperto" il canale di comunicazione tcp-uart essi vengono inviati tal quali a SCSGATE che li interpreta esattamente come il protocollo UDP.

I dati che provengono da SCSGATE (se è stato "aperto" il canale di comunicazione tcp-uart) vengono impacchettati ed inviati tal quali all'indirizzo IP ed alla porta che hanno effettuato l'ultimo invio a ESP_SCSGATE.

I comandi validi sono:

#setup {"uart":"tcp"} apre il canale di comunicazione TCP verso SCSGATE come sopra descritto. Il canale rimane aperto sino a che non viene ricevuto un pacchetto UDP. Dopo questo comando ogni altro messaggio che non inizi con # verrà automaticamente girato a scsgate che risponderà sempre in TCP al chiamante. Le chiamate lecite e le relative risposte sono elencate nel paragrafo "il protocollo seriale di scsgate"

#setup {"uart":"no"} chiude il canale di comunicazione TCP verso SCSGATE come sopra descritto.

#request {"device":"dd", "command":"cc"} invia il comando scs "cc" al dispositivo con indirizzo "dd". Comando e indirizzo vanno espressi con caratteri ascii esadecimali (come nel protocollo UDP in modalità ascii).

Test con il PC - protocollo TCP

Può essere effettuato utilizzando l'applicativo KNXSCSGATE di cui viene anche fornito il sorgente in VB6.

Nel menu "Communication" va indicata la scelta TCP, dopodichè nella casella IP address va indicato l'indirizzo IP di raspberry. Obbligatoriamente va usata la porta 5045.

ATTENZIONE: non usate contemporaneamente knxscsgate e altri programmi che facciano chiamate tcp/http alla medesima porta.

Cliccate una o due volte il tasto "query firmware" fino a che non ricevete una risposta positiva.

Usate la funzione "serial monitor" per inviare e ricevere dati secondo il protocollo di scsgate, per esempio posizionate il cursore nella finestra a destra e digitate "h" (minuscolo). Il dispositivo deve rispondere con i settaggi correnti. La conversazione completa a video sarà circa così:

```
communication on TCP port:5045

@MAkh
SCSgate V 18.04
@M[A|X] : modo ascii | hex
@F[0|1|2|3|4] : filtro
@q : query version
@b : buffer clear
@r : read immed.
@R : read defer.
@c : cancel def.
@W[0-F][data] write
@w[value][destin] write
```

A questo punto attivate il log a video con il comando @l – il dispositivo risponde “k”. Provate ad accendere e spegnere qualche luce, a video dovreste vedere così:

```
SCS[0]: A8 32 00 12 01 21 A3
SCS[1]: A5
SCS[2]: A8 B8 32 12 01 99 A3
SCS[3]: A8 B8 32 12 01 99 A3
SCS[4]: A8 B8 32 12 01 99 A3

SCS[5]: A8 32 00 12 00 20 A3
SCS[6]: A5
SCS[7]: A8 B8 32 12 00 98 A3
SCS[8]: A8 B8 32 12 00 98 A3
SCS[9]: A8 B8 32 12 00 98 A3

SCS[0]: A8 31 00 12 01 22 A3
SCS[1]: A5
SCS[2]: A8 B8 31 12 01 9A A3
SCS[3]: A8 B8 31 12 01 9A A3
SCS[4]: A8 B8 31 12 01 9A A3
```

Le spiegazioni di quello che vedete le trovate sul documento di analisi del protocollo SCS (appunti). Il secondo byte della prima riga è l’indirizzo del dispositivo acceso o spento (nel nostro esempio 31). – evidenziati in giallo gli indirizzi dei dispositivi, in azzurro i comandi inviati e gli stati in risposta.

Provate ad accendere e spegnere il dispositivo con i comando “@w131” e “@w031” – se il dispositivo risponde siete a posto!

Ri-premete il tasto “serial monitor” per uscire da questa modalità, quindi cliccate su “Open channel” per accedere a tutte le funzioni del programma (monitoraggio stato dispositivi e accensione/spegnimento).

IL PROTOCOLLO SERIALE DI SCSGATE

Vale sia per la comunicazione TCP che UDP. I comandi che arrivano nel pacchetto wifi vengono trasposti tal quali al PIC che li interpreta e li esegue.

Lo scambio di dati viene sempre sollecitato da un comando inviato al dispositivo – ogni comando ha il seguente formato (i simboli < e > vengono utilizzati come separatori):

<@><comando><valore><dati opzionali>

Il modulo risponde sempre ma in vario modo a seconda del comando.

La connessione può essere effettuata in UDP sulla porta specificata all’atto della configurazione WiFi (default 52056), oppure in TCP sulla porta 5045 – in quest’ultimo caso riferirsi innanzitutto al paragrafo “Comunicazione TCP”

@M<valore>

Imposta la modalità di comunicazione sulla porta seriale; i valori possono essere i seguenti:

X : esadecimale (default): i dati vengono scambiati in esadecimale puro (bytes da 0x00 a 0xFF)

A : ascii: i dati vengono scambiati con caratteri ascii, per esempio l'esadecimale 0x2A va inviato con 2 caratteri, il carattere '2' seguito dal carattere 'A'. In modalità ascii la lunghezza effettiva dei dati inviati risulta quindi doppia rispetto alle lunghezze specificate che sono quindi lunghezze logiche. La modalità ascii è utile per fare test diretti con programmi standard di interfaccia seriale (tipo “putty”).

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@F<valore>

Permette di applicare un filtro sui messaggi SCS per ignorare quelli che non interessano; i valori espressi **sempre in ascii** possono essere i seguenti:

0 : nessun filtro

1 : i messaggi doppi o tripli vengono trasmessi in seriale una volta sola.

2 : i messaggi di ACK (0xA5) vengono ignorati.

3 : comprende sia il filtro “**1**” che il filtro “**2**”.

4 : i messaggi di stato vengono ignorati.

Il valore impostato viene memorizzato nella eeprom del PIC e rimane valido anche spegnendo e riaccendendo il dispositivo.

Sono accettati anche i valori 5-6-7: il filtro applicato consiste nella somma dei filtri 1-2-4 così come il valore è sommato (7=4+2+1)

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@A<tipo><numero><valore>

Permette di applicare un filtro su di un byte dei messaggi SCS per ignorare quelli che non interessano; i valori espressi **sempre in ascii** possono essere i seguenti:

tipo: può valere “**i**” (includi) oppure “**e**” (escludi) oppure “**0**” o altro (rimuovi il filtro)

numero: da “**1**” a “**9**” indica il numero del byte da osservare in ogni telegramma

valore: da “**00**” a “**FF**” indica il valore che deve essere trovato su tale byte per escludere o includere il telegramma tra quelli ricevuti

Il filtro impostato viene memorizzato nella eeprom del PIC e rimane valido anche spegnendo e riaccendendo il dispositivo.

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@B<tipo><numero><valore>

Permette di applicare un filtro su di un byte dei messaggi SCS per ignorare quelli che non interessano; i valori espressi **sempre in ascii** possono essere i seguenti:

tipo: può valere "**i**" (includi) oppure "**e**" (escludi) oppure "**0**" o altro (rimuovi il filtro)

numero: da "**1**" a "**9**" indica il numero del byte da osservare in ogni telegramma

valore: da "**00**" a "**FF**" indica il valore che deve essere trovato su tale byte per escludere o includere il telegramma tra quelli ricevuti

Il filtro impostato viene memorizzato nella eeprom del PIC e rimane valido anche spegnendo e riaccendendo il dispositivo.

Risposte possibili: <**k**> : tutto ok <**E**> : errore

Nota sui filtri: utilizzando più di un filtro (@F - @A - @B) questi vengono applicati sequenzialmente e i telegrammi in uscita devono superare TUTTI i filtri; se un solo filtro nega l'uscita il telegramma viene scartato.

@r

Permette di leggere dal dispositivo il successivo telegramma ricevuto e bufferizzato; la lettura lo rimuove dal buffer. Il dispositivo ha un buffer in grado di contenere fino a 10 telegrammi; la mancata lettura dei messaggi provoca un overflow del buffer con conseguente perdita dei telegrammi più vecchi. Se è stata impostata l'opzione di abbreviazione (vedi @Y) i dati saranno riportati in forma abbreviata.

