IMPLEMENTACIÓN DE UN SISTEMA DE MEDICIÓN Y REGISTRO DE DATOS CON RELOJ DE TIEMPO REAL Y ALMACENAMIENTO EN TARJETA MICROSD

IMPLEMENTATION OF A DATALOGGER SYSTEM WITH REAL-TIME CLOCK AND STORAGE IN A MICROSD CARD

Miguel Ángel Bañuelos Saucedo
Universidad Nacional Autónoma de México
miguel.banuelos@icat.unam.mx

Resumen

Los sistemas de medición y registro de datos (*dataloggers*) de tipo comercial pueden ser relativamente costosos. En el presente trabajo se muestra la implementación de un sistema de registro de datos, utilizando una plataforma de hardware libre (tarjeta Arduino); lo cual lo convierte en un dispositivo de bajo costo, configurable a las necesidades específicas del usuario, y una alternativa al uso de equipos comerciales. El sistema utiliza un módulo de expansión para el manejo de una tarjeta de memoria microSD, donde se almacenan los datos en formato compatible con hojas de cálculo. También cuenta con un módulo reloj de tiempo real para almacenar cada dato con su marca de tiempo. Se muestra el desempeño del sistema utilizando diferentes fuentes de alimentación: cargador de celular, batería de respaldo de un teléfono celular, fuente de laboratorio y un arreglo de pilas recargables de NiMH (Níquel, Metal, Hidruro). Finalmente, se presentan algunas opciones para reducir el consumo de energía que permiten que el sistema funcione durante 12 días con cuatro pilas AA de NiMH.

Palabras Claves: Arduino*, Datalogger*, registro de datos, reloj de tiempo real, tarjeta microSD.

Abstract

Commercial data acquisition and recording systems (dataloggers) can be relatively expensive. In this work, the development of a measuring and data recording system based on open hardware (Arduino board) is presented. The

resulting device is a low cost and general-purpose alternative to the use of commercial equipment. The system uses an expansion card for handling a microSD memory card, where data is recorded in a datasheet compatible format. It also has a real-time clock module which allows to register each data sample with its time-stamp. The performance of the system using different power sources such as a cellular phone charger, a cellular phone back-up battery, a laboratory power supply and a set of rechargeable NiMH (Nickel-metal hydride) batteries is also presented. Finally, some options for reducing the energy consumption are presented, which allow the system to operate for twelve days using four AA NiMH batteries.

Keywords: *Arduino, datalogger, data recording, microSD card, real-time clock.*

1. Introducción

En muchas aplicaciones se requiere de un registro contínuo y autónomo de mediciones durante largos períodos, que pueden ir desde horas hasta meses; por ejemplo: mediciones meteorológicas, de estudios de suelos, deformaciones en edificios volcánicos, etc. Los instrumentos utilizados en estas aplicaciones proporcionan información invaluable sobre el comportamiento de un proceso. Sin embargo, la compra de equipo de registro de datos puede ser un problema para laboratorios con presupuesto limitado. Una de las opciones más económicas son los registradores de datos basados en USB como los desarrollados por DLP Design [Design, 2014]. Cuestan alrededor de 50 dólares americanos, cuentan con 8 canales que se pueden configurar como entrada/salida digital o entrada analógica a un convertidor analógico-digital de 10 bits. Presentan el inconveniente de requerir de la conexión constante a una computadora.

Los registradores de datos de tipo autónomo (que no requieren estar conectados a una computadora), tienen un costo a partir de los 140 dólares americanos, y muy comúnmente se trata de dispositivos que miden temperatura y humedad como el RHT20 de Extech [Extech, 2013]. Se pueden encontrar modelos que tienen entradas analógicas de propósito general como el Onset HOBO U12-013, a un costo de 150 dólares [Onset, 2016]. Este modelo cuenta con sensores de humedad y temperatura ambiente, y dos entradas adicionales de 0 a 2.5 V que se conectan a

un convertidor analógico-digital de 12 bits. Los precios mencionados no incluyen licencia de software, impuestos, ni gastos de envío.

