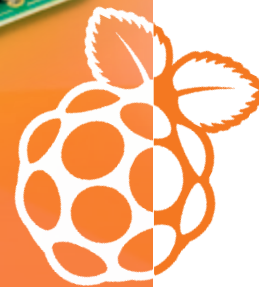
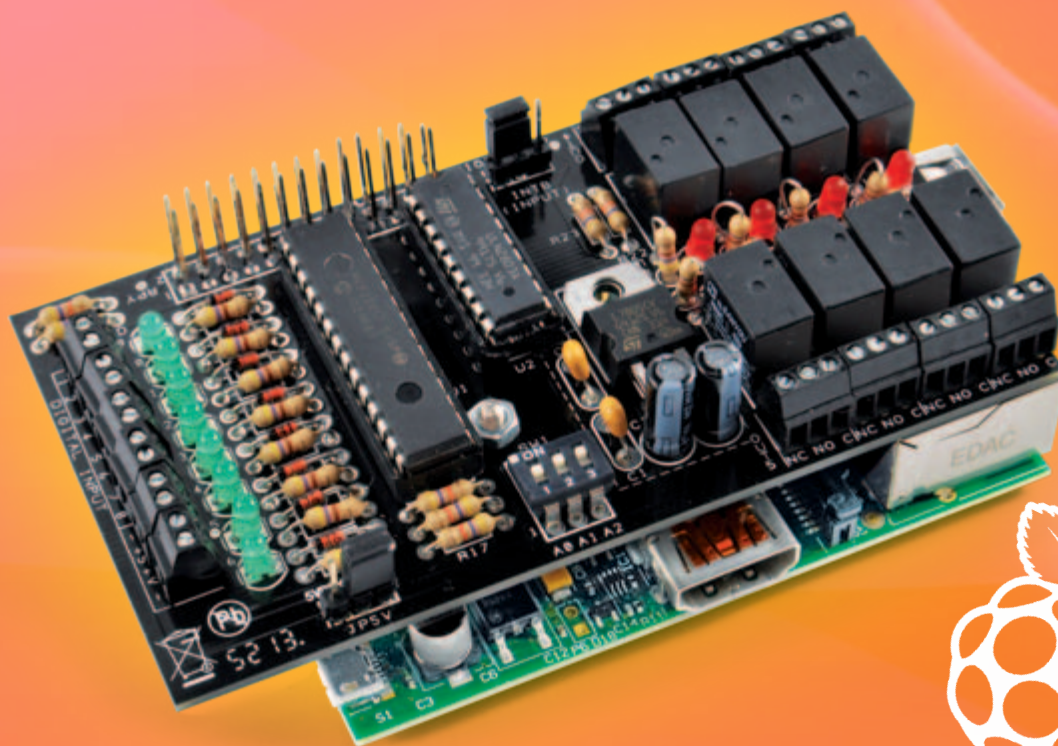


SHIELD I²C DE EXPANSIÓN E/S PARA RASPBERRY PI



Controlamos hasta 64 salidas a relé y leemos un máximo de 64 entradas digitales utilizando solo dos líneas de la Raspberry Pi: las del bus I²C.

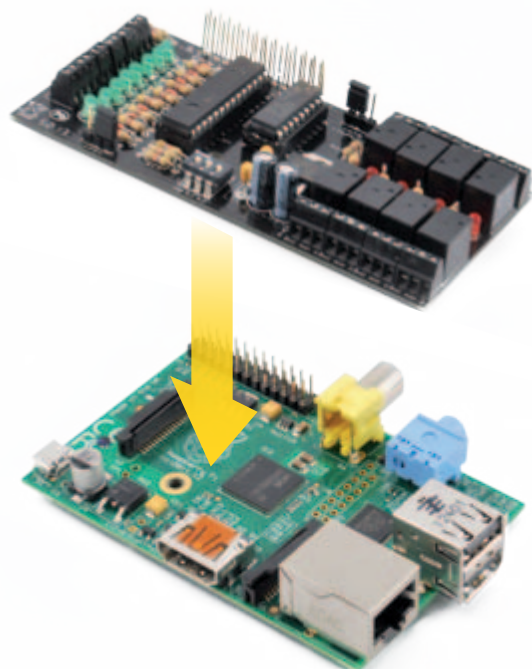
Presentamos un nuevo shield para Raspberry Pi basado en el integrado MCP23017 que permite aumentar el número de entradas/salidas digitales para ponerlas a disposición de nuestras aplicaciones. El shield de expansión aquí descrito permite disponer de ocho entradas y otras tantas salidas digitales. El estado de las ocho entradas viene

representado por un LED para cada una. Cada una de las ocho salidas controla un relé al que se pueden conectar cargas externas. En cada una de las salidas está presente también un LED, que hace visible su estado. Para acceder a las entradas/salidas del shield se utiliza el bus I²C. El direccionamiento del shield es configurable mediante un DIP Switch

a tres posiciones, permitiendo la conexión y la gestión de un máximo de ocho shield simultáneamente, cada uno con una dirección diferente; esto permite expandir el sistema hasta un total de 64 entradas y 64 salidas controlables individualmente. En este artículo os presentamos también la librería Wiring-PI2, una librería Open Source en

MARCO MAGAGNIN

licencia GNU LGPLv3 utilizable en Python, que permite acceder a las entradas individuales y manejarlas de un modo decididamente simple. Recordamos que el shield de expansión presentado en este artículo convive tranquilamente con los otros shield que presentaremos y también con algunos diseñados por terceras partes. Una gran cantidad de posibilidades de conexiones que permiten la realización de aplicaciones capaces de gestionar ambientes también notablemente complejos. La realización de aplicaciones de este tipo requiere un salto de nivel respecto a los proyectos que hemos presentado hasta ahora. Principalmente es necesario proyectar y realizar arquitecturas software capaces de disociar el uso físico de dispositivos externos con respecto a los módulos de procesamiento reales. En particular, necesitamos realizar programas capaces de reaccionar a los "eventos" externos generados por las entradas y los sensores a ellas conectados que, generalmente, se presentan en modo asíncrono. Es también