Risposta:

<lunghezza> : sempre e solo un carattere (ascii o esadecimale puro) da '**0**' a '**F**' che indica la lunghezza logica dei dati rimanenti (se vale '0' significa che non ci sono telegrammi in attesa di scodamento)

<dati> : i dati del telegramma in esadecimale puro oppure in ascii a seconda del modo operativo

@R

Permette di leggere dal dispositivo il successivo telegramma ricevuto e bufferizzato; la lettura lo rimuove dal buffer. Il dispositivo ha un buffer in grado di contenere fino a 10 telegrammi; la mancata lettura dei messaggi provoca un overflow del buffer con conseguente perdita dei telegrammi più vecchi. Se il buffer non contiene messaggi la risposta viene differita: la risposta avrà luogo solo quando il buffer conterrà un telegramma. Se è stata impostata l'opzione di abbreviazione (vedi @Y) i dati saranno riportati in forma abbreviata.

Risposta:

<lunghezza> : sempre e solo un carattere (ascii o esadecimale puro) da '**0**' a '**F**' che indica la lunghezza logica dei dati rimanenti (non può essere 0)

<dati> : i dati del telegramma in esadecimale puro oppure in ascii a seconda del modo operativo

@c

Permette di uscire dallo stato di attesa innescato dal comando @R.

Risposta: <**k**>

@W<valore><dati>

Permette di trasmettere al dispositivo un telegramma da inviare in rete

<valore> : sempre e solo un carattere (ascii o esadecimale puro) da '**0**' a '**F**' che indica la lunghezza logica dei dati rimanenti

<dati> : i dati del telegramma in esadecimale puro oppure in ascii a seconda del modo operativo

Risposte possibili: <k> : tutto ok <E> : errore

@w<valore><dati>

Modalità rapida per mandare un telegramma di comando da inviare in rete

<valore> : sempre e solo un carattere (ascii o esadecimale puro) da '**0**' a '**F**' che rappresenta il comando da inviare al dispositivo ('0' accendi, '1' spegni)

<dati> : indirizzo dell'attuatore in esadecimale puro (1 byte) oppure in ascii (2 bytes) a seconda del modo operativo

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@Y0

Chiusura modalità abbreviata dei telegrammi. Dopo questo settaggio i telegrammi ricevuti con il comando @r oppure @R oppure @l verranno ritornati in modalità normale.

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@Y1

Settaggio modalità abbreviata dei telegrammi. Dopo questo settaggio i telegrammi ricevuti con il comando @r oppure @R oppure @l verranno ritornati in modalità abbreviata, corrispondenti ai bytes 2-3-4-5 del telegramma ricevuto (destinatario, mittente, tipo, comando). Praticamente il telegramma reale escluso il prefisso 0xA8, il checksum ed il suffisso 0xA3.

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@Y3

Settaggio modalità abbreviata dei telegrammi. Dopo questo settaggio i telegrammi ricevuti con il comando @r oppure @R oppure @l e i telegrammi scritti con il comando @W verranno trattati in modalità abbreviata, corrispondenti ai bytes 2-3-4-5 del telegramma (destinatario, mittente, tipo, comando). Praticamente il telegramma reale escluso il prefisso 0xA8, il checksum ed il suffisso 0xA3.

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@y<dati brevi>

Modalità abbreviata per mandare un telegramma di comando da inviare in rete – utilizzabile solo in modalità X (esadecimale) – indipendente dal settaggio @Y

<dati brevi> : 4 caratteri in formato esadecimale puro – corrispondenti ai bytes 2-3-4-5 del telegramma da inviare (destinatario, mittente, tipo, comando). Praticamente il telegramma reale escluso il prefisso 0xA8, il checksum ed il suffisso 0xA3.

Risposte possibili: <**k**> : tutto ok <**E**> : errore

I valore impostati con i comandi @M, @F, @D , @Y1, @Y3 vengono memorizzati nella eeprom del PIC e rimangono impostati anche spegnendo e riaccendendo il dispositivo.

@<0x15>

Dopo questo comando i settaggi successivi NON verranno più memorizzati in eeprom ma impostati in maniera temporanea, fino a che il dispositivo non viene resettato, oppure fino alla ricezione del comando **@<0x16>**. Inoltre in questa modalità il dispositivo eviterà anche la classica risposta di acknowledge “k”

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@t<valore><dati>

Modalità rapida per mandare un telegramma di interrogazione da inviare in rete

<dati> : indirizzo dell'attuatore in esadecimale puro (1 byte) oppure in ascii (2 bytes) a seconda del modo operativo

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@b

Pulisce tutti i buffers di ricezione.

Risposta : <**k**> : tutto ok

@h

Espone un menu di help

@q

Query version.

Risposta : <**k**> seguito dalla VERSIONE : tutto ok

@l (L minuscolo)

Attiva un log a video dei messaggi ricevuti e trasmessi

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@d

Dump di tutti i buffers di ricezione – solo in modalita ASCII.

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@D<indirizzo>

Permette di conoscere gli indirizzi dei dispositivi scoperti sul bus.

<indice di partenza scansione> : carattere esadecimale puro oppure due caratteri ascii che indicano da quale indice partire con la scansione. L'indirizzo reale si ottiene dal calcolo (<indice> * 2) – 1. (l'indirizzo di settore/linea viene ignorato, si presuppone uguale per tutto l'impianto e viene memorizzato all'indice 0)

Risposta: D<indice><tipo>

<indice > : carattere esadecimale puro oppure due caratteri ascii che indicano quale indice valido è stato trovato. L'indirizzo reale si ottiene dal calcolo (<indice> * 2) – 1.

<tipo > : carattere esadecimale puro oppure due caratteri ascii che indicano che tipo di dispositivo è stato trovato (1=switch 3=dimmer 8=tapparella 0xFF=nessuno)

@z

Elenca in formato testuale gli indirizzi dei dispositivi scoperti sul bus ed il relativo "tipo" (1=switch, 4=dimmer, 8=tapparella, 9=tapparella gestita a %)

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@s

Sbilancia l'indice dei buffers di ricezione simulando una ricezione di messaggio – solo in modalita ASCII-

Risposte possibili: <**k**> : tutto ok <**E**> : errore

@Ux

Comandi di settaggio e lista delle tapparelle in modalità percentuale. Il suffisso x può assumere i seguenti valori:

0 La modalità gestione a percentuale viene disattivata

1 Modalità di attesa setup automatico. Dopo questo comando ogni tapparella va chiusa, stoppata, aperta completamente, stoppata – per calcolare i tempi di salita.

2 Conclusione del setup automatico, i tempi calcolati vengono memorizzati nella eeprom del PIC e si entra in modalità 3.

3 La modalità a percentuale è attiva. Se il log @l è attivo, oltre ai telegrammi verranno notificate anche le posizioni intermedie raggiunte; in formato HEX rappresentate in 3 bytes nella forma "u<device><posizione>", in formato ASCII con righe con CRLF in testa seguito sempre da "u<device><posizione>" dove device e posizione sono espressi in esadecimale su 2 bytes ciascuno.

4 La modalità percentuale è attiva ma le posizioni intermedie non vengono notificate sul log.

5 Questo comando elenca in formato ascii lo stato delle tapparelle a percentuale, con l'indirizzo del dispositivo, la posizione massima, la posizione attuale, la direzione di movimento attuale, la posizione richiesta, il timeout attuale.

6 Elenco in formato HEX dello stato di una tapparella, il cui indirizzo va specificato subito dopo il "6". In formato ASCII invece questo comando può essere usato per impostare manualmente una nuova tapparella o cambiare il tempo di posizionamento. Dopo questo comando il PIC chiede l'indirizzo della tapparella, da digitare in 2 caratteri (00-7F) e quindi espone e richiede il valore (in decimi di secondo) per il tempo di salita. Dopo il tempo va digitato <invio> o <spazio>. Se non si digita nulla rimane impostato il tempo precedente.

7 Imposta il flag di pubblicazione immediata della posizione di tutte le tapparelle

8 nn

9 Azzera e ripulisce dalla eeprom la tabella tapparelle a percentuale

@u<indirizzo><percentuale>

Richiesta posizionamento tapparella gestita a percentuale

<indirizzo> : indirizzo scs della tapparella da comandare in esadecimale puro (1 byte) oppure in ascii (2 bytes) a seconda del modo operativo.

<percentuale> : percentuale di apertura richiesta da 0 a 100 (hex 00-64) in esadecimale puro (1 byte) oppure in ascii (2 bytes) a seconda del modo operativo

Risposte possibili: <**k**> : tutto ok <**E**> : errore

Comandi di settaggio

Il ricevitore dei segnali sul bus ha un valore di "sensibilità" che è preimpostato ad un valore base che dovrebbe essere sempre adeguato. Tuttavia in casi particolari può essere utili ritararlo, per abbassare la sensibilità qualora vengano loggati dei messaggi spuri, generati da disturbi sulla linea, o (più raramente) aumentata qualora non riceva i telegrammi. Il valore (Vref) può variare da 1 a 31; il valore corrente viene esposto dal comando @h dato in modalità ascii.

