Abbiniamo l'elettrodo per il riconoscimento di gesture 3D a Raspberry Pi, per realizzare un'applicazione in cui con i gesti facciamo scorrere su uno schermo HDMI delle immagini.

# GestIC: PROVIAMOLA CON RASPBERRY PI

di MATTEO DESTRO

Nel fascicolo di giugno scorso abbiamo descritto un nuovo elettrodo GestIC pensato e realizzato per l'interfacciamento con le board Arduino Uno Rev. 3, Arduino Leonardo Rev. 3 e Raspberry Pi B+/2.0, e vi abbiamo proposto un'applicazione con Arduino, promettendovi di proporre a breve l'abbinamento a Raspberry PI; ebbene, è giunto il momento di vedere anche l'interfacciamento con questa scheda e analizzare l'implementazione della libreria necessaria a gestire l'integrato MGC3130. La libreria è

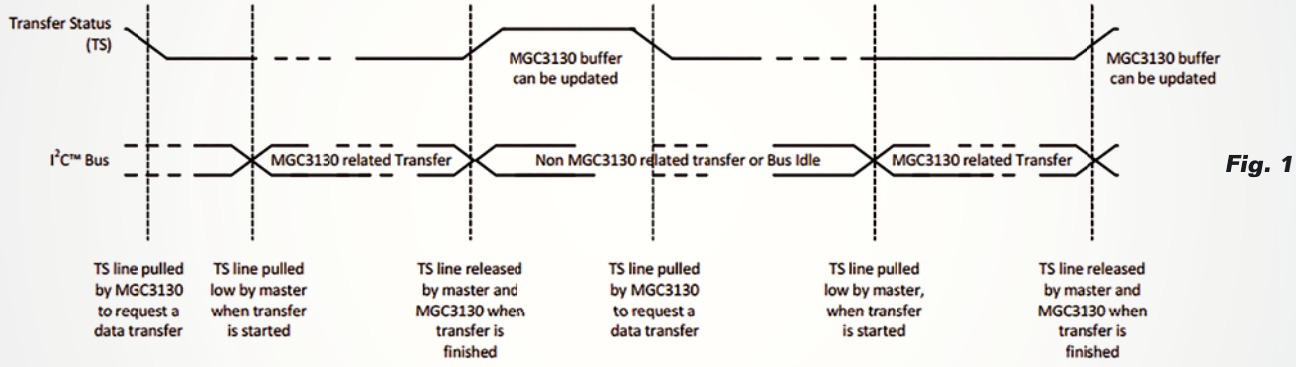



*Fig. 1*

realizzata in Python, ovvero un linguaggio di programmazione dinamico orientato agli oggetti utilizzabile per lo sviluppo software di qualsiasi tipologia. Inoltre permette lo sviluppo di codice di qualità e di facile manutenibilità, offre un forte supporto all'integrazione con altri linguaggi e programmi, è fornito di un'estesa libreria standard e può essere appreso in pochi giorni nelle sue funzioni di base.

## BREVE PANORAMICA SULL'HARDWARE

Abbiamo già affrontato nella precedente puntata la descrizione dello schema elettrico del nostro nuovo elettrodo, quindi non ci ripetiamo (rinviandovi per gli approfondimenti del caso al fascicolo di giugno) e ci soffermeremo solo su alcuni brevi punti senza andare nei dettagli.
Come si può osservare dallo schema elettrico, della scheda Raspberry Pi vengono usati pochi pin tra cui i segnali SDA e SCL collegati direttamente all'integrato MGC3130, il quale lavora con una logica che ha lo stesso livello di tensione di quello della Raspberry Pi e quindi non necessita di adattatori di livello, diversamente da quanto invece è richiesto da Arduino. Stesso discorso per le linee di I/O, che vengono portate direttamente all'integrato e in particolare le ormai note linee TS e RESET. Brevemente ricordiamo che la linea TS serve per indicare alla Raspberry Pi che sono pronti dei dati nel buffer di MGC3130 come conseguenza del riconoscimento di una gesture. Non appena Raspberry Pi si "*accorge*" che la linea TS è andata a livello logico basso, provvede ad occupare tale linea configurandola come uscita e portando anch'esso la linea a livello logico basso; a questo punto può iniziare la lettura dei dati presenti nel buffer dell'MGC3130 sfruttando l'interfaccia di comunicazione I²C. Conclusa la lettura dei dati, la linea viene riportata a livello logico alto e riconfigurata come ingresso in attesa di un prossimo evento. La **Fig. 1** mette in evidenza l'andamento della linea TS rispetto al bus I²C.

La linea RESET, come suggerisce il nome, serve per resettare l'integrato MGC3130 in caso di necessità. Solitamente si tende a mantenere la linea di RESET a livello logico basso durante il power-up dell'integrato e a rilasciare tale linea solo ad alimentazione stabilizzata (linea RESET riportata a livello logico alto). Infine alla scheda Raspberry Pi sono stati collegati tre pulsanti P1, P2 e P3 più un jumper J1. Solitamente per questa tipologia di componenti si prevede sempre una resistenza di pull-up da collegare sull'ingresso del microcontrollore che ne legge lo stato. Nel nostro caso si è preferito sfruttare la resistenza di pull-up interna messa a disposizione dal SOC (System On chip) presente su Raspberry Pi (Broadcom BCM2835 per Raspberry B+ e Broadcom BCM2836 per Raspberry 2). Il SOC permette di configurare gli ingressi sia con una resistenza di pull-up che di pull-down oppure nessuna delle due (non consigliato). Nella nostra applicazione, diversamente da quanto fatto per Arduino, abbiamo sfruttato solo i tre pulsanti P1, P2 e P3, tralasciando il jumper J1.

