```cpp
/*
 * This is the .cpp file for the ccs821 VOC sensor
 * The library for this sensor was retrieved on line:
 * https://learn.adafruit.com/adafruit-ccs811-air-quality-sensor/arduino-wiring-test
 * MECH 45X Team 26 did not write Part 1, the on line library
 *
 * Therefore Part 1 is not properly commented because the
 * the team does not understand the code.
 *
 * Part 2 was written by Team 26 and is properly commented.
 *
 * Part 1 begins...
 */

 #include "CCS821.h"

/**************************************************************************/
/*!
    @brief  Setups the I2C interface and hardware and checks for communication.
    @param  addr Optional I2C address the sensor can be found on. Default is 0x5A
    @returns True if device is set up, false on any failure
*/
/**************************************************************************/
bool Adafruit_CCS811::begin(uint8_t addr)
{
  _i2caddr = addr;

  _i2c_init();

  SWReset();
  delay(100);

  //check that the HW id is correct
  if(this->read8(CCS811_HW_ID) != CCS811_HW_ID_CODE)
    return false;

  //try to start the app
  this->write(CCS811_BOOTLOADER_APP_START, NULL, 0);
  delay(100);

  //make sure there are no errors and we have entered application mode
  if(checkError()) return false;
  if(!_status.FW_MODE) return false;

  disableInterrupt();

  //default to read every second
  setDriveMode(CCS811_DRIVE_MODE_1SEC);

  return true;
}

/**************************************************************************/
/*!
    @brief  sample rate of the sensor.
    @param  mode one of CCS811_DRIVE_MODE_IDLE, CCS811_DRIVE_MODE_1SEC,
    CCS811_DRIVE_MODE_10SEC, CCS811_DRIVE_MODE_60SEC, CCS811_DRIVE_MODE_250MS.
*/
void Adafruit_CCS811::setDriveMode(uint8_t mode)
{
  _meas_mode.DRIVE_MODE = mode;
  this->write8(CCS811_MEAS_MODE, _meas_mode.get());
}

/**************************************************************************/
/*!
    @brief  enable the data ready interrupt pin on the device.
*/
/**************************************************************************/
```

```
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()
        and geteCO2()
103     @returns 0 if no error, error code otherwise.
104 */
105 /**************************************************************************/
106 uint8_t Adafruit_CCS811::readData()
107 {
108   if(!available())
109     return false;
110   else{
111     uint8_t buf[8];
112     this->read(CCS811_ALG_RESULT_DATA, buf, 8);
113
114     _eCO2 = ((uint16_t)buf[0] << 8) | ((uint16_t)buf[1]);
115     _TVOC = ((uint16_t)buf[2] << 8) | ((uint16_t)buf[3]);
116
117     if(_status.ERROR)
118       return buf[5];
119
120     else return 0;
121   }
122 }
123
124 /**************************************************************************/
125 /*!
126     @brief  set the humidity and temperature compensation for the sensor.
127     @param humidity the humidity data as a percentage. For 55% humidity, pass in
        integer 55.
128     @param temperature the temperature in degrees C as a decimal number. For 25.5
        degrees C, pass in 25.5
129 */
130 /**************************************************************************/
131 void Adafruit_CCS811::setEnvironmentalData(uint8_t humidity, double temperature)
132 {
133   /* Humidity is stored as an unsigned 16 bits in 1/512%RH. The
134   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   double Adafruit_CCS811::calculateTemperature()
167   {
168     uint8_t buf[4];
169     this->read(CCS811_NTC, buf, 4);
170
171     uint32_t vref = ((uint32_t)buf[0] << 8) | buf[1];
172     uint32_t vntc = ((uint32_t)buf[2] << 8) | buf[3];
173
174     //from ams ccs811 app note
175     uint32_t rntc = vntc * CCS811_REF_RESISTOR / vref;
176
177     double ntc_temp;
178     ntc_temp = log((double)rntc / CCS811_REF_RESISTOR); // 1
179     ntc_temp /= 3380; // 2
180     ntc_temp += 1.0 / (25 + 273.15); // 3
181     ntc_temp = 1.0 / ntc_temp; // 4
182     ntc_temp -= 273.15; // 5
183     return ntc_temp - _tempOffset;
184
185   }
186
187   /**************************************************************************/
188   /*!
189       @brief  set interrupt thresholds
190       @param low_med the level below which an interrupt will be triggered.
191       @param med_high the level above which the interrupt will ge triggered.
192       @param hysteresis optional histeresis level. Defaults to 50
193   */
194   /**************************************************************************/
195   void Adafruit_CCS811::setThresholds(uint16_t low_med, uint16_t med_high, uint8_t
      hysteresis)
196   {
197     uint8_t buf[] = {(uint8_t)((low_med >> 8) & 0xF), (uint8_t)(low_med & 0xF),
198     (uint8_t)((med_high >> 8) & 0xF), (uint8_t)(med_high & 0xF), hysteresis};
199
200     this->write(CCS811_THRESHOLDS, buf, 5);
201   }
202
```

```
/**************************************************************************/
/*!