@i

Questo comando può essere dato quando sul bus non circolano assolutamente telegrammi. Il PIC legge la tensione sul bus ed imposta una sensibilità medio-alta adatta allo scopo. Il comando può essere dato solo in modalità ascii.

@I+ (i maiuscolo)

Questo aumenta la sensibilità di 1 punto. Il comando può essere dato solo in modalità ascii.

@I- (i maiuscolo)

Questo diminuisce la sensibilità di 1 punto. Il comando può essere dato solo in modalità ascii.

@Snnn<spazio>

Questo comando imposta un valore di tempo (timeout) impostato per riconoscere la fine di un telegramma. Il valore standard di timeout, visibile anche con il comando @h, è di 190, corrispondente a circa 264 microsecondi. Abbassando questo valore aumenta il timeout, diminuendo questo valore aumenta il timeout. Ogni punto di questo valore vale 4 microsecondi. La relazione è la seguente: timeout (uS) = (256 – valore) * 4

Il valore digitato deve essere terminato con uno spazio o con <enter>

@<0xEC>

0xEC = valore esadecimale (non ascii!)

Questo comando imposta la velocità di comunicazione sulla porta seriale al valore standard (115200 baud);

@<0xED>

0xED = valore esadecimale (non ascii!)

Questo comando imposta la velocità di comunicazione sulla porta seriale al valore di 38400 baud;

@<0xEE>

0xEE = valore esadecimale (non ascii!)

Questo comando imposta la velocità di comunicazione sulla porta seriale al valore minimo (19200 baud);

GESTIONE TAPPARELLE A PERCENTUALE

Va chiarito che la gestione della percentuale di apertura è gestita a tempo; significa che per esempio una tapparella è aperta al 50% quando è trascorsa la metà del tempo necessario all'apertura completa (ma probabilmente a questo punto la tapparella sarà a meno della metà di apertura a causa del tempo impiegato ad aprire le righe). Inoltre, probabilmente la discesa sarà leggermente più veloce e quindi la percentuale di apertura potrà differire. Ad ogni chiusura completa comunque la percentuale va a zero ed il calcolo riparte da capo.

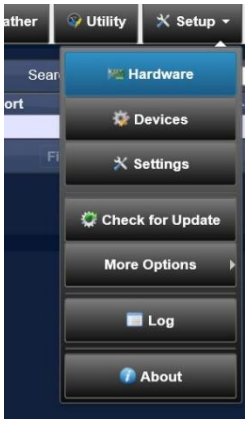
Per conoscere e gestire la percentuale di apertura sono quindi necessarie delle operazioni preliminari. Tali operazioni, pur essendo descritte nel paragrafo "MQTT" al capitolo "Pubblicazione censimento dispositivi", possono essere eseguite anche senza una connessione mqtt, limitandosi a memorizzare i dati delle tapparelle a percentuale.

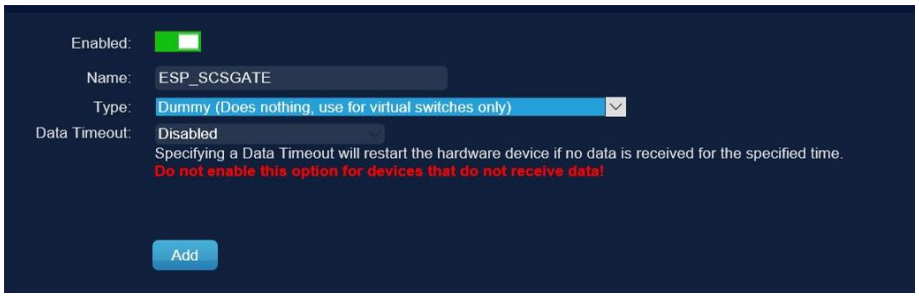
DOMOTICZ - HTTP

Sappiate che non sono affatto un esperto di domoticz – la mini guida che segue serva a mostrarvi i passi che ho fatto e come l'ho integrato.

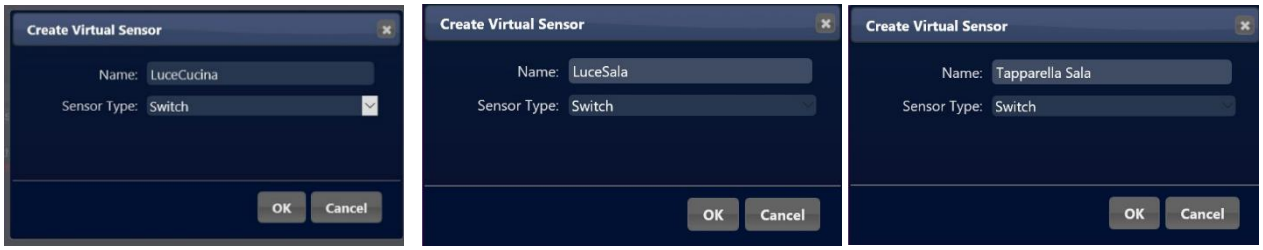
Prima ancora di cimentarvi con domoticz è necessario "sniffare" l'impianto, magari usando come già detto il mio programmino VB6, per scoprire tutti gli indirizzi e i comandi associati ai vostri dispositivi SCS.

Primo passo: in domoticz ho definito un hardware virtuale (di tipo Dummy) e l'ho chiamato ESP_SCSGATE

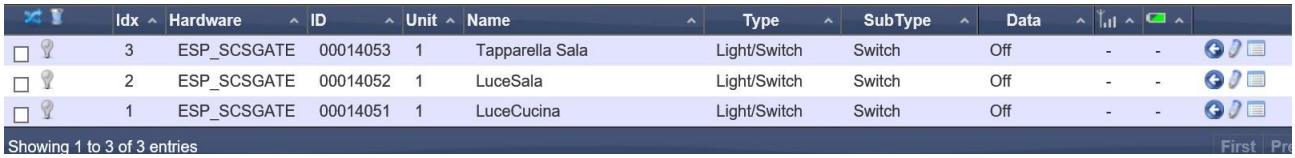




Associati a questo hardware ho definito dei "sensori virtuali" – uno per ciascun attuatore SCS che voglio controllare

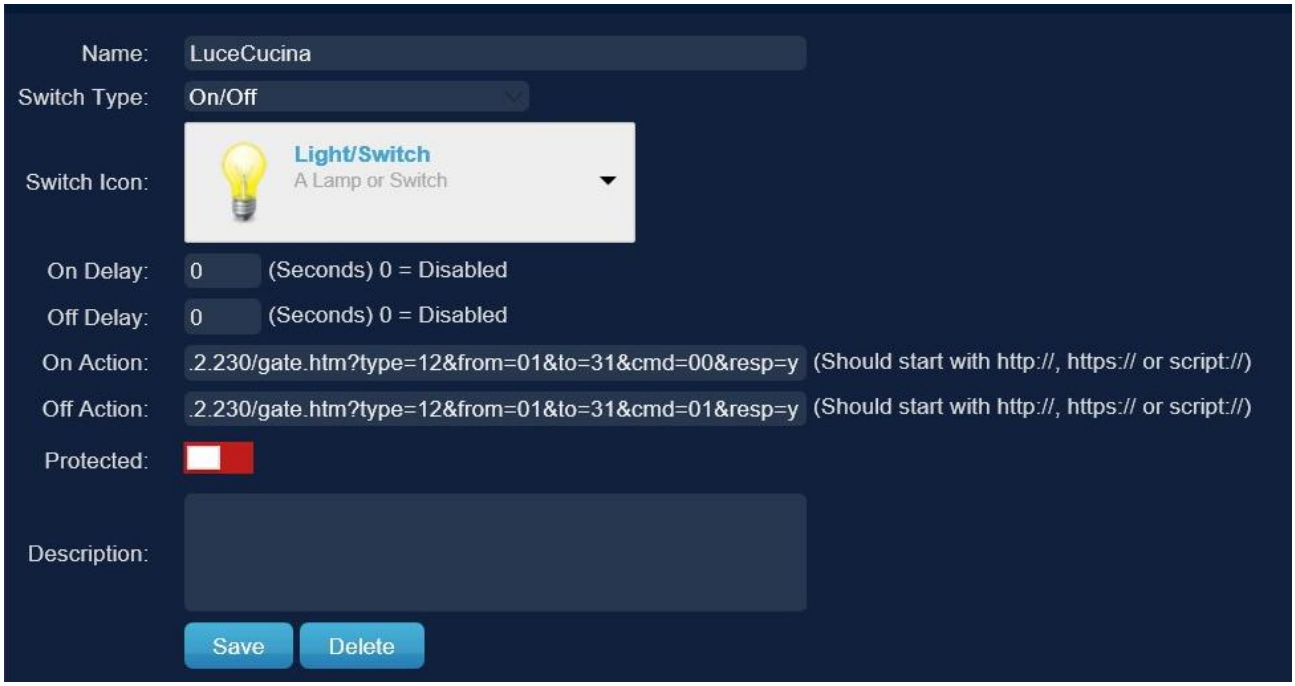


Li ho definiti di tipo "Switch" – per curiosità ho provato anche "Selector switch" che però non mi sembra adatto al controllo di un normale attuatore. Quindi la lista dei dispositivi (Setup – devices) era questa:



Cliccando sul bottone "switches" appare poi così:

Cliccando sul bottone “Edit” si impostano le proprietà dello switch:



Sulla casella “On action” (accendi) ho scritto il **comando HTTP** che richiama il mio RASPY_SCSGATE che è all’indirizzo 192.168.2.230, con i bytes di comando che ho scoperto sniffando A8 31 01 12 00 xx A3

http://192.168.2.230/gate.htm?type=12&from=01&to=31&cmd=00&resp=y

stessa cosa per “Off action” (spegni)

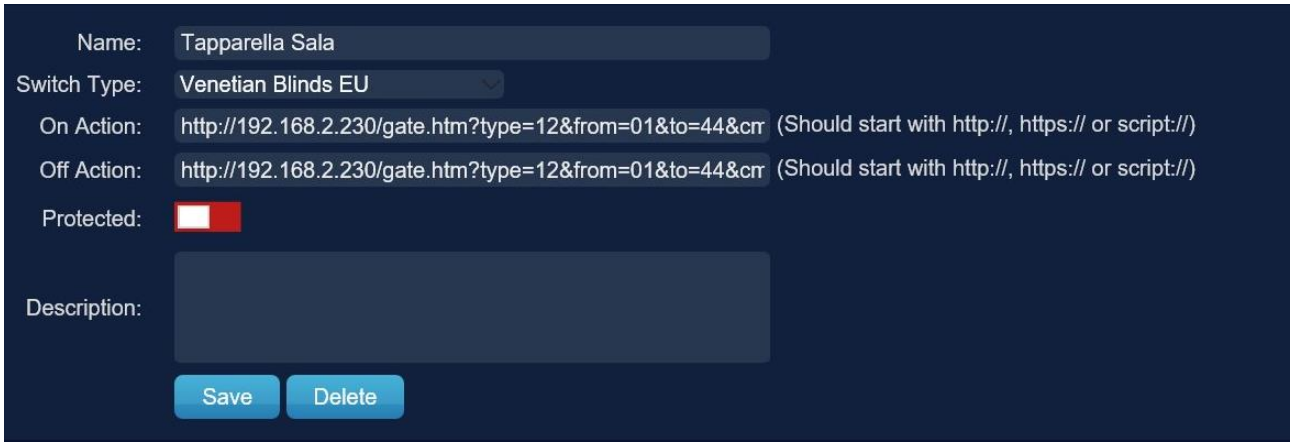
http://192.168.2.230/gate.htm?type=12&from=01&to=31&cmd=01&resp=y

n.b. Se raspy_scsgate e domoticz sono installati sul medesimo raspberry, l’indirizzo IP con cui si richiameranno a vicenda sarà sempre 127.0.0.1

L’esempio di cui sopra è valido se SCSTCP è stato fatto partire sulla porta 80 – diversamente va aggiunto il numero di porta (es. http://192.168.2.230:5045/gate.htm...)

i “selector switches” consentono ulteriori comandi personalizzabili e associabili. Limite attuale: ad ogni bottone è possibile associare un solo telegramma.

Ho salvato ed ho fatto la stessa cosa per gli altri switches. Per la tapparella ho scelto come “switch-type” : Venetian blinds EU – ciò imposta l’icona della tapparella e i tre bottoni alza/stop/abbassa. Peccato che poi si possono inserire le stringhe di comando solo per on/off:



Sulla videata degli “switches” cliccate sulla stellina di ciascuno così che diventi gialla: questi switches verranno così presentati sulla videata principale (la dashboard).

Sulla dashboard, cliccando sulla lampadina dello switch, essa si accenderà/spegnerà e contestualmente domoticz lancerà la richiesta HTTP verso raspy_scsgate.

Questa era la parte facile. Un po’ più complicato è il viceversa, cioè far accendere o spegnere la lampadina sulla dashboard quando la lampadina viene accesa da un interruttore fisico. Qui entra in ballo la procedura di callback che viene richiesta dal parametro di chiamata &resp=y. Se ne deduce che perché venga richiamata è indispensabile fare almeno una chiamata di switch (ne basta una sola, su di uno switch qualunque).

Prima di tutto bisogna indicare a raspy_scsgate l’indirizzo ip e parametri della callback. Va quindi inserita l’opportuna direttiva nel file scsconfig (generalmente nella seconda riga). Indirizzo ip e porta saranno quelli relativi a domoticz.

```
{"httpcallback":"json.htm?type=command&param=udevices&script=scsgate_json.lua","
httpip":"192.168.0.111","httpport":"8080"}
```

n.b. Se raspy_scsgate e domoticz sono installati sul medesimo raspberry, l’indirizzo IP con cui si richiameranno a vicenda sarà sempre 127.0.0.1

Il parametro serve a impostare la pagina che verrà richiamata dal gate per il callback, quando intercetta telegrammi che viaggiano sul bus.

Al momento della chiamata di callback (di tipo GET) verranno aggiunti 4 parametri che riportato i dati del dispositivo che ha cambiato stato: “&type=xx&from=xx&to=xx&cmd=xx

Come avete visto, viene richiamato lo script scsgate_json.lua che va personalizzato secondo le vostre esigenze e messo nella directory di domoticz denominata “**scripts/lua parsers**” . Quello che vi mostro è un esempio, quello che

ho usato io nei test. Serve a convertire l'indirizzo SCS nel numero di device di domoticz (idx), e a convertire il byte di comando in una azione da effettuarsi sul device virtuale. Tenete conto che nei "comandi" scs (type=12) l'indirizzo del device interessato è quello di destinazione, nei telegrammi di stato invece l'indirizzo del device interessato è quello di provenienza.

Ecco lo script "demo" che io ho usato. Contiene alcune istruzioni "print" di per se inutili che possono servire in fase di debug, poiché vengono mostrate in domoticz con il bottone setup – log.

```
-- Example of JSON parser handling GET data with the following structure

--
http://192.168.1.17:8080/json.htm?type=command&param=udevices&script=scsgate_json.lua&type=xx&from=x
x&to=xx&cmd=xx

-- retrieve the GET params
local type = uri['type'];
local from = uri['from'];
local to   = uri['to'];
local cmd  = uri['cmd'];


    print ("type="..type.."", from="..from.."", to="..to.."", cmd="..cmd.." ")


-- UpdateDevice']='idx|nValue|sValue'
--                 idx= id device
--                 nValue=
--                 sValue=

-- 20
-- domoticz_updateDevice(1,0,'Off')
local device = '00';
local idx;
local nCmd;
local sCmd;

if ((type == '12') or (type == '15')) then     -- 27
    if ((to > '01') and (to < '80')) then
        device = to
    else
        device = from
    end
end

if     (device == '31') then
    idx = 1
elseif (device == '32') then
    idx = 2
elseif (device == '33') then
    idx = 3
elseif (device == '34') then
    idx = 4
else
    idx = 0
end

if (cmd == '00') then
        nCmd = 1
        sCmd = "On"
elseif (cmd == '01') then
        nCmd = 0
        sCmd = "Off"
else
        nCmd = 9
        sCmd = 9
end

 print ("idx="..idx.."", nCmd="..nCmd.."", sCmd="..sCmd.." ")
```

```
 if ((idx > 0) and (nCmd < 9)) then
      domoticz_updateDevice(idx,nCmd,sCmd)
 end

-- Retrieve the request content
--s = request['content'];

-- Update some devices (index are here for this example)
--local id = domoticz_applyJsonPath(s,'.id')
--local s = domoticz_applyJsonPath(s,'.temperature')
--domoticz_updateDevice(id,'',s)
```

Se avete qualche monitor wifi (io uso wireshark) potete verificare i colloqui. Ho potuto constatare che esp_scsgate è abbastanza reattivo e che invece domoticz non è particolarmente veloce (forse perche l'ho installato su windows).