Los sistemas comerciales de registro de datos están orientados a que el cliente adquiera los sensores desarrollados por la misma compañía. Esto deja fuera de uso la gran variedad de sensores que existen en el mercado, en particular los que se comunican con los protocolos seriales I^2C (Inter-integrated circuit) o SPI (Serial Peripheral Interface).

2. Métodos

Si se desea construir un sistema de registro de datos flexible y de bajo costo, se puede optar por un diseño basado en un microcontrolador; o bien, basado en una plataforma de hardware libre. El uso de un microcontrolador es una alternativa que presenta el máximo de flexibilidad, pues se puede diseñar el sistema para necesidades muy específicas. Los microcontroladores suelen incluir módulos internos con funciones de conversión analógica a digital y reloj de tiempo real, útil este último para añadir una marca de tiempo a los datos registrados. Sin embargo, los microcontroladores suelen contar con una capacidad limitada de memoria no-volátil para el registro de las mediciones. Por otro lado, la recuperación de los datos requiere del desarrollo de un programa adicional.

Entre las plataformas de hardware que se pueden considerar se encuentran: Arduino, Raspberry Pi (con un costo de 40 dólares americanos), y Beaglebone (Con un costo de 56 dólares americanos). La primera es más económica y más fácil de manejar que las segundas, pues éstas requieren de la instalación y manejo de Linux como sistema operativo. Otra opción es la tarjeta STM32 Nucleo-32 (con un costo de 11 dólares americanos), cuya disposición de pines es compatible con la tarjeta Arduino Nano, y está basada en un microcontrolador ARM de 32 bits; o bien, alguna de las tarjetas Curiosity basadas en microcontroladores de 8, 16 y 32 bits de Microchip, y con un costo a partir de los 30 dólares.

De entre las opciones mencionadas, la plataforma Arduino destaca por su fácil adquisición, programación sencilla, bajo costo, y gran cantidad de bibliotecas de uso libre para el manejo de sensores y módulos diversos. Una de las limitantes de esta

plataforma es la escasa capacidad de depuración del programa en comparación con el resto de las herramientas mencionadas, ya que no es capaz, por ejemplo, de hacer ejecuciones paso a paso.

En este documento se detalla el procedimiento para construir un sistema de medición y registro de datos (*datalogger*) utilizando una tarjeta Arduino Nano ($160 pesos, para la versión clon), que es una de las tarjetas más económicas de la familia, y está basada en un microcontrolador Atmel ATmega 328P de 8 bits. El código de programa y las conexiones son totalmente compatible con la tarjeta Arduino UNO, ya que ambas utilizan el mismo procesador. Se seleccionó esta plataforma debido a que es de hardware abierto, y cuenta con numerosas bibliotecas para el manejo de sensores, por lo que fácilmente se podrá modificar el sistema propuesto para necesidades particulares. Para complementar el funcionamiento, a la tarjeta se le añadió un módulo de memoria microSD y un módulo reloj en tiempo real (figura 1).

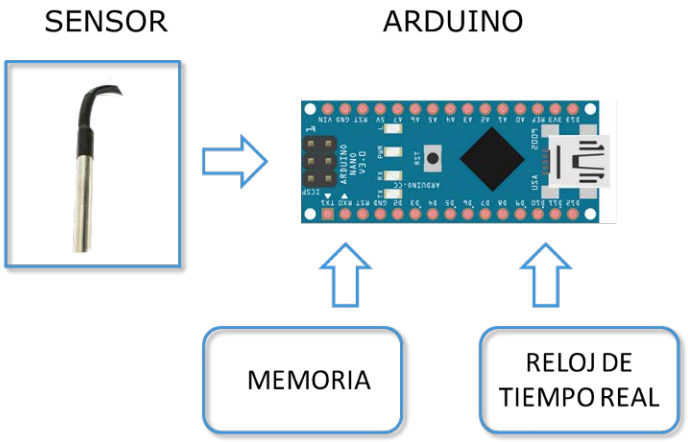

Figura 1 Esquema a bloques del sistema de registro de datos.

