Unknown Title

RobTillaart ⋮

RobTillaart/**SGP30**

Arduino library for SGP30 environment sensor

2 Contributors | 0 Issues | 7 Stars | 3 Forks

SGP30

Biblioteca Arduino para sensor ambiental SGP30

Advertencia: experimental, la biblioteca aún no está funcional completa.

Descripción

El SGP30 de Sensirion es un sensor ambiental que mide H2 y Etanol en el aire. A partir de estos números, un algoritmo interno en el sensor deriva un equivalente de CO2 y una medición de TVOC. La biblioteca tiene una conversión experimental para H2 y Etanol.

Las unidades de CO2 son ppm, las unidades de TVOC son ppb. Las unidades para H2 y Etanol son ppm. Tenga en cuenta que para concentraciones más grandes, la resolución de las mediciones disminuye, consulte la hoja de datos.

La biblioteca admite 2 tipos de interfaces, una síncrona y una asíncrona. La interfaz de sincronización se bloquea durante un máximo de 40 milisegundos, lo que fue suficiente para activar la implementación de una interfaz asíncrona.

Nota: la interfaz de sincronización se implementa con la interfaz asíncrona.

Nota: las versiones anteriores a la 0.2.0 están obsoletas debido a un error en **setBaseline()** .

Frecuencia de muestreo

Los valores de CO2 y TVOC se pueden leer hasta una vez por segundo (1 Hz). Etanol y H2, los datos sin procesar se pueden muestrear hasta 40 Hz.

Los primeros 15 segundos el sensor necesita estabilizarse. A partir de entonces se obtienen datos reales.

rendimiento I2C

El SGP30 funciona con bus I2C a 100 KHz y 400 KHz. En una breve prueba funcionó bien hasta 500 KHz. Un reloj I2C más rápido no le da a la interfaz de sincronización mucha ganancia (relativa), sin embargo, para la interfaz asíncrona, la ganancia relativa es mucho mayor.

(ejecución de prueba indicativa con UNO - IDE 1.8.13, solo CO2 y TVOC, tiempos en micros)

velocidad I2C	medida()	pedido()	leer()
100kHz	12360	336	732
200kHz	12212	196	408
300kHz	12168	144	300
400kHz	12140	132	264
500kHz	12128	124	236

Tenga en cuenta que el bloqueo de la medida () toma de 11 a 12 milisegundos adicionales.

Múltiples sensores.

El sensor SGP30 tiene una dirección I2C fija 0x58, por lo que solo se puede usar un sensor por bus I2C. Si necesita más, debe usar un multiplexor I2C o una MCU con múltiples buses I2C o cambiar el VCC como una especie de señal ChipSelect.

- https://github.com/RobTillaart/TCA9548 (multiplexor I2C de 8 canales)

Interfaz

```
#include "SGP30.h"
```

Constructor

- **SGP30(TwoWire *wire = &Wire)** Constructor con la interfaz Wire opcional como parámetro.
- **bool begin()** inicia el bus I2C y devuelve verdadero si la dirección del dispositivo 0x58 está visible en el bus I2C.
- **bool begin(uint8_t sda, uint8_t scl)** idem, para el ESP32 donde uno puede elegir los pines I2C.
- **bool isConnected()** comprueba si la dirección 0x58 está visible en el bus I2C.

- **void GenericReset()** ¡ADVERTENCIA reinicia todos los dispositivos I2C en el bus que admiten esta llamada!

Meta

- **bool getID()** lee el ID del sensor en 12 bytes. (necesita reelaboración).
- **uint16_t getFeatureSet()** devuelve 0x0022, indica que los comandos utilizados en esta biblioteca son compatibles.
- **bool MeasureTest()** verifica que el chip esté funcionando.

Mediciones síncronas

- **uint32_t lastMeasurement()** marca de tiempo en milisegundos de la última medición de sincronización realizada. Esta función de conveniencia es útil para evitar leer el sensor con demasiada frecuencia.
- **bool measure(bool all = false)**si all == false, solo se actualizan los valores de TVOC y CO2 (lento debido al bloqueo), si all == true, también se actualizan los valores de H2 y Etanol (aún más lento). Tenga en cuenta que la medición es lenta ya que hay un bloqueo activo hasta que se realiza el sensor. Si la última medición es hace menos de un segundo, no se realiza ninguna medición y la función devuelve falso.

Mediciones sincrónicas A

Con la interfaz asíncrona, el usuario debe controlar que las lecturas estén separadas por al menos un segundo. El usuario también debe tener cuidado de no mezclar diferentes solicitudes. Ver ejemplos.