DOMOTICZ - MQTT

MQTT si sta sempre più diffondendo come standard di comunicazione tra dispositivi domotici e relativi sistemi di controllo. E' basato su di un server centrale, detto "broker" a cui vengono inviati ("pubblicati") i messaggi di controllo e di stato, distinti per argomento ("topic") e che si occupa di distribuirli ai "client" che hanno "sottoscritto" i corrispondenti argomenti.

Nel nostro caso il sistema di controllo testato è stato ancora una volta DOMOTICZ, che a sua volta è stato definito "client" del broker MQTT attraverso un apposito PLUGIN (https://github.com/emontnemery/domoticz_mqtt_discovery) che peraltro è dichiarato compatibile con HOME ASSISTANT.

Tale plugin funzionava bene con le luci e male con dimmer e tapparelle per cui ho provveduto a correggerlo in casa, sto proponendo le mie correzioni all'autore.

La versione da me corretta la trovate qui: https://github.com/papergion/domoticz_mqtt_discovery

Come broker MQTT ho usato MOSQUITTO perché semplice, leggero e disponibile anche in windows (test effettuati solo in windows).

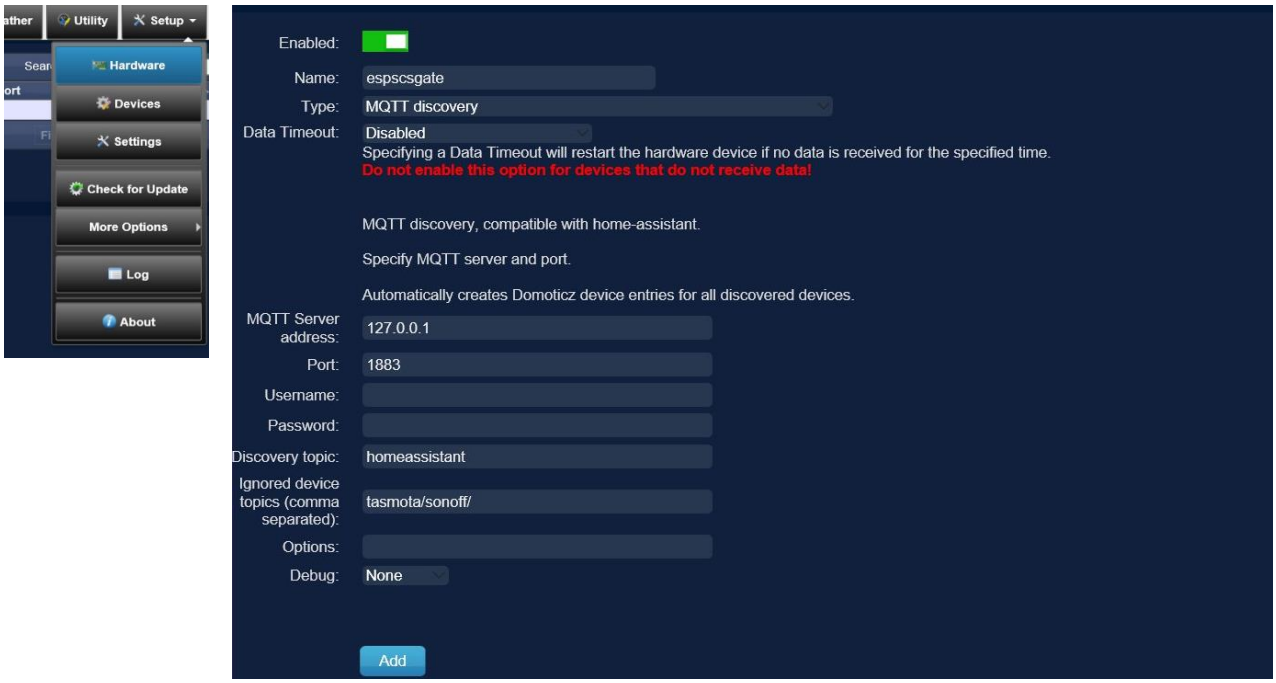
Anche ESP_SCSGATE va configurato per la connessione al medesimo broker – accertatevi sul monitor/log del broker che la connessione venga correttamente accettata.

Come già detto non sono affatto un esperto di domoticz – la mini guida che segue serva a mostrarvi i passi che ho fatto e come l'ho integrato.

Primo passo: localizzate la directory di installazione di domoticz (in windows è C:\Program Files (x86)\Domoticz) – qui dovrebbe esserci una cartella "plugins" – se non c'è createla. Dentro la cartella plugins inserite la cartella "mqtt_discovery-development" contenente il plugin "plugin.py". E' indispensabile avere sul computer anche "python" – io ho dovuto usate la v.3.5.2 perché con la più recente 3.7 in windows non funzionava neppure domoticz.

Dopo aver copiato il plugin, domoticz va spento e riacceso – controllate sul log di non avere errori.

Se tutto è corretto, andando a censire nuovo hardware dovreste trovare anche l'hardware "MQTT Discovery"



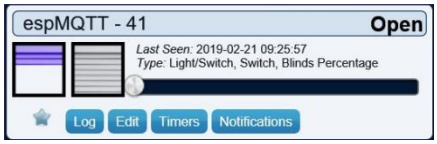
Va digitato il nome (io ho usato espscsgate) e indirizzo e porta del broker. Cliccate su ADD.

I dispositivi (SWITCHES) non possono essere aggiunti manualmente, è indispensabile procedere al censimento automatico:

Connettete raspy_scsgate al bus.

Seguendo le istruzioni iniziali, preparate il file “scsconfig” e poi lanciate il programma “scsdiscovery”

A questo punto refreshate la pagina di domoticz “SWITCHES” e ci troverete i vostri dispositivi, censiti con dei nomi standard:



Cliccando sulla stellina essi appariranno anche sulla DASHBOARD di domoticz. Potete provarli cliccando sulla lampadina, variano lo slide dei dimmer o delle tapparelle, premendo le icone delle tapparelle.

Dopo aver così individuato la corrispondenza con i vostri dispositivi, per ciascuno potete entrare in EDIT e cambiare opportunamente il nome.

Rifacendo da capo il censimento i nuovi nomi verranno mantenuti e non ci saranno duplicazioni.

HOME ASSISTANT - MQTT

Ho installato home-assistant inizialmente su windows 10 per comodità di test, consapevole del fatto che in una installazione windows non tutte le funzionalità vengono garantite, successivamente l'ho installato su raspberry.

Preciso che la versione homeassistant era 0.85.1 – lo dico perché ho riscontrato nei fatti che homeassistant è un bellissimo progetto, ma non spicca per stabilità. Consiglio di non cambiare senza motivo la versione che avete installato perché potreste avere delle sorprese. Recentemente sono passato alla versione 0.90.1 e poi alla 0.114.0: tutto continua a funzionare.

Come broker MQTT ho usato MOSQUITTO perché semplice, leggero e disponibile anche in windows.

Primo passo: localizzate la directory contenente il file di configurazione "configuration.yaml" – editate tale file.

Aggiungete le definizioni per mqtt:

```
mqtt:
  broker: 127.0.0.1
  port: 1883
  discovery: true
  discovery_prefix: homeassistant
```

Ovviamente usate ip address e port del vostro broket mqtt.