Para el almacenamiento de datos se seleccionó un módulo genérico para tarjeta microSD, el cual tiene un costo de $60 pesos (sin incluir la tarjeta microSD), que proporciona al Arduino una gran capacidad de almacenamiento de datos. No se tuvo ningún problema al utilizar una tarjeta de 8 GBytes. Esto, en comparación con la capacidad de almacenamiento propia de la tarjeta Arduino que es de sólo 1 KByte [Arduino, 2018]. En el mercado es posible encontrar módulos de expansión microSD

originales y clones. En la figura 2 se muestra un ejemplo de ambas versiones. En la tabla 1 se indican la equivalencia de pines entre los módulos. En este caso, el módulo cuenta con pines para su inserción a una tarjeta protoboard.

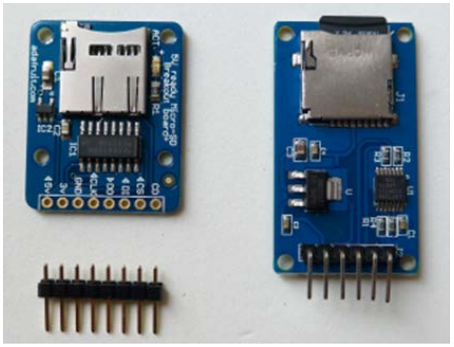

Figura 2 Módulos microSD: Adafruit original (Izquierda) y clon genérico (derecha).

Tabla 1 Equivalencia entre pines de los módulos Adafruit y clon.

Módulo Adafruit	**Módulo clon**
5 V	VCC
3 V	No existe
GND	GND
CLK	SCK
D0	MOSI
DI	MISO
CS	CS
CD	No existe

Para utilizar el módulo microSD clon se puede partir de la información y las bibliotecas de programación del fabricante Adafruit [Adafruit, 2018]. La comunicación entre la tarjeta Arduino y el módulo microSD utiliza el protocolo SPI (*Serial Peripheral Interface*), que fue desarrollado para el intercambio de datos entre un microprocesador y módulos periféricos. Este protocolo maneja líneas separadas para envío, recepción de datos (comunicación full-duplex), más una señal de reloj. Por otro lado, el protocolo I^2C (*Inter Integrated Circuit*), utiliza una sola línea para el envío y recepción de datos [Leens, 2009], [Linke, 2008]. Con el protocolo SPI se puede alcanzar una velocidad de transmisión de 20 MHz, en comparación con 3.4 MHz del protocolo I^2C [NXP, 2014], [Usach, 2015]. En la tabla 2 se indica una descripción de la función de cada pin del módulo.

Tabla 2 Descripción de la operación de los pines del módulo microSD.

PIN	Función
VCC	Entrada de alimentación +5 Volt
GND	Conexión a tierra
SCK	Señal de reloj
MOSI	Entrada de datos
MISO	Salida de datos
CS	Señal de activación del módulo

Para que la tarjeta microSD pueda ser utilizada por el módulo es necesario que el formato sea de tipo FAT16 o FAT32. Para conectar el módulo microSD a la tarjeta Arduino se utilizó el diagrama de la figura 3. A manera de demostración, al sistema se le ha añadido un sensor de temperatura integrado LM35 [Texas Instruments, 2016], cuya salida de voltaje es equivalente a 10 mV/°C.

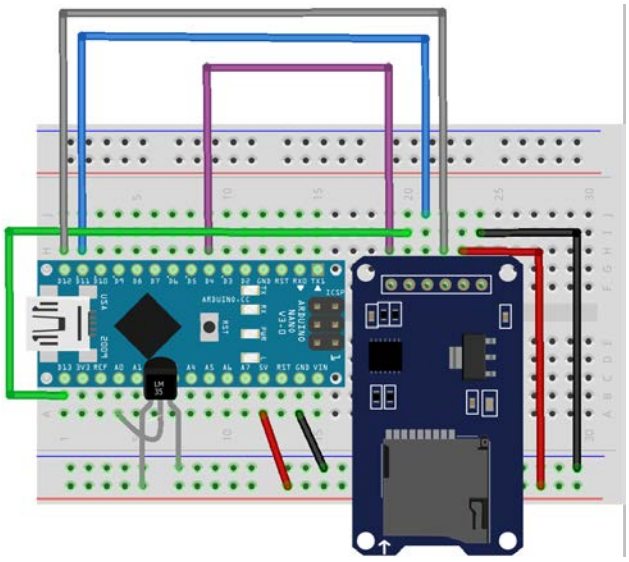

Figura 3 Diagrama de conexiones del módulo microSD, la tarjeta Arduino NANO y un sensor de temperatura LM35.