    @brief  trigger a software reset of the device
*/
/**************************************************************************/
void Adafruit_CCS811::SWReset()
{
  //reset sequence from the datasheet
  uint8_t seq[] = {0x11, 0xE5, 0x72, 0x8A};
  this->write(CCS811_SW_RESET, seq, 4);
}

/**************************************************************************/
/*!
    @brief   read the status register and store any errors.
    @returns the error bits from the status register of the device.
*/
/**************************************************************************/
bool Adafruit_CCS811::checkError()
{
  _status.set(read8(CCS811_STATUS));
  return _status.ERROR;
}

/**************************************************************************/
/*!
    @brief  write one byte of data to the specified register
    @param  reg the register to write to
    @param  value the value to write
*/
/**************************************************************************/
void Adafruit_CCS811::write8(byte reg, byte value)
{
  this->write(reg, &value, 1);
}

/**************************************************************************/
/*!
    @brief  read one byte of data from the specified register
    @param  reg the register to read
    @returns one byte of register data
*/
/**************************************************************************/
uint8_t Adafruit_CCS811::read8(byte reg)
{
  uint8_t ret;
  this->read(reg, &ret, 1);

  return ret;
}

void Adafruit_CCS811::_i2c_init()
{
  Wire.begin();
}

void Adafruit_CCS811::read(uint8_t reg, uint8_t *buf, uint8_t num)
{
  uint8_t value;
  uint8_t pos = 0;

  //on arduino we need to read in 32 byte chunks
  while(pos < num){

    uint8_t read_now = min((uint8_t)32, (uint8_t)(num - pos));
    Wire.beginTransmission((uint8_t)_i2caddr);
    Wire.write((uint8_t)reg + pos);
    Wire.endTransmission();
    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.beginTransmission((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           delay(5000);
307           return true;
308       }
309   }
310
311   bool Adafruit_CCS811::run_voc(void) {
312       /*
313        * Run the VOC sensor
314        * Take measurements until enough measurements have been taken to calculate the
           average
315        * use read_voc() to read from sensor
316        */
317       is_average_taken = false;
318       read_count = 1;
319       error_count = 0;
320       while(is_average_taken == false  && error_count < MAX_ERROR_COUNT) {read_voc();}
321
322       if(is_average_taken) {return true;}
323       else if(error_count >= MAX_ERROR_COUNT) {return false;}
324   }
325
326   void Adafruit_CCS811::read_voc(void) {
327       /*
328        * Read values from voc sensor
329        * IF data is read and max read count has not been exceed
330        * THEN fill_buffer and print_readings and read_count ++
331        * calculate_average_reading
332        * print_average_reading
333        */
334       if(available()){
335           float temp = calculateTemperature();
336           if(!readData() && read_count <= MAX_READ_COUNT){
337               fill_buffer();
338               print_readings();
339               read_count += 1;
```

```
            error_count = 0;
        }
        else {
          error_count ++;
          Serial.print("ERROR #");
          Serial.println(error_count);
          delay(500);
        }
    }
    calculate_average_reading();
    print_average_reading();
}

void Adafruit_CCS811::fill_buffer(void) {
    /*
     * add new values to buffers
     */
    eCO2_buf[read_count-1] = geteCO2();
    TVOC_buf[read_count-1] = getTVOC();
}

void Adafruit_CCS811::print_readings(void) {
    /*
     * Print readings
     */
    Serial.print("VOC Reading #:");
    Serial.print(read_count);
    Serial.print(", CO2: ");
    Serial.print(geteCO2());
    Serial.print("ppm, TVOC: ");
    Serial.print(getTVOC());
    Serial.println("ppb");
}

void Adafruit_CCS811::calculate_average_reading(void) {
    /*
     * Calculate the average reading if enough readings have been taken
     */
    if(read_count > MAX_READ_COUNT) {
        eCO2_ave = 0;
        TVOC_ave = 0;
        for(int k = 0; k < MAX_READ_COUNT; k++) {
            eCO2_ave += eCO2_buf[k];
            TVOC_ave += TVOC_buf[k];
        }
        eCO2_ave = eCO2_ave / MAX_READ_COUNT;
        TVOC_ave = TVOC_ave / MAX_READ_COUNT;

        read_count = 1;
        is_average_taken = true;
    }
}

void Adafruit_CCS811::print_average_reading(void) {
    /*
    * print average reading values
    */
    if(is_average_taken) {
        Serial.println("-----------------------");
        Serial.println("VOC Sensor Average Readings:");
        Serial.println("-----------------------");
        Serial.print("CCS eCO2 Average: ");
        Serial.println(eCO2_ave);
        Serial.print("CCS TVOC Average: ");
        Serial.println(TVOC_ave);
    }
}

// Getter functions for VOC parameters
```

```
409  float Adafruit_CCS811::get_eCO2_ave(void) {
410      return eCO2_ave;
411  }
412  float Adafruit_CCS811::get_TVOC_ave(void) {
413      return TVOC_ave;
414  }
415
```