Servicio Meteorológico Nacional
Argentina

Puesta en valor de adquisidores de datos Campbell Scientific Inc. CR10X

Nota Técnica SMN 2023-141

Julián A. Lell[1,2]

[1] *Dirección Central de Monitoreo del Clima, Servicio Meteorológico Nacional*

[2] *Grupo de Estudios de la Radiación Solar, Universidad Nacional de Luján*

Junio 2023

Ministerio de Defensa
Presidencia de la Nación

Resumen

Los adquisidores de datos Campbell Scientific Inc. modelo CR10X fueron introducidos al mercado en 1995, y discontinuados probablemente en 2004, con la introducción del modelo CR1000. Desde un punto de vista instrumental, los adquisidores CR10X siguen siendo equipos de gran calidad y precisión (estando debidamente calibrados), pero han quedado obsoletos desde un punto de vista operativo: el método de programación requiere comandos *ad hoc*, que podrían considerarse "de bajo nivel", no pudiendo utilizarse el lenguaje de alto nivel "CRBasic" (introducido en 2000 con el adquisidor modelo CR5000) de mayor versatilidad y utilizado hasta la actualidad. Adicionalmente, los dispositivos CR10X poseen un único puerto de conexión "CS I/O", propietario de Campbell Scientific Inc., y carecen de otras opciones de conectividad como puertos RS-232, Ethernet, USB, etc.; esta carencia de alternativas de conectividad hace que la programación y recuperación de datos requiera de accesorios tales como el "SC32A" (interfaz RS-232 optoacoplada), el cual ha sido discontinuado por la firma. En el presente documento se presenta un desarrollo que permite el acceso completo al adquisidor CR10X a través de un programa en lenguaje Python, utilizando un adaptador implementado en un módulo "Arduino UNO", de manera que resulta posible utilizar este modelo de adquisidores en cualquier tipo de plataforma (Linux, Windows, etc.), sin la necesidad de contar con accesorios o *software* original de la firma Campbell Sci. Inc..

Abstract

Campbell Scientific's CR10X dataloggers were introduced on the market in 1995, and discontinued in 2004, with the introduction of CR1000 model. From an instrumental point of view, CR10X dataloggers are devices of great quality and accuracy even to this day (provided they're properly calibrated), but they've been rendered obsolete from a functional point of view: the programming method requires *ad hoc* commands, which can be considered low-level, not being possible the use of the high-level language "CRBasic" (introduced in 2000 with the model CR5000), of greater versatility and used in the present day. On the other hand, CR10X devices feature a single connection port "CS I/O", Campbell Sci. Inc. proprietary, and lack any other connectivity option like RS-232 ports, Ethernet, USB, etc.; this lack of connectivity alternatives makes the programming and retrieval of data to require accessories such as the "SC32A" (opto-coupled RS-232 interface), which has been discontinued by the company. This document presents a development which allows for complete access to CR10X datalogger through a script written in Python language, using a hardware adapter implemented in "Arduino UNO" module, making possible the use of this model of dataloggers with any kind of platform (Linux, Windows, etc.), without the need of accessories neither original software from Campbell Sci. Inc..

Palabras clave: CR10X, *datalogger*, Campbell Scientific, actualización, puesta en valor.

Citar como:

Lell, J. A., 2023: Desarrollo de *software* y *hardware* para la puesta en valor de adquisidores de datos Campbell Scientific CR10X. Nota Técnica SMN 2023-141.

1. INTRODUCCIÓN

Con el paso de los años, es común que los instrumentos científicos se vuelvan obsoletos, fundamentalmente debido a la falta de soporte del fabricante, y a la desaparición de repuestos y periféricos necesarios para su utilización; adicionalmente, la introducción de nuevos equipos con mayores prestaciones (memoria, velocidad, conectividad, lenguajes de programación más flexibles, accesorios, etc.) también marca la obsolescencia de un instrumento perteneciente a una generación previa. En el campo de los adquisidores de datos (en adelante denominados *dataloggers*), los avances en el estado del arte de la microelectrónica han significado un aumento de la capacidad de almacenamiento, aumento en la velocidad de procesamiento de datos e incorporación de nuevas opciones de conectividad; sin embargo, el núcleo de un *datalogger*, constituido por un conversor analógico-digital (ADC por su sigla en inglés), un multiplexor analógico, una memoria “cache” y un reloj, no ha sufrido variaciones de relevancia en las últimas décadas, y por tal motivo es posible obtener mediciones precisas con *dataloggers* de 30 años de antigüedad o más, siempre y cuando se encuentren debidamente calibrados.

Los *dataloggers* CR10X son instrumentos de gran calidad, los cuales cuentan con 6 canales diferenciales (pudiendo utilizarse como 12 canales SE -*single ended*- independientes), y puertos de entrada/ salida digital, entre otras características. Sin embargo, podría decirse que estos instrumentos han quedado obsoletos debido a la carencia de una serie de características: no poseen conectividad propia por RS-232, USB, Ethernet o WI-FI, sino solo a través de periféricos de elevado costo (muchos de ellos discontinuados); adicionalmente la programación de estas unidades no puede realizarse a través del lenguaje CRBasic (propietario de Campbell Scientific y utilizado en todos sus dispositivos a partir del año 2000).

En este trabajo se presenta el desarrollo de un módulo clasificable como “interfaz serial multimodo” (en adelante ISM), el cual permite la conectividad con un *datalogger* CR10X a través de un puerto USB. La ISM es complementada con un *script* de Python que permite un acceso amplio a la memoria y registros del CR10X, permitiendo incluso la lectura del programa y la re-programación. De esta forma, el CR10X adquiere las prestaciones más relevantes de los equipos modernos, e incluso los supera gracias a la posibilidad de acceso a los datos con *software* de código abierto desde cualquier sistema operativo (los equipos modernos de la firma Campbell Scientific requieren del *software* propietario y de código cerrado “LoggerNet”, el cual solo se encuentra disponible para plataformas Microsoft Windows).