La conexión entre la tarjeta Arduino y el módulo microSD se hizo de acuerdo con la tabla 3, la cual es compatible con tarjetas Arduino NANO o Arduino UNO. Los pines digitales D11, D12 y D13, no se pueden intercambiar por otros, debido a que se encuentran conectados con el módulo interno de comunicación SPI (pines 15 a 17 del microcontrolador). Por otro lado, el pin digital D4, que controla la señal *Chip Select,* sí se puede intercambiar por otro pin, previo ajuste del código.

Tabla 3 Conexión del módulo microSD a la tarjeta Arduino.

Módulo microSD	Arduino
CS	D4
MOSI	D11
MISO	D12
CLK	D13
VCC	5 V
GND	GND

Utilizando un osciloscopio digital Keysight DSOX1102G (70 MHz, 2 GS/s) se pudo medir la duración del ciclo *loop* (cuando se lee un solo canal) y se estableció que es de aproximadamente 16.3 ms, de tal manera que añadiendo un retardo de 984 ms se podrían obtener datos cada 1000 ms (esto se comprobó con el osciloscopio y se obtuvo que la frecuencia de registro de datos fue de 0.9998 Hz).

El programa de ejemplo que proporciona el ambiente de programación (IDE, *Integrated Development Environment*) del Arduino fue modificado para leer un solo sensor:

```
/*  Datalogger_SD_1ch.ino   Programa modificado del código de ejemplo.
 Se requieren las siguientes conexiones
 CS -> D4
MOSI -> D11
MISO -> D12
CLK -> D13
VCC -> 5 V
GND -> GND
Debe haber un sensor analógico en el canal A0  */
#include <SPI.h>
#include <SD.h>
const int chipSelect = 4;
const int analogPin = 0;
int sensor = 0;
void setup() {
  // Inicializa la comunicación serial
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  Serial.print("Inicializando tarjeta SD ...");
// see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
```

```
    Serial.println("Fallo en tarjeta, o no hay tarjeta");
    // No hace nada más:
    return;
  }
  Serial.println("Tarjeta lista");
  pinMode(7, OUTPUT);  // pin testigo
}
void loop() {
  digitalWrite(7, HIGH);
  // make a string for assembling the data to log:
  String dataString = "";
  // lee el sensor y genera una cadena:
 sensor = analogRead(analogPin);
    dataString += String(sensor);
  // Abre el archivo en la memoria SD
  File dataFile = SD.open("datalog.txt", FILE_WRITE);
  // Revisa si el archivo está disponible para escribir:
  if (dataFile) {
    dataFile.println(dataString);
dataFile.close();
    // Envía la cadena también por el puerto serial
    Serial.println(dataString);
  }
  // Si el archivo no se encuentra abierto, manda un error:
  else
{
    Serial.println("Error al abrir el archivo datalog.txt");
  }
  digitalWrite(7, LOW);

  delay(984);   // Toma un dato cada segundo
  //delay(59984);   // Toma un dato cada minuto
  //delay(299984);  // Toma un dato cada 5 minutos
  //delay(599984);  // Toma un dato cada 10 minutos
}
```

Al final del listado se añadió un conjunto de comandos *delay*, los cuales permiten cambiar la velocidad de muestreo entre un segundo; o bien, 1, 5, y 10 minutos.

Una vez que se ha cargado el programa en la tarjeta Arduino comenzará automáticamente el registro de datos en el archivo *datalog.txt*. En esta versión, el archivo de datos no se sobrescribe cada vez que se enciende la tarjeta Arduino, sino que los nuevos datos son añadidos a los ya existentes.