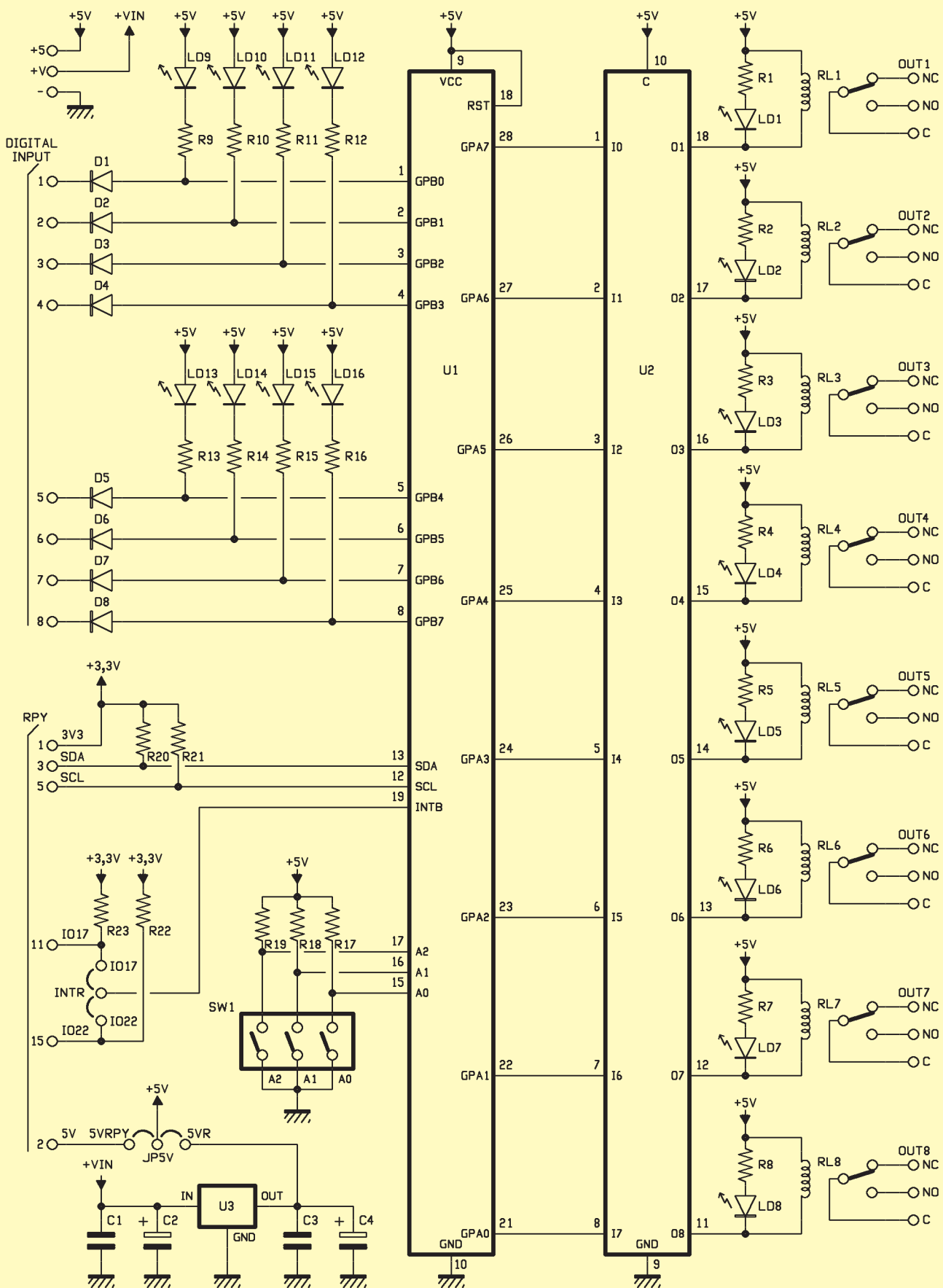


necesario proyectar correctamente los programas que gestionan los periféricos que inciden sobre el mismo recurso de comunicación. Por ejemplo, diferentes aplicaciones pueden querer acceder a diferentes periféricos que utilizan el mismo bus I²C: es precisamente el caso del shield de expansión que presentamos en este artículo. Si una aplicación está ya utilizando el bus, para leer o escribir el estado de un E/S de su "competencia", las otras aplicaciones que requieren utilizar el mismo bus para gestionar sus propias E/S, encontrándolo ocupado (busy), no podrán hacer otra cosa que pasar a error. La primera solución que podría venir en mente sería realizar una única enorme aplicación que gestione todas las funcionalidades en modo monolítico, un poco como un enorme microcontrolador. Es fácil de entender que este modo de proceder comporta más aspectos negativos que positivos: rigidez de la aplicación, dificultad de diseño, realización, mantenimiento y actualización, dificultad de temporización tras las diferentes partes, fragilidad (si una parte, aunque sea insignificante, acaba en error, cae toda la aplicación). Para resolver una exigencia de aplicación de este tipo es necesario aplicar arquitecturas capaces de soportar aplicaciones concurrentes y colaborativas y de gestionar las comunicaciones entre los diversos actores, basado en servidores que orquestan el uso de recursos para compartir y donde las diferentes funcionalidades aplicativas tienen que ser realizadas como clientes que requieren el uso de los recursos "centralizados" mediante peticiones basadas sobre protocolos compartidos. Hay muchas alternativas posibles: socket TCP/IP, funcionalidad base de datos, simples colas y cosas así. Lo importante es que los mecanis-

mos de comunicación sean capaces de gestionar correctamente el fenómeno, por ejemplo generando código o gestionando "semáforos" y "bloques" para coordinar correctamente las peticiones de los clientes. Los clientes, finalmente, cuando tienen necesidad de acceder a uno de los recursos gestionados de forma centralizada, no tienen que acceder directamente pero tienen que utilizar el canal de comunicación predefinido por aquel recurso, con el protocolo y la modalidad prevista. En resumen, en el modo embebido el diseño no debe ser referido a la misma aplicación pero debe tener en cuenta la arquitectura que se quiere realizar, en su totalidad. ¿Mejor o peor respecto a los microcontroladores? Esta pregunta no tiene una respuesta definitiva. Depende de lo que se quiere realizar y de los requisitos asociados. Seguramente, en un futuro, nos encontraremos siempre más en la necesidad de integrar y hacer colaborar estos dos ambientes, junto a otros como FPGA y PLC. Desde los próximos números de la revista empezaremos a afrontar este tipo de problemáticas, quizás más conectados a la disciplina "informática" que a la electrónica, pero quizás, como habíamos sostenido muchas veces, debemos lidiar con la necesidad, para quién realiza aplicaciones basadas sobre dispositivos electrónicos destinados al mundo "embebido", de ampliar las competencias propias también al mundo de la informática y de la gestión de las redes de comunicación.

ESQUEMA ELÉCTRICO

El circuito está proyectado alrededor del integrado MCP23017, de Microchip, que ofrece 16 entradas/salidas digitales direccionadas mediante el bus I²C. De esta forma, podemos direccionar las ocho



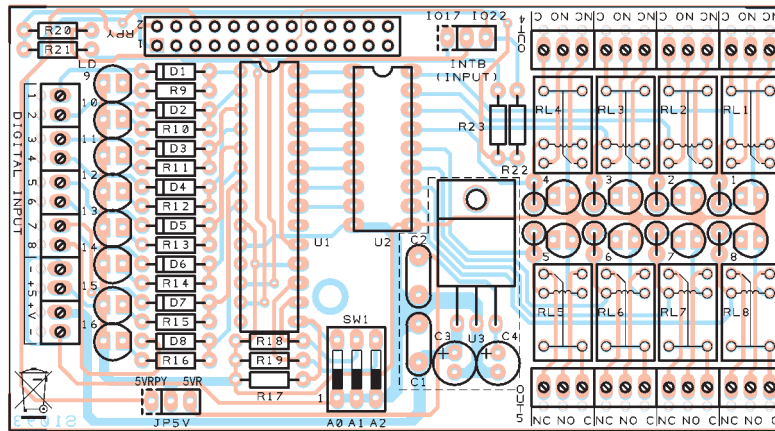
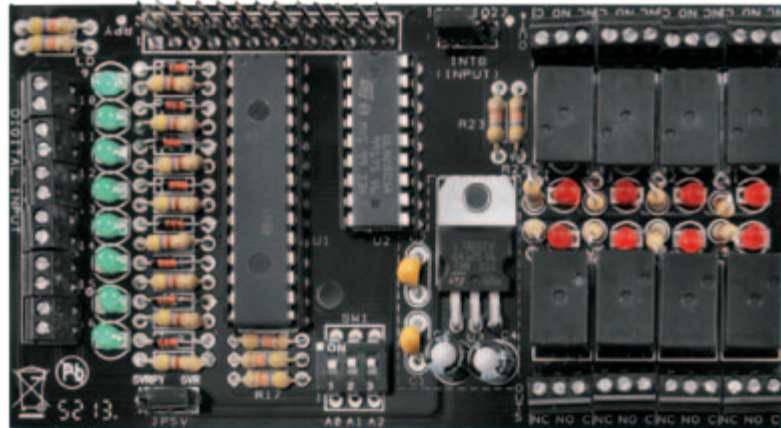
[plano de **MONTAJE**]

Lista de materiales:

- R1÷R16: 470 ohm
- R17÷R23: 4,7 kohm
- C1, C2: 100 nF multicapa
- C3, C4: 100 µF/16 V electrolítico
- D1÷D8: 1N4148
- LD1÷LD8: LED 3 mm rojo
- LD9÷LD16: LED 3 mm verde
- SW1: Dip-Switch 3 vías
- RL1÷RL8: Relé miniatura 5V
- U1: MCP23017-E/SP
- U2: ULN2803
- U3: 7805

Varios:

- Zócalo 14+14
- Zócalo 9+9
- Tira de 3 pines macho (2 pz.)
- Jumper (2 pz.)
- Conector hembra 2x13 cod. CTF/13+13
- Terminal 2 polos paso 2,54 mm (6 pz.)
- Terminal 3 polos paso 2,54 mm (8 pz.)
- Torreta M/F 18 mm
- Tuerca 3 MA
- Tornillo 8 mm 3 MA
- Circuito impreso



entradas y las ocho salidas digitales de nuestro shield utilizando solo los dos pines del conector de Raspberry Pi que forman el bus I²C. Obviamente el bus no es utilizado en modo exclusivo por nuestro shield, se pueden conectar otros periféricos siempre que estén configurados con direcciones diferentes.

