```cpp
/*
 * This is the .cpp file for the MH-Z19 CO2 Sensor
 * This code was exclusively written by MECH 45X Team 26
 */

#include "MHZ19.h"
#include "Time.h"

MHZ19::MHZ19() {
    first_time = true;
}

MHZ19::~MHZ19() {
}

void MHZ19::set_transistor(int pin) {
    /*
     * Set pin mode of co2 transistor pin
     */
    co2_transistor_control = pin;
    pinMode(co2_transistor_control,OUTPUT);
}

void MHZ19::begin_timer(void) {
    /*
     * Function to start timer
     * Only called if timer has not been started
     * Turns CO2 sensor on and starts timer
     * Prints start time
     */
    co2_ppm_average_uncalibrated = -1;
    co2_ppm_average_calibrated = -1;
    digitalWrite(co2_transistor_control, HIGH);
    start_time = now();
    Serial.println("--------------");
    Serial.print("CO2 start time: ");
    Serial.println(start_time);
    Serial.println("--------------");
    first_time = false;
}

bool MHZ19::check_begin_reading(void) {
    /*
     * Check if sensor has been on long enough to start reading
     * Print how long the sensor has been on
     * Return true if sensor is on long enough
     * else return false
     */
    current_time = now();
    duration = current_time - start_time;
    Serial.println("-----------------");
    Serial.print("CO2 Duration: ");
    Serial.println(duration);
    Serial.println("-----------------");

    if(duration >= CO2_START_UP_TIME) {
        Serial.println("Three minutes have elapsed since starting CO2 sensor!");
        return(true);
    } else{return(false);}
}

bool MHZ19::make_sensor_read(void) {
    /*
     * Method to make CO2 sensor read
     * Sensor turns off every time enough readings have been taken
     * Sensor turns back on again to take more readings
     *
     * IF sensor off (first_time == true), call begin_timer()
     * ELSE IF not enough readings have been taken
```

```
70          *      IF sensor on long enough, take a reading
71          *      ELSE return false
72          * IF enough readings have been taken
73          *      turn sensor off, first_time = true
74          *      return true
75          * ELSE return false
76          */
77         if(first_time) {
78             function_call_count = 0;
79             begin_timer();
80             return(false);
81         }
82         else if(function_call_count < MAX_FUNCTION_CALL_COUNT) {
83             if(check_begin_reading()) {
84                 Serial.println("----------------------");
85                 Serial.print("Function Call Count: ");
86                 Serial.println(function_call_count);
87                 Serial.println("----------------------");
88                 run_sensor();
89                 function_call_count ++;
90             } else {return(false);}
91         }
92
93
94         if(function_call_count >= MAX_FUNCTION_CALL_COUNT) {
95             first_time = true;
96             digitalWrite(co2_transistor_control, LOW);
97             return(true);
98         } else{return(false);}
99     }
100
101    void MHZ19::calibrate_sensor(void) {
102        /*
103         * Method to make CO2 sensor read without turning off
104         * Turn sensor on and wait until it warms upper_bound
105         * Take sensor readings forever
106         */
107        if(first_time) {
108            function_call_count = 0;
109            begin_timer();
110        }
111
112        if(check_begin_reading()) {
113            Serial.println("----------------------");
114            Serial.print("Function Call Count: ");
115            Serial.println(function_call_count);
116            Serial.println("----------------------");
117            run_sensor();
118            function_call_count ++;
119        }
120    }
121
122    bool MHZ19::run_sensor(void) {
123        /*
124         * Run the MHZ19 sensor
125         * Set ppm to zero
126         * clear the frame_buffer
127         * drain the serial buffer
128         * read from the sensor
129         * print reading
130         * add the reading to the average value buffer
131         * calculate average value
132         */
133        co2_ppm = -1;
134        co2_ppm_average_uncalibrated = 0;
135        co2_ppm_average_calibrated = 0;
136        is_average_taken = false;
137        does_sensor_work = true;
138        reading_count = 1;
```

```
139
140         serial_drain();
141
142         while(is_average_taken == false && does_sensor_work == true) {
143             memset(frame_buffer, 0, 9);
144             read_sensor();
145             print_current_reading();
146             add_to_ave_buf();
147             calculate_average_reading();
148             print_average_reading();
149             co2_ppm = -1;
150         }
151         if(is_average_taken == true) {return(true);}
152         else {return(false);}
153     }
154
155     void MHZ19::print_current_reading(void) {
156         /*
157          * Prints current reading if reading is valid (i.e. co2_ppm > 0)
158          * and if the maximum number of readings haven't been exceeded
159          */
160         if(co2_ppm > 0) {
161           Serial.print("MHZ19 CO2 PPM Reading ");
162           Serial.print(reading_count);
163           Serial.print(": ");
164           Serial.println(co2_ppm);
165         }
166         else {
167           Serial.println("Error reading CO2 PPM from MHZ19");
168         }
169     }
170
171     void MHZ19::add_to_ave_buf(void) {
172         /*
173          * IF a valid value of co2 is read and the number of reading is less than the max,
174          * THEN add current value to buffer
175          */
176         if(co2_ppm > 0 && reading_count <= NUMBER_OF_VALUES) {
177             mhz19_buffer[reading_count - 1] = co2_ppm;
178             reading_count += 1;
179         }
180     }
181
182     void MHZ19::calculate_average_reading(void) {
183         /*
184          * IF the number of readings exceeds the number of values to be read,
185          * THEN calculate the average
186          */
187         if(reading_count > NUMBER_OF_VALUES) {
188             for(int k = 0; k < NUMBER_OF_VALUES; k++) {co2_ppm_average_uncalibrated +=
            mhz19_buffer[k];}
189
190             co2_ppm_average_uncalibrated = co2_ppm_average_uncalibrated / ( NUMBER_OF_VALUES
             );
191             is_average_taken = true;
192         }
193     }
194
195     void MHZ19::apply_calibration_curve(void) {
196         /*
197          * Method to apply calibration curve
198          * calibrated_value = a0 + uncalibrated_value * a1
199          */
200         co2_ppm_average_calibrated = calib_a0 + co2_ppm_average_uncalibrated * calib_a1;
201     }
202
203     void MHZ19::print_average_reading(void) {
204         /*
205          * IF the average has been taken (co2_ppm_average > 0)
```

```
206         * THEN print the average
207         */
208        if(co2_ppm_average_uncalibrated > 0) {
209            Serial.println("-----------------------------");
210            Serial.print("CO2 PPM Average Reading (Uncalibrated): ");
211            Serial.println(co2_ppm_average_uncalibrated);
212            apply_calibration_curve();
213            Serial.print("CO2 PPM Average Reading (Calibrated): ");
214            Serial.println(co2_ppm_average_calibrated);
215            Serial.println("-----------------------------");
216        }
217    }
218
219    void MHZ19::read_sensor(void) {
220        /*
221         *  Start Serial1 connection
222         *  Send command to read from sensor to the sensor
223         *  Read from the sensor (fill_from_buffer();)
224         *  Calculate PPM for CO2
225         *  End Serial connection
226        */
227
228        Serial1.begin(9600);
229        Serial1.write(mhz19_read_command, 9);
230        delay(1000);
231        fill_frame_buffer();
232        co2_ppm = 256*frame_buffer[2] + frame_buffer[3];
233        Serial1.end();
234    }
235
236    void MHZ19::serial_drain(void) {
237      /*
238       * Drains serial buffer when sensor is turned on
239       */
240        while (Serial1.available() > 0) {
241            drain = Serial1.available();
242            Serial.print("-- Draining buffer: ");
243        }
244    }
245
246    void MHZ19::frame_sync(void) {
247        /*
248         * Sync frames so that frames are added to the frame_buffer in the correct order
249         * IF correct byte is read, THEN add to buffer and move on to next byte
250         * ELSE read byte and discard
251         * IF no bytes are available to read and the frames have not been synced, THEN send
           command to read from sensor again
252         *
253         * frame_sync_count keeps track of how many frames are added to frame_buffer
254         * frame_read_count keeps track of how many frames are read but not added to buffer
           (fails if too many frames read)
255         */
256        sync_state = false;
257        frame_sync_count = 0;
258        frame_read_count = 0;
259        byte_sum = 0;
260
261        while (!sync_state && Serial1.available() > 0 && frame_read_count <
           MAX_FRAME_READ_COUNT) {
262            current_byte = Serial1.read();
263
264            if (current_byte == MHZ19_ZEROTH_BYTE && frame_sync_count == 0) {
265                frame_buffer[frame_sync_count] = current_byte;
266                byte_sum = current_byte;
267                frame_sync_count = 1;
268            }
269            else if (current_byte == MHZ19_FIRST_BYTE && frame_sync_count == 1) {
270                frame_buffer[frame_sync_count] = current_byte;
271                byte_sum += current_byte;
```

```
272             sync_state = true;
273             frame_sync_count = 2;
274         }
275         else {
276             if(debug) {
277                 Serial.print("-- Frame syncing... ");
278                 Serial.println(current_byte, HEX);
279             }
280 
281             frame_read_count ++;
282         }
283 
284         if (!sync_state && !(Serial1.available() > 0) && frame_read_count <
          MAX_FRAME_READ_COUNT) {
285             Serial1.write(mhz19_read_command, 9);
286 
287             if(debug) {
288                 Serial.println("----------------------------------------");
289                 Serial.println("Read command has been sent to CO2 sensor");
290                 Serial.println("----------------------------------------");
291             }
292 
293             delay(500);
294         }
295     }
296 }
297 
298 void MHZ19::fill_frame_buffer(void) {
299   /*
300    * Sync frames
301    * Read byte into frame_buffer
302    */
303     frame_sync();
304 
305     while(sync_state && Serial1.available() > 0 && frame_sync_count < MAX_FRAME_LEN) {
306         current_byte = Serial1.read();
307         frame_buffer[frame_sync_count] = current_byte;
308         byte_sum += current_byte;
309         frame_sync_count++;
310     }
311 }
312 
313 // getter functions
314 int MHZ19::get_co2_ave_uncalibrated(void) {
315     return co2_ppm_average_uncalibrated;
316 }
317 
318 int MHZ19::get_co2_ave_calibrated(void) {
319     return co2_ppm_average_calibrated;
320 }
321 
322 int MHZ19::get_co2_reading(void) {
323     return co2_ppm;
324 }
325 
326 void MHZ19::reset_co2_ave(void) {
327     co2_ppm_average_uncalibrated = -1;
328     co2_ppm_average_calibrated = -1;
329 }
330 
```