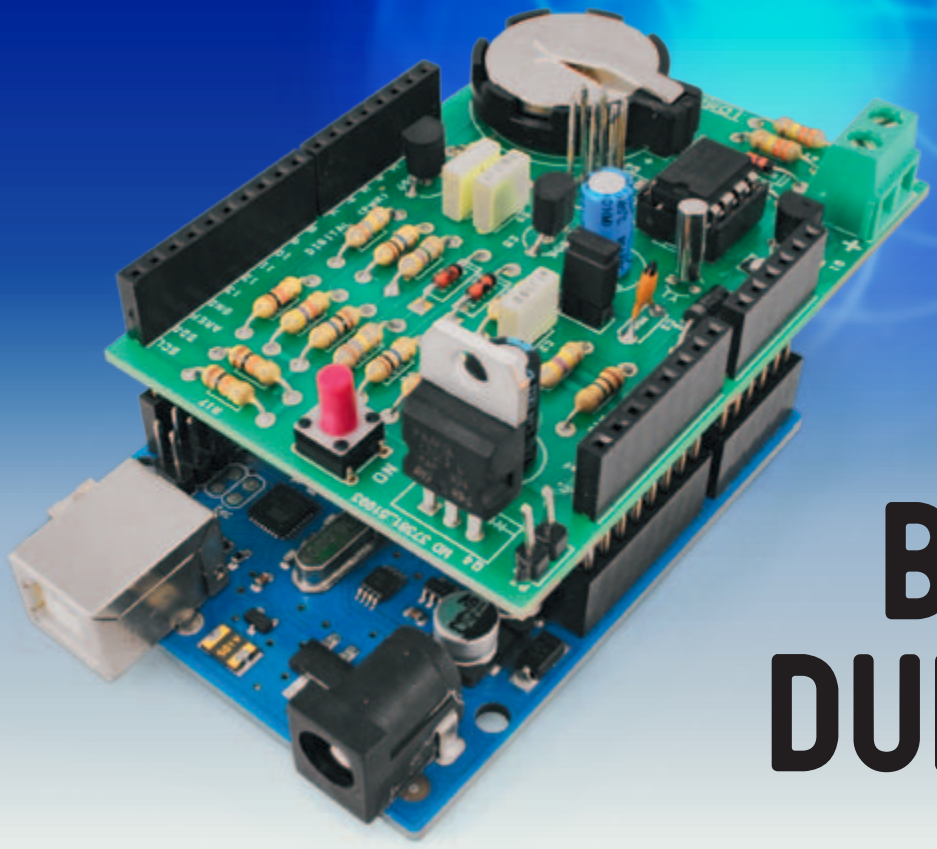


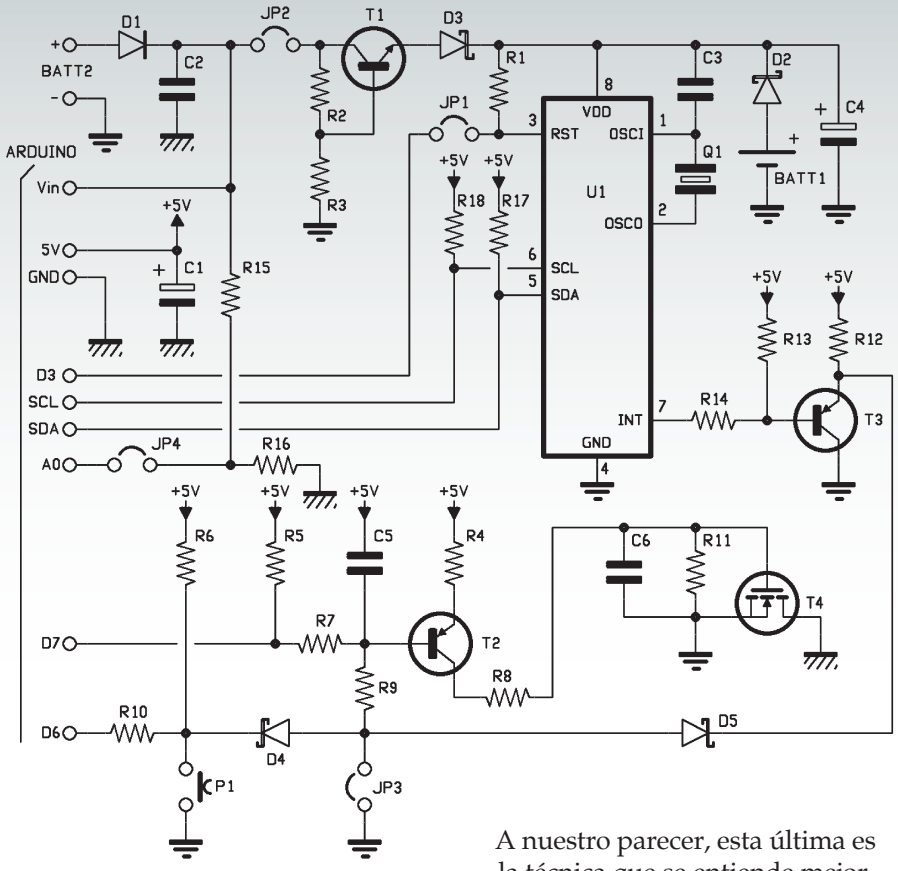
Optimizamos la autonomía de los sistemas Arduino alimentados por batería, gracias a un temporizador basado en RTC con función despertador programable.