El integrado MCP23017 (U1 en el mismo esquema eléctrico) pone a disposición del usuario 16 líneas de E/S (I/O) subdivididas en dos bancos de ocho, denominados GPA y GPB. Cada una de las líneas del integrado es configurable individualmente como entrada o como salida. En nuestro shield, el banco GPA está completamente dedicado a la gestión de las sali-

das a relé mientras que el banco GPB está dedicado a la gestión de las entradas digitales. Las líneas de salida digital, provenientes del banco GPA del integrado MCP23017, están unidas a los pines de entrada del integrado U2, un ULN2803, especialmente diseñado para controlar cargas inductivas, como las bobinas de los relés.

En el interior del integrado, cuyo esquema es visible en la Fig. 1, están presentes ocho etapas Darlington conectadas con los emisores en común (conectadas en el pin 9) y los colectores a las salidas, cada una de las cuales puede proporcionar hasta 500 mA. Como inciso, aunque no es nuestro caso, para controlar cargas que requie-

ren corrientes superiores a los 500 mA, pueden ser conectados en paralelo más etapas del integrado. En el interior del chip están presentes también diodos de protección, necesarios para controlar cargas inductivas. Aplicaciones típicas son el control de motores, el accionamiento de relés, solenoides, lámparas, etc. Las entradas 1-8, caracterizadas por una alta impedancia de entrada, pueden ser excitadas directamente desde los puertos de E/S de un microcontrolador o de un integrado como el MCP23017. Pero volvamos a nuestro esquema eléctrico. Analicemos una sola línea de salida de las 8 disponibles en cuanto que el esquema de las otras 7 se repite de forma idéntica. Cuan-

do, por ejemplo, el pin GPA7 se encuentra a nivel lógico alto (más de 2,5V hasta 5V), la base del transistor Darlington conectada con la entrada 1 del integrado ULN2803, es polarizada mediante el divisor resistivo RA, RB y RC, visible en el esquema de la Fig. 1. En esta condición el transistor está en saturación y su colector se lleva a un potencial de pocas centenas de milivoltios, dejando fluir la corriente en la bobina del relé. El estado de conducción del transistor produce también el encendido del LED LD1, polarizado por medio de la resistencia R1. Cuando el pin GPA7 se encuentra a nivel lógico bajo, el Darlington correspondiente se encuentra en estado de corte, y entonces su colector, como un interruptor abierto, no permite el paso de corriente en la bobina del relé. También el LED, al no estar alimentado, no se enciende. El diodo DA sirve para suprimir la extracorrente inversa generada de la bobina del relé cuando pasa del estado excitado al estado de reposo, tensiones que podrían dañar el transistor. En los terminales de tornillo de la tarjeta están disponibles todos los terminales del relé: el terminal central C, el terminal “normalmente cerrado” NC y el “normalmente abierto” NO. La conexión de los terminales es visible en el plano de montaje. Como ya comentamos, las entradas digitales están conectadas a los pines del banco GPB del integrado MCP23017. También en este caso la descripción de un módulo vale para todos los demás. Tomemos la entrada digital 1, que está conectada al pin GPB0. La resistencia de pull-up R9, en reposo, mantiene la entrada a nivel alto. De hecho está conectada al positivo de la tensión de alimentación a través del LED LD9 que, en este estado, permanece apagado. El diodo D1

sirve para proteger la entrada del integrado, limitando a 5V la tensión máxima que puede llegar al pin de entrada. En efecto, aplicando una tensión superior, D1 no conduce más y el pin GPB0 “ve” solamente los 5V de la alimentación, por efecto de la presencia de R9/LD9. Esta configuración permite hacer funcionar el circuito con niveles lógicos en entrada de 12 voltios o más, salvaguardando siempre las entradas. Todas las entradas digitales tienen la masa en común, que corresponde al negativo de alimentación; negativo y positivo están presentes también sobre dos contactos del terminal, de manera que están fácilmente disponibles los 5V con los que proporcionar alimentación a los circuitos externos. El shield requiere una alimentación externa con una tensión comprendida entre 9 y 12V, separada respecto de Raspberry Pi, ya que la salida a 5V de esta última no es capaz de proporcionar la corriente necesaria a la alimentación del shield. El circuito integrado U2 (LM7805) regula y estabiliza la tensión de entrada para proporcionar los 5V necesarios para la tarjeta. El shield, sin embargo, es capaz de alimentar Raspberry Pi. Si se quiere mantener separadas las alimentaciones del shield y de la tarjeta Raspberry Pi, el puente JP5V tiene que posicionarse entre el pin central y el pin 5VR. Para hacer que el alimentador del shield alimente también la tarjeta Raspberry Pi, el puente JP5V tie-

ne que estar posicionado entre el pin central y el pin 5VRPY. La salida INTB, que permite gestionar la interrupción proveniente de las entradas del banco GPB puede conectarse, mediante el puente INTR, a GPIO22 o GPIO17 de Raspberry Pi, de manera que esté disponible a eventuales aplicaciones. El Dip Switch conectado a los pines A0, A1 y A2 del integrado, permite definir la dirección I²C asignada al integrado. En el plano de montaje se puede ver la disposición de los componentes. La única sugerencia, aparte de las precauciones normales, es posicionar el conector doble denominado RPY de modo que distancie el circuito impreso del shield del de la Raspberry Pi para que nos permite el montaje “en paquete” en el caso en que se

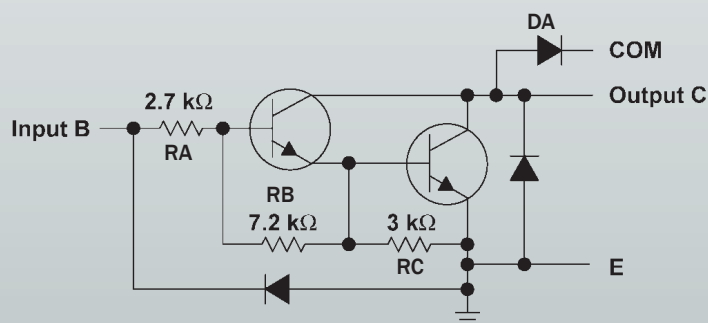
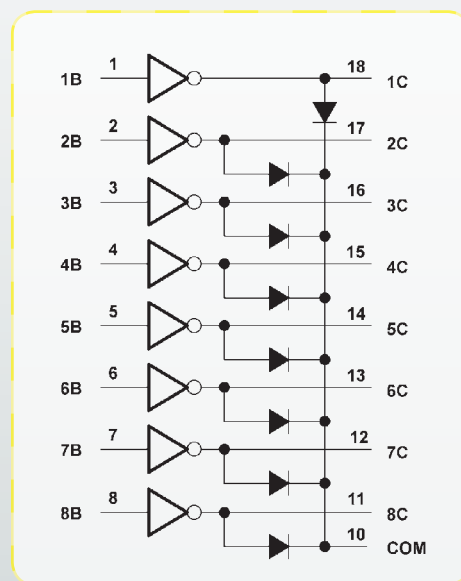


Fig. 1



Adaptador con terminales para Raspberry Pi

El shield adaptador para Raspberry Pi (FT1104) simplifica el camino a quien desea experimentar las posibilidades del mundo embebido. Permite realizar circuitos o conectar periféricos externos sin necesidad de realizar circuitos impresos o de tener que utilizar el soldador. El shield está dotado de terminal a 26 polos, tira de pines M/H de paso 2,54 mm y conector macho de 26 pines, que llevan al exterior todas las líneas E/S presentes en el conector de 26 pines de la placa Raspberry Pi. El conector pasante de 26 contactos del shield permite instalar sobre él otros shields en “cascada” para realizar complejas aplicaciones multifuncionales. Utilizando el cable plano incluido en el kit del shield y el adaptador FT1072K (no incluido), es posible conectar de manera muy simple la Raspberry Pi a una breadboard común sobre la que haya realizado la propia aplicación. La ventaja principal de esta solución es la gran flexibilidad de uso al tiempo que una mayor seguridad de uso de los GPIO, que quedan protegidos del riesgo de cortocircuito o contactos accidentales entre los pines o con otras partes metálicas de la Raspberry Pi. Estos productos están disponibles en nuestra web (www.nuevaelectronica.com).