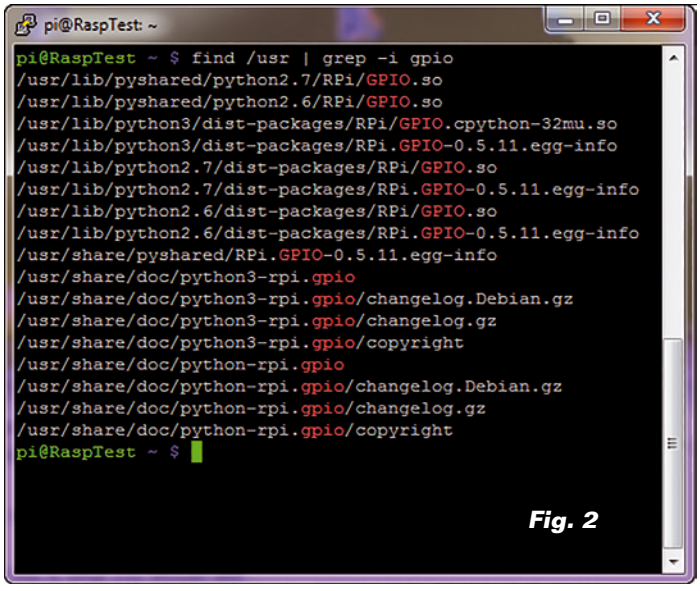



*Fig. 2*

Il led LD8, anch'esso collegato a un pin di I/O, serve a indicare l'avvenuto riconoscimento di una gesture; la gestione del LED LD8 non dipende dalla libreria ma dal codice della demo.
Ricordiamo infine che con il nuovo elettrodo si è deciso di sfruttare gli I/O estesi EIO2, EIO3, EIO6 e EIO7, messi a disposizione da MGC3130, ai quali abbiamo collegato dei LED per segnalare alcune delle gesture riconosciute. Durante la parametrizzazione è quindi possibile decidere quale gesture deve essere monitorata e riportata su una delle possibili uscite o una combinazione di esse. Di seguito l'elenco:

- *Flick West to East;*
- *Flick East to West;*
- *Flick North to South;*
- *Flick South to North;*
- *Single Tap North;*
- *Single Tap South;*
- *Single Tap West;*
- *Single Tap East;*
- *Single Tap Centre;*
- *Clock Wise;*
- *Counter Clock wise.*

## CONFIGURARE RASPBERRY PI

Prima di addentrarci nella descrizione della libreria Python è necessario eseguire dei passi intermedi di configurazione di Rasperry Pi. Vediamo passo passo come procedere: per prima cosa è necessario installare la libreria Python per la gestione degli I/O di Rasperry Pi e, se già installata, provvedere ad aggiornarla all'ultima release disponibile che attualmente si attesta alla 0.5.11.
Per verificare l'attuale versione della libreria GPIO eseguire il seguente comando:

```
find /usr | grep -i gpio
```

La **Fig. 2** evidenzia il feedback del comando inviato; in questo caso la libreria risulta aggiornata all'ultima release. Se la release della libreria dovesse essere inferiore alla 0.5.11 provvedere ad un suo aggiornamento usando i seguenti comandi:

```
sudo apt-get update
sudo apt-get upgrade
```

oppure utilizzare i seguenti:

```
sudo apt-get install python-rpi.gpio
sudo apt-get install python3-rpi.gpio
```

Passiamo ora alla configurazione della periferica I²C messa a disposizione da Raspberry Pi; procediamo

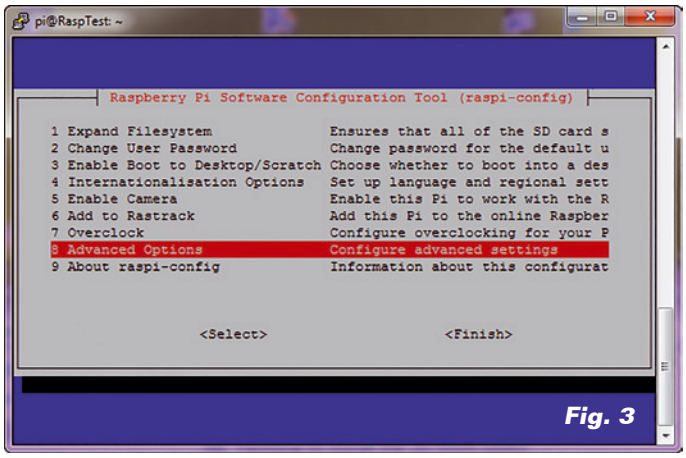



**Fig. 3**

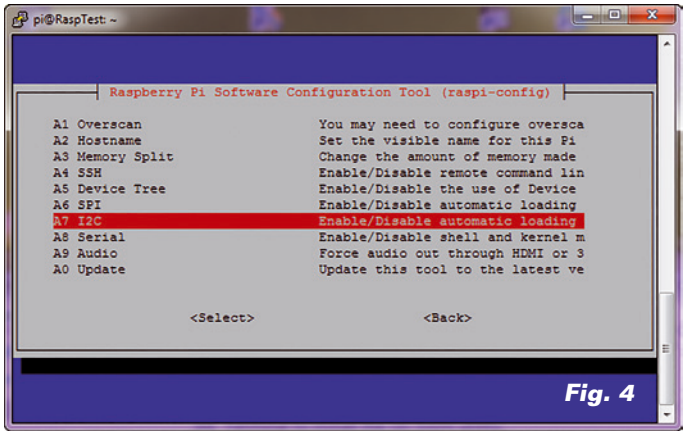



**Fig. 4**

con l'installazione della libreria Python "**smbus**" per la gestione del bus I²C:

```
sudo apt-get install python-smbus
```

seguita dai tools I²C:

```
sudo apt-get install i2c-tools
```

Dobbiamo poi abilitare il supporto del bus I²C da parte del kernel; per fare ciò è necessario richiamare la finestra di configurazione, digitando il seguente comando:

```
sudo raspi-config
```

Come visibile in **Fig. 3**, selezioniamo la voce "*Advanced Options*", quindi:

- nella nuova schermata selezioniamo la voce "**A7 I²C Enable/Disable automatic loading**", come in **Fig. 4**;
- nelle seguenti schermate dobbiamo rispondere in sequenza "**Yes**", "**Ok**", "**Yes**" e infine ancora "**Ok**";
- usciamo dalla finestra di configurazione selezionando la voce "**Finish**".