El sistema descrito consume 21 mA de corriente (medidos con un multímetro portátil Fluke modelo 87V, ±1% de exactitud ± 2 dígitos), por lo que, si se alimenta con una pila de respaldo de celular de 2,000 mAh de capacidad, teóricamente será posible registrar datos durante 100 hrs (4.16 días). Se hizo una prueba con una pila de respaldo genérica, con las citadas características. Contrario al dato esperado, se encontró que el sistema sólo duró funcionando 44 horas, partiendo de la carga completa, proporcionada por un cargador Panasonic BQ-390 (capacidad: cuatro pilas AA o dos pilas AAA). Este resultado se puede deber a una especificación incorrecta por parte del fabricante, o a una excesiva pérdida de energía debido al convertidor CD-CD interno de la batería, el cual se encarga de convertir el voltaje de 3.7 V de una pila de litio en los 5 V de salida del conector USB (figura 4), donde se observa el convertidor DC-DC (indicado con la flecha roja).

Figura 4 Pila de respaldo de celular genérica).

Módulo Reloj-Calendario TinyRTC DS1307

Para añadir un reloj-calendario al sistema, y almacenar cada dato con hora y fecha, se utilizó un módulo TinyRTC [Gizmo, 2016]. Este módulo, que tiene un costo de $70 pesos, está basado en el reloj de tiempo real (*Real-Time Clock*) DS1307 y cuenta con un socket para una pila de litio de 3 V (CR2032 o similar). La configuración y uso del módulo se realizaron mediante un protocolo serial de comunicación I^2C, cuyas bibliotecas se encuentran disponibles en la plataforma del Arduino. Un módulo TinyRTC genérico se muestra en la figura 5, y una descripción de sus pines en tabla 4.

Las conexiones entre el módulo TinyRTC y la tarjeta Arduino se enlistan en la tabla 5. La tarjeta Arduino utiliza los pines A4 y A5 para la comunicación mediante el protocolo I^2C, por lo que no estarán disponibles para su uso como entradas analógicas. Un posible diagrama de conexiones, que incluye el módulo microSD, se muestra en la figura 6.

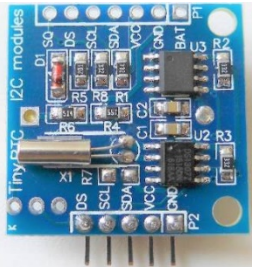

Figura 5 Módulo TinyRTC.

Tabla 4 Descripción de los pines del módulo TinyRTC.

Pin	Función
DS	Línea de datos de un sensor de temperatura DS18B20 (no instalado)
SCL	Reloj
SDA	Datos
VCC	Alimentación +5 V
GND	Tierra

Tabla 5 Conexiones entre el módulo TinyRTC y la tarjeta Arduino.

TinyRTC	Arduino
DS	
SCL	A5
SDA	A4
VCC	5V
GND	GND

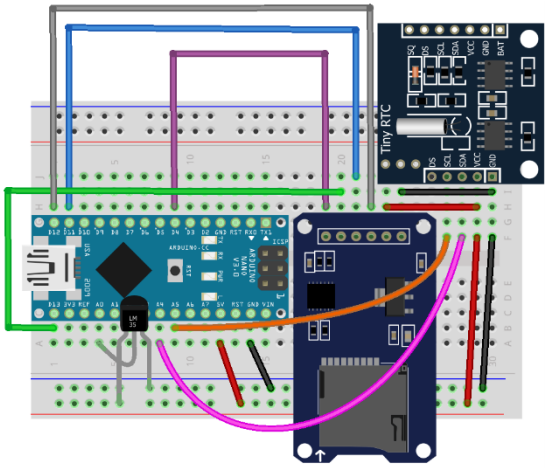

Figura 6 Diagrama de conexiones del módulo TinyRTC, el módulo microSD, la tarjeta Arduino y un sensor de temperatura LM35.