Raspberry Pi son simplemente pasantes.

USO PRÁCTICO DEL SHIELD

Conectamos el shield sobre la tarjeta Raspberry Pi, prestando atención a la correspondencia de los pines de los respectivos conectores y verificando que la parte inferior del shield no esté en contacto con los conectores USB o Ethernet. En caso de duda, protegemos los conectores mismos con la cinta aislante. Conectamos un alimentador de 9 Vcc a los terminales de alimentación del shield (+V y -) y alimentamos la tarjeta Raspberry Pi. Para comunicar con el integrado MCP23017 es necesario utilizar el bus I²C y por consiguiente debemos activar el módulo de gestión del mismo bus que, como ya sabéis, si nos seguís en la revista, en la instalación predefinida de Raspbian, está deshabilitada. Para quién nos sigue habitualmente, esta operación no supone un problema; para quién es nuevo en el ambiente Raspbian, el sistema operativo GNU/Linux de Raspberry Pi, le dirigimos a las indicaciones que aparecen en el libro “Raspberry Pi il mio primo Linux embedded” (en italiano, vendido por www.futurashop.it) o la documentación disponible en www.raspbian.org. En el artículo describimos solo las operaciones

quiera utilizar conjuntamente a otros shield de expansión o, por citar otro, el shield GSM que presentaremos en breve. La posibilidad de “amontonar” más tarjetas viene dada por el hecho que el integrado MCP23017 está conectado al GPIO de la Raspberry Pi únicamente con los pines correspondientes a la comunicación I²C y a las eventuales alimentaciones. Todos los otros pines de la

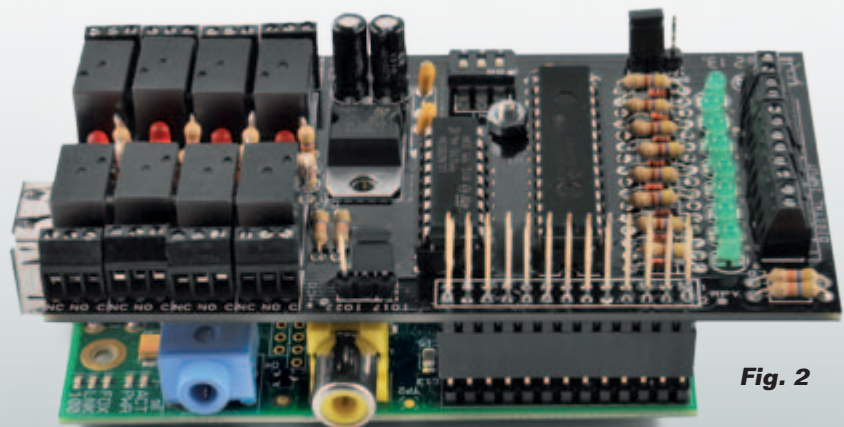


Fig. 2

específicas para utilizar el shield de I/O. Recordamos que para los artículos de la revista dedicada a Raspberry Pi hemos adoptado el sistema operativo Raspbian (en constante actualización y mejora) y a los instrumentos de gestión remota, mediante protocolo SSH, utilizando PuTTY (o KiTTY) y WINScp para la conexión. Entonces, posicionamos todos los DIP switch de direccionamiento del shield en "ON" (GND), el jumper JP5V en la posición 5VR (alimentación separada), montamos el shield sobre el conector de Raspberry Pi, realizamos todas las conexiones y aplicamos tensión. ¿Nada de humo? ¡Bien, estamos en el buen camino!

Ahora debemos habilitar el driver para la gestión del bus I²C, instalar la librería Python para la gestión del integrado MCP23017 y realizar los primeros programas para comprobar que todo funciona como debe ser. Recordemos brevemente el proceso para actualizar el sistema operativo y habilitar el driver para el bus I²C: primero utilizamos PuTTY para abrir una ventana de terminal, nos registramos como el usuario "root" y tecleamos los comandos necesarios para actualizar la base de datos de los paquetes del sistema operativo:

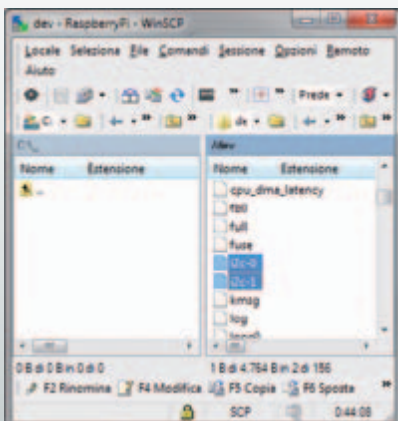


Fig. 7

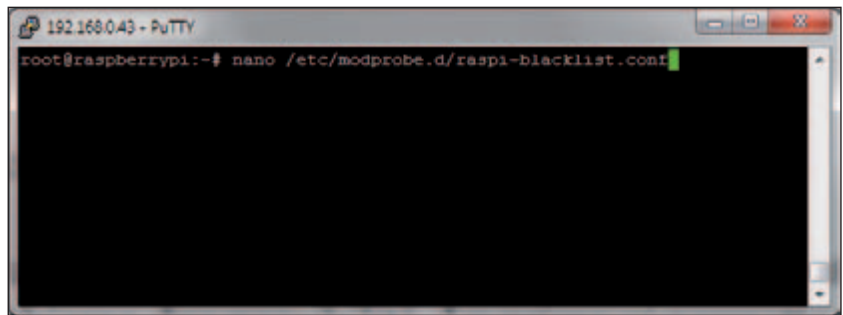


Fig. 3

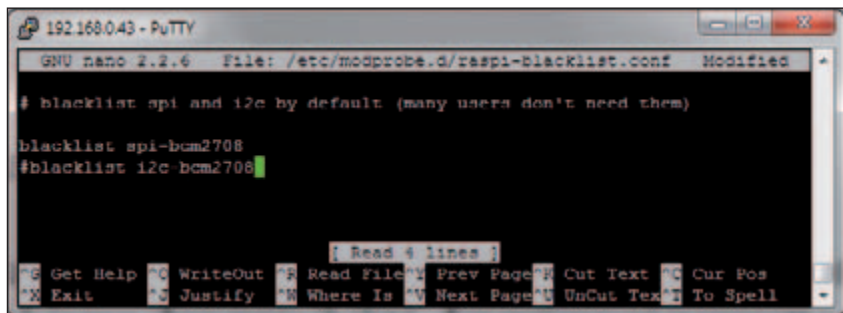


Fig. 4

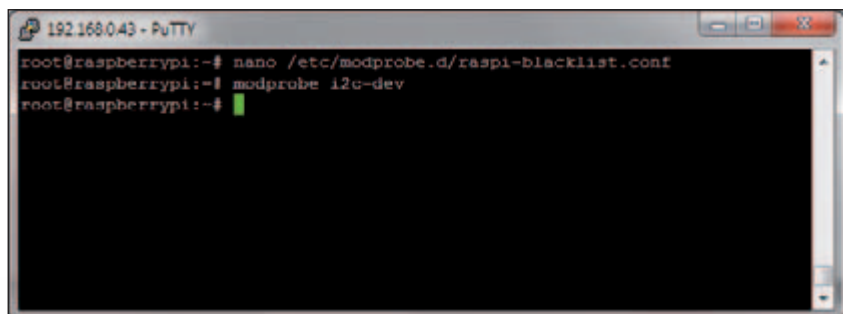


Fig. 5

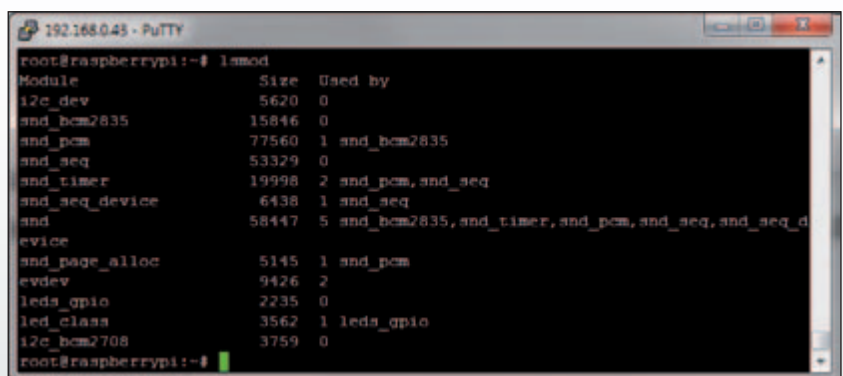


Fig. 6

apt-get update
apt-get upgrade
 Para... resucitar y hacer utilizable el módulo de gestión del bus I²C es necesario quitarlo de la *blacklist* que lo tiene "prisionero" y después "añadirlo" al conjunto de

módulos reconocidos por *kernel*. Abramos el archivo de configuración que contiene el elenco de módulos *blacklisted* (en la lista negra), con el comando (Fig. 3):
nano /etc/modprobe.d/raspi-blacklist.

