

```

1  /*
2  * This is the .cpp file for the ccs821 VOC sensor
3  * The library for this sensor was retrieved on line:
4  * https://learn.adafruit.com/adafruit-ccs811-air-quality-sensor/arduino-wiring-test
5  * MECH 45X Team 26 did not write Part 1, the on line library
6  *
7  * Therefore Part 1 is not properly commented because the
8  * the team does not understand the code.
9  *
10 * Part 2 was written by Team 26 and is properly commented.
11 *
12 * Part 1 begins...
13 */
14
15 #include "CCS821.h"
16
17 /*****
18  *!
19   @brief  Setups the I2C interface and hardware and checks for communication.
20   @param  addr Optional I2C address the sensor can be found on. Default is 0x5A
21   @returns True if device is set up, false on any failure
22  */
23 /*****
24 bool Adafruit_CCS811::begin(uint8_t addr)
25 {
26     _i2caddr = addr;
27
28     _i2c_init();
29
30     SWReset();
31     delay(100);
32
33     //check that the HW id is correct
34     if(this->read8(CCS811_HW_ID) != CCS811_HW_ID_CODE)
35         return false;
36
37     //try to start the app
38     this->write(CCS811_BOOTLOADER_APP_START, NULL, 0);
39     delay(100);
40
41     //make sure there are no errors and we have entered application mode
42     if(checkError()) return false;
43     if(!_status.FW_MODE) return false;
44
45     disableInterrupt();
46
47     //default to read every second
48     setDriveMode(CCS811_DRIVE_MODE_1SEC);
49
50     return true;
51 }
52
53 /*****
54  *!
55   @brief  sample rate of the sensor.
56   @param  mode one of CCS811_DRIVE_MODE_IDLE, CCS811_DRIVE_MODE_1SEC,
57           CCS811_DRIVE_MODE_10SEC, CCS811_DRIVE_MODE_60SEC, CCS811_DRIVE_MODE_250MS.
58  */
59 void Adafruit_CCS811::setDriveMode(uint8_t mode)
60 {
61     _meas_mode.DRIVE_MODE = mode;
62     this->write8(CCS811_MEAS_MODE, _meas_mode.get());
63 }
64
65 /*****
66  *!
67   @brief  enable the data ready interrupt pin on the device.
68  */
69 /*****

```

```

69 void Adafruit_CCS811::enableInterrupt()
70 {
71     _meas_mode.INT_DATARDY = 1;
72     this->write8(CCS811_MEAS_MODE, _meas_mode.get());
73 }
74
75 /*****
76  *!
77  * @brief  disable the data ready interrupt pin on the device
78  */
79 /*****/
80 void Adafruit_CCS811::disableInterrupt()
81 {
82     _meas_mode.INT_DATARDY = 0;
83     this->write8(CCS811_MEAS_MODE, _meas_mode.get());
84 }
85
86 /*****
87  *!
88  * @brief  checks if data is available to be read.
89  * @returns True if data is ready, false otherwise.
90  */
91 /*****/
92 bool Adafruit_CCS811::available()
93 {
94     _status.set(read8(CCS811_STATUS));
95     if(!_status.DATA_READY)
96         return false;
97     else return true;
98 }
99
100 /*****
101  *!
102  * @brief  read and store the sensor data. This data can be accessed with getTVOC()
103  *          and geteCO2()
104  * @returns 0 if no error, error code otherwise.
105  */
106 /*****/
107 uint8_t Adafruit_CCS811::readData()
108 {
109     if(!available())
110         return false;
111     else{
112         uint8_t buf[8];
113         this->read(CCS811_ALG_RESULT_DATA, buf, 8);
114
115         _eCO2 = ((uint16_t)buf[0] << 8) | ((uint16_t)buf[1]);
116         _TVOC = ((uint16_t)buf[2] << 8) | ((uint16_t)buf[3]);
117
118         if(_status.ERROR)
119             return buf[5];
120
121         else return 0;
122     }
123 }
124
125 /*****
126  *!
127  * @brief  set the humidity and temperature compensation for the sensor.
128  * @param  humidity the humidity data as a percentage. For 55% humidity, pass in
129  *          integer 55.
130  * @param  temperature the temperature in degrees C as a decimal number. For 25.5
131  *          degrees C, pass in 25.5
132  */
133 /*****/
134 void Adafruit_CCS811::setEnvironmentalData(uint8_t humidity, double temperature)
135 {
136     /* Humidity is stored as an unsigned 16 bits in 1/512%RH. The
137     default value is 50% = 0x64, 0x00. As an example 48.5%