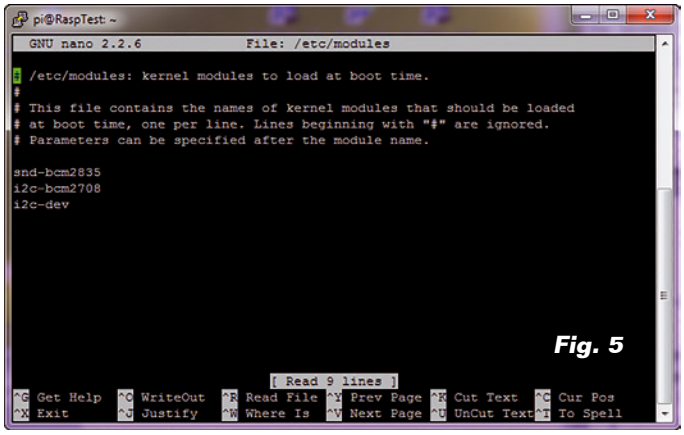



**Fig. 5**

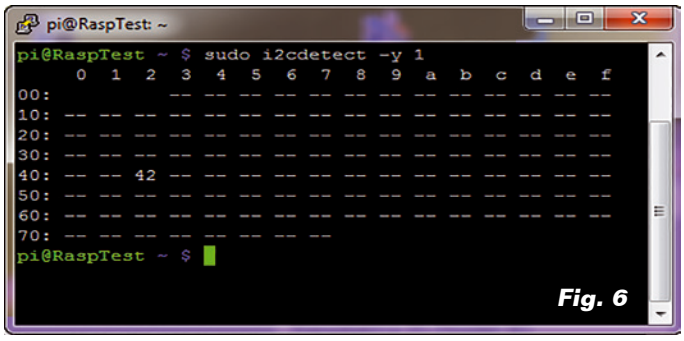



**Fig. 6**

Adesso eseguiamo il reboot del sistema eseguendo il comando:

```
sudo reboot
```

Dopo avere riavviato Raspberry Pi apriamo con un editor di testo generico il file "**modules**" mediante il comando:

```
sudo nano /etc/modules
```

A questo punto dobbiamo aggiungere, se mancanti, le seguenti due righe (**Fig. 5**):

```
i2c-bcm2708
i2c-dev
```

Per ultimo bisogna verificare se nella vostra distribuzione è presente il seguente file:

```
/etc/modprobe.d/raspi-blacklist.conf
```

In caso affermativo apritelo con un editor di testo usando il comando:

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

e, se presenti, commentate (utilizzando il carattere "#") le seguenti istruzioni:

```
blacklist spi-bcm2708
```

```
blacklist i2c-bcm2708
```

A questo punto non rimane che riavviare nuovamente la Raspberry Pi e verificare che la comunicazione I²C sia attiva; per fare ciò eseguite il comando seguente:

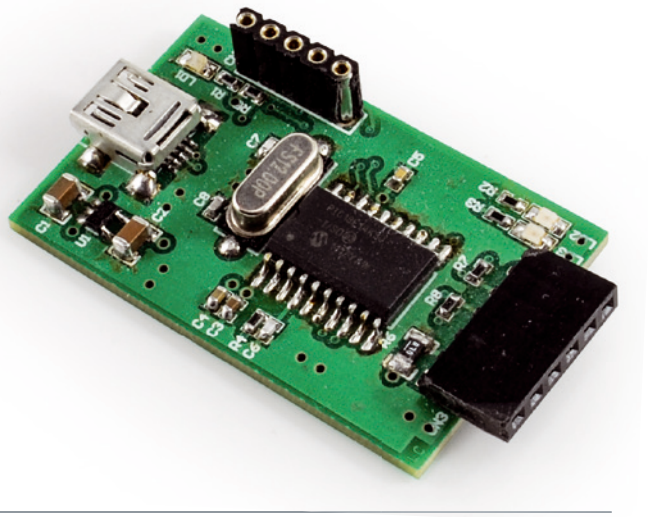
```
sudo i2cdetect -y 1
```

Se tutto è andato a buon fine, il comando inviato ritorna una tabella come quella mostrata in **Fig. 6**. In pratica il comando analizza il bus I²C in cerca di periferiche e nel nostro caso ha individuato l'integrato MGC3130 con indirizzo 0x42.
La configurazione dell'hardware I²C, e relativa libreria di gestione, è conclusa; ora si rende necessario scaricare e installare un modulo Python che simula la pressione dei tasti della tastiera, cosicché si possa associare una gesture a un tasto, ad esempio i tasti funzione F1, F2 ecc.
Il modulo da scaricare è reperibile al seguente link:

```
https://github.com/SavinaRoja/PyUserInput
```

Scaricate il file .zip e scompattatelo in una directory: ad esempio la "**home**" o, se preferite, in una directory da voi creata per l'occasione.
Tra i file scompattati ne troverete uno con nome "**Setup.py**" che vi servirà per installare il modulo nella vostra distribuzione e renderla disponibile nel path Python, cosicché la si possa includere nei propri progetti.
Digitate quindi il comando:

```
python setup.py install
```



Se tutte le dipendenze sono verificate, il modulo viene installato e reso subito disponibile per essere integrato nei vostri progetti. Tuttavia potrebbe succedere che non siano rispettate tutte le dipendenze e che si renda necessario installare il modulo aggiuntivo "**pi3d**" scaricabile dal seguente link:

```
https://github.com/tipam/pi3d
```

Come per il precedente scaricare, scompattare e infine installare il seguente comando:

```
python setup.py install
```

Infine potrebbe essere necessaria la libreria "**xlib**" per Python la quale può essere installata con il seguente comando:

```
sudo apt-get install python-xlib
```

Bene, arrivati a questo punto abbiamo tutto quello che ci serve per introdurre e studiare la libreria di gestione dell'integrato MGC3130 sviluppata in Python.