Fig. 8



Fig. 9

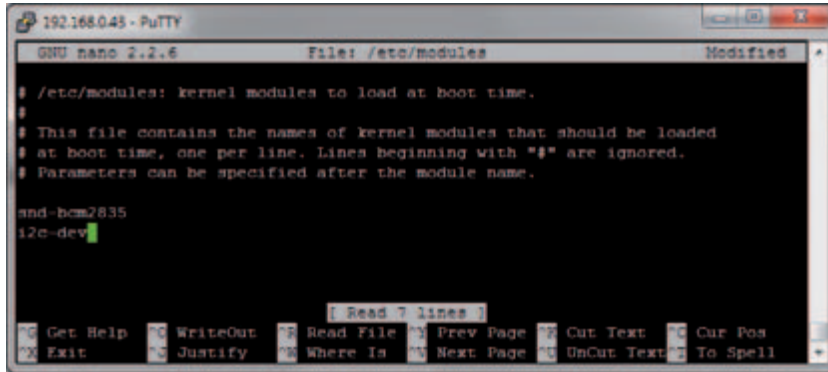


Fig. 10

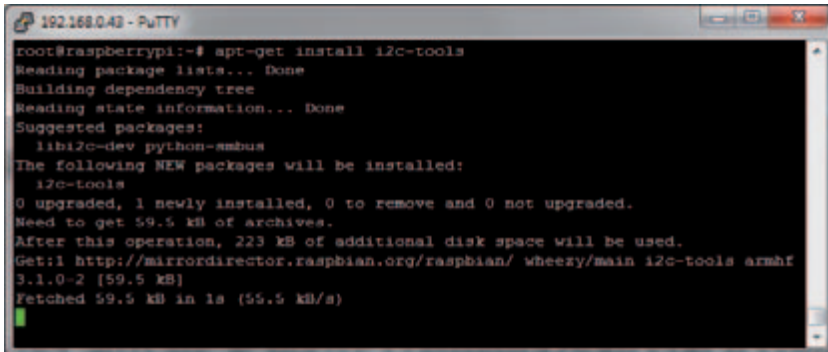


Fig. 11

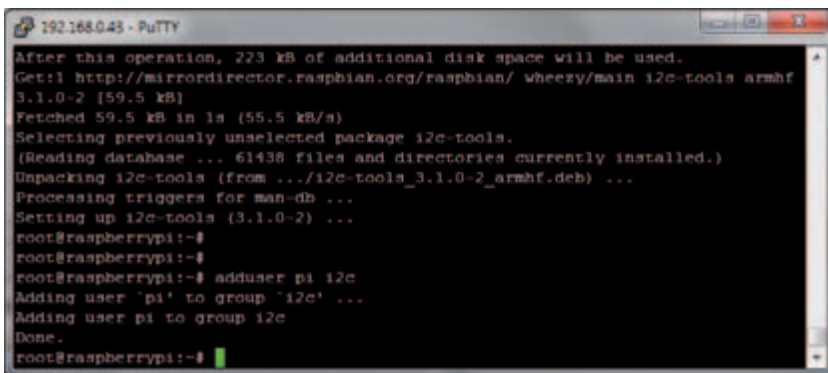
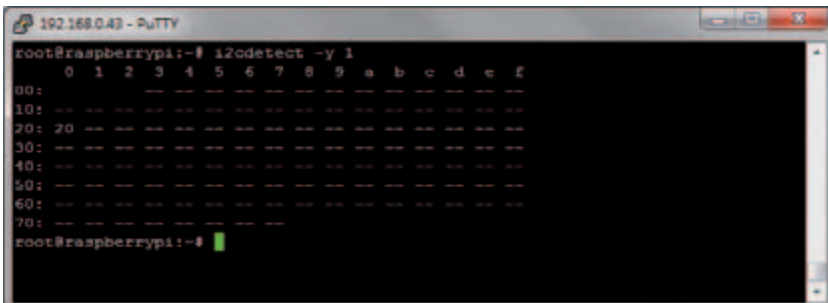


Fig. 12



conf

Nano es un editor de texto mínimo que funciona en ambiente terminal.

Eliminamos el módulo I²C de la *blacklist* eliminando la línea o, como hemos preferido nosotros, comentándola con un “#” (Fig. 4). Pulsamos Ctrl-x y después “y” a la solicitud de guardar el archivo, después de las modificaciones. Ejecutamos un *reboot* para hacer efectivas las modificaciones.

Ahora debemos hacer de manera que el módulo “liberado” sea cargado y llegue a ser 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 RaspberryPi se mantenga encendido. Al siguiente arranque, el módulo tendrá que ser cargado nuevamente por comando.

La segunda posibilidad nos permite cargar el módulo, en automático, al *boot* del sistema operativo, y hacerlo enseguida disponible a las aplicaciones, condición indispensable en un sistema *server unattended* (servidor desatendido).

La primera posibilidad requiere el uso del comando *modprobe*. Escribimos (Fig. 5):

modprobe i2c-dev

Podemos ver el éxito de la activación de los driver con el comando que muestra la lista de todos los módulos instalados (Fig. 6):

lsmod

Dado que en Linux todo (o casi) es un archivo si vamos en la carpeta */dev* vemos aparecer los archivos de conexion a los device *i2c-0* y *i2c-1* (Fig. 7).

El comando *modprobe* permite

cargar y descargar los módulos en tiempo de ejecución (*run time*) y mantiene sus efectos siempre que la RaspberryPi esté encendida. En caso de apagado, o aunque sea solo *reboot*, los módulos adjuntos deberán ser recargados manualmente.

Esta condición no es adecuada a funcionar con una aplicación independiente (*stand alone*), que tiene que funcionar en modo automático.

El comando *modprobe*, con la opción *remove*, puede ser utilizado para desactivar un módulo cargado previamente.

```
modprobe -r i2c-dev
```

Si se desea que el módulo *i2c-dev* sea cargado al encender la RaspberryPi, es necesario habilitar la carga permanente, que se realiza modificando el archivo de configuración */etc/modules*, que contiene la lista de los driver para cargar al momento del boot. En caso contrario debemos recordarnos de cargar el módulo a cada encendido con *modprobe*. Para modificar el archivo podemos usar el comando (Fig. 8):

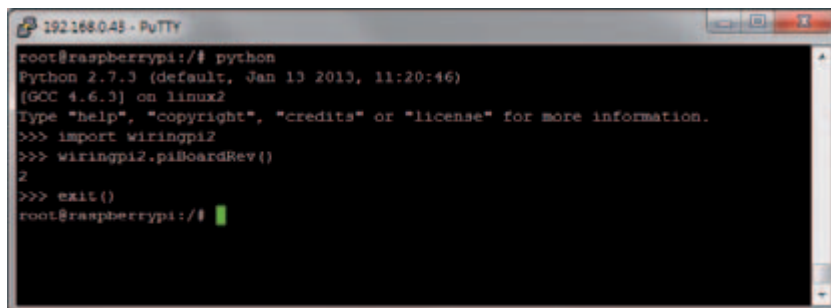
```
nano /etc/modules
```

y añadir una nueva línea, al archivo de configuración, que contenga el comando (Fig. 9):

```
i2c-dev
```

Pulsar CTRL-x y después "y" para guardar las modificaciones al archivo y salir.

