HALRUN, HAL_PI_GPIO E L'INTERFACCIAMENTO CON IL RASPBERRY PI IN AMBIENTE RASPBIAN PREEMPT-RT

(versione del 19/05/2020)

Introduzione

Il software LinuxCNC utilizza una versione modificata del sistema operativo Linux e in particolare una versione cosiddetta RT (Real Time) denominata Preempt_RT. Questa versione modificata del sistema operativo Linux cerca di garantire la temporizzazione esatta degli eventi, caratteristica fondamentale per il controllo di dispositivi.

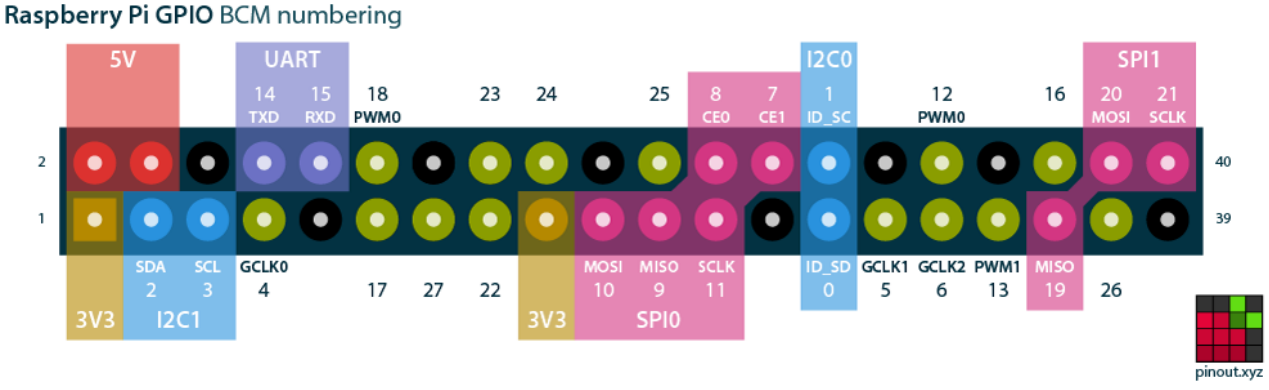
L'utilizzazione delle caratteristiche RealTime viene fatta tramite una serie di funzioni che vengono dette HAL (Hardware Abstraction Layer).

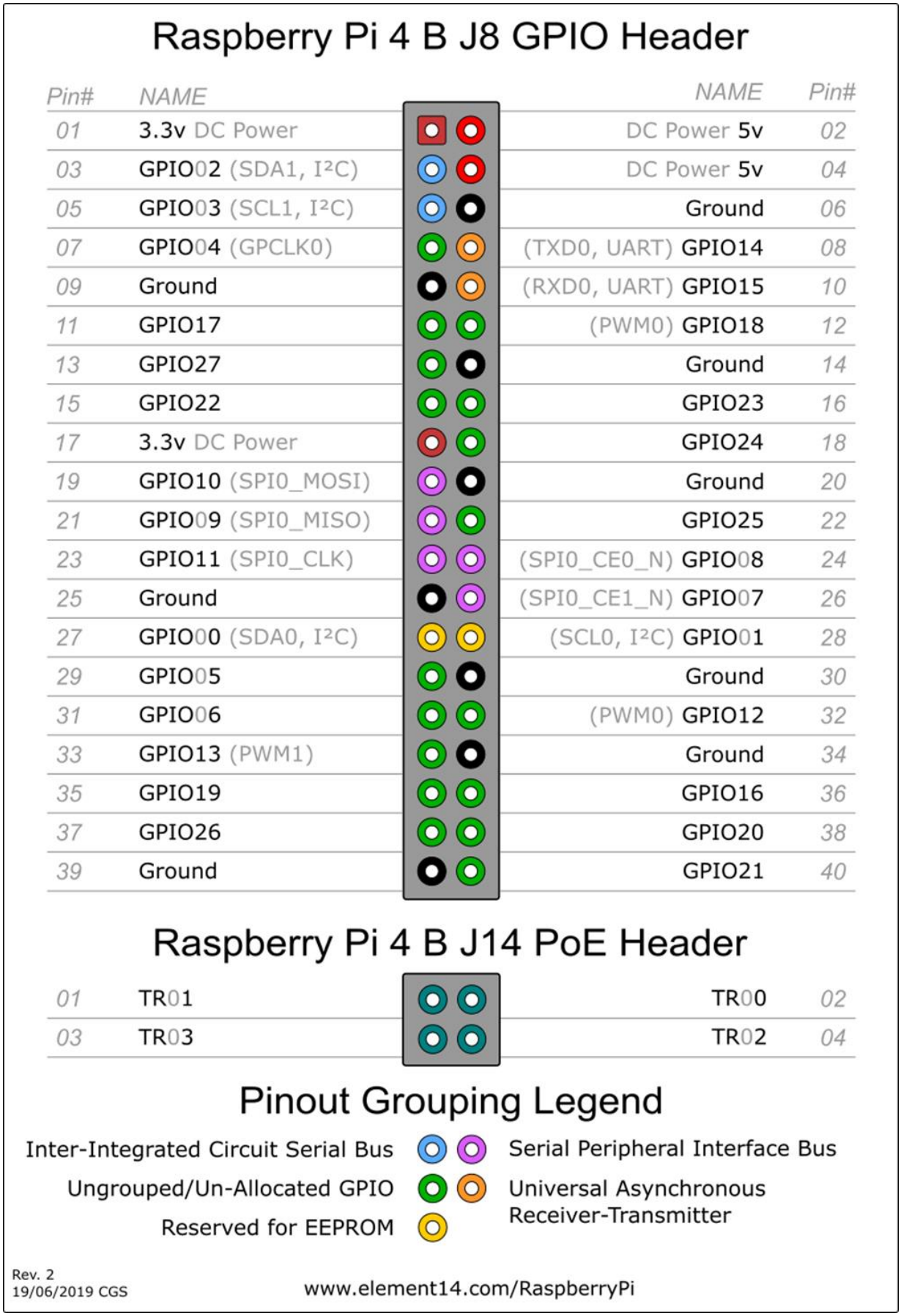
Pertanto riferendosi a quanto, ad esempio, riportato su wikipedia: (https://it.wikipedia.org/wiki/Hardware_abstraction_layer):

"Hardware Abstraction Layer (HAL) o strato di astrazione dall'hardware è un insieme di funzioni di I/O il più possibile generiche e semplici, il cui compito è di tenere conto di tutte le differenze fra dispositivi fisici diversi al posto del programma che lo userà, nascondendogli la vera identità e natura di essi.

Dotando un programma di un HAL se ne migliora la portabilità su altri tipi di computer/sistemi operativi e la funzionalità con dispositivi diversi, perché eventuali modifiche e adattamenti vanno fatti solamente nell'HAL senza toccare il codice del programma stesso; inoltre è relativamente facile aggiungere, all'occorrenza, una sezione all'HAL per gestire un dispositivo che non era stato inizialmente previsto."

Preempt_RT, sviluppato principalmente in ambiente x86, è stato anche adattato al processore ARM e in particolare al sistema operativo Raspbian (*https://www.youtube.com/watch?v=ntu55fiU18w*) consentendo così l'utilizzazione del LinuxCNC su RaspBerry. Inoltre è stato sviluppato un modulo HAL (hal_pi_gpio) che consente l'uso delle porte GPIO del Raspberry (https://it.pinout.xyz/#) direttamente dall'ambiente HAL.





La disposizione e la loro denominazione dei pin del Raspberry è evidenziata anche dal comando

```
pi@raspberrypi:~ $ gpio readall
```

```
+-----+-----+---------+------+---+---Pi 4B--+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
|     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
|   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5v      |     |     |
|   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
|   4 |   7 | GPIO. 7 |   IN | 0 |  7 || 8  | 0 | IN   | TxD     | 15  | 14  |
|     |     |      0v |      |   |  9 || 10 | 1 | IN   | RxD     | 16  | 15  |
|  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
|  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
|  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
|     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
|  10 |  12 |    MOSI | ALT0 | 0 | 19 || 20 |   |      | 0v      |     |     |
|   9 |  13 |    MISO | ALT0 | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
|  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 1 | OUT  | CE0     | 10  | 8   |
|     |     |      0v |      |   | 25 || 26 | 1 | OUT  | CE1     | 11  | 7   |
|   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
|   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
|   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
|  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
|  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
|  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
|     |     |      0v |      |   | 39 || 40 | 1 | IN   | GPIO.29 | 29  | 21  |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+---Pi 4B--+---+------+---------+-----+-----+
```

In questo documento si mostrerà come si può utilizzare questo modulo (*hal_pi_gpio*) insieme alle altre funzioni HAL per l'interfacciamento del Raspberry con alcuni dispositivi tramite il programma *halrun*.

Il modulo hal_pi_gpio

Come già detto, per potere usare le porte di GPIO del Raspberry tramite HAL è necessario il modulo hal_pi_gpio. Analogamente agli altri moduli, per caricare questo modulo la riga da inserire nel file .hal dei comandi sarà:

```
loadrt hal_pi_gpio [dir=… exclude=…]
```

I parametri *dir* ed *exclude* sono opzionali e servono per descrivere, tramite *dir*, quali funzione hanno i vari pin (input o output) e tramite *exclude* quelli realmente utilizzati. Se non viene specificato nulla con *exclude*, tutti i pin saranno utilizzabili e se si omette l'opzione *dir* tutti i pin saranno utilizzabili come output.

