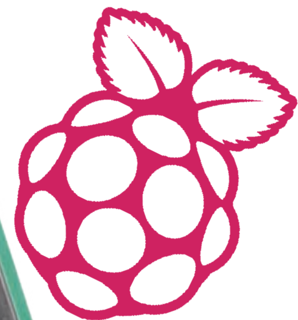
SHIELD DE EXPANSIÓN LCD PARA RASPBERRY PI

Ya está aquí el shield de expansión LCD para Raspberry Pi que permite realizar una interfaz de control externa para nuestras aplicaciones sin necesidad de tener conectados constantemente video, teclado y ratón.

La especificidad de un micro PC como Raspberry Pi respecto a un microcontrolador es aquella de poder desarrollar distintas tareas compartiendo los recursos. Además de los periféricos actualmente disponibles en la tarjeta, es posible añadir otros en una cadena casi infinita, cuyo límite viene dado por el uso de las líneas de comunicación, por la capacidad de procesamiento y la memoria disponible. Mucho depende también de cómo son diseñadas y realizadas las aplicaciones, sobre todo en lo que se refiere a la capacidad de respetarse el uno al otro y de compartir los recursos utilizados sin obstaculizarse o creando "bloques" irresolubles (deadlock).
Con esta visión comenzamos a presentaros una serie de shield de expansión especialmente fabricados para Raspberry Pi y diseñados de manera que puedan ser apilados el uno sobre el otro y ser utilizados a la vez. El shield de expansión LCD que presentamos hoy permite realizar una interfaz de control externa para nuestras aplicaciones.



MARCO MAGAGNIN

El integrado **MCP23017**

El integrado MCP23017 permite realizar un periférico de expansión con 16 canales de I/O digitales, controlables por medio de comunicaciones basadas en el bus I2C. Disponible en formato 28-PDIP, 28-SOIC y 28-SSOP, en nuestro circuito hemos adoptado la “cómoda” configuración de 28 pin PDIP. Los pines son visibles en **Fig. A**.
El sistema master, para controlar el integrado, debe ajustar correctamente los registros de configuración, cada uno de 8 bit, mediante los cuales es posible seleccionar la dirección de cada entrada (in o out) y la polaridad.
En el esquema de **Fig. B** es visible la arquitectura del integrado. Los pines de I/O están agrupados en dos puertos que controlan 8 pin cada uno: PORTA y PORTB. Mediante los registros de control IODIRA/B es posible establecer la dirección de funcionamiento de cada pin de I/O. La polaridad de cada puerta puede ser invertida interviniendo sobre el registro Polarity Inversion. Obviamente el sistema de control master es capaz de leer todos los registros.

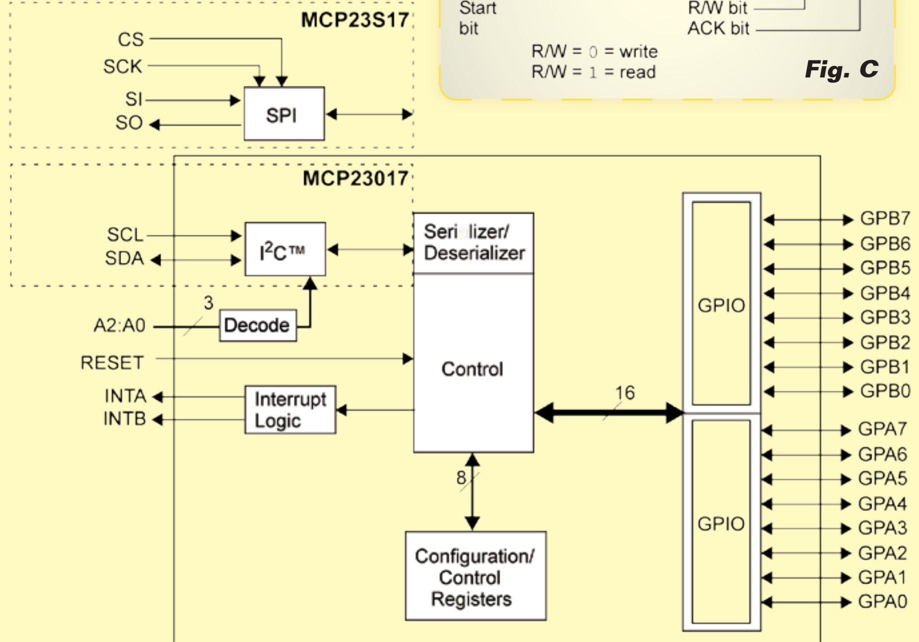


Fig. B

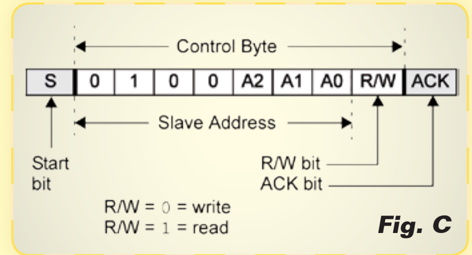


Fig. C

Registro de direccionamiento
El registro de direccionamiento del integrado MCP23017 tiene una longitud de 7 bit (**Fig. C**). Los cuatro bit más significativos deben ser ajustados al valor fijo “0100” e identifican el fabricante del chip. Los valores de los tres bit restantes son establecidos en base a los niveles de tensión asignados a los pines A1, A2 y A3 del integrado. Esta configuración permite direccionar sobre el mismo bus I2C hasta ocho integrados MCP23017 por un total de 128 pin de I/O. El último bit del registro de dirección permite ajustar el tipo de operación que se quiere ejecutar, “0” para ejecutar una operación de escritura de registros y “1” para ejecutar una operación de lectura.

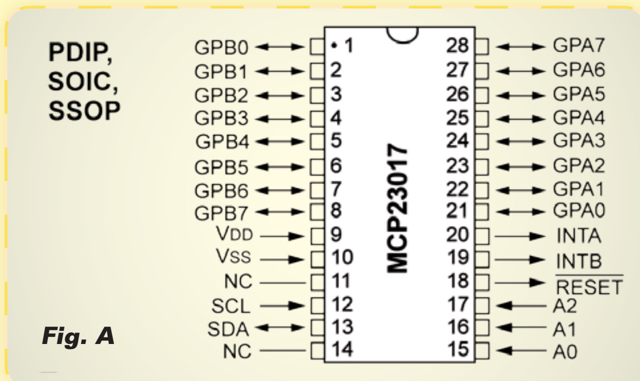


Fig. A

Registros de gestión
Para poder controlar el funcionamiento del integrado MCP23017 es necesario escribir y leer una serie de registros en una secuencia bien definida. Una panorámica de los registros disponibles la tenemos en **Fig. D**: la primera columna de la tabla indica el nombre del registro, la segunda columna, la dirección hexadecimal con el cual es posible leer o escribir el contenido del registro.
Los registros IODIRA e IODIRB, de 8 bit cada uno, permiten ajustar cada pin de I/O sobre los puertos A o B como entrada o como salida. Para configurar el pin como entrada, el correspondiente valor del bit en el registro debe ser establecido a “1”, mientras para configurarlo como salida debe ser establecido a “0”. Los registros GPIOA y GPIOB permiten leer el estado de los pines de entrada de los respectivos puertos. Si el bit correspondiente a la posición del pin se establece a “0”, el pin está a nivel bajo, de otra manera se encuentra a nivel alto.
Igualmente para ajustar los niveles de los pines configurados como salida es necesario componer los contenidos de los registros OLATA y OLATB de manera que el bit correspondiente a cada pin asuma el valor “0” en el caso se quiera poner a nivel bajo o “1” para ponerlo a nivel alto.
Los registros IPOLA e IPOLB permiten ajustar cada pin de I/O de manera que trabaje con lógica invertida. Si el bit correspondiente a un determinado pin es ajustado a “1” los niveles serán presentados en lógica invertida.
Los registros GPPUA y GPPUB permiten activar o no la resistencia interna de pull-up para cada pin de I/O. Si el bit correspondiente a un determinado pin se pone a “1” la resistencia de pull-up se activa.
Finalmente, profundizaremos en la gestión de las interrupciones disponibles en el

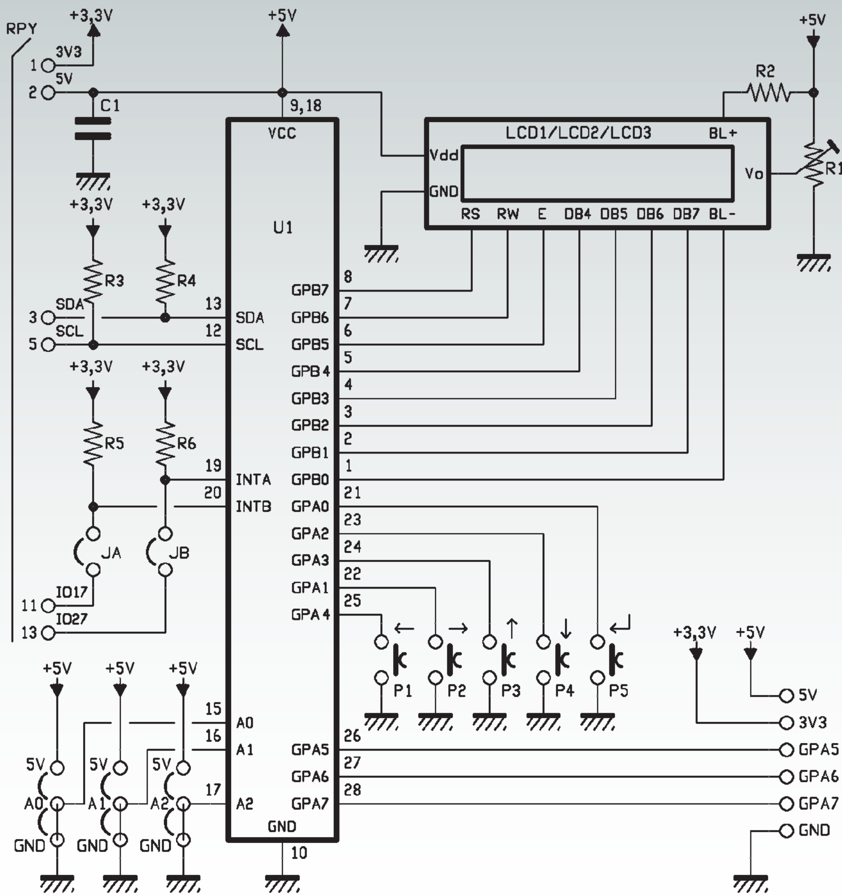
***Fig. D** - Mapa de la memoria que contiene los valores de los registros de control del integrado MCP23017.*

Register Name	Address (hex)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	POR/RST value
IODIRA	00	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IODIRB	01	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IPOLA	02	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
IPOLB	03	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
GPINTENA	04	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
GPINTENB	05	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
DEFVALA	06	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
DEFVALB	07	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
INTCONA	08	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
INTCONB	09	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
IOCON	0A	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
IOCON	0B	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
GPPUA	0C	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
GPPUB	0D	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
INTFA	0E	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INTO	0000 0000
INTFB	0F	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INTO	0000 0000
INTCAPA	10	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
INTCAPB	11	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
GPIOA	12	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
GPIOB	13	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
OLATA	14	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000
OLATB	15	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000

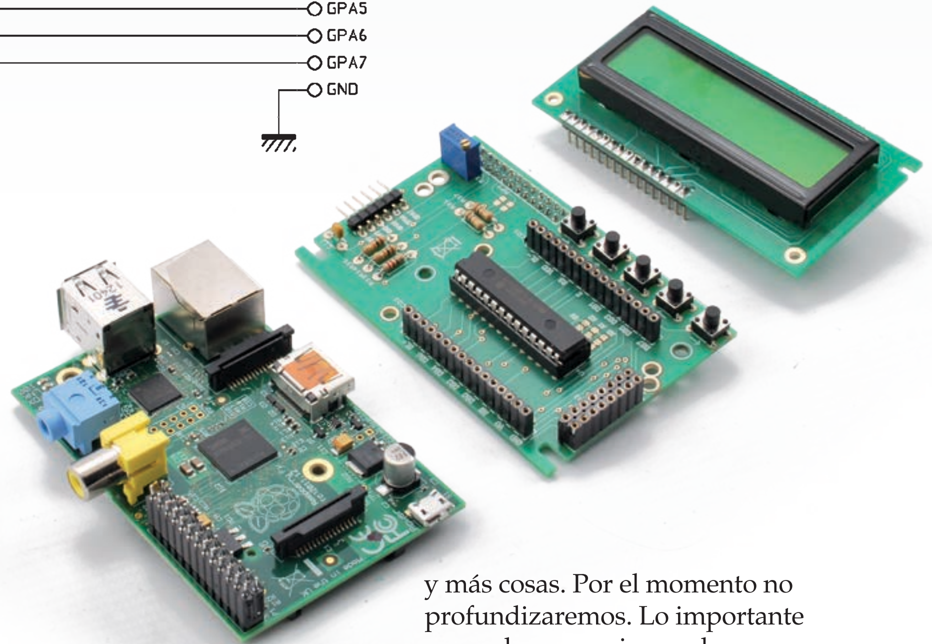
integrado MCP23017: los pines 19 y 20 del integrado están conectados a los terminales del módulo de gestión de las interrupciones, INTB para el banco B, INTA para el banco A respectivamente.
En el integrado MCP23017 es posible configurar cada pin de manera que al variar su estado se active el pin de interrupción correspondiente al banco al que pertenece. De este modo, por parte del sistema de gestión master, no es necesario leer continuamente todos los pines para comprobar si alguno cambia de estado, es suficiente gestionar las interrupciones INTA y/o INTB para saber cuando se produce una variación en uno de los pines monitorizados. Después se lee el registro de estado del banco correspondiente y se identifica, comparándolo con los valores anteriores, cual o cuales han sufrido una variación. Antes de activar la monitorización de los pines con los registros GPINTENA y GPINTENB, es necesario configurar las condiciones en base a las cuales debe ser activada la interupción. Para hacer esto se utilizan las parejas de registros DEFVALA, DEFVALB y INTCONA, INTCONB. La primera pareja de registros permite configurar, para cada pin de los dos bancos, un nivel de referencia para controlar la activación de la interrupción. En cada caso esta configuración depende de la configuración de los registros INTCON. Si es configurado a "0" el bit correspondiente a un pin, la interrupción se activara cuando el pin tome el nivel "1", lo opuesto en el otro caso.
La configuración de los bits de los registros INTCONA e INTCONB sin embargo permite ajustar si la interrupción debe ser controlada por las condiciones ajustadas en los registros DEFVAL, o simplemente a cada cambio de estado del nivel de los pines. Si se ajusta a "0" el bit correspondiente a un pin en el registro INTCON, la interrupción será activada con cada cambio de estado del pin, si se ajusta a "1", la interrupción será activada según las condiciones indicadas por los registros DEFVAL. Las condiciones por las cuales es posible hacerse cargo de un interrupción desde los pines INTA e INTB dependen de la configuración del bit INTCOL (bit 1) del registro IOCON, que permite configurar distintas opciones de funcionamiento del integrado. El valor predefinido es "0" que significa que los pines de interrupción están normalmente a nivel alto que se convierte en bajo al activarse la interrupción.
Para saber cuál de los pines monitorizados ha activado la interrupción si lee uno de los registros de la pareja INTFA, INTFB. El o los bit ajustados a "1" corresponden al o a los bit que han generado la interrupción. El nivel de los pines interesados se puede detectar leyendo el registro correspondiente INTCAPA o INTCAPB.
Estos registros reflejan el estado de los pines en el momento de la interrupción y lo mantienen hasta que son leídos.
En el datasheet del integrado encontraréis más información sobre su funcionamiento.

En este artículo describimos el hardware del shield de expansión, la librería que permite una notable simplificación del uso del shield y profundizaremos en el integrado MCP23017 usado como interfaz entre Raspberry Pi y los módulos LCD.
Antes de empezar la descripción, una nota sobre el desarrollo de aplicaciones. Hemos mencionado la necesidad de diseñar las aplicaciones destinadas a los sistemas embebidos y entre estos aquellos basados en el sistema operativo GNU/Linux, de manera que sean capaces de colaborar entre ellos utilizando los recursos disponibles de manera oportuna.. Tomamos como ejemplo el shield descrito en este artículo que como veremos requiere el uso del bus I²C para la comunicación con Raspberry Pi. Si construimos una aplicación que gestiona el display LCD en exclusiva, el software a escribir equivale alque hemos escrito para un microcontrolador: un programa en un ciclo infinito que gestiona los pulsadores y que presenta sobre la pantalla lo escrito en base a la lógica predefinida.
Sin embargo si queremos que el display LCD este en servicio respecto a una multitud de aplicaciones debemos gestionar las cosas de manera diferente. Supongamos por ejemplo que queremos utilizar siempre el shield como en el caso anterior, pero que sea capaz también de visualizar mensajes (provenientes de aplicaciones distintas que trabajan en concomitancia) como la llegada de una llamada telefónica, la variación de una entrada de I/O operada vía web, la falta de corriente o de conectividad de red o cualquier otra cosa que sea necesaria.
En una arquitectura de este tipo, distintas aplicaciones funcionan simultáneamente sobre el sistema y al ocurrir eventos predefinidos,

[esquena **ELÉCTRICO**]



requieren utilizar el bus de comunicación I²C hacia el LCD para comunicar su mensaje
Si una aplicación tiene ocupado ya el bus por un motivo cualquiera, las otras aplicaciones que requieren utilizar el mismo bus, encontrándolo ocupado (busy) no podrán hacer otra cosa que dar un error. La primera solución que podría venir en mente es realizar una única y enorme aplicación que gestione todas las necesidades de manera monolítica, un poco como en un enorme microcontrolador. Es intuitivo que este modo de proceder comporte más aspectos negativos que positivos: rigidez de la aplicación, dificultad de diseño, fabricación, mantenimiento y actualización, dificultad de temporización entre las distintas partes, fragilidad (si una parte, aunque insignificante va en bloque, cae toda la aplicación). Para resolver una exigencia aplicativa de este tipo es necesario recurrir a las arquitecturas capaces de soportar aplicaciones concurrentes y colaboradores, gestionar comunicaciones entre los distintos actores, basadas en servidores que orquestan el uso de recursos para compartir y donde las distintas exigencias aplicativas deben ser realizadas como cliente que requieren el uso de los recursos "centralizados" mediante peticiones basadas en protocolos compartidos. Por ejemplo, supongamos que dos recursos utilicen de manera exclusiva los recursos que tienen asignados: el shield LCD que utiliza el bus I²C y un shield GSM/GPS que utiliza el puerto serie.
Para cada uno de estos recursos es necesario realizar un "servidor" que gestione el recurso autónomamente y que acepte las peticiones de los distintos clientes sobre un canal de comunicación predefinido. Hay muchas alternativas posibles: socket TCP/IP, funcionalidad database, archivos simples y más cosas. Por el momento no profundizaremos. Lo importante es que los mecanismos de comunicación sean capaces de gestionar correctamente el fenómeno, por ejemplo generando código o gestionando "semáforos" para coordinar correctamente las peticiones de los clientes. Los clientes cuando tienen necesidad de acceder a uno de los recursos gestionados centralmente no deben acceder directamente, pero si deben utilizar el canal de comunicación prede-



finido por aquel recurso con el protocolo y las modalidades previstas. En resumen, en el mundo embebido el diseño no debe ser realizado solo para una aplicación, debe tener en cuenta la arquitectura que se quiere realizar en su conjunto. ¿Mejor o peor respecto a los microcontroladores? Esta pregunta no tiene una respuesta definitiva. Depende de aquello que se quiere realizar y los requisitos asociados. Sin embargo seguramente nos encontraremos siempre más en las condiciones de integrar y hacer colaborar estos dos ambientes junto a otros como FPGA y PLC.
Tras esta premisa, volvamos a nuestro shield para el cual esta vez os presentamos un simple programa de uso, ya adaptado para ser transformado según la arquitectura que os hemos anticipado.

ESQUEMA ELECTRICO

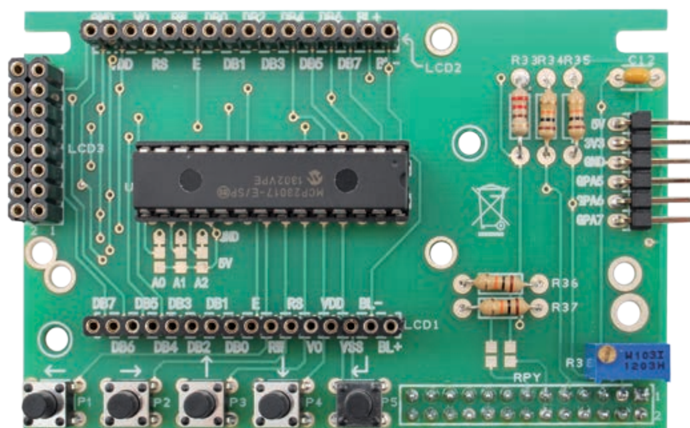
Para evitar utilizar la mayor parte de los pines de entrada/salida presentes en el conector de Raspberry Pi hemos utilizado el integrado de Microchip MCP23017 que ofrece 16 entradas/salidas digitales controladas mediante el bus I^2C. De esta manera, utilizando de forma no exclusiva solo dos pines del conector de Raspberry Pi, podemos controlar una pantalla LCD, interactuar con cinco pulsadores y gestionar tres entradas auxiliares.
La configuración base está pensada para realizar un sistema de menú en cascada que permite elegir entre jerarquías de opciones para después configurar parámetros o visualizar sets particulares de informaciones. En realidad, desde programa podemos utilizar los pulsadores y las entradas como mejor creamos. El shield está provisto de distintos conectores capaces de acoger distintos tipos de LCD y en particular los

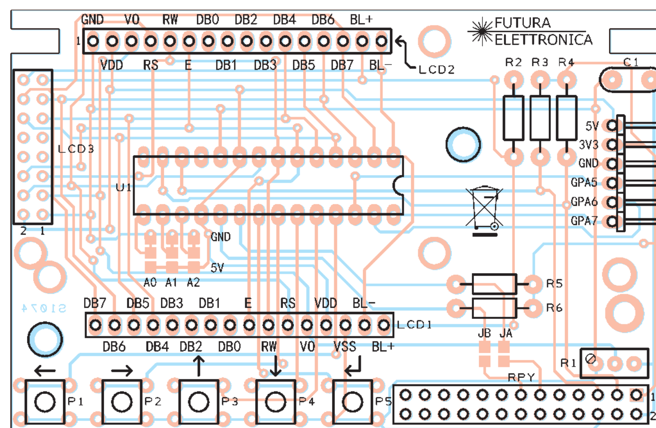
Rapiro, un robot humanoide con el corazón en forma de frambuesa

Entre las múltiples aplicaciones más o menos evidentes, desde los web-server a los sistemas de reconocimiento de matrículas, desde los media center a las estaciones meteorológicas, encontramos aquí una de las primeras aplicaciones en el campo de la robótica de la económica placa Linux Raspberry Pi: Rapiro, un pequeño robot humanoide completamente programable y, como tal, destinado principalmente al mundo "educacional". Rapiro dispone de 12 servomotores y es capaz de caminar, mover la cabeza y responder a los comandos vocales; en el interior, una tarjeta Raspberry Pi se ocupa de la gestión de todos los sistemas y de la conexión al controlador remoto, típicamente un smartphone. Para la conexión se utiliza un dongle Bluetooth y un programa específico para cargar sobre el smartphone.

El pequeño robot (alrededor de 30 centímetros de alto) dispone también de una cámara, ojos que se iluminan gracias a dos LED, sensor de distancia y altavoz.
Para el control de los servos se emplea una tarjeta compatible Arduino que garantiza la máxima flexibilidad al programar.
Según los defensores del proyecto (que lo han propuesto en Kickstarter y ya han recogido la financiación necesaria), Rapiro costara cuarta parte quelos robot parecidos y una décima que otros robots humanoides basados en tarjetas Linux embebido. (www.kickstarter.com).

[plano de **MONTAJE**]





Lista de Materiales:

C1: 100 nF multicapa
R1: Potenciómetro multivuelta 10 kohm
R2: 100 ohm
R3÷R6: 10 kohm
P1÷P5: Microswitch
U1: MCP23017-E/SP

Varios:
- Display LCD
- Torreta F/F 18 mm
- Tornillo 10mm 3 MA (2 pz.)
- Zócalo 14+14
- Tira 6 pines macho 90°
- Tira 8 pines hembra (2 pz.)
- Tira 16 pines hembra (2 pz.)
- Conector Hembra 26 polos para RaspberryPi
- Circuito impreso

siguientes modelos que, aun teniendo dimensiones diferentes, comparten la misma configuración de los pines de comunicación datos y comandos:

- LCD 16x2 retroiluminado (cod. 1446-ACM1602B-FL-YBW);
- LCD 16x2 retroiluminado blanco/negro (cod. 1446-LCD16x2WB);
- LCD 8x2 retroiluminado azul (cod. 1446- LCD8x2BN);

Volviendo al esquema eléctrico, el integrado MCP23017 que describimos más en profundidad en el recuadro dedicado, pone a nuestra disposición 16 líneas de I/O subdivididas en dos bancos de ocho líneas cada uno llamadas GPA y GPB.
El banco GPB está todo dedicado a la gestión del display LCD come sigue: los pin GPB1, GPB2, GPB3 y GPB4 están conectados a las cuatro líneas datos del LCD, respectivamente DB7, DB6, DB5 y DB4; el pin GPB5 controla la entrada ENABLE del LCD, el pin GPB6 la entrada R/W y el pin GPB7 la entrada RS que permite seleccionar los registros de instrucciones y escritura.
Los cinco pulsadores están conectados a los pines del banco GPA: GPA0 al pulsador P5, GPA1 al pulsador P2, GPA2 al pulsador P4, GPA3 al pulsador P3 y GPA4 al pulsador P1.
La alimentación de 5V está conectada por el pin 2 del conector GPIO de Raspberry Pi y la masa está conectada al pin 6. El potenciómetro de ajuste de 10 kohm, conectado al pin V0 del LCD permite regular el contraste de la pantalla.
Las salidas INTA e INTB están conectadas mediante los puentes JB y JA a los GPIO27 y GPIO17 de Raspberry Pi, de manera que los deja disponibles a eventuales aplicaciones. Sobre el shield están disponibles en un conector de pines también los GPA5, GPA6 y GPA7, del banco GPA del integrado MCP23017, "avanzados" por la gestión de los pulsadores.
En el plano de montaje es posible ver la disposición de los componentes. Una única sugerencia, aparte de las típicas atenciones, es posicionar el doble conector de pines de manera que distancie el circuito impreso del shield del de Raspberry Pi para permitir el montaje "en paquete" en el caso que se quieras utilizar conjuntamente a otros shield.
La posibilidad de "empaquetamiento" viene dada por el hecho que el integrado MCP23017 está conectado al GPIO de Raspberry Pi únicamente con los pines necesarios para la comunicación I²C y a la alimentación del shield. Todos los otros pines de Raspberry Pi son simplemente pasantes. Además, aunque también una buena parte de las I/O del integrado MCP23017 son utilizadas para la gestión del monitor LCD y los pulsadores de comando presentes en el shield, quedan 3 que están disponibles para eventuales usos personalizados.

USO PRÁCTICO DEL SHIELD

Como primera operación monta-

mos uno de los posibles display LCD sobre el shield y después montamos el shield sobre el conector de Raspberry Pi, prestando atención a que la parte inferior del shield no quede en contacto con los conectores USB o Ethernet. En caso de duda protegemos los conectores mismos con cinta aislante. Conectamos los periféricos, red y alimentación a Raspberry Pi en la forma habitual y aplicamos tensión.
Para comunicar con el integrado MCP23017 es necesario utilizar el bus I²C y por consiguiente debemos activar el módulo de gestión del bus I²C que como ya sabréis si nos seguís desde hace un tiempo en la instalación predefinida de Raspbian esta deshabilitado.
Recordamos solo que para los artículos sobre la revista dedicados a Raspberry Pi hemos adoptado el sistema operativo Raspbian, en constante actualización y mejora y los instrumentos de gestión remota a través del protocolo SSH, Putty (o Kitty) y WINScp.
Ahora debemos habilitar el driver para la gestione del bus I²C, instalar la librería en Python para la gestión del LCD y realizar un primer programa de prueba para ver que todo funciona como debe.
Recordamos brevemente el proceso para actualizar el sistema operativo y habilitar el driver para el bus I²C. Antes de nada damos los comandos (como usuario "root"):

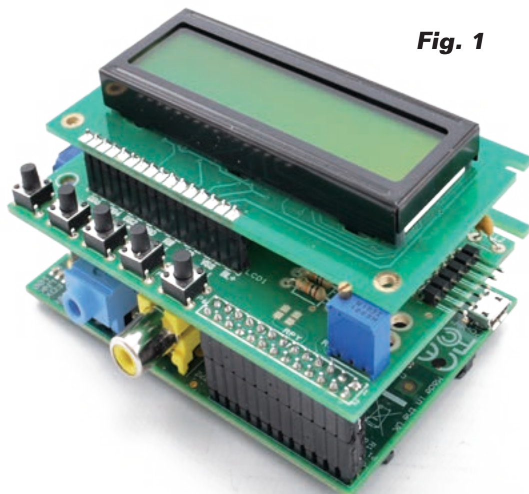


Fig. 1

```
192.168.0.43 - PuTTY
root@raspberrypi:~# nano /etc/modprobe.d/raspi-blacklist.conf
```

Fig. 2

```
192.168.0.43 - PuTTY
  GNU nano 2.2.6    File: /etc/modprobe.d/raspi-blacklist.conf    Modified

# blacklist spi and i2c by default (many users don't need them)

blacklist spi-bcm2708
#blacklist i2c-bcm2708

                              [ Read 4 lines ]
^G Get Help  ^O WriteOut  ^R Read File^Y Prev Page^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page^U UnCut Tex^T To Spell
```

Fig. 3

```
192.168.0.43 - PuTTY
root@raspberrypi:~# nano /etc/modprobe.d/raspi-blacklist.conf
root@raspberrypi:~# modprobe i2c-dev
root@raspberrypi:~#
```

Fig. 4

```
192.168.0.43 PuTTY
root@raspberrypi:~# lsmod
Module                  Size  Used by
i2c_dev                 5620  0
snd_bcm2835            15846  0
snd_pcm                77560  1 snd_bcm2835
snd_seq                53329  0
snd_timer              19998  2 snd_pcm,snd_seq
snd_seq_device          6438  1 snd_seq
snd                    58447  5 snd_bcm2835,snd_timer,snd_pcm,snd_seq,snd_seq_d
evice
snd_page_alloc          5145  1 snd_pcm
evdev                   9426  2
leds_gpio               2235  0
led_class               3562  1 leds_gpio
i2c_bcm2700             3759  0
root@raspberrypi:~#
```

Fig. 5

```
192.168.0.43 PuTTY
root@raspberrypi:~# lsmod
Module                  Size  Used by
i2c_dev                 5620  0
snd_bcm2835            15846  0
snd_pcm                77560  1 snd_bcm2835
snd_seq                53329  0
snd_timer              19998  2 snd_pcm,snd_seq
snd_seq_device          6438  1 snd_seq
snd                    58447  5 snd_bcm2835,snd_timer,snd_pcm,snd_seq,snd_seq_d
evice
snd_page_alloc          5145  1 snd_pcm
evdev                   9426  2
leds_gpio               2235  0
led_class               3562  1 leds_gpio
i2c_bcm2700             3759  0
root@raspberrypi:~#
```

Fig. 6

```
192.168.0.43 - PuTTY
root@raspberrypi:~# nano /etc/modules
```

Fig. 7

```
192.168.0.43 - PuTTY
  GNU nano 2.2.6              File: /etc/modules                     Modified

# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

snd-bcm2835
i2c dev

                              [ Read 7 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Fig. 8

```
192.168.0.43 - PuTTY
root@raspberrypi:~# apt-get install i2c-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  libi2c-dev python-smbus
The following NEW packages will be installed:
  i2c-tools
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 59.5 kB of archives.
After this operation, 223 kB of additional disk space will be used.
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main i2c-tools armhf
3.1.0-2 [59.5 kB]
Fetched 59.5 kB in 1s (55.5 kB/s)
```

Fig. 9

```
192.168.0.43 - PuTTY
After this operation, 223 kB of additional disk space will be used.
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main i2c-tools armhf
3.1.0-2 [59.5 kB]
Fetched 59.5 kB in 1s (55.5 kB/s)
Selecting previously unselected package i2c-tools.
(Reading database ... 61438 files and directories currently installed.)
Unpacking i2c-tools (from .../i2c-tools_3.1.0-2_armhf.deb) ...
Processing triggers for man-db ...
Setting up i2c-tools (3.1.0-2) ...
root@raspberrypi:~#
root@raspberrypi:~#
root@raspberrypi:~# adduser pi i2c
Adding user `pi' to group `i2c' ...
Adding user pi to group i2c
Done.
root@raspberrypi:~#
```

Fig. 10

apt-get update
apt-get upgrade

Para poner utilizable el módulo de gestión del bus I^2C es necesario quitarlo de la blacklist y después "añadirlo" al conjunto de módulos conocidos del kernel.
Abrimos el archivo de configuración que contiene el listado de los módulos blacklisted con el comando (**Fig. 2**):

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

Nano es un editor de texto mínimo que funciona en ambiente terminal.
Eliminamos el módulo I^2C de la blacklist cancelando la línea o, como hemos hecho nosotros, comentándola con un "#" (**Fig. 3**).
Pulsamos Ctrl-X y después Y a la petición de guardar el archivo después de las modificaciones.
Ejecutamos un reboot para hacer efectivas las modificaciones.
Ahora debemos hacer que el módulo módulo "liberado" sea cargado y se convierta en parte integrante del kernel. Para esta operación tenemos dos posibilidades: la primera nos permite cargar el módulo por comando, y tiene validez para todo el tiempo en el cual Raspberry Pi permanece encendido. Al siguiente boot el módulo deberá ser recargado por comando. La segunda nos permite cargar el módulo directamente al boot del sistema operativo y tenerlo disponible a las aplicaciones justo después del boot, condición indispensable en un sistema servidor desatendido.
La primera posibilidad requiere el uso del comando *modprobe*. Escribimos (**Fig. 4**):

modprobe i2c-dev

Podemos ver el buen éxito de la

activación de los driver con el comando que muestra la lista de todos los módulos instalados (**Fig. 5**):

lsmod

Ya que en GNU/Linux todo (o casi) es un archivo, si vamos en la carpeta /dev vemos aparecer los archivos de conexión a los device i2c-0 e i2c-1 (**Fig. 6**).
El comando *modprobe* permite cargar y descargar los módulos en tiempo de ejecución y mantiene sus efectos mientras que Raspberry Pi permanece encendido. En caso de apagado, o solo de reboot, nuestro módulo deberá ser recargado manualmente.
Esta condición no es adecuada para funcionar con una aplicación independiente, que debe funcionar en modo automático. El comando *modprobe*, con la opción *remove*, puede ser utilizado también para desactivar un módulo cargado anteriormente.

modprobe -r i2c-dev

Si se desea que el módulo sea cargado al encender el Raspberry Pi es necesario habilitar la carga permanente del driver I²C que se realiza modificando oportunamente el archivo de configuración /etc/modules, que contiene la lista de los driver de carga en el momento del boot.
En caso contrario debemos recordarnos de cargar el módulo en cada encendido con *modprobe*.
Para modificar el archivo podemos usar el comando:

nano /etc/modules

y añadir una nueva línea al archivo de configuración que contiene (**Fig. 7**)

i2c-dev

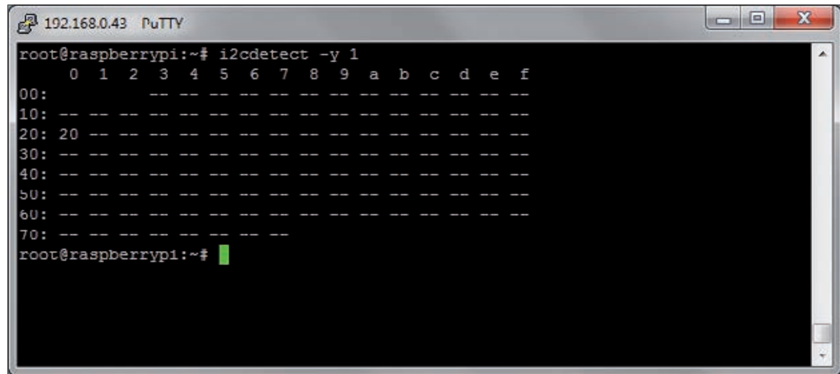


Fig. 11

Pulsar CTRL X y después Y para guardar las modificaciones en el archivo y salir (**Fig. 8**).
Ahora instalamos el paquete i2c-tools que nos proporciona una serie de funciones utilizables en la línea de comando para verificar el funcionamiento del bus I²C (**Fig. 9**):

apt-get install i2c-tools

Añadimos nuestro usuario *pi* al grupo I²C (**Fig. 10**):

adduser pi i2c

Ejecutamos el reboot de Raspberry Pi para activar las nuevas configuraciones con el comando
reboot

Después que Raspberry Pi se ha reiniciado y os habéis reconectado con Putty o Kitty, comprobar si sobre el bus I²C esta visible el conversor ADC con el comando:

i2cdetect -y 0 per Raspberry Pi rev. 1
o
i2cdetect -y 1 per Raspberry Pi rev. 2

Deberéis obtener un resultado parecido a aquel visible en **Fig. 11** donde la dirección 0x20 identifica el integrado MCP23017.
Ahora podemos instalar la librería de soporte al LCD. Utilizaremos la librería puesta a disposición con licencia BSD por Adafruit Industries en el repositorio GitHub.
El mejor procedimiento para descargarla es utilizar el instrumento de gestión de las versiones de software "git".
Para instalar "git" usamos el comando:

apt-get install git

después vamos a la carpeta home con el comando:

cd /home

creamos una carpeta para nuestro proyecto, por ejemplo:

mkdir LCD

nos posicionamos en la carpeta con el comando:

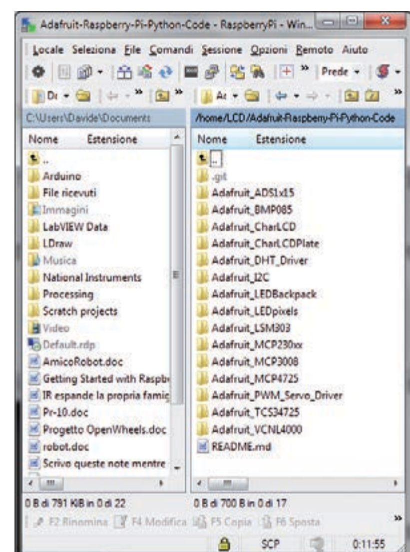
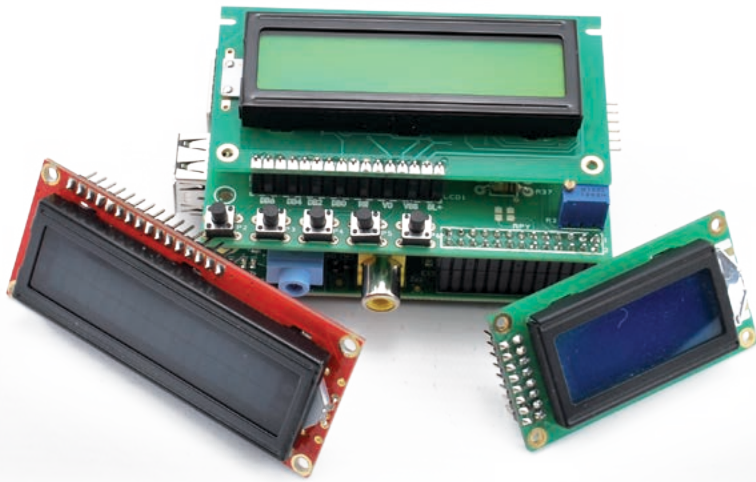


Fig. 12



CD LCD

y descargamos las librerías Python realizadas por Adafruit con el comando:

git clone https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git

finalmente nos posicionamos en la carpeta que contiene la librería para la gestión del LCD con los comandos:

cd Adafruit-Raspberry-Pi-Python-Code

Podemos ver el contenido de la carpeta posicionándonos sobre la misma con WinSCP (**Fig. 12**).
Nos posicionamos en la carpeta que contiene la librería Adafruit_CharLCDPlate con el comando:

cd Adafruit_CharLCDPlate

Instalamos la librería para la gestión del protocolo I²C con el lenguaje Python:

apt-get install python-smbus

En este punto estamos casi listos para comprobar el funcionamiento del shield LCD.
En la librería Adafruit_CharLCDPlate, los pines GPA6 y GPA7 se utilizan para la gestión de los colores de un LCD en color RGB. En nuestro caso, utilizando LCD monocromáticos, dejamos disponibles esos pines. Para poderlos utilizar es necesario modificar la librería de manera que se pueda utilizare los pines, por ejemplo, como entradas.
Para comprender las modificaciones, hacer referencia a las informaciones contenidas en el recuadro dedicado al integrado MCP23017. La librería contiene el código que expresa la clase Adafruit_CharLCDPlate para la gestión de los display LCD. En el constructor de la clase hemos modificado el ajuste inicial de los registros para hacer los pines GPA5, GPA6 y GPA7 de entrada, con resistencia de pull up interna y que funcionen en lógica invertida (valor "1" cuando el pin está a masa).
Podéis descargar la librería modificada en la dirección *www.nuevaelectronica.com.* En particular hemos modificado las siguientes líneas:

```
riga 99  [ 0b11111111,   # IODIRA    R+G LEDs=outputs, buttons=inputs orig. 0b00111111
riga 101   0b11111111,   # IPOLA     Invert polarity on button inputs orig. 0b00111111
riga 112   0b11111111,   # GPPUA     Enable pull-ups on buttons orig. 0b00111111
```

En la línea 425 hemos modificado el método *backlight* de manera que eliminamos el uso de los pines del banco GPA para la gestión del color.

```
  def backlight(self, color):
      c = ~color
#     self.porta = (self.porta & 0b00111111) | ((c & 0b011) << 6)
      self.portb = (self.portb & 0b11111110) | ((c & 0b100) >> 2)
      # Has to be done as two writes because sequential operation is off.
#     self.i2c.bus.write_byte_data(
#       self.i2c.address, self.MCP23017_GPIOA, self.porta)
      self.i2c.bus.write_byte_data(
        self.i2c.address, self.MCP23017_GPIOB, self.portb)
```

Finalmente (línea 436) hemos añadido tres métodos para la lectura de los pines "avanzados" individuales.

```
  # Read and return bitmask of pin A5
  def buttonpin5(self):
      return (self.i2c.readU8(self.MCP23017_GPIOA) >> 5) & 1

  # Read and return bitmask of pin A6
  def buttonpin6(self):
      return (self.i2c.readU8(self.MCP23017_GPIOA) >> 6) & 1

  # Read and return bitmask of pin A7

  def buttonpin7(self):
   return (self.i2c.readU8(self.MCP23017_GPIOA) >> 7) & 1
```

Hemos preparado un programa de prueba modificando el programa de test incluido en la carpeta de la librería, obteniendo el programa visible en el **Listado 1**.
El programa, pulsando los botones presentes en el shield, permite obtener informaciones sobre el funcionamiento y el estado del sistema, sintetizados en el display LCD.
Copiar el programa en un archivo de nombre LCDProva.py, o descargarlo desde la web *www.nuevaelectronica.com.*
Recordar lanzar el comando:

modprobe i2c-dev

Posicionaros en la carpeta Adafruit_CharLCDPlate y lanzar el programa con el comando:

python LCDProva.py

Deberíais ver el mensaje de apertura sobre el display LCD y después el menú de los pulsadores.
Si no veis nada o la pantalla tiene un contraste excesivo, podéis regularlo mediante el potenciómetro R1

Listado 1

```
#!/usr/bin/python

from time import sleep
from Adafruit_CharLCDPlate import Adafruit_CharLCDPlate
from subprocess import *

cmd_ip = "ip addr show eth0 | grep inet | awk '{print $2}' | cut -d/ -f1"
cmd_cpu = "mpstat | awk '$11 ~ /[0-9.]+/ { print 100 - $11}'"
cmd_dfh = "df -h | grep /dev/root | awk '{ print $2 }'"
cmd_dfh_p = "df -h | grep /dev/root | awk '{ print $5 }'"
cmd_ps = "ps -ef | wc -l"

def run_cmd(cmd):
        p = Popen(cmd, shell=True, stdout=PIPE)
        output = p.communicate()[0]
        return output

class CPU(object):
   def __init__(self):
      """Init a CPU status object"""
      stat_fd = open('/proc/stat')
      stat_buf = stat_fd.readlines()[0].split()

        self.prev_total = float(stat_buf[1]) + float(stat_buf[2]) + float(stat_buf[3]) + float(stat_buf[4]) +
float(stat_buf[5]) + float(stat_buf[6]) + float(stat_buf[7])
      self.prev_idle = float(stat_buf[4])
      stat_fd.close()

   def usage(self):
      """return the actual usage of cpu (in %)"""
      stat_fd = open('/proc/stat')
      stat_buf = stat_fd.readlines()[0].split()
       total = float(stat_buf[1]) + float(stat_buf[2]) + float(stat_buf[3]) + float(stat_buf[4]) + float(stat_
buf[5]) + float(stat_buf[6]) + float(stat_buf[7])
      idle = float(stat_buf[4])
      stat_fd.close()
      diff_idle = idle - self.prev_idle
      diff_total = total - self.prev_total
      usage = 1000.0 * (diff_total - diff_idle) / diff_total
      usage = usage / 10
      usage = round(usage, 1)
      self.prev_total = total
      self.prev_idle = idle
      return usage


   def memUsage():
      free_fd = os.popen('free -b')
      free_buf = free_fd.readlines()[1].split()
      usage = (float(free_buf[2]) / (float(free_buf[1]))) * 100
      usage = round(usage, 1)
      return usage
```

(Continua)

Listado 1 – CONTINUACIÓN

```
cpu = CPU()

# Initialize the LCD plate.  Should auto-detect correct I2C bus.  If not,
# pass '0' for early 256 MB Model B boards or '1' for all later versions
lcd = Adafruit_CharLCDPlate()

# Clear display and show greeting, pause 1 sec
lcd.clear()
lcd.message("  RaspberryPi\n ElettronicaIn")
sleep(5)
lcd.clear()
lcd.message("P1=M P2=IP P3=CPU\nP4=DSK  P5=Proc.")

# Poll buttons, display message & set backlight accordingly
btn = ((lcd.LEFT  , 'Pulsante 1' , lcd.BLUE),
       (lcd.UP    , 'Pulsante 2' , lcd.BLUE),
       (lcd.DOWN  , 'Pulsante 3' , lcd.BLUE),
       (lcd.RIGHT , 'Pulsante 4' , lcd.BLUE),
       (lcd.SELECT, ''           , lcd.ON))
prev = -1

while True:
    pin5 = lcd.buttonpin5()
    pin6 = lcd.buttonpin6()
    pin7 = lcd.buttonpin7()
    pinT = pin5 * 100 + pin6 * 10 + pin7
    pinS = ("000" + str(pinT))[-3:]

    for b in btn:
        if lcd.buttonPressed(b[0]):
            if b is not prev:
                print b[0]
                if b[0] == 0:
                   lcd.clear()
                   lcd.backlight(lcd.ON)
                   ps = run_cmd(cmd_ps)
                   lcd.message('N.Processi %s\nP1=Menu  pin %s' % (ps , str(pinS) ))
                elif b[0] == 1:
                   lcd.clear()
                   lcd.backlight(lcd.ON)
                   ipaddr = run_cmd(cmd_ip)
                   lcd.message('IP %s \nP1=Menu  pin %s' % ( ipaddr , str(pinS) ))
                elif b[0] == 2:
                   lcd.clear()
                   lcd.backlight(lcd.ON)
                   dfh = run_cmd(cmd_dfh)
                   dfh_p = run_cmd(cmd_dfh_p)
                   lcd.message('DSK %s USED %s\nP1=Menu  pin %s' % (dfh.strip(), dfh_p.strip() , str(pinS)))
                elif b[0] == 3:
                   lcd.clear()
                   lcd.backlight(lcd.ON)
                   cpu1 = float(cpu.usage())
                   lcd.message('CPU %.2f%%\nP1=Menu  pin %s' % (cpu1 , str(pinS)))
                elif b[0] == 4:
                   lcd.clear()
                   lcd.backlight(lcd.ON)
                   lcd.message("P1=M P2=IP P3=CPU\nP4=DSK  P5=Proc.")
                prev = b
            break
```

hasta obtener un resultado satisfactorio.
Como podéis ver, el listado es muy simple: la mayor parte del trabajo es desarrollado por los métodos proporcionados por la librería. A la apertura del programa importamos la clase sleep, para gestionar los retrasos de ejecución, la librería Adafruit_CharLCDPlate y la librería *subprocess*. La primera instrucción del programa llama un método que permite determinar a qué revisión pertenece el Raspberry Pi en nuestra posesión y ajustar en consecuencia el bus I2C, 0 o 1. Después se envía el mensaje de apertura sobre el display y sucesivamente, con

Fig. 13

Fig. 14

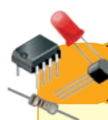
Fig. 15

Fig. 16

Fig. 17

un retardo de un segundo, el mensaje de menú. Finalmente en el ciclo *while* se leen los valores de los tres pines de entrada y después se captura qué pulsador esta accionado para ejecutar la función correspondiente con la presentación de los resultados sobre el display.
Los valores de los pines se presentan en el display abajo a la derecha de cada comando.
Con fines didácticos, en este programa de ejemplo, hemos querido presentar el resultado de algunos comandos que permiten recoger informaciones sobre el estado y/o el funcionamiento del sistema. Es inútil recordar que los mismos comandos pueden ser tecleados en la ventana terminal como comandos normales, obteniendo los mismos resultados.
Los comandos son ejecutados lanzando procesos externos al programa en ejecución y recuperando el resultado al final de la ejecución (**Listado 1**).
En particular la presión del pulsador P2 llama al menú; el pulsador P3 muestra la dirección IP de Raspberry Pi y el estado de los tres pin de entrada adjuntos, P4 el uso de CPU, P5 el espacio ocupado sobre la SD Card o su disco - tanto en GB como en porcentaje - y P6 el número de procesos en ejecución sobre Raspberry Pi. Para modificar el estado de los pines de entrada podéis utilizar experimentalmente y con mucha atención un cable con los terminales hembra/hembra a los extremos. Conectar un extremo del cable al terminal a masa y el otro terminal a uno de los pin de entrada. Pulsar uno de los pulsadores desde P3 a P6 y deberíais ver el estado de los pines abajo a la derecha (Fig 13, 14, 15, 16, 17).
En los próximos números os presentaremos otros shield de expansión y profundizaremos los modos de diseño y realización de programas que respeten el esquema de arquitectura descrito al inicio del artículo.

(178025)



el MATERIAL

Todos los componentes necesarios para realizar el proyecto descrito en estas páginas son fácilmente localizables en el mercado; el diseño del circuito impreso y el firmware pueden ser descargados gratuitamente desde la web de la revista (www.nuevaelectronica.com). El shield LCD esta también disponible como kit de montaje (cod. FT1074K) al precio de 17,00 Euros; el kit se suministra con el display LCD 16x2 retroiluminado (código ACM1602B-FL-YBW). Con este kit pueden ser utilizados también otros display como el modelo LCD 16x2 retroiluminado blanco/negro cod. LCD16x2WB (15,00 Euros) y el display LCD 8x2 retroiluminato azul cod. LCD8x2BN (7,50 Euros).

Precios IVA incluido sin gastos de envío.
Puede hacer su pedido en:
www.nuevaelectronica.com
pedidos@nuevaelectronica.com