## MGC3130 E LA LIBRERIA PER RASPBERRY PI

Parliamo ora della libreria scritta in Python per gestire l'integrato MGC3130, la quale si rifà sulla falsa riga di quella già presentata per Arduino. Cominciamo con una puntualizzazione sulla gestione delle strutture dati in Python necessarie a gestire il flusso dati proveniente dall'integrato MGC3130. In Python le strutture dati vengono costruite in modo leggermente differente rispetto al linguaggio di programmazione C, quindi di seguito riportiamo un confronto tra la struttura dati scritta in C per Arduino rispetto alla nuova libreria in Python per Raspberry Pi. Come vi ricorderete dalla precedente puntata, la struttura dati usata con Arduino è composta come mostrato nel **Listato 1**.
Ora mostriamo l'equivalente scritta in Python (**Listato 2**): notate che l'impostazione è differente ma la funzione svolta è la medesima.
Anche se sembrano diverse in realtà sono uguali e svolgono lo stesso compito. Nel nostro caso per accedere a questa struttura dati è necessario dichia-

### Listato 1

```
union GestureOutput {
        uint32_t Gesture;
        uint8_t  GestArray[4];
        struct {
                uint8_t  Byte_0;
                uint8_t  Byte_1;
                uint8_t  Byte_2;
                uint8_t  Byte_3;
        } GestureByte;
        struct {
                uint8_t TouchSouth              :1;  // GESTURE_TOUCH_SOUTH              0x00000001
                uint8_t TouchWest               :1;  // GESTURE_TOUCH_WEST               0x00000002
                uint8_t TouchNorth              :1;  // GESTURE_TOUCH_NORTH              0x00000004
                uint8_t TouchEast               :1;  // GESTURE_TOUCH_EAST               0x00000008
                uint8_t TouchCentre             :1;  // GESTURE_TOUCH_CENTRE             0x00000010
                uint8_t TapSouth                :1;  // GESTURE_TAP_SOUTH                0x00000020
                uint8_t TapWest                 :1;  // GESTURE_TAP_WEST                 0x00000040
                uint8_t TapNorth                :1;  // GESTURE_TAP_NORTH                0x00000080
                uint8_t TapEast                 :1;  // GESTURE_TAP_EAST                 0x00000100
                uint8_t TapCentre               :1;  // GESTURE_TAP_CENTRE               0x00000200
                uint8_t DoubleTapSouth          :1;  // GESTURE_DOUBLE_TAP_SOUTH         0x00000400
                uint8_t DoubleTapWest           :1;  // GESTURE_DOUBLE_TAP_WEST          0x00000800
                uint8_t DoubleTapNorth          :1;  // GESTURE_DOUBLE_TAP_NORTH         0x00001000
                uint8_t DoubleTapEast           :1;  // GESTURE_DOUBLE_TAP_EAST          0x00002000
                uint8_t DoubleTapCentre         :1;  // GESTURE_DOUBLE_TAP_CENTRE        0x00004000
                uint8_t GestWestEast            :1;  // GESTURE_WEST_EAST                0x00008000
                uint8_t GestEastWest            :1;  // GESTURE_EAST_WEST                0x00010000
                uint8_t GestSouthNorth          :1;  // GESTURE_SOUTH_NORTH              0x00020000
                uint8_t GestNorthSouth          :1;  // GESTURE_NORTH_SOUTH              0x00040000
                uint8_t EdgeGestWestEast        :1;  // GESTURE_EDGE_WEST_EAST           0x00080000
                uint8_t EdgeGestEastWest        :1;  // GESTURE_EDGE_EAST_WEST           0x00100000
                uint8_t EdgeGestSouthNorth      :1;  // GESTURE_EDGE_SOUTH_NORTH         0x00200000
                uint8_t EdgeGestNorthSouth      :1;  // GESTURE_EDGE_NORTH_SOUTH         0x00400000
                uint8_t GestClockWise           :1;  // GESTURE_CLOCK_WISE               0x00800000
                uint8_t GestCounterClockWise    :1;  // GESTURE_COUNTER_CLOCK_WISE       0x01000000
                uint8_t FreeBit                 :7;
        } Bit;
} GestureOutput;
```

## Listato 2

```
class GestureBit(Structure):
      _fields_ = [("TouchSouth",          c_uint32, 1),
            ("TouchWest",           c_uint32, 1),
            ("TouchNorth",          c_uint32, 1),
            («TouchEast»,           c_uint32, 1),
            («TouchCentre»,         c_uint32, 1),
            ("TapSouth",            c_uint32, 1),
            ("TapWest",             c_uint32, 1),
            ("TapNorth",            c_uint32, 1),
            ("TapEast",             c_uint32, 1),
            ("TapCentre",           c_uint32, 1),
            ("DoubleTapSouth",      c_uint32, 1),
            ("DoubleTapWest",       c_uint32, 1),
            ("DoubleTapNorth",      c_uint32, 1),
            ("DoubleTapEast",       c_uint32, 1),
            ("DoubleTapCentre",     c_uint32, 1),
            ("GestWestEast",        c_uint32, 1),
            ("GestEastWest",        c_uint32, 1),
            ("GestSouthNorth",      c_uint32, 1),
            ("GestNorthSouth",      c_uint32, 1),
            ("EdgeGestWestEast",    c_uint32, 1),
            ("EdgeGestEastWest",    c_uint32, 1),
            ("EdgeGestSouthNorth",  c_uint32, 1),
            ("EdgeGestNorthSouth",  c_uint32, 1),
            ("GestClockWise",       c_uint32, 1),
            ("GestCounterClockWise", c_uint32, 1),
            ("Free",                c_uint32, 7)]

class GestureByte(Structure):
      _fields_ = [("Byte0", c_uint8),
            ("Byte1", c_uint8),
            ("Byte2", c_uint8),
            ("Byte3", c_uint8)]

class Gesture(Union):
      _fields_ = [("Gesture32Bit", GestureBit),
            ("GestureByte",  GestureByte),
            ("GestureLong",  c_uint32),
            ("GestArray",    c_ubyte * 4)]
```

rare una variabile nel file Python che richiama tale struttura, ad esempio:

```
_GestureOutput = Gesture()
```