Per la modifica del valore di *exclude* si può usare una maschera binaria che utilizza 0 per indicare se il pin è utilizzato e 1 in caso contrario. Questa maschera si riferisce alla numerazione dei pin BCM dal 27 a 2 (i pin 1 e 0 non sono utilizzati).
Analogamente per la modifica dei valori di *dir* si può usare una maschera binaria che utilizza 0 per indicare se il pin è in uscita e 1 se in ingresso. Anche questa maschera si riferisce alla numerazione dei pin BCM dal 27 a 2 (i pin 1 e 0 non sono utilizzati).

```
halcmd: loadrt hal_pi_gpio
halcmd: show pin
Component Pins:
Owner   Type  Dir         Value  Name
     4  bit   IN          FALSE  hal_pi_gpio.pin-03-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-05-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-07-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-08-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-10-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-11-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-12-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-13-out
```

```
     4  bit   IN          FALSE  hal_pi_gpio.pin-15-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-16-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-18-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-19-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-21-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-22-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-23-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-24-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-26-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-29-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-31-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-32-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-33-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-35-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-36-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-37-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-38-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-40-out
     4  s32   OUT             0  hal_pi_gpio.read.time
     4  s32   OUT             0  hal_pi_gpio.write.time
```

Quindi indicando *dir*=-1 (che da un valore binario con tutti i bit uguali ad 1) significa che tutti i pin sono di output (valore di default) mentre utilizzando *dir*=0 (che da un valore binario con tutti i bit uguali a 0) significa che tutti i pin sono di input.

Ad esempio nel caso dell'utilizzazione del Raspberry, se i pin utilizzati e il tipo (8 In e 8 Out) sono quelli riportati nello schema nella tabella seguente, i valori da riportare sono i seguenti:

```
    dir= 1412864    (decimale di 0x 00000101011000111100000000)
exclude=  536671    (decimale di 0x 0x00000100000110000001011111)
```

```
halcmd: loadrt hal_pi_gpio dir=1412864 exclude=536671
```

a cui corrisponderà questa configurazione:

```
halcmd: show pin
Component Pins:
Owner   Type  Dir         Value  Name
     4  bit   IN          FALSE  hal_pi_gpio.pin-11-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-12-out
     4  bit   OUT         FALSE  hal_pi_gpio.pin-13-in
     4  bit   OUT         FALSE  hal_pi_gpio.pin-16-in
     4  bit   OUT         FALSE  hal_pi_gpio.pin-18-in
     4  bit   IN          FALSE  hal_pi_gpio.pin-19-out
     4  bit   OUT         FALSE  hal_pi_gpio.pin-21-in
     4  bit   OUT         FALSE  hal_pi_gpio.pin-22-in
     4  bit   IN          FALSE  hal_pi_gpio.pin-23-out
     4  bit   OUT          TRUE  hal_pi_gpio.pin-26-in
     4  bit   IN          FALSE  hal_pi_gpio.pin-32-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-33-out
     4  bit   OUT         FALSE  hal_pi_gpio.pin-35-in
     4  bit   OUT         FALSE  hal_pi_gpio.pin-36-in
     4  bit   OUT         FALSE  hal_pi_gpio.pin-37-in
     4  bit   IN          FALSE  hal_pi_gpio.pin-38-out
     4  bit   IN          FALSE  hal_pi_gpio.pin-40-out
     4  s32   OUT             0  hal_pi_gpio.read.time
     4  s32   OUT             0  hal_pi_gpio.write.time
```

BCM GPIO	Board	Nome wiringPi	Exclude 1=si 0=no	Dir 1=out 0=in
27	13	GPIO.2	0	0
26	37	GPIO.25	0	0
25	22	GPIO.6	0	0
24	18	GPIO.5	0	0
23	16	GPIO.4	0	0
22	40	GPIO.3	1	0
21	15	GPIO.29, SCLK1, PCMDout	0	1
20	38	GPIO.28, MOSI1,PCMfs	0	1
19	35	GPIO.24, MISO1, GOUI16	0	0
18	12	GPIO.1, PWM0	0	1
17	11	GPIO.0	0	1
16	36	GPIO.27, PCMDin	0	0
15	10	RXD	1	0
14	8	TXD	1	0
13	33	GPIO.23, PWM1	0	1
12	32	GPIO.26, PWM0	0	1
11	23	SCLK0	0	1
10	19	MOSI0	0	1
9	21	MISO0	0	0
8	24	CE0, SCE0	1	0
7	26	CE1, SCE1	0	0
6	31	GPIO.22	1	0
5	29	GPIO.21	1	0
4	7	GPIO.7, GCLK0	1	0
3	5	SCL.1	1	0
2	3	SDA.1	1	0

Il modulo *hal_pi_gpio* presenta alcune funzioni necessarie al suo funzionamento. Le funzioni disponibili sono visibili tramite il comando *show funct:*

```
halcmd: show funct
Exported Functions:
Owner   CodeAddr  Arg       FP   Users  Name
 00004  76f38ba4  00000000  NO       0   hal_pi_gpio.read
 00004  76f38b08  00000000  NO       0   hal_pi_gpio.write
```

Queste due funzioni si occupano di leggere e scrivere i valori dei pin durante il procedere del programma. Dovranno essere associate al thread più veloce (FP=0) ed è buona norma forzare la lettura degli input prima di qualsiasi altra operazione ed effettuare la scrittura degli output alla fine di tutte le operazioni.
Queste condizioni comportano la definizione delle seguenti righe:

```
loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000
addf hal_pi_gpio.read fast 1
addf hal_pi_gpio.write fast -1
```

Per verificare tutto ciò completiamo il programma con 2 ulteriori comandi:

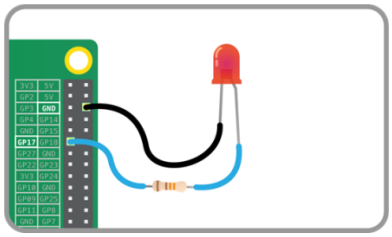
```
loadusr halmeter
start
```

Adesso possiamo passare alla verifica sperimentale.

Pulsanti, bottoni, finecorsa, interruttori, switch.

La verifica più semplice da fare è quella di usare un pulsante. Prendiamo ad esempio un classico finecorsa. Questo presenta tre terminali, a volte numerati da 1 a 3 oppure C, NO ed NC. Nella condizione standard il contatto tra 1 e 2 risulta chiuso e tra 1 e 3 risulta aperto. Alla pressione del tasto le condizioni si invertono. Quindi se vogliamo utilizzare un pin di input, ad esempio il GPIO.5, sarà sufficiente collegare il terminale 1 (*C=Common*) al pin 18, il terminale 2 (*NO=Normally Open*) al piedino 1 del Raspberry (3.3v) e il terminale 3 (NC=Normally Closed) al piedino 6 del Raspberry (GND). C'è da notare che se il pin 18 non è collegato, il suo stato è FALSE, mentre nel caso del pin 26, nel caso non sia collegato, il suo stato è TRUE. Tramite *halmeter* è possibile vedere il cambiamento di stato che interviene alla pressione del tasto. Quindi nel caso in cui si voglia invertire la condizione sarà sufficiente scambiare i terminali 2 e 3 oppure utilizzare il pin 26, che parte dallo stato di TRUE, collegato al terminale C e il piedino NO al GND.

Accensione, spegnimento, led e relè.



Un'altra semplicissima verifica è quella relativa all'attivazione e la disattivazione di un pin. Per fare ciò possiamo utilizzare un led collegato ad un piedino. Il terminale più lungo del led va collegato al pin del RaspBerry tramite una resistenza (può andare bene anche una da 330 Ohm) e il terminale più corto sul GND. Ad esempio si può utilizzare il pin 11 (BCM 17 – wiringPi 0).

Ovviamente in questo caso il comando per l'accensione del led sarà

```
halcmd: setp hal_pi_gpio.pin-11-out 1
```

e quello per lo spegnimento sarà

```
halcmd: setp hal_pi_gpio.pin-11-out 0
```

Nel caso in cui sull'uscita ci sia un dispositivo (ad esempio un piccolo motore DC a 3.3V) questi comandi comporteranno anche l'accensione e lo spegnimento del dispositivo.

Pertanto se, ad esempio, si vuole attivare il pin 11 tramite il finecorsa presente sul pin 18 basterà utilizzare il seguente comando che inoltre definirà un nuovo segnale chiamato *led*

```
halcmd: net led <= hal_pi_gpio.pin-18-in => hal_pi_gpio.pin-11-out
```

In questo caso il pulsante comanderà l'accensione del led.

HAL ha numerose funzioni per gestire i segnali (http://linuxcnc.org/docs/html/ sezione "Realtime components and kernel modules"). Ad esempio, per accendere e spegnere il led, si può usare anche la funzione *toggle* che cambia lo stato di un bit (http://linuxcnc.org/docs/html/man/man9/toggle.9.html).

Per fare ciò basta aggiungere i seguenti comandi:

```
halcmd: loadrt toggle
halcmd: addf toggle.0 fast
```

```
halcmd: net scatto toggle.0.in <= hal_pi_gpio.pin-18-in
halcmd: net led toggle.0.out
```

Adesso alla pressione dello switch il led si accenderà e spegnerà alternativamente.

In modo analogo si possono usare pure dei relè.

Facendo riferimento al relè in figura che è in grado di pilotare 2 canali, collegando l'alimentazione 5V a VCC, il GND e due pin del Raspberry, ad esempio il 3 e il 5, agli ingressi IN1 e IN2 del relè, i seguenti comandi agiranno sui due canali.

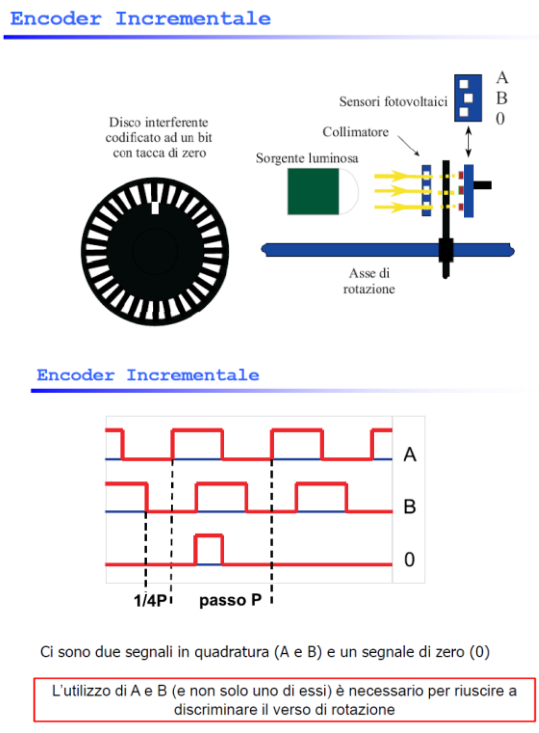
```
halcmd: setp hal_pi_gpio.pin-38-out 1
halcmd: setp hal_pi_gpio.pin-38-out 0
halcmd: setp hal_pi_gpio.pin-40-out 0
halcmd: setp hal_pi_gpio.pin-40-out 1
```

Per pilotare un canale del relè dal pulsante con il comando *toggle* si dovranno usare i seguenti comandi:

```
halcmd: unlinkp toggle.0.out
halcmd: net R1 toggle.0.out => hal_pi_gpio.pin-38-out
```

Adesso, alla pressione dello switch, il relè si accenderà e spegnerà alternativamente.