ARDUINO BATERÍA DURADERA

Ing. TOMMASO y ALESSANDRO GIUSTO

Como alrededor de Arduino han nacido rápidamente tantas librerías y hardware, aquello que falta en el panorama de la tarjeta del equipo Arduino es una solución cualitativamente válida para optimizar el consumo eléctrico cuando el sistema se alimente por batería. Es cierto que existen técnicas software que permiten limitar la frecuencia del reloj principal de manera que disminuya la absorción de corriente pero desde nuestro punto de vista no son capaces de garantizar una duración aceptable de la batería. Hasta ahora la alimentación por batería para Arduino se ha pensado siempre para hacer frente a breves y temporales pérdidas de la red eléctrica, y nunca como fuente principal de energía; también porque la alimentación de los sistemas electrónicos por baterías es una aplicación bastante crítica y requiere de técnicas especiales combinadas de hardware/software para limitar el consumo de energía. En general en una tarjeta con microprocesador o microcontrolador, el componente que requiere más

[esquema **ELÉCTRICO**]



energía es seguramente la CPU; de esta consideración han nacido técnicas que aprovechan al máximo la capacidad de las modernas MCU para trabajar al 100 % solo por el tiempo necesario para procesar los datos de entrada/salida, y de ponerse (a través de instrucciones específicas) en estado de "sleep" o bajo consumo (las secciones no necesarias del micro, como los registros de E/S, los timer y los módulos PWM se apagan) por el tiempo restante. Cuando esto no basta, para optimizar el consumo y aumentar la autonomía dada por la batería, se puede recurrir a soluciones que apagan completamente el sistema y lo vuelven a encender periódicamente o cuando lo requiere un evento externo; en la práctica, se trata de utilizar interruptores temporizados o controlados por circuitos o eventos.

A nuestro parecer, esta última es la técnica que se entiende mejor para utilizarse en sistemas basados en Arduino y que ofrece las mejores prestaciones con respecto al tiempo de duración de la carga de la batería.

NUESTRA SOLUCIÓN

El shield presentado en estas páginas es algo de este estilo: se trata de un temporizador con Reloj de Tiempo Real (RTC o *Real Time Clock*) programable por Arduino capaz de, cuando este se apaga, despertarlo el día y hora prefijados. Específicamente, el shield está pensado para ser combinado con las tarjetas Arduino Uno. Está diseñado para recibir (a través de un terminal) la tensión de alimentación proveniente de una batería (son soportadas tensiones de alimentación comprendidas entre 5 y 12 voltios) y, aprovechando una particular funcionalidad del chip PCF8593T (un reloj/calendario), que permite encender el sistema a intervalos fijos seleccionables por software según las condiciones encontradas.

Como veremos luego con más detalle, el hardware del shield permite al software Arduino elegir cuando "ser despertado" y, una vez encendido, a través del estado de una salida digital, por cuanto tiempo permanecer encendido e incluso cuando apagarse.

Resumiendo, se puede decir que la técnica general de funcionamiento es la siguiente (**Fig. 1**):

- el sistema se apaga por un determinado intervalo de tiempo;
- en un cierto momento el reloj despierta a Arduino; este último decide permanecer despierto y pasa a analizar el propio estado interno, así como el de cualquier pin/sensores externos;
- terminado el análisis de las entradas y realizada la actuali-