Tale dichiarazione può essere fatta come tipo globale e quindi visibile in ogni funzione dichiarata nel file di libreria, a patto che in ogni funzione si dichiari la seguente:

```
global _GestureOutput
```

così facendo si indica all'interprete Python che la variabile è di tipo globale. Per accedere al valore, ad esempio, della gesture Tap su elettrodo Sud si deve scrivere la seguente riga di codice:

```
if (_GestureOutput.Gesture32Bit.TapSouth):
```

Se invece si vuole impostare il valore di tale bit si deve usare questa sintassi:

```
_GestureOutput.Gesture32Bit.TapSouth = 0
```

Oltre a livello di bit è possibile accedere ai dati a livello di singolo byte per un totale di quattro byte (32 bit totali di cui sette liberi per sviluppi futuri) oppure direttamente come double word agendo in un solo colpo su tutti i 32 bit della struttura. Quindi l'accesso diventerebbe:

```
_GestureOutput.GestureByte.Byte0 = 0x00
_GestureOutput.GestureLong = 0x00000000
```

Dopo questa premessa analizziamo le funzioni messe a disposizione dalla nostra libreria; in particolare, abbiamo le seguenti funzioni pubbliche:

```
def MGC3130_SetAdd(Addr)
def MGC3130_ResetDevice(GPIO, Rst):
def MGC3130_ExitResetDevice(GPIO, Rst):
def MGC3130_Begin(GPIO, Ts, Rst):
def MGC3130_GetTsLineStatus(GPIO, Ts):
def MGC3130_ReleaseTsLine(GPIO, Ts):
def MGC3130_GetEvent():
def MGC3130_DecodeGesture():
```

La funzione "**MGC3130_SetAdd**" serve per impostare l'indirizzo hardware occupato sul bus I²C da MGC3130, nel nostro caso è sempre e solo 0x42. L'unico parametro da passare a questa funzione è appunto l'indirizzo hardware.
La funzione "**MGC3130_ResetDevice**", come suggerisce il nome, serve per mantenere resettato l'integrato MGC3130 fintantoché non viene rimossa tale condizione. I parametri da passarle sono due: il primo è la libreria GPIO necessaria per gestire le linee di I/O generiche, mentre il secondo è il pin associato alla linea di RESET.
La funzione opposta è "**MGC3130_ExitResetDevice**" la quale rimuove la condizione di RESET; i parametri da passarle sono i medesimi della funzione precedente.
La funzione "**MGC3130_Begin**" serve per inizializzare le linee di comunicazione tra la Raspberry Pi e l'MGC3130. I parametri da passare alla funzione sono la libreria GPIO, il numero di pin per la linea TS e il numero di pin per la linea RESET.
Il pin TS viene impostato come ingresso con resistenza di pull-up interna abilitata, mentre la linea RESET

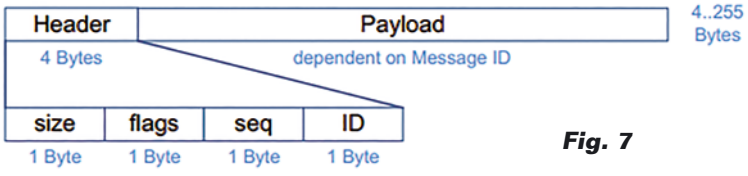



*Fig. 7*

## Listato 3

```
GPIO.setmode(GPIO.BCM)  # set board mode to Broadcom
GPIO.setup(JUMPER_J1, GPIO.IN, pull_up_down=GPIO.PUD_UP)          # Jumper
GPIO.setup(PULSE_P1,  GPIO.IN, pull_up_down=GPIO.PUD_UP)          # Pulsante P1
GPIO.setup(PULSE_P2,  GPIO.IN, pull_up_down=GPIO.PUD_UP)          # Pulsante P2
GPIO.setup(PULSE_P3,  GPIO.IN, pull_up_down=GPIO.PUD_UP)          # Pulsante P3
GPIO.setup(LED_DIAGNOSTIC, GPIO.OUT, initial=GPIO.HIGH)           # Led

GPIO.add_event_detect(PULSE_P1, GPIO.FALLING, bouncetime=100)
GPIO.add_event_detect(PULSE_P2, GPIO.FALLING, bouncetime=100)
GPIO.add_event_detect(PULSE_P3, GPIO.FALLING, bouncetime=100)
```