Para utilizar el módulo TinyRTC se requiere la biblioteca RTClib [Jeelab, 2017]. A continuación se muestra el código que permite realizar una lectura del convertidor analógico-digital cada segundo, a partir de la información del reloj de tiempo real, y almacena cada dato junto con su marca de tiempo en la memoria microSD:

```
//Datalogger_simple2a_60seg.ino
/*
 * Se corrige el encabezado del archivo CSV
 */
#include <SD.h>                    //Bibliotecas
#include <Wire.h>
#include <RTClib.h>
RTC_DS1307 RTC;                    // Crea un objeto de tipo reloj de tiempo real
const int chipSelect = 4;          //CS pin del módulo
const int analogPin = 0;
String nameFile = "";
File logfile;                      //Variable tipo File
void setup() {
SD.begin(chipSelect);
RTC.begin();
Wire.begin();
Serial.begin(9600);
  // create a new file
 char filename[] = "LOGGER00.CSV";
 for (uint8_t i = 0; i < 100; i++) {    //Genera el nombre de archivo
 filename[6] = i/10 + '0';
 filename[7] = i%10 + '0';
 if (! SD.exists(filename)) {
 // only open a new file if it doesn't exist
 logfile = SD.open(filename, FILE_WRITE);
  //logfile.close();
  break;  // leave the loop!
   }
 }
 if (! logfile) {
   Serial.println("Error al abrir el archivo");
 }
Serial.print("Logging to: ");
Serial.println(filename);
nameFile = filename;
logfile=SD.open(filename, FILE_WRITE);          //Will open and will write once just
for headers
logfile.println(" Dato, Fecha, Hora");     //Print headers (not saved yet)
```

```
logfile.close();                              //Print saved
}
void loop(){
DateTime now = RTC.now();                     //Clock call
 if(now.second()==0){                         //Toma una muestra por minuto
logfile=SD.open(nameFile, FILE_WRITE);        //Will open and will write date
now =RTC.now();
String dataString = "";
// lee el sensor y genera una cadena:
int sensor = analogRead(analogPin);
float temp = 500*sensor/1024.0;
dataString += String(temp) + "," + getDate() + "," + getTime();
logfile.println(dataString);                  //Print date and time(not saved yet)
logfile.close();   //Print saved
Serial.println(dataString);                   //Print date and time(not saved yet)
 }
  delay(1000);                                //Evita más de una lectura en el mismo segundo 00
}
String getTime(void)
 {       // Lee la hora del RTC
  String returnString = "";
  DateTime now = RTC.now();
  now =RTC.now();
  if (now.hour() < 10)
  returnString += "0" + String(now.hour());
  else
  returnString += String(now.hour());
  returnString += ":";
  if (now.minute() < 10)
  returnString += "0" + String(now.minute());
  else
   returnString += String(now.minute());
  returnString += ":";
  if (now.second() < 10)
  returnString += "0" + String(now.second());
  else
  returnString += String(now.second());
  return returnString;
}
String getDate(void)
 {       // Lee la fecha del RTC
  String returnString = "";
  DateTime now = RTC.now();
  now =RTC.now();
```

```
  if (now.day() < 10)
    returnString += "0" + String(now.day());
  else
  returnString += String(now.day());
  returnString += "/";
  if (now.month() < 10)
  returnString += "0" + String(now.month());
  else
  returnString += String(now.month());
  returnString += "/";
  if (now.year() < 10)
  returnString += "0" + String(now.year());
  else
  returnString += String(now.year());
  return returnString;
}
```

A diferencia del anterior programa, en este último, se cambia automáticamente el nombre del archivo de destino incrementándose de *logger00* a *logger99*, lo cual facilitó la diferenciación de los diferentes registros; además, son compatibles con hojas de cálculo. En la figura 7 se muestra la tarjeta Arduino Nano, junto con los módulos microSD y RTC montados en una tarjeta protoboard.

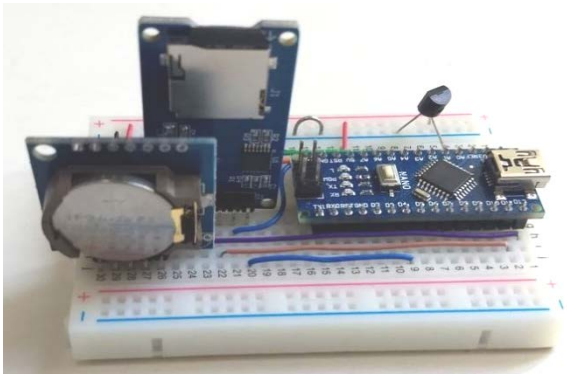

Figura 7 Fotografía del sistema montado en una tarjeta protoboard.