2. EL *DATALOGGER* CAMPBELL SCIENTIFIC CR10X

El datalogger CR10X posee un panel frontal con terminales tipo “bornera”, más un conector DB9-F (denominado por la fábrica “CS I/O”). La figura 1 muestra una fotografía del *datalogger*.

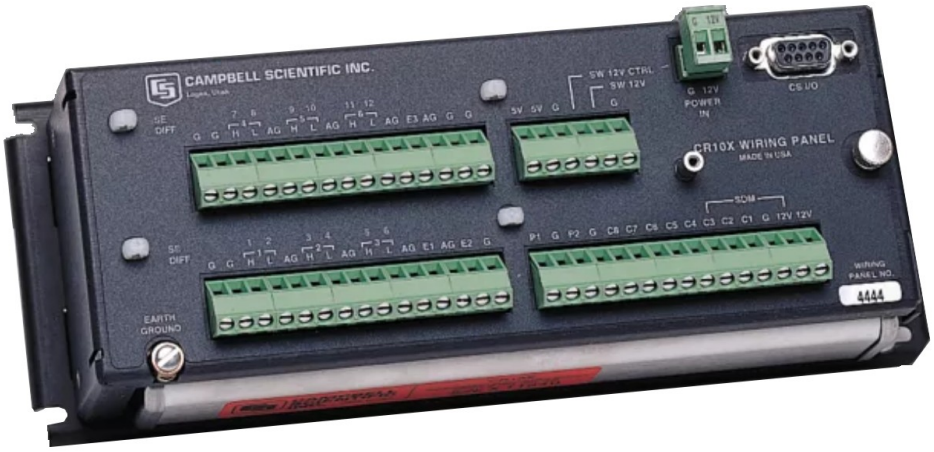

Figura 1: Panel frontal del *datalogger* CR10X.

El puerto CS I/O del CR10X posee la siguiente configuración de pines (conector presentado en la fig. 2):

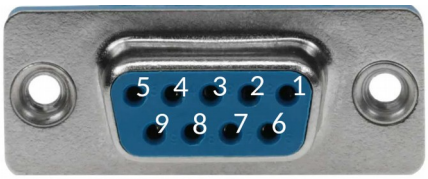

Figura 2: Conector DB9-F.

1:Salida 5V **2:Tierra (GND)** **3:Entrada “RING”** **4:Entrada serial “RX”** **5:Salida “ME”**
6: Salida “SDE” 7: E/S de reloj (CLK/HS) 8:Salida 12V **9: Salida serial “TX”**

Es importante destacar que la configuración de pines de este conector difiere completamente respecto a la utilizada históricamente en conexiones de tipo serial RS-232 y RS-485. De hecho, si se conecta un adaptador comercial de RS-232 a USB, el mismo quedará inutilizado debido a la presencia de tensiones de alimentación de 5 y 12V en el conector del CR10X.

3. INTERFAZ SERIAL MULTIMODO

La ISM está constituida por un módulo “Arduino UNO” (basado en los chips ATmega328P y ATmega8U2). La función de la Interfaz es permitir la comunicación serial (UART) con el *datalogger* CR10X, mediante la conexión a PC a través del puerto USB (ATmega8U2).

La salida de 5V del CR10X es utilizada para alimentar al Arduino UNO. Luego, la comunicación se realiza a través de los pines 4 y 9, con los siguientes parámetros: comunicación asincrónica de 8 bits, sin paridad, bit de inicio y bit de parada, a 300/1200/9600/76800 baudios. Lo más importante a tener en cuenta es que los niveles lógicos utilizados por el CR10X se encuentran invertidos respecto a los utilizados por el módulo Arduino, y es necesario corregir esto por *software* (programa en Arduino). Una vez conectada la ISM al *datalogger*, el protocolo de conexión es el siguiente:

- El pin 3 (RING) debe ser puerto en HIGH (5V) por el Arduino para indicar al CR10X que se desea iniciar la comunicación.
- El *datalogger* responderá poniendo en HIGH el pin 5 (ME), y lo mantendrá en este estado mientras dure la comunicación, o hasta transcurridos 40 segundos después de recibido el último caracter desde el Arduino.
- Para evitar la desconexión del CR10X, el Arduino pondrá en HIGH la entrada RING cada vez que detecte que la salida ME se ha puesto en LOW (0V), lo cual ocurre típicamente luego de 40 segundos de inactividad.

En la figura 3 se indican todas las conexiones necesarias para establecer y mantener la comunicación serial entre el CR10X y el módulo Arduino UNO.

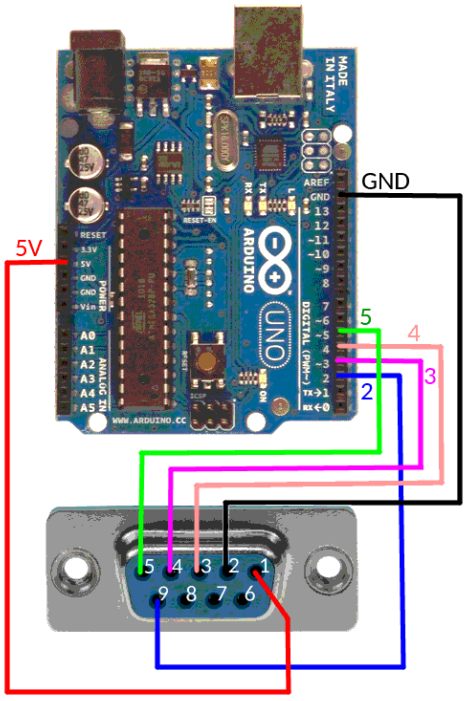

Figura 3: Conexión entre el datalogger CR10X y el módulo Arduino (ISM)
CR10X:Arduino: 1:5V, 2:GND, 3:4, 4:3, 5:5, 9:2.