Il modulo encoder e sim_encoder.



Abbiamo già visto precedentemente il funzionamento di un encoder digitale incrementale. Quelli ottici rotazionali hanno due segnali di uscita, A e B, che emettono impulsi quando il dispositivo viene ruotato. Utilizzati insieme, i segnali A e B indicano sia il verificarsi che la direzione del movimento. Molti encoder incrementali hanno un segnale di uscita aggiuntivo, in genere designato con Z, che indica che l'encoder si trova in una particolare posizione di riferimento. Gli impulsi emessi dalle uscite A e B sono codificati in quadratura, il che significa che quando l'encoder incrementale si muove a una velocità costante, il *duty-cycle* di ogni impulso è del 50% (cioè la forma d'onda è un'onda quadra) e c'è una differenza di fase di 90 gradi tra A e B. In un determinato momento, la differenza di fase tra i segnali A e B sarà positiva o negativa a seconda della direzione di movimento dell'encoder. Nel caso di un encoder rotativo, la differenza di fase è di 90 gradi per la rotazione in senso orario e di 90 gradi per la rotazione in senso antiorario o viceversa, a seconda del design del dispositivo. La frequenza degli impulsi sull'uscita A o B è direttamente proporzionale alla velocità dell'encoder (variazione della posizione); frequenze più alte indicano un movimento rapido, mentre le frequenze più basse indicano velocità più lente.

Halrun fornisce due componenti *realtime* per l'encoder, *encoder* e *sim_encoder*. Il primo serve per acquisire ed elaborare i segnali mentre il secondo è una comoda utility per simularne il funzionamento. Vediamo come operano.

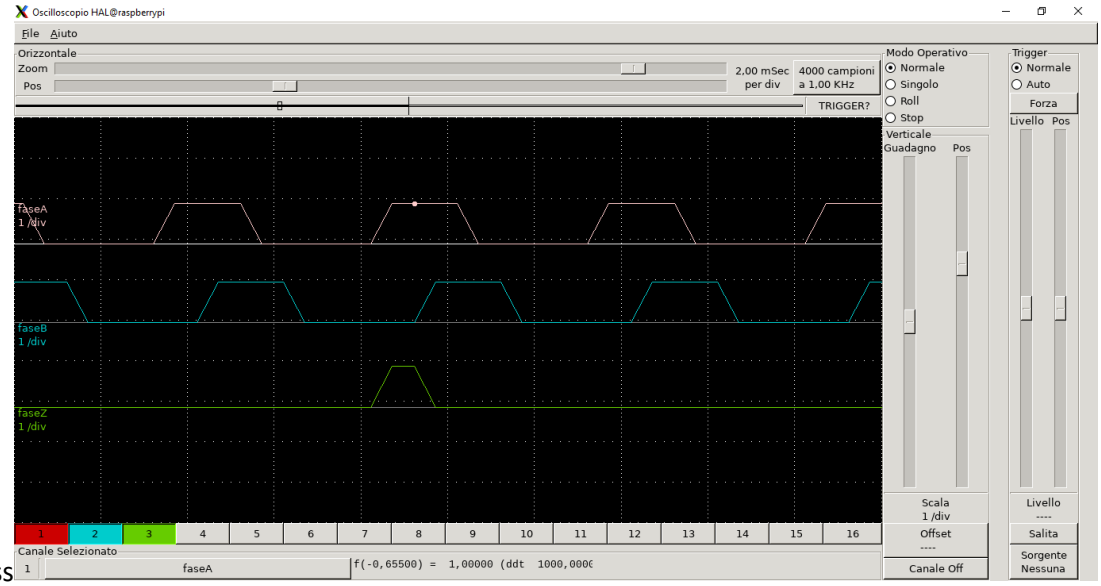
Dopo il loro caricamento e una volta definiti i *threads* necessari al loro funzionamento, aggiungiamo il richiamo delle funzioni.

```
loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000
loadrt sim_encoder num_chan=1
loadrt encoder num_chan=1
addf sim-encoder.make-pulses fast
addf sim-encoder.update-speed slow
addf encoder.update-counters fast
addf encoder.capture-position slow
```

Definiamo i tre segnali (*faseA, faseB e faseZ*) collegando l'uscita del *sim-encoder* simulato all'ingresso del *encoder* e regoliamo i parametri necessari per il funzionamento, velocità e *ppr* (*pulse per revolution*), carichiamo l'oscilloscopio virtuale e avviamo il programma.

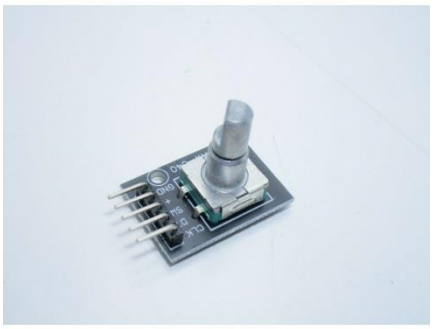
```
setp sim-encoder.0.speed 1
setp sim-encoder.0.ppr 200
loadusr halscope
start
```

Collegando i tre segnali (*faseA, faseB e faseZ*) ai tre canali dell'oscilloscopio virtuale vedremo una schermata simile a questa. Vedremo poi come ricavare altri parametri utili alle nostre necessità, come la velocità.



SS

Un encoder può anche essere usato come contatore o trasduttore di posizione. Utilizziamo quindi un encoder come quelli in figura. Questo tipo di encoder non ha il canale Z, ma uno switch che si attiva alla pressione del tasto. In questo caso colleghiamo, oltre a VCC e GND, il pin denominato CLK (S1) alla fase A sul pin 16 del Raspberry, quello DT (S2) alla fase B sul pin 18 e SW (Key) sul pin 26. Poiché alla pressione del tasto SW il pin viene portato sul GND, si deve utilizzare il pin 26 perché normalmente è in condizione True. Per questo motivo inoltre è necessario utilizzare la condizione logica *not* che nega la condizione in ingresso.



```
loadrt hal_pi_gpio dir=1412864 exclude=536671
loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000
addf hal_pi_gpio.read fast 1
addf hal_pi_gpio.write fast -1

loadrt encoder num_chan=1
addf encoder.update-counters fast
addf encoder.capture-position slow

loadrt not
addf not.0 fast

net faseA <= hal_pi_gpio.pin-16-in  => encoder.0.phase-A
net faseB <= hal_pi_gpio.pin-18-in  => encoder.0.phase-B
net tasto <= hal_pi_gpio.pin-26-in => not.0.in
net reset <= not.0.out => encoder.0.reset
net posizione <= encoder.0.counts

loadusr halmeter
start
```

L'analisi dei segnali riportati da *halmeter* evidenzia che ad ogni scatto il contatore incrementa il suo valore di quattro unità in quanto tiene conto sia del fronte di salita che quello di discesa. Inoltre la pressione del tasto SW porta a zero il contatore.

Il modulo pwmgen.

Halrun è corredato di diversi moduli, utili per le varie attività da svolgere. Tra questi uno dei più utili è sicuramente *pwmgen,* un modulo capace di generare dei segnali digitali periodici che attivano e disattivano una uscita per frazioni costanti del tempo di ciclo. Questa modalità viene usata ad esempio per variare la luminosità apparente di un led o la velocità di un motore DC.

Per la sua utilizzazione sarà necessario inserire il seguente comando (vedi anche il file "*Hal Tutorial*", http://linuxcnc.org/docs/html/man/man9/pwmgen.9.html):

```
loadrt pwmgen output_type=0
```

Tramite il comando *show funct* si può notare la presenza di due funzioni che corredano il modulo, *pwmgen.make-pulses* e *pwmgen.update* (per i dettagli consultare la guida di Hal).

```
halcmd: show funct
Exported Functions:
Owner   CodeAddr  Arg       FP   Users  Name
 00004  76f31ba4  00000000  NO       1   hal_pi_gpio.read
 00004  76f31b08  00000000  NO       1   hal_pi_gpio.write
 00012  760f2880  7631a188  NO       0   pwmgen.make-pulses
 00012  760f2a0c  7631a188  YES      0   pwmgen.update
```

La *pwmgen.make-pulses* si occupa di generare gli impulsi, attività che non richiede il floating point e quindi da collegare al thread *fast*. L'altra *pwmgen.update* si occupa di gestire i vari parametri e richiedendo il floating point deve essere collegato al thread *slow*.

```
addf pwmgen.update slow
addf pwmgen.make-pulses fast
```

Tramite il comando *show pin* possiamo visualizzare i vari parametri del modulo *pwmgen*.

```
halcmd: show pin
Component Pins:
Owner   Type  Dir         Value  Name
.........
.........
    12  float OUT             0  pwmgen.0.curr-dc
    12  bit   I/O         FALSE  pwmgen.0.dither-pwm
    12  bit   IN          FALSE  pwmgen.0.enable
    12  float I/O             1  pwmgen.0.max-dc
    12  float I/O             0  pwmgen.0.min-dc
    12  float I/O             0  pwmgen.0.offset
    12  bit   OUT         FALSE  pwmgen.0.pwm
    12  float I/O             0  pwmgen.0.pwm-freq
    12  float I/O             1  pwmgen.0.scale
    12  float IN              0  pwmgen.0.value
    12  s32   OUT             0  pwmgen.make-pulses.time
    12  s32   OUT             0  pwmgen.update.time
     9  s32   OUT             0  slow.time
```

I principali parametri di interesse sono:

pwmgen.0.pwm-freq: frequenza del segnale (cicli al secondo)

pwmgen.0.scale: range (definito dall'utente) di variazione del valore del segnale
pwmgen.0.value: valore (definito dall'utente) del segnale o *duty-cycle*
pwmgen.0.enable: abilita o meno la generazione del segnale
pwmgen.0.pwm: segnale generato

Se, ad esempio, volessimo far lampeggiare un led posto sul piedino 33 con un intervallo di 1 secondo di accensione e 1 secondo di spegnimento, comandando tale attività tramite un pulsante connesso al piedino 18, potremo usare i seguenti comandi:

```
setp pwmgen.0.pwm-freq 0.5
setp pwmgen.0.scale 100
setp pwmgen.0.value 50
net led-pwm pwmgen.0.pwm => hal_pi_gpio.pin-33-out
net led pwmgen.0.enable <= hal_pi_gpio.pin-18-in
start
```

E' possibile anche variare i parametri per modulare la luminosità del led. In questo caso potremmo usare una frequenza di 50 Hz e variare il valore di *pwmgen.0.value* tra 0 e 100.

```
setp pwmgen.0.pwm-freq 50
setp pwmgen.0.value 30 (o 50, 70, etc…)
```

La modalità PWM è utilizzata anche dai servomotori. In questo caso la frequenza da utilizzare è pari a 50 Hz mentre i valori di escursione del servomotore si hanno con valori di *duty-cycle* compresi tra il 2.5 e il 12.5 % del tempo di ciclo.