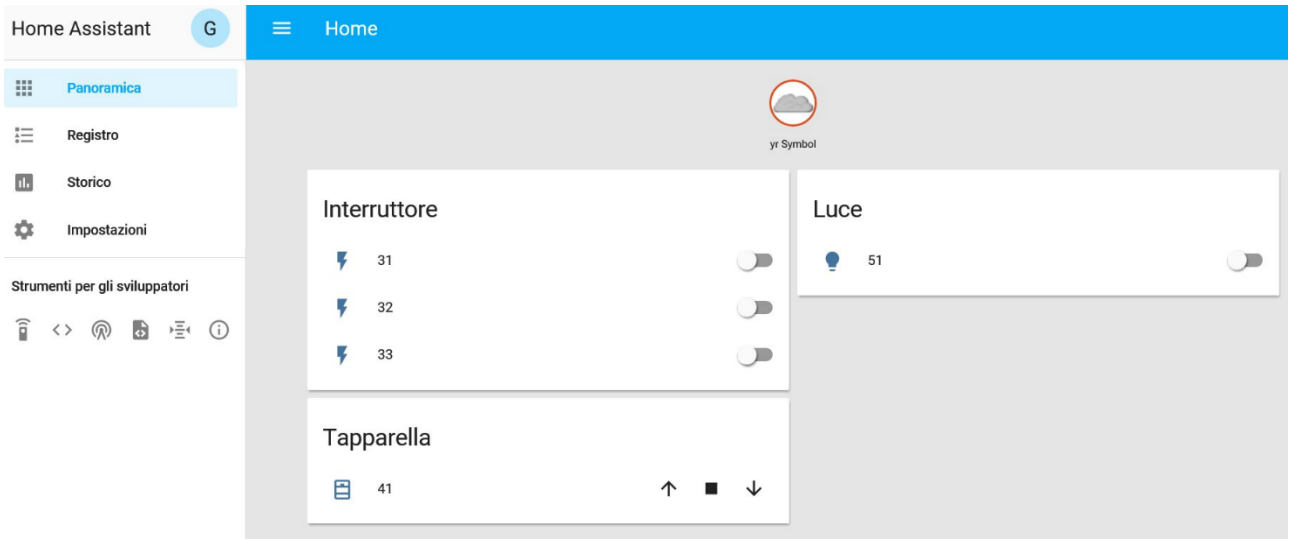
Dopo aver modificato il file, home assistant va spento e riacceso – controllate sul log di non avere errori.

Se tutto è corretto, andando nella pagina "Impostazoni -> integrazioni" dovreste vedere MQTT come configurato:

Connettete raspy_scsgate al bus.

Seguendo le istruzioni iniziali, preparate il file "scsconfig" e poi lanciate il programma "scsdiscovery"

A questo punto refreshate la pagina iniziale (“panoramica”) di home assistant e ci troverete i vostri dispositivi, censiti con dei nomi standard, corrispondenti agli indirizzi SCS:



Cliccando sulle icone potrete verificarne la funzionalità.

Secondo le indicazioni dell’help, mqtt discovery avrebbe dovuto aggiungere automaticamente le medesime definizioni anche sul file “configuration.yaml”. Non è stato così: forse non ho fatto qualche settaggio, forse la versione windows non funziona bene, o forse ho capito male io.

Invece ciò non avviene e quindi spegnendo e riaccendendo homeassistant le definizioni scompaiono: per renderle permanenti bisogna invece modificare manualmente configuration.yaml, approfittandone per mettere nomi effettivi:

```
mqtt:
  broker: 127.0.0.1     (se sulla stessa macchina, scrivere  localhost )
  port: 1883
  discovery: true
  discovery_prefix: homeassistant

switch:
  - platform: mqtt
    name: Luce cucina (31)
    command_topic: "scs/switch/set/31"
    state_topic: "scs/switch/state/31"

  - platform: mqtt
    name: Luce sala (32)
    command_topic: "scs/switch/set/32"
    state_topic: "scs/switch/state/32"

  - platform: mqtt
    name: Luce bagno (33)
    command_topic: "scs/switch/set/33"
    state_topic: "scs/switch/state/33"

cover:
```

(tapparella gestita up/down/stop)

```
  - platform: mqtt
    name: Sala grande (41)
    command_topic: "scs/cover/set/41"
```

```
    state_topic: "scs/cover/state/41"
```

(tapparella gestita a percentuale)

```
  - platform: mqtt
    name: Sala piccola (42)
    command_topic: "scs/cover/set/42"
    state_topic: "scs/cover/state/42"
    set_position_topic: "scs/cover/setposition/42"
    position_topic: "scs/cover/value/42"

light:
  - platform: mqtt
    name: Dimmer Sala (51)
    command_topic: "scs/switch/set/51"
    state_topic: "scs/switch/state/51"
    brightness_command_topic: "scs/switch/setlevel/51"
    brightness_state_topic: "scs/switch/value/51"
```

SE conoscete già gli indirizzi SCS dei vostri dispositivi potete tranquillamente evitare tutto il giro di “discover” e censirli direttamente nel file “configuration.yaml”

HOME ASSISTANT (HASSIO) - MQTT

L'installazione su home-assistant basato su Hassio presenta qualche attenzione in più riguardante soprattutto il broker Mosquitto che se non installato correttamente può pregiudicare il corretto funzionamento.

Per la corretta installazione (o reinstallazione) di Mosquitto in ambiente Hassio fate riferimento a questa guida: https://indomus.it/guide/configurare-correttamente-mqtt-su-hassio-versione-addon-dalla-v3-in-poi/

È molto particolareggiata e seguendola attentamente riuscirete ad installarlo correttamente.

A questo punto vale tutto quanto detto per l'installazione su home-assistant "raspbian".

Dovrete censirli manualmente in configuration.yaml (attenzione, SOLO i dispositivi, NON il server mqtt che viene invece gestito da hassio).

Altra attenzione: usando questa procedura il broker mosquitto verrà definito accessibile solo con user e password, che andranno quindi usati coma parametri di scsdiscover e scsgate_x.

OPENHAB

Non conosco openHAB, gli esempi che seguono mi sono stati mandati da Omar, che ringrazio:

file “broker.things”:

```
mqtt:broker:RpiBroker " Comandi bticino" [ host=" IP_SERVER_MQTT ",
secure=false, username="USER", password="PASSOWORD" ]
```

file “interruttori.items”

```
Switch          GF_LivingDining_Light         "Luce Sala"
<light>           (GF_LivingDining, gLight)         ["Lighting",
"Switchable"]   {channel="mqtt:topic:RpiBroker:interruttori:lamp1",
alexa="PowerController.powerState"}

Rollershutter   GF_LivingDining_Shutter       "Tapparella Sala"
<rollershutter>   (GF_LivingDining, gShutter)       ["Rollershutter"]
{channel="mqtt:topic:RpiBroker:tapparelle:roller1", alexa="Blind" [
actionMappings="Close=ON,Open=OFF,Lower=ON,Raise=OFF",
stateMappings="Closed=100,Open=0"]}
```

file “mqtt.things”

```
Bridge mqtt:broker:RpiBroker "Comandi bticino" [ host="IP_SERVER_MQTT",
secure=false, user="USER", password="PASSWORD" ]
{
    Thing topic interruttori "Interruttori bticino " {
    Channels:
    Type switch : lamp1 "Luce Sala" [ stateTopic="scs/switch/state/25",
commandTopic="scs/switch/set/25", on="ON", off="OFF" ]
    ........ Altri interruttori
    }
    Thing topic tapparelle "Tapparelle bticino " {
    Channels:
        Type rollershutter : roller1 "Tapparella Sala" [
stateTopic="scs/cover/value/14", commandTopic="scs/cover/setposition/14",
transformationPattern="JS:invertpercent.js",
transformationPatternOut="JS:invertpercent.js" ]
     .......... Altre tapparelle
    }
}
```