- **void request()** envía una solicitud al sensor para leer CO2 y TVOC.
- **bool read()** devuelve verdadero si la última solicitud es hace más de 12 milisegundos, se leen y actualizan el CO2 y el TVOC. De lo contrario, se devuelve falso.
- **void requestRaw()** envía una solicitud al sensor para leer H2 y Etanol.
- **bool readRaw()** devuelve verdadero si la última solicitud es hace más de 25 milisegundos, el H2 y el etanol se leen y actualizan. De lo contrario, se devuelve falso.

Obtener los datos

La biblioteca almacena en caché los últimos valores leídos y estas son las funciones para acceder a ellos.

- **uint16_t getTVOC()** obtiene la concentración de TVOC (ppb)
- **uint16_t getCO2()** obtiene la concentración **equivalente** de CO2 (ppm)
- **uint16_t getH2_raw()** obtiene el H2 sin procesar. Unidades desconocidas.
- **uint16_t getEthanol_raw()** obtiene el etanol sin procesar. Unidades desconocidas.

Calibración

Consulte la hoja de datos para conocer el rango operativo, figura 7.

- **float setRelHumidity(float T, float RH)** establece la compensación de temperatura (5-55°C) y humedad **relativa** (10-95%). Estos valores se pueden obtener, por ejemplo, de un sensor SHT30, DHT22 o similar. La función devuelve la humedad absoluta.
- **void setAbsHumidity(float absoluteHumidity)** establece la compensación de la humedad **absoluta** . La concentración está en gramos por metro cúbico (g/m3)

Funciones de línea de base

Las funciones de línea de base dan al sensor un valor de referencia. Después de correr en una condición conocida, por ejemplo, al aire libre, se pueden obtener los valores de referencia como una especie de calibración. Lea atentamente la hoja de datos antes de utilizar estas funciones.

Nota: si el sensor no tiene lecturas hechas, estos valores tienden a ir a cero. Esto se debe a que las líneas base se basan en lecturas recientes.

- **bool getBaseline(uint16_t *CO2, uint16_t *TVOC)** recupera los valores de referencia del sensor.
- **void setBaseline(uint16_t CO2, uint16_t TVOC)** establece los valores de referencia.

Para obtener resultados más rápidos y precisos para el TVOC en malas condiciones de aire, lea **la Línea de base inceptiva para las mediciones de TVOC** (no probado)

- **bool getTVOCBaseline(uint16_t *TVOC)** recupera el valor inicial de TVOC del sensor.
- **void setTVOCBaseline(uint16_t TVOC)** establece el valor inicial de TVOC.

Misceláneas

- **int lastError()** devuelve el último error. (necesita reelaboración)

Etanol experimental H2

usar bajo su propio riesgo.

Desde la versión 0.1.2, la biblioteca cuenta con soporte experimental para la concentración de H2 y etanol en ppm.

Se deben usar estas funciones más como una indicación relativa que como una medida absoluta, ya que definitivamente no está calibrada. Las ejecuciones con diferente temperatura y humedad (diferentes días) dan valores muy diferentes, incluido el desbordamiento y el infinito.

- **float getH2()** obtiene la concentración de H2. Unidades ppm.
- **float getEthanol()** obtiene la concentración de etanol. Unidades ppm.

Las referencias utilizadas se basan en
(1) el promedio de datos sin procesar en el aire exterior a 22 °C a 1 metro y (2) la suposición de que esto es 0,4 resp. 0,5 ppm. (Tenga en cuenta solo 1 dígito significativo) como se menciona en la hoja de datos P2.

- **void setSrefH2(uint16_t s = 13119)** // 13119 es mi medida.
- **uint16_t getSrefH2()** devuelve el conjunto de valores.

- **void setSrefEthanol(uint16_t s = 18472)** // 18472 es mi medida.
- **uint16_t getSrefEthanol()** devuelve el conjunto de valores.

Operacional

Ver ejemplos

Enlaces

https://www.adafruit.com/product/3709 - el sensor.

Futuro

Debe

- mejorar la documentación

Debería

- prueba
    - tableros diferentes
    - diferentes gases / atmósfera si es posible.

Podría

- rehacer **getID()**
- hacer definiciones para los números mágicos (comandos)
- mover código de .h a .cpp
- mejorar/combinar la función **de comando privado ()**
- agregar/ampliar el manejo de errores
- mejor nombre para **medirPrueba()**

La verificación de CRC + manejo de errores (desde 0.1.4) agrega alrededor de 330 bytes PROGMEM en una UNO. Es posible que se necesite una clase mínima que solo lea CO2 y TVOC, sin líneas de base, etc. para las plataformas más pequeñas.

No