```

```

135 humidity would be 0x61, 0x00.*/
136
137 /* Temperature is stored as an unsigned 16 bits integer in 1/512
138 degrees; there is an offset: 0 maps to -25°C. The default value is
139 25°C = 0x64, 0x00. As an example 23.5% temperature would be
140 0x61, 0x00.
141 The internal algorithm uses these values (or default values if
142 not set by the application) to compensate for changes in
143 relative humidity and ambient temperature.*/
144
145 uint8_t hum_perc = humidity << 1;
146
147 float fractional = modf(temperature, &temperature);
148 uint16_t temp_high = (((uint16_t)temperature + 25) << 9);
149 uint16_t temp_low = ((uint16_t)(fractional / 0.001953125) & 0x1FF);
150
151 uint16_t temp_conv = (temp_high | temp_low);
152
153 uint8_t buf[] = {hum_perc, 0x00,
154                 (uint8_t)((temp_conv >> 8) & 0xFF), (uint8_t)(temp_conv & 0xFF)};
155
156 this->write(CCS811_ENV_DATA, buf, 4);
157
158 }
159
160 /*****
161  *!
162  * @brief calculate the temperature using the onboard NTC resistor.
163  * @returns temperature as a double.
164  */
165 /*****
166  *
167  */
168 double Adafruit_CCS811::calculateTemperature()
169 {
170     uint8_t buf[4];
171     this->read(CCS811_NTC, buf, 4);
172
173     uint32_t vref = ((uint32_t)buf[0] << 8) | buf[1];
174     uint32_t vntc = ((uint32_t)buf[2] << 8) | buf[3];
175
176     //from ams ccs811 app note
177     uint32_t rntc = vntc * CCS811_REF_RESISTOR / vref;
178
179     double ntc_temp;
180     ntc_temp = log((double)rntc / CCS811_REF_RESISTOR); // 1
181     ntc_temp /= 3380; // 2
182     ntc_temp += 1.0 / (25 + 273.15); // 3
183     ntc_temp = 1.0 / ntc_temp; // 4
184     ntc_temp -= 273.15; // 5
185     return ntc_temp - _tempOffset;
186 }
187
188 /*****
189  *!
190  * @brief set interrupt thresholds
191  * @param low_med the level below which an interrupt will be triggered.
192  * @param med_high the level above which the interrupt will ge triggered.
193  * @param hysteresis optional hysteresis level. Defaults to 50
194  */
195 /*****
196  *
197  */
198 void Adafruit_CCS811::setThresholds(uint16_t low_med, uint16_t med_high, uint8_t
hysteresis)
199 {
200     uint8_t buf[] = {(uint8_t)((low_med >> 8) & 0xF), (uint8_t)(low_med & 0xF),
201                     (uint8_t)((med_high >> 8) & 0xF), (uint8_t)(med_high & 0xF), hysteresis};
202
203     this->write(CCS811_THRESHOLDS, buf, 5);
204 }

```

```

203 /*****
204  /*!
205   @brief  trigger a software reset of the device
206  */
207  *****/
208 void Adafruit_CCS811::SWReset()
209 {
210     //reset sequence from the datasheet
211     uint8_t seq[] = {0x11, 0xE5, 0x72, 0x8A};
212     this->write(CCS811_SW_RESET, seq, 4);
213 }
214
215 /*****
216  /*!
217   @brief  read the status register and store any errors.
218   @returns the error bits from the status register of the device.
219  */
220  *****/
221 bool Adafruit_CCS811::checkError()
222 {
223     _status.set(read8(CCS811_STATUS));
224     return _status.ERROR;
225 }
226
227 /*****
228  /*!
229   @brief  write one byte of data to the specified register
230   @param  reg the register to write to
231   @param  value the value to write
232  */
233  *****/
234 void Adafruit_CCS811::write8(byte reg, byte value)
235 {
236     this->write(reg, &value, 1);
237 }
238
239 /*****
240  /*!
241   @brief  read one byte of data from the specified register
242   @param  reg the register to read
243   @returns one byte of register data
244  */
245  *****/
246 uint8_t Adafruit_CCS811::read8(byte reg)
247 {
248     uint8_t ret;
249     this->read(reg, &ret, 1);
250
251     return ret;
252 }
253
254 void Adafruit_CCS811::_i2c_init()
255 {
256     Wire.begin();
257 }
258
259 void Adafruit_CCS811::read(uint8_t reg, uint8_t *buf, uint8_t num)
260 {
261     uint8_t value;
262     uint8_t pos = 0;
263
264     //on arduino we need to read in 32 byte chunks
265     while(pos < num){
266
267         uint8_t read_now = min((uint8_t)32, (uint8_t)(num - pos));
268         Wire.beginTransaction((uint8_t)_i2caddr);
269         Wire.write((uint8_t)reg + pos);
270         Wire.endTransmission();
271         Wire.requestFrom((uint8_t)_i2caddr, read_now);

```

```

272
273     for(int i=0; i<read_now; i++){
274         buf[pos] = Wire.read();
275         pos++;
276     }
277 }
278 }
279
280 void Adafruit_CCS811::write(uint8_t reg, uint8_t *buf, uint8_t num)
281 {
282     Wire.beginTransaction((uint8_t)_i2caddr);
283     Wire.write((uint8_t)reg);
284     Wire.write((uint8_t *)buf, num);
285     Wire.endTransmission();
286 }
287
288 /*
289  * Part 2: code written by team 26
290  * This code was written by Team 26
291  * This code is properly commented
292  */
293
294 bool Adafruit_CCS811::start_voc(void) {
295     /*
296      * Start voc sensor using the library's begin() function
297      * If sensor is started, calibrate temperature
298      */
299     Serial.println("Trying to start VOC Sensor...");
300     if(!begin()){
301         Serial.println("Failed to start CC2821 VOC sensor! Wiring is likely incorrect.");
302         return false;
303     }
304     else {
305         Serial.println("Successfully started VOC Sensor!");
306         return true;
307     }
308 }
309
310 void Adafruit_CCS811::run_voc(void) {
311     /*
312      * Run the VOC sensor
313      * Take measurements until enough measurements have been taken to calculate the
314      * average
315      * use read_voc() to read from sensor
316      */
317     is_average_taken = false;
318     read_count = 1;
319     while(is_average_taken == false) {read_voc();}
320 }
321
322 void Adafruit_CCS811::read_voc(void) {
323     /*
324      * Read values from voc sensor
325      * IF data is read and max read count has not been exceed
326      * THEN fill_buffer and print_readings and read_count ++
327      * calculate_average_reading
328      * print_average_reading
329      */
330     if(available()){
331         float temp = calculateTemperature();
332         if(!readData() && read_count <= MAX_READ_COUNT){
333             fill_buffer();
334             print_readings();
335             read_count += 1;
336         } else{Serial.println("ERROR!");}
337     }
338     calculate_average_reading();
339     print_average_reading();
340 }

```

```

340
341 void Adafruit_CCS811::fill_buffer(void) {
342     /*
343      * add new values to buffers
344      */
345     eCO2_buf[read_count-1] = geteCO2();
346     TVOC_buf[read_count-1] = getTVOC();
347 }
348
349 void Adafruit_CCS811::print_readings(void) {
350     /*
351      * Print readings
352      */
353     Serial.print("Reading #:");
354     Serial.print(read_count);
355     Serial.print(", CO2: ");
356     Serial.print(geteCO2());
357     Serial.print("ppm, TVOC: ");
358     Serial.print(getTVOC());
359     Serial.println("ppb");
360 }
361
362 void Adafruit_CCS811::calculate_average_reading(void) {
363     /*
364      * Calculate the average reading if enough readings have been taken
365      */
366     if(read_count > MAX_READ_COUNT) {
367         eCO2_ave = 0;
368         TVOC_ave = 0;
369         for(int k = 0; k < MAX_READ_COUNT; k++) {
370             eCO2_ave += eCO2_buf[k];
371             TVOC_ave += TVOC_buf[k];
372         }
373         eCO2_ave = eCO2_ave / MAX_READ_COUNT;
374         TVOC_ave = TVOC_ave / MAX_READ_COUNT;
375
376         read_count = 1;
377         is_average_taken = true;
378     }
379 }
380
381 void Adafruit_CCS811::print_average_reading(void) {
382     /*
383      * print average reading values
384      */
385     if(is_average_taken) {
386         Serial.println("-----");
387         Serial.println("VOC Sensor Average Readings:");
388         Serial.println("-----");
389         Serial.print("CCS eCO2 Average: ");
390         Serial.println(eCO2_ave);
391         Serial.print("CCS TVOC Average: ");
392         Serial.println(TVOC_ave);
393     }
394 }
395
396 // Getter functions for VOC parameters
397 float Adafruit_CCS811::get_eCO2_ave(void) {return eCO2_ave;}
398 float Adafruit_CCS811::get_TVOC_ave(void) {return TVOC_ave;}
399

```