```
/*
 * This is the .cpp file for the PMS7003 sensor
 * This code was written exclusively by MECH 45X Team 26
 */
PM_7003::PM_7003() {
    current_byte = 0;
    packetdata.frame_length = MAX_FRAME_LENGTH;
    frame_length = MAX_FRAME_LENGTH;
    first_time = true;
    pm_avgpm2_5 = -1;
}

PM_7003::~PM_7003() {
}

void PM_7003::set_transistor(int ground_pin, int tx_pin) {
    /*
     * Set transistor and set pin mode for transistors
     * tx_pin turns tx transistor on and off
     * ground_pin turns power to sensor on and off (transistor goes to goround)
     */
    pm_ground_control = ground_pin;
    pm_tx_control = tx_pin;
    pinMode(pm_ground_control,OUTPUT);
    pinMode(pm_tx_control,OUTPUT);
}

void PM_7003::begin_timer(void) {
    /*
     * Turn sensor on and start timer
     * (time how long sensor has been on)
     */
    digitalWrite(pm_ground_control, HIGH);
    digitalWrite(pm_tx_control, HIGH);
    start_time = now();
    Serial.println("--------------------");
    Serial.print("PMS Start time: ");
    Serial.println(start_time);
    Serial.println("--------------------");
    pm_avgpm2_5 = -1;
    first_time = false;
}

bool PM_7003::check_begin_reading(void) {
    /*
     * Check if the sensor has been on long enough to begin reading
     * duration >= PMS_START_UP_TIME
     */
    current_time = now();
    duration = current_time - start_time;
    Serial.println("-----------------");
    Serial.print("PMS Duration: ");
    Serial.println(duration);
    Serial.println("-----------------");

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

bool PM_7003::make_sensor_read(void) {
    /*
     * Get senor to read
     * Start timer if necessary
     * Check if timer has been on long enough to rad from sensor
     * If sensor has been on long enough, start reading
     * If enough readings have been taken, turn sensor off
     */
```

```
70      if(first_time) {
71          function_call_count = 0;
72          begin_timer();
73          return(false);
74      }
75      else if(function_call_count < MAX_FUNCTION_CALL_COUNT) {
76          if(check_begin_reading()) {
77              Serial.println("------------------------");
78              Serial.print("PMS Function Call Count: ");
79              Serial.println(function_call_count);
80              Serial.println("------------------------");
81              run_PM_sensor();
82              function_call_count ++;
83          } else {return(false);}
84      }
85
86      if(function_call_count >= MAX_FUNCTION_CALL_COUNT) {
87          first_time = true;
88          digitalWrite(pm_ground_control, LOW);
89          digitalWrite(pm_tx_control, LOW);
90          return(true);
91      } else{return(false);}
92  }
93
94  void PM_7003::calibrate_sensor(void) {
95      /*
96       * Start timer, if necessary
97       * Wait until the sensor has been on long enough before reading
98       * Once sensor has been on long enough, 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("PMS Function Call Count: ");
108         Serial.println(function_call_count);
109         Serial.println("------------------------");
110         run_PM_sensor();
111         function_call_count ++;
112     }
113 }
114
115 bool PM_7003::run_PM_sensor(void) {
116     /*
117      * Start serial connection
118      * Initialize variables
119      * drain_serial() and read_sensor() until enough values have been read
120      * Take average
121      * end serial connection
122      */
123     Serial1.begin(9600);
124     read_count = 1;
125     done_reading = false;
126     frame_sync_count = 0;
127     while(!done_reading && frame_sync_count < MAX_FRAME_SYNC_COUNT) {
128         drain_serial();
129         delay(750);
130         read_sensor();
131     }
132
133     Serial1.end();
134
135     if(done_reading) {
136         Serial.println("--------------------------");
137         Serial.print("PM 2.5 Average Reading: ");
138         Serial.println(pm_avgpm2_5);
```

```
        Serial.println("--------------------------");
        return true;
    }
    else if(!done_reading && frame_sync_count >= MAX_FRAME_SYNC_COUNT){return false;}
}

void PM_7003::drain_serial(void) {
    /*
    * Drains serial buffer if there are more than 32 entries