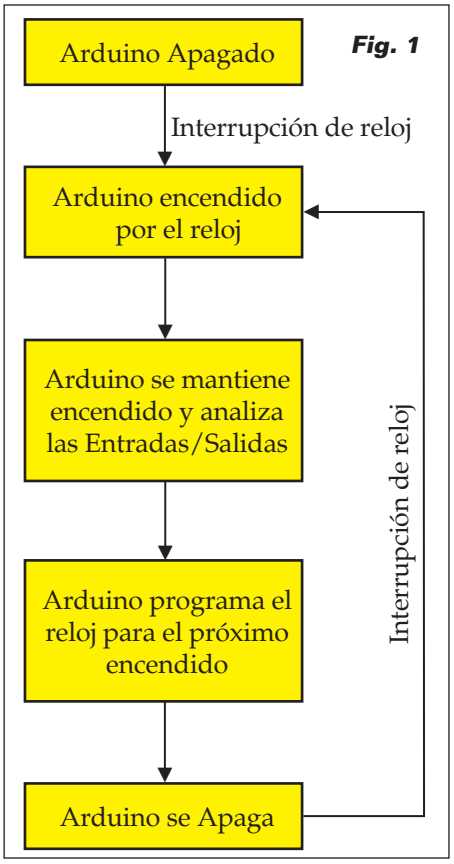


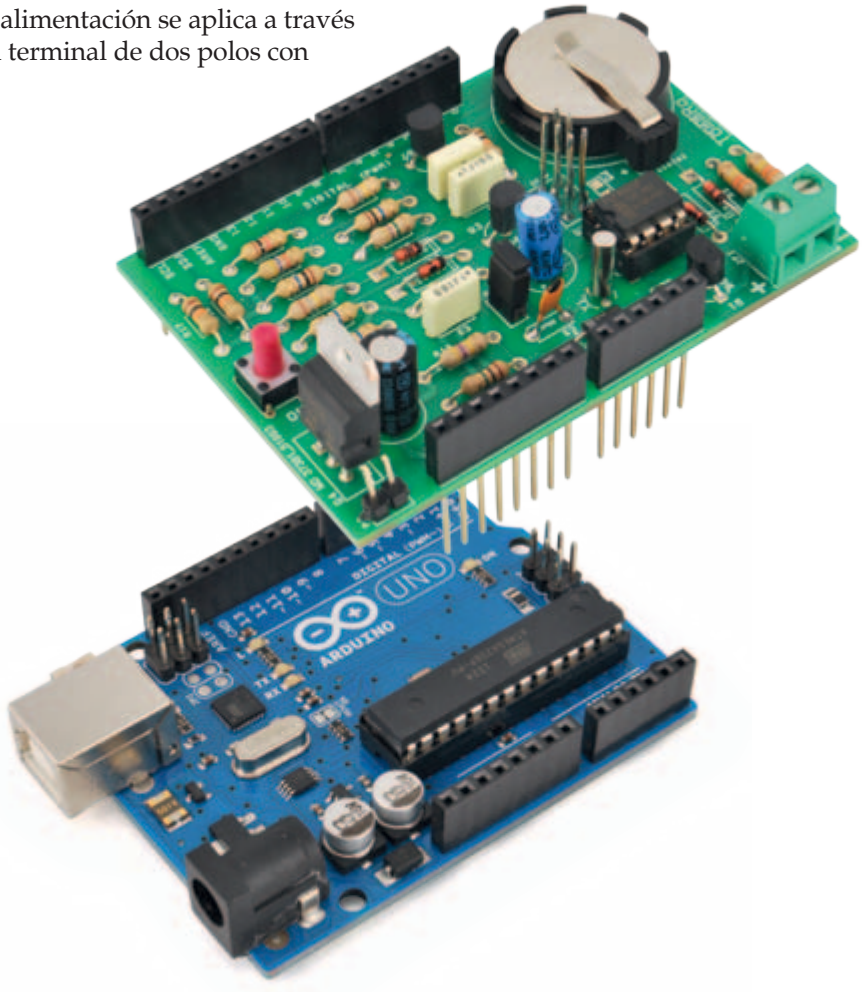
Fig. 1

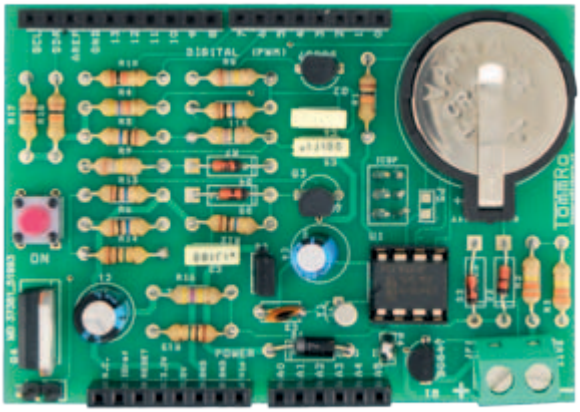
zación de las salidas, Arduino programa el reloj indicando el instante en cual tendrá que ser despertado; después decide apagarse;
- por todo el tiempo restante, el sistema queda apagado y todo el mecanismo se repite en el siguiente encendido.

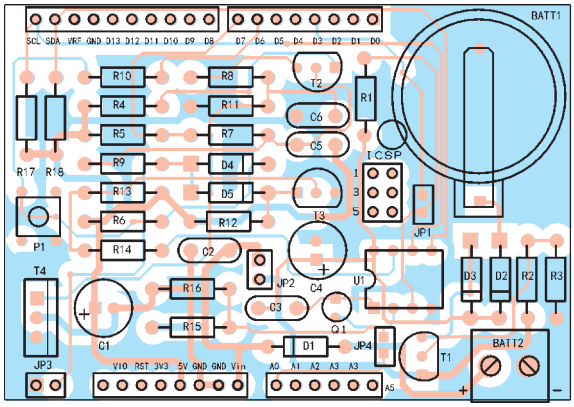
Es bastante intuitivo entender que si el software Arduino se realiza ad hoc, maximizando los tiempos de apagado, la autonomía de funcionamiento de un sistema Arduino alimentado por batería puede alargarse incluso algunos meses o años. Como con los acumuladores, hay que medirlo teniendo en cuenta el fenómeno de la autodescarga.
Claramente el sistema es más adecuado en condiciones en las cuales no es necesario un control en tiempo real, para el análisis de condiciones de peligro en el cual los tiempos deben ser mínimos o para sistemas que deben estar siempre online no parece muy adecuado; está sin embargo pensado para situaciones en las cuales se deben monitorizar medidas físicas que permanecen estables o varían lentamente en el tiempo (por ejemplo la temperatura de una habitación o el ambiente, los parámetros meteorológicos, la luminosidad ambiental, etc…).
Dado que el shield contiene en su interior el chip PCF8593T, puede utilizarse también como shield RTC para proporcionar a Arduino las funcionalidades típicas de un reloj y calendario.
El shield utiliza dos pines digitales (D6 y D7) para la gestión del sistema de encendido; el puerto I²C se utiliza para la comunicación/programación del componente PCF8593T. Hemos previsto la posibilidad (a través de un puente) de usar el pin analógico A0 para leer la tensión de alimentación proveniente de la batería, de tal manera que conozcamos el nivel de carga y eventualmente enviar informes del caso o apagar definitivamente el sistema o partes del mismo.
En el shield, además, está presente un pulsador para encender Arduino de manera forzada (manualmente, mientras se encuentra en un intervalo de apagado); claramente las condiciones serán previstas en fase de escritura del firmware. Tenemos también un jumper que si lo insertamos, permitirá mantener Arduino siempre encendido; eso puede ser útil en algunos casos particulares.