viene impostata come uscita con valore logico iniziale alto. L'inizializzazione prevede di mantenere in RESET forzato l'integrato per un periodo di 250mSec portandolo così alle condizioni iniziali come se fosse avvenuto un power-up.
La funzione "**MGC3130_GetTsLineStatus**" serve per monitorare la linea TS e intercettare quando l'integrato MGC3130 la impegna per avvisare che è stata riconosciuta una gesture. Non appena il sistema si accorge che la linea TS è stata impegnata, cambia lo stato del pin andando lui stesso ad occupare la linea TS, potendo così iniziare la procedura di lettura dei dati contenuti nel buffer dell'integrato MGC3130. Conclusa la lettura dei dati si può richiamare la funzione "**MGC3130_ReleaseTsLine**" rilasciando così la linea TS.
Per entrambe le funzioni appena esposte bisogna passare due parametri: il primo è la libreria GPIO, mentre il secondo è il pin cui è collegata la linea TS.
La funzione "**MGC3130_GetEvent**" serve per leggere i dati memorizzati nel buffer dell'integrato MGC3130 e salvarli in apposite strutture dati private utilizzate dalla libreria. Sarà poi una successiva funzione a provvedere alla decodifica dei dati ricevuti e a renderli disponibili all'utente finale per la propria applicazione. Vi ricordiamo, come già ampiamente detto nelle precedenti puntate, che il pacchetto dati utilizzato per la lettura dei dati caricati nel buffer dell'integrato MGC3130 è del tipo raffigurato in **Fig. 7**. L'indirizzo contenuto nel pacchetto è quello hardware dell'integrato dove il bit meno significativo indica se si sta eseguendo una scrittura ("0") oppure una lettura ("1"). La sezione "**MGC3130 message**" può essere espansa come indicato in **Fig. 8** identificando due sezioni "**Header**" e "**Payload**". Espandendo ancora la sezione "**Header**", come in **Fig. 9**, si nota che nei primi quattro byte ritroviamo la lunghezza del pacchetto dati ricevuto e, cosa molto importante, l'ID che identifica il pacchetto dati. Nel caso di una lettura dati dopo il riconoscimento di una gesture l'ID del pacchetto sarà 0x91. Per l'elenco dei codici ID disponibili vi invitiamo a rileggere la precedente puntata.
La libreria gestisce anche l'ID 0x83 che identifica la revisione di libreria caricata nell'integrato. Questa informazione viene letta allo start-up dopo avere tolto la condizione di RESET. Attualmente c'è una limitazione nel numero di byte che si possono leggere dal bus I²C sfruttando la libreria "**smbus**", ovvero la lunghezza massima del buffer di ricezione della libreria è impostato a 32 byte il che è un attimino limitante. Questo comporta che il pacchetto dati con ID 0x83 letto dal buffer di MGC3130 è troncato. Per chi volesse cimentarsi è possibile scaricare il sorgente della libreria "**smbus**", cambiarne la lunghezza del buffer, ricompilare e re-installare la libreria nella propria distribuzione.
La funzione "**MGC3130_GetEvent**" applica dei filtri ai dati letti dal buffer di MGC3130 per eliminare le parti non necessarie ai nostri scopi. I dati fondamentali sono le informazioni inerenti le "**GestureInfo**" e le "**TouchInfo**" alle quali si applicano delle maschere di filtro per eliminare i bit indesiderati (la "**MASK_GESTURE_RAW**" viene applicata alle "**GestureInfo**" mentre la "**MASK_TOUCH_RAW**" viene applicata a "**TouchInfo**"). Questi valori, se non per validi motivi, non devono mai essere modificati dall'utente.
L'ultima funzione "**MGC3130_DecodeGesture**" decodifica i dati letti dalla precedente funzione e li ren-


*Fig. 8*

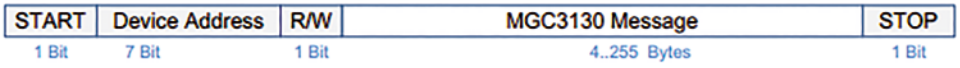
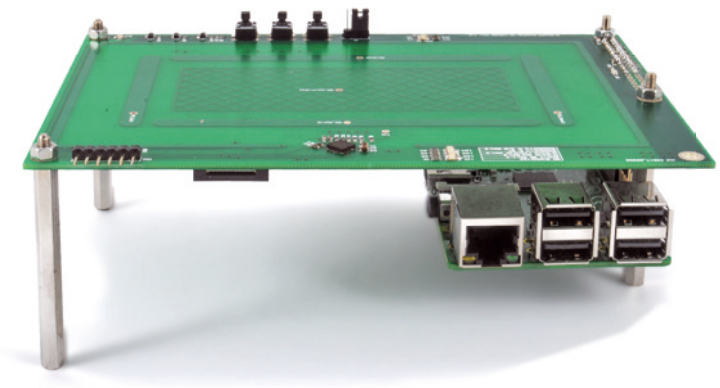



*Fig. 9*



de disponibili attraverso una struttura dati pubblica che l'utente può utilizzare nella propria applicazione (vedere la struttura dati descritta precedentemente). La libreria si accorge delle variazioni intercettate dall'integrato MGC3130 e le riporta sui rispettivi bit della struttura dati pubblica. La gesture è riconosciuta quando il bit corrispondente va a livello logico "**1**". Volendo, ma non è necessario, è possibile filtrare le gesture che non si vogliono utilizzare nel proprio progetto; infatti si può configurare una costante di filtro, presente nel file **MGC3130_DefVar.py**, che permette di filtrare le gesture non volute. La costante si chiama "**MASK_FILTER_GESTURE**" ed è un valore a 32 bit configurabile bit a bit. Se il bit è a "**1**" logico significa che si desidera filtrare la gesture viceversa la si vuole mantenere.
Per favorire lo studio della libreria ci sono delle parti di codice che possono essere attivate/disattivate attraverso delle variabili booleane appositamente dichiarate. Se la variabile è "**True**" attiva la parte di codice interessata, viceversa la disattiva. Le variabili sono le seguenti:

```
1) EnablePrintMGC3130RawFirmwareInfo = False
2) EnablePrintMGC3130RawData         = False
3) EnablePrintMGC3130Gesture         = False
4) EnablePrintMGC3130xyz             = False
```

Per impostazione predefinita, le variabili sono tutte settate al valore "**False**"; se attivate, vedrete a schermo la stampa dei dati letti dall'integrato ed eventualmente le relative decodifiche, in particolare in ordine abbiamo:

```
1) Stampa in formato RAW dei dati inerenti la revisione
   FW caricata nell'integrato
2) Stampa in formato RAW dei dati inerenti la gesture
   riconosciuta
3) Stampa della gesture riconosciuta dopo la decodifica
4) Stampa delle coordinate x, y e z
```

Ad esempio, se la variabile booleana "`EnablePrintMGC3130RawData`" è settata a **True**" l'intercettazione della gesture "**Flick East to West**" genera una tabella dati così formattata:

```
##############################################
Row data from MGC3130
Header: 1A081191
Payload: 1F01 | 86 | 80 | 0073 | 03100000 | 00000000
| 0000 | 000000000000
##############################################
```

Per aumentarne leggibilità e interpretazione, il pacchetto dati viene diviso tra "***Header***" e "**Payload**"; in più il "***Payload***" viene suddiviso in sotto-pacchetti (vedere il datasheet della libreria GestIC).