    * Reads entries to drain serial buffer
    */
    if (Serial1.available() > 32) {
        drain = Serial1.available();
        Serial.println("-- Draining buffer: ");
        Serial.println(Serial1.available(), DEC);
        for (int drain_index = drain; drain_index > 0; drain_index--) {Serial1.read();}
    }
}

void PM_7003::frame_sync(void) {
  /*
   * syncs frames for PM sensor
   * checks that frames are being read in correct order
   * exits when it confirms that frames are being read correctly
  */
    sync_state = false;
    frame_count = 0;
    byte_sum = 0;

    while (!sync_state && frame_sync_count < MAX_FRAME_SYNC_COUNT){
        current_byte = Serial1.read();

        if(current_byte == FIRST_BYTE && frame_count == 0) {
            frame_buffer[frame_count] = current_byte;
            packetdata.start_frame[0] = current_byte;
            byte_sum = current_byte;
            frame_sync_count = 1;
            frame_count = 1;
        }
        else if(current_byte == SECOND_BYTE && frame_count == 1){
            frame_buffer[frame_count] = current_byte;
            packetdata.start_frame[1] = current_byte;
            byte_sum = byte_sum + current_byte;
            frame_count = 2;
            frame_sync_count = 1;
            sync_state = true;
        }
        else{
            frame_sync_count++;

            if(frame_sync_count >= 10) {
                Serial.print("frame count: ");
                Serial.println(frame_sync_count);
            }

            if(debug) {
                Serial.println("frame is syncing");
                Serial.print("Current character: ");
                Serial.println(current_byte, HEX);
                Serial.print("frame count: ");
                Serial.println(frame_sync_count);
            }

            delay(750);

            if(frame_sync_count >= MAX_FRAME_SYNC_COUNT) {
                Serial.println("------------------------");
                Serial.println("Max frame count exceeded");
                Serial.println("------------------------");
```

```
208             }
209 
210         }
211     }
212 }
213 
214 void PM_7003::read_sensor(void) {
215     /*
216      * Sync the frames
217      * read bytes and fill frame_buffer
218      * use data_switch to calculate different parameters
219      * print_messages once all values have been read.
220      * done_reading = true if enough values have been read
221      */
222     frame_sync();
223 
224     while(sync_state == true && Serial1.available() > 0) {
225         current_byte = Serial1.read();
226         frame_buffer[frame_count] = current_byte;
227         byte_sum = byte_sum + current_byte;
228         frame_count++;
229         uint16_t current_data = frame_buffer[frame_count-1]+(frame_buffer[frame_count-2
        ]<<8);
230         data_switch(current_data);
231 
232         if (frame_count >= frame_length && read_count <= MAX_READ_COUNT) {
233             print_messages();
234             read_count++;
235             break;
236         }
237     }
238 
239     if (read_count > MAX_READ_COUNT) {
240         pm_avgpm2_5 = 0;
241         pm_avgpm1_75 = 0;
242         pm_avgpm0_75 = 0;
243         pm_avgpm0_4 = 0;
244         for(int k = 0; k < MAX_READ_COUNT; k++) {pm_avgpm1_75 += pm1_75_buf[k];}
245         for(int k = 0; k < MAX_READ_COUNT; k++) {pm_avgpm0_75 += pm0_75_buf[k];}
246         for(int k = 0; k < MAX_READ_COUNT; k++) {pm_avgpm0_4 += pm0_4_buf[k];}
247         float pm_avg04_f = 3668*exp(-2.265*pow(10,-6) * (pm_avgpm0_4/MAX_READ_COUNT)) +
        25.63*exp(0.0001089*(pm_avgpm0_4/MAX_READ_COUNT));
248         float pm_avg075_f = 329.9*exp(5.122*pow(10,-5) * (pm_avgpm0_75/MAX_READ_COUNT))
        + 21.26*exp(0.0002764*(pm_avgpm0_75/MAX_READ_COUNT));
249         float pm_avg175_f = 1.941*pow(10,-12)*pow((pm_avgpm0_75/MAX_READ_COUNT),4) +-
        2.409*pow(10,-8)*pow((pm_avgpm0_75/MAX_READ_COUNT),3) + 0.0001295*pow((
        pm_avgpm0_75/MAX_READ_COUNT),2)+ -0.02592*(pm_avgpm0_75/MAX_READ_COUNT)+ 30.16;
250         float pm_avg_fvol = pm_avg04_f*4/3*3.14159265359*pow((400/2*pow(10,-9)),3)+
        pm_avg075_f*4/3*3.14159265359*pow((750/2*pow(10,-9)),3)+pm_avg175_f*4/3*
        3.14159265359*pow((1750/2*pow(10,-9)),3);