3. Resultados

Se construyó un solo sistema de registro de datos. La operación del convertidor analógico-digital de la tarjeta Arduino toma como referencia el voltaje de alimentación; por lo tanto, para evaluar la influencia de la alimentación en su

operación, se probaron cuatro fuentes distintas: una batería de respaldo de celular, un cargador de celular, una fuente de laboratorio y un juego de cuatro pilas recargables AA de NiMH. Las pruebas se realizaron de manera secuencial. En ellas se guardó un dato de temperatura cada minuto, proporcionado por un sensor de temperatura LM35, hasta completar 1000 lecturas. Los datos adquiridos se muestran graficados en la figura 8, las muestras se tomaron secuencialmente y las gráficas se encuentran desfasadas verticalmente para mayor claridad.. Resulta práctico considerar como fuente de energía a una pila de respaldo de celular, debido a la gran capacidad que tienen. Sin embargo, se aprecia que al utilizarla; o bien, al emplear el cargador de celular, las lecturas presentan un nivel de ruido mayor. Esto es debido al convertidor de voltaje CD-CD de tipo conmutado con que cuentan. El nivel de ruido disminuye notablemente al utilizar una fuente de laboratorio de tipo lineal o las pilas recargables, presentándose únicamente el error típico del convertidor analógico-digital (± 2 LSB) [Atmel, 2016].

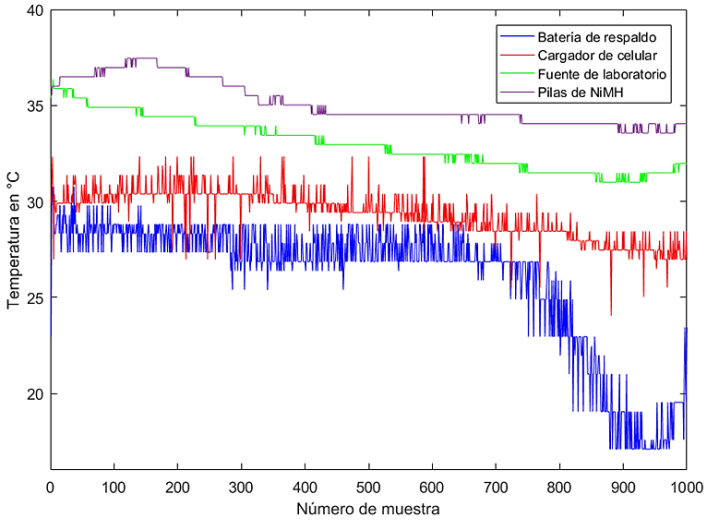

Figura 8 Secuencia de registro del sistema implementado utilizando cuatro tipos de fuentes de voltaje.

4. Discusión

La utilidad de un sistema de registro de datos se basa, además de en una adecuada exactitud en la medición, en el nivel de autonomía. Esto depende de la capacidad de almacenamiento y de la duración del suministro de energía. El uso de

una tarjeta microSD para el almacenamiento de datos en un sistema de registro automático proporciona una capacidad muy alta, donde con facilidad se puede medir la evolución de un proceso por espacio de meses. La limitante es entonces la fuente de alimentación, en especial si se requiere que el volumen o el peso del dispositivo sean reducidos. Para la tarjeta Arduino Nano se midió un consumo de corriente de 21 mA. Este valor puede bajar a 18 mA si se desconecta el LED que indica la alimentación de la tarjeta (PWR), y a 7 mA si se utilizan las funciones de bajo consumo de energía y se activa el modo SLEEP del Arduino. A esto, sin embargo, se le debe añadir el consumo de los módulos microSD y RTC que suman 5.5 mA. Esto deja claro que el sistema tiene capacidades limitadas de ahorro de energía, lo cual se debe fundamentalmente a que no se cuenta con la opción de que el microcontrolador opere a una menor frecuencia que la de 16 MHz con que trabaja por omisión. En estas condiciones, se utilizó un paquete de cuatro pilas recargables tamaño AA de NiMH con una capacidad nominal de 1,900 mAh. Con esta alimentación, fue posible registrar un dato de temperatura cada minuto, con hora y fecha, por espacio de 12 días. El archivo resultante tuvo una extensión de 402 kB. Esto significa que una memoria de 8 GB de capacidad podría contener casi 20 mil experimentos como el indicado. Por otro lado, si no se requiere una gran precisión en la marca de tiempo, es suficiente con utilizar el programa de manejo de la tarjeta microSD, con lo que se simplifica el sistema y se reduce su costo.