Las líneas de código (a cargarse en el Arduino) que cumplen las funciones mencionadas arriba son las siguientes:

```
1.#include <SoftwareSerial.h>  // Librería para comunic. Seriales en puertos digitales.
2.SoftwareSerial mySerial(2, 3, true); // RX, TX.  El parámetro "true" indica
                                        lógica invertida para el puerto serial
3.                    // emulado por software con la librería "SoftwareSerial".
4.void setup() {
5.   Serial.begin(9600);  //Inicio el puerto serial.
6.   mySerial.begin(9600);//Inicio el p. serial emulado (invertido) para el CR10X.
7.   pinMode(4, OUTPUT);  //El pin 4 (Ard) se conecta al pin 3 (RING) del CR10X.
8.   pinMode(5, INPUT);   //El pin 5 (Ard) se conecta al pin 5 (ME) del CR10X.
9.   activar_CR10X();}    //Función que activa la comunicación del CR10X.

10.void loop() {
11.   if (digitalRead(5) == LOW){//Si el CR10X finalizó la comunicación (ME en LOW)
12.      delay(200);
13.      activar_CR10X();}       //Se llama a la función que activa el CR10X.
14.
15.   if (mySerial.available()){ //Si CR10X envió a mySerial [9(CR10X)→2 (Ard.)]...
16.      Serial.write(mySerial.read());} // … se reenvían al puerto serial (USB →
PC)

17.   if (Serial.available()){ //Si se recibe desde el puerto serial (USB)...
18.      mySerial.write(Serial.read());}}//se reenvía a mySerial[2(Ard.)→4(CR10X)]

19.void activar_CR10X() {
20.   digitalWrite(4, HIGH);  // Se aplica 5V a la entrada RING del CR10X [4
(Arduino) → 3(CR10X)]
21.   delay(1000);                // Retardo para darle tiempo al CR10X
22.   while(digitalRead(5) == LOW){  // Mientras no haya respuesta del CR10X (ME en
LOW)
23.      digitalWrite(13, HIGH);} // Se enciende LED del Arduino (pin digital 13).
24.   digitalWrite(13, LOW); // Si hay respuesta del CR10X (ME=HIGH), apaga el LED.
25.   delay(100); // Retardo de 100 ms para poner RING en LOW.
26.   digitalWrite(4, LOW);  // Se pone LOW la entrada RING (CR10X está activo).
27.   delay(100);

28.   mySerial.print("\r"); //Se envía "retorno de carro" para que el CR10X pueda
29.   delay(80);// estimar la velocidad de transf. de datos (baud rate, 9600).
30.   mySerial.print("\r"); // Se envía otro "retorno de carro".

31.   while (mySerial.available()){ // CR10X responderá con retorno de carro, un
                                    "line feed", y un ">".
32.      mySerial.read();}}    // Es necesario leer estos caracteres para que no
                                 queden en el buffer
33.            // del puerto serial emulado por software (mySerial), puesto que no
34.            //  deben ser enviados a la PC.
```

4. CÓDIGO PYTHON PARA EL CONTROL DEL *DATALOGGER* CR10X

Con el fin de utilizar el *datalogger* CR10X al máximo de su capacidad, se elaboró un *script* que contiene las siguientes funciones:

1) Inicio del puerto serial.
2) Solicitud de inicio de comunicación.
3) Comprobación del estado y visualización del los "*flags*" de usuario.
4) Visualización de memoria utilizada, tensión de la batería interna, punteros y *flags* de error.
5) Solicitud de la fecha y hora del *datalogger*.
6) Actualización de la fecha y hora del *datalogger*.
7) Bajada de programa residente en el *datalogger*.
8) Subida de programa al *datalogger* (con compilación y ejecución inmediata en el instrumento).
9) Recuperación (completa) de mediciones almacenadas en la "memoria final".
10) Blanqueado de *flags* de error.

El código completo (programa para módulo Arduino y código Python) puede ser descargado de https://gitlab.smn.gov.ar/jlell_smn/control-con-python-e-interfaz-para-campbell-sci.-cr10x. El código es abierto y libre, y se ofrece sin garantías de buen funcionamiento.

4.1) Inicio del puerto serial

Esta función debe ser ejecutada antes de iniciar la comunicación. Dependiendo de las características del sistema operativo, el puerto serial asignado tendrá los prefijos "ttyUSB", "ttyACM" o "COM", debiendo seleccionarse el adecuado de forma manual.

```
def abro_serial(intento):
    try:
      ser.close()
    except:
      pass
    while True:
      #puerto = '/dev/ttyUSB'+str(intento)      #Opción 1 para Linux
      #puerto = '/dev/ttyACM'+str(intento)     #Opción 2 para Linux
      puerto = 'COM'+str(intento)                  #Opción única para Windows

       try:
        ser = serial.Serial(
          port=puerto,\
          baudrate=9600,\
          parity=serial.PARITY_NONE,\
          stopbits=serial.STOPBITS_ONE,\
          bytesize=serial.EIGHTBITS,\
            timeout=0)
        print("\nConectado al puerto serial: " + ser.portstr)
        break
       except:
        print('\nEl puerto serial '+puerto+' no está disponible.')
        intento = intento + 1
        if intento >= 5: intento = 0  #Se intenta abrir solo los primeros 5
    puertos.
```

```
    time.sleep(2)
  ser.flushInput()
  ser.flushOutput()
  return ser, intento
```

4.2) Solicitud de inicio de comunicación.
Esta función es realizada automáticamente por el módulo Arduino, por lo cual no debería ser necesaria.

```
def communication_request():
  ser.flushInput()
  ser.flushOutput()
  print('Enviando solicitud de conexión...')
  conectado = False
  line = []
  counter = 0
  online = False
  while conectado == False:
    ser.write(bytearray.fromhex("0d"))
    time.sleep(0.08)
    ser.write(bytearray.fromhex("0d"))
    time.sleep(0.08)
    ser.write(bytearray.fromhex("0d"))
    time.sleep(0.1)
    for c in ser.read():
      line.append(c)
      if len(line) == 3:
        if line == [13, 10, 42]:
          print('Ready')
          conectado = True
          online = True
        else:
          line = line[1:]
  line = []
  return online
```

