WiringPI : una libreria di interfaccia per il GPIO di Raspberry Pi

Per utilizzare la libreria WiringPi GPIO, è necessario inserire nel programma il suo file di intestazione :

 # include <wiringPi.h>

Potrebbe anche essere necessario aggiungere in fase di copilazione/linking :

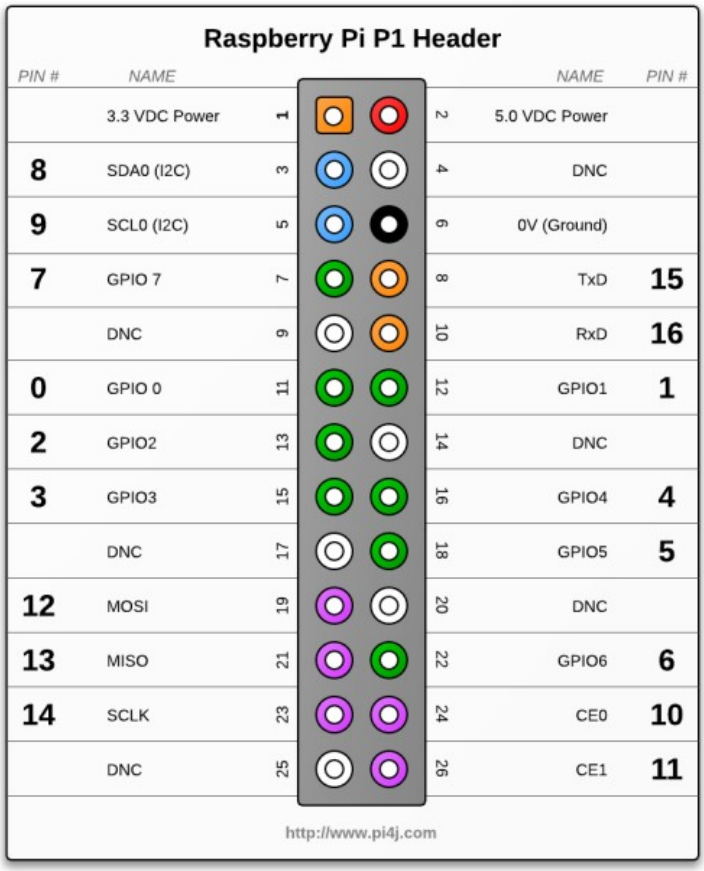
-I/usr/local/include -L/usr/local/lib -lwiringPi

Setup

wiringPi prima del suo utilizzo deve essere inizializzata con una delle seguenti funzioni:

int wiringPiSetup (void) ;
int wiringPiSetupGpio (void) ;
int wiringPiSetupPhys (void) ;
int wiringPiSetupSys (void ;

Nota : In wiringPi rev1 se queste funzioni per un qualsiasi motivo falliscono restituiscono un codice di errore. Nella rev2 le funzioni restituiscono sempre zero. Se si vuole che rev2 si comporti la rev1, bisogna impostare la variabile d'ambiente : WIRINGPI_CODES ad un qualsiasi valore (ha solo bisogno di esserci).



Le differenze tra le funzioni di setup sono :

wiringPiSetup *(void) ;*

Inizializza wiringPi e presuppone che il programma chiamante utilizzerà lo schema di *numerazione dei pin wiringPi* . Questo è uno schema di numerazione semplificato, che fornisce una mappatura tra i numeri da 0 a 16 e i reali numeri dei pin definiti da Broadcom per GPIO . Questa funzione deve essere chiamata con privilegi di root .

wiringPiSetupGpio *(void) ;*

Come la precedente, ma permette ai programmi di utilizzare i *numeri dei pin GPIO Broadcom* direttamente senza remapping . Questa funzione deve essere chiamata con privilegi di root.
Attenzione alla differenza di numerazione fra i pin della rev1 e rev2 di PI.

wiringPiSetupPhys *(void) ;*

Identica alla precedente, ma permette ai programmi di utilizzare *i numeri fisici dei pin sul connettore GPIO* .Come le precedenti questa funzione deve essere chiamata con i privilegi di root .