Se estima que el costo global del sistema presentado se aproxima a los $1 200 pesos. Esto incluye además de la tarjeta Arduino y los módulos de memoria y reloj de tiempo-real, el gabinete, una memoria microSD de 8 GB, las pilas, el adaptador para pilas, y el cargador. De esta manera, es posible que para algunas tareas de medición sea posible encontrar un dispositivo comercial más económico.

5. Conclusiones

La tarjeta Arduino Nano constituye una opción económica y de fácil configuración para el desarrollo de múltiples aplicaciones. En este trabajo se presentó el desarrollo de un sistema de medición y registro de datos basado en dicha tarjeta, al que se conectó un módulo de memoria microSD y un módulo de reloj en tiempo real. Como

ejemplo de aplicación se utilizó un sensor de temperatura LM35 y se incluyen los códigos fuente de los programas. Como la tarjeta Arduino Nano no está orientada a aplicaciones de bajo consumo, la duración de los registros está ligada al tipo de fuente de energía utilizada. Sin embargo, fue posible utilizar el sistema por espacio de 12 días, alimentándolo con cuatro pilas recargables AA de NiMH. El sistema es de bajo costo y flexible, ya que se puede reconfigurar para funcionar con una gran cantidad de sensores para los cuales existen bibliotecas de uso libre. Otra ventaja es la fácil recuperación de los datos, ya que se almacenan en una memoria microSD que se puede leer en cualquier computadora personal.

El sistema presentado servirá como base para desarrollar un registrador de datos de humedad y pH en suelos artificiales (tecnosuelos). Se requiere la medición en múltiples puntos, por lo que el bajo costo del sistema desarrollado permitirá construir varios registradores sin que eso represente un presupuesto elevado.

Agradecimientos

Este trabajo forma parte del proyecto PAPIIT DGAPA-UNAM IN108118 Construcción y monitoreo de tecnosuelos con materiales de desecho para la revegetación urbana.

6. Bibliografía y Referencias

[1] Adafruit. MicroSD card breakout board tutorial. https://learn.adafruit.com/adafruit-micro-sd-breakout-board-card-tutorial?view=all, mayo de 2018.

[2] Arduino. (2018). Arduino Nano. https://store.arduino.cc/usa/arduino-nano.

[3] Atmel. (2016). ATmega328/P Datasheet complete. In: Atmel Corporation.

[4] Design, D. (2014). DLP-IO8-G 8-Channel Data Acquisition Board. http://www.dlpdesign.com/usb/io8.php, agosto de 2018.

[5] Extech (2013). RHT20 Humidity/Temperature datalogger. http://www.extech.com/resources/RHT20_DS-en.pdf, agosto de 2018.

[6] Gizmo. (2016). TinyRTC I2C Module. Gizmo Mechatronix Central. 6 págs.

[7] Jeelab. (2017). RTC Library. https://github.com/adafruit/RTClib, junio de 2018

[8] Leens, F. (2009). An introduction to I^2C and SPI protocols. *IEEE* Instrumentation *& Measurement Magazine, 12*(1), 8-13. doi:10.1109/MIM.2009.4762946.

[9] Linke, B. (2008). Overview of 1-Wire Technology and its Use. https://www.maximintegrated.com/en/app-notes/index.mvp/id/1796, agosto 2018.

[10] NXP. (2014). I2C-bus specification and user manual. NXP Semiconductors.

[11] Onset. (2016). HOBO U12-013 data logger. http://www.onsetcomp.com/products/data-loggers/u12-013, agosto de 2018.

[12] Texas Instruments, Inc. (2016). LM35 Precision Centigrade Temperature Sensors. Texas Instruments Inc. 31 págs.

[13] Usach, M. (2015). AN-1248 Application Note. SPI Interface. Analog Devices Inc. 8 págs.