4.3) Comprobación del estado y visualización del los "*flags*" de usuario.
La utilidad de esta función consiste fundamentalmente en conocer el estado de los *flags* de usuario (bits configurables en tiempo de ejecución que condicionan el funcionamiento del *datalogger).*

```
def registers_request():
  user_flags = [0,0,0,0,0,0,0,0]
  ser.reset_input_buffer()
  ser.reset_output_buffer()

  print('Enviando solicitud de registros')
  contador = 0
  conectado = False
  receiving_time = False
```

```
9.
10. send_order(ser,'K\r')
11.
12. line = []
13. while True:
14.   for c in ser.read():
15.     line.append(c)
16.   if len(line) == 3:
17.     if line == [75, 13, 10]:
18.       print('Obteniendo registros')
19.       contador = 0
20.       line = []
21.       break
22.
23. while True:
24.   for c in ser.read():
25.     line.append(c)
26.
27.   if line[-4:-2] == [127, 0]:
28.
29.     minutes = line[0]*256+line[1]
30.     seconds = (line[2]*256 + line[3])/10
31.     hours = int(minutes/60)
32.     minutes = minutes%60
33.
34.     if hours >= 10:
35.       string_hours = str(hours)
36.     else:
37.       string_hours = '0'+str(hours)
38.     if minutes >= 10:
39.       string_minutes = str(minutes)
40.     else:
41.       string_minutes = '0'+str(minutes)
42.     if seconds >= 10:
43.       string_seconds = str(seconds)
44.     else:
45.       string_seconds = '0'+str(seconds)
46.
47.     print('-----------------------------------------------')
48.     print('Reloj interno (HH:MM:SS.S)')
49.     print(string_hours+':'+string_minutes+':'+string_seconds)
50.
51.     user_flags[0] = line[4]%2
52.     user_flags[1] = int(line[4]/2)%2
53.     user_flags[2] = int(line[4]/4)%2
54.     user_flags[3] = int(line[4]/8)%2
55.     user_flags[4] = int(line[4]/16)%2
56.     user_flags[5] = int(line[4]/32)%2
57.     user_flags[6] = int(line[4]/64)%2
58.     user_flags[7] = int(line[4]/128)%2
59.
```

```
    print('-----------------------------------------------')
    print('Flags de usuario (Flag8....Flag1)')
    print(str(user_flags[7])+str(user_flags[6])+str(user_flags[5])+
      str(user_flags[4])+str(user_flags[3])+str(user_flags[2])+
      str(user_flags[1])+str(user_flags[0]))

    if len(line) >= 9:
      mantissa = 0
      bit_value = 0.5
      for i in range(8):
        mantissa = mantissa + (int(line[6]/(2**(7-i)))%2)*bit_value
        bit_value = bit_value*0.5
      for i in range(8):
        mantissa = mantissa + (int(line[7]/(2**(7-i)))%2)*bit_value
        bit_value = bit_value*0.5
      for i in range(8):
        mantissa = mantissa + (int(line[8]/(2**(7-i)))%2)*bit_value
        bit_value = bit_value*0.5
      data_1 = (-1)**(int(line[5]/128)%2)*mantissa*2**((line[5]%128)-64)

      print('-----------------------------------------------')
      print('Data 1:')
      print(data_1)

    if len(line) >= 13:
      mantissa = 0
      bit_value = 0.5
      for i in range(8):
        mantissa = mantissa + (int(line[10]/(2**(7-i)))%2)*bit_value
        bit_value = bit_value*0.5
      for i in range(8):
        mantissa = mantissa + (int(line[11]/(2**(7-i)))%2)*bit_value
        bit_value = bit_value*0.5
      for i in range(8):
        mantissa = mantissa + (int(line[12]/(2**(7-i)))%2)*bit_value
        bit_value = bit_value*0.5
      data_2 = (-1)**(int(line[9]/128)%2)*mantissa*2**((line[9]%128)-64)

      print('-----------------------------------------------')
      print('Data 2:')
       print(data_2)
     return 0

   contador = contador + 1
   if contador >= 500:
     print('Timeout: '+str(len(line)))
     return 0
 return 0
```

4.4) Visualización de memoria utilizada, tensión de la batería interna, punteros y *flags* de error.
Esta función resulta de mucha utilidad para realizar un diagnóstico del *datalogger*.

```
def data_request():
      1.  ser.reset_input_buffer()
      2.  ser.reset_output_buffer()
      3.  print('Sending data request')
      4.  error_counter = 0
      5.  conectado = False
      6.  receiving_time = False
      7.  line = []
      8.
      9.  ser.flushInput()
      10.  ser.flushOutput()
      11.  while conectado == False:
      12.
      13.    send_order(ser,'A\r')
      14.
      15.    for c in ser.read():
      16.      line.append(c)
      17.      if len(line) == 2:
      18.        if line == [65, 13]:
      19.          print('Recibiendo datos')
      20.          conectado = True
      21.          receiving_time = True
      22.          line = []
      23.        else:
      24.          error_counter = error_counter + 1
      25.          if error_counter == 100:
      26.            return 1
      27.          line = line[1:]
      28.    error_counter = 0
      29.    while receiving_time == True:
      30.      for c in ser.read():
      31.        line.append(c)
      32.        if len(line) > 50:
      33.          if line[-10] == 67:
      34.            try:
      35.              if 78+sum(line[:-9]) == int(bytes(line[-9:-5]).decode("utf-8")):
      36.                try:
      37.                  time_string = bytes(line[2:-11]).decode("utf-8")
      38.                  print('------Estado del sistema-------')
      39.                  #print(time_string)
      40.                  result1 = int(re.search('R(.*)F', time_string).group(1)[1:-2])
      41.                  print('Referencia (DSP location): ' + str(result1))
      42.                  result2 = int(re.search('F(.*)V', time_string).group(1)[1:-2])
      43.                  print('Final Storage Locations llenas: ' + str(result2))
      44.                  result3 = int(re.search('V(.*)A', time_string).group(1)[:-1])
      45.                  print('Versión del datalogger: ' + str(result3))
      46.                  result4 = int(re.search('A(.*)L', time_string).group(1)[:-1])
      47.                  print('Final Storage Area: ' + str(result4))
```

```
                result5 = int(re.search('L(.*)E', time_string).group(1)[1:-2])
                print('Ubicación del MPTR: ' + str(result5))
                result6 = re.search('E(.*)M', time_string).group(1)
                result6A = int(result6[:2])
                result6B = int(result6[3:5])
                result6C = int(result6[6:8])

                print('Número de errores E08: ' + str(result6A))
                print('Número de errores tipo overrun: ' + str(result6B))
                print('Número de paradas debido a baja tensión: ' + str(result6C))
                result7 = int(re.search('M(.*)B', time_string).group(1)[:4])
                print('Memoria total: ' + str(result7) + ' kB')
                result8 = float(re.search('B(.*)', time_string).group(1)[1:])
                print('Tensión de batería interna: ' + str(result8) + ' V')
                print('-------------------------')
                receiving_time = False

              except:
                print('No se pudo leer')
                receiving_time = False
                break
            else:
              line = line[1:]
          except:
            print('Datos con errores...')
            error_counter = error_counter + 1
            if error_counter >= 20:
              time_completed = False
              return 1
            line = line[1:]

  for c in ser.read():
    line.append(c)
  line = []
  return 0
```