ESQUEMA ELÉCTRICO

Empezamos a analizar el esquema eléctrico del shield. La tensión de alimentación se aplica a través del terminal de dos polos con las siglas BATT2; el diodo D1 funciona como protección de la inversión de la polaridad de la alimentación.
El corazón del circuito es el integrado U1, un PCF8593T producido por NXP; se trata de un Real Time Clock programable, que para funcionar utiliza un cuarzo de 32.768 Hz (Q1), gracias al cual se marca el reloj del oscilador interno. La frecuencia de oscilación se establece con la ayuda del condensador C3, mientras C4 ayuda a nivelar la tensión de alimentación. El reloj/calendario puede alimentarse tanto por la batería principal del sistema, como por la pila de botón CR2032 presente en el shield; en el primer caso debe







Lista de materiales

R1: 10 kohm
R2: 330 kohm
R3: 330 kohm
R4: 4,7 kohm
R5: 10 Mohm
R6: 10 Mohm
R7: 4,7 Mohm
R8, R9: 180 kohm
R10: 1 kohm
R11: 10 Mohm
R12: 10 Mohm
R13: 10 Mohm
R14: 1 Mohm
R15: 1 Mohm 1%
R16: 470 kohm 1 %
R17, R18: 10 kohm
C1: 220 µF 16 VL electrolítico
C2, C5: 100 nF 100 VL poliéster
C3: 22 pF cerámico
C4: 100 µF 16 VL electrolítico
C6: 10 nF 63 VL poliéster
U1: PCF8593P
T1: BC547
T2: BC557
T3: BC557
T4: P36NF06
D1: 1N4007
D2: SD103
D3: SD103
D4: SD103
D5: SD103
P1: Microswitch
Q1: Cuarzo 32.768 kHz

Varios:
- Terminal 2 polos
- Tira de 2 pines macho (2 pz.)
- Jumper (2 pz.)
- Zócalo 2x4, 300 mils.
- Porta-batería para CR2032
- Tira de pines macho/hembra 2x3
- Tira de pines macho/hembra 6
- Tira de pines macho/hembra 8 (2 pz.)
- Tira de pines macho/hembra 10
- Batería CR2032
- Circuito impreso

cerrarse el jumper JP2 y la tensión llega desde la etapa basada en T1, que es un regulador lineal serie que asegura tensión constante independientemente (dentro de ciertos límites) del valor de la tensión de alimentación proporcionada en el terminal BATT2.
En el segundo, se abre el JP2 y cierra el JP4.
Si la pila de botón está insertada también cuando el circuito es alimentado por un acumulador externo, es útil para mantener alimentado el integrado RTC (U1) durante el cambio de la batería principal, de manera que no se pierda la configuración actual del reloj/calendario.
Los diodos D2 y D3 sirven para que una u otra fuente de alimentación hagan funcionar el Real Time Clock sin que una interfiera con la otra; esto es importante porque la tensión aplicada en BATT2 es por lo general superior a la de la pila de botón (3 V) y en ausencia de los diodos terminaría con la sobrecarga de la pila misma. Hay que señalar que ambos diodos son Schottky y esto no es una casualidad: los hemos elegido debido a su mínima caída de tensión (poco más de 0,2 voltios) de tal manera que se mantenga correctamente en funcionamiento U1 también con la pila de 3 voltios.
El PCF8593T dispone de un pin de salida INT, que es la verdadera peculiaridad, dado que, respecto a integrados como el más conocido DS1307 de Dallas, el componente de NXP dispone de una funcionalidad de despertador programable: a la hora y fecha indicada, el chip pone a nivel lógico alto la línea (pin 7) y con él es posible activar otras funciones. En nuestro caso, el pin sirve para alimentar Arduino, para encenderlo. Una vez alimentado, Arduino controla la propia línea

D7 de tal manera que hace entrar en estado ON el MOSFET T4 del shield, el cual conecta la masa de la batería con la masa de Arduino, manteniendo la tarjeta encendida. Cuando Arduino tiene que apagarse, deja ir D7 a nivel alto (T2 va en cortocircuito y deja bloqueado el MOSFET, el cual desconecta la masa de Arduino del negativo de la batería, abriendo el circuito de alimentación principal. Si Arduino tiene que permanecer encendido para continuar una tarea, mantiene la línea D7 a nivel bajo y entonces fuerza la saturación de T2, cuyo colector alimenta la puerta del MOSFET.
En nuestro esquema, la resistencia R5 sirve de pull-up cuando Arduino está apagado, de lo contrario la línea D7 estaría flotante (no garantizaría un estado lógico estable); si Arduino se apaga, R5 mantiene la base de T2 a nivel alto y la puerta de T4 a cero lógico.
El pulsador P1 permite encender de manera forzada Arduino: pulsándolo, se cierra a masa el nodo R6, R10, D4 y la base de T2 se pone a nivel bajo, lo que fuerza el PNP en saturación y el MOSFET en conducción; cuando recibe tensión, Arduino lleva a cero lógico D7. El mismo discurso vale para el jumper JP3: si se cierra, pone a masa el nodo D4, D5, R9, haciendo conducir T2 y T4.
Los diodos (son también Schottky, con el fin de minimizar la caída de tensión sobre ellos) D4 y D5 forman una puerta lógica AND que permite llevar a nivel bajo a la base de T2 tanto al pulsador P1, como al transistor T3 (controlado por la salida INT del PCF8593T) sin que los dos circuitos influyan en los demás. D4 sirve también para aislar la línea D6 de Arduino cuando el transistor T3 pasa a saturación.
Concluimos la descripción del shield analizando las líneas de los pines de conexión de Arduino, aún no descritas: se utilizan los contactos del puerto I²C (SDA e SCL, dotados de resistencias pull-up, que son respectivamente R17 e R18) que serán utilizadas para la comunicación con el PCF8593T; el contacto D6 sirve para decir a Arduino quién ha pedido el encendido, mientras la A0 se usa, si el jumper JP4 se cierra, para permitir a Arduino tomar el control de la tensión de la batería (el divisor R15/R16 baja la tensión de alimentación de manera que no supere la máxima permitida por las líneas analógicas).

LIBRERÍA ARDUINO BATTERY SHIELD

Como hemos visto en el artículo, para el correcto funcionamiento del sistema es necesario que todo el software sea escrito teniendo en cuenta cómo se realiza el mecanismo de encendido y apagado de la electrónica.
Para facilitar la integración por parte de los usuarios finales, hemos realizado una librería software que implementa las funcionalidades básicas. Se definen tanto las funciones de "bajo nivel" (entre ellas la inicialización del sistema, encendido, auto alimentación y apagado de Arduino, lectura del valor analógico de la tensión de alimentación) como aquellas de "nivel medio", entre las cuales está el arranque del chip PCF8593T y las relativas a la programación de los instantes de encendido. Se definen también las funciones de "alto nivel" típicas de un componente reloj/calendario, es decir, programación y lectura de fecha y hora, verificación del nuevo año (incremento del año), cambio

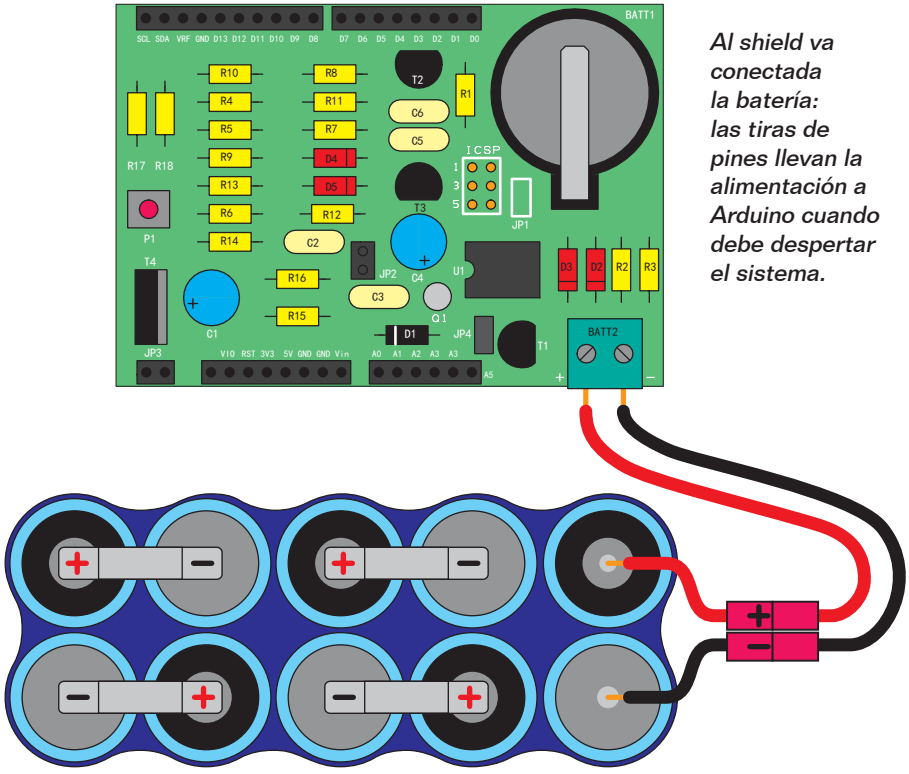
Funciones de la **librería**

Para la gestión de las funcionalidades del Battery shield es necesario una librería que emplea las funciones aquí descritas.

- **void begin()**: inicializa el hardware y software Arduino necesarios para el shield.
- **void powerON()**: configura el hardware Arduino para que se alimentado.
- **void turnOFF()**: configura el hardware Arduino para el apagado.
- **bool onFromButton()**: verifica si Arduino ha sido encendido a causa de presión del pulsador de encendido.
- **float getBatteryVoltage()**: lee el valor de la tensión de la batería de alimentación.
- **void startSecAlarmPCF8593, void startMinAlarmPCF8593, void startHourAlarmPCF8593**: seleccionamos el instante del próximo evento de encendido (es posible especificar por cuantos segundos, minutos o incluso horas Arduino debe permanecer apagado antes de volver a despertarse); como entrada se aceptan valores comprendidos entre 1 y 99 extremos incluidos.
- **void writeClockANDDataPCF8593()**: programa la fecha y hora actual (pasados como parámetros horas, minutos, segundos, día, mes y año) del RTC.
- **void readClockANDDataPCF8593()**: lee la fecha y hora actual indicada por el RTC.
- **bool checkProgrammedPCF8593()**: testea si el reloj/calendario de RTC está encendido y programado.
- **bool checkHourLegPCF8593():** verifica si por un determinado día debe ser activa la hora solar o legal.
- **void checkNewYear()**: testea si el calendario ha pasado el fin de año y actualiza el número de año.
- **byte getDayOfWeek()**: determina el día de la semana de una fecha.
- **bool YearBisestile()**: determina si un determinado año es bisiesto.

automático desde la hora legal a solar y viceversa, etc.
Como veremos más adelante, analizando el ejemplo de programación, utilizando estas funciones no debéis preocuparos de la gestión directa del hardware, ya que podéis contar con una interfaz software que se encarga y enmascara los detalles relativos.
La librería define un objeto "Battery" que tiene los siguientes métodos públicos (es decir, utilizables en el programa Arduino).