Ahora instalamos el paquete *i2c-tools*, que nos proporciona una serie de funciones utilizables a línea de comando para verificar el funcionamiento del bus I²C, como siempre, no antes de haber actualizado la lista de paquetes de la distribución con el comando:



```
192.168.0.43 - PuTTY
root@raspberrypi:~# python
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import wiringpi2
>>> wiringpi2.piBoardRev()
2
>>> exit()
root@raspberrypi:~#
```

Fig. 13

```
apt-get update
```

Sigamos con el comando (Fig. 10):

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

Añadimos nuestro usuario *pi* al grupo I²C (Fig. 11), con:

```
adduser pi i2c
```

Para activar las nuevas configuraciones, realizamos el *reboot* de la Raspberry Pi con el comando:

```
reboot
```

Después que la RaspberryPi se ha reiniciado, os habéis conectado nuevamente con PuTTY; ejecutado, si es necesario, el comando *modprobe i2c-dev*, verificamos si el integrado es visible sobre el bus I²C con el comando:

```
i2cdetect -y 0 per RaspberryPi rev. 1
```

o con el comando:

```
i2cdetect -y 1 per RaspberryPi rev. 2
```

Deberéis obtener un resultado similar al que aparece en Fig. 12, donde la dirección 0x20 identifica el integrado MCP23017.

Ahora es tiempo de instalar la librería par la gestión del integrado MCP23017. Hemos optado por la librería *wiringpi2*, un excelente sistema de gestión del GPIO de Raspberry Pi escrita en C por Gordon "Drogon" Henderson y "adaptada" al uso con el lengua-

je Python por Phil "Gadgetoid" Howard. Esta librería, distribuida con licencia GNU LGPLv3, incluye diferentes módulos, para el uso de integrados específicos que utilizan los buses I²C y SPI: uno de estos es el MCP23017, utilizado en nuestro shield.

El enfoque adoptado en la librería para el uso con el MCP23017 es asignar una referencia numérica a cada pin de entrada/salida del integrado. La numeración parte de un número definido a placer, haciendo posible utilizar más de un integrado sobre una misma tarjeta, cada uno con los pines reconocibles por distinto intervalo de numeración. Como veremos dentro de poco, nosotros hemos asignado un intervalo de números a los pines del integrado que toman los valores a partir de 65, para el pin GPA0, al 80 para el pin GPB7.

Antes de describir cualquier ejemplo de uso de la librería, procedemos a instalarla. La librería *wiringpi2* está disponible como paquete (*package*) adjunto al lenguaje Python. Para gestionar los paquetes de Python necesitamos el gestor de paquetes Python "pip". Pip es un acrónimo recursivo que proviene de "Pip Installs Python". En orden, instalamos el paquete "pip" y después, la librería "wiringpi2". Actualizamos de nuevo la lista de paquetes de la distribución con el comando:

```
apt-get update
```

Instalamos las dependencias re-

Listado 1

```
#!/usr/bin/python

# rele_test.py

import wiringpi2 as wiringpi # importa librería wiringpi2
from time import sleep

pin_base = 65      # impone la numeración de los pines a partir del 65
i2c_addr = 0x20    # dirección obtenida con los pines A0, A1, A2 a GND

wiringpi.wiringPiSetup()          # inicialización librería wiringpi2
wiringpi.mcp23017Setup(pin_base,i2c_addr) # impone pin y dirección i2c
wiringpi.pinMode(65, 1)          # pone GPA0 como salida
wiringpi.digitalWrite(65, 0)     # pone GPA0 a 0 (0V, off)
wiringpi.pinMode(66, 1)          # pone GPA1 como salida
wiringpi.digitalWrite(66, 0)     # pone GPA1 a 0 (0V, off)
wiringpi.pinMode(67, 1)          # pone GPA2 como salida
wiringpi.digitalWrite(67, 0)     # pone GPA2 a 0 (0V, off)
wiringpi.pinMode(68, 1)          # pone GPA3 como salida
wiringpi.digitalWrite(68, 0)     # pone GPA3 a 0 (0V, off)
wiringpi.pinMode(69, 1)          # pone GPA4 como salida
wiringpi.digitalWrite(69, 0)     # pone GPA4 a 0 (0V, off)
wiringpi.pinMode(70, 1)          # pone GPA5 como salida
wiringpi.digitalWrite(70, 0)     # pone GPA5 a 0 (0V, off)
wiringpi.pinMode(71, 1)          # pone GPA6 como salida
wiringpi.digitalWrite(71, 0)     # pone GPA6 a 0 (0V, off)
wiringpi.pinMode(72, 1)          # pone GPA7 como salida
wiringpi.digitalWrite(72, 0)     # pone GPA7 a 0 (0V, off)

wiringpi.pinMode(73, 0)          # pone GPB0 como entrada
wiringpi.pullUpDnControl(73, 0)  # pull-up y pull-down internos desactivados
wiringpi.pinMode(74, 0)          # pone GPB1 como entrada
wiringpi.pullUpDnControl(74, 0)  # pull-up y pull-down internos desactivados
wiringpi.pinMode(75, 0)          # pone GPB2 como entrada
wiringpi.pullUpDnControl(75, 0)  # pull-up y pull-down internos desactivados
wiringpi.pinMode(76, 0)          # pone GPB3 como entrada
wiringpi.pullUpDnControl(76, 0)  # pull-up y pull-down internos desactivados
wiringpi.pinMode(77, 0)          # pone GPB4 como entrada
wiringpi.pullUpDnControl(77, 0)  # pull-up y pull-down internos desactivados
wiringpi.pinMode(78, 0)          # pone GPB5 como entrada
wiringpi.pullUpDnControl(78, 0)  # pull-up y pull-down internos desactivados
wiringpi.pinMode(79, 0)          # pone GPB6 como entrada
wiringpi.pullUpDnControl(79, 0)  # pull-up y pull-down internos desactivados
wiringpi.pinMode(80, 0)          # pone GPB7 como entrada
wiringpi.pullUpDnControl(80, 0)  # pull-up y pull-down internos desactivados

try:
    while True:
        if not wiringpi.digitalRead(73): # lógica invertida (pull-up externo)
            wiringpi.digitalWrite(65, 1) # impone GPA1 a 1 (3V3, on)
        else:
            wiringpi.digitalWrite(65, 0) # impone GPA1 a 0 (0V, off)

        if not wiringpi.digitalRead(74):
            wiringpi.digitalWrite(66, 1)
        else:
            wiringpi.digitalWrite(66, 0)

        if not wiringpi.digitalRead(75):
            wiringpi.digitalWrite(67, 1)
        else:
            wiringpi.digitalWrite(67, 0)

        if not wiringpi.digitalRead(76):
            wiringpi.digitalWrite(68, 1)
        else:
            wiringpi.digitalWrite(68, 0)

        if not wiringpi.digitalRead(77):
            wiringpi.digitalWrite(69, 1)
        else:
            wiringpi.digitalWrite(69, 0)

        if not wiringpi.digitalRead(78):
            wiringpi.digitalWrite(70, 1)
        else:
```

(Continúa)

queridas por el paquete pip, con:

```
apt-get install python-dev
```

y finalmente el paquete pip mismo:

```
apt-get install python-pip
```

Ahora, utilizando el gestor de paquetes pip podemos instalar la librería propiamente:

```
pip install wiringpi2
```

Terminado este proceso, podemos verificar si todo ha funcionado correctamente. Abramos el intérprete Python en línea de comandos, escribimos simplemente “python” en el prompt de la ventana del terminal, y verificamos la correcta instalación de la librería wiringpi2 tecleando las instrucciones (**Fig. 13**):

```
import wiringpi2
wiringpi2.piBoardRev()
```

La respuesta, que contiene el número de versión de la librería, nos confirma que todo funciona correctamente.

Ahora utilizaremos la librería para realizar algún programa de ejemplo. El primer programa, visible en el **Listado 1**, permite enganchar cada pin de entrada del banco GPB del integrado, al correspondiente pin de salida del banco GPA. Los comentarios no deberían dejar dudas sobre la lógica del programa, entonces describimos únicamente las instrucciones “try: - except:”, introducidas por primera vez en nuestros artículos. Estas instrucciones permiten “tender una trampa” los errores. Si incluimos un conjunto de instrucciones dentro de un grupo “try:” (prueba), en caso de error el programa, en vez de interrumpir la ejecución, intercepta el error y dispara las instrucciones contenidas en

Listado 1 – segue

```
wiringpi.digitalWrite(70, 0)

if not wiringpi.digitalRead(79):
    wiringpi.digitalWrite(71, 1)
else:
    wiringpi.digitalWrite(71, 0)

if not wiringpi.digitalRead(80):
    wiringpi.digitalWrite(72, 1)
else:
    wiringpi.digitalWrite(72, 0)

sleep(0.05)

except:
    wiringpi.digitalWrite(65, 0) # pone GPAL a 0 (0V, off)
    wiringpi.pinMode(65, 0)      # pone de nuevo GPAL como entrada
    wiringpi.digitalWrite(66, 0)
    wiringpi.pinMode(66, 0)
    wiringpi.digitalWrite(67, 0)
    wiringpi.pinMode(67, 0)
    wiringpi.digitalWrite(68, 0)
    wiringpi.pinMode(68, 0)
    wiringpi.digitalWrite(69, 0)
    wiringpi.pinMode(69, 0)
    wiringpi.digitalWrite(70, 0)
    wiringpi.pinMode(70, 0)
    wiringpi.digitalWrite(71, 0)
    wiringpi.pinMode(71, 0)
    wiringpi.digitalWrite(72, 0)
    wiringpi.pinMode(72, 0)
    print " "
    print "fine"
```

el interior del grupo “except:”. En este grupo, en nuestro caso, hemos insertado las instrucciones necesarias para reportar todos los pines del integrado configurados como entrada. Este mecanismo funciona también cuando se interrumpe el bucle principal del programa con CTRL-c.

Para probar el programa podemos conectar pulsadores entre la masa y cada pin de entrada del shield; como alternativa podemos conectar un terminal de un puente de cable al terminal de masa y con el otro terminal tocar los conectores de los terminales de entrada. Con el instrumento WINScp, copiamos el programa del **Listado 1** en una carpeta de la Raspberry Pi dentro de la carpeta */home*: por ejemplo, */home/Rele*.

Damos un nombre al programa: por ejemplo “rele-test.py”. Posicionémonos en la carpeta */home/Rele* y lanzamos el programa con el comando:

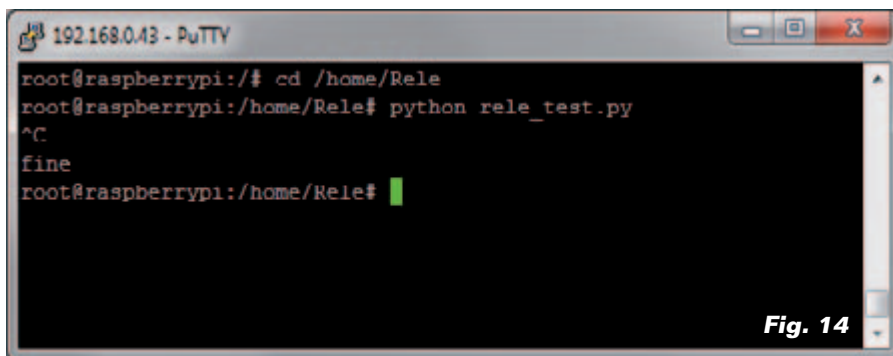
```
python rele-test.py
```

Ahora, llevando a masa (con los pulsadores o el jumper) las entradas del shield, veremos encenderse los LED y cerrarse los relé de las correspondientes salidas, por todo el tiempo durante el cual la entrada esté conectada a masa. Para terminar el programa, damos al comando CTRL-c (**Fig. 14**). Como hemos descrito en la parte introductoria del artículo, este programa, cuando funciona, mantiene ocupado constantemente el bus I²C, impidiendo a otros programas el acceso. Para superar tales problemas es necesario desacoplar el uso del bus respecto a los programas.

Un método es el escribir programas que vinculen el bus únicamente por el tiempo necesario para ejecutar las acciones requeridas, y después lo liberen para

dejar el campo libre a otros que lo requieran. Os ofrecemos un ejemplo de esta solución en el **Listado 2** y en el **Listado 3**. La solución propuesta está compuesta por un programa, para lanzar una sola vez al encender la Raspberry Pi, para inicializar el integrado MCP23017 y configurar los pines en el modo requerido por el shield. El segundo programa se lanza cada vez que se desea modificar el estado de un pin de salida. En nuestro ejemplo, lo lanzaremos en momentos temporales definidos utilizando el

planificador cron. En este modo podremos realizar una aplicación temporizada como un sistema de irrigación o un sistema de control luces día/noche. El programa del **Listado 2** se deriva de la parte de configuración inicial del programa mostrado en el **Listado 1**. También el programa presentado en el **Listado 3** parte de fragmentos de código del **Listado 1** añadiendo la posibilidad de recibir parámetros del exterior. En nuestro caso hemos previsto dos: el nombre del pin del cual queremos modificar el estado, en



```
192.168.0.43 - PuTTY
root@raspberrypi:/# cd /home/Rele
root@raspberrypi:/home/Rele# python rele_test.py
^C
fine
root@raspberrypi:/home/Rele#
```

Fig. 14

Listado 2

```
#!/usr/bin/python

# rele_set.py

import wiringpi2 as wiringpi # importa libreria wiringpi2

from time import sleep
import sys

pin_base = 65          # impone la numeración de los pines a partir del 65
i2c_addr = 0x20       # dirección obtenida con los pines A0, A1, A2 a GND

try:
    wiringpi.wiringPiSetup()                # inicializa libreria wiringpi2
    wiringpi.mcp23017Setup(pin_base,i2c_addr) # impone pin e indirizzo i2c
    wiringpi.pinMode(65, 1)                 # pone GPA0 como salida
    wiringpi.digitalWrite(65, 0)           # pone GPA0 a 0 (0V, off)
    wiringpi.pinMode(66, 1)                 # pone GPA1 como salida
    wiringpi.digitalWrite(66, 0)           # pone GPA1 a 0 (0V, off)
    wiringpi.pinMode(67, 1)                 # pone GPA2 como salida
    wiringpi.digitalWrite(67, 0)           # pone GPA2 a 0 (0V, off)
    wiringpi.pinMode(68, 1)                 # pone GPA3 como salida
    wiringpi.digitalWrite(68, 0)           # pone GPA3 a 0 (0V, off)
    wiringpi.pinMode(69, 1)                 # pone GPA4 como salida
    wiringpi.digitalWrite(69, 0)           # pone GPA4 a 0 (0V, off)
    wiringpi.pinMode(70, 1)                 # pone GPA5 como salida
    wiringpi.digitalWrite(70, 0)           # pone GPA5 a 0 (0V, off)
    wiringpi.pinMode(71, 1)                 # pone GPA6 como salida
    wiringpi.digitalWrite(71, 0)           # pone GPA6 a 0 (0V, off)
    wiringpi.pinMode(72, 1)                 # pone GPA7 como salida
    wiringpi.digitalWrite(72, 0)           # pone GPA7 a 0 (0V, off)

    wiringpi.pinMode(73, 0)                 # pone GPB0 como entrada
    wiringpi.pullUpDnControl(73, 0)         # pull-up y pull-down internos desactivados
    wiringpi.pinMode(74, 0)                 # pone GPB1 como entrada
    wiringpi.pullUpDnControl(74, 0)         # pull-up y pull-down internos desactivados
    wiringpi.pinMode(75, 0)                 # pone GPB2 como entrada
    wiringpi.pullUpDnControl(75, 0)         # pull-up y pull-down internos desactivados
    wiringpi.pinMode(76, 0)                 # pone GPB3 como entrada
    wiringpi.pullUpDnControl(76, 0)         # pull-up y pull-down internos desactivados
    wiringpi.pinMode(77, 0)                 # pone GPB4 como entrada
    wiringpi.pullUpDnControl(77, 0)         # pull-up y pull-down internos desactivados
    wiringpi.pinMode(78, 0)                 # pone GPB5 como entrada
    wiringpi.pullUpDnControl(78, 0)         # pull-up y pull-down internos desactivados
    wiringpi.pinMode(79, 0)                 # pone GPB6 como entrada
    wiringpi.pullUpDnControl(79, 0)         # pull-up y pull-down internos desactivados
    wiringpi.pinMode(80, 0)                 # pone GPB7 como entrada
    wiringpi.pullUpDnControl(80, 0)         # pull-up y pull-down internos desactivados

except Exception, e:
    print e
```

la forma Ax y el valor del estado que deseamos poner: "ON" u "OFF". El formato con el cual lanzaremos el comando será entonces:

```
python rele_pin_set.py A1 ON
```

Para leer los parámetros dentro del programa se utiliza el método "argv" del módulo "sys". En Python, el módulo "sys" hace disponible una serie de funciones y de variables que permiten interactuar con el ambiente *runtime* dentro del cual viene realizado el programa Python. En particu-

lar, "argv" contiene la lista de los argumentos pasados al programa en el momento del lanzamiento. Los parámetros están disponibles en forma de array donde el elemento argv[0] contiene el nombre del programa y los sucesivos

argv[i] contienen los argumentos, en el orden en el cual han sido escritos en línea de comando.

En nuestro caso tendremos:

```
argv[0] = "rele_pin_set"
```

```
argv[1] = "A1"
```

```
argv[2] = "ON"
```

Los parámetros adquiridos son utilizados para personalizar las instrucciones de configuración del pin. Dado que tenemos asignados al primer pin el valor 65, para obtener la correspondencia entre el parámetro pasado al programa, y el correspondiente número gestionado por la librería, utilizamos la instrucción:

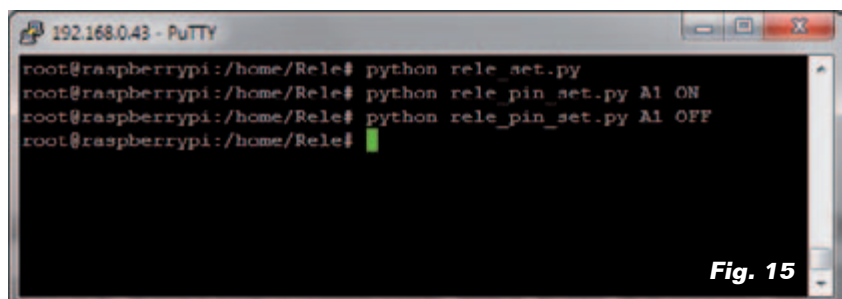
```
Pin_num = int(Pin[1:]) + 65
```

El objetivo de esta instrucción es eliminar del primer parámetro la "A" inicial, y sumar 65 al valor numérico resultante para obtener el valor correspondiente del pin reconocido de la librería. Lanzamos manualmente el primer programa para configurar inicialmente el integrado MCP23017; lanzamos después el segundo programa, variando los parámetros y verificamos los resultados que obtenemos (Fig. 15).

Pasamos ahora a configurar el planificador "cron".

El planificador "cron" es el servicio predefinido, en Linux, para la ejecución por tiempo de programas y comandos.

Para funcionar, cron se apoya en una tabla que contiene las "directivas" sobre programas para ejecutar y define en que



```
192.168.0.43 - PUTTY
root@raspberrypi:/home/Rele# python rele_set.py
root@raspberrypi:/home/Rele# python rele_pin_set.py A1 ON
root@raspberrypi:/home/Rele# python rele_pin_set.py A1 OFF
root@raspberrypi:/home/Rele#
```

Fig. 15

instantes lanzarlos. La tabla utilizada por cron se gestiona con la funcionalidad crontab

crontab <opción>

donde opción puede asumir los valores *-e*, *-l* y *-r*.

El comando *crontab -e* nos permite abrir el archivo de configuración predefinido de crontab, como se ve en **Fig. 16**.

Cada línea del archivo de configuración tiene la siguiente estructura:

```

* * * * * comando a ejecutar
-----
| | | | |
| | | | |--- día de la semana (0-6) (0=Domingo)
| | | +---- mes (1-12)
| | +----- día del mes (1-31)
| +----- hora (0-23)
+----- min (0-59)

```

Cada parámetro puede ser configurado como serie de valores (por ejemplo 0, 3, 7, 9) como un intervalo (por ejemplo 3-7) o una combinación de los dos (ejemplo: 0,3,4-6,9) o, aún, como frecuencia */5 (cada 5 minutos, horas, etc.).

En nuestro caso queremos que por cada minuto par sea ejecutado el programa *rele_pin_set.py*, con los parámetros impuestos de manera se cierre el relé conectado al pin GPBA1 y a cada minuto impar venga nuevamente ejecutado el mismo programa con los parámetros impuestos para abrir el relé. En la ventana que nos muestra el contenido del archivo, añadimos al final las líneas de dirección como en la **Fig. 16**:

```
0-58/2 * * * * python /home/Rele/rele_pin_set.py A1 ON > /dev/null
```

que significa hacer ejecutar el programa a cada minuto par de cada hora, día y mes, y:

```
1-59/2 * * * * python /home/Rele/rele_pin_set.py A1 OFF > /dev/null
```

Salimos del archivo con <CTRL>x,

Listado 3

```

#!/usr/bin/python

# rele_pin_set.py

import wiringpi2 as wiringpi
from time import sleep
import sys

Pin = sys.argv[1]
Stato = sys.argv[2]

pin_base = 65      # impone la numeración de los pines a partir del 65
i2c_addr = 0x20    # dirección obtenida con los pines A0, A1, A2 a GND

Pin_num = int(Pin[1:]) + 65

try:
    wiringpi.wiringPiSetup()          # inicializa librería wiringpi
    wiringpi.mcp23017Setup(pin_base,i2c_addr) # pone pin y dirección i2c
    if Stato == "ON":
        wiringpi.digitalWrite(Pin_num, 1) # pone el pin a 1 (3V3, on)
    else:
        wiringpi.digitalWrite(Pin_num, 0) # pone el pin a 0 (0V, off)
except Exception, e:
    print e

```

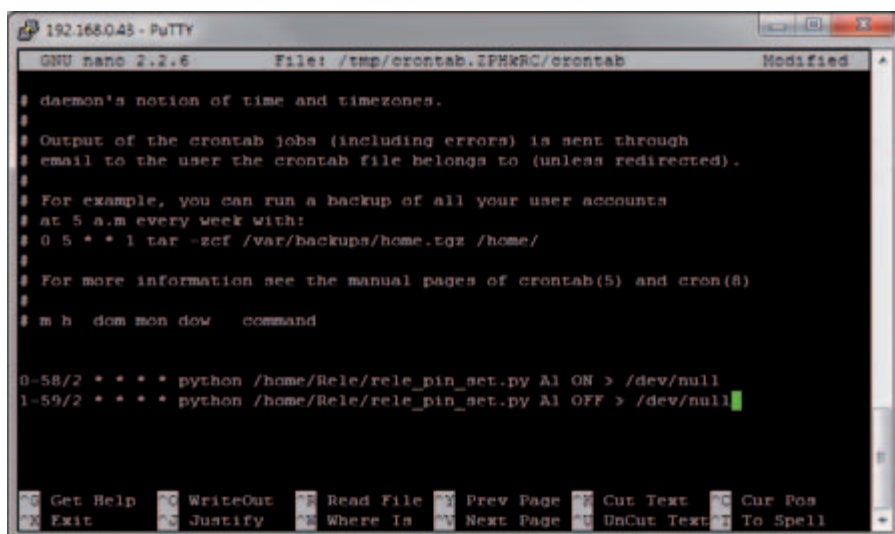
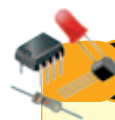


Fig. 16

respondemos "y" a la pregunta de guardado y confirmamos con el nombre del archivo con "Invío". Desde ahora, cada minuto se lanzará el programa que, alternativamente, activará y desactivará el relé. Para modificar los ajustes de cron, rellamamos la funcionalidad: *crontab -e* Para ver los ajustes utilizamos la opción: *crontab -l* Para interrumpir la ejecución automática escribimos: *crontab -r*

(183039)



el MATERIAL

Todos los componentes utilizados en este proyecto son de fácil disponibilidad y el master del PCB se puede descargar gratuitamente del sitio de la revista. El circuito está también disponible en kit (cod. FT1093K) al precio de 32,00 Euros, IVA incluido.

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