4.5) Solicitud de fecha y hora del *datalogger*.
Esta función permite conocer el estado del *Real Time Clock* (RTC) del *datalogger*.

```
def time_request():
  ser.reset_input_buffer()
  ser.reset_output_buffer()
  time.sleep(0.1)

  print('Enviando solicitud de fecha y hora')
  error_counter = 0
  conectado = False
  receiving_time = False
  line = []
   while conectado == False:
```

```
11.
12.   send_order(ser,'C\r')
13.
14.   for c in ser.read():
15.     if c != 255:
16.       line.append(c)
17.     if len(line) == 2:
18.       if line == [67, 13]:
19.         print('Recibiendo la fecha y hora')
20.         conectado = True
21.         receiving_time = True
22.         line = []
23.       else:
24.         error_counter = error_counter + 1
25.         if error_counter == 100:
26.           return 1
27.         line = line[1:]
28.   error_counter = 0
29.   while receiving_time == True:
30.     for c in ser.read():
31.       line.append(c)
32.       if len(line) == 33:
33.         try:
34.           if 80+sum(line[:-9]) == int(bytes(line[-9:-5]).decode("utf-8")):
35.             try:
36.               time_string = bytes(line[2:-11]).decode("utf-8")
37.               print(time_string)
38.               receiving_time = False
39.             except:
40.               print('No se pudo leer')
41.               receiving_time = False
42.               break
43.           else:
44.             line = line[1:]
45.         except:
46.           print('Data with errors...')
47.           error_counter = error_counter + 1
48.           if error_counter >= 20:
49.             time_completed = False
50.             ser.flushInput()
51.             ser.flushOutput()
52.             return 1
53.           line = line[1:]
54. for c in ser.read():
55.   line.append(c)
56. line = []
57. return 0
```

4.6) Actualización de la fecha y hora del *datalogger*.
Esta función permite actualizar el *Real Time Clock* (RTC) del *datalogger*.

```
def time_update():
    ser.reset_input_buffer()
    ser.reset_output_buffer()
    time.sleep(0.05)

    line = []
    print('Actualizando el reloj del CR10X...')

    fmt = '%Y.%m.%d %H:%M:%S'
    current_time = time.strftime(fmt)
    julian_day = datetime.datetime.strptime(current_time, fmt).timetuple().tm_yday
    new_time = current_time[2:4]+':'+str(julian_day)+':'+current_time[11:]+'C'

    for element in new_time:
      ser.write(bytearray(element.encode()))
      time.sleep(0.02)
    ser.write(bytearray.fromhex("0d"))
    time.sleep(0.1)

    while True:
      for c in ser.read():
        line.append(c)
        if len(line) == 17:
          try:
            if new_time[:-1] == bytes(line[:-2]).decode("utf-8"):
              print('El reloj del CR10X fue actualizado.')
              print(new_time[:-1])
              return 0
          except:
              print('El reloj no fue actualizado.')
              return 1

    return 0
```

4.7) Bajada de programa residente en el *datalogger.*
Esta función permite obtener el programa cargado en la memoria del *datalogger*. El mismo no contendrá comentarios, puesto que es obtenido a partir de los comandos binarios residentes en la memoria de programa.

```
def retrieve_program():
    ser.reset_input_buffer()
    ser.reset_output_buffer()

    intentos = 0
    leyendo = False
    print('Recuperando programa...')

```

```
send_order(ser,'2718H\r')

line = []
while True:
  for c in ser.read():
    line.append(c)
  if len(line) == 11:
    if line == [50,55,49,56,72,13,10,13,10,62,17]:
      intentos = 0
      break
  intentos = intentos + 1
  if intentos == 100:
    print('Recuperación de programa falló en 2718H')
    return 1

ser.write(bytearray('*'.encode()))
time.sleep(0.1)
line = []
while True:
  for c in ser.read():
    line.append(c)
  if len(line) == 8:
    if line == [13,10,77,79,68,69,32,17]:
      intentos = 0
      break
  intentos = intentos + 1
  if intentos == 100:
    print('Recuperación de programa falló en *')
    return 1

ser.write(bytearray('D'.encode()))
time.sleep(0.1)
line = []
while True:
  for c in ser.read():
    line.append(c)
  if len(line) >= 12:
     intentos = 0
     break
  intentos = intentos + 1
  if intentos == 100:
    print('Recuperación de programa falló en D')
    return 1

ser.write(bytearray('1'.encode()))
time.sleep(0.1)
line = []
while True:
  for c in ser.read():
    line.append(c)
```

```
  if len(line) == 2:
    if line == [49,17]:
      intentos = 0
      break
  intentos = intentos + 1
  if intentos == 100:
    print('Program retrieval failed in 1')
    return 1

ser.write(bytearray('A'.encode()))
time.sleep(0.1)
line = []
while True:
  for c in ser.read():
    line.append(c)
  if len(line) == 8:
    if line == [13,10,48,49,58,125,13,10]:
      intentos = 0
      break
  intentos = intentos + 1
  if intentos == 100:
    print('Recuperación de programa falló en A')
    return 1

line = []
while True:
  for c in ser.read():
    line.append(c)

  if line[-2:] == [5,5]:
    print(bytes(line[:-2]).decode("utf-8"))

    fmt = '%Y.%m.%d %H:%M:%S'
    current_time = time.strftime(fmt)
    header_string = ';Programa recuperado del datalogger\r\n;'+current_time+'\r\
n;DAT-ON V1.0 - 2023 - SMN\r\n\r\n'
    newFile = open("programa_obtenido.txt", "wb")
    file_header = list(bytes(header_string, 'utf-8'))

    file_data = file_header + line[:-2]
    for byte in file_data:
      newFile.write(byte.to_bytes(1, byteorder='big'))
    newFile.close()

     break

 line = []
 while True:
   for c in ser.read():
     line.append(c)
```

```
    if line[-2:] == [62,17]:
      ser.write(bytearray('*'.encode()))
      time.sleep(0.1)
      ser.write(bytearray('0'.encode()))
      break

  for c in ser.read():
    line.append(c)
  line = []
  return 0
```