- **void begin()**: inicializa los pin D6 y D7 como entrada y salida digitales respectivamente y además enciende el reloj si este es detectado como parado.
- **void powerON()**:pin D7 a nivel bajo de manera que Arduino pueda permanecer encendido.
- **void turnOFF()**: al contrario de **powerON()**,D6 a nivel alto de manera que Arduino pueda apagarse.
- **bool onFromButton()**: en el análisis del esquema eléctrico hemos encontrado el pulsador P1, que sirve para encender Arduino pasando la programación del reloj; para distinguir si el encendido ha sido efectuado por el PCF8593T o por la presión del pulsador se utiliza el pin digital D6 de Arduino. La función **onFromButton()** testea el estado de D6 y por consiguiente identifica si el encendido deriva del pulsador (resultado de la función true) o por el reloj (resultado false).
- **float getBatteryVoltage()**: como hemos ya hemos, es posible utilizar el pin analógico A0 de Arduino para controlar la tensión de alimentación de la batería. Esta función se puede utilizar para leer tal valor; la función ejecuta ya todas las conversiones y tiene en cuenta las posibles caídas de tensión debidas a distintos componentes (en particular al diodo D15) y retorna en formato float (número en coma flotante) la tensión de la batería.
- **void startSecAlarmPCF8593**, **void startMinAlarmPCF8593**, **void startHourAlarmPCF8593**: con estas funciones se define el instante del próximo evento de encendido. En particular, se puede especificar (tomando como referencia el instante de ejecución de las funciones) con respecto a segundos, minutos u horas, cuánto debe permanecer apagado Arduino antes de ser despertado por el reloj. Las funciones aceptan como entrada un parámetro de un byte cuyos valores válidos son solo aquellos comprendidos entre 1 y 99, ambos incluidos.
- **void writeClockANDDataPCF8593()**: como el integrado PCF8593T, además de ser utilizado para determinar los instantes de encendido, puede ser utilizado genéricamente como RTC y reloj/calendario, esta función permite programar la fecha y hora actual (pasados como parámetros horas, minutos, segundos, día, mes y año). La función determina y programa en automático el día de la semana y si está activa la hora solar o la legal.
- **void readClockANDDataPCF8593()**: al contrario de la anterior función, permite leer la fecha y hora actual indicada por el chip PCF8593T. Como calendario retorna el día, mes y año, más el día de la semana; como reloj retorna la hora, los minutos, los segundos y las centésimas de segundo.
- **bool checkProgrammedPCF8593()**: permite testear si el reloj/calendario del chip PCF8593T han sido encendidos y programados. En caso afirmativo retorna true; en



Al shield va conectada la batería: las tiras de pines llevan la alimentación a Arduino cuando debe despertar el sistema.

caso negativo retorna false (en este caso encenderá llamando a la oportuna función).

- **bool checkHourLegPCF8593()**: permite verificar si para un determinado día (identificado por día de la semana, día y mes) debe estar activa la hora solar o legal (útil para ejecutar en automático el cambio de hora).
- **void checkNewYear()**: testea si el calendario ha pasado del final del año y en caso afirmativo, actualiza (incrementa de 1) el número del año.
- **byte getDayOfWeek()**: determina el día de la semana de una fecha (identificado por día, mes y año).
- **bool YearBisestile()**: determina si un determinado año es bisiesto.

PRIMER EJEMPLO FIRMWARE

Para entender el funcionamiento y la lógica de programación a utilizar con el battery shield, veamos dos ejemplos de código Arduino, el primero de los cuales está escrito en el **Listado 1**. Este, simplemente programa el battery shield para que Arduino se mantenga casi siempre apagado, encendiéndose cada 10 segundos y simula la ejecución de un programa (a través delay), después se apaga de nuevo en espera del próximo ciclo de ejecución.
Después de incluir las librerías utilizadas (en particular la "Battery.h" especifica del shield) se define el pin digital de gestión del LED presente en la tarjeta Arduino y además se define un objeto Battery que se utilizará para la gestión del mismo shield.
En la función de setup de Arduino (que, recordamos, es la primera que se ejecuta al encender) se programa el pin del LED y este se enciende; a continuación se arranca la librería Wire y la Serial (esta última utilizada para la depuración y seguir mejor el flujo del software).
Después se inicializa el objeto "batteryShield" a través de la función **batteryShield.begin()**; esta última ha sido escrita de manera que inicialice todos los recursos (tanto hardware como software) necesarios para gestionar el shield (de manera que "esconda" al usuario todos los detalles implementados de bajo nivel).
La próxima instrucción es la "batteryShield.powerON()" que configura el shield (en particular

Listado 1

```
//*****************************************************************
//* Inclusión librerías   *
//*****************************************************************
#include <Battery.h>
#include <Wire.h>
#include <Serial.h>

//*****************************************************************
//* Definición pin I/O Arduino     *
//*****************************************************************/
// Led tarjeta Arduino
const int pinBoardLed = 13;

//*****************************************************************
//* Variables programa    *
//*****************************************************************/
// Objeto battery shield
Battery batteryShield;

//*****************************************************************
//* Código programa       *
//*****************************************************************/

// Inicialización tarjeta
void setup() {
  // Inicializo I/O
  pinMode(pinBoardLed, OUTPUT);
  // Enciendo led Arduino
  digitalWrite(pinBoardLed, HIGH);

  // Arranque librería Wire
  Wire.begin();
  // Arranque puerta serie
  Serial.begin(115200);

  // Inicializo battery shield
  batteryShield.begin();
  // Autoalimentación Arduino
  batteryShield.powerON();

  // Si hay encendido de pulsador
  if (batteryShield.onFromButton() == true)
    Serial.println("Acceso da pulsante");
  // Si hay encendido no de pulsador
  else
    Serial.println("Encendido de reloj");

  // Observo/señalo tensión batería
  Serial.println(batteryShield.getBatteryVoltage());
}    // Cerrado función setup

// Programa Principal
void loop() {
  // Attesa (tiempo tarjeta encendida)
  delay(2500);

  // Programo reloja para próximo encendido (en 10 segundos)
  batteryShield.startSecAlarmPCF8593(10);
  // Apago led Arduino
  digitalWrite(pinBoardLed, LOW);
  // Apago Arduino
  batteryShield.turnOFF();
}    // Cierre función loop
```

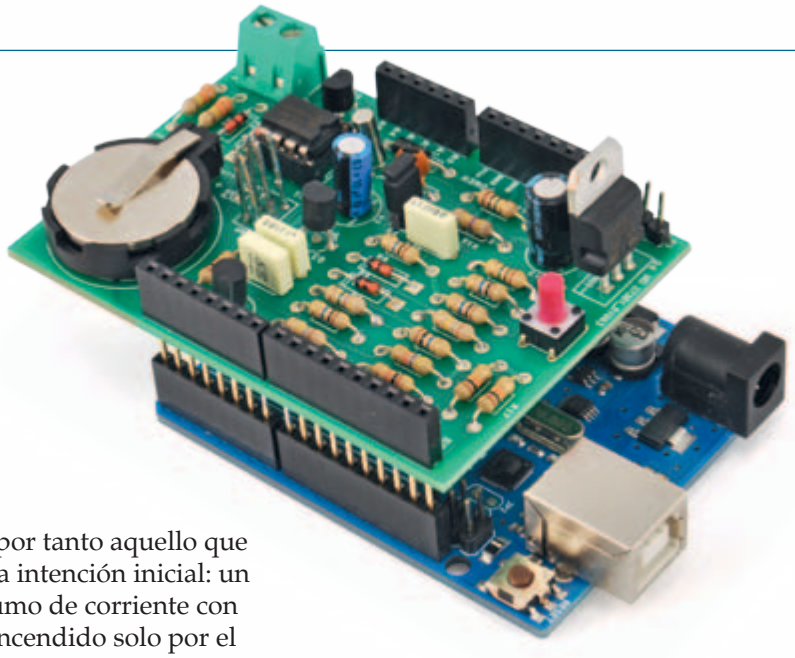
Listado 2

```
//*************************************************************************
//* Inclusión librerías   *
//*************************************************************************/
#include <EEPROM.h>
#include <Battery.h>
#include <Wire.h>
#include <Serial.h>

//*************************************************************************
//* Definición pin I/O Arduino     *
//*************************************************************************/
// Led tarjeta Arduino
const int pinBoardLed = 13;

//*************************************************************************
//* Variables programa    *
//*************************************************************************/
// Objeto battery shield
Battery batteryShield;

//*************************************************************************
//* Código programa       *
//*************************************************************************/
// Inicialización Tarjeta
void setup() {
  // Inicializo I/O
  pinMode(pinBoardLed, OUTPUT);
  // Enciendo led Arduino
  digitalWrite(pinBoardLed, HIGH);

  // Arranque librería Wire
  Wire.begin();
  // Arranque puerta serie
  Serial.begin(115200);

  // Inicializo battery shield
  batteryShield.begin();
  // Autoalimentación Arduino
  batteryShield.powerON();
}    // Cierre función setup

// Programa Principal
void loop() {
  int contatore;

  // Leo contador de EEPROM
    contatore  =  (((int)  (EEPROM.read(0x0000))  <<  8)  +  (int)  (EEPROM.
read(0x0001)));
  // Incremento contador
  contatore++;
  // Envío contador
  Serial.println(contatore);
  // Memorizo contador
  EEPROM.write(0x0000, (byte) (contatore >> 8));
  EEPROM.write(0x0001, (byte) (contatore & 0x00FF));

  // Programo reloj para próximo encendido (en 30 segundos)
  batteryShield.startSecAlarmPCF8593(30);
  // Apago led Arduino
  digitalWrite(pinBoardLed, LOW);
  // Apago Arduino
  batteryShield.turnOFF();
}    // Cierre función loop
```

el pin digital D7) de manera que Arduino es alimentado constantemente (en la práctica es una especie de autoalimentación en la cual Arduino decide si permanecer encendido o no). Llamando "batteryShield.onFromButton()" se identifica la causa del encendido (pulsador de ON o encendido programado por reloj) y envía un mensaje de debug por el puerto serie. Después se lee y envía por el puerto serie la tensión de alimentación de la batería; el valor visualizado es ya el real de la batería, teniendo en cuenta todas las caídas de tensión debidas a diodos y varios divisores resistivos. A este punto, el software Arduino está en ejecución y la tarjeta, en estas condiciones, se mantiene siempre alimentada (la batería lo permite). Terminada la función "setup()" se ejecuta el bucle "loop()"; la primera instrucción es un delay (de alrededor 2,5 segundos) que simula el funcionamiento de la rutina main de un programa genérico (lectura de entradas, ajuste de salidas, etc.). A continuación se ha programado que el próximo instante de encendido (instrucción "batteryShield.startSecAlarmPCF8593(10)") será en 10 segundos, entonces se apaga el LED de Arduino y después se quita la alimentación de la tarjeta en su conjunto (instrucción "batteryShield.turnOFF()"). Esta última instrucción es bloqueadora, es decir, además de programar el pin digital D7, entra en un ciclo infinito del cual no saldrá más (hasta que físicamente no se quite la alimentación, cuando la tarjeta se apaga). Transcurrido el tiempo programado de 10 segundos, el PCF8593T vuelve a encender la tarjeta y todo el mecanismo se repite (el software reinicia desde el setup como si fuese el primer encendido).

SEGUNDO EJEMPLO FIRMWARE

Veamos ahora un segundo ejemplo de código Arduino (**Listado 2**) que utiliza las funcionalidades del shield para obtener un encendido cada 30 segundos y memoriza (en EEPROM) un contador (a 16 bit) de encendido. También en este caso se incluyen librerías (en particular la "Battery.h" específica del shield); se definen el pin digital correspondiente al LED de Arduino y después un objeto Battery, que se utilizará para acceder al shield.



En la función de setup de Arduino se programa y se enciende el pin del LED; después se arranca la librería Wire y la Serial (esta última utilizada como debug para aclarar mejor el flujo del software). A continuación se inicializa el objeto "batteryShield" (función batteryShield.begin()) de manera que se inicializan todos los recursos (tanto hardware como software) necesarios para la gestión del shield. La próxima instrucción es la "batteryShield.powerON()" que indica al shield de mantener Arduino constantemente alimentado.
En este punto el software Arduino está en ejecución y la tarjeta, en estas condiciones, se mantiene alimentadade forma continua. Terminada la función "setup()" se ejecuta la "loop()"; las primeras instrucciones leen los primeros dos byte de la EEPROM y los utilizan para "construir" el contador de 16 bit de los encendidos; tal contador se va incrementando y actualizando el valor en la EEPROM (el mismo contador es enviado sobre el puerto serie para la depuración). A continuación se programa que el próximo instante de encendido (instrucción "batteryShield.startSecAlarmPCF8593(30)") será en 30 segundos, se apaga el led de Arduino y después se quita la alimentación de la tarjeta en su conjunto (instrucción "batteryShield.turnOFF()"). Pasado el tiempo programado de 30 segundos, el PCF8593T vuelve a encender la tarjeta y todo el mecanismo se repite (el software vuelve a empezar con el setup como si fuese su primer encendido). El tiempo de ejecución de este ejemplo es seguramente más rápido que el anterior; la tarjeta quedará encendida pocos instantes mientras la mayor parte del tiempo estará apagada. Hemos realizado por tanto aquello que era nuestra intención inicial: un bajo consumo de corriente con Arduino encendido solo por el tiempo necesario.

REALIZACIÓN PRÁCTICA

El hardware del sistema presentado está constituido por una tarjeta Arduino (suponemos la Uno Rev. 3) y por un Battery shield; este último suministrado como kit de montaje que puedes encontrar en nuestra web (*www.nuevaelectronica.com*) y todos sus componentes son de montaje convencional, lo que permite la realización esté al alcance de todos.
Adquirido lo necesario, se puede empezar por los componentes de perfil más bajo (resistencias, diodos, condensadores cerámicos y electrolíticos, puentes, etc..) y después pasar al soporte de la pila, los terminales de 2 polos. Proceder entonces con el integrado PCF8593P (primero soldar el zócalo, después insertar el integrado prestando atención a la posición de la muesca que indica el pin 1) y finalmente las tiras macho/hembra para la inserción del shield en Arduino. Completado el montaje, colocar la pila de botón en su lugar.

CONCLUSIONES

El shield presentado permite realizar sistemas Arduino alimentados por batería que, si se programan correctamente, pueden funcionar sin sustituir la batería durante bastantes años.
El shield ha sido proyectado para Arduino Uno Rev. 3, el nivel de iniciación de Arduino, pero se puede adaptar a las otras placas Arduino. Los sketch de ejemplo presentados en estas páginas tienen el objetivo de explicar cómo utilizar las librerías desarrolladas; no representan aplicaciones reales pero pueden ser utilizados como punto de partida para vuestros proyectos.

(186075)

el MATERIAL

Todos los componentes utilizados para realizar el shield descrito en este artículo son fáciles de encontrar en el mercado. En la web de Nueva Electrónica se puede adquirir el kit de este proyecto (cod. BATTERYSHIELD, 19,50 Euros) y también la tarjeta Arduino Uno Rev.3 (24,50 Euros). El sketch para realizar el proyecto también se puede descargar gratuitamente de la misma web.

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