Ad esempio possiamo utilizzare il piedino 33 sul RaspBerry, collegandolo al terminale arancio del servomotore. Il terminale rosso andrà ai 5V e quello nero al GND. I comandi conseguenti saranno:

```
net motore pwmgen.0.pwm => hal_pi_gpio.pin-33-out
setp pwmgen.0.value 2.5 (o 5, 10, etc…)
```

Per visualizzare meglio il movimento possiamo, ad esempio, generare un'onda triangolare che vada da 2.5 a 12.5 in un intervallo di 2 secondi. Per fare ciò possiamo usare il generatore di forme d'onda *siggen* già visto in precedenza.

```
loadrt siggen
addf siggen.0.update slow
setp siggen.0.amplitude 5
setp siggen.0.offset 7.5
setp siggen.0.frequency 0.5
net serv siggen.0.triangle => pwmgen.0.value
```

Vedremo il servo motore andare avanti e indietro in un intervallo di tempo pari a 2 secondi. Per arrestare il motore si può disabilitare l'*enable* del *pwmgen* oppure porre il valore della *frequency* del *siggen* a zero.

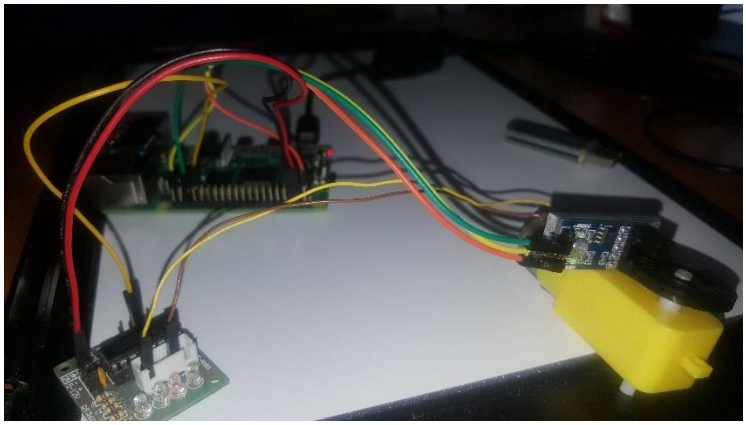
```
setp pwmgen.0.enable 0
```

oppure

```
setp siggen.0.frequency 0
```

Uso di *pwmgen* per il controllo di un motore DC

Prendiamo il caso di utilizzazione di un motore DC come questo in figura. Tramite la modalità PWM possiamo regolare la velocità di rotazione del motore DC ma non possiamo cambiarne il verso di rotazione perché non è possibile invertire la polarità dei pin. Per pilotare il motore, poiché la corrente in uscita dal pin del Raspberry non è sufficiente a muovere il motore, è necessario un collegamento come in foto, utilizzando il driver del motore unipolare.



Quindi bisogna collegare i due poli del motore uno su un pin del driver (ad esempio B) e l'altro al 5° pin GND. Quindi collegare l'ingresso corrispondente (per B si deve utilizzare IN2) al pin del Raspberry (ad esempio il pin 33). Infine alimentare il driver tramite 5V, presi dai pin 4 (5V) e 6 (GND).

Per completare il tutto è possibile usare un disco codificato con 20 fori da montare sull'asse del motore e il sensore di velocità da montare a cavallo del disco. Anche il sensore dovrà essere alimentato a 5V e il pin denominato D0 collegato al pin 36 del Raspberry. In questo caso l'encoder verrà usato come contatore e potrà essere usato per calcolare la velocità di rotazione del disco.

Il programma HAL sarà il seguente:

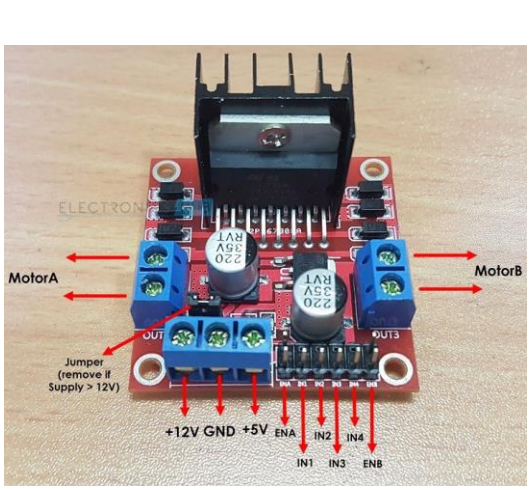
```
loadrt hal_pi_gpio dir=1412864 exclude=536671
loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000
addf hal_pi_gpio.read fast 1
addf hal_pi_gpio.write fast -1

loadrt pwmgen output_type=0
addf pwmgen.update slow
addf pwmgen.make-pulses fast
setp pwmgen.0.pwm-freq 50
setp pwmgen.0.scale 100
setp pwmgen.0.value 50
net moto pwmgen.0.pwm => hal_pi_gpio.pin-33-out
net avvio pwmgen.0.enable

loadrt encoder num_chan=1
addf encoder.update-counters fast
addf encoder.capture-position slow

net enco <= hal_pi_gpio.pin-36-in  => encoder.0.phase-A
setp encoder.0.counter-mode True
```

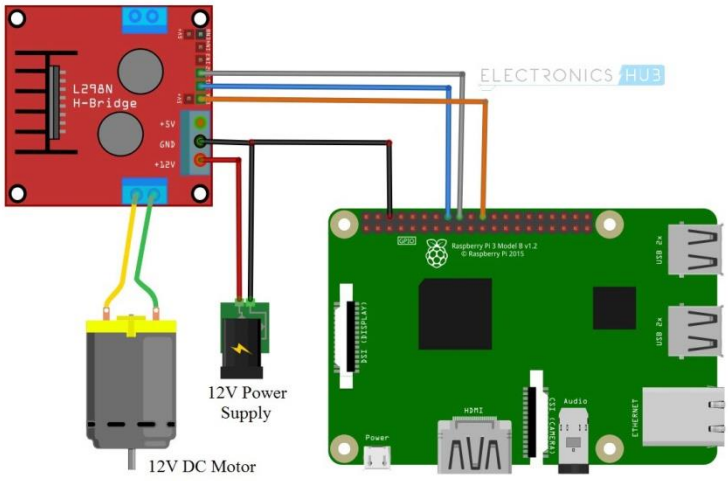
```
setp encoder.0.position-scale 20

sets avvio True
loadusr halmeter
start
```

Per utilizzare l'encoder come contatore è necessario definirne la modalità operativa (`setp encoder.0.counter-mode True`) e definire il numero di fori presenti sul disco (`setp encoder.0.position-scale 20`). Fatto ciò, il parametro *encoder.0.velocity-rpm* darà direttamente il valore dei giri al minuto (*rpm = rounds per minute*). Variando il valore di *pwmgen.0.value (duty-cycle)* varierà la velocità del motore. Bisogna notare però che, in relazione all'inerzia del motore, i valori molto bassi di *pwmgen.0.value* possono non produrre alcun movimento. Può essere utile visualizzare tramite *halscope* la variazione della velocità al variare del duty-cycle. Come si può facilmente intuire, la velocità potrebbe quindi essere regolata su un valore prefissato modificando automaticamente il valore del *duty-cycle* in funzione della velocità misurata, cosa che tuttavia esula dal nostro ambito.

Uso di *pwmgen* per il controllo di un motore DC tramite ponte H.

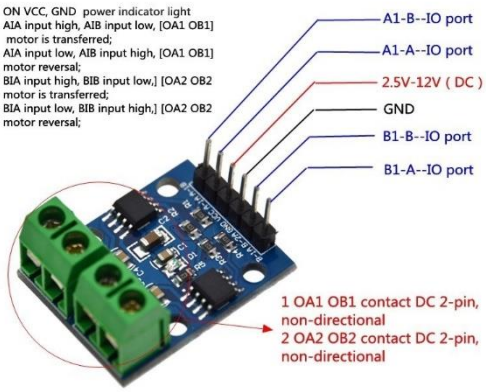
Come detto, tramite la modalità PWM possiamo regolare la velocità di rotazione del motore DC ma non possiamo cambiarne il verso di rotazione perché con lo schema usato non è possibile invertire la polarità dei pin. Per fare ciò è quindi necessario l'uso di un cosiddetto ponte H e l'uso di due pin digitali del Raspberry. Se si usa il modulo L298N, lo schema di collegamento è visibile in figura. Con questa tipologia di scheda è possibile pilotare anche motori con voltaggio diverso da 5V e potenza maggiore di quella fornibile dal Raspberry.





(https://www.electronicshub.org/raspberry-pi-l298n-interface-tutorial-control-dc-motor-l298n-raspberry-pi/)

In questo caso possiamo utilizzare ancora il modulo *pwmgen* con l'opzione di uscita 2 (vedi *Hal Tutorial*). Con questa opzione vengono resi disponibili 2 pin (*pwmgen.0.down* e *pwmgen.0.up*) che collegati agli input I1 e I2 del driver L298N piloteranno il motore in verso e velocità. Per comodità in questo esempio utilizzeremo i pin 38 e 40 del connettore GPIO del Raspberry.