4.8) Subida de programa al *datalogger*.

Esta función permite cargar un programa en la memoria del *datalogger*. Todos los caracteres del archivo (aún los comentarios) serán enviados al *datalogger*, y será este el encargado de omitir los comentarios y compilar los comandos para almacenar el programa en formato binario en la memoria de programa. El programa debe nombrarse como "programa_a_subir.txt".

```
def send_program():
  ser.reset_input_buffer()
  ser.reset_output_buffer()

  intentos = 0
  leyendo = False
  print('Sending program...')

  send_order(ser,'2718H\r')

  line = []
  while True:
    for c in ser.read():
      line.append(c)
    if len(line) == 11:
      if line == [50,55,49,56,72,13,10,13,10,62,17]:
        intentos = 0
        break
    intentos = intentos + 1
    if intentos == 500:
      print('Recuperación de programa falló en 2718H')
      return 1

  ser.write(bytearray('*'.encode()))
  time.sleep(0.1)
  line = []
  while True:
    for c in ser.read():
      line.append(c)
    if len(line) == 8:
      if line == [13,10,77,79,68,69,32,17]:
        intentos = 0
        break
```

```
  intentos = intentos + 1
  if intentos == 100:
    print('Recuperación de programa falló en *')
    print(line)
    return 1

ser.write(bytearray('D'.encode()))
time.sleep(0.1)
line = []
while True:
  for c in ser.read():
    line.append(c)
  if len(line) >= 12:
     intentos = 0
     break
  intentos = intentos + 1
  if intentos == 200:
    print(line)
    print('Recuperación de programa falló en D')
    return 1

ser.write(bytearray('2'.encode()))
time.sleep(0.1)
line = []
while True:
  for c in ser.read():
    line.append(c)
  if len(line) == 2:
    if line == [50,17]:
      intentos = 0
      break
  intentos = intentos + 1
  if intentos == 200:
    print(line)
    print('Program retrieval failed in 2')
    return 1

ser.write(bytearray('A'.encode()))
time.sleep(0.1)
line = []
while True:
  for c in ser.read():
    line.append(c)
  if len(line) == 6:
    if line == [13,10,48,50,58,60]:
      intentos = 0
      break
  intentos = intentos + 1
  if intentos == 200:
    print(line)
```

```
    print('Recuperación de programa falló en A')
    return 1

#A partir de este punto debo enviar el programa
#Primero debo cargarlo desde el archivo
program_file = open("programa_a_subir.txt", "rb")
contenido = program_file.read()
print(contenido)
program_file.close()

S1, S0 = get_signature(contenido)

ser.write(contenido)
# Envío de firma para el buffer actual (página 255, C7, CR10X Operator's Manual)
ser.write(bytearray.fromhex("03"))
ser.write(bytearray.fromhex("03"))
time.sleep(0.1)

line = []
while True:
  for c in ser.read():
    line.append(c)

  if len(line) == 5:
    if line[:2] == [3,3] and line[4] == 60:
      if S1 == line[2] and S0 == line[3]:
        print('>> Firma verificada <<')
        intentos = 0
        break

time.sleep(0.1)
# Carga del buffer actual y reinicio de firma (página 255, C7, CR10X Operator's Manual)
ser.write(bytearray.fromhex("04"))
ser.write(bytearray.fromhex("04"))

line = []
while True:
  for c in ser.read():
    line.append(c)

  if len(line) == 3:
    if line == [4,4,60]:
      intentos = 0
      break

time.sleep(0.1)
# Carga el buffer actual, salida y compilación de programa.
ser.write(bytearray.fromhex("05"))
ser.write(bytearray.fromhex("05"))
```

```
132.
133. line = []
134. while True:
135.   for c in ser.read():
136.     line.append(c)
137.
138.   if len(line) == 2:
139.     if line[:2] == [5,5]:
140.       print('La unidad CR10X está compilando el programa...')
141.       break
142.
143. line = []
144. while True:
145.   for c in ser.read():
146.     line.append(c)
147.
148.   if line == [13,10,49,51,58,48,48,48,48,32,32,13,10,62,17]:
149.     print('Compilación exitosa..!')
150.     intentos = 0
151.     break
152.
153. ser.write(bytearray('*'.encode()))
154. time.sleep(0.1)
155. line = []
156. while True:
157.   for c in ser.read():
158.     line.append(c)
159.   if len(line) == 8:
160.     if line == [13,10,77,79,68,69,32,17]:
161.       intentos = 0
162.       break
163.
164. ser.write(bytearray('0'.encode()))
165. time.sleep(0.1)
166. line = []
167. while True:
168.   for c in ser.read():
169.     line.append(c)
170.   if line == [48,48,13,10,76,79,71,32,49,50,13,10,42]:
171.     print('El nuevo programa está corriendo exitosamente..!')
172.     intentos = 0
173.     break
174.
175. for c in ser.read():
176.   line.append(c)
177. line = []
178. return 0
```