wiringPiSetupSys *(void) ;*

Iinizializza wiringPi utilizzando l'interfaccia ***/sys/class/GPIO*** anziché accedere direttamente all'hardware. Può esser chiamata senza i privilegi di root e prevede che i pin GPIO siano esportati prima utilizzando il programma GPIO . Numerazione dei pin è quella nativa cioè con la numerazione Broadcom GPIO (come con la wiringPiSetupGpio), in modo da essere consapevoli delle differenze tra le schede di revisione 1 e 2.
Nota : In questa modalità è possibile usare solo i pin che sono stati esportati tramite l'interfaccia /sys/class/GPIO prima di eseguire il programma. È possibile farlo in uno script di shell separata , oppure utilizzando all'interno del programma una *system()* che richiamo il programma ***gpio***.
Quando si inizializza con questa funzione alcune funzioni non hanno alcun effetto perché necessitano dei privilegi root. Se necessario è comunque possibile usare il *system()* per chiamare ***gpio*** per impostare o cambiare la modalità..

Funzioni di base

Queste funzioni lavorano direttamente sul Raspberry Pi e anche con molte espansioni GPIO esterne, infatti non tutte le espansioni supportano tutte le funzioni per esempio il PiFace è preconfigurata per gli ingressi e le uscite fisse, e Raspberry non ha hardware analogico a bordo.

void pinMode (*int pin, int mode*) ;

Imposta la modalità del pin *pin* secondo il parametro *mode* che può assumere i valori INPUT , OUTPUT , PWM_OUTPUT o GPIO_CLOCK . Si noti che solo wiringPi pin 1 (BCM_GPIO 18) supporta l'uscita PWM e solo wiringPi pin 7 (BCM_GPIO 4) supporta le modalità di uscita di clock . Questa funzione non ha alcun effetto in modalità Sys . Se si ha bisogno di cambiare il modo pin , bisogna utilizzare il programma GPIO in uno script prima di lanciare il programma.

void pullUpDnControl (*int pin, int pud*) ;

Imposta *pin* (settato come ingresso) la modalità *pud*, che può assumere i valori :
PUD_OFF (senza pull up/down)
PUD_DOWN (a 0 v)
PUD_UP (a 3.3v)

Il BCM2835 ha i resistori interni sia di pull-down che pull-up (a differenza di Arduino). Il valore della resistenza interna di circa 50KΩ .
Se il Raspberry Pi GPIO è in modalità Sys la funzione non ha alcun effetto sui pin. Se si ha bisogno di attivare un pull-up/pull-down , allora si può fare con il programma GPIO in uno script prima di lanciare il programma.

void digitalWrite (*int pin , int value*) ;
Imposta *pin* (settato come uscita) a *value* che può assume i valori HIGH o LOW (1 oppure 0). Da notare che qualsiasi valore diverso da zero viene trattato come HIGH, lo 0 è LOW.

void pwmWrite (*int pin , int value*) ;
Scrive *value* nel registro PWM per il *pin* dato. *value* può variare da 0 a 1024, altri dispositivi PWM possono avere diverse gamme di PWM. Il pin PWM on-board di Raspberry Pi ha, è il pin 1 (BMC_GPIO 18 , Phys 12). Questa funzione in modalità Sys non è in grado di controllare il PWM on-board del PI.

int digitalRead (*int pin*) ;
Questa funzione restituisce il valore letto su *pin*. Sarà HIGH o LOW a seconda del livello logico al pin .

analogRead (*int pin*) ;
Restituisce il valore letto sul *pin* analogico di ingresso in dotazione . Per abilitare questa funzione bisogna utilizzare moduli analogici aggiuntivi come il Gertboard , scheda analogica quick2Wire , ecc

analogWrite (*int pin , int value*) ;
Scrive il valore su *pin* analogico in dotazione . Questa funzione si utilizza in presenza di moduli aggiuntivi esterni.

Funzioni Specifiche della libreria WiringPi

Queste funzioni non fanno parte del set di funzioni wiringPi, ma agiscono specificamente sull' hardware Raspberry Pi, comunque moduli aggiuntivi esterni possono utilizzare alcune di queste funzioni.

void digitalWriteByte (*int value*) ;
Scrive *value* sui primi 8 pin di GPIO . E ' un modo rapido per impostare tutti gli 8 bit contemporaneamente.

pwmSetMode (*int mode*) ;
Il generatore PWM può funzionare in due modi "balanced" e "mark:space". Il primo modo è quello di default. È possibile passare da una modalità all'altra fornendo il parametro *mode*:
PWM_MODE_BAL per passare alla modalità balanced
PWM_MODE_MS per passare alla modalità mark:space.

pwmSetRange (*unsigned int range*) ;
Imposta l'intervallo nel generatore PWM (il limite superiore). Il valore di default è 1024 .

pwmSetClock (*int divisor*) ;

Imposta il divisore per il clock PWM .

Nota : Le funzioni di controllo PWM non possono essere utilizzati in modalità Sys . Per approfondire la conoscenza del sistema PWM , leggere il manuale delle periferiche ARM Broadcom .

piBoardRev (*void*) ;
Restituisce 1 o 2 cioè la revisione del Raspberry Pi in uso. I pin fra le due revisioni cambiano per cui sarà necessario utilizzare questa funzione se si utilizza la numerazione dei pin BCM_GPIO.

wpiPinToGpio (*int wPiPin*) ;
Restituisce il numero di pin BCM_GPIO che corrisponde *wPiPin* (numero pin wiringPi).

physPinToGpio (*int physPin*) ;
Restituisce il numero di pin BCM_GPIO che corrisponde a *physPin* (numero pin fisico).

setPadDrive (*int gruppo , int value*) ;
Forza l'impostazione dei driver pad per un particolare gruppo di pin . Ci sono 3 gruppi di pin e il valore va da 0 a 7.

Timing

unsigned int Millis (*void*)
Restituisce un numero senza segno di 32 bit, che rappresenta il numero di millisecondi trascorsi dal momento che il programma ha chiamato una delle funzioni 4 funzioni di inizializzazione di wiringPi. Si azzera dopo 49 giorni circa.

unsigned int micros (*void*)
Restituisce un numero senza segno di 32 bit, che rappresenta il numero di microsecondi trascorsi dal momento che il programma ha chiamato una delle funzioni 4 funzioni di inizializzazione di wiringPi. Si azzera dopo 71 minuti circa.

void delay (*unsigned int howLong*)
L'esecuzione del programma si ferma per un periodo di tempo pari a *howLong* millisecondi . A causa della natura multi- tasking di Linux il ritardo potrebbe essere più lungo . Il ritardo massimo è di circa 49 giorni (è un intero senza segno di 32 bit).

void delayMicroseconds (*unsigned int howLong*)
L'esecuzione del programma si ferma per un periodo di tempo pari a *howLong* microsecondi. A causa della natura multi- tasking di Linux potrebbe essere più lungo. Il ritardo massimo è di circa 71 minuti (è un intero senza segno di 32 bit).

Ritardi sotto i 100 microsecondi sono ottenibili con loop, ritardi superiori ai 100 microsecondi usando la funzione *nanosleep().* Tenere sempre presente le implicazioni di ritardi molto brevi sulle prestazioni globali del sistema , soprattutto se utilizzando thread.

Priorità, Interruzioni e Threads

WiringPi fornisce funzioni di supporto alla gestione della priorità di un processo o di un thread e per lanciare un nuovo thread all'interno di un processo.

Program or Thread Priority

int piHiPri (*int priority*) ;

Questa funzione tenta di aumentare la priorità di un programma (o un thread in un programma multi- threaded) abilitando così il real time schedulling. Il parametro priorità varia da 0 (il default) a 99. La funzione non rende il programma più veloce, ma farà in modo che a run-time al processo venga assegnato un time slice maggiore. Il valore del parametro è in relazione agli altri processi nel senso che se abbiamo un programma a priorità 1 ed un altro a 2 si avrà lo stesso effetto impostando il primo a 10 e l'altro a 90 (a patto che nessun altro programma sia in esecuzione con priorità più elevata). Il valore di ritorno è 0 per il successo e -1 in caso di errore . In qust'ultimo caso, il codice dovrebbe prevedere la lettura della *variabile globale errno*. La funzione può esser lanciata dall'utente root.

Interruzioni

La gestione dell'interruzione nel nostro programma libera il processore consentendogli di compiere altre operazioni in attesa dell' interrupt. Un pin di GPIO può essere impostato perché l'interruzione avvenga su un fronte di salita di discesa o entrambi.

int waitForInterrupt (*int pin , int timeout*) ;

Si attende un interrupt su *pin* per un tempo *timeout*, durante questo intervallo di tempo e fino al suo scadere il programma è in stallo. Il parametro *timeout* è espresso in millisecondi , se impostato a -1 che significa attesa infinita. Il valore di ritorno della funzione sarà -1 in caso di errore (ed errno sarà impostata in modo appropriato) , 0 se è scaduta , o 1 se si è avuta un' interruzione.
Prima di chiamare waitForInterrupt , è necessario inizializzare il pin GPIO a rispondere all'interruzione su di un fronte di salita o di discesa di un segnale ad onda quadra. Al momento l'unico modo per farlo è quello di utilizzare il programma GPIO. Ad esempio se vogliamo settare un interrupt su di un fronte di discesa sul pin GPIO 0 , per configurare l'hardware , dobbiamo eseguire:

```
#gpio edge 0 falling
```

prima di eseguire il programma o con una *system()* all'interno del programma .

int wiringPiISR (*int pin, int edgeType, void (*function)(void)*) ;

Questa funzione chiama *function* quando viene ricevuta un' interruzione su *pin* sul fronte *edgeType*. Che può assumere i seguenti valori :
INT_EDGE_FALLING
INT_EDGE_RISING
INT_EDGE_BOTH
INT_EDGE_SETUP
INT_EDGE_SETUP vuol dire che il pin è stato configurato altrove (ad esempio con il programma GPIO) , negli altri casi prevale la specifica del parametro su quella definita dal programma gpio. Il numero di *pin* dipende dalla modalità di setup iniziale. wiringPiISR può essere eseguita in qualsiasi modalità, e non sono necessari i privilegi di root.
La funzione associata all' interruzione verrà eseguita ad una priorità elevata (se il programma è eseguito utilizzando sudo , o come root) ed in concorrenza con il programma principale, ed ha accesso a tutte le variabili globali , gli handles di apertura di file e così via.

Programmazione concorrente (multi-threading)

wiringPi ha un'interfaccia semplificata per l'implementazione Linux di thread Posix , così come un meccanismo semplificato per accedere alla mutua esclusione di mutex. Utilizzando queste funzioni è possibile creare un nuovo processo (una funzione all'interno del programma principale) che gira in concorrenza con il programma principale semplicemente passando variabili tra di loro.

int piThreadCreate (*name*) ;

Crea un thread *name* che è una funzione di tipo PI_THREAD dichiarata nel programma. *name* viene eseguita in concorrenza con il programma principale. Un esempio di utilizzo può essere quello di avere questa funzione che attendere un interrupt mentre il programma svolge altre attività . Il thread può indicare un evento o un'azione utilizzando le variabili globali per comunicare ritornare al programma principale , o di altri thread .

Funzioni sono dichiarate come segue :

```
PI_THREAD (  myThread  )
{
  .. codice che gira in concorrenza con il programma
     principale probabilmente in un loop infinito

}
```

Nel programma principale sarà fatta partire:

```
x = piThreadCreate ( myThread ) ;
if ( x != 0 )
  printf ("non è partita")
```

Questo non è altro che un' interfaccia semplificata per il meccanismo dei thread Posix supportati da Linux .

piLock (*int keyNum*) ;
piUnlock (*int keyNum*) ;

Queste funzioni consentono di sincronizzare gli aggiornamenti delle variabili dal programma principale per qualsiasi thread in esecuzione. *keyNum* è un numero da 0 a 3 e rappresenta una "chiave " . Quando un altro processo tenta di bloccare la stessa chiave , verrà bloccato finché il primo processo ha sbloccato la stessa chiave .
Le due funzioni servono per garantire scambio di dati validi fra i due processi, in caso contrario è possibile che il thread si "risvegli" mentre i dati vengolo letti / modificati dall' altro thread e quindi siano inconsistenti. In sostanza

```
…….
……..
piLock( 2 )

   zona critica cioè la parte di codice
   del processo in cui le variabili
   vengono referenziate cioè lette o scritte

piUnlock( 2 )
……
```

Serial Library

WiringPi include una libreria di gestione della porta seriale. Si può utilizzare la porta seriale on-board , o qualsiasi dispositivo seriale USB senza distinzioni particolari tra di loro . Basta specificare il nome del dispositivo nella funzione open iniziale. Per utilizzarlo, è necessario assicurarsi che il programma includa il seguente file :

 # include <wiringSerial.h>

Per la gestione dell aporta seriale sono disponibili le seguenti funzioni :

int serialOpen (*char *device, int baud*) ;

Apre e inizializza il dispositivo seriale *device* e imposta la velocità di trasmissione *baud* . Si imposta la porta in modalità "raw", e il timeout di lettura a 10 secondi . Il valore di ritorno è il descrittore di file o -1 per un errore , nel qual caso controllare la variabile di sistema errno.

serialClose void (*int fd*) ;

Chiude il dispositivo identificato dal descrittore di file *fd* .

void serialPutchar (*int fd, unsigned char c*) ;

Invia il byte *c* alla periferica seriale identificata dal descrittore di file *fd*.

void serialPuts (*int fd , char * s*) ;

Invia la stringa *s* con il suo terminatore al dispositivo seriale identificato dal descrittore di file *fd*.

void serialPrintf (*int fd , char * messaggio , ...*) ;

Emula la funzione di sistema printf al dispositivo seriale .

int serialDataAvail (*int fd*) ;

Restituisce il numero di caratteri disponibili per la lettura , o -1 per qualsiasi condizione di errore.

int serialGetchar (*int fd*) ;

Restituisce il successivo carattere disponibile sul dispositivo seriale. Se non sono disponibili caratteri la funzione si blocca per 10 secondi e se dopo tale intervallo non vi sono dati la funzione restituità -1.

void serialFlush (*int fd*) ;

Svuota il buffer di ricezione dei dati.

Nota : Il descrittore di file *fd* restituito è un descrittore di file standard di Linux . È possibile quindi utilizzare la lettura standard, *write (), ad* esempio nel caso di scrittura di un grande blocco di dati binari al posto di di *serialPutchar ()* o *serialPuts ()* .

Advanced Serial Port Control

wiringSerial per fornisce un controllo semplificato adatto alla maggior parte delle applicazioni , se si ha necessità di un controllo avanzato del dispositivo seriale, ad esempio il controllo di parità , oppure impostare la linea seriale in modalità 7 bit, potremo realizzare questo programma :

 # include <termios.h>

e nella funzione :

```
struct termios options ;

  tcgetattr ( fd, &options ) ; // Read current options
  options.c_cflag &= ~CSIZE ;  // Mask out size
  options.c_cflag |= CS7 ;     // Or in 7-bits
  options.c_cflag |= PARENB ;  // Enable Parity - even by default
  tcsetattr (fd, &options) ;   // Set new options
```

fd è il descrittore di file che restituisce *serialOpen ( )* .

SPI Library

WiringPi include una libreria che facilita l'utilizzo dell' interfaccia SPI del Raspberry Pi. Prima di usare l'interfaccia SPI, potrebbe essere necessario caricare i driver SPI nel kernel utilizzando l'utilità GPIO :

gpio load spi

Se si vuole dimensione un buffer maggiore di 4KB , allora (in KB) sulla riga di comando :

gpio load spi 100

che assegnerà un buffer di 100 KB . Per utilizzare la libreria SPI , è necessario:

 # include <wiringPiSPI.h>

nel programma . Programmi devono essere linkati con *– l wiringPi* come al solito .

Funzioni disponibili :

int wiringPiSPISetup (*int channel, int speed*) ;

Questo è il modo per inizializzare un canale (PI ha due canali denominati 0 e 1) . Il parametro velocità è un numero intero compreso tra 500.000 e 32.000.000 e rappresenta la velocità di clock SPI in Hz . Il valore restituito è il descrittore di file per il dispositivo o -1 in caso di errore . Come al solito per capire il motivo dell' errore verificare la variabile di sistema *errno.*

int wiringPiSPIDataRW (*int channel, unsigned char *data, int len*) ;

Esegue una scrittura / lettura simultanea sul bus SPI selezionato . I dati presenti nel buffer vengono sovrascritti dai dati restituiti da SPI .
Questo è tutto quello che c'è nella libreria di supporto. Utilizzando le funzioni read() è possibile fare una semplice lettura sul bus SPI. La write() può essere la migliore da utilizzare per l' invio di dati a catene di shift registers, o quelle strisce di LED dove inviare triplette RGB di dati. Dispositivi

come convertitori A/D e D/A di solito per lavorare hanno bisogno di eseguire read/write concorrente.

I2C Library

WiringPi include una libreria che può rendere più facile l'utilizzo dell' interfaccia I2C del Raspberry Pi . Prima di poter utilizzare l'interfaccia I2C , potrebbe essere necessario utilizzare l'utilità GPIO per caricare i driver I2C nel kernel :

gpio load i2c

Se avete bisogno di un baud rate diverso da 100Kbps di default , quindi è possibile fornire questo sulla riga di comando :

gpio load i2c 1000

imposterà la velocità di trasmissione a 1000kbps.

Per utilizzare la libreria I2C , è necessario inserire nel programma :

 # include <wiringPiI2C.h>

È possibile utilizzare i comandi di sistema per controllare i dispositivi I2C , ad esempio il programma ***i2cdetect*** . Ricordiamo infatti che su un Raspberry Pi rev1 il dispositivo è 0 , e su rev2 il dispositivo è 1 . Esempio

i2cdetect –y 0 # Rev 1
i2cdetect -y 1 # Rev 2

E' possibile utilizzare il comando gpio per eseguire i2cdetect e ottenere i parametri corrretti per la revisione della scheda : *gpio i2cdetect*

Funzioni disponibili

int wiringPiI2CSetup (*int devid*) ;

Questa funzione inizializza il sistema I2C con l' identificativo del dispositivo *devid*. wiringPiI2CSetup ( ) funzionerà indipendentemente da rev di PI aprendo il dispositivo appropriato in */dev* .
Il valore restituito è il filehandle standard di Linux , vi è un errore se -1. Ad esempio l'espansione MCP23017 GPIO ha di solito Id 0x20 numero che si passa in wiringPiI2CSetup( ). Nelle funzioni che seguono se l valore di ritorno è -1 si è avuto un errore e per comprenderlo è possibile testare la variabile di sistema errno.

int wiringPiI2CRead (*int fd*) ;

Lettura sul dispositivo a cui è associato *fd*.

int wiringPiI2CWrite (*int fd , int dati*) ;

Scrittura di *dati* sul dispositivo *fd*. Alcuni dispositivi accettano dati in questo modo , senza la necessità di accedere ai registri interni .

int wiringPiI2CWriteReg8 (*int fd , int reg , dati int*) ;

***int* wiringPiI2CWriteReg16 (int fd , int reg , dati int) ;**

Scrivono un valore di 8 o 16 bit nel registro del dispositivo *fd* .

***int* wiringPiI2CReadReg8 (*int fd , int reg*) ;**
***int* wiringPiI2CReadReg16 (*int fd , int reg*) ;**
Leggono un valore di 8 o 16 bit dal registro del dispositivo *fd* .

Shift Library

Le funzioni di questa libreria consentono di spostare i valori a 8 bit fuori/dentro Pi, da dispositivi come shift- register (ad esempio il 74×595) e così via. Il programma deve includere i file :

 # include <wiringPi.h>
 # include <wiringShift.h>

Le funzioni disponibili sono :

***uint8_t* shiftIn (*uint8_t dataPin, uint8_t clockPin, uint8_t bitorder*) ;**
Legge gli 8 bit di che si presentano su *datapin* in *bitorder* quando *clockpin* è alto. Il parametro *bitorder* assume i valori LSBFIRST o MSBFIRST. La funzione restituisce il valore di 8 bit.

***void* shiftOut (*uint8_t dataPin, uint8_t clockPin, uint8_t bitorder, uint8_t val*) ;**
Scrive gli 8 bit di *val* su *datapin* in *bitorder* quando *clockpin* è alto.

Software PWM Library

WiringPi include un gestore PWM software -driven in grado di emettere un segnale PWM su uno qualsiasi dei pin GPIO del Raspberry Pi .
Ci sono alcune limitazioni. Per mantenere un basso utilizzo della CPU , la larghezza minima impulso è 100µS . Che combinati con il default suggerito gamma di 100 dà una frequenza PWM di 100Hz. È possibile ridurre l'intervallo per ottenere una frequenza più alta, a scapito della risoluzione, o l'aumento per avere più risoluzione , ma che abbasserà la frequenza. Se si modifica l' ampiezza d'impulso nel codice del driver , quindi essere consapevoli che i ritardi inferiori a 100µS wiringPi lo fa in un loop software , il che significa che l'utilizzo della CPU aumenterà drasticamente , e controllare più di un pin sarà quasi impossibile .
Si noti inoltre che, mentre le routine si girano a una priorità e in tempo reale superiore, Linux può ancora influire sulla precisione del segnale generato .
Tuttavia , all'interno di questi limiti, è realizzabile il controllo di una luce / LED o un motore.

Per utilizzare :

 # include <wiringPi.h>
 # include <softPwm.h>

In fase di compilazione del programma è necessario includere la libreria pthread e wiringPi :
gcc -o myprog mioprog.c - l wiringPi -lpthread

Come al solito è necessario inizializzare wiringPi con le funzioni *wiringPiSetup(), wiringPiSetupGpio () o wiringPiSetupPhys(). wiringPiSetupSys ( )* non è abbastanza veloce , quindi è necessario eseguire il programmo con sudo.
Sono disponibili le funzioni :

int softPwmCreate (*int pin, int initialValue, int pwmRange*) ;

Crea un pin PWM controllato da software. La numerazione dei pin dipende come al solito da quale funzione di inizializzazione si è utilizzata. Se pwmRange viene posto a 100, allora il valore può variare da 0 (off) a 100 (on). Se il valore di ritorno è diverso da 0 controllare la variabile errno per capire l'errore.

void softPwmWrite (*int pin, int value*) ;

Aggiorna il *value* PWM sul *pin* dato . Il *value* deve esser compreso nel range specificato in *softPwmCreate()* come anche *pin.*

Note

- Ogni " ciclo" di uscita PWM prende 10mS con il valore gamma predefinito di 100 , in modo da cercare di modificare il valore di PWM più di 100 volte al secondo sarà inutile .
- Ogni pin attivato in modalità softPWM utilizza circa il 0,5% della CPU .
- Non esiste attualmente alcun modo per disattivare softPWM su un pin mentre il programma è in esecuzione .
- Per mantenere l'uscita PWM è necessario mantenere il programma in esecuzione.

Software Tone Library

WiringPi include un gestore di suono in grado di emettere un segnale ad onda quadra su un qualsiasi pin del GPIO di Raspberry Pi. Per mantenere un basso utilizzo della CPU , la larghezza minima impulso è 100µS, per cui si ottiene una frequenza massima di 1 / 0.0002 = 5000 Hz .
Si noti inoltre che, anche se le routine girano a una priorità elevata, Linux può influire sulla precisione del segnale generato. All 'interno di questi limiti , è possibile ottenere semplici toni su altoparlanti ad alta impedenza o piezo.

Per utilizzare :
 # include <wiringPi.h>
 # include <softTone.h>

Quando si compila il programma è necessario includere le librerie pthread e wiringPi :

gcc -o myprog mioprog.c -lwiringPi -lpthread

Come al solito è necessario inizializzare wiringPi con le funzioni *wiringPiSetup(), wiringPiSetupGpio () o wiringPiSetupPhys(). wiringPiSetupSys ( )* non è abbastanza veloce , quindi è necessario eseguire il programmo con sudo.
Sono disponibili le funzioni :

int softToneCreate (*int pin*) ;

Definisce il pin su cui emettere il segnale. È possibile utilizzare qualsiasi pin GPIO con la numerazione definita della funzione di inzializzazione utilizzata. Il valore di ritorno è 0 per il successo, se diverso da 0 controllare come al solito la variabile errno per capire l'errore.

void softToneWrite (*int pin, int freq*) ;

Emette un suono alla frequenza *freq* pin *pin*. Il tono verrà riprodotto finché non si imposta la frequenza a 0 .

Note

- Ogni pin attivato in modalità Softtone utilizza circa il 0,5% della CPU .
- Il suono verrà emasso sino a quando il programma è in esecuzione.