Se invece si utilizzano motori di tipologia analoga a quello con il motoriduttore, che hanno una carico compatibile con quello fornibile dal Raspberry, è possibile usare anche un'altra versione del ponte H (figura a fianco). In questo caso l'alimentazione a 5V va sui due piedini centrali, mentre i pin 38 e 40 andranno collegati rispettivamente ai pin A1-A ed A1-B.

L'uso della opzione 2 del modulo pwmgen mette a disposizione i seguenti pin:

```
halcmd: show pin
Component Pins:
Owner   Type  Dir         Value  Name
……
     4  s32   OUT             0  hal_pi_gpio.read.time
     4  s32   OUT             0  hal_pi_gpio.write.time
    12  float OUT             0  pwmgen.0.curr-dc
    12  bit   I/O         FALSE  pwmgen.0.dither-pwm
    12  bit   OUT         FALSE  pwmgen.0.down
    12  bit   IN          FALSE  pwmgen.0.enable
    12  float I/O             1  pwmgen.0.max-dc
    12  float I/O             0  pwmgen.0.min-dc
    12  float I/O             0  pwmgen.0.offset
    12  float I/O            50  pwmgen.0.pwm-freq
    12  float I/O           100  pwmgen.0.scale
    12  bit   OUT         FALSE  pwmgen.0.up
    12  float IN             50  pwmgen.0.value
    12  s32   OUT             0  pwmgen.make-pulses.time
    12  s32   OUT             0  pwmgen.update.time
     9  s32   OUT             0  slow.time
```

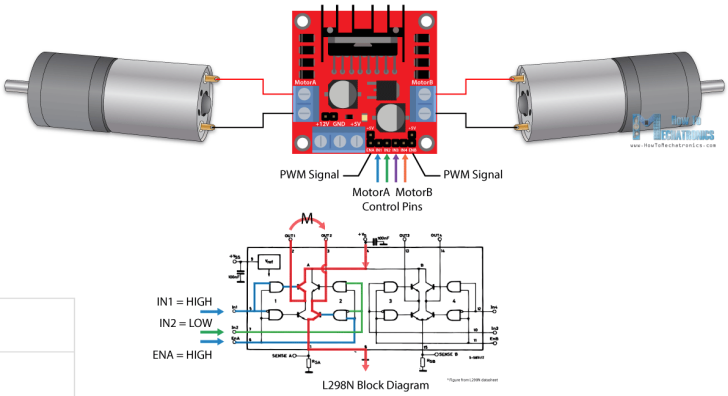
Pertanto il programma HAL sarà il seguente:

```
loadrt hal_pi_gpio dir=1412864 exclude=536671
loadrt threads name1=fast period1=100000 name2=slow period2=1000000
addf hal_pi_gpio.read fast 1
addf hal_pi_gpio.write fast -1
loadrt pwmgen output_type=2
addf pwmgen.update slow
addf pwmgen.make-pulses fast
setp pwmgen.0.pwm-freq 50
setp pwmgen.0.scale 100
setp pwmgen.0.value 20
net up pwmgen.0.up => hal_pi_gpio.pin-38-out
net down pwmgen.0.down => hal_pi_gpio.pin-40-out
net avvio pwmgen.0.enable
sets avvio True
start
```

Variando il valore di pwmgen.0.value tra -100 e 100 cambierà corrispondentemente la velocità e il verso di rotazione, ma nel caso di utilizzazione del motoriduttore, che in genere funziona a circa 6V, il valore da dare a pwmgen.0.value dovrebbe variare tra -50 e 50.

(N.B. In relazione all'inerzia dei motori, a volte valori molto bassi di *pwmgen.0.value* non producono movimento. Rispetto alla figura riportata sopra il filo arancio, che gestirebbe il PWM, in questo caso risulta non necessario).

I due driver utilizzati consentono di gestire 2 motori contemporaneamente secondo lo schema riportato in figura. Ciò consente ad un veicolo dotato di due motori paralleli di andare avanti, indietro, a destra, a sinistra, freno e folle seguendo lo schema riportato in tabella.

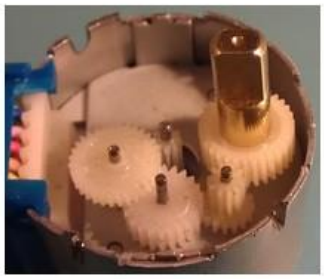


Enable Motor	High – Enable	Low – Disable
Direction 1	IN1 – High	IN2 – Low
Direction 2	IN1 – Low	IN2 – High
Coasting	IN1 – Low	IN2 – Low
Break	IN1 – High	IN2 – High

Il modulo stepgen

Halrun è dotato di un modulo di gestione dei motori stepper molto utile e variegato. Il caricamento del modulo prevede, oltre alla indicazione del tipo di motore stepper utilizzato, anche il tipo di controllo che può essere realizzato (in posizione o in velocità).

Prendiamo il caso di un motore stepper tipo questo in figura. E' un piccolo motore stepper unipolare con motoriduttore alimentabile a 5V, con 2048 step per giro (https://www.ne555.it/motore-stepper-28byj-48/) gestito da un modulino basato sull'integrato ULN2003. Presenta 5 fili, 4 dei quali sono le fasi (A, B, C e D) e il quinto è il terminale comune (GND). Per il suo movimento bisognerà attivare in modo cadenzato le 4 fasi, cosi da produrre la rotazione. Questo tipo di motore è indicato nel modulo *stepgen* come tipo 5 (vedi "*Hal tutorial*", pg.46). In base a queste indicazioni e volendo gestire il motore in velocità bisognerà utilizzare i seguenti comandi:



```
loadrt hal_pi_gpio dir=15314952 exclude=51793892
loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000
loadrt stepgen step_type=5 ctrl_type=v
```

Per questo tipo di motori sono necessari 4 pin di uscita, ognuno per ogni fase.

```
net a stepgen.0.phase-A => hal_pi_gpio.pin-32-out
net b stepgen.0.phase-B => hal_pi_gpio.pin-33-out
net c stepgen.0.phase-C => hal_pi_gpio.pin-38-out
```

```
net d stepgen.0.phase-D => hal_pi_gpio.pin-40-out
```

E' necessario collegare i threads *fast* e *slow* alle funzioni utilizzate dal modulo,

```
addf stepgen.update-freq slow
addf stepgen.make-pulses fast
addf hal_pi_gpio.write fast 1
```

e infine definire i parametri del motore

```
setp stepgen.0.position-scale 2048
setp stepgen.0.enable 1
net speed stepgen.0.velocity-cmd
sets speed 0.25 # 4 secondi per un giro
start
```

Il motore non è molto veloce, quindi *speed*=0.25 è il valore più alto che si può utilizzare partendo da fermo. Valori maggiori possono essere raggiunti aumentando gradualmente la velocità. Ovviamente valori negativi comportano l'inversione del moto. E' possibile utilizzare anche il parametro *step_type=9* ma, come si potra' vedere dal grafico corrispondente, è necessario un numero doppio di step per fare un giro e quindi la velocità verrà dimezzata.

Se invece si vuole gestire il motore in posizione, il file HAL avrà questi comandi:

```
loadrt hal_pi_gpio dir=1412864 exclude=536671
loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000

loadrt stepgen step_type=5 ctrl_type=p

net a stepgen.0.phase-A => hal_pi_gpio.pin-32-out
net b stepgen.0.phase-B => hal_pi_gpio.pin-33-out
net c stepgen.0.phase-C => hal_pi_gpio.pin-38-out
net d stepgen.0.phase-D => hal_pi_gpio.pin-40-out

addf stepgen.update-freq slow
addf stepgen.make-pulses fast
addf hal_pi_gpio.write fast 1

setp stepgen.0.position-scale 2048
setp stepgen.0.maxvel 0.2
setp stepgen.0.enable 1
net pos stepgen.0.position-cmd
start
```

In questo caso, variando il valore del segnale *pos,* varierà la posizione del motore. In particolare il valore di *pos=1* indicherà il giro completo.

Anche in questo caso è possibile utilizzare anche il parametro *step_type=9* ma, aumentando così la risoluzione, sarà necessario un numero doppio di step per fare un giro. Pertanto il valore di *stepgen.0.position-scale* dovrà essere 4096 per avere un giro completo del motore.

Controllo di motori stepper bipolari.

I motori stepper più utilizzati sono quelli bipolari come questo in figura. Per la loro più efficace utilizzazione sono disponibili dei driver come l' A4988 o il DRV8825 di cui abbiamo già parlato in precedenza. Questi driver, oltre ad aggiungere la possibilità di gestire il micropasso, cioè una frazione del passo costruttivo del motore, consentono il loro controllo con soli 3 pin digitali: STEP che fa avanzare il motore di un passo (o micropasso se attivo),

DIR che pilota la direzione e ENABLE che abilita il movimento del motore (se il driver lo consente). Inoltre consentono di alimentare il motore con un voltaggio compreso tra circa 8-24 V e fino a oltre 2A. Per l'interfacciamento con il Raspberry è comodo utilizzare una shield come quella riportata in figura, che gestisce il micropasso tramite i dip-switch.