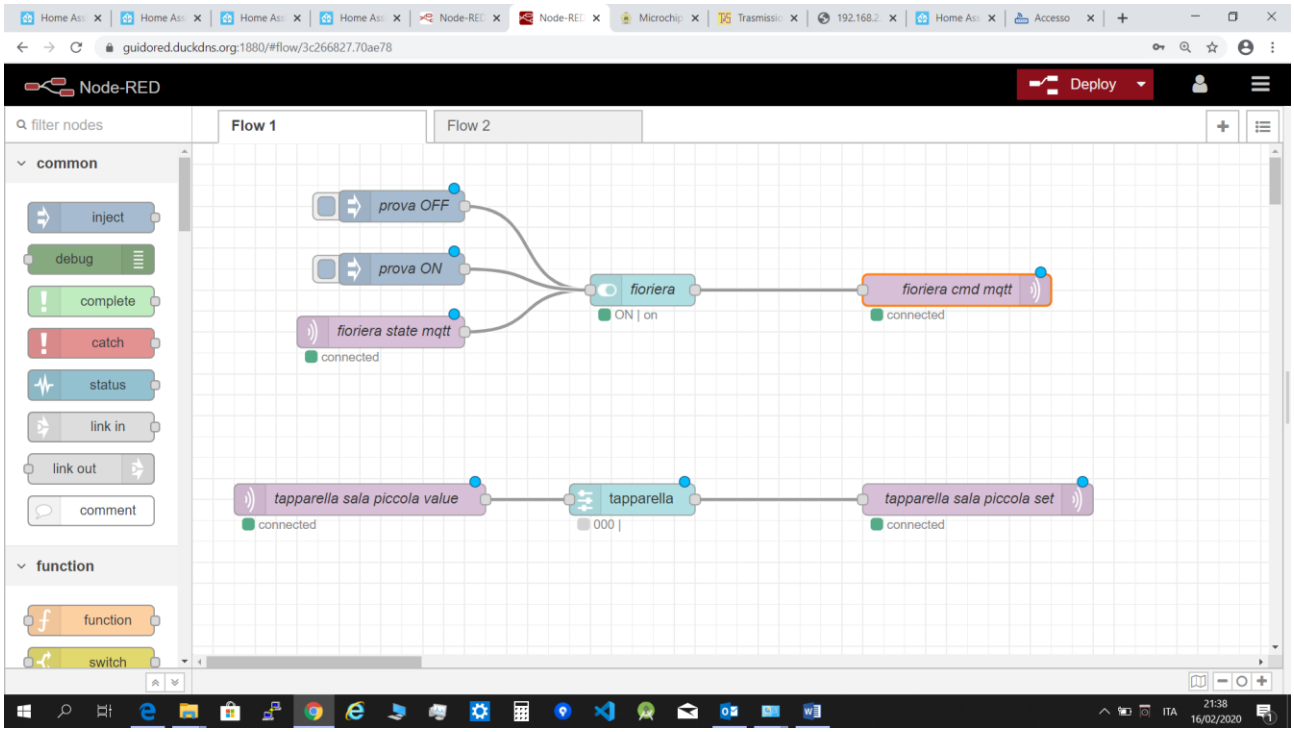
File “invertpercent.js”: è una funzione indispensabile per “rovesciare” la percentuale di apertura delle tapparelle, che in openhab viene gestita al contrario (percentuale di chiusura).

```
(function(i)  {
   var percent_shelly = parseInt(i,10);
   var percent_oh = (100.0- percent_shelly);
   return percent_oh.toFixed(0);
})(input)
```

NODE-RED

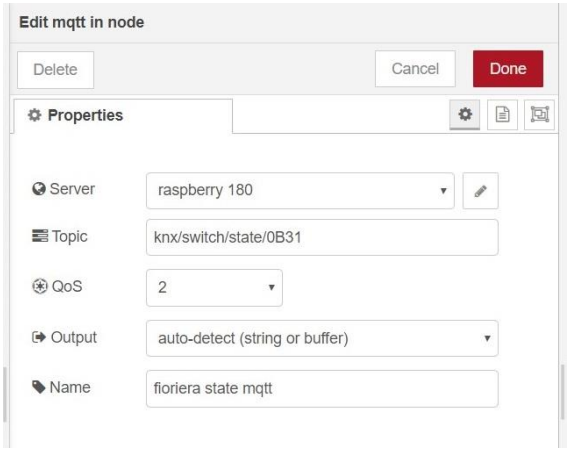
Node-red è un altro sistema domotico diffuso e apprezzato che consente una facile interazione attraverso un broker mqtt.

Ecco un esempio di definizione di uno switch luce e di una tapparella gestita a percentuale (ovviamente i modi di definizione possono essere molteplici):

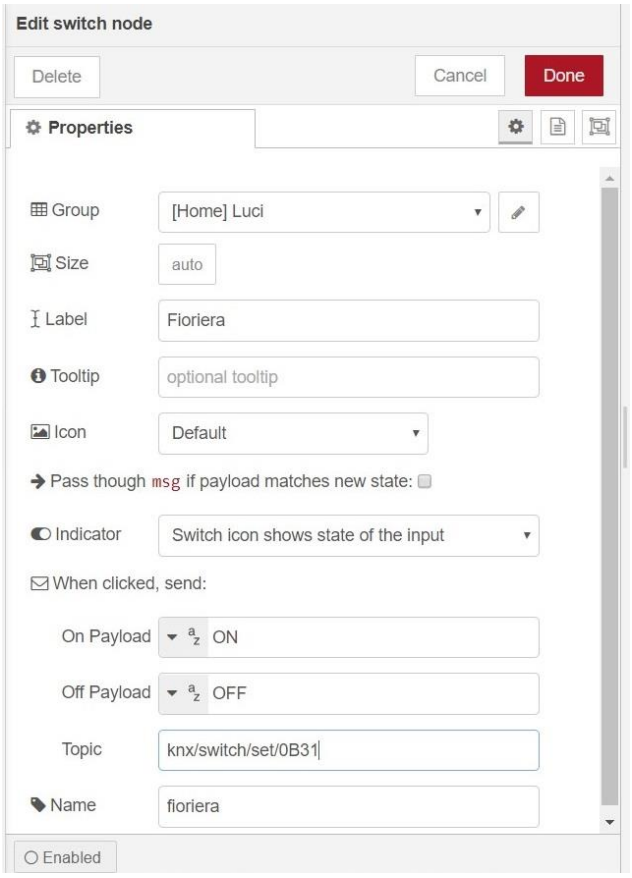


I blocchi “inject” servono solo come test.

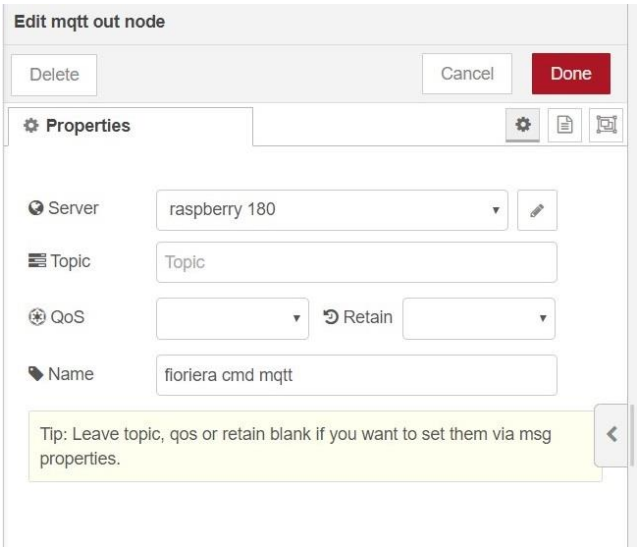
Il blocco che legge lo stato della luce (fioriera state mqtt) è un blocco “mqtt-in”:



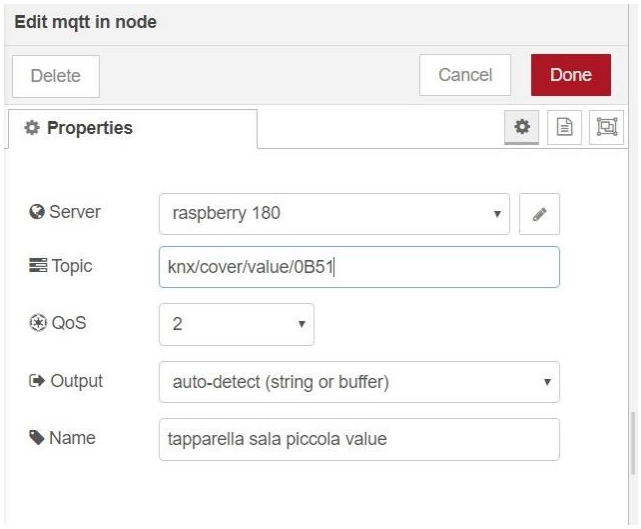
Il blocco luce (fioriera) è un blocco “switch”:



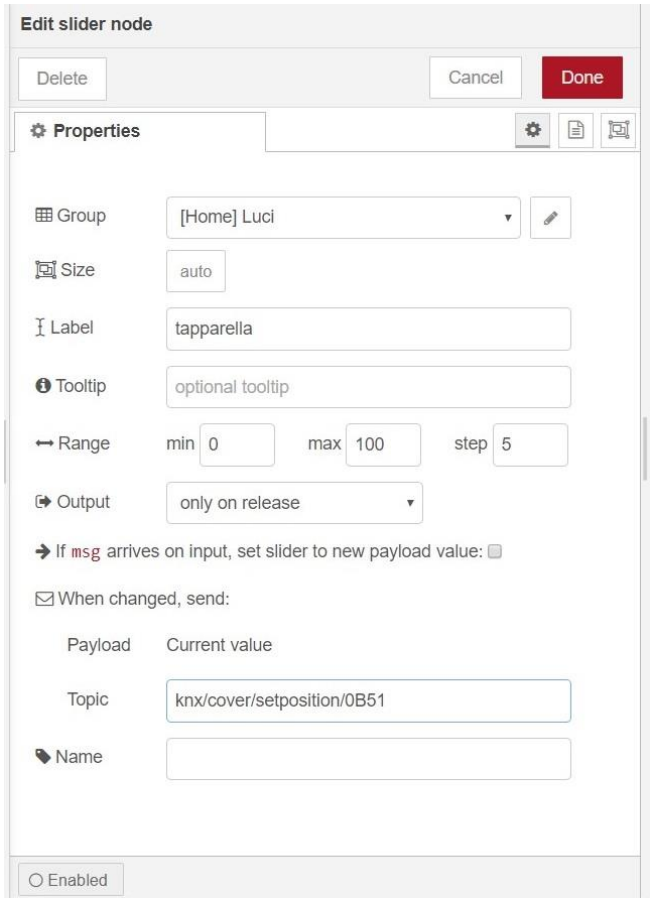
Il blocco di comando è un “mqtt-out”:



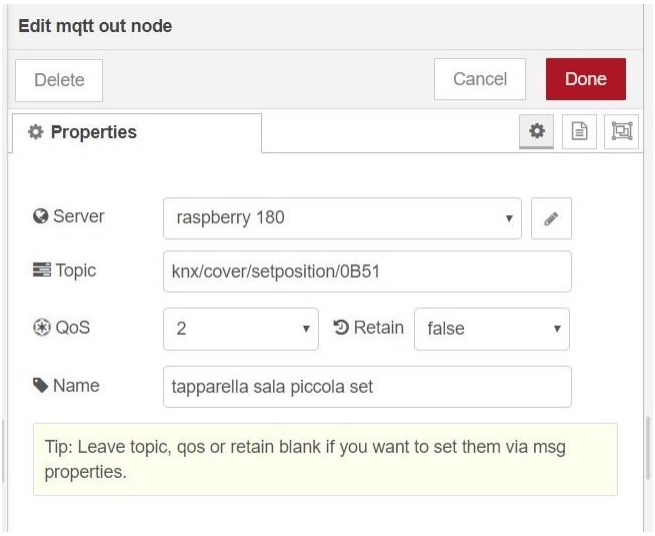
Il blocco che legge lo stato della tapparella è un blocco “mqtt-in”:



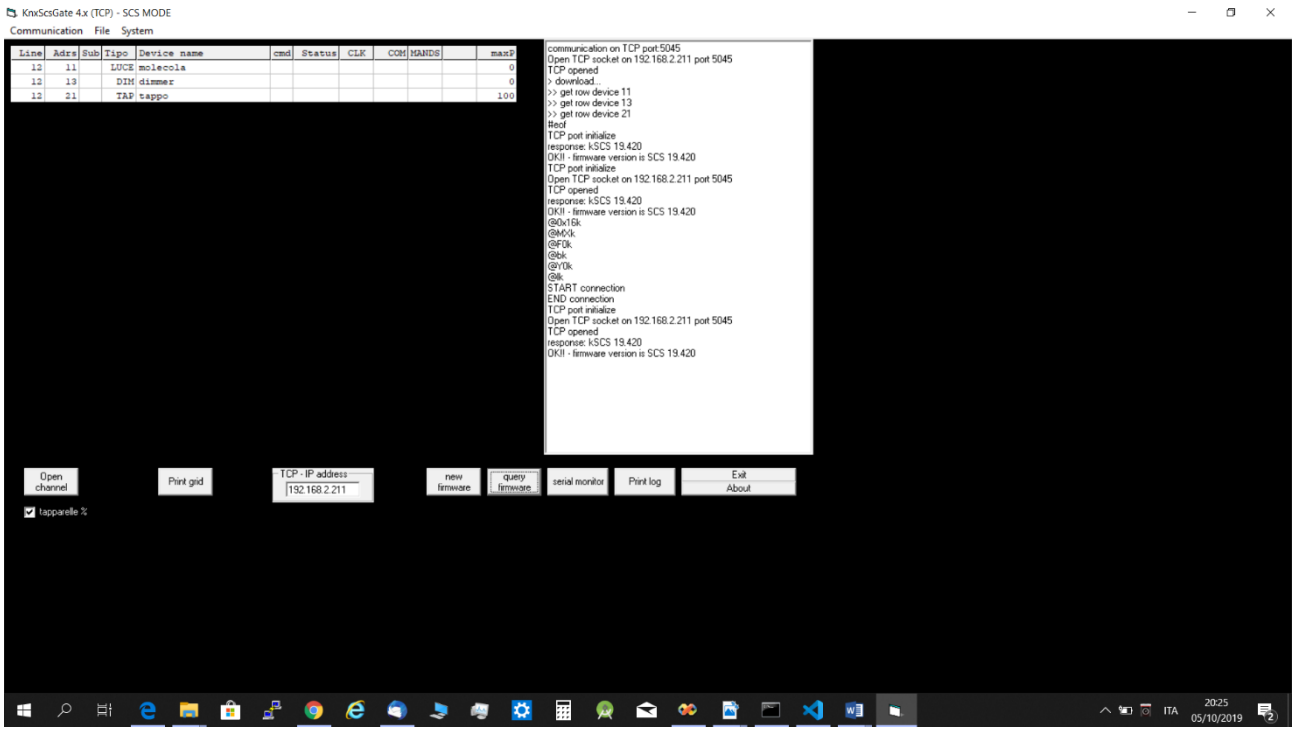
Il blocco tapparella è un blocco “slider”:



Il blocco di comando è un “mqtt-out”:



KNXSCSGATE – VERSIONE TCP



E' una nuova versione del precedente programma che lavora in TCP anziché UDP. I vantaggi sono i seguenti:

La funzione di "new firmware" è più sicura a causa della natura del protocollo tcp

Nel menu sono state aggiunte le funzioni di "download" e "upload" che consentono di scaricare, modificare, ricaricare nell'esp8266 la tabella dei dispositivi. Quindi particolarmente utile se si usano le tapparelle a % (settaggio dei tempi) o l'interfaccia diretta per ALEXA (settaggio dei nomi). Ma comunque utile per avere a disposizione un riepilogo di dispositivi ed indirizzi.

In congiunzione con "Open channel" consente di censire e aggiungere rapidamente nuovi dispositivi ed effettuarne l'upload. Una alternativa comoda e rapida alle funzioni …/mqttdevices:

Menu "communication" scegliere TCP

Digitare nella apposita casella l'IP address del dispositivo

Cliccare su "query firmware" una o più volte – la casella dell'ipaddress diventerà verde (comunicazione tcp effettuata) ed il responso dovrà essere "OK – firmware version: xxxx"

Open channel: si apre un canale di comunicazione tra il bus domotico e l'interfaccia

Accendere/spegnere tutti i vari dispositivi: appariranno nella griglia

Man mano aggiungere a mano i nomi così da identificarli correttamente

Se si vogliono gestire tapparelle a percentuale spuntare la casellina in basso a sinistra ed inserire nella colonna maxP i tempi di salita (in decimi di secondo)

Al termine USCIRE dalla modalità MONITOR.

Nel menu “File” scegliere “UPLOAD data”

Salvare anche in un file locale: dal menu “file” scegliere “Save data”

I dispositivi possono essere aggiunti (quando non correttamente intercettati) o modificati o cancellati:

Per aggiungere un dispositivo effettuare dapprima il download dei dispositivi precedentemente caricati sull’ESP – il download aggiungerà delle righe vuote su cui potranno essere scritti.

Obbligatorio inserire il codice di linea/settore (line), l’indirizzo (Adrs), il Tipo può valere LUCE, DIM, TAP, GEN (luce,dimmer,tapparella, generico). Facoltativa la descrizione e il tempo di salita per le tapparelle.

Per cancellare una casella premere <esc>, per correggere un carattere cancellarlo con il tasto <indietro>, per eliminare un dispositivo cancellate la casella Adrs o Tipo di quel dispositivo

Nel menu è stata aggiunta la possibilità di settare la frequenza di lavoro dell’esp8266 a 80 o a 160 mhz

ATTENZIONE: se usate questo sistema per correggere i dispositivi e poi volete usare il censimento automatico (discovery) in homeassistant o domoticz, usate la funzione …/mqttdevices?request=resend (se usaste prepare e send ri-cancellereste tutto da capo)

DISCLAIMER

Non posso garantire che l’integrazione con Domoticz, con Home-assistant e con Alexa sia perfetta in quanto il test si è limitato alle prove suddette, né posso garantire che le prossime versioni di Domoticz e Home assistant e Alexa (e Philips hue) mantengano inalterata questo tipo di interfaccia.

Non mi ritengo responsabile di danni che potreste causare al vostro impianto domotico per imperizia o negligenza o per non aver adottato le indispensabili precauzioni di sicurezza.