251         float pm_avg_fmass = pm_avg_fvol*1.65*pow(100,3)*10*1000*1000000;
252 
253         pm_avgpm2_5 = pm_avg_fmass;
254         done_reading = true;
255     }
256 }
257 
258 void PM_7003::data_switch(uint16_t current_data) {
259     /*
260      * data_switch uses current data and frame_count
261      * to assign values to parameters
262      */
263     switch (frame_count) {
264     case 4:
265         packetdata.frame_length = current_data;
266         frame_length = current_data + frame_count;
267         break;
268     case 6:
269         packetdata.concPM1_0_factory = current_data;
```

```
270             break;
271         case 8:
272             packetdata.concPM2_5_factory = current_data;
273             break;
274         case 10:
275             packetdata.concPM10_0_factory = current_data;
276             break;
277         case 12:
278             packetdata.concPM1_0_ambient = current_data;
279             break;
280         case 14:
281             packetdata.concPM2_5_ambient = current_data;
282             break;
283         case 16:
284             packetdata.concPM10_0_ambient = current_data;
285             break;
286         case 18:
287             packetdata.countPM0_3um = current_data;
288             break;
289         case 20:
290             packetdata.countPM0_5um = current_data;
291             break;
292         case 22:
293             packetdata.countPM1_0um = current_data;
294             break;
295         case 24:
296             packetdata.countPM2_5um = current_data;
297             break;
298         case 26:
299             packetdata.countPM5_0um = current_data;
300             break;
301         case 28:
302             packetdata.countPM10_0um = current_data;
303             break;
304         case 29:
305             current_data = frame_buffer[frame_count-1];
306             packetdata.version = current_data;
307           break;
308         case 30:
309             current_data = frame_buffer[frame_count-1];
310             packetdata.error = current_data;
311             break;
312         case 32:
313             packetdata.checksum = current_data;
314             byte_sum -= ((current_data>>8)+(current_data&0xFF));
315             break;
316         default:
317             break;
318         }
319     }
320 
321     void PM_7003::print_messages(void){
322         /*
323          * Print messages to string and Serial screen
324          */
325         sprintf(print_buffer, ", %02x, %02x, %04x, ",
326             packetdata.start_frame[0], packetdata.start_frame[1], packetdata.frame_length);
327         sprintf(print_buffer, "%s%04d, %04d, %04d, ", print_buffer,
328             packetdata.concPM1_0_factory, packetdata.concPM2_5_factory, packetdata.
                concPM10_0_factory);
329         sprintf(print_buffer, "%s%04d, %04d, %04d, ", print_buffer,
330             packetdata.concPM1_0_ambient, packetdata.concPM2_5_ambient, packetdata.
                concPM10_0_ambient);
331         sprintf(print_buffer, "%s%04d, %04d, %04d, %04d, %04d, %04d, ", print_buffer,
332             packetdata.countPM0_3um, packetdata.countPM0_5um, packetdata.countPM1_0um,
333             packetdata.countPM2_5um, packetdata.countPM5_0um, packetdata.countPM10_0um);
334         sprintf(print_buffer, "%s%02d, %02d, ", print_buffer,
335             packetdata.version, packetdata.error);
336 
```

```
337         float pm0_4_f = packetdata.countPM0_3um - packetdata.countPM0_5um;
338         float pm0_75_f = packetdata.countPM0_5um - packetdata.countPM1_0um;
339         float pm1_75_f = packetdata.countPM1_0um - packetdata.countPM2_5um;
340         pm1_75_buf[read_count-1] = pm1_75_f;
341         pm0_75_buf[read_count-1] = pm0_75_f;
342         pm0_4_buf[read_count-1] = pm0_4_f;
343 
344         if(debug) {
345             Serial.println(print_buffer);
346         }
347 
348         Serial.print("PM 2.5 Reading #");
349         Serial.print(read_count);
350         Serial.print(": ");
351         Serial.println(pm1_75_buf[read_count-1]);
352     }
353 
354     float PM_7003::get_pm_ave(void) {
355         return pm_avgpm2_5;
356     }
357 
358     void PM_7003::reset_pm_ave(void) {
359         pm_avgpm2_5 = -1.0;
360     }
361 
```