Per l'uso con Halrun, il modulo *stepgen* prima visto, può essere utilizzato specificando lo *step_type* con il parametro 0.

```
loadrt hal_pi_gpio dir=15314952 exclude=51793892
loadrt stepgen step_type=0 ctrl_type=v
loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000

addf hal_pi_gpio.read fast 1
addf hal_pi_gpio.write fast -1
addf stepgen.make-pulses fast
```

Esiste adesso una funzione (*stepgen.capture-position*) che risulta molto utile per la gestione del motore, poiché si occupa di gestire le scale e tenere conto della posizione via via raggiunta.

```
halcmd: show funct
Exported Functions:
Owner   CodeAddr  Arg       FP   Users  Name
 00004  76f9fba4  00000000  NO       1   hal_pi_gpio.read
 00004  76f9fb08  00000000  NO       1   hal_pi_gpio.write
 00007  76374e48  76388150  YES      0   stepgen.capture-position
 00007  76374bb0  76388150  NO       1   stepgen.make-pulses
 00007  76374f74  76388150  YES      0   stepgen.update-freq
```

Poiché le funzioni *stepgen.capture-position* e *stepgen.update-freq* utilizzano il Floating Point, queste dovranno essere collegate al thread *slow*.

```
addf stepgen.capture-position slow
addf stepgen.update-freq slow
```

Se si utilizza il motore NEMA17, che ha una risoluzione di 200 passi al giro, e con il microstep ad 1/32 (a causa dei tre dip-switch attivi) per fare un giro serviranno 6400 step.

```
setp stepgen.0.position-scale 6400 # 200 passi a giro * 32 microstep
setp stepgen.0.enable 1
```

Con riferimento ai pin STEP=19, DIR=23 ed ENA=11, possiamo gestire i parametri del modulo *stepgen* con i seguenti comandi

```
net step stepgen.0.step => hal_pi_gpio.pin-19-out
net dir  stepgen.0.dir  => hal_pi_gpio.pin-23-out
setp hal_pi_gpio.pin-11-out 0
```

Per concludere aggiungiamo i comandi di gestione della velocità

```
net X-vel => stepgen.0.velocity-cmd
sets X-vel 0.5
loadusr halmeter
start
```

Variando X-vel (espresso in giri al secondo), si varierà la velocità. Per effetto della elevata risoluzione, i valori di velocità saranno relativamente bassi.

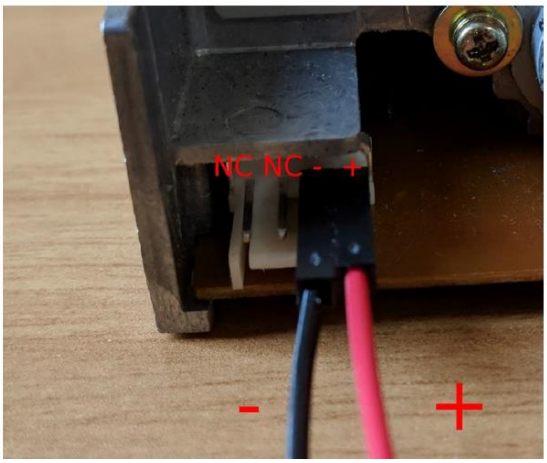
Anche in questo caso si può pilotare il motore in posizione. Il programma, con le opportune modifiche, sarà questo:

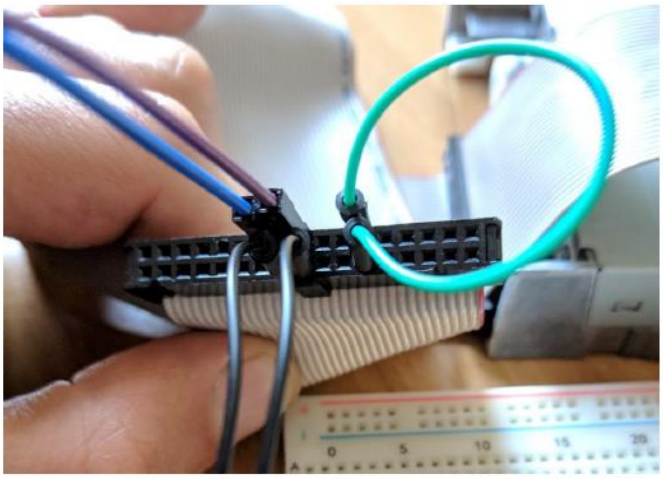
```
loadrt hal_pi_gpio dir=1412864 exclude=536671
loadrt stepgen step_type=0 ctrl_type=p
loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000

addf hal_pi_gpio.read fast 1
addf hal_pi_gpio.write fast -1
addf stepgen.make-pulses fast

addf stepgen.capture-position slow
addf stepgen.update-freq slow

setp stepgen.0.position-scale 6400 # 200 passi a giro * 32 microstep
net ena stepgen.0.enable =>  hal_pi_gpio.pin-11-out

net step stepgen.0.step => hal_pi_gpio.pin-19-out
net dir  stepgen.0.dir  => hal_pi_gpio.pin-23-out

net Z-pos => stepgen.0.position-cmd
sets ena 0
sets Z-pos 0
loadusr halmeter

start
```

In questo modo, tutti i movimenti saranno riferiti in modo assoluto rispetto alla posizione *Z-pos=0*. Quindi, partendo da *Z-pos =0*, il comando *sets Z-pos=1* porterà il motore a compiere 1 giro in avanti e se, da questa posizione, si effettua il comando *sets Z-pos=-1* il motore compierà due giri indietro.

Questo programma, opportunamente modificato, può anche essere usato per pilotare un vecchio floppy direttamente in modalità *step/dir* (vedi anche il file pdf “Interfacciamento floppy”).

L’alimentazione del floppy (5V) può essere ricavata, ad esempio, dalla porta USB del computer, da un alimentatore per smartphone oppure dal connettore 5v del





Poi sul connettore del floppy bisognerà collegare il pin 11 e 12 tra loro, il pin 18 (DIR) e il pin 20 (STEP) con due pin del Raspberry. Analogamente a quanto fatto quando abbiamo usato il motore stepper unipolare, utilizziamo rispettivamente i pin 23 e 19 sul Raspberry.

Tenendo conto che l'escursione completa del floppy richiede circa 100 passi, avremo:

```
loadrt hal_pi_gpio dir=1412864 exclude=536671
loadrt stepgen step_type=0 ctrl_type=p
loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000

addf hal_pi_gpio.write fast
addf stepgen.make-pulses fast

addf stepgen.capture-position slow
addf stepgen.update-freq slow

setp stepgen.0.position-scale 1 # escursione 100 passi
setp stepgen.0.maxvel 100

net step stepgen.0.step => hal_pi_gpio.pin-19-out
net dir  stepgen.0.dir  => hal_pi_gpio.pin-23-out
net Z-pos => stepgen.0.position-cmd
sets Z-pos 50

setp stepgen.0.enable True

loadusr halmeter

start
```

Variando Z-pos tra 0 e 100, possiamo muovere la testina del floppy lungo tutto il percorso.

Il modulo encoder usato come potenziometro

Il floppy ha alcuni limiti. Non può andare a valori negativi di Z-pos e oltre 100. Inoltre la risoluzione modesta impedisce l'uso di valori frazionari di Z-pos. Per tenere conto di ciò possiamo utilizzare una tra le numerose funzioni che mette a disposizione HAL. Una di queste è la *knob2float* che si occupa appunto di convertire gli impulsi di un encoder in un numero in virgola mobile all'interno di un range prefissato (http://linuxcnc.org/docs/devel/html/man/man9/knob2float.9.html). Per il suo utilizzo aggiungiamo i comandi seguenti al programma visto in precedenza:

```
loadrt knob2float
addf knob2float.0 slow
```

```
net conta encoder.0.counts => knob2float.0.counts
setp knob2float.0.scale 0.5
setp knob2float.0.enable True
setp knob2float.0.min-out 0
setp knob2float.0.max-out 100
net pos <= knob2float.0.out
```

In questo modo, ad ogni scatto dell'encoder, il valore di *pos* varia di 1 e può prendere soltanto i valori compresi tra 0 e 100.

Chiudiamo infine il programma con le istruzioni che, come al solito, consentono l'avvio e la visualizzazione del funzionamento:

```
loadusr halmeter
start
```

Un esempio combinato: encoder e floppy

Vediamo ora come assemblare gli ultimi due esempi. Supponiamo di dovere pilotare il movimento del floppy attraverso l'encoder tra il suo limite minimo (0) e quello massimo (100). Inoltre, tramite il pulsante, vogliamo variare il numero di step (1 o 2) per ogni scatto ed evidenziarne la scelta con un led. Per gestire questa alternativa risulta molto utile la funzione *mux2* (http://linuxcnc.org/docs/html/man/man9/mux2.9.html) che seleziona uno tra due valori in ingresso sulla base di un parametro 0/1. Seguendo quanto visto in precedenza il programma HAL sarà:

```
loadrt hal_pi_gpio dir=1412864 exclude=536671
loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000
loadrt encoder num_chan=1
loadrt stepgen step_type=0 ctrl_type=p
loadrt toggle
loadrt mux2
loadrt knob2float

addf hal_pi_gpio.read fast 1
addf hal_pi_gpio.write fast -1
addf stepgen.make-pulses fast
addf encoder.update-counters fast
addf encoder.capture-position slow
addf toggle.0 fast
addf mux2.0 slow
addf knob2float.0 slow

addf stepgen.capture-position slow
addf stepgen.update-freq slow

net Fase_A encoder.0.phase-A <= hal_pi_gpio.pin-16-in
net Fase_B encoder.0.phase-B <= hal_pi_gpio.pin-18-in
net scatto toggle.0.in <= hal_pi_gpio.pin-26-in
net led toggle.0.out => hal_pi_gpio.pin-40-out mux2.0.sel
net scala mux2.0.out => knob2float.0.scale
net conta encoder.0.counts knob2float.0.counts

setp mux2.0.in0 0.25
setp mux2.0.in1 0.5

setp encoder.0.counter-mode False
setp knob2float.0.enable True
setp knob2float.0.min-out 0
```

```
setp knob2float.0.max-out 100
setp stepgen.0.position-scale 1
setp stepgen.0.maxvel 100
setp stepgen.0.enable 1

net step stepgen.0.step => hal_pi_gpio.pin-19-out
net dir  stepgen.0.dir  => hal_pi_gpio.pin-23-out
net Z-pos => stepgen.0.position-cmd  <= knob2float.0.out
loadusr halmeter
start
```

Il modulo hal_input

Il Raspberry, al contrario di Arduino per esempio, non ha sui pin della GPIO pin in grado di gestire segnali analogici sia in input che in output. Per quello che riguarda l'output si può cercare a volte di sopperire a questa mancanza tramite l'uso di PWM (per esempio nel controllo dei motori DC) ma per quello che riguarda l'input la cosa non è di immediata risoluzione.

Un interessante modulo che consente di avere un gran numero di controlli è il modulo *hal_input*. Questo modulo permette di dialogare con numerose periferiche di input, quali tastiere, mouse, etc. ma risulta di grande utilità pratica con dispositivi quali i joypad che, semplicemente e basso prezzo, consentono di dotare il Raspberry di un certo numero di ingressi digitali e analogici. In particolare si possono usare i 2 joystick che sono presenti sul joypad per simulare 4 ingressi analogici (X, Y, Z e RZ).

Vediamo come si può fare.

Per prima cosa, dopo avere collegato sulla porta USB il joypad, da terminale digitiamo il comando:

```
cat /proc/bus/input/devices
```

Dovrebbe apparire una schermata come questa:

```
I: Bus=0003 Vendor=046d Product=c219 Version=0111
N: Name="Logitech Logitech Cordless RumblePad 2"
P: Phys=usb-3f980000.usb-1.3/input0
S:              Sysfs=/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.3/1-
1.3:1.0/0003:046D:C219.0003/input/input1
U: Uniq=
H: Handlers=js0 event0
B: PROP=0
B: EV=20001b
B: KEY=fff0000 0 0 0 0 0 0 0 0 0
B: ABS=30027
B: MSC=10
B: FF=1 7030000 0 0
```

Questa schermata conferma che il dispositivo di output è stato riconosciuto (in particolare, in questo caso, si tratta di un "*Logitech Logitech Cordless RumblePad 2*", che utilizza un handlers n. 0.

Sulla base di queste informazioni il comando da dare ad *halrun* può essere uno dei due seguenti:

```
loadusr -W hal_input -KRAL Rumble
```

oppure

```
loadusr -W hal_input -KRAL :0
```

Col primo comando, il dispositivo di input viene riconosciuto tramite una parte dell'identificativo Name="*Logitech Logitech Cordless RumblePad 2*". Con il secondo comando il dispositivo di input viene riconosciuto tramite il numero di handler. Se il dispositivo di input è l'unico, è sufficiente usare la seconda opzione, che non può avere equivoci. In caso contrario è meglio usare la prima opzione.

Quindi digitiamo: `halrun –I`

Apparirà il prompt di halcmd.

Riferendosi alla documentazione del modulo *hal_input* di LinuxCNC (http://linuxcnc.org/docs/html/man/man1/hal_input.1.html) l' opzione –W impone l'attesa del caricamento del modulo mentre –KRAL serve a selezionare i vari tipi di input (K=key, R=relative A=absolute L=led)

```
loadusr -W hal_input -KRAL :0
```

Fatto ciò, digitiamo `show pin`.

Verranno fuori una sfilza di possibili input tipo questi *(N.B. I nomi dei bottoni e dei cursori potrebbero variare in relazione al tipo di dispositivo)*:

```
halcmd: show pin
Component Pins:
Owner   Type  Dir         Value  Name
     4  s32   OUT             0  input.0.abs-hat0x-counts
     4  s32   IN              0  input.0.abs-hat0x-flat
     4  s32   IN              0  input.0.abs-hat0x-fuzz
     4  bit   OUT         FALSE  input.0.abs-hat0x-is-neg
     4  bit   OUT         FALSE  input.0.abs-hat0x-is-pos
     4  float IN              0  input.0.abs-hat0x-offset
     4  float OUT             0  input.0.abs-hat0x-position
     4  float IN              1  input.0.abs-hat0x-scale
     4  s32   OUT             0  input.0.abs-hat0y-counts
     4  s32   IN              0  input.0.abs-hat0y-flat
     4  s32   IN              0  input.0.abs-hat0y-fuzz
     4  bit   OUT         FALSE  input.0.abs-hat0y-is-neg
     4  bit   OUT         FALSE  input.0.abs-hat0y-is-pos
     4  float IN              0  input.0.abs-hat0y-offset
     4  float OUT             0  input.0.abs-hat0y-position
     4  float IN              1  input.0.abs-hat0y-scale
     4  s32   OUT             0  input.0.abs-rx-counts
     4  s32   IN            128  input.0.abs-rx-flat
     4  s32   IN             16  input.0.abs-rx-fuzz
     4  bit   OUT         FALSE  input.0.abs-rx-is-neg
     4  bit   OUT         FALSE  input.0.abs-rx-is-pos
     4  float IN           -0.5  input.0.abs-rx-offset
     4  float OUT  1.525902e-05  input.0.abs-rx-position
     4  float IN        32767.5  input.0.abs-rx-scale
     4  s32   OUT             0  input.0.abs-ry-counts
     4  s32   IN            128  input.0.abs-ry-flat
     4  s32   IN             16  input.0.abs-ry-fuzz
     4  bit   OUT         FALSE  input.0.abs-ry-is-neg
     4  bit   OUT         FALSE  input.0.abs-ry-is-pos
     4  float IN           -0.5  input.0.abs-ry-offset
     4  float OUT  1.525902e-05  input.0.abs-ry-position
```

```
4  float IN        32767.5  input.0.abs-ry-scale
4  s32   OUT             0  input.0.abs-rz-counts
4  s32   IN              0  input.0.abs-rz-flat
4  s32   IN              0  input.0.abs-rz-fuzz
4  bit   OUT          TRUE  input.0.abs-rz-is-neg
4  bit   OUT         FALSE  input.0.abs-rz-is-pos
4  float IN          127.5  input.0.abs-rz-offset
4  float OUT            -1  input.0.abs-rz-position
4  float IN          127.5  input.0.abs-rz-scale
4  s32   OUT             0  input.0.abs-x-counts
4  s32   IN            128  input.0.abs-x-flat
4  s32   IN             16  input.0.abs-x-fuzz
4  bit   OUT         FALSE  input.0.abs-x-is-neg
4  bit   OUT         FALSE  input.0.abs-x-is-pos
4  float IN           -0.5  input.0.abs-x-offset
4  float OUT  1.525902e-05  input.0.abs-x-position
4  float IN        32767.5  input.0.abs-x-scale
4  s32   OUT             0  input.0.abs-y-counts
4  s32   IN            128  input.0.abs-y-flat
4  s32   IN             16  input.0.abs-y-fuzz
4  bit   OUT         FALSE  input.0.abs-y-is-neg
4  bit   OUT         FALSE  input.0.abs-y-is-pos
4  float IN           -0.5  input.0.abs-y-offset
4  float OUT  1.525902e-05  input.0.abs-y-position
4  float IN        32767.5  input.0.abs-y-scale
4  s32   OUT             0  input.0.abs-z-counts
4  s32   IN              0  input.0.abs-z-flat
4  s32   IN              0  input.0.abs-z-fuzz
4  bit   OUT          TRUE  input.0.abs-z-is-neg
4  bit   OUT         FALSE  input.0.abs-z-is-pos
4  float IN          127.5  input.0.abs-z-offset
4  float OUT            -1  input.0.abs-z-position
4  float IN          127.5  input.0.abs-z-scale
4  bit   OUT         FALSE  input.0.btn-a
4  bit   OUT          TRUE  input.0.btn-a-not
4  bit   OUT         FALSE  input.0.btn-b
4  bit   OUT          TRUE  input.0.btn-b-not
4  bit   OUT         FALSE  input.0.btn-mode
4  bit   OUT          TRUE  input.0.btn-mode-not
4  bit   OUT         FALSE  input.0.btn-select
4  bit   OUT          TRUE  input.0.btn-select-not
4  bit   OUT         FALSE  input.0.btn-start
4  bit   OUT          TRUE  input.0.btn-start-not
4  bit   OUT         FALSE  input.0.btn-thumbl
4  bit   OUT          TRUE  input.0.btn-thumbl-not
4  bit   OUT         FALSE  input.0.btn-thumbr
4  bit   OUT          TRUE  input.0.btn-thumbr-not
4  bit   OUT         FALSE  input.0.btn-tl
4  bit   OUT          TRUE  input.0.btn-tl-not
4  bit   OUT         FALSE  input.0.btn-tr
4  bit   OUT          TRUE  input.0.btn-tr-not
4  bit   OUT         FALSE  input.0.btn-x
4  bit   OUT          TRUE  input.0.btn-x-not
4  bit   OUT         FALSE  input.0.btn-y
4  bit   OUT          TRUE  input.0.btn-y-not
```

Tra questi distinguiamo quelli con *abs* che si riferiscono ai cursori analogici e quelli con *btn* che si riferiscono ai pulsanti. Ad esempio *input.0.abs-x-position* assumerà valori compresi tra -1 e 1 quando il cursore orizzontale del joystick di sinistra va da sinistra a destra. Analogamente *input.0.abs-y-position* assumerà valori compresi tra -1 e 1 quando il cursore verticale del joystick di sinistra va da sotto a sopra. Invece *input.0.abs-z-position* e

input.0.abs-rz-position si riferiscono al joystick di destra. Per verificare ciò possiamo caricare *halmeter*. Selezionando i sopracitati segnali, vedremo il loro variare con il movimento dei joystick.

Adesso riprendiamo l'esempio del motore stepper pilotato in posizione. Aggiungiamo la riga che carica *hal_input* e modifichiamo la seguente

```
net Z-pos => stepgen.0.position-cmd <= input.0.abs-x-position
```

Adesso, quando il programma sarà attivo, sarà possibile mandare avanti e indietro il motore di un giro agendo direttamente sul joystick di sinistra. Analogamente possiamo fare per la velocità modificando la riga

```
net X-vel => stepgen.0.velocity-cmd <= input.0.abs-z-position
```

Adesso, quando il programma sarà attivo, sarà possibile mandare avanti e indietro con velocità via via crescenti o decrescenti il motore agendo direttamente sull'altro joystick.

E per accendere il led? Se utilizziamo il tasto *a*, basta semplicemente aggiungere

```
net led input.0.btn-a => hal_pi_gpio.pin-29-out
```

Il led rimarrà acceso fintanto che il tasto rimane premuto.

Pannello di controllo joystick

Per provare e verificare i vari comandi, possiamo preparare un pannello utilizzando quanto riportato sul capitolo su *pyVCP* di "*Hal Tutorial*"
Creiamo un file di nome *joypanel.xml* e cominciamo col definire un contenitore orizzontale con un bordo rialzato largo 6 pixel:

```
<!-- Test panel for the hal_pi_gpio for Joystick Logitech Rumble -->
<pyvcp>
<hbox>
  <relief>RIDGE</relief>
    <bd>6</bd>
```

Per visualizzare i movimenti dei due joystick (orizzontale e verticale) definiamo 4 quadranti grandi 250 pixel con valori numerici compresi tra -1 e 1 *denominati asseX, asseY, asseZ e asseA*, con tacche piccole ogni 0.1 e grandi ogni 0.25. Quindi chiudiamo il contenitore orizzontale.

```
      <meter>
            <halpin>"assex"</halpin>
            <text>"X"</text>
            <subtext>"asse X"</subtext>
            <size>250</size>
            <min_>-1</min_>
            <max_>1</max_>
            <majorscale>0.25</majorscale>
            <minorscale>0.1</minorscale>
     </meter>
<meter>
      <halpin>"assey"</halpin>
```

```
      <text>"Y"</text>
      <subtext>"asse Y"</subtext>
      <size>250</size>
      <min_>-1</min_>
      <max_>1</max_>
      <majorscale>0.25</majorscale>
      <minorscale>0.1</minorscale>
</meter>

<meter>
      <halpin>"assez"</halpin>
      <text>"Z"</text>
      <subtext>"asse Z"</subtext>
      <size>250</size>
      <min_>-1</min_>
      <max_>1</max_>
      <majorscale>0.25</majorscale>
      <minorscale>0.1</minorscale>
</meter>

<meter>
      <halpin>"assea"</halpin>
      <text>"A"</text>
      <subtext>"asse A"</subtext>
      <size>250</size>
      <min_>-1</min_>
      <max_>1</max_>
      <majorscale>0.25</majorscale>
      <minorscale>0.1</minorscale>
</meter>

</hbox>
```

Per visualizzare lo stato dei bottoni definiamo un altro box orizzontale dove inseriamo 4 led rotondi grandi 40 pixel chiamati *led-01, led-02, …, led-04* a cui assegniamo quattro colori diversi *(blue, green, red, yellow)* che rifletteranno lo stato dei bottoni superiori, e successivamente inseriamo 4 led rettangolari anch'essi di gradezza 40x40 chiamati *led-05, led-06 led-07, led-08* a cui assegniamo altri quattro colori diversi (*white, cyan, purple, gray)* quando sono accesi. Tutti i led saranno di colore *black* quando sono spenti.

```
<hbox>
   <led>
                  <halpin>"led-01"</halpin>
                  <size>40</size>
                  <on_color>"blue"</on_color>
                  <off_color>"black"</off_color>
   </led>
   <led>
                  <halpin>"led-02"</halpin>
                  <size>40</size>
                  <on_color>"green"</on_color>
                  <off_color>"black"</off_color>
   </led>
   <led>
                  <halpin>"led-03"</halpin>
                  <size>40</size>
                  <on_color>"red"</on_color>
                  <off_color>"black"</off_color>
   </led>
   <led>
                  <halpin>"led-04"</halpin>
                  <size>40</size>
```

```
                <on_color>"yellow"</on_color>
                <off_color>"black"</off_color>
    </led>
    <rectled>
                <halpin>"led-05"</halpin>
                <height>"40"</height>
                <width>"40"</width>
                <on_color>"white"</on_color>
                <off_color>"black"</off_color>
    </rectled>
    <rectled>
                <halpin>"led-06"</halpin>
                <height>"40"</height>
                <width>"40"</width>
                <on_color>"cyan"</on_color>
                <off_color>"black"</off_color>
          </rectled>
    <rectled>
                <halpin>"led-07"</halpin>
                <height>"40"</height>
                <width>"40"</width>
                <on_color>"purple"</on_color>
                <off_color>"black"</off_color>
    </rectled>
    <rectled>
                <halpin>"led-08"</halpin>
                <height>"40"</height >
                <width>"40"</width >
                <on_color>"gray"</on_color>
                <off_color>"black"</off_color>
  </rectled>
```

Infine per visualizzare l'effetto del joystick a croce, definiamo due label e due valori numerici interi (*contax* e *contay*) con segno che verranno incrementati e decrementati, chiudendo infine il contenitore orizzontale e la definizione del pannello

```
    <label>
           <text>"Conta X ="</text>
           <font>("Helvetica", 20)</font>
    </label>
    <s32>
           <halpin>"contax"</halpin>
           <font>("Helvetica",24)</font>
           <format>"6d"</format>
           <width>6</width>
    </s32>
    <label>
           <text>"Conta Y ="</text>
           <font>("Helvetica", 20)</font>
    </label>
    <s32>
           <halpin>"contay"</halpin>
           <font>("Helvetica",24)</font>
           <format>"6d"</format>
           <width>6</width>
      </s32>
</hbox>
</pyvcp>
```

Passiamo alla descrizione del file HAL. Poiché insieme alla visualizzazione del pannello, vogliamo pilotare un led (o un motore DC) con un segnale PWM gestito dal segnale del joystick chiamato *assex*, oltre ai comandi necessari per la gestione del Raspberry, dei threads

e del joystick, abbiamo aggiunto il comando che richiama il pannello *pyVCP* definito nel file *joypanel.xml* .

```
loadrt hal_pi_gpio dir=1412864 exclude=53667
loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000
loadusr -W hal_input -KRAL :0
loadusr -Wn joypanel pyvcp -c joypanel joypanel.xml
```

Quindi carichiamo i moduli che ci possono servire: il *pwmgen* e *updown*

```
loadrt pwmgen output_type=0
loadrt updown count=2
```

Il modulo *pwmgen* ci serve per la gestione della luminosità del led (o la velocità del motore DC) come già visto in precedenza.
Il modulo *updown* *(http://linuxcnc.org/docs/2.7/html/man/man9/updown.9.html)* risulta utile come contatore degli impulsi del cursore a croce. Ce ne servono 2, uno per x e l'altro per y.
Come sempre aggiungiamo le funzioni di aggiornamento dei moduli caricati ai thread *fast* e *slow,* e la definizione dei parametri per il PWM (scala 1 e frequenza 50 Hz)*:*

```
addf hal_pi_gpio.read fast 1
addf hal_pi_gpio.write fast -1
addf pwmgen.make-pulses fast
addf pwmgen.update slow
addf updown.0 slow
addf updown.1 slow

setp pwmgen.0.enable True
setp pwmgen.0.pwm-freq 50
setp pwmgen.0.scale 1
```

Con i comandi seguenti definiamo i collegamenti tra gli input e gli output. I valori provenienti dai joystick vanno ai corrispondenti valori sui *meter* definiti con il pannello *joypanel*. Inoltre il valore *assex* viene anche inviato come input del generatore di PWM, così da pilotare l'uscita sul pin corrispondente. *(N.B. I nomi dei bottoni e dei cursori potrebbero variare in relazione al tipo di dispositivo)*

```
net assex <= input.0.abs-x-position  => joypanel.assex pwmgen.0.value
net assey <= input.0.abs-y-position  => joypanel.assey
net assez <= input.0.abs-z-position  => joypanel.assez
net assea <= input.0.abs-rz-position => joypanel.assea
```

Si passa poi alla connessione fra i led ed i bottoni ed il led del PWM con il pin di uscita

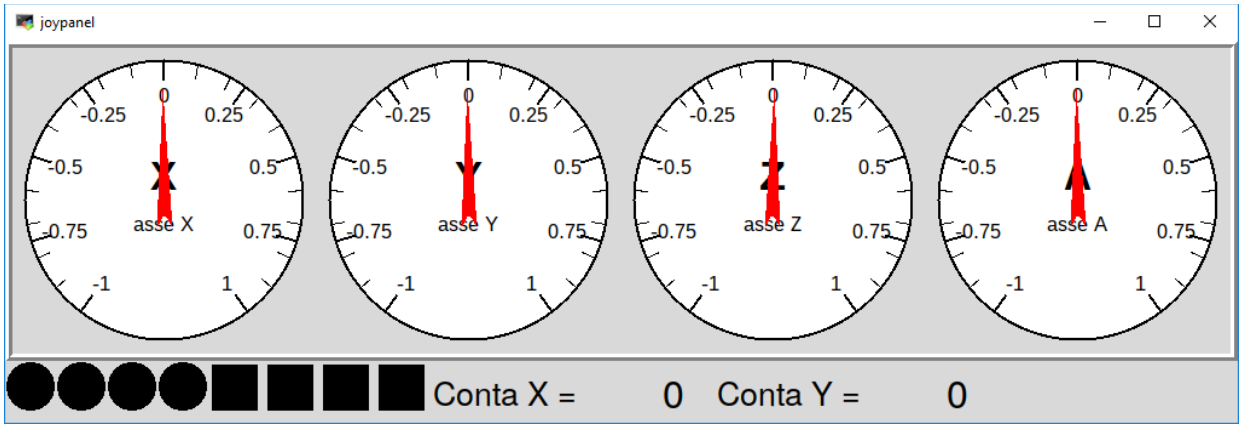
```
net led1 <= input.0.btn-a  => joypanel.led-01
net led2 <= input.0.btn-b  => joypanel.led-02
net led3 <= input.0.btn-c  => joypanel.led-03
net led4 <= input.0.btn-x  => joypanel.led-04
net led5 <= input.0.btn-y  => joypanel.led-05
net led6 <= input.0.btn-z  => joypanel.led-06
net led7 <= input.0.btn-tl => joypanel.led-07
net led8 <= input.0.btn-tr => joypanel.led-08
net led-pwm <= pwmgen.0.pwm => hal_pi_gpio.pin-23-out
```

Con le righe seguenti definiamo il comportamento dei bottoni del cursore a croce. Infatti se il valore è negativo, il contatore viene decrementato, se positivo incrementato. Il valore del contatore (*contax* e *contay*) viene inviato al corrispondente valore sul pannello.

```
net dox updown.0.countdown <= input.0.abs-hat0x-is-neg
net upx updown.0.countup   <= input.0.abs-hat0x-is-pos
net doy updown.1.countdown <= input.0.abs-hat0y-is-neg
net upy updown.1.countup   <= input.0.abs-hat0y-is-pos
net contax updown.0.count  => joypanel.contax
net contay updown.1.count  => joypanel.contay
```

Il programma si conclude con il caricamento del visualizzatore delle variabili e l'avvio:

```
loadusr halmeter
start
```



Il pannello di nome *joypad* apparirà così. Muovendo i due joystick, si aggiorneranno i quadranti e con lo spostamento del primo nella zona positiva andrà via via aumentando la luminosità del led (o la velocità del motore). La pressione di tasti farà accendere i vari led colorati e muovendo il cursore a croce si attiveranno i contatori.