```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 transistor pin
     * set pinMode for transistor pin
     */
    co2_transistor_control = pin;
    pinMode(co2_transistor_control,OUTPUT);
}

void MHZ19::begin_timer(void) {
    /*
     * Turn transistor on
     * Save time at which transistor is turned on
     * Time is used for timing purposes
     * change first_time to false
     *
     * first_time indicates whether or not timer has been started
     * and transistor has been turned on
     */
    co2_ppm_average = -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 whether enough time has passed to begin reading
     * return true if enough time has passed
     * else 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) {
    /*
     * turn transistor on and start timer if this hasn't already been done
     * read from sensor if enough time has passed
     * return true if enough measurements have been taken
     * else false
```

```
70          */
71         if(first_time) {
72             function_call_count = 0;
73             begin_timer();
74             return(false);
75         }
76         else if(function_call_count < MAX_FUNCTION_CALL_COUNT) {
77             if(check_begin_reading()) {
78                 Serial.println("----------------------");
79                 Serial.print("Function Call Count: ");
80                 Serial.println(function_call_count);
81                 Serial.println("----------------------");
82                 run_sensor();
83                 function_call_count ++;
84             } else {return(false);}
85         }
86 
87 
88         if(function_call_count >= MAX_FUNCTION_CALL_COUNT) {
89             first_time = true;
90             digitalWrite(co2_transistor_control, LOW);
91             return(true);
92         } else{return(false);}
93     }
94 
95     void MHZ19::calibrate_sensor(void) {
96         /*
97          * Turn sensor on and wait for warm-upper_bound
98          * Following warm-up, read forever
99          */
100        if(first_time) {
101            function_call_count = 0;
102            begin_timer();
103        }
104
105        if(check_begin_reading()) {
106            Serial.println("----------------------");
107            Serial.print("Function Call Count: ");
108            Serial.println(function_call_count);
109            Serial.println("----------------------");
110            run_sensor();
111            function_call_count ++;
112        }
113    }
114
115    bool MHZ19::run_sensor(void) {
116        /*
117         * Run the MHZ19 sensor
118         * Set ppm to zero
119         * clear the frame_buffer
120         * drain the serial buffer
121         * read from the sensor
122         * print reading
123         * add the reading to the average value buffer
124         * calculate average value
125         */
126        co2_ppm = -1;
127        co2_ppm_average = 0;
128        is_average_taken = false;
129        does_sensor_work = true;
130        reading_count = 1;
131
132        serial_drain();
133
134        while(is_average_taken == false && does_sensor_work == true) {
135            memset(frame_buffer, 0, 9);
136            read_sensor();
137            print_current_reading();
138            add_to_ave_buf();
```

```
        calculate_average_reading();
        print_average_reading();
        co2_ppm = -1;
    }
    if(is_average_taken == true) {return(true);}
    else {return(false);}
}

void MHZ19::print_current_reading(void) {
    /*
     * Prints current reading if reading is valid (i.e. co2_ppm > 0)
     * and if the maximum number of readings haven't been exceeded
     */
    if(co2_ppm > 0) {
      Serial.print("MHZ19 CO2 PPM Reading ");
      Serial.print(reading_count);
      Serial.print(": ");
      Serial.println(co2_ppm);
    }
    else {
      Serial.println("Error reading CO2 PPM from MHZ19");
    }
}

void MHZ19::add_to_ave_buf(void) {
    /*
     * IF a valid value of co2 is read and the number of reading is less than the max,
     * THEN add current value to buffer
     */
    if(co2_ppm > 0 && reading_count <= NUMBER_OF_VALUES) {
        mhz19_buffer[reading_count - 1] = co2_ppm;
        reading_count += 1;
    }
}

void MHZ19::calculate_average_reading(void) {
    /*
     * IF the number of readings exceeds the number of values to be read,
     * THEN calculate the average
     */
    if(reading_count > NUMBER_OF_VALUES) {
        for(int k = 0; k < NUMBER_OF_VALUES; k++) {co2_ppm_average += mhz19_buffer[k];}

        co2_ppm_average = co2_ppm_average / ( NUMBER_OF_VALUES );

        is_average_taken = true;
    }
}

void MHZ19::print_average_reading(void) {
    /*
     * IF the average has been taken (co2_ppm_average > 0)
     * THEN print the average
     */
    if(co2_ppm_average > 0) {
        Serial.println("----------------------------");
        Serial.print("CO2 PPM Average Reading: ");
        Serial.println(co2_ppm_average);
        Serial.println("----------------------------");
    }
}

void MHZ19::read_sensor(void) {
    /*
     *  Start Serial1 connection
     *  Send command to read from sensor to the sensor
     *  Read from the sensor (fill_from_buffer();)
     *  Calculate PPM for CO2
     *  End Serial connection
```

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

```
            }

            delay(500);
        }
    }
}

void MHZ19::fill_frame_buffer(void) {
  /*
   * Sync frames
   * Read byte into frame_buffer
   */
    frame_sync();

    while(sync_state && Serial1.available() > 0 && frame_sync_count < MAX_FRAME_LEN) {
        current_byte = Serial1.read();
        frame_buffer[frame_sync_count] = current_byte;
        byte_sum += current_byte;
        frame_sync_count++;
    }
}

// getter and setter functions
int MHZ19::get_co2_ave(void) {
    return co2_ppm_average;
}

int MHZ19::get_co2_reading(void) {
    return co2_ppm;
}

void MHZ19::reset_co2_ave(void) {
    co2_ppm_average = -1;
}

```