## MGC3130 DEMO

Descriviamo ora il nostro programma demo, sempre scritto in Python, che ci dà la possibilità di gestire un programma di visualizzazione delle immagini direttamente con le gesture riconosciute da MGC3130. Il programma si chiama "**Eye Of Gnome**" e può essere installato sulla propria Raspberry Pi semplicemente eseguendo il seguente comando:

```
sudo apt-get install eog
```

Analizziamo ora la demo realizzata per Raspberry Pi, il file si chiama "**DemoGestic.py**" e come per tutti i file scritti in Python in testa trova le dichiarazioni di importazione dei moduli necessari al suo corretto funzionamento, tra questi ritroviamo la nostra libreria per la gestione dell'integrato MGC3130 (`import MGC3130`), la libreria per la gestione degli I/O (`import Rpi.GPIO`), una libreria per la gestione dei Thread (`import threading`), una libreria per la gestione dei tempi (`import time`) e infine la libreria di emulazione dei pulsanti della tastiera (`from pykeyboard import PyKeyboard`).
Per semplicità di gestione del codice si dichiarano delle variabili a valore costante che identificano i pin di I/O della Raspberry Pi; in particolare, abbiamo le linee TS e RESET nonché le linee SDA e SCL, i pulsanti, il jumper e il LED di diagnostica. Tutte queste definizioni vengono utili nel momento in cui si vanno a configurare i pin di I/O da utilizzare con Raspberry Pi e tutte le volte che se ne vuole testare il valore o modificarne lo stato.
Come si può osservare nel **Listato 3**, con la funzione "**setup**" si definisce se il pin di interesse è un ingresso o una uscita e, nel caso dei pin di ingresso, si attivano anche le resistenze di pull-up interne. Inoltre per i pulsanti di ingresso P1, P2 e P3 si definisce un ulteriore parametro che introduce la gestione dell'evento pressione pulsante "**add_event_detect**" il

quale interviene sui fronti di discesa con un tempo di debouncing di 100mSec. Nel ciclo infinito del main ritroviamo quindi la gestione dei tre pulsanti che sono in attesa che si scateni l'evento sopra configurato, ad esempio:

```
if GPIO.event_detected(PULSE_P1):
```

Questa istruzione condizionale è in attesa che venga intercettato l'evento di pressione del pulsante P1 per eseguire la propria funzionalità. In particolare abbiamo definito che il pulsante P1 attiva la demo e quindi permette l'associazione di una gesture con un tasto della tastiera, il pulsante P2 disabilita quanto appena detto, mentre il pulsante P3 chiude la demo liberando le porte di I/O dalle configurazioni fatte. Prima di entrare in un ciclo "**while**" infinito il codice esegue una serie di configurazioni tra cui l'indirizzo hardware che occupa l'integrato MGC3130 sul bus $I^2C$, nonché l'inizializzazione delle linee di I/O necessarie a gestire lo stesso. In questo secondo caso, alla funzione di configurazione viene passata come parametro anche la libreria GPIO; infatti questa libreria è bene che sia dichiarata in un solo punto e poi eventualmente passata alle funzioni che ne necessitano. La sezione di codice che si occupa di rilevare la variazione sulla linea TS e quindi provvedere alla lettura del buffer dell'integrato MGC3130 è la seguente:

```
if (MGC3130.MGC3130_GetTsLineStatus(GPIO, MGC3130_TS_LINE) == True):
    MGC3130.MGC3130_GetEvent()
    MGC3130.MGC3130_DecodeGesture()
    MGC3130.MGC3130_ReleaseTsLine(GPIO, MGC3130_TS_LINE)
```

Utilizzando la funzione di libreria:

```
def MGC3130_GetTsLineStatus(GPIO, Ts):
```

si testa la linea TS in attesa che vada a livello logico basso per poi impegnarla e iniziare la procedura di lettura dei dati tramite la funzione di libreria:

```
def MGC3130_GetEvent():
```

Seguono le funzioni di decodifica dei dati letti per estrapolare le gesture riconosciute e la funzione per il rilascio della linea TS:

```
def MGC3130_DecodeGesture():
def MGC3130_ReleaseTsLine(GPIO, Ts):
```

Descriviamo ora l'interazione tra le gesture riconosciute e il visualizzatore di immagini "**Eye Of Gnome**". Una volta avviato il software e selezionato il direttorio con le immagini da guardare è possibile eseguire le seguenti operazioni:

1) sfogliare le immagini verso destra, gesture Flick West to East; simula il tasto "***freccia destra***";
2) sfogliare le immagini verso sinistra, gesture Flick East to West; simula il tasto "***freccia sinistra***";
3) ingrandire l'immagine (Zoom in), gesture Touch North Electrode; simula il tasto "**+**";
4) rimpicciolire l'immagine (Zoom out), gesture Touch South Electrode; simula il tasto "**-**".
5) visualizzare a schermo intero l'immagine, gesture Touch Centre; simula il tasto "***F5***";
6) ruotare l'immagine in senso orario, gesture Clockwise; simula la combinazione "***CTRL + R***";
7) ruotare l'immagine in senso antiorario, gesture Counter Clockwise; simula tasto "***SHIFT + CTRL + R***"";
8) quando l'immagine è ingrandita è possibile spostare la stessa sullo schermo usando le gesture Flick West to East, Flick East to West, Flick North to South e Flick South to North.

Ma come fa il nostro codice a emulare i pulsanti della tastiera? Lo fa grazie alla libreria "**PyUserInput**". Una volta importata questa, con il comando:

```
import PyKeyboard
```

si deve creare l'oggetto:

```
KeyPressed = PyKeyboard()
```

L'oggetto creato può gestire i seguenti eventi:
- ***press_key('h')***; l'esempio emula la pressione del tasto "*h*";
- ***release_key('h')***; l'esempio emula il rilascio del tasto "*h*". Ad un "*press_key*" segue sempre obbligatoriamente un "*release_key*" (a meno che la situazione non richieda di mantenere premuto un tasto per un determinato lasso di tempo);
- ***tap_key('h')***; l'esempio emula il tocco di un tasto della tastiera;
- ***type_string('hello')***; l'esempio emula l'invio di una stringa.

Grazie a questi oggetti si possono anche simulare le combinazioni di tasti; ad esempio, per simulare "**shift**" + "**ctrl**"+ "**M**" bisogna scrivere il codice seguente:

```
KeyPressed.press_key(KeyPressed.shift_key)
KeyPressed.press_key(KeyPressed.control_key)
KeyPressed.tap_key('M')
KeyPressed.release_key(KeyPressed.control_key)
KeyPressed.release_key(KeyPressed.shift_key)
```

Dopo questa premessa possiamo descrivere come abbiamo usato questa libreria per i nostri scopi. Ad esempio, il riconoscimento della gesture "**West to East**" viene associato al tasto freccia verso destra:

```
if (MGC3130._GestOutput.Gesture32Bit.GestWestEast == 1):
    MGC3130._GestOutput.Gesture32Bit.GestWestEast = 0
    GPIO.output(LED_DIAGNOSTIC, GPIO.LOW)
    BaseTimeTimer = threading.Timer(1, SlowBaseTime)
    BaseTimeTimer.start()
    KeyPressed.tap_key(KeyPressed.right_key)
```

Simulando il tasto freccia destro le immagini vengono scrollate verso destra. L'equivalente per lo scrolling delle immagini verso sinistra sarà:

```
if (MGC3130._GestOutput.Gesture32Bit.GestEastWest == 1):
    MGC3130._GestOutput.Gesture32Bit.GestEastWest = 0
    GPIO.output(LED_DIAGNOSTIC, GPIO.LOW)
    BaseTimeTimer = threading.Timer(1, SlowBaseTime)
    BaseTimeTimer.start()
    KeyPressed.tap_key(KeyPressed.left_key)
```

Quando una gesture viene riconosciuta viene fatto lampeggiare per un secondo il led LD8. Si è quindi creato un Thread che viene attivato e richiamato dopo un periodo di un secondo dopo ogni riconoscimento di una gesture. Più in dettaglio quando la gesture viene riconosciuta si accende il led LD8 sfruttando la linea di codice:

```
GPIO.output(LED_DIAGNOSTIC, GPIO.LOW)
```

Subito dopo viene attivato il Thread che si occuperà di spegnere il led LD8 dopo un secondo.
Più precisamente l'attivazione del Thread viene fatta eseguendo il codice:

```
BaseTimeTimer = threading.Timer(1, SlowBaseTime)
BaseTimeTimer.start()
```

La funzione richiamata dal Thread è la seguente:

```
def SlowBaseTime():
    global BaseTimeTimer

    GPIO.output(LED_DIAGNOSTIC, GPIO.HIGH)
    aseTimeTimer.cancel()
```

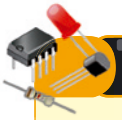
Questa funzione spegne il led LD8 e cancella il Thread richiamato cosicché sia possibile richiamarlo al prossimo evento.
Con questo abbiamo terminato la descrizione della nostra demo di gestione dell'elettrodo per MGC3130 abbinato a una scheda Raspberry Pi B+ o 2.0. Ricordatevi che per attivare l'interazione tra le gesture e la libreria che emula i tasti della tastiera dovete premere il pulsante P1, viceversa per disattivarla premere P2.

## DUE PAROLE SU "EOG" E LA NOSTRA DEMO

Per avviare il visualizzatore di immagini "*EOG*" precedentemente installato bisogna, prima di tutto, avviare il server "*X*" con il comando "*startx*". Dopodiché cliccare sulla voce "*Menu → Graphics → Image Viewer*" per aprire il programma "*EOG*". Selezionare la cartella contenente le foto su cui si vuole lavorare cliccando sulla voce di menu "*Image → Open*". Conclusa questa fase possiamo avviare lanostra demo Python utilizzando "*LXTerminal*" e al prompt digitare "*python DemoGestic.py*" per avviare la demo.
Ancora una volta vi ricordiamo che per attivare l'interazione tra la demo e il visualizzatore di immagini dovete premere sul pulsanti P1, per disattivare l'interazione premere su P2 e per uscire dalla demo e liberare le porte di I/O premere su P3.

## CONCLUSIONI

In questo quarto ed ultimo articolo dedicato alla tecnologia GestIC della Microchip abbiamo imparato ad utilizzare il nuovo elettrodo già descritto -sul piano hardware- nel fascicolo scorso (giugno 2015) con la scheda Raspberry Pi B+/2.0, abbiamo studiato la libreria Python di gestione dell'integrato MGC3130 e mostrato un esempio pratico di utilizzo dell'elettrodo.
Ora dopo questa rassegna di progetti sul riconoscimento delle gesture, possiamo dire di avervi dato le basi per poter sviluppare i vostri progetti integrando così il riconoscimento gestuale in una moltitudine di diverse applicazioni, come la visualizzazione di pagine di menu, di immagini, ma anche l'apertura e la chiusura di tapparelle e tende motorizzate, l'accensione e lo spegnimento di luci e via di seguito.



## per il MATERIALE

Il software Aurea di Microchip può essere scaricato dal sito www.microchip.com/gestic.
Il sensore di prossimità (cod. MGC3130) è reperibile presso Futura Elettronica a Euro 9,80, così come la scheda Gesture (cod. FT1214M) che viene fornita assemblata al prezzo di Euro 58,00. Tutti i prezzi sono IVA compresa.

Il materiale va richiesto a:
Futura Elettronica, Via Adige 11, 21013 Gallarate (VA)
Tel: 0331-799775 • Fax: 0331-792287
http://www.futurashop.it