4.9) Recuperación (completa) de mediciones almacenadas en la “memoria final”.
Esta función realiza una descarga completa de los datos almacenados en la *Final Storage* del *datalogger*. Modificando la instrucción “send_order(ser,'1G\r')” es posible realizar una descarga parcial (ej.: “send_order(ser,'1500G\r')” realiza una decarga a partir de la posición 1500 de la *Final Storage*.)

```
def retrieve_measurements():
    #ser.flushInput()
    #ser.flushOutput()
    ser.reset_input_buffer()
    ser.reset_output_buffer()

    intentos = 0
    leyendo = False
    print('Retrieving measurements...')

    send_order(ser,'1A\r')

    line = []
    while True:
      for c in ser.read():
        line.append(c)

      if line[-3:] == [13, 10, 42]:
        if sum(line[:-9]) == int(bytes(line[-9:-5]).decode("utf-8")):
          data_string = bytes(line[2:-11]).decode("utf-8")
          n_registros = int(re.search('F(.*)V', data_string).group(1)[1:-2])
          print('Filled Final Storage Locations: ' + str(n_registros))
          pages_1024 = int(n_registros/1024)
          rest_ = n_registros%1024
          break

      intentos = intentos + 1
      if intentos == 50000:
        print('Measurements retrieval failed in 1A')
        return 1

    send_order(ser,'2413J\r')

    line = []
    while True:
      for c in ser.read():
        line.append(c)
        #print(line)
      if len(line) == 8:
        intentos = 0
        break
      intentos = intentos + 1
      if intentos == 500:
        print('Measurements retrieval failed in 2413J')
        return 1

```

```
ser.write(bytearray.fromhex("00"))
time.sleep(0.02)
ser.write(bytearray.fromhex("40"))
time.sleep(0.02)
ser.write(bytearray.fromhex("00"))
time.sleep(0.02)
ser.write(bytearray.fromhex("00"))
time.sleep(0.1)

line = []
while True:
  for c in ser.read():
    line.append(c)
    #print(line)
  if len(line) == 4:
    intentos = 0
    break
  intentos = intentos + 1
  if intentos == 500:
    print('Measurements retrieval failed in 0 64 0 0')
    return 1

send_order(ser,'1G\r')

line = []
while True:
  for c in ser.read():
    line.append(c)
    #print(line)
  if len(line) == 22: #28
    intentos = 0
    break
  intentos = intentos + 1
  if intentos == 500:
    print('Measurements retrieval failed in 1G')
    return 1

send_order(ser,'K\r')

line = []
while True:
  for c in ser.read():
    line.append(c)
    #print(line)
  if len(line) == 13:
    intentos = 0
    break
  intentos = intentos + 1
  if intentos == 500:
    print('Measurements retrieval failed in K')
```

```
96.     return 1
97.
98.  stream_datos = []
99.  for i in range(pages_1024):
100.
101.   send_order(ser,'1024F\r')
102.
103.   line = []
104.   while len(line) < 2057: #número de palabras x 2 + 2 + 7
105.     for c in ser.read():
106.       line.append(c)
107.
108.   #Verify if signatures match
109.   S1, S0 = get_signature(line[7:-2])
110.   if S1 == line[-2] and S0 == line[-1]:
111.     print('Page '+str(i)+'/'+str(pages_1024)+' has been recovered OK...')
112.     stream_datos = stream_datos + line[7:-2]
113.   else:
114.     print('Page '+str(i)+' signature error..!')
115.     return 1
116.
117.  if rest_ > 0:
118.   rest_string = str(rest_)
119.   rest_string_len = len(rest_string)
120.   for i in range(rest_string_len):
121.     ser.write(bytearray(rest_string[i].encode()))
122.     time.sleep(0.02)
123.
124.   send_order(ser,'F\r')
125.
126.   line = []
127.   while len(line) < (rest_*2 + 5 + rest_string_len): #número de palabras x 2 +
      2 + 7
128.     for c in ser.read():
129.       line.append(c)
130.
131.   #Verify if signatures match
132.   S1, S0 = get_signature(line[3+rest_string_len:-2])
133.   if S1 == line[-2] and S0 == line[-1]:
134.     print('Page '+str(pages_1024)+'/'+str(pages_1024)+' has been recovered
      OK...')
135.     stream_datos = stream_datos + line[3+rest_string_len:-2]
136.   else:
137.     print('Page '+str(i)+' signature error..!')
138.     return 1
139.
140.
141.  print(bytes(line))
142.
143.  i = 0
144.  salto = 0
```

```
table_dump = ''
while True:
  i = i + salto
  if stream_datos[i] & 0b11111100 == 0b11111100:#Inicio detectado
    if i > 0:
      table_dump = table_dump + '\n'
    Array_ID = (stream_datos[i] & 0b11)*256 + stream_datos[i+1]
    Year = stream_datos[i+2]*256 + stream_datos[i+3]
    Day_of_Year = stream_datos[i+4]*256 + stream_datos[i+5]
    Minutes = stream_datos[i+6]*256 + stream_datos[i+7]

    #Modern format
    table_dump = table_dump + jdtodate (Year, Day_of_Year, Minutes)

    #Old format
    #table_dump = table_dump + str(Array_ID) + ',' + str(Year) + ',' +\
    #str(Day_of_Year) + ',' + str(Hour)

    salto = 8
    if i+salto >= len(stream_datos):
      break
    continue

  if stream_datos[i] & 0b11100 == 0b11100:
    if stream_datos[i] & 0b100000 == 0: #First byte of 4 byte value
      AxxxxxGH = stream_datos[i] & 0b10000011
      if AxxxxxGH == 0:
        multiplicador = 1
      if AxxxxxGH == 0b10000000:
        multiplicador = 0.1
      if AxxxxxGH == 1:
        multiplicador = 0.01
      if AxxxxxGH == 0b10000001:
        multiplicador = 0.001
      if AxxxxxGH == 0b10:
        multiplicador = 0.0001
      if AxxxxxGH == 0b10000010:
        multiplicador = 0.00001
      if stream_datos[i] & 0b1000000 == 0b1000000:
        signo = -1
      else:
        signo = 1
      valor = signo*multiplicador*((stream_datos[i+2]&0b1)*2**16 +
stream_datos[i+1]*256 + stream_datos[i+3])
      table_dump = table_dump + ',' + str(int(valor*1E5)/1E5)
      salto = 4
      if i+salto >= len(stream_datos):
        break
      continue

    if stream_datos[i] & 0b11100000 == 0b100000:
```

```
      print('Error while decoding bytes')
      return 1
      #print('Third byte of 4 byte value')
    if stream_datos[i] == 0b01111111:
      print('First byte of 2 byte -dummy- word')
      salto = 2 #The seccond byte is irrelevant
      if i+salto >= len(stream_datos):
        break
      continue

  else:
      xBCxxxxx = stream_datos[i] & 0b1100000
      if xBCxxxxx == 0:
        multiplicador = 1
      if xBCxxxxx == 0b0100000:
        multiplicador = 0.1
      if xBCxxxxx == 0b1000000:
        multiplicador = 0.01
      if xBCxxxxx == 0b1100000:
        multiplicador = 0.001
      if stream_datos[i] & 0b10000000 == 0b10000000:
        signo = -1
      else:
        signo = 1
      valor = signo*multiplicador*((stream_datos[i]&0b11111)*256 +
stream_datos[i+1])
      table_dump = table_dump + ',' + str(int(valor*1E5)/1E5)
      salto = 2 #The seccond byte is irrelevant
      if i+salto >= len(stream_datos):
        break
      continue
   salto = 1

print(table_dump)
fmt = '%Y_%m_%d_%H_%M_%S'
current_time = time.strftime(fmt)

file_name = 'CR10X_'+current_time+'.dat'
newFile = open(file_name, "w")
newFile.write(table_dump)
newFile.close()

send_order(ser,'2A')

for c in ser.read():
  line.append(c)
line = []
return 0
```

4.10) Blanqueado de *flags* de error.

Esta función permite borrar los *flags* de error (bits manipulados por el propio *datalogger* para indicar distintos eventos indeseados en el sistema, tales como un corte de energía o llenado de la memoria).

```
def clear_errors():
    #ser.flushInput()
    #ser.flushOutput()
    ser.reset_input_buffer()
    ser.reset_output_buffer()

    line = []
    intentos = 0
    print('Limpiando los flags de error...')

    while True:
      ser.write(bytearray('8'.encode()))
      time.sleep(0.02)
      ser.write(bytearray('8'.encode()))
      time.sleep(0.02)
      ser.write(bytearray('8'.encode()))
      time.sleep(0.02)
      ser.write(bytearray('8'.encode()))
      time.sleep(0.02)
      ser.write(bytearray('A'.encode()))
      time.sleep(0.02)
      ser.write(bytearray('\r'.encode()))
      time.sleep(0.02)

      for c in ser.read():
        line.append(c)
        if len(line) >= 6:
          if line == [56, 56, 56, 56, 65, 13]:
            print('Errores borrados!')
            return 0
          else:
            line = line[1:]
      intentos = intentos + 1
      if intentos >= 100:
         print('Los errores no pudieron ser borrados!')
         return 1
```

4.11) Funciones adicionales.

Para el correcto funcionamiento de las funciones definidas anteriormente es necesario contar con las siguientes funciones adicionales:

```
def get_signature(data_):
    S1 = 0xAA
    S0 = 0xAA
    for transmited_byte in data_:
      T1 = S1
```

```
    S1 = S0
    T2 = (S0*2 + int(S0/128)%2)%256
    S0 = (T2 + T1 + transmited_byte)%256
  return S1, S0

def send_order(serial_,order_):
  order_len = len(order_)
  for i in range(order_len):
    try:
      ser.write(bytearray(order_[i].encode()))
    except:
      print('Error al enviar datos al puerto serial.')
      return 1
    if i == order_len-1:
      time.sleep(0.1)
    else:
      time.sleep(0.02)
  return 0

def jdtodate (year, julian_day, time_):
  jdate = str(year)+str(julian_day)
  fmt = '%Y%j'
  datestd = datetime.datetime.strptime(jdate, fmt).date()

  month_string = ''
  if datestd.month < 10:
    month_string = '0'
  month_string = month_string + str(datestd.month)

  day_string = ''
  if datestd.day < 10:
    day_string = '0'
  day_string = day_string + str(datestd.day)

  hour_string = ''
  if int(time_/100) < 10:
    hour_string = '0'
  hour_string = hour_string + str(int(time_/100))

  minute_string = ''
  if time_%100 < 10:
    minute_string = '0'
  minute_string = minute_string + str(time_%100)

  date_string = '\"'+str(year)+'-'+month_string+'-'+day_string+' '+hour_string +\
  ':'+minute_string+':00\"'

  return(date_string)
```

5. CONCLUSIONES

El desarrollo realizado en el Servicio Meteorológico Nacional permite utilizar adquisidores de datos Campbell Scientific CR10X (y posiblemente CR21X, por sus similares características) con las ventajas de la programación de código abierto en lenguaje Python. La Interfaz Serial Multimodo (ISM) para *dataloggers* Campbell Scientific Inc., sumada al conjunto de funciones de conectividad implementadas en Python 3.X, otorgan una “segunda vida” a adquisidores de datos que anteriormente a este trabajo podrían haber sido considerados como obsoletos. Las instrucciones y código presentados en este documento permiten a cualquier agente con formación técnica mínima la integración de la ISM y el uso del *datalogger* CR10X con Windows/Linux sin necesidad de contar con *software/hardware* comercial (LoggerNet/SC32A).

6. REFERENCIAS

1986-2003 CAMPBELL SCIENTIFIC, INC.: CR10X Measurement and Control Module Operator's Manual. REVISION: 2/03
(https://s.campbellsci.com/documents/au/manuals/cr10x.pdf)

Arduino® UNO R3: Product Reference Manual SKU